Features Relevant to Short-Term Wireless Channel Utilization Prediction

by

Sepehr Kazemian

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Wireless channel utilization prediction is useful in a number of applications, such as the recently proposed modalities of LTE networks allowing them to use unlicensed bands (LTE-U), otherwise used by Wi-Fi devices. Wireless utilization, as we are also able to also confirm, exhibits non-stationary behavior. The presented research provides an overall prediction strategy that can be implemented at the network edge. While the legacy view of "busy" hours vs. "non-busy" hours is still relevant, we approach the modeling with a finer definition for this busy/non-busy distinction. We split the utilization time series into intervals, each of them approximated as a stationary process modeled as a Markov chain. Each of those micro-models captures the short-term behavior and is characterized by its steady state distribution. The steady state distributions are used to define similarity among intervals in terms of their short-term behavior, i.e., the micro-models become a "library" of prior behaviors. We use a shallow neural network that combines features that express the similarity to a set of prior intervals, together with features arising from the time series using an auto-regressive model following the Box-Jenkins method, alongside features capturing straightforward step-to-step (lag one) transitions. The shallow network allow us to interpret the relative importance of the various features. It allows us to glean from the weights assigned why naive models (predicting next what has just been observed)

are quite potent, and especially, and unsurprisingly, for non-busy hours. Moreover, we evaluate our prediction setup over "coarsened" utilization ranges since, for most applications, the granularity of prediction need not be fine as long as it describes distinct utilization regimes (e.g., "idle", "lightly loaded", "moderately loaded", "heavily loaded", "(almost) saturated"). The evaluation is carried out by predicting the utilization of Wi-Fi channels. Specifically for Wi-Fi channels, the architecture of our prediction platform exploits the utilization self-reporting performed by Access Points in Beacon frames. An extensive data collection experiment was designed and carried out, forming the real world data over which our prediction scheme is evaluated.

An ice becomes water in the course of time
The water that has become a pearl,
will never again become water!

– Rumi.

Acknowledgements

First of all, I would like to deeply thank Natural Sciences and Engineering Research Council of Canada (NSERC) and Cybera Rapid Access Cloud. The former for funding my research projects and thesis through the Discovery Grant program and the latter for providing me cloud computing environments to exploit research and academic experiments using free computational resources.

I am highly grateful to my supervisor, Ioanis Nikolaidis, for constantly educating, supporting, and guiding me through each stage of my Master's program. Through the meetings and conversations, he always inspired me to think outside the box and his feedback enlightened my research path.

I would also like to take this opportunity to thank my colleagues in the Networks Group. I have worked with a brilliant set of lab-mates. Madi, Alan, and Mark always inspired me for the best. Similarly, for the group meetings and presentation that were helpful for learning different research topics and perspectives. Advice from knowledgeable professors, Dr. Ioanis Nikolaidis, Dr. Omid Ardakanian, and Dr. Martha Steenstrup, helped me a lot in improving my soft-skills.

I am thankful to all of my friends who although are living in different parts of the world, they always supported me. Having them is a blessing. Also, I would like to thank my dear friends, Arash and Pouneh, who were always supportive and left us with their precious memories.

Finally, my special thanks to my parents, who have been devoting themselves for

a bright future of mine, and my brother, my always best friend, who always has been helping me to progress in my life. Nothing was possible without them.

Contents

1 Introduction **1**

2 Related Work and Background **9**
2.1 Background on Wireless Networking 9
2.2 Prediction Methodologies 11
2.3 Clustering and Lumping 15
2.4 Cost Functions 16

3 A Model for Short-Term Dynamics **19**
3.1 Interval Comparison Metric 21
3.2 Seasonality Reduction 22
3.3 Condensing the Intraday Seasonality 25
3.3.1 τ Evaluation Benchmark 26

4 Feature Engineering **29**
4.1 Simple Prediction Methods as Benchmarks 29
4.2 Auto-Regressive Models 31
4.3 Short-Term Dynamics Distance 35
4.4 Selected Features 39

5 State Lumping and the Cost of Misprediction **41**
5.1 Simplifying $\mathbf{P}_{\tau}^{(\mathbf{i})}$ and Lumpability 42
5.1.1 Lumpability 42
5.1.2 Quasi-Lumpability 44
5.2 The Cost of Misprediction 48

6 Experimental Evaluation **53**
6.1 Data Acquisition and Experiment Setup 53
6.2 τ and Feature Engineering 55
6.2.1 Results 57

7 Conclusions and Future Work **82**

References **86**

List of Tables

6.1 Accuracy for naïve predictor, under global lumping vs. separate, for each $\tau = 30$ minute interval, lumping. 61
6.2 Boundaries from lumping into 6 classes (lumped states) with $\tau = 30$ minutes, $C_l = 10\%$, and $C_u = 25\%$. 63
6.3 Accuracy for feature-engineered NN (red) vs. naïve (blue) predictor (6 classes, $\tau = 30$ minutes, $C_l = 10\%$, $C_u = 25\%$). 64
6.4 The weights of layer 2 in the NN for a typical busy interval (10am-10:30am) capturing relative importance of the features. 67
6.5 A supporting figure 68
6.6 Accuracy for feature engineered NN for global vs. MI (with $\tau = 5$ minutes) vs. separate $\tau = 30$-minute interval lumping using 6 lumpted states (utilization ranges) in all cases. 72

List of Figures

1.1 Example channel utilization time series (as CU values). 6

2.1 QBSS Load Element (duplicated from [17]). 10

3.1 Concatenated time series of, respectively, typical idle and busy hour utilization on channel 1 over 90 days. 21
3.2 Dot product of $\pi_5^{(0)}$ with a similar (similarity 99%) steady state vector, $\pi_5^{(1)}$. 23
3.3 Dot product of $\pi_5^{(0)}$ with a dissimilar (similarity 53%) steady state vector, $\pi_5^{(150)}$. 24
3.4 Dot product of $\pi_5^{(0)}$ and $\pi_5^{(i)}$ using data collected over 90 days. Each time interval to which an i corresponds is shown as hour of the day (row) and five-minute interval within the hour (column). 25

4.1 Example auto-correlation of CU values during a busy hours interval (12pm-5:30pm) of a randomly chosen day. 34
4.2 Example auto-correlation of CU values during an idle hours interval (12am-5:30am) of a randomly chosen day. 35
4.3 Example auto-correlation of DL of CU values during a busy hours interval (12pm-5:30pm) of a randomly chosen day. 36
4.4 Example auto-correlation of DL of CU values during an idle hours interval (12am-5:30am) of a randomly chosen day. 37
4.5 Dot product of $\mathbf{P}_5^{(j)}$ with $\mathbf{P}_5^{(i)}$ within a day ($j = 0, ..., 11$ with one line for each j and $i = 0, ..., 287$). 38
4.6 Dot product of $\mathbf{P}_5^{(j)}$ with $\mathbf{P}_5^{(i)}$ within an hour ($j = 0, ..., 11$ with one line for each j and $i = 0, ..., 287$). 39

5.1 An example of state space reduction of the transition matrix leading to a 3×3 lumped matrix. For visual clarity, self-loops and arrows are not shown. State numbers/ranges refer to the corresponding CU values represented by a state. 42
5.2 An example of lumping. 44
5.3 An example of quasi-lumping. 46

5.4 An example of class mapping on Rayleigh function, $y = 0.0$ when the right utilization range is predicted assuming the right one is in the range of $[46.3\% - 58\%]$. 50

6.1 The neural network architecture used. 58
6.2 Accuracy for naïve predictor, under global lumping (red) vs. separate (blue), for each $\tau = 30$ minute interval, lumping. 60
6.3 Accuracy for feature-engineered NN (red) vs. naïve (blue) predictor (6 classes, $\tau = 30$ minutes, $C_l = 10\%$, $C_u = 25\%$). 65
6.4 Accuracy for feature-engineered NN assuming access to the current interval's steady state (red), or, the upcoming stationary features (blue) 66
6.5 Building up the NN's accuracy vs. a naïve predictor baseline. 69
6.6 Accuracy for $\tau = 60$, $\tau = 30$, $\tau = 20$, and $\tau = 10$ minutes. 71
6.7 Cross-entropy vs. asymmetric (Rayleigh-based) loss function performance for 6 lumped states (utilization ranges). 74
6.8 Cross-entropy vs. asymmetric (Rayleigh-based) loss function performance for 5 lumped states (utilization ranges). 75
6.9 Comparison of wrongly predicted classes, at various times of the day, for normal cross-entropy and proposed asymmetric loss function for 5 lumped states (utilization ranges) and $\tau = 60$ minutes. 79
6.10 The impact on (cumulative) accuracy of different utilization range constraints imposed during lumping. 80
6.11 Multi-step prediction accuracy ($\tau = 60$ minutes). 81

Chapter 1

Introduction

We consider the problem of predicting the short-term utilization of a wireless channel based on measurements collected over the recent past, where the past spans enough days to allow seasonality patterns and trends to emerge. Predicting the short-term utilization can be beneficial to a variety of applications such as Long Term Evolution (LTE) operation in unlicensed bands (LTE-U), whereby an LTE-U small cell is also observing the utilization of the unlicensed channels, such as those of IEEE 802.11 Wi-Fi and uses the predictions to offload some part of its traffic, over the unlicensed band channels. The other prototypical application is a spontaneous peer-to-peer Wi-Fi (e.g. AdHoc mode) communication for, e.g. file transfers, using the same channels as those of co-exisiting managed (Access Points with clients) networks. A third application is the operation of Wi-Fi sensor nodes that are permanently placed but have limited energy resources. Minimizing they need to be ON (reducing their duty cycle) to perform a data transfer can be informed by predicting if the future utilization of the channel is favorable. The more congested the medium, the longer their ON time – the more energy-expensive the attempt to transmit at that point in time. Conceivably, by predicting the short-term utilization of the network, a sensor can attempt to transfer data if it finds the network less congested, and it can postpone the transmission otherwise. These examples serve as motivation, but

they are broadly applicable in cases where utilization needs to be determined for reasons of resource allocation over the near to medium term, spanning numerous wireless co-existence scenarios [25]. Additionally, prediction of utilization is useful when transceivers need to select the "best" channel (ostensibly the least utilized) from a set of options, but the particulars of channel selection are outside the scope of this work.

In this work we explore how to predict utilization of the next time-step, shown as $\hat{u}(t+1)$, based on collected utilization data by processing $u(t)$, the utilization measurements at time t. We define the error of prediction as $error = |\hat{u}(t+1) - u(t+1)|$, which we wish to keep it small. Ideally, the error is 0 and $\hat{u}(t + (1 \times \sigma)) = u(t + (1 \times \sigma))$. However, in most cases the error is not zero and gets either a negative value or a positive one – the sign matters. We use the prediction to decide how much of the capacity we can use, then, behave greedily to use up the remaining capacity, we are overutilizing the available capacity (when the $error < 0$) and underutilizing (when the $error > 0$). To this end, an asymmetric cost function to potentially quantify differently the over- vs. under-estimation of the utilization is also incorporated in our work.

We will subsequently adopt an abstract view of the applications. Assume that a node turns on and can only perform a single measurement, i.e., receive the utilization of a nearby AP (possibly, the AP to which it intends to transact with), informing it of $u(t)$. Then, based on this measurement it has to predict the utilization for the next time-step, $\hat{u}(t+1)$. We address the question of what is a good predictor for the $u(t+1)$ without requiring any modifications to current protocol standards. Also, given some anecdotal evidence (that are confirmed later in this study) that a naïve predictor works very well in many cases, we touch the issue of how to construct an easily explainable predictor that outperforms the naïve one. Here the naïve predictor is simply defined as one that, having measured the $u(t)$ (for the most recent "timeslot"),

predicts that the $\hat{u}(t+1)$ (for the next timeslot) will remain the same, i.e. $\hat{u}(t+1) = u(t)$.

A possible implementation of our proposal assumes that a device (possibly an Access Point (AP) or an edge device) is able to continuously monitor the utilization of the channels and, employing low computational resource-demand prediction algorithms, inform interested parties of future utilization estimates.

Most traffic traces of wireless traffic exhibit familiar patterns such as the one in Figure 1.1. Visually one can identify "clusters" of values (circled areas) of similar utilization values as well as peaks and valleys characterizing particular times of the day. It is not at all certain that the process being observed is stationary (rather, it almost certainly is not) but we will assume that (a) the prediction process can be made aware of the time/day/etc., and could in principle switch from one model to another based on that, and (b) over a short interval (which we will denote by τ) the utilization can be approximated well by a stationary process, which we narrow it to be a Markovian process.

The collected data in our study comes from the APs' measured utilization values carried in AP Beacon frames. The demand for higher efficiency forced Institute of Electrical and Electronics Engineers Standards (IEEE Std) community to encode more information regarding the environment in frames generated by nodes/wireless stations (STAs). The purpose of this extra information is to provide more information, such as busy airtime and supported protocols, to nodes. In this work, we exploit the information broadcasted from APs about busy airtime (amount of channel busy time per second). STAs can then use busy airtime in order to make decisions about either sending out some data or postponing it to prevent congestion and/or packet loss. To this end, the IEEE standard introduced the QoS Basic Service Set (QBSS) Load Element as part of the 802.11e amendment in 2012 [17]. This information element is advertised in every management packet, e.g. Beacon and Probe Response

frame, of an AP that supports Wi-Fi Multimedia (WMM) [17]. Virtually all APs in use today (2020) support the 802.11e extensions and they are responsible to sense the channel and broadcast the amount of busy airtime at each second. Utilization is included in one component of the Load Element, called *Channel Utilization* (CU), a field of 8 bits, whose values range from 0 to 255 linearly mapping to the range from 0% to 100% and express the utilization of the wireless medium, u, seen by the AP (including the traffic with its own clients) as assessed by the carrier sense mechanism. The carrier sense mechanism includes also the Network Allocation Vector (NAV) "virtual" carrier sensing. It also incorporates the load seen by the AP caused by IEEE 802.11 transmissions of other transmitters on the same channel that are not associated with the specific AP. Non-802.11 transmissions are also, indirectly, accounted for, as long as their signal strength is sufficient that the physical carrier-sensing of the AP assesses the medium as being busy.

Let $u(t, c, l)$ be the utilization of channel c at time t at location l. At a specific location, we can receive utilization values from different APs, that can be shown as $u(t, c_{AP\_1})$, $u(t, c_{AP\_2})$, ... , $u(t, c_{AP\_N})$. As the co-located APs operating on a channel report approximately the same CU value for that channel, an observer at a fixed location can focus on the AP with the strongest received AP beacons signal as it is reporting the utilization as seen at a location very close to the observer. For the sake of this thesis, we also restrict channels to receive beacons from the non-overlapping Wi-Fi channels indicated as c_1, c_6, c_{11} in the 2.4GHz for respectively channels 1, 6, and 11. For a specific channel at a location we can further simplify the notation and write it down as $u(t)$ where t is time as a discrete positive value. For predicting the utilization values of the next i_{th} interval with interval length of σ, we denote $u(t + (i \times \sigma))$ as the measured utilization value, and $\hat{u}(t + (i \times \sigma))$ as the predicted one. Our main goal is to predict $\hat{u}(t + (1 \times \sigma))$ based on collecting and processing the $u(t)$ measurements announced in Beacons.

Our methodology consists of training a shallow Neural Network (NN) by several alternative inputs, and producing a class membership output where the class corresponds to a range of, consecutive, utilization values, thus making the prediction less granular, but tunable to adapt to application needs. The incremental way of building the training data for NN allows us to capture the intuitively simple, and often effective, naïve predictor, and then to demonstrate its refinement by bringing in the non-stationary adjustment and a (unique to our work to the best of our knowledge) distance metric relative to previous short-term dynamics. By gleaning into the NN weights, we can unearth the relative importance of the various features fed to the NN.

In this work, no modification of the protocol stack is performed that would render the approach incompatible with IEEE 802.11. For predictors, even the most complex deep-learning ones, it is difficult to predict $\hat{u}(t+1)$ with a reasonable accuracy, in steps of $\frac{100}{256}\%$. Hence, the objective is to start from a 256 step representation, and reduce it into a smaller one which we will subsequently use in the prediction process. This task of "coarsening" can be performed by *state lumping* as we will elaborate in Chapter 5. The act of lumping states, creates "cliques" of states that, seen collectively as a "meta-state", can result in the process remaining in the meta-state for prolonged periods of time. Our lumped states represents a range of utilization values. The lumped state will make naïve predictors work very well (the process is predicted to remain in the same state at the next step). With lumping, we may construct more than one lumped state (in the evaluation we limit the lumped states to, at most, seven), and hence a rich set of lumped states, each representing a different utilization range, are produced, while "transient" states also emerge linking those cliques.

The CU time-series shown in Figure 1.1 suggests fluctuations are, mostly, happening between groups of utilization values. A feature of the current study is that we analyze a real data set collected over several months in an academic campus en-

vironment. One of our contributions is that we describe the short-term dynamics using first-order Markov chains. Specifically, we captured around 120 days worth of data of Wi-Fi AP Beacon transmissions from a plethora of APs in a campus building at the University of Alberta. The majority of APs in the environment are centrally managed to provide ubiquitous Wi-Fi service. Each AP transmits approximately 10 Beacons per second. All of the observed APs support WMM and QBSS; so, they report the CU values at each beacon. To capture the beacons, we use Wi-Fi "sniffers" at various locations so they can capture the reported CU values from different APs. Our approach is based on broadcasted, unencrypted frames, and does not involve any information relating the data to specific individuals/users.

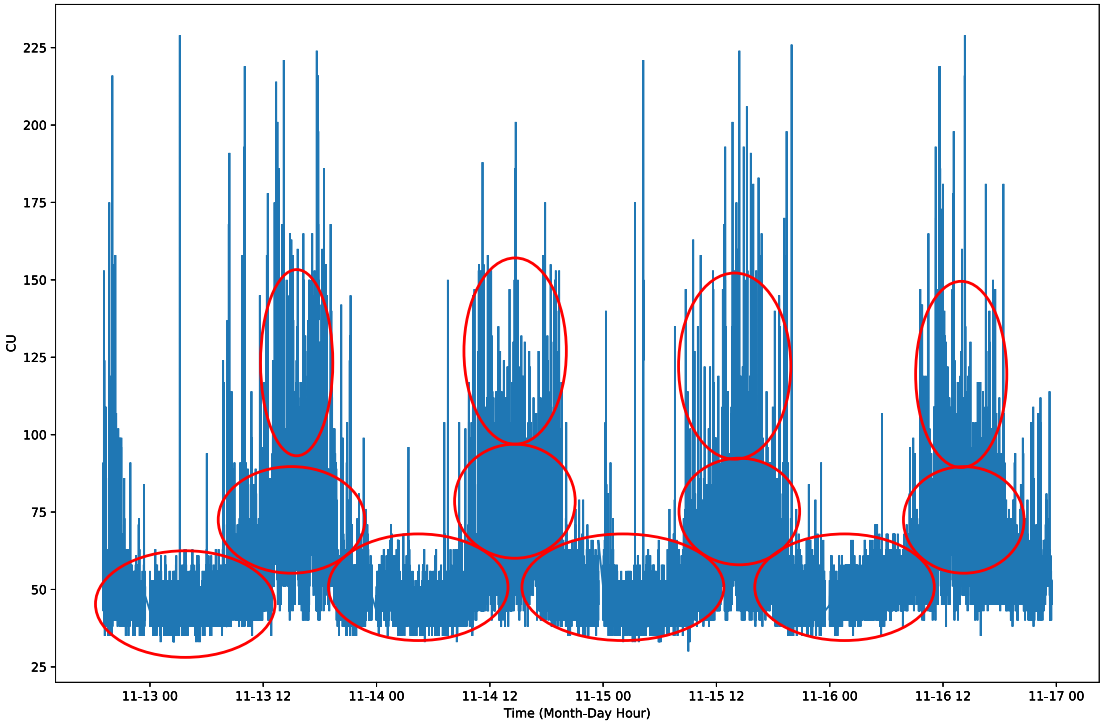


Figure 1.1: Example channel utilization time series (as CU values).

To summarize, the contributions if this thesis are as follow:

a) we conduct an extensive data collection task to acquire a data set which is quite unique using a channel utilization data collection methodology which is, to the best of our knowledge, new and original,
b) we introduce a metric which allows us to both express the stochastic behaviour of time-series over short time intervals and compare the behaviour across different intervals,
c) we propose models, key among them being one of an artificial neural network, that combines features related to the short-term and longer-term behavior of the utilization,
d) we evaluate the performance of various prediction alternatives and discuss their tradeoffs, noting that a naïve predictor can still be quite powerful but the synthesis of additional features provides a notable improvement,
e) we consider asymmetric misprediction costs to capture situations where the intention is to use the remainder (non-utilized) part of the channel, and where congestion is more costly than under-utilization, and, finally,
f) throughout the thesis we try to address the potential complexity of the prediction mechanisms, by making the predictions less granular yet still useful in network operations by using *state lumping* techniques to group utilization levels together in coarser units.

After a presentation of related work in Chapter 2, we introduce in Chapter 3 a method to compare the short-term dynamics of intervals across the utilization time series. This method, along with more features are reviewed in Chapter 4 as they form the basis for training our models. Chapter 5 introduces the state lumping and subsequent "summarization" of utilization values into utilization ranges, as well as the asymmetric loss function we propose to fit the needs of systems where prediction is used to decide how much of the system capacity to utilize without causing congestion. Chapter 6 is a collection of performance results, and it starts with a de-

scription of the data collection experiment, followed by comparisons between naïve and more elaborate models, using the weights of a shallow neural network to interpret the impact of the various features on the prediction. Chapter 7 provides concluding remarks and future work.

Chapter 2

Related Work and Background

In this chapter, we, first, provide some background information on wireless networking. Then we introduce different methodologies for anticipatory networking methods. These are methods used in networking for performing prediction in various contexts and have been recently [8] surveyed in their various forms and applications. Finally, to investigate the cost of misprediction in networking field, we go through different cost functions exploited in this area of research. This thesis relies on a synthesis of knowledge from various areas of research. We present the key related work that influenced this synthesis on various technical facets.

2.1 Background on Wireless Networking

Typical APs deployed in urban settings are permanently located at fixed locations. As such, they experience the ebb and flow of fluctuating wireless medium use across various time scales. The particular patterns of the wireless medium use are related to the type of use, e.g., residential vs. commercial setting etc. We specifically consider an educational institution environment, in which the peak traffic is expected during business hours, and the off-peak hours are typically late night to early morning hours. The traffic patterns also change during weekends and holidays. We look at the specific

case of a campus environment, with a plethora of deployed APs providing generic Internet access services within in a single building. Without harming generality we consider the working week traffic, with weekends/holidays being extensions to the presented models.

The QBSS Load Element

As mentioned in the introduction, the QoS Basic Service Set (QBSS) Load Element was introduced as part of the 802.11e standard amendment in 2012 [17]. This information element is advertised in every Beacon and every Probe Response frame of an AP that supports Wi-Fi Multimedia (WMM) [18]. The QBSS Load element information format depicted in the standard is shown in Figure 2.1. The Station Count field shown in the figure, is interpreted as an unsigned integer that indicates the total number of STAs currently associated with this BSS. The other component of the Load Element is an 8 bits channel utilization (CU) field ranges from 0 to 255. Here, Channel_Busy_Time is the number of microseconds during which the carrier sense mechanism has indicated a channel busy, and dot11_CU_Beacon_Intervals represents the number of consecutive beacon intervals during which the channel busy time is measured (default value: 100ms) [17]. The percentage utilization of the wireless medium reported by the APs is represented as an 8-bit integer, as follows:

$$((\frac{Channel\_Busy\_Time}{dot11\_CU\_Beacon\_Intervals \times dot11\_Beacon\_Period \times 1024}) \times 255)$$

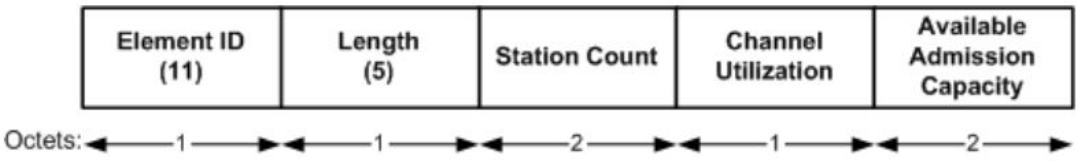


Figure 2.1: QBSS Load Element (duplicated from [17]).

Channel switching is not considered in the current study. This decision is motivated in part by the anticipation that many applications will be unable to perform channel switching, e.g., if they try to minimize energy use, or come pre-configured to use a specific channel because a specific AP is serving them on that channel. Channel selection makes sense for the case where APs can be controlled, and then schemes such as DCA [9] and similar works, e.g., [2], apply. We assume we have no form of control over the APs' operating channels. A per-channel application of what we propose here is possible but channel switching admits a wider set of options, given that one can attempt to minimize the number of tests before picking the right channel – something outside the scope of the current work.

2.2 Prediction Methodologies

Due to the prevalence of Wi-Fi communication in buildings, researchers made some efforts to bring users and APs-based observation into practical use. Furthermore, most of the techniques proposed in one wireless networking problem may be reusable in others as they share common properties both in data transmission and architecture. Consequently, reviewing available techniques in all areas of wireless networking can be fruitful.

Occupancy detection [27]–[29] methods based on Wi-Fi traffic is a well researched area. As an example, Balaji et al. [4] takes advantage of the authentication, authorization, and accounting logs of the APs in buildings to detect the occupancy but is not concerned about channel utilization statistics. We also assume that access to such logs, or any kind of user traffic-recording on the side of the APs, is not possible, or restricted for privacy reasons. To this end, using the information in management packets which APs broadcast, e.g. Beacons as we do, involves no identifying information of client devices. Moreover, in most environments, it is common for only a subset of APs to be centrally managed (if at all). The chances of the channel being

utilized due to client devices associated with APs that are outside ones' control is generally high.

Efforts have been made to use Wi-Fi management packets (e.g. Probe packets) for prediction. Wei Wang, et al. [27] used a Markov based Feedback Recurrent Neural Network (F-RNN) to predict the occupancy of a building. They include the Markov model because of the stochastic and chronologically interdependent features of Wi-Fi data packets they had to deal with. The work bears some resemblance to our work due to the Markovian facet; however, they are not concerned about time-series prediction and, more importantly, in addition to collecting identifying information of the client devices (MAC addresses), they include ad-hoc pre-processing steps to filter certain MAC addresses. Among other things, it is implied that part of this pre-processing is needed to distinguish e.g., two (or more) MAC addresses belonging to the same occupant to avoid counting them as two (or more) occupants, hence adding a-priori manually-introduced constraints. Moreover, the term F-RNN suggests to the reader relevance to the family of Recurrent Neural Networks (RNNs), but in reality it is a very limited form of recurrence considered, via a feedback mechanism, that lacks several of the structural elements of RNNs, e.g., no "forget" gates are used, etc.

Closer to the Cognitive Radio (CR) literature, we find [31], trying to predict whether the channel is idle or busy, and hence not attempt a finer-grain view of the channel utilization as we do. Nevertheless, they adopt a non-stationary Hidden Markov Chain to deal with the perceived non-stationarity of the channel but, crucially, the validation presented in [31] is only using simulations of primary user following exponential busy and idle distributions. Another work that proposed a Markov model to fit real measurements, [14], used real-time measurements made in the 928-948 MHz pager band, but does not indicate steps taken to deal with non-stationarity (the period of observations is unclear as well) and, similarly to [31],

the objective is the busy/idle distinction of the primary user. In addition, Xing et. al [30], point out to the importance of prediction in CR to tackle with under utilized spectrum. Our proposed model and loss function, covers both under- and over-utilization of the channel.

As mentioned earlier, armed with the observations made regarding Markovian models, we follow an explicit Markov chain construction whose state transition probabilities are time-dependent (Section 5.1). Contrary to busy/idle indicators, we are concerned with the prediction of the degree of utilization. As we describe in the following sections, the prediction is somewhat simplified by a coarse representation of the utilization into a small set of utilization ranges.

Temporal correlation among the time-series data plays the key role in many of the prediction methodologies suggested. As an example, many of the network planning and controlling models involve traffic loads within a day or week. Two of the most widely used time-series models depend on linear dynamics of the system are (I) Auto-Regressive and Moving Average (ARMA) [16] and (II) Kalman filters.

The ARMA model is a generalization and mixture of two simple Auto-Regressive (AR) and Moving Average (MA) models. The model is reffered to as ARMA(p,q) model where p is the order of the AR part and q is the order of the MA part. The use of p are q is shown in Equation 2.1 where Z_t is the process of the white noise errors, and $\{\Phi_i\}_{i=1}^{p}$ and $\{\Theta_j\}_{j=1}^{q}$ are the parameters. For $p = 0$ the equation simplifies to a MA model and for $q = 0$ it is a AR model. The expression 2.2 uses the *lag operator* of $L^i X_t := X_{t-i}$. Unlike ARMA which is applicable to stationary time-series, the Auto-Regressive Integrated Moving Average (ARIMA) is suited for non-stationary time-series. In ARIMA, shown as ARIMA(p,q,d), other than q and p, a new term d, known as degree of differencing, is introduced for differentiations of the time-series for converting a non-stationary of series into the stationary one. In [22], a low-order ARIMA model, ARIMA(0-3, 0-1, 0-2), was explored alongside a

wavelet multi-resolution analysis in a methodology of spotting the time of upgrading an IP backbone network based on time-series data. In other works in anticipatory networking, for reducing the handoff latency in a large scale wireless network handoffs were predicted by using an exponential weighted moving average which is the same as an ARIMA model with no AR term, p [26]. Also, for detecting and tracking mobile nodes, an AR model to predict signal-to-noise ratio (SNR) values was proposed [19].

$$X_t = Z_t + \sum_{i=1}^{p} \Phi_i X_{t-i} + \sum_{j=1}^{q} \Theta_j Z_{t-j} \tag{2.1}$$

$$\Phi(L) X_t = \Theta(L) Z_t \tag{2.2}$$

Kalman filters are mainly adopted to model the linear dependence of the system states and are widely applied in time-series analysis for dynamics of linear systems when there is an uncertainty in variance of the historical data. For a multivariate time-series $\{x_t \in R^n : t \in \tau\}$, Kalman filter estimates state x_t, as shown in Equation 2.3, where A_t is the state transition, B_t relates the control input u_t to the state x_t, and the random variable w_t is a multivariate normal noise process [8]. In [32], Kalman filters were used to predict velocity and selective forwarding when broadcasting information to individual vehicles beyond the transmission range based on inter-vehicle communication systems under the objective that the forwarding be as fast as possible. Work in mobility prediction [33] employed a Kalman filter for real-time tracking of the location and dynamic motion of a mobile station.

$$x_t = A_t x_{t-1} + B_t u_t + w_t, t = 0, 1, ..., \tag{2.3}$$

As opposed to ARIMA and Kalman filter approach, our effort is to include some useful features to our model which force the model to become as flexible as possible from dependence on specific lags, and able to handle non-stationarity. We used the

basic ideas of linear predictors such as ARIMA as features in our non-linear model, and enriched them with more.

2.3 Clustering and Lumping

In this thesis, we use concepts of *clustering* and *lumping*. Abstractly, they bear similarities, but we distinguish them because of the two different types of uses they find in our work. Clustering is used as a means to group together periods of time over which the behavior of the utilization process is similar. Lumping is used as a means to reduce the number of distinct utilization values, by transforming them into ranges of utilization. Hence, the clustering is with respect to the temporal behavior of the utilization time series. The lumping is with respect to the values in this the series.

For predictors, even the most complex deep-learning ones, it is challenging to predict $\hat{u}(t+1)$ with a reasonable accuracy, in steps of $\frac{100}{256}\%$. Hence, the objective is to start from a 256 step representation, and reduce it into a smaller one which we will subsequently use for predicting the next state. This task of coarsening can be done by *state lumping* [7] as we will elaborate in Chapter 5. Our lumped states represent ranges of utilization values. The act of lumping states, creates "cliques" of states that, seen collectively as a meta-state, can result in the process remaining in the meta-state for prolonged periods of time.

On the other hand, for *clustering*, we consider the steady-state distribution of the utilization process, assuming it can be approximated as a first-order Markov process. Each time interval (whose duration is discussed later) is seen as a separate Markov process. Similarity of the stochastic behavior of those intervals is captured as similarity of their steady-state vectors. Based on this similarity, clusters can be formed using the K-means [21] algorithm. K-means is one of the most commonly used techniques available in anticipatory networking. This unsupervised clustering

technique splits a dataset into K groups, each with a centroid. The choice of centroids is optimized by minimizing the intra-cluster sum of squares. To put it differently, a datapoint belongs to a cluster if its Euclidean distance to the reference cluster is less than that to the others. It is common to have different clustering results when starting from different initial data points. In [23], K-means was used to build a data-driven model for predicting and avoiding packet delivery failures. Froehlich and Krumm, [13], exploited K-means to find similarities involved in vehicular paths trip history for an end-to-end routing prediction of GPS based vehicles.

The purpose of clustering (a-la K-means) in our work is to extend the, simplistic, distinction of the traffic in "idle" and "busy" hours, to a wider swath of behaviors as that is what, naturally, transpires during a working day. In this sense K is meant to be larger than 2, but not necessarily a large number. In essence the clusters capture traffic utilization behaviors "between" idle- and busy-hour extremes.

2.4 Cost Functions

In [9], Cisco proposed a dynamic channel assignment (DCA) algorithm by defining a cost metric for each channel and they update it once every second. If the cost metric (CM) of the new channel is higher than the predefined threshold, the AP will stay on that channel; otherwise, it continues searching for other channels in the same manner. The CM needs a predefined threshold which is a hard-coded value. This threshold value cannot be expected to be the same for different environments. Defining a correct and accurate threshold is one of the challenges that we indirectly try to solve by proposing the lumping method which best fits the data. Moreover, the CM is a value which comprises interference, noise, and load of the channel. In other words, CM is a weighted Signal to Noise Interference Ratio (SNIR). Our proposed method is based upon the CU values which is the end product of all of the aforementioned factors, such as interference and noise. Hence, our lumping method is able to find

appropriate ranges of utilization specific to the given environment/setup. Moreover, depending on the application, some restrictions can be applied on our proposed model to best fit the deployment environment.

Different definitions of rewards and penalties in wireless networks have been exploited. One way for predicting or taking an action with respect to a reward/penalty function is to define all effective variables in the problem and, usually, develop an optimization problem or/and a loss function out of it.

For example, Jiayu et. al designed a model of a system [15] to selectively transmit data streams by taking advantage of data streams information such as their importance level, transmission delay constraints, power function, and data sizes. For the power function, they calculated the power consumption for different receivers with different transmission rates. In their system, they assume a given number of data stream tasks, each for a destination with a prior known size. The transmitters are assumed to support different known transmission rates. The unknown reward for each task is a function of the variables capturing the data size and transmission rate. Consequently, they formulated the problem as an optimization problem where the goal is to maximize the reward. To put it differently, the optimization problem attempts to maximize the volume of data transmitted. They show this to be an NP-hard problem. To this end, they have developed a dynamic programming algorithm for the optimal solution in pseudopolynomial time. Their idea of incorporating optimization for the defined reward function provides them with an optimal or near-optimal solution for this problem. Their algorithm of the pseudopolynomial time complexity is unsuitable for real-time scenarios or for devices with low computational power, such as APs. More importantly, this reward function is effective when all the environmental and data variables are known. In this thesis, our knowledge about the environment is assumed incomplete, e.g. we do not know anything about the data service types of each data stream or whether they are periodic, or if they

have dependencies among them.

Reward functions appear in Reinforcement Learning (RL). RL algorithms use Agents for interacting with an environment. The goal of the RL algorithms is to make correct decisions toward the optimal or semi-optimal path to maximize the reward. Due to having a partial view of our environment and its stochastic changes, learning environment through exploration is not possible for RL Agents. However, the reward function can play the role of the cost function we seek. Some efforts have been made to incorporate RL into wireless networks. Nasim et al. [3] tried to address TCP fairness by using RL techniques to achieve fair resource allocation for nodes of a wireless mesh network. Their state is a tuple combining fairness and aggressiveness. It is impossible to increase one of them without affecting the other one. The idea of having two contrary concepts as "fairness" and "aggressiveness" at each state and defining a reward function that takes both into the account is similar to our problem. Similarly, in our problem, "missed chance" for maximizing the channel utilization is in contrast to sending too much data, causing "congestion". However, in contrast to [3], for our setting, existence of one of them (e.g., congestion) leads to absence of the other (underutilization).

Moreover, in our problem, creating congestion is highly unfavorable compared to missing opportunities to fully utilize the channel. As a result, an asymmetric function needs to be defined. Additionally, so far agents in RL have been incorporated in problem formulations that learn the environment only within a simulation which is not applicable in many settings with many unknown external variables involved. Our problem is also not easily replicatable within simulation as there exist a number of unknown features, such as interference, number of users, etc. all of which are in continuous flux.

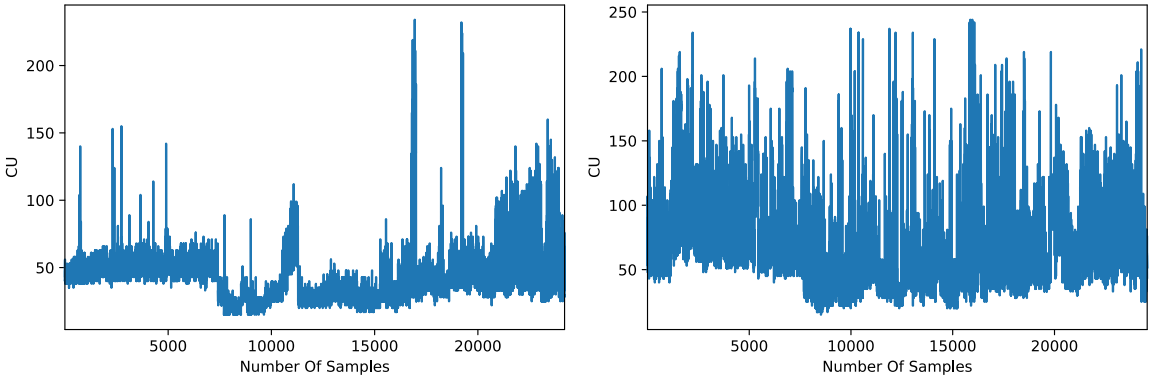
Chapter 3

A Model for Short-Term Dynamics

The problem at hand is time-series forecasting, also known as sequential prediction. For this problem, a learning algorithm seeks patterns or trends within the historical data. Any repetitive increase or decrease, due to existing features of the environment or system, is a trend. A trend does not have to be linear. Sometimes, as in our problem, the trend is affected by seasonal patterns. Seasonality influences data differently and may solely depend on time of the day, day of the week, and month of the year, etc. As many factors influence the produced data, its fluctuations may not follow a fixed frequency. It is customary in forecasting to paradoxically refer to "cyclic" pattern, when there is no fixed frequency involved in the time-series. While, if data follows an unchanging frequency with respect to regular time intervals (e.g. day), then it is a *seasonal* pattern. Our very first effort is to identify and reduce seasonality. We focus on daily seasonality for the rest of this work.

In this chapter, we first, introduce types of time-dependent intensity in wireless networking, namely *busy* and *idle (non-busy)* hours. The particular channel utilization trace from a Wi-Fi channel at one particular location corresponds to the dynamics of the wireless users present at the particular location. We emphasize that one can employ a *naïve* predictor, i.e., one that measures the recent utilization and uses it as the prediction for the next short-term interval, which often works quite

well in reality. This is not surprising given the catatonic state of utilization during time intervals (e.g. hours) without lots of user activity, what is termed idle or non-busy hours. As an example, consider Figure 3.1a which concatenates the utilization values of the channel in the 12-12:30 AM period across 90 days (where 12-12:30 AM is typically an idle hour). Their strong similarity across days is typical. On the other hand, a distinguishing feature of busy hour traffic is the increase in the channel utilization values as multitude number of users operate on the wireless channels. In busy hours the traffic seems bursty due to the inherent best-effort behavior of higher layer protocols which are typically used (e.g. UDP) and the sum total of user demand for traffic. Consequently, the range of channel utilization fluctuations in the busy hours is wider than during idle hours. Figure 3.1b concatenates the utilization values over 90 days for the 3-3:30 PM interval, capturing a typical busy hour behaviour. A busy "hour" is a concept that does not correspond always to a 60-minute duration. Rather the term "hour" is used generically for any particular interval of appreciable duration that users and engineers would find the behaviour of a channel (or any resource that matters) punctuated by a particular high-use pattern. While the similarity across 90 days is evident, there are also gradual shifts in the utilization patterns, making the separation between busy and idle hours less crisp. To put it differently, idle and busy hours follow various distributions which we will attempt to track in our models. The subtle shift and change in utilization patterns causes the distributions of the aforementioned traffic variations to get mixed, complicating the process of recognizing some particular patterns. We, will subsequently, investigate a metric allowing us to (a) express the stochastic behavior of the time series over short time intervals, and (b) to allow the comparison of the behavior across different intervals.



(a) Concatenated idle hour (12am-12:30am) utilization.

(b) Concatenated busy hour (3pm-3:30pm) utilization.

Figure 3.1: Concatenated time series of, respectively, typical idle and busy hour utilization on channel 1 over 90 days.

3.1 Interval Comparison Metric

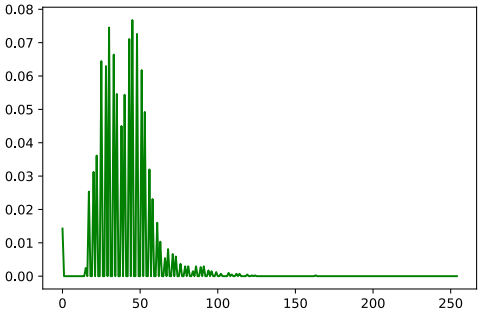
In this work, we split time in τ minutes long intervals, and the dynamics of utilization of a channel over each, separate, τ interval is modeled by a Markov Chain. We use the notation $\mathbf{P}_\tau^{(i)}$ to indicate the transition matrix for the i–th τ minute long interval. For example if $\tau = 30$ then, within a single day, i ranges from 0 to 47, starting from 12:00 AM. The $\mathbf{P}_\tau^{(i)}$ matrices are constructed as first-order Markov approximations from the observed utilization transitions. The self-transitions in $\mathbf{P}_\tau^{(i)}$ correspond to behavior that allows naïve predictors to be successful, i.e., the naïve predictor success is captured by the dominance (if the case) of the diagonal elements. Corresponding to each $\mathbf{P}_\tau^{(i)}$ is a steady-state distribution $\pi_\tau^{(i)}$. In order to compare the transition matrices of τ minutes intervals, we translate transition matrices, $\mathbf{P}_\tau^{(i)}$, to their corresponding steady-state vector, $\pi_\tau^{(i)}$. The steady-state vector captures the probability to find the system in a particular state over the long run. Similarities among the steady-state vectors of each interval can be seen as similarity among the corresponding transition matrices. Having the steady-state vectors out of the

transition matrices for each τ minutes interval, we seek a distance metric for the stochastic behaviour between any two intervals. The dot product of $\pi_\tau^{(i)}$ vectors acts as a distance metric of the stochastic behavior between any two intervals (see Figure 3.2 and 3.3). Among many methods available and tested to measure similarity between two non-zero vectors, we decided to use the dot product. The intuition behind this choice is that we are more interested in overall state-wise difference of two vectors instead of measuring their overall distance. To put it differently, while dot product of the vectors calculates the coincide area under the curve of them, methods like cosine and Euclidean distance measures, respectively, the cosine of the angle between two vectors and straight line distance between them. We tested different distance methods such as cosine, Hamming, and Euclidean, we found the dot product of two vectors as an adequate similarity measurement method for our purpose. Figure 3.2 shows coincide of two similar vectors, both from idle-hours, with similarity of 99%. On the other hand, Figure 3.3 shows coincide of two dissimilar vectors, one from busy-hours and the other from idle-hours, with similarity of 53%, i.e. essentially dissimilar.

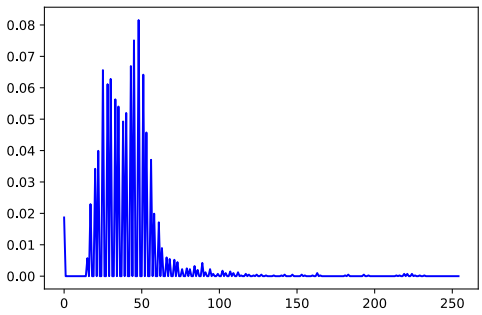
Table 3.4 shows the dot product of $\pi_5^{(0)}$, an idle-hour steady-state, with the rest of steady-state vectors for $\tau = 5$ minutes, over the entire day. As can be seen in this table, as time goes from idle hours to busy, the inner product of the vectors decreases. The results show how this metric can be helpful in identifying the relation of various times of the day, via the dot product of the corresponding τ minute interval steady state.

3.2 Seasonality Reduction

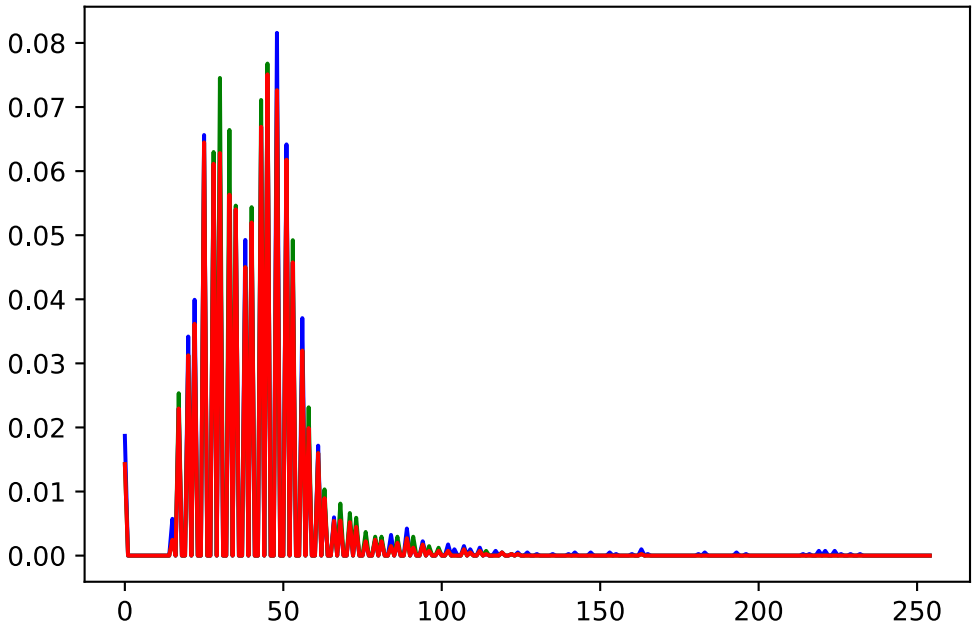
The notion of busy and idle hours is different during the weekends and holidays from the weekdays. In this work, we focus on building a model for predicting channel utilization values over weekdays as we expect they contain complex patterns. Without



(a) π_5^0 12am-12:05am over 90 days.

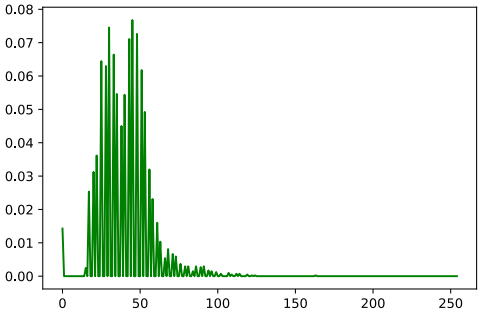


(b) π_5^1 12:05am-12:10am over 90 days.

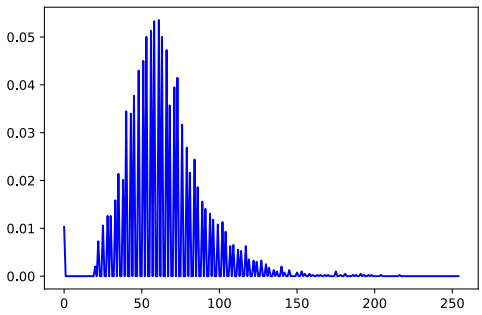


(c) Dot product of π_5^0 (green) and π_5^1 (blue) – red indicates area of co-incidence.

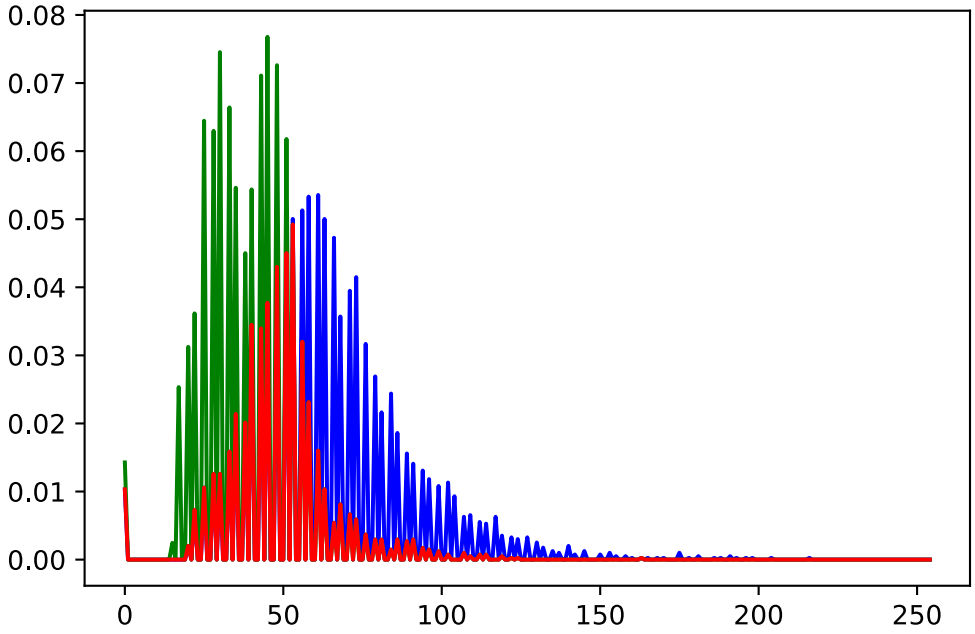
Figure 3.2: Dot product of $\pi_5^{(0)}$ with a similar (similarity 99%) steady state vector, $\pi_5^{(1)}$.



(a) π_5^0 12am-12:05am over 90 days.



(b) π_5^0 12pm-12:05pm over 90 days.



(c) Dot product of π_5^0 (green) and π_5^{144} (blue) – red indicates area of co-incidence.

Figure 3.3: Dot product of $\pi_5^{(0)}$ with a dissimilar (similarity 53%) steady state vector, $\pi_5^{(150)}$.

		Minutes Of An Hour											
		00''-05''	05''-10''	10''-15''	15''-20''	20''-25''	25''-30''	30''-35''	35''-40''	40''-45''	45''-50''	50''-55''	55''-00''
Hours Of Day	0'	1.0	0.9949	0.9944	0.9938	0.993	0.9904	0.9915	0.993	0.9926	0.9899	0.9859	0.9924
	1'	0.9901	0.9846	0.992	0.9828	0.9869	0.9911	0.9932	0.9866	0.9899	0.9881	0.9864	0.9918
	2'	0.9966	0.9938	0.9918	0.9898	0.9906	0.9901	0.9874	0.9882	0.9907	0.9936	0.9887	0.9892
	3'	0.9886	0.9737	0.9929	0.9904	0.9917	0.9891	0.9912	0.9923	0.9915	0.9887	0.9907	0.9915
	4'	0.9887	0.9926	0.9932	0.9941	0.994	0.9887	0.9896	0.995	0.994	0.9898	0.9902	0.9887
	5'	0.9887	0.9856	0.9836	0.9884	0.9929	0.9868	0.9882	0.9852	0.9827	0.9861	0.9861	0.9878
	6'	0.987	0.9846	0.9851	0.9829	0.9752	0.9777	0.9821	0.9798	0.9743	0.9719	0.9688	0.9778
	7'	0.9653	0.9702	0.9683	0.9601	0.9409	0.9281	0.9272	0.9061	0.8915	0.8559	0.8322	0.8588
	8'	0.8632	0.8572	0.8789	0.8549	0.8469	0.8334	0.8465	0.8458	0.8068	0.8057	0.7788	0.7834
	9'	0.8236	0.8313	0.8038	0.8051	0.7879	0.8223	0.7986	0.8009	0.7691	0.7715	0.7645	0.7386
	10'	0.7348	0.7469	0.7088	0.7228	0.7137	0.6993	0.6883	0.6603	0.6488	0.6427	0.6296	0.6488
	11'	0.6171	0.6032	0.6068	0.6306	0.587	0.5968	0.5932	0.5994	0.5931	0.5482	0.5467	0.559
	12'	0.5397	0.5364	0.5725	0.597	0.623	0.6036	0.6063	0.6076	0.6383	0.6364	0.5823	0.6115
	13'	0.5998	0.6262	0.614	0.6141	0.6008	0.6097	0.5944	0.6305	0.5957	0.6077	0.6277	0.5989
	14'	0.6271	0.6571	0.6777	0.6915	0.6576	0.6841	0.6865	0.6654	0.6927	0.703	0.6929	0.7123
	15'	0.7106	0.6871	0.6966	0.7194	0.7259	0.7293	0.729	0.7064	0.7374	0.7172	0.7445	0.733
	16'	0.7273	0.7613	0.7487	0.7665	0.764	0.7719	0.8145	0.805	0.7874	0.8364	0.8303	0.8183
	17'	0.7924	0.8106	0.8371	0.8508	0.8707	0.8645	0.8728	0.8877	0.8645	0.859	0.8715	0.8701
	18'	0.8705	0.8921	0.8988	0.9145	0.9187	0.918	0.9229	0.9019	0.9058	0.9169	0.905	0.9198
	19'	0.9063	0.8999	0.9274	0.93	0.9182	0.9233	0.9257	0.9221	0.9348	0.9393	0.9505	0.9521
	20'	0.9451	0.9391	0.9591	0.9611	0.9654	0.9538	0.9624	0.96	0.9677	0.9434	0.9555	0.9673
	21'	0.9659	0.9746	0.9764	0.9701	0.9726	0.9825	0.9806	0.9842	0.9836	0.9819	0.9839	0.9759
	22'	0.984	0.9883	0.9826	0.9821	0.9838	0.9858	0.9873	0.9803	0.9877	0.9808	0.9838	0.9825
	23'	0.9829	0.9907	0.99	0.9883	0.9893	0.9852	0.9892	0.9916	0.9927	0.9853	0.9904	0.9917

Figure 3.4: Dot product of $\pi_5^{(0)}$ and $\pi_5^{(i)}$ using data collected over 90 days. Each time interval to which an i corresponds is shown as hour of the day (row) and five-minute interval within the hour (column).

hurting the generality of our result to weekends and holidays, all the remaining work focus on daily traffic patterns on workdays.

3.3 Condensing the Intraday Seasonality

Let us split the time into τ-minutes intervals. Finding an accurate value for τ in order to separate the seasonality variations within a day is a key ingredient of this work. Choosing a fixed and small value of τ simplifies the problem. To put it differently,

we can use a small τ to be representative of each particular type of day across days without effects of seasonality. We can then build models separately for τ minutes interval of the day. The intuition behind dismantling the time into intervals is to remove some significant seasonality impact coming from changes in number of user which plays a role in utilization fluctuations throughout a day. The task of choosing τ values is a trade-off between fidelity of the patterns spotted over time and having sufficient samples/ measured data to compute the π. In other words, by increasing τ, existing patterns in idle and busy hours might get mixed in the same interval. However, the longer the chosen τ is, the more observed data we have to build an accurate model around it.

As we do not want to lose any pattern within a day (especially within busy hours), smaller τ would be more helpful. But, it leads to a sparser transition matrix due to having few samples (possibly even no samples for some of the 256 states) which is unfavorable. Note that when determining the transition probabilities of $\mathbf{P}_{\tau}^{(i)}$, we use the transitions only for the CU values we have collected in the corresponding τ minutes interval.

Now, the question is that having no knowledge about the environment the data is coming from, how can we choose a value for τ which aids us the most to reduce the seasonality within a day? We will explore it later by experiments with τ values less than an hour (τ = 60-minutes) which do not affect the overall accuracy by much. However, to answer the raised question, in the next subsection we propose a procedure to evaluate the goodness of a τ value by using the metric, dot product of steady-state of intervals, introduced earlier in this chapter.

3.3.1 τ Evaluation Benchmark

One approach to distinguish the traffic variations from each other would be to choose flexible values for τ, meaning different τ values for each part of the day. This approach

is based on the premise that idle hours are similar to each other, and separately similar to each other are the busy hours. Using the dot product as similarity metric, we can attempt to generate "merged intervals" (MIs). MIs, are similar intervals, that, at least in principle, can appear at completely different times of the day. Pragmatically, we expect that similar intervals appear at the same periods of the day, e.g., when traffic is picking up in the beginning of working hours and when it is dying out at the end of working hours. We will use the MIs to create a benchmark across all collected data. The intuition behind this assumption is that the MI approach is aware of the whole collected data and the metric helps us to distinguish intervals based on their similarity across the whole data set without regard from when in the time series they may have appeared.

In this process, we first, choose a small value for τ, then, produce the transition matrix $\mathbf{P}_{\tau}^{(i)}$ for each interval in the whole dataset. As an example, for $\tau = 5$ minutes, we get $\frac{\mathbf{60_{minutes-in-an-hour} \times 24_{hours-in-a-day}}}{\mathbf{5_{value-of-\tau}}} = 288$ intervals within a day, starting from 12:00 AM. Taking the first one, we will approximate the dynamics by a first order Markov chain where the transition matrix of time interval 12:00 A.M. to 12:05 A.M., shown as $\mathbf{P}_5^0$. We will merge this interval with one whose steady-state vectors is the most similar. In this approach we are not bounded to merge only neighbouring intervals. We choose τ as small as 5 minutes in that it is small enough to not loose any pattern within a day and also we have enough samples to create dense transition matrices over all days of collected data.

Let us denote α as the dot product of two steady-state vectors, belonging to two different intervals. α is a value between 0 and 1; the higher it is, the more similar two steady-state vectors are. The process we use to derive a small set of clusters proceeds in a greedy fashion, "merging" the intervals that are the most similar to each other. This merging operation involves the summing (with suitable normalization) of the transition matrices of the corresponding intervals. Thus the intervals in the

same cluster now have a, single, combined transition matrix, and therefore a single steady-state vector. Correspondingly, the α dot product is naturally extended from being between intervals, to being between two clusters, or between a cluster and an interval. The process of merging clusters proceeds in a greedy fashion as long as there are pairs of clusters whose dot product is less than a given α^* or if there are more than a certain maximum, CLUSTER_MAX, number of clusters. In our experiments $\alpha^* = 0.98$ and CLUSTER_MAX=10. Nevertheless, it was observed that the CLUSTER_MAX constraint, which would allow for cases with $\alpha > 0.98$ to be merged to reduce the number of clusters, had no impact.

The algorithm adopts the same idea of K-means [21], with a difference in the assigning a predefined threshold value instead of choosing a specific number of clusters, K. In our cases K is a desired maximum number of clusters, if the threshold is too tight to allow for merging to happen. Similar to the K-means, no technique is available to cluster the intervals in a more optimal way, neither recursive approach nor dynamic programming [24]. Moreover, same as K-means, changing the sequence of clustering points in space (steady-state vectors in our case) could end up with different results.

Chapter 4

Feature Engineering

In the previous chapter, we discussed types of time-dependent behavior in wireless channel utilization and we tried to reduce their seasonal effect on time-series by dismantling time into intervals. We further introduced inner product of steady-state vectors as a metric allowing us to (a) express the stochastic behavior of the time-series over short time intervals, and (b) to allow the comparison of the behavior across different intervals.

In this chapter, we investigate other features of our time-series to enhance the forecasting capability of a predictor. Our goal is to provide important features to the learning algorithm in order to increase its accuracy in predicting channel utilization. Our main focus in this chapter is on finding features which help the learning algorithm better capture the dynamics of the utilization time series data.

4.1 Simple Prediction Methods as Benchmarks

The incremental way of building the training data for a predictor, such as a Neural Network, allows us to capture the intuitively simple, and often effective, naïve predictor. But the naïve predictor will be one of the mechanisms selected as feature; allowing us to see when it is relevant and when not. We first need to understand

the behaviour of a very simple predictor. Sometimes these predictors, as simple they are, are good forecasting methods; however, in many cases, they solely serve as a benchmark as they are incapable of spotting complex features of time-series, that can be often non-linear. Consequently, in this work, we compare our proposed models with these simple ones to ensure that our method outperforms all the simpler alternatives. These simple methods also reveal the basic features of time-series important in prediction. By spotting these basic features, we later can incrementally add features to make a powerful and more complicated, predictor. First, we review some of the standard, well-known, forecasting methods [1].

Averaging Method

This method predicts h future values by the average (mean) of the n previous data. An alternative is *running average* which involves all the previous historical data in the prediction of the next unseen data. As previous, we denote historical data for time T as $y_{T-n}, ..., y_T$ and the prediction of the next h time as $\hat{y}_{T+h}$; so for the average method we have:

$$\hat{y}_{T+h} = (y_{T-n} + y_{T-(n-1)} + ... + y_T)/T \tag{4.1}$$

Naïve Method

The naive method forecasts all the h future values as the value of the last observation. So for this method we have:

$$\hat{y}_{T+h} = y_T \tag{4.2}$$

Naïve - Drift Method

A naïve - drift method is a variation of the naïve method that allows the forecasts to fluctuate over time. The amount of change over time (drift) is set to be the average

change seen in the data.

$$\hat{y}_{T+h} = y_T + \frac{h}{T-1} \sum_{t=T-(n-1)}^{T} (y_t - y_{t-1}) = y_T + h(\frac{y_T - y_{T-n}}{T-1}) \tag{4.3}$$

As we will discuss later, information from the naïve method aids a non-linear predictor to increase its accuracy. To this end, for the rest of the work, we use the naïve method as a testing benchmark against our model. Moreover, as it is one of the simplest prediction methods, we incrementally add other features described in this section, on top of the naïve method. We denote U as the previous class of utilization value from 0 to 255.

4.2 Auto-Regressive Models

In the general case, we deal with two variations of forecasting: using only the previous values of the time-series for predicting future values which is called *univariate time series forecasting*, and using predictors other than the series (a.k.a. exogenous variables) for prediction which is called *multi variate time series forecasting*. Auto-Regressive (AR) models, is a forecasting algorithm solely using the information from the previous historical time-series data for predicting the future values.

The AR model [16] attempts to find patterns in a *stationary* time-series data by taking advantage of *auto regression* in order to involve its lags as features. A stationary time-series is the one whose properties do not depend upon the time period over which the series is observed. Consequently, time-series with trends, or seasonality are not stationary; however, cyclic behaviour in time-series data with no trend or seasonality is still stationary.

The technique in AR for translating a non-stationary time-series into a stationary one is to, first, take the logarithm of observations to stabilise the variance and, second, compute the differences between two consecutive observations at a certain

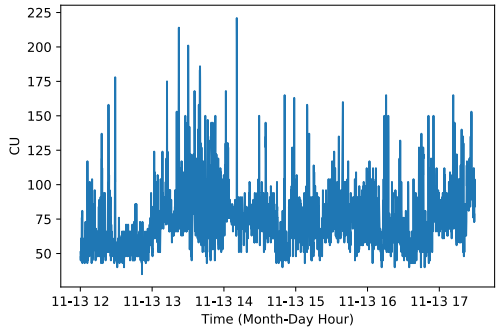
lag value, L, (a.k.a. differencing) to stabilise the mean of the time-series. The technique of differencing the logarithm (DL) values of observations at time t, is $DL = \log(CU_t) - \log(CU_{t-L})$. The end result of applying DL on a time-series is a stationary time-series, shown as Σ. We use the notation of Σ_L for a stationary time-series with lag L. Within Σ, all the trends and seasonality involved in the original time-series will be diminished or eliminated. It is possible more than one differencing would be needed to convert a non-stationary time-series into a stationary one. The auto-correlation function (ACF) of a stationary time-series reaches to zero fairly quickly while the ACF of a non-stationary time-series decreases slowly. This is used as the criterion to decide if further differencing is necessary.

In AR models, finding Σ is not the end of the story. AR uses an auto-regression to create a linear combination of predictors at different lags. Later enhancements of AR model, as discussed in Chapter 2, like ARIMA (abbreviation of "Auto-Regressive Integrated Moving Average") use the same idea, restricted to the relations between output value and previous values at specific lags. The stronger the correlation between the output value and a specific lagged value, the more weight the AR model assigns to that. If all lag values show low or no correlation with the output, then the AR model is not appropriate for predicting a time-series problem. Non-linear predictors can take advantage of the existing correlation between the output value and previously observed values in Σ. We endeavour for the model to capture the correlation between observed Σ_L values by incorporating them as features to the model. In the AR model, the lag value, L, was assumed as 1 since the goal was to predict exactly the next step. For several-step ahead prediction, the lag matches the number of steps. The question at this point is how many steps from the historical data we need to incorporate in our model. The very naïve answer would be the more information we provide to the learning algorithm, the better it can do the prediction. However, this answer is not quite informative as each lagged term has a coefficient/weight.

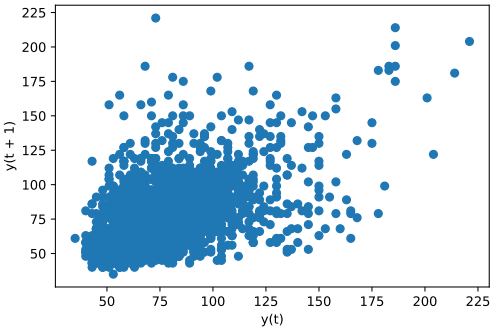
Hence adding more lags may cause the coefficient of the most recent lags or most important ones to diminish. On the other, going farther back in time provides the model with more information. In time series analysis, the Box–Jenkins method [6], applies ARMA or ARIMA models to find the best fit of a time-series model to past values of it. This method [6] aids us in determining the number of lags, β, going back in time that we need to retain.

In our problem, we are dealing with more than one trend and seasonality. The known ones, as mentioned before, are busy and idle hours. We tried to eliminate this seasonality by slicing the time of a day into τ minutes intervals. However, it is not enough to diminish all sources of seasonality and trends involved in the final channel utilization value. Some of the trends and seasonality in our data that we are aware of their existence are influenced by factors such as the number of users utilizing an access point (AP), interference created by other APs on the one we investigate, and so on. To decide on the non-stationarity of the time series, we observed its ACF. In examples, within a random day of the collected data, we take one interval from idle hours and one from busy hours to show the impact of seasonality and trend within data. Assuming $L = 1$, Figure 4.1 and 4.2 show the data and its ACF within a day, from 12:00-5:30 AM (as idle hour) and 12:00-5:30 PM (as busy hour) respectively. As can be seen in figure 4.2, within busy hours the ACF slowly converges to 0 after more than 1000 lags. However, for the idle hours (figure 4.2), its ACF converges to 0 fairly quickly, after some lags. It shows that, within idle hours, we do not have much seasonality or trend in our time-series. On the other hand, as for the busy-hours, we need to decrease the effect of seasonality and/or trend, so that the learning algorithm can better identify the pattern within data with no seasonality and trend involved.

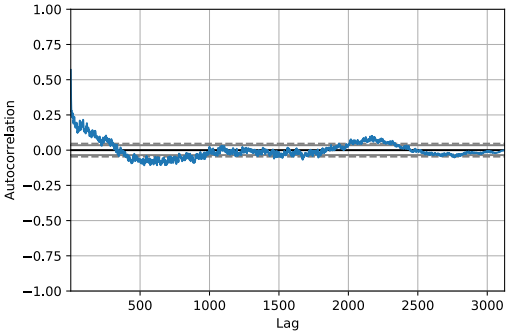
We apply the standard DL technique on the time-series. The DL of the same busy and idle hours intervals are shown in Figures 4.3 and 4.4. As for the busy hours, this technique reduced/eliminated the effect of seasonality and/or trend; so that, the



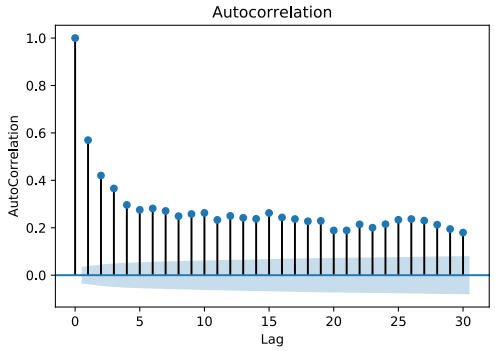
(a) CU 12pm-5:30pm (busy hours) of a randomly chosen day.



(b) Relation of $y(t)$ and $y(t+1)$ for the data in (a).



(c) ACF for the data in (a) for all lags.

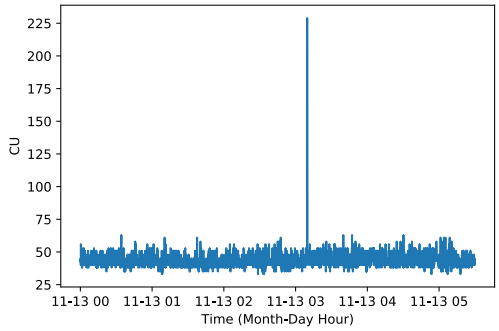


(d) First 30 lags of ACF for the data in (a).

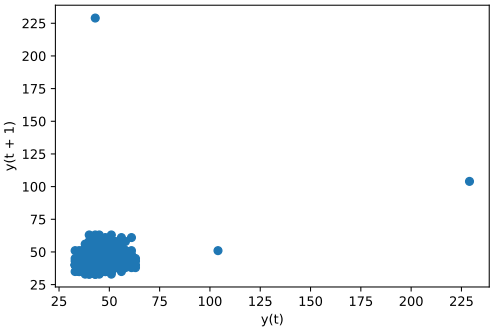
Figure 4.1: Example auto-correlation of CU values during a busy hours interval (12pm-5:30pm) of a randomly chosen day.

resulting values shown in Figure 4.3 are essentially stationary as the ACF suggests. It has no effect on the idle hours as it was stationary in the first place.

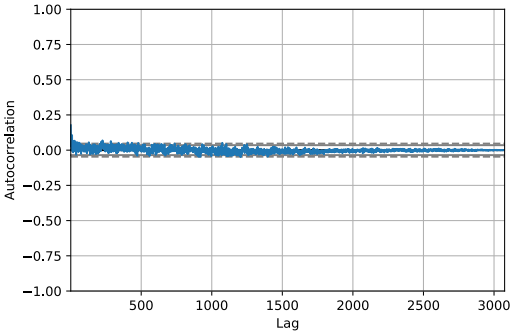
Box–Jenkins provides us with number of lags (e.g. determined to be β =47 for τ =30 minutes) we need to retrieve from the Σ_L. At time t, by providing all values from Σ_L^t to $\Sigma_L^{t-\beta}$ as features to the learning algorithm, we hope it can unearth existing pattern in utilization fluctuations within time-series with less or without seasonality and trends involved. We will call these values the *stationary values* and use them as features, with the understanding that unless we perform multi-step prediction, they



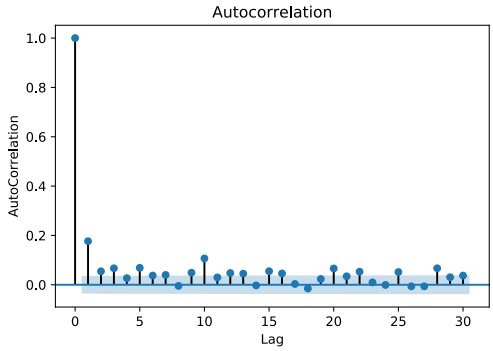
(a) CU 12am-5:30am (idle hours) of a randomly chosen day.



(b) Relation of $y(t)$ and $y(t+1)$ for the data in (a).



(c) ACF for the data in (a) for all lags.



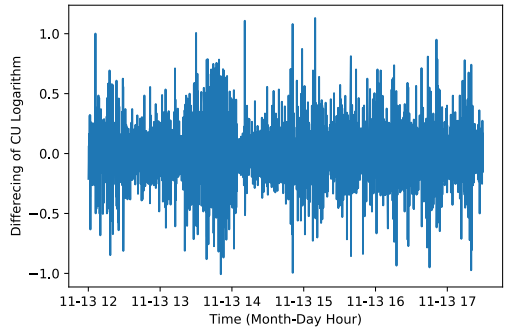
(d) First 30 lags of ACF for the data in (a).

Figure 4.2: Example auto-correlation of CU values during an idle hours interval (12am-5:30am) of a randomly chosen day.

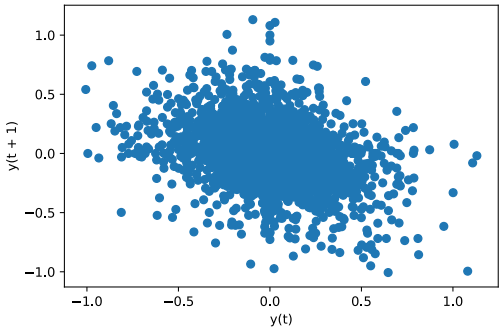
boil down to just one value, the one at lag one.

4.3 Short-Term Dynamics Distance

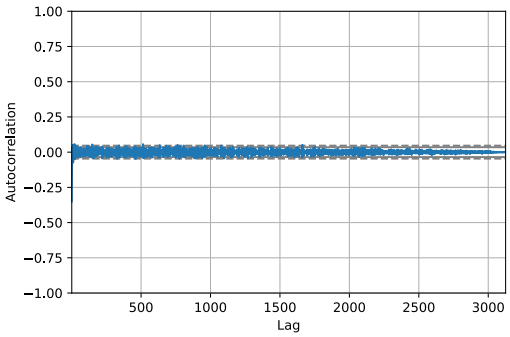
In the previous chapter, we investigated the metric of dot product of two steady-state vectors for capturing intervals to each other. Experiments revealed that encoding the exact time offset within intervals (i.e. exact time within the current τ minute interval) does not help the learning algorithm in forecasting. However, the separation



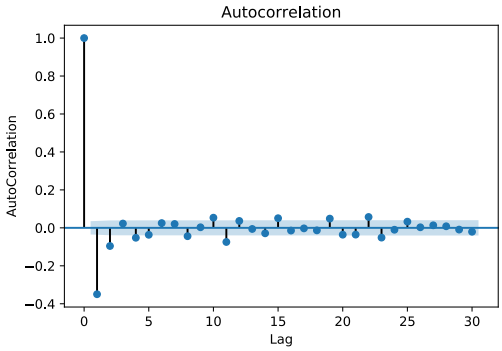
(a) DL of CU 12pm-5:30pm (busy hours) of a randomly chosen day.



(b) Relation of $y(t)$ and $y(t+1)$ for the data in (a).



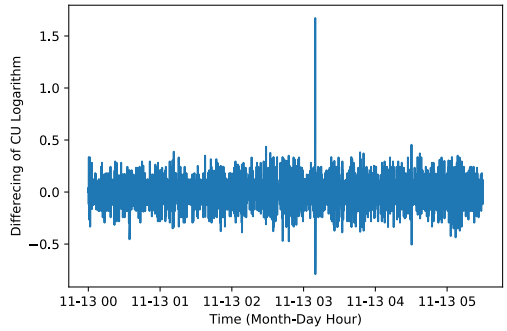
(c) ACF for the data in (a) for all lags.



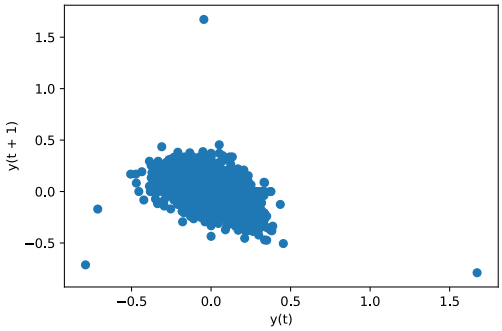
(d) First 30 lags of ACF for the data in (a).

Figure 4.3: Example auto-correlation of DL of CU values during a busy hours interval (12pm-5:30pm) of a randomly chosen day.

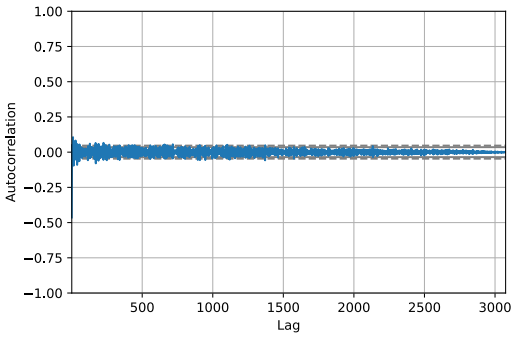
of busy and idle hours gives the model a rough idea of which part of a day the observed data is from, without giving the exact time of the day. The pertinence of the dot product as a feature can be seen by the example in Figure 4.5 in which the first twelve 5 minute intervals of the day (from midnight to 1 AM – presumably an idle hour) are compared, via the dot product, across the entire day's 5 minute intervals ($\tau = 5$). Notice that particular periods of time (distance metric in the 0.7 to 0.9 range) suggests that there are periods where neither the idle nor the busy hour behavior prevails, i.e., they correspond to intervals with a mix of behaviors. Similar



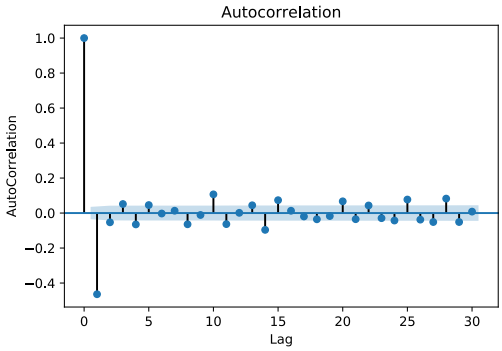
(a) DL of CU 12am-5:30am (idle hours) of a randomly chosen day.



(b) Relation of $y(t)$ and $y(t+1)$ for the data in (a).



(c) ACF for the data in (a) for all lags.



(d) First 30 lags of ACF for the data in (a).

Figure 4.4: Example auto-correlation of DL of CU values during an idle hours interval (12am-5:30am) of a randomly chosen day.

plots can be created if one selects an interval from the busy hour to compare against. We will subsequently use the dot product metric of the most recent interval steady state approximation against each of a library of steady-state approximations from collected data, each for a specific τ minutes interval within a 24 hour period. The point being that an observer can create such collections over the entire period that it is observing the utilization of the channel. As we later find out, the distance metric to other prior intervals is in itself an implicit indicator of time, as it results in expressing the influences of other intervals on the current interval. Additionally, it turns out

that the steady state of a "typical" day, meant as a representative collection of the dynamics of the same time period across all observed days, is sufficiently powerful to express the relation of the current interval to other intervals.

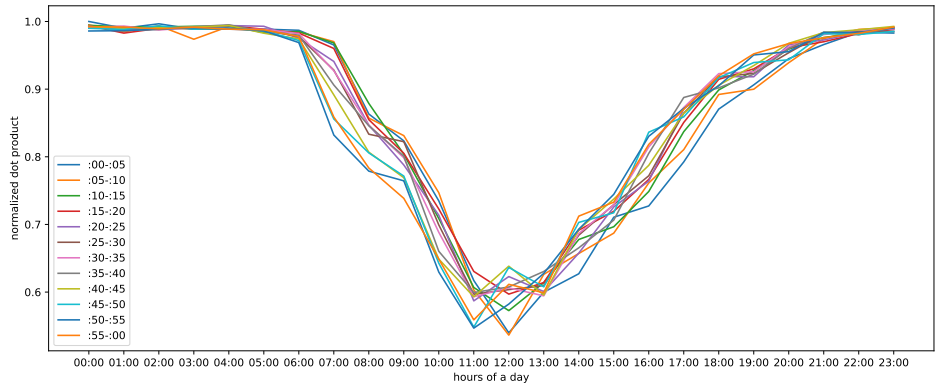


Figure 4.5: Dot product of $\mathbf{P}_5^{(j)}$ with $\mathbf{P}_5^{(i)}$ within a day ($j = 0, ..., 11$ with one line for each j and $i = 0, ..., 287$).

In addition, Figure 4.6 shows how dot product of $\mathbf{P}_5^{(0)}$ and $\mathbf{P}_5^{(i)}$ changes within an hour. Fluctuations individually are not following an overall observable pattern within an hour; however, within a day, their correlation decrease as they are further apart in time.

The idea is to inform the learning algorithm about the time of a day indirectly by changes in dot products of intervals. We expect the learning algorithm understand the patterns within intervals and estimate the time of the day of each observation within data. So, the idea is to incorporate the dot product of previously observed τ minutes steady-state vectors with the day library ones as features into the learning algorithm. We use the notation Δ_i for showing the timing features where $i = 0, ..., j$ and j is the number of intervals existing in day library. By doing so, we expect the data better capture the time of the day and use it in prediction.

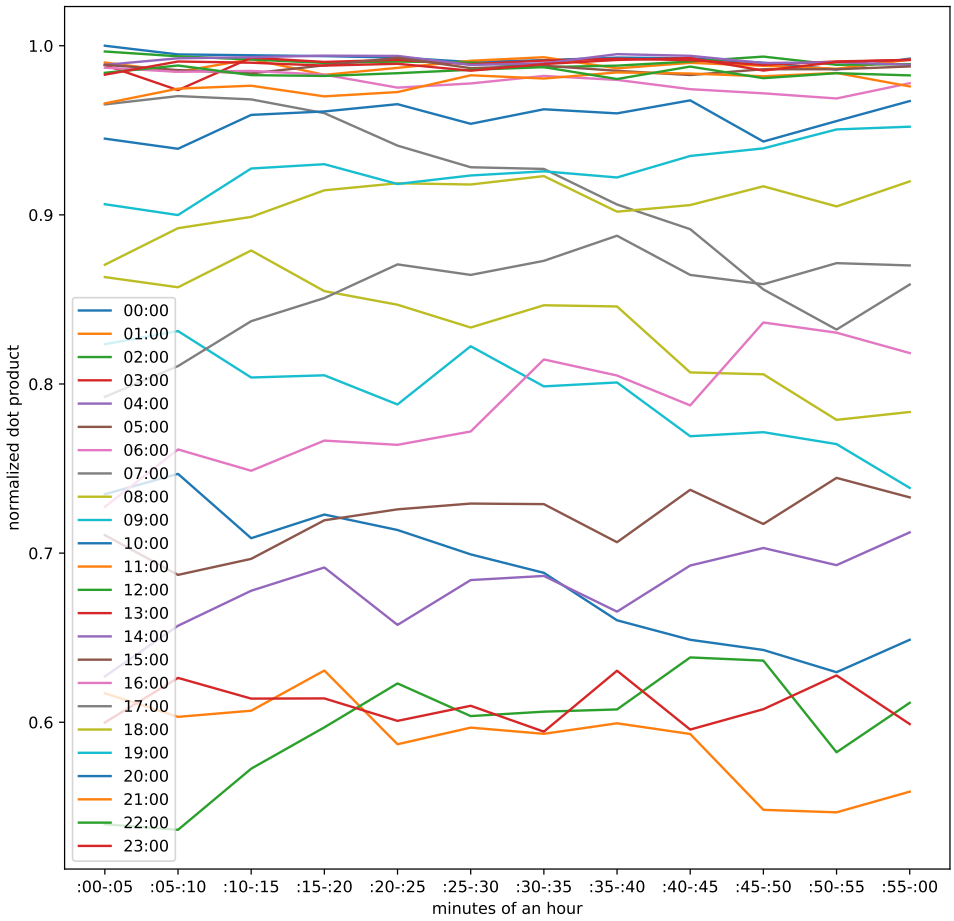


Figure 4.6: Dot product of $\mathbf{P}_5^{(j)}$ with $\mathbf{P}_5^{(i)}$ within an hour ($j = 0, ..., 11$ with one line for each j and $i = 0, ..., 287$).

4.4 Selected Features

In this chapter we went through the features important for the learning algorithm to better learn patterns within environment and data. Towards this end we considered:

- transforming non-stationary data into stationary, and,
- separating, for the sake of seasonality, idle from busy hours.

The features we will subsequently use in the learning algorithm are:

- the value of observed channel utilization in the stationary time-series, shown as Σ_L^t where L and t refer to lag value and number of lags we retrieve respectively,
- the dot product metric against the previous interval and against a library of a day's worth of intervals, shown as Δ, and, finally,
- the previously seen channel utilization, U, understood so far to be the current level of utilization, but, as shown in the next chapter, made into coarser utilization ranges.

Chapter 5

State Lumping and the Cost of Misprediction

In previous chapters, we, first, investigated a metric allowing us to group time intervals with similar stochastic behavior and/or to compare the behavior of the time serious between intervals. We also reduced the seasonality and trends within the time-series by splitting a day into τ minutes interval. Finally, we identified features by which our model can better adopt to the environment and to perform forecasting.

Our measurements collect fine-grained values of CU, between 0 and 255. For most applications, an accuracy of 100/255th of a percentage is unnecessary. We adopt a summarized version of the transition matrices $\mathbf{P}_{\tau}^{(i)}$, reducing them from 256×256 to a smaller size (typically from 5×5 to 7×7) by lumping together ranges of utilization values. The decision to lump the matrices is also helpful in one more way: a limitation of short τ intervals is that it is unlikely that all 256 utilization values are observed during any particular interval, let alone enough pairs of successive values to help build a sufficiently rich transition matrix for a first-order Markov model of each τ interval.

Consequently, in this chapter we first, introduce a method of state grouping, namely "state lumping". Then, we address the problem of misprediction cost within

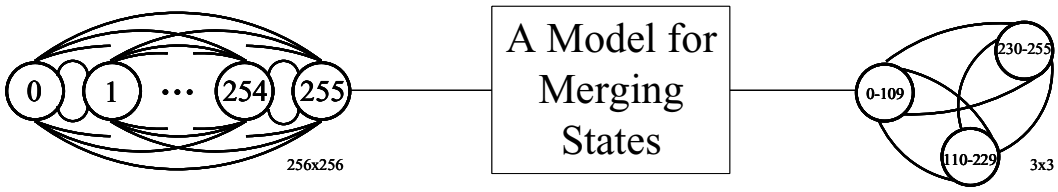


Figure 5.1: An example of state space reduction of the transition matrix leading to a 3×3 lumped matrix. For visual clarity, self-loops and arrows are not shown. State numbers/ranges refer to the corresponding CU values represented by a state.

the established lumped states.

5.1 Simplifying $\mathbf{P}_{\tau}^{(\mathbf{i})}$ and Lumpability

Consider time progresses in time steps of length σ, then the evolution of the channel utilization of an AP on a specific channel from time t to $t + \sigma$ will be approximated by a first-order Markov chain. We can trivially produce estimates for the transition probabilities of the transition matrix, $\mathbf{P}_{\tau}^{(i)}$, by counting the corresponding transitions between successive CU values from the collected data, and normalizing them accordingly. Technically, $\mathbf{P}_{\tau}^{(i)}$ is a 256×256 matrix, but we will take several steps to simplify it further as described in the next subsections. Our objective is to start from a 256×256 matrix, for each τ minute interval, and reduce the model to a small (no more than 7×7) matrix which we will subsequently use for predicting the next state.

5.1.1 Lumpability

State lumping of a Markov chain reduces a large state space into a smaller one [7]. A new, produced, Markov chain has states that result from combining states of the original chain together in one state. The new chain has approximately the same behaviour as the original one but in a coarser level of detail.

A discrete time Markov chain is lumpable with respect to a given state space partition $S = \bigcup_i S_i$ with $S_i \bigcap S_j = \varnothing \ \forall i \neq j$ if its transition probability matrix $\mathbf{P}_\tau^{(i)}$ satisfies the lumpability condition:

$$\forall S_i, S_j \subset S \quad \forall s \in S_i : \quad \sum_{s' \in S_j} p_{s,s'} = k_{i,j} \quad \forall i, j, \tag{5.1}$$

where $p_{s,s'}$ is the one-step transition probability from state s to state s', and $k_{i,j}$ is a constant depending only on i and j. The $k_{i,j}$ are the elements of the lumped matrix, $\mathbf{K}$. That is, a Markov chain is lumpable if the transition probability from each state in a given partition to another partition is the same. The probability of transitioning from a given state to a partition is equivalent to the sum of the transition probabilities from the given state to each state in the partition. In the lumping process, the transition probability from each state of a partition to another partition must be the same. Moreover, for each state, the probability of a transition to a partition is the sum of transition probabilities from the state to each state in that partition.

Figure 5.2 clarifies a 2×2 brute-force lumping procedure. At the very top of the figure, the original transition matrix is shown. We need to find all possible permutations of the original matrix states, four such permutations are shown in the figure. Then for each of the permuted ones, we take different partitions to find the one which satisfies the lumpability condition. The final lumped matrix has the partitions of $S = \{1, 3\} \bigcup \{2, 4\}$ ($S_1 = \{1, 3\}, S_2 = \{2, 4\}$) with $k_{1,1} = 0.6, k_{1,2} = 0.4, k_{2,1} = 0.7$, and $k_{2,2} = 0.3$. It is worth to highlight that the lumping procedure shown in the figure suggests a brute-force approach, which means many of the generated groups are calculated more than once. Hence, it is not an optimized algorithm and it is depicted this way just for the sake of clarifying the notion of lumping. As it is shown in the figure, only one of the partitions satisfies the lumping conditions. However, in many real cases none of the produced partitions will meet this condition.

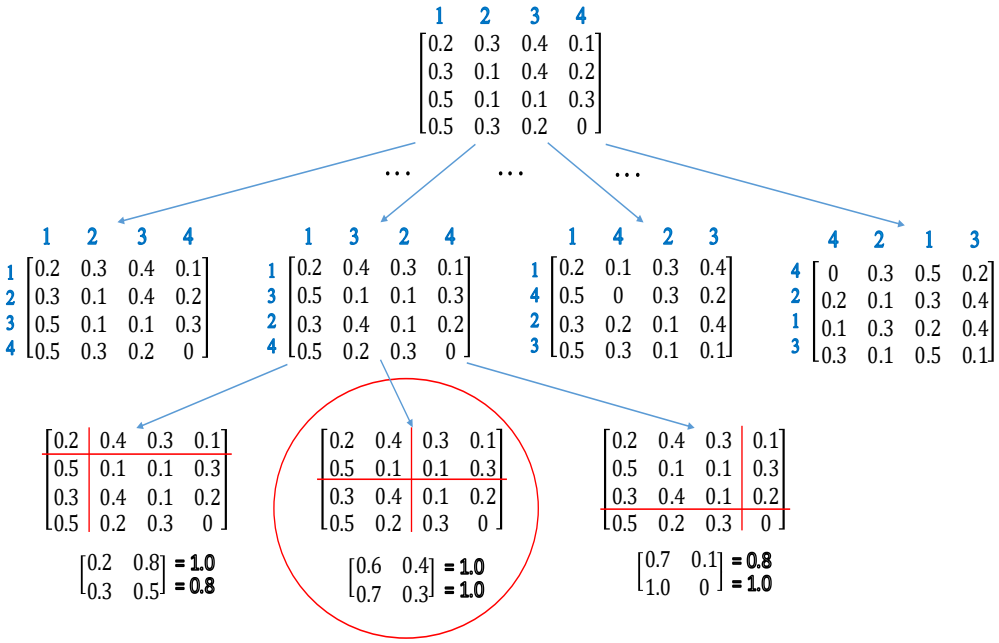


Figure 5.2: An example of lumping.

5.1.2 Quasi-Lumpability

Markov chains are seldom lumpable, in that we cannot generally satisfy the equals relation in Equation 5.1 regardless of how we group the state space. In those cases, we adopt the relaxed definition known as quasi-lumpability [10]. Intuitively, quasi-lumpability allows for the sum of rows of the resulting lumped matrices to not sum up to 1. Formally, a Markov chain is ϵ quasi-lumpable with respect to a given state space partition $S = \bigcup_i S_i$ with $S_i \bigcap S_j = \varnothing \; \forall i \neq j$ if its transition probability matrix $\mathbf{P}$ can be written as $\mathbf{P} = \mathbf{P}^{-} + \mathbf{P}^{\epsilon}$, where the elements of $\mathbf{P}^{-}$ are lower or equal to

the corresponding elements of $\mathbf{P}$ and satisfies the lumpability condition:

$$\forall S_i, S_j \subset S \quad \forall s \in S_i : \sum_{s' \in S_j} p^-_{s,s'} = k_{i,j} \quad \forall i \neq j \tag{5.2}$$

where now, $p^-_{s,s'}$ is the one-step transition probability from state s to state s' in the matrix $\mathbf{P}^-$ and no element in $\mathbf{P}^\epsilon$ is greater than ϵ. A more helpful formulation is to permute the rows and columns of the transition matrix to bring it in the form $\mathbf{P} = diag(\mathbf{P}_{1,1}, \mathbf{P}_{2,2}, ..., \mathbf{P}_{N,N}) + \mathbf{E}$ where $\mathbf{E}$ is an "error" matrix, and the $\mathbf{P}_{i,i}$ represent sub-matrices that correspond to the i–th lumped state (for N lumps) – hence the derived $\mathbf{K}$ matrix would be of $N \times N$. The closeness of the matrix approximation, or technically *degree of coupling* is judged by the norm $\|\mathbf{E}\|_\infty$. The process of determining the right permutation does not take away from the complexity of the problem, but as we see next, practical considerations make this a tractable exercise for our purposes.

Figure 5.3 clarifies a 2×2 quasi-lumping procedure. At the very top of the figure, transition matrix P is the original matrix we try to lump. Here, we assume the state space partition of $S = \{1, 2, 3\} \bigcup \{4, 5, 6\}$ ($S_1 = \{1, 2, 3\}, S_2 = \{4, 5, 6\}$) is given. The state space of this partitioning with $\epsilon = 0.01$ satisfies the quasi-lumpability condition in Equation 5.2, and several pairs of (P^-, P^ϵ) can be found that satisfy it. However, the constant value k may not be the same. For this example, $k_{1,1} = 0.58, k_{1,2} = 0.41, k_{2,1} = 0.67$, and $k_{2,2} = 0.32$.

Technical Considerations

The permutations implied by the $\mathbf{P}_{i,i}$-based formulation of lumping allow for lumping together any collection of states. This is meaningless in our setting since via lumping we are trying to produce *ranges* of utilization values we can collectively consider as a single state. To this end, we add the constraint of lumping solely the neighboring states together. Additionally, we do not need to consider all possible lumping

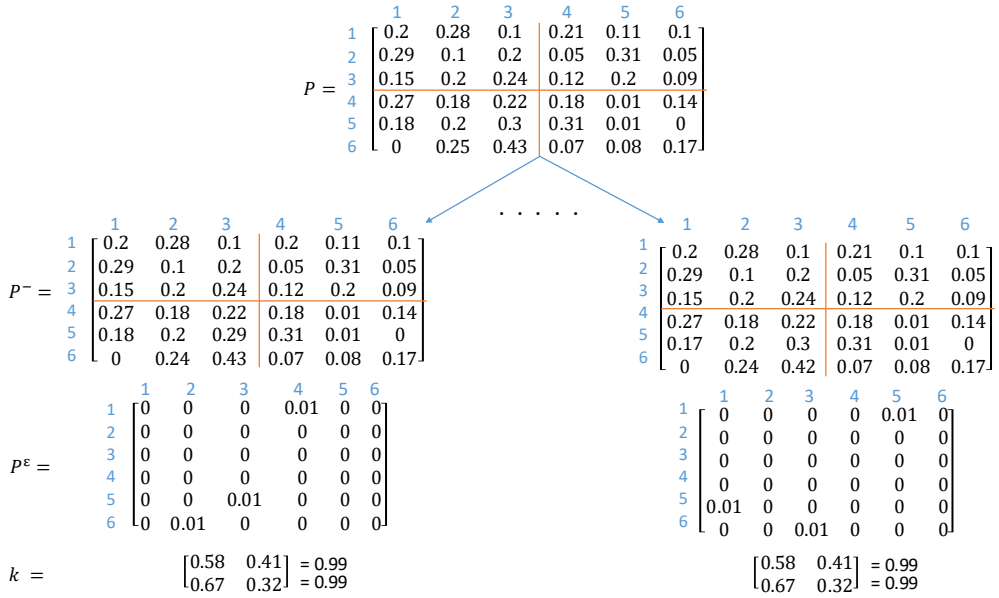


Figure 5.3: An example of quasi-lumping.

possibilities. Namely, we set the maximum number of lumped states to be equal to seven. Across all the combinations tested, we pick the one which minimizes $\|\mathbf{E}\|_\infty$. Clearly, it is up to the available computational resources to search for larger lumped models, but as the results illustrate, a seven state maximum lumped representation produced extremely good results.

A practical consideration is that not all of the 256 CU values are observed throughout the data, and in particular if we construct the matrices as suggested for each τ minutes interval, no observations for transitions to/from a significant fraction of those 256 states may be found. The number of non-observed states decreases as τ increases. As a consequence, for creating an irreducible transition matrix, the number of states in the matrix is reduced by removing the states that are not reached (zero in both rows and columns). It is the reduced matrix that is subsequently

lumped. Thus a final stage is to adjust the lumped matrix to include states that were not present in the collected data. Our strategy is to lump states not visited to their closest meta-states. By doing so, the number of lumped states is retained.

For example, if we never visited states 10 and 11 in our dataset, and the lumped states just before and just after the particular states are (7, 8, 9) – to the "left" – and (12, 13) – to the "right", the final lumping incorporates 10 into the range to the left and 11 into the range to the right. Extremes are merged to their next adjacent range. So, for example, if we never visited states 200-255 and the adjacent range lumped together ends with state 199, we merge 200-255 to the lump which includes 199.

In addition, in our setting not only the dimensions of the lumped matrix is important, but the range of utilizations coverage of each lumped state. Consequently, we have some *technical constraints* for the utilization ranges/classes to not be narrower than a certain percentage or larger than another percentage. As an example, a meta-state in the lumped matrix can be the result of lumping the first 200 neighbouring states of the original matrix with 256 states. As each state in our original matrix has $100/255 \approx 0.4\%$ of the network utilization, 200 states covers up to 80% of it. In many cases like in the idle-hours, the CU barely reaches a value higher than 100 ($\approx 40\%$ of utilization). So, without any constraints, the first class generated from our model is too broad that includes the whole channel utilization values within idle hours by providing us with no information regarding the status of the network. Hence, there is a need for constraints on the range of utilization of each meta-state. The choice of utilization ranges can be tailored to the application needs. We use the notation C_u and C_l for constraints to express how many adjacent states can be lumped at-most and at-least, respectively, in each range.

Moreover, it is the case that the more states present in the original matrix, the more lumped matrices with the same degree of coupling, $\|\mathbf{E}\|_\infty$, could be produced.

Consequently, for breaking the tie, we define another metric to decrease the number of candidate lumped matrices. While for a matrix with n states, the error term is $\|\mathbf{E}\|_\infty$ (or more simply $max\{E_1, E_2, ..., E_n\}$), our tie breaker term is $(E_1 + E_2 + ...E_n)$. That is, among the best lumped matrices which have the minimum value of $\|\mathbf{E}\|_\infty$, we choose the one with least summed error throughout the whole matrix. The intuition behind using this term is to find the lumped matrices with the lowest error across all rows produced in the lumping process. For example, a lumped matrix with the lowest $\|\mathbf{E}\|_\infty = \alpha$ may have errors in many rows equal to α; while, another lumped matrix may have only a row equal to α and with no error in rest of the rows. Although by definition both of these matrices have the same degree of coupling, the latter one is preferred.

We resorted in a recursive implementation that searches over all possible combinations of lumped states, checked against satisfying constraints for each meta-state in terms of channel utilization, number of produced lumped states, and lumping neighbouring states, as well as keeping track of quasi-lumping alternatives. This is in contrast to exact lumpability for which there exist highly efficient algorithms, such as the $O(m \log(n))$ algorithm using a splay tree [11], where m are the number of transitions and n is the number of states. Future work could include a more efficient form for implementing the lumping, albeit it is only a means to an end as far as the current thesis is concerned, and no further attention was paid to it.

5.2 The Cost of Misprediction

Misprediction of utilization can, depending on application, result on two different forms of "costs", (I) opportunities missed to utilize the available unutilized capacity, and, (II) congestion caused by attempting to utilize more than the available unutilized capacity. One can argue that (II) is more "costly" than (I), as it could impact existing users/uses of the channel. To this end, we introduce asymmetric

costs to our problem. The goal is to predict the correct class (as per the quasi-lumping explained) of utilization while minimizing the costs of misprediction. To model the cost, although mixture of different well-known function, like Gaussian, are applicable, we take an inherently asymmetric function, namely a *Rayleigh* function. The choice of asymmetric function for modeling the difference between the cost of dismissing opportunities and creating congestion is application dependent and it has no influence on the method we are proposing. The cost value from the Rayleigh function comes from mapping the correct class of utilization to the value 0 on the function and corresponding wrong classes map to lateral parts. If the corresponding utilization range causes congestion, it maps on the side of the function with the higher slope and if it causes channel under-utilization, it maps on the side with lower slope. In both scenarios, the worst case of incorrect class of utilization, meaning the difference of highest utilization percentage of the correct class with the lowest utilization percentage of the incorrect class for the over-utilizing case and vice-versa for the under-utilized case, map on the function. Figure 5.4 shows an example where utilization are grouped in for 6 ranges with utilization values of [0% - 24.313%], [24.313% - 46.274%], [46.274% - 58.039%], [58.039% - 77.254%], [77.254% - 89.019%], and [89.019% - 100%] coming from lumping of one of the busy hour interval with $\tau = 30$ minutes, $C_l = 10\%$, and $C_u = 25\%$. Assuming the utilization value falls into the third class, predicting the fourth, fifth, or sixth class of utilization causes the channel to remain under-utilized. On the other hand, predicting the first or second class of utilization leads to an over-utilized channel. As it is shown in the figure, the penalties defined for classes lead to over-utilization is way harsher than the ones for classes lead to under-utilization. Our aim is to reduce the cost of misprediction without hurting the accuracy of the model.

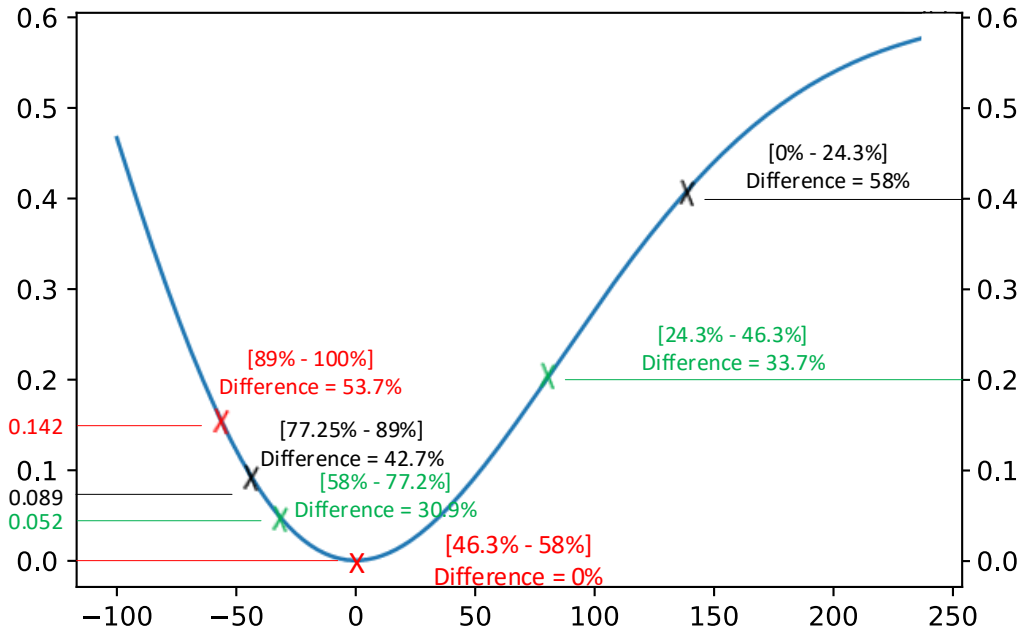


Figure 5.4: An example of class mapping on Rayleigh function, $y = 0.0$ when the right utilization range is predicted assuming the right one is in the range of $[46.3\% - 58\%]$.

Custom Loss Function Method

Our approach for reducing the cost of misprediction is to define an appropriate cost function for the learning algorithm we will use. In this work, we use cost function and loss function interchangeably. We are essentially dealing with a multi-class classification in which learning algorithms calculate probabilities of incidence for each class. For calculating the probabilities, the function predominantly used in learning algorithms is the *softmax* function whose input is a vector consisting of real numbers and normalizes it into a probability distribution. Then, the class with highest probability of incidence will be predicted as the next class of utilization. So

that if a learning algorithm with softmax function chooses a class, it has the highest probability of happening from the model perspective. As an example, giving an input vector consisting of 4 classes with values of $[c1 = 2, c2 = 4, c3 = 6, c4 = 4]$, the Softmax function converts it to a probability vector of $[c1 = 12.5\%, c2 = 25\%, c3 = 37.5\%, c4 = 25\%]$, so that the model will choose class 3 with the highest probability of incidence. In practice, the best choice of a loss function for a multi-class problem is *cross-entropy* with values coming from the softmax function. The goal of cross-entropy is to modify the probabilities coming from the softmax activation function by changing the weights of the learner. For the cross-entropy loss function with hypothetical occurrences probabilities of $\{y_1, y_2, ...,_n\}$ for n classes each of which is observed $\{k_1, k_2, ..., k_n\}$ times in training set, the likelihood of occurrence would be:

$$P[data|model] := {y_1}^{k_1} {y_2}^{k_2} ... {y_n}^{k_n} \tag{5.3}$$

By taking the logarithm of the likelihood and normalizing it, we express cross-entropy:

$$-\frac{1}{N} \log P[data|model] = -\frac{1}{N} \sum_i k_i \log y_i = -\sum_i {y_i}' \log y_i \tag{5.4}$$

The cross-entropy is a convex function; so, it guaranties existence of a global minimum. The hypothetical occurrence probabilities of $[y_1, y_2, ..., y_n]$, shown as vector Y, are coming from the softmax activation function; hence, changing them has no impact on the convexity of the cross-entropy. The learning algorithms learn the model based upon the occurrences probabilities. As a consequence, giving higher weights to incident's probability of classes with lower penalty values has no impact on the correct prediction accuracy of the learning algorithm. However, in cases of uncertainty, the model is inclined to the classes with less probability of higher punishment. The lower the values coming from the Rayleigh function, the more favorable it is for us; so by taking the complement of results from multiplication of penalty vector and

the vector of incident's probability, and normalizing it accordingly, we get higher values for the classes with lower penalty values. Assuming $[P_1, P_2, ..., P_n]$, shown as vector P, as the penalty vector. By multiplying the penalty vector and the vector of incident's probability of classes and normalizing them, we get .

$$P \times Y = [P_1 \times y_1, P_2 \times y_2, ..., P_n \times y_n] \tag{5.5}$$

Assuming $y_l = y_k$, for $1 \leq k, l \leq n$, and $P_l \leq P_k$, PY_l and PY_k would be $(P_l \times y_l) \leq (P_k \times y_k)$. Even after normalizing the PY, the ratio of P_k over P_l stays the same. Consequently, in the case of uncertainty, this model propels us to choose the class with less value of punishment.

Chapter 6

Experimental Evaluation

In previous chapters, we, first, investigated a metric allowing us to (a) express the stochastic behavior of the time series over short time intervals, and (b) to allow the comparison of the behavior across different intervals. Then, we reduced the seasonality and trends within the time-series by splitting a day into τ minutes interval. Later, we identified features by which our model can better adopt to the environment and to perform forecasting. Finally, we found out the prediction with an accuracy of 100/255th of a percentage is an overkill for most of applications; so, we adopted a Channel Utilization (CU) grouping technique, namely lumpability, to address it.

In this chapter the goal is to examine the experiment setup, environment, and limitations. Then, we define the architecture, best designed for our prediction task. Finally, we go through experiments and its results.

6.1 Data Acquisition and Experiment Setup

We collected Wi-Fi AP Beacon transmissions from a plethora of APs in a campus building at the University of Alberta and a residential environment. The majority of APs are centrally managed to provide ubiquitous Wi-Fi service. Each AP transmits approximately 10 beacons per second. All of the observed APs support

WMM and QBSS; hence, they report the channel utilization values at each beacon. For capturing the channel utilization values, a low cost means is to deploy WiFi frame sniffers at various locations so they can capture reported channel utilization values of the APs in the environment, recording the CU they report in their Beacon frames. This, passive, form of data collection is also consistent with the basic tenet of any good observation methodology, i.e., not disturbing the traffic transmitted on the wireless medium. The strategy is also meaningful because various APs operate within a building, and, in particular in residential environments, they are not under the control of a single entity/operator. Occasionally, APs may change their channels to better meet the needs of the occupants; hence, we may lose CU values of a specific channel, if no AP in range of our sniffer are operating on that channel. This is a limitation of the data collection strategy we employed, but it was not a limitation in the particular environment where we performed the data collection. That is, APs operated almost continuously on the same channel. In the occasional switchover of an AP to another channel, there were other nearby APs operating in the original channel to allow us to have uninterrupted observations of that channel's utilization. AP channel switchovers were rare.

The capture of the Beacon frames was performed using inexpensive Wi-Fi sniffers on the 2.4 and 5 GHz bands. However, in the campus environment, the 5 GHz channels are barely utilized as users are distributed among the 23 available non-overlapped channels over this band. Consequently, we solely focus on 2.4 GHz band which are highly utilized at busy hour. As we are armed with limited number of sniffers, sniffers perform channel switching on three non-overlapping channels (1, 6, and 11) to capture beacon frames of each one. Channel switching was taking place every 2 seconds; so that we have $\sigma = 6$ seconds. To this end, we have CU values of 2 seconds for every 6 second intervals. In order to have uniform CU values over time, we decided to use the maximum CU values reported in the two seconds of 6 seconds

observation intervals.

The setup managed (accounting for settling times once switching channel) to capture at least 10 beacons per AP over a six second period, regardless of channel. The loss of some beacons (for approximately four of every six seconds) was examined and found to have no significant impact. Specifically, to reduce the impact of missing beacon CU measurements, we decided to use the maximum CU reported in the two seconds of observation interval and use it as the maximum CU over the six second interval, as it was found close to the maximum had we continuously listened to the same channel. Finally, the sniffers were connected to the wired infrastructure and streamed the data collected to a cloud-based back-end, which we used to represent our edge computing platform.

We used OpenWRT operating systems (OS) [12] on our sniffers. This Linux-based OS gives us the capability of programming the router. Instead of trying to create a single, static firmware, OpenWRT provides a fully writable filesystem with package management. We programmed the sniffer to perform channel switching in monitor mode. The router switches the channel once every 2 seconds, sniffs solely the Beacon packets, and sends the traces to a cloud-based back-end for further processing and analysis.

In this work, we analysed data from each AP independently. The referenced AP might change the channel based upon the controller's order. As utilization of each channel at a specific time is different and follows different trends and seasonality, we present runs solely on data gathered from one channel of the referenced AP; channel number 1.

6.2 τ and Feature Engineering

We recap that the goal is to split the time into τ-minute time-intervals in order to diminish the traffic seasonality within a day that depends on factors such as the

number of the users, their usage patterns, etc. A large τ is problematic as it would allow the mixing of busy and idle hour behavior. A small τ is problematic because of the relative paucity of data (samples), let alone restrictive for interesting dynamics to develop. For this reason, we will look into τ intervals between 5 and 60 minutes during typical busy and idle hours periods of the day.

For each interval (representing the same time of the day across all days), a learning algorithm can be run. During the data collection process, a library of τ units long interval steady state vectors is created based on prior measurements. For the purposes of this study, we used the first 30 days as a dataset to derive the library of steady state distributions for each τ-minute interval for a *prototypical* day.

Naïve Predictor

This predictor returns prediction the same class (meta-state) as the class of the previously measured CU. The classes are as defined by the lumped model corresponding to the particular τ minute interval. Although this predictor is simple, it exhibits good performance. Our goal is to build a predictor, not much more complicated than the naïve one. To this end, we use as input the last observed class of utilization, encoded in one-hot encoding.

Neural Network Model

A model exclusively used here is a standard Neural Network with softmax activation function [5], where the generated probability value is converted to a class prediction. The classes used are the same derived by the lumped Markov chain for each corresponding τ minute period. Because of the τ minute intervals used in the definitions of utilization classes, the derivation of the lumped matrices is already "aware" of the temporal aspect of the process. We therefore endow the training data for the Neural Network depicted on Figure 6.1 with the described features. As the provided figure

and equations below show, the input features are the previous stationary values (by extracting previous values of stationary time-series), short-term dynamics distance, and one-hot encoded of the previous CU class. Within the equations below, the "stationary" and "other" features are shown by "st" and "other" as subscript in parentheses, and the layer number is provided as subscript in braces. In all experiments, the stationary features fed into the NN is the previous stationary values of CU unless otherwise is stated (for example, for $\tau = 30$ minutes these are 48 values for a day).

$$X = \begin{bmatrix} x^{(st)} \\ X^{(other)} \end{bmatrix}, X^{(other)} = \begin{bmatrix} X^{(time)} \\ X^{(CU\_class)} \end{bmatrix} \tag{6.1}$$

$$h^{(st)} = (X^{(st)})^T.W^{(st)<1>} + b^{(st)<1>} \tag{6.2}$$

$$h^{(other)} = (X^{(other)})^T.W^{(other)<1>} + b^{(other)<1>} \tag{6.3}$$

$$h_1 = \begin{bmatrix} h^{(st)} \\ h^{(other)} \end{bmatrix} \tag{6.4}$$

$$h_2 = h_1^T.W^{<2>} + b^{<2>} \tag{6.5}$$

$$\hat{Y} = \sigma(h_2^T.W^{<3>} + b^{<3>}) \tag{6.6}$$

6.2.1 Results

Our data set contains 130 days worth of data. For the sake of simplicity, and without hurting the generality and applicability of the presented techniques, we narrow our attention to 98 working days (32 days were weekends and holidays). We do the training on 80% of data each time, and testing it on 20% of data. For Neural Network,

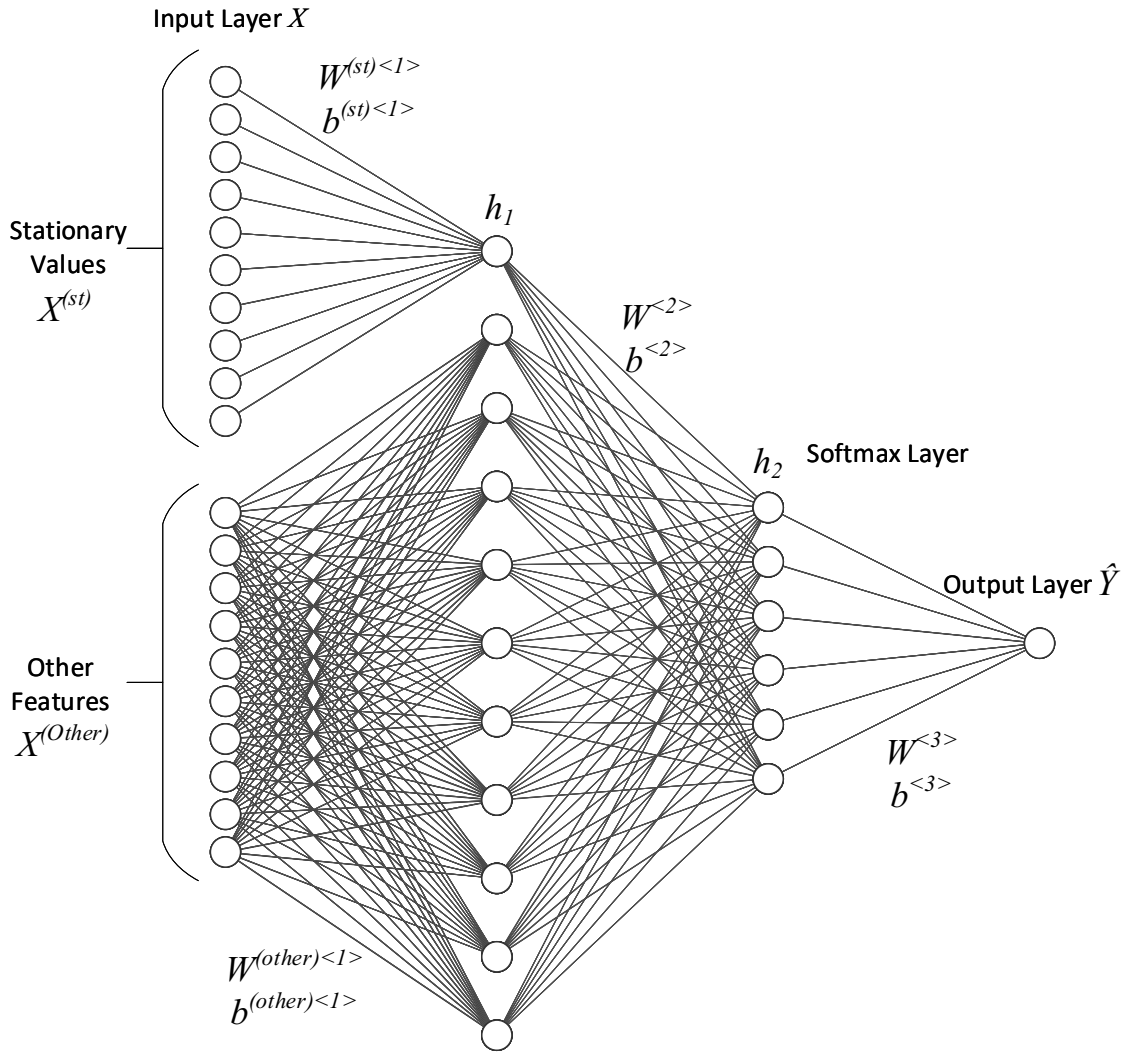


Figure 6.1: The neural network architecture used.

we run the model for 2000 epochs and we also use early stopping technique. We take advantage of ADAM (ADAptive Moment) optimizer [20] in our model with initial step size of 0.001. For updating network weights iteratively based on the training data, this optimization algorithm is being used instead of classical stochastic gradient descent (SGD). This algorithm is taking advantage of benefits coming from two

other extensions of SGD descent, namely AdaGrad (abbreviation of Adaptive Gradient Algorithm) and RMSProp (abbreviation of Root Mean Square Propagation). The former adapts the learning rate to the parameters performing smaller updates for parameters associated with frequently occurring features and larger updates for parameters associated with infrequent features, making it suitable for learning sparse data patterns. In addition, the latter maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight, making it suitable for online and non-stationary problems. Both of these algorithms use the basic idea of Momentum algorithm which tries to solve the SGD problem that highly oscillates across the slopes of the areas where surface curves are more steeply in one dimension than the others, like local optima. Momentum algorithm helps accelerate SGD in the relevant direction and dampens its oscillations. We used $\beta 1$ and $\beta 2$ hyper-parameters of ADAM algorithm, the exponential decay rate for the first and second moment estimates, to be the default values used in [20], 0.9 and 0.999 respectively. The decision of using the default values is made as changing these values only alter the pace of optimizer in falling into global minimum. Different values were tested and they had no impact in the final result of the prediction. To this end, we stick with the default values stated in paper. Moreover, an epsilon used in this algorithm for preventing division by zero while updating the weights, has the value of 10^{-8}.

Global Lumping

First, we establish a comparison baseline by assuming that the process is stationary and hence, a single transition matrix (and corresponding lumped matrix) can describe the entire process, i.e. without considering τ minutes intervals separately. By this assumption, we can assess effect of dismantling time into τ minutes intervals as it removes some trends and seasonality within data. By lumping the single transition

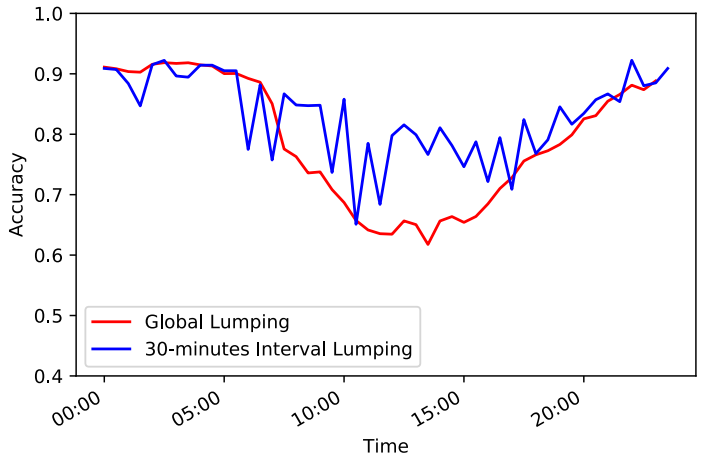


Figure 6.2: Accuracy for naïve predictor, under global lumping (red) vs. separate (blue), for each $\tau = 30$ minute interval, lumping.

matrix of the training data with lumping constraint of $C_l = 10\%$ and $C_u = 25\%$, we end up with a six-states lumped matrix corresponding to the following intervals of CU values (and utilization figures in parentheses): [0-62] (0-24.313%), [62-100] (24.313-39.215%), [100- 133] (39.215-52.156%), [133-186] (52.156-73.33%), [186-220] (73.33-86.274%), and [220-255] (86.274-100%); and producing an quasi-lumping error of $\mathbf{E}_\infty = 0.557617$ and summed error of 34.0684 throughout the lumping. Table 6.1a shows the accuracy, precision, recall, and f1-score results of this experiment for each 30 minutes within a day for the naïve predictor. Results prove a high accuracy for the idle hour intervals and as we move to busy hours, the accuracy is reduced.

To conclude that separating into independent time-intervals affects the prediction, we use lumping on each separate interval in subsequent steps. We set the number of lumped states to 6, and τ to 30 minutes. The results of the experiment using naïve predictor are shown in Table 6.1b. Again accuracy decreases when looking at the busy hours. However, we observe enhancement in prediction accuracy when τ

time	accuracy	precision	recall	f1-score
00:00-00:30	0.91	0.91	0.91	0.91
00:30-01:00	0.91	0.91	0.91	0.91
01:00-01:30	0.9	0.9	0.9	0.9
01:30-02:00	0.9	0.9	0.9	0.9
02:00-02:30	0.92	0.92	0.92	0.92
02:30-03:00	0.92	0.92	0.92	0.92
03:00-03:30	0.92	0.92	0.92	0.92
03:30-04:00	0.92	0.92	0.92	0.92
04:00-04:30	0.91	0.91	0.91	0.91
04:30-05:00	0.91	0.91	0.91	0.91
05:00-05:30	0.9	0.9	0.9	0.9
05:30-06:00	0.9	0.9	0.9	0.9
06:00-06:30	0.89	0.89	0.89	0.89
06:30-07:00	0.89	0.89	0.89	0.89
07:00-07:30	0.85	0.85	0.85	0.85
07:30-08:00	0.78	0.78	0.78	0.78
08:00-08:30	0.76	0.76	0.76	0.76
08:30-09:00	0.74	0.74	0.74	0.74
09:00-09:30	0.74	0.74	0.74	0.74
09:30-10:00	0.71	0.71	0.71	0.71
10:00-10:30	0.69	0.69	0.69	0.69
10:30-11:00	0.66	0.66	0.66	0.66
11:00-11:30	0.64	0.64	0.64	0.64
11:30-12:00	0.64	0.64	0.64	0.64
12:00-12:30	0.63	0.63	0.63	0.63
12:30-13:00	0.66	0.66	0.66	0.66
13:00-13:30	0.65	0.65	0.65	0.65
13:30-14:00	0.62	0.62	0.62	0.62
14:00-14:30	0.66	0.66	0.66	0.66
14:30-15:00	0.66	0.66	0.66	0.66
15:00-15:30	0.65	0.65	0.65	0.65
15:30-16:00	0.66	0.66	0.66	0.66
16:00-16:30	0.68	0.68	0.68	0.68
16:30-17:00	0.71	0.71	0.71	0.71
17:00-17:30	0.73	0.73	0.73	0.73
17:30-18:00	0.76	0.76	0.76	0.76
18:00-18:30	0.77	0.77	0.77	0.77
18:30-19:00	0.77	0.77	0.77	0.77
19:00-19:30	0.78	0.78	0.78	0.78
19:30-20:00	0.8	0.8	0.8	0.8
20:00-20:30	0.83	0.83	0.83	0.83
20:30-21:00	0.83	0.83	0.83	0.83
21:00-21:30	0.85	0.85	0.85	0.85
21:30-22:00	0.87	0.87	0.87	0.87
22:00-22:30	0.88	0.88	0.88	0.88
22:30-23:00	0.87	0.87	0.87	0.87
23:00-23:30	0.89	0.89	0.89	0.89
23:30-00:00	0.91	0.91	0.91	0.91

(a) Accuracy using one, global, lumping.

time	accuracy	precision	recall	f1-score
00:00-00:30	0.91	0.91	0.91	0.91
00:30-01:00	0.91	0.91	0.91	0.91
01:00-01:30	0.88	0.88	0.88	0.88
01:30-02:00	0.85	0.85	0.85	0.85
02:00-02:30	0.92	0.92	0.92	0.92
02:30-03:00	0.92	0.92	0.92	0.92
03:00-03:30	0.9	0.9	0.9	0.9
03:30-04:00	0.89	0.89	0.89	0.89
04:00-04:30	0.91	0.91	0.91	0.91
04:30-05:00	0.91	0.91	0.91	0.91
05:00-05:30	0.91	0.91	0.91	0.91
05:30-06:00	0.91	0.91	0.91	0.91
06:00-06:30	0.77	0.77	0.77	0.77
06:30-07:00	0.88	0.88	0.88	0.88
07:00-07:30	0.76	0.76	0.76	0.76
07:30-08:00	0.87	0.87	0.87	0.87
08:00-08:30	0.85	0.85	0.85	0.85
08:30-09:00	0.85	0.85	0.85	0.85
09:00-09:30	0.85	0.85	0.85	0.85
09:30-10:00	0.74	0.74	0.74	0.74
10:00-10:30	0.86	0.86	0.86	0.86
10:30-11:00	0.65	0.65	0.65	0.65
11:00-11:30	0.78	0.78	0.78	0.78
11:30-12:00	0.68	0.68	0.68	0.68
12:00-12:30	0.8	0.8	0.8	0.8
12:30-13:00	0.82	0.82	0.82	0.82
13:00-13:30	0.8	0.8	0.8	0.8
13:30-14:00	0.77	0.77	0.77	0.77
14:00-14:30	0.81	0.81	0.81	0.81
14:30-15:00	0.78	0.78	0.78	0.78
15:00-15:30	0.75	0.75	0.75	0.75
15:30-16:00	0.79	0.79	0.79	0.79
16:00-16:30	0.72	0.72	0.72	0.72
16:30-17:00	0.79	0.79	0.79	0.79
17:00-17:30	0.71	0.71	0.71	0.71
17:30-18:00	0.82	0.82	0.82	0.82
18:00-18:30	0.77	0.77	0.77	0.77
18:30-19:00	0.79	0.79	0.79	0.79
19:00-19:30	0.85	0.85	0.85	0.85
19:30-20:00	0.82	0.82	0.82	0.82
20:00-20:30	0.83	0.83	0.83	0.83
20:30-21:00	0.86	0.86	0.86	0.86
21:00-21:30	0.87	0.87	0.87	0.87
21:30-22:00	0.85	0.85	0.85	0.85
22:00-22:30	0.92	0.92	0.92	0.92
22:30-23:00	0.88	0.88	0.88	0.88
23:00-23:30	0.89	0.89	0.89	0.89
23:30-00:00	0.91	0.91	0.91	0.91

(b) Accuracy for separate, every τ, lumping.

Table 6.1: Accuracy for naïve predictor, under global lumping vs. separate, for each $\tau = 30$ minute interval, lumping.

minute intervals are separated which encourages us to continue further with separate τ minutes intervals for the remainder of this study. Figure 6.2 provides a pictorial view of the accuracy columns in Table 6.6.

Features Analysis

In this experiment, we first compare our feature engineered NN with naïve predictor to assess the capabilities of our model for $\tau = 30$ minutes. Then, we investigate the importance of our features within the model. The classes from the lumping method have the constraints of $C_l = 10\%$ and $C_u = 25\%$. For lumping states with aforementioned constraints and 6 classes we show in Table 6.2 the boundaries (as utilization percentages) between the classes. Notice that each class does not span one-sixth of of the utilization range and that, suggesting that the lumping responds to the different traffic mix behavior at different times of the day.

Tables 6.3b and 6.3a show the results of experiments for the naïve predictor against our feature engineered NN. As it is shown in these tables, the feature engineered NN greatly outperforms the naïve predictor across all intervals. The comparison of the resulting accuracy is depicted also in Figure 6.3.

The process of evaluating accuracy assumes that we are operating in real-time and we can only know the past behavior of the time series. As such, we are unaware of the current τ interval's steady-state distribution because it is still progressing, and we can only evaluate it post-factor. We can quantify the loss in prediction accuracy assuming we had access to an oracle for the current steady state or for the upcoming stationary features (stationary values of upcoming CU). Figure 6.2.1 demonstrates this gap. Compared to Figure 6.3, the the results in Figure 6.2.1 suggests that what would impact more the performance of the predictor is knowledge of the current steady state, but the more interesting observation is in seeing that, even if we had a-priori access to the steady state distribution, our prediction would not have been

time	boundaries	‖E‖∞	tie breaker error (E1+E2+...En)
00:00-00:30	[0% - 24.31% - 35.28% - 51.38% - 65.1% - 77.25% - 100.0%]	1.0	40.08
00:30-01:00	[0% - 24.31% - 38.03% - 48.25% - 59.22% - 78.06% - 100.0%]	1.0	29.23
01:00-01:30	[0% - 23.14% - 37.25% - 48.25% - 59.22% - 76.06% - 100.0%]	1.0	20.43
01:30-02:00	[0% - 10.195% - 29.02% - 52.16% - 68.25% - 79.2% - 100.0%]	0.55	20.51
02:00-02:30	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]	1.0	31.17
02:30-03:00	[0% - 24.31% - 47.06% - 57.25% - 68.25% - 83.1% - 100.0%]	1.0	30.1
03:00-03:30	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]	1.0	22.76
03:30-04:00	[0% - 23.14% - 34.12% - 45.1% - 56.1% - 77.25% - 100.0%]	0.75	18.25
04:00-04:30	[0% - 24.31% - 38.03% - 49.03% - 61.2% - 84.3% - 100.0%]	1.0	28.12
04:30-05:00	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]	1.0	23.83
05:00-05:30	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 87.06% - 100.0%]	1.0	23.83
05:30-06:00	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 87.06% - 100.0%]	1.0	45.51
06:00-06:30	[0% - 10.195% - 23.14% - 33.34% - 54.12% - 65.1% - 100.0%]	0.73	27.19
06:30-07:00	[0% - 24.31% - 36.1% - 50.2% - 60.4% - 76.06% - 100.0%]	1.0	30.33
07:00-07:30	[0% - 17.25% - 39.22% - 50.2% - 61.2% - 75.3% - 100.0%]	1.0	45.51
07:30-08:00	[0% - 11.375% - 36.1% - 48.25% - 59.22% - 80.4% - 100.0%]	1.0	39.05
08:00-08:30	[0% - 10.195% - 30.2% - 52.16% - 63.12% - 75.3% - 100.0%]	1.0	42.14
08:30-09:00	[0% - 13.336% - 37.25% - 48.25% - 60.4% - 84.3% - 100.0%]	1.0	43.87
09:00-09:30	[0% - 12.16% - 36.1% - 47.06% - 67.06% - 87.06% - 100.0%]	1.0	40.35
09:30-10:00	[0% - 19.22% - 43.12% - 53.34% - 67.06% - 79.2% - 100.0%]	1.0	45.68
10:00-10:30	[0% - 11.375% - 36.1% - 49.03% - 59.22% - 81.2% - 100.0%]	0.62	37.01
10:30-11:00	[0% - 23.14% - 39.22% - 56.1% - 67.06% - 86.25% - 100.0%]	0.56	36.01
11:00-11:30	[0% - 10.195% - 33.34% - 56.1% - 70.2% - 81.2% - 100.0%]	0.75	40.64
11:30-12:00	[0% - 10.195% - 22.36% - 45.1% - 69.0% - 79.2% - 100.0%]	0.63	40.84
12:00-12:30	[0% - 11.375% - 36.1% - 54.12% - 76.06% - 89.0% - 100.0%]	1.0	40.8
12:30-13:00	[0% - 13.336% - 38.03% - 48.25% - 62.34% - 77.25% - 100.0%]	1.0	44.8
13:00-13:30	[0% - 13.336% - 37.25% - 61.2% - 72.2% - 82.4% - 100.0%]	1.0	41.72
13:30-14:00	[0% - 12.16% - 36.1% - 57.25% - 68.25% - 80.4% - 100.0%]	1.0	39.44
14:00-14:30	[0% - 11.375% - 36.1% - 57.25% - 69.0% - 89.0% - 100.0%]	0.73	36.38
14:30-15:00	[0% - 13.336% - 38.03% - 53.34% - 71.4% - 89.0% - 100.0%]	0.89	44.92
15:00-15:30	[0% - 13.336% - 35.28% - 59.22% - 71.4% - 82.4% - 100.0%]	0.75	48.57
15:30-16:00	[0% - 11.375% - 36.1% - 60.4% - 76.06% - 87.06% - 100.0%]	1.0	42.14
16:00-16:30	[0% - 24.31% - 46.28% - 58.03% - 77.25% - 88.25% - 100.0%]	0.8	40.61
16:30-17:00	[0% - 10.195% - 33.34% - 58.03% - 74.1% - 89.0% - 100.0%]	1.0	39.62
17:00-17:30	[0% - 14.12% - 28.23% - 52.16% - 63.12% - 75.3% - 100.0%]	0.83	50.69
17:30-18:00	[0% - 11.375% - 36.1% - 48.25% - 63.12% - 82.4% - 100.0%]	1.0	43.4
18:00-18:30	[0% - 19.22% - 42.34% - 56.1% - 66.25% - 77.25% - 100.0%]	1.0	52.83
18:30-19:00	[0% - 17.25% - 41.2% - 52.16% - 62.34% - 79.2% - 100.0%]	0.67	39.61
19:00-19:30	[0% - 10.195% - 34.12% - 51.38% - 68.25% - 79.2% - 100.0%]	1.0	38.72
19:30-20:00	[0% - 13.336% - 37.25% - 52.16% - 64.3% - 88.25% - 100.0%]	0.7	43.27
20:00-20:30	[0% - 24.31% - 49.03% - 59.22% - 70.2% - 80.4% - 100.0%]	1.0	43.07
20:30-21:00	[0% - 10.195% - 32.16% - 56.1% - 67.06% - 78.06% - 100.0%]	1.0	36.09
21:00-21:30	[0% - 10.195% - 34.12% - 47.06% - 62.34% - 80.4% - 100.0%]	1.0	37.08
21:30-22:00	[0% - 10.195% - 32.16% - 50.2% - 60.4% - 75.3% - 100.0%]	1.0	38.6
22:00-22:30	[0% - 10.195% - 32.16% - 50.2% - 60.4% - 75.3% - 100.0%]	1.0	22.81
22:30-23:00	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 80.4% - 100.0%]	1.0	34.62
23:00-23:30	[0% - 24.31% - 35.28% - 47.06% - 58.03% - 82.4% - 100.0%]	1.0	34.14
23:30-00:00	[0% - 23.14% - 37.25% - 48.25% - 59.22% - 76.06% - 100.0%]	1.0	30.33

Table 6.2: Boundaries from lumping into 6 classes (lumped states) with $\tau = 30$ minutes, $C_l = 10\%$, and $C_u = 25\%$.

exact.

The weights in layer 2 of our NN ($W^{<2>}$) NN, provide us with hints of the importance of each one of the features. We investigate importance of features for two cases, one during a busy hour (Table 6.4) and one during an idle hour (Table 6.5). Having 6 classes and $\tau = 30$ minutes, the features would be: one-hot encoded value

time	accuracy	precision	recall	f1-score
00:00-00:30	0.95	0.96	0.95	0.96
00:30-01:00	0.96	0.96	0.96	0.96
01:00-01:30	0.94	0.95	0.94	0.94
01:30-02:00	0.93	0.92	0.93	0.92
02:00-02:30	0.96	0.96	0.96	0.96
02:30-03:00	0.96	0.96	0.96	0.96
03:00-03:30	0.95	0.95	0.95	0.95
03:30-04:00	0.95	0.95	0.95	0.95
04:00-04:30	0.96	0.96	0.96	0.96
04:30-05:00	0.95	0.95	0.95	0.95
05:00-05:30	0.94	0.95	0.94	0.95
05:30-06:00	0.94	0.95	0.94	0.95
06:00-06:30	0.9	0.9	0.9	0.9
06:30-07:00	0.94	0.94	0.94	0.94
07:00-07:30	0.87	0.87	0.87	0.87
07:30-08:00	0.92	0.92	0.92	0.92
08:00-08:30	0.93	0.93	0.93	0.93
08:30-09:00	0.91	0.9	0.91	0.9
09:00-09:30	0.92	0.91	0.92	0.91
09:30-10:00	0.86	0.86	0.86	0.85
10:00-10:30	0.92	0.92	0.92	0.91
10:30-11:00	0.82	0.83	0.82	0.82
11:00-11:30	0.89	0.89	0.89	0.89
11:30-12:00	0.83	0.83	0.83	0.83
12:00-12:30	0.89	0.88	0.89	0.88
12:30-13:00	0.88	0.88	0.88	0.88
13:00-13:30	0.88	0.87	0.88	0.87
13:30-14:00	0.88	0.88	0.88	0.87
14:00-14:30	0.9	0.9	0.9	0.89
14:30-15:00	0.88	0.87	0.88	0.87
15:00-15:30	0.86	0.86	0.86	0.85
15:30-16:00	0.89	0.88	0.89	0.89
16:00-16:30	0.86	0.87	0.86	0.86
16:30-17:00	0.9	0.9	0.9	0.89
17:00-17:30	0.85	0.85	0.85	0.85
17:30-18:00	0.9	0.9	0.9	0.9
18:00-18:30	0.89	0.9	0.89	0.89
18:30-19:00	0.9	0.9	0.9	0.9
19:00-19:30	0.92	0.91	0.92	0.91
19:30-20:00	0.91	0.91	0.91	0.91
20:00-20:30	0.91	0.92	0.91	0.91
20:30-21:00	0.92	0.92	0.92	0.92
21:00-21:30	0.93	0.93	0.93	0.93
21:30-22:00	0.93	0.93	0.93	0.93
22:00-22:30	0.96	0.96	0.96	0.96
22:30-23:00	0.95	0.95	0.95	0.95
23:00-23:30	0.94	0.94	0.94	0.94
23:30-00:00	0.94	0.95	0.94	0.94

(a) Accuracy for NN predictor.

time	accuracy	precision	recall	f1-score
00:00-00:30	0.91	0.91	0.91	0.91
00:30-01:00	0.91	0.91	0.91	0.91
01:00-01:30	0.88	0.88	0.88	0.88
01:30-02:00	0.85	0.85	0.85	0.85
02:00-02:30	0.92	0.92	0.92	0.92
02:30-03:00	0.92	0.92	0.92	0.92
03:00-03:30	0.9	0.9	0.9	0.9
03:30-04:00	0.89	0.89	0.89	0.89
04:00-04:30	0.91	0.91	0.91	0.91
04:30-05:00	0.89	0.89	0.89	0.89
05:00-05:30	0.91	0.91	0.91	0.91
05:30-06:00	0.79	0.79	0.79	0.79
06:00-06:30	0.77	0.77	0.77	0.77
06:30-07:00	0.88	0.88	0.88	0.88
07:00-07:30	0.76	0.76	0.76	0.76
07:30-08:00	0.87	0.87	0.87	0.87
08:00-08:30	0.85	0.85	0.85	0.85
08:30-09:00	0.85	0.85	0.85	0.85
09:00-09:30	0.85	0.85	0.85	0.85
09:30-10:00	0.74	0.74	0.74	0.74
10:00-10:30	0.86	0.86	0.86	0.86
10:30-11:00	0.65	0.65	0.65	0.65
11:00-11:30	0.78	0.78	0.78	0.78
11:30-12:00	0.68	0.68	0.68	0.68
12:00-12:30	0.8	0.8	0.8	0.8
12:30-13:00	0.82	0.82	0.82	0.82
13:00-13:30	0.8	0.8	0.8	0.8
13:30-14:00	0.77	0.77	0.77	0.77
14:00-14:30	0.81	0.81	0.81	0.81
14:30-15:00	0.78	0.78	0.78	0.78
15:00-15:30	0.75	0.75	0.75	0.75
15:30-16:00	0.79	0.79	0.79	0.79
16:00-16:30	0.72	0.72	0.72	0.72
16:30-17:00	0.79	0.79	0.79	0.79
17:00-17:30	0.71	0.71	0.71	0.71
17:30-18:00	0.82	0.82	0.82	0.82
18:00-18:30	0.77	0.77	0.77	0.77
18:30-19:00	0.79	0.79	0.79	0.79
19:00-19:30	0.85	0.85	0.85	0.85
19:30-20:00	0.82	0.82	0.82	0.82
20:00-20:30	0.83	0.83	0.83	0.83
20:30-21:00	0.86	0.86	0.86	0.86
21:00-21:30	0.87	0.87	0.87	0.87
21:30-22:00	0.85	0.85	0.85	0.85
22:00-22:30	0.92	0.92	0.92	0.92
22:30-23:00	0.88	0.88	0.88	0.88
23:00-23:30	0.89	0.89	0.89	0.89
23:30-00:00	0.88	0.88	0.88	0.88

(b) Accuracy for naïve predictor.

Table 6.3: Accuracy for feature-engineered NN (red) vs. naïve (blue) predictor (6 classes, $\tau = 30$ minutes, $C_l = 10\%$, $C_u = 25\%$).

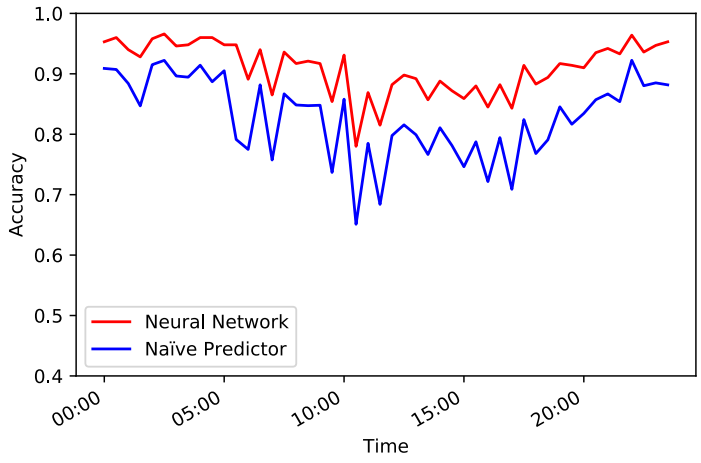


Figure 6.3: Accuracy for feature-engineered NN (red) vs. naïve (blue) predictor (6 classes, $\tau = 30$ minutes, $C_l = 10\%$, $C_u = 25\%$).

of previous CU class ($X^{CU_c lass}$), 48 timing features (X^{time}), and 1 stationary value (h^{st}) which is the outcome of a NN with no hidden units (or Logistic Regression) with 48 stationary values as input (X^{st}).

From Tables 6.4 and 6.5 we can analyze the importance of features for each class of utilization. Among the 6 one-hot encoded features of previously seen utilization class, each class not only gives high priority to the one feature related to that class, but also the neighbouring classes. For example, as for busy-hours, class 1 gives the highest weight to its class feature, $W_{1,1}^{(CU_c lass)<2>}$, and then a relatively high weight to class 2, $W_{1,2}^{(CU_c lass)<2>}$. Class 2 also gives the approximately same weights to class 2, $W_{2,2}^{(CU_c lass)<2>}$, and classes 1 $W_{2,1}^{(CU_c lass)<2>}$. It also gives a high weight to class 3, $W_{2,3}^{(CU_c lass)<2>}$. The same behaviour can be seen for the rest of classes and for the idle hours as well. However, by contrast the idle hours is less probable to have high CU values; the classes related to high utilization values have less feature importance, and generally the model predicts a behavior "stuck" at class 1 (the lowest utilization

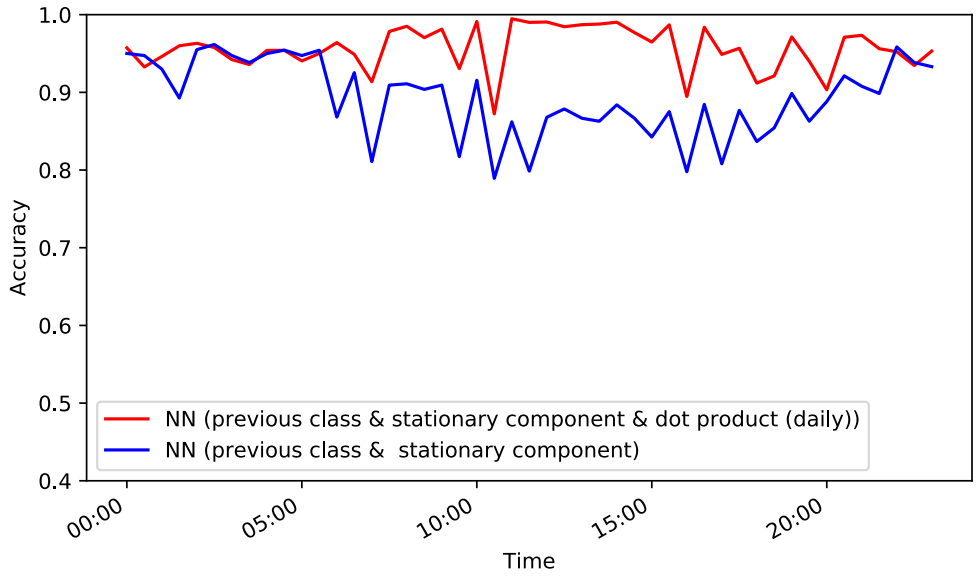


Figure 6.4: Accuracy for feature-engineered NN assuming access to the current interval's steady state (red), or, the upcoming stationary features (blue) are accessible to the predictor (6 classes, $\tau = 30$ minutes, $C_l = 10\%$, $C_u = 25\%$).

range).

The importance of each of the library of 48 steady state intervals, $W^{(timing)<2>}$, is different for each class and as their influence is spread across all classes, their weights are smaller. The strongest influence feature is the stationary one. As less knowledge is available for the predictor to learn from the previous features, h^{st} becomes more helpful. Consequently, the learning algorithm uses this feature as an extra knowledge to learn about the less observed classes. One example is for the case of $W_1^{(st)<2>}$ in which its weight pushes the predictor toward choosing other classes.

As in the tables 6.4 and 6.5 we can analyze importance of features for each class of utilization. Among the 6 one-hot encoded features of previously seen class of utilization, each class not only gives high priority to the one feature related to that

Feature Name	weight	[0%-19.22%]	[19.22%-43.14%]	[43.14%-53.33%]	[53.33%-67.06%]	[67.06%-79.22%]	[79.22%-100%]
Previous CU Class $W_{i,c}^{(CU\_class)<2>}$	$W_1^{(CU\_class)<2>}$	8.77	2.71	-2.25	-3.55	-3.6	-2.08
	$W_2^{(CU\_class)<2>}$	4.56	1.74	-0.9	-1.94	-1.91	-1.55
	$W_3^{(CU\_class)<2>}$	-1.37	-0.09	1.18	0.69	0.59	-0.99
	$W_4^{(CU\_class)<2>}$	-4.51	-1.2	1.22	2.14	1.79	0.56
	$W_5^{(CU\_class)<2>}$	-5.26	-1.26	1.19	2.3	1.9	1.12
	$W_6^{(CU\_class)<2>}$	-2.26	-1.89	-0.42	0.37	1.27	2.93
Dot Product For Time	$W_1^{(timing)<2>}$	0.26	0.09	-0.06	-0.16	-0.17	0.05
	$W_2^{(timing)<2>}$	0.15	-0.05	-0.05	-0.04	-0.04	0.03
	$W_3^{(timing)<2>}$	0.06	0.01	-0.01	0.0	-0.09	0.03
	$W_4^{(timing)<2>}$	0.53	-0.47	0.14	-0.14	-0.18	0.11
	$W_5^{(timing)<2>}$	0.23	-0.25	0.01	-0.12	0.09	0.04
	$W_6^{(timing)<2>}$	0.65	-0.68	0.01	-0.13	0.02	0.12
	$W_7^{(timing)<2>}$	-0.04	-0.08	0.06	-0.1	0.07	0.09
	$W_8^{(timing)<2>}$	0.37	-0.14	-0.08	-0.16	-0.06	0.07
	$W_9^{(timing)<2>}$	0.25	-0.15	-0.05	-0.01	-0.11	0.07
	$W_{10}^{(timing)<2>}$	0.57	-0.36	-0.14	-0.09	-0.05	0.07
	$W_{11}^{(timing)<2>}$	0.67	-0.5	-0.05	-0.1	-0.07	0.05
	$W_{12}^{(timing)<2>}$	0.41	-0.07	0.18	-0.2	-0.4	0.08
	$W_{13}^{(timing)<2>}$	-0.12	0.28	0.01	-0.18	-0.02	0.04
	$W_{14}^{(timing)<2>}$	-0.39	0.37	0.07	-0.12	0.16	-0.09
	$W_{15}^{(timing)<2>}$	-0.57	0.32	-0.22	0.16	0.46	-0.15
	$W_{16}^{(timing)<2>}$	-0.2	-0.02	-0.2	0.05	0.56	-0.18
	$W_{17}^{(timing)<2>}$	-0.76	0.49	-0.22	0.28	0.44	-0.23
	$W_{18}^{(timing)<2>}$	-0.45	0.03	-0.26	0.25	0.71	-0.29
	$W_{19}^{(timing)<2>}$	0.01	-0.11	-0.41	0.13	0.63	-0.25
	$W_{20}^{(timing)<2>}$	-0.11	0.1	-0.44	0.17	0.59	-0.32
	$W_{21}^{(timing)<2>}$	0.22	-0.27	-0.41	0.11	0.63	-0.28
	$W_{22}^{(timing)<2>}$	-0.88	0.4	0.24	0.14	0.19	-0.09
	$W_{23}^{(timing)<2>}$	-0.3	0.27	0.12	-0.09	0.22	-0.22
	$W_{24}^{(timing)<2>}$	-0.34	0.34	0.3	-0.12	-0.1	-0.08
	$W_{25}^{(timing)<2>}$	-0.53	0.33	0.69	-0.15	-0.34	0.01
	$W_{26}^{(timing)<2>}$	-0.69	0.46	0.44	0.09	-0.18	-0.1
	$W_{27}^{(timing)<2>}$	-0.11	0.23	0.22	-0.02	-0.27	-0.05
	$W_{28}^{(timing)<2>}$	-1.01	0.15	0.88	0.05	-0.03	-0.04
	$W_{29}^{(timing)<2>}$	0.14	0.06	0.06	0.06	-0.29	-0.04
	$W_{30}^{(timing)<2>}$	-0.22	0.13	0.36	0.06	-0.28	-0.06
	$W_{31}^{(timing)<2>}$	-0.66	0.28	0.4	0.24	-0.25	-0.01
	$W_{32}^{(timing)<2>}$	-0.41	0.08	0.26	0.28	-0.11	-0.11
	$W_{33}^{(timing)<2>}$	0.42	0.26	-0.1	0.06	-0.67	0.03
	$W_{34}^{(timing)<2>}$	0.29	0.02	0.21	0.13	-0.74	0.09
	$W_{35}^{(timing)<2>}$	0.6	-0.12	0.54	-0.07	-1.14	0.19
	$W_{36}^{(timing)<2>}$	0.96	-0.17	0.0	0.14	-0.94	0.01
	$W_{37}^{(timing)<2>}$	1.09	0.09	-0.13	0.03	-1.16	0.07
	$W_{38}^{(timing)<2>}$	1.32	-0.56	-0.34	0.18	-0.62	0.02
	$W_{39}^{(timing)<2>}$	0.99	-0.13	-0.33	0.07	-0.55	-0.04
	$W_{40}^{(timing)<2>}$	0.15	0.47	-0.26	0.13	-0.43	-0.07
	$W_{41}^{(timing)<2>}$	-0.07	0.4	-0.15	-0.0	-0.1	-0.08
	$W_{42}^{(timing)<2>}$	-0.2	0.15	-0.16	0.08	0.29	-0.17
	$W_{43}^{(timing)<2>}$	-0.34	0.2	0.01	0.14	0.06	-0.08
	$W_{44}^{(timing)<2>}$	0.06	0.04	0.03	-0.07	-0.07	0.0
	$W_{45}^{(timing)<2>}$	0.65	-0.58	-0.08	-0.08	0.06	0.03
	$W_{46}^{(timing)<2>}$	0.33	0.16	-0.17	-0.07	-0.36	0.12
	$W_{47}^{(timing)<2>}$	-0.13	0.55	-0.09	-0.07	-0.34	0.08
	$W_{48}^{(timing)<2>}$	0.16	0.16	-0.13	-0.07	-0.17	0.04
Stationary Value $W_{i,c}^{(st)<2>}$	$W_1^{(st)<2>}$	-22.19	-4.93	6.39	8.38	8.63	3.73

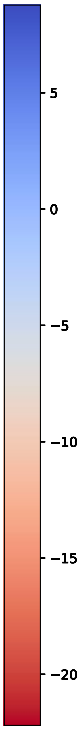


Table 6.4: The weights of layer 2 in the NN for a typical busy interval (10am-10:30am) capturing relative importance of the features.

class, but also the neighbouring classes. For example, as for busy-hours, class 1 gives the highest priority to the weight of feature for that class, $W_{1,1}^{(CU_c class)<2>}$, and then a relatively high priority to $W_{1,2}^{(CU_c class)<2>}$ which is for class 2. Class 2 also gives the highest priority to $W_{2,2}^{(CU_c class)<2>}$ and then $W_{2,1}^{(CU_c class)<2>}$ and $W_{2,3}^{(CU_c class)<2>}$. Similar behaviour can be seen for the idle-hours. However, as for the idle-hours is less probable to have high CU values, the classes related to high utilization values

Feature Name	weight	[0%-24.31%]	[24.31%-38.04%]	[38.04%-48.24%]	[48.24%-59.22%]	[59.22%-78.04%]	[78.04%-100%]
Previous CU Class $W_{i,c}^{(CU\_class)<2>}$	$W_1^{(CU\_class)<2>}$	5.46	0.55	-1.77	-1.51	-2.19	-0.54
	$W_2^{(CU\_class)<2>}$	1.52	0.85	0.2	-1.12	-1.24	-0.2
	$W_3^{(CU\_class)<2>}$	-1.61	0.51	1.53	0.25	-0.62	-0.07
	$W_4^{(CU\_class)<2>}$	-2.41	-0.47	0.55	1.53	0.85	-0.04
	$W_5^{(CU\_class)<2>}$	-2.17	-1.34	-0.52	0.89	2.3	0.83
	$W_6^{(CU\_class)<2>}$	-0.75	-0.08	-0.01	-0.02	0.85	-0.0
Dot Product For Time	$W_1^{(timing)<2>}$	0.09	-0.1	-0.12	0.04	0.08	0.01
	$W_2^{(timing)<2>}$	-0.02	-0.21	-0.0	0.07	0.15	0.02
	$W_3^{(timing)<2>}$	-0.05	-0.05	-0.14	0.02	0.19	0.03
	$W_4^{(timing)<2>}$	0.28	0.3	-0.53	-0.11	0.05	0.0
	$W_5^{(timing)<2>}$	0.14	-0.38	0.1	0.06	0.07	0.0
	$W_6^{(timing)<2>}$	0.51	-0.48	-0.05	0.01	0.02	-0.01
	$W_7^{(timing)<2>}$	0.35	-0.36	-0.05	0.03	0.03	-0.0
	$W_8^{(timing)<2>}$	0.36	-0.68	0.18	0.07	0.07	0.0
	$W_9^{(timing)<2>}$	0.3	-0.88	0.34	0.09	0.13	0.01
	$W_{10}^{(timing)<2>}$	0.2	-0.32	-0.01	0.06	0.06	0.0
	$W_{11}^{(timing)<2>}$	-0.02	-0.04	-0.05	0.04	0.07	0.0
	$W_{12}^{(timing)<2>}$	-0.19	0.26	-0.2	0.04	0.08	0.01
	$W_{13}^{(timing)<2>}$	-0.1	0.31	-0.27	0.04	0.02	-0.0
	$W_{14}^{(timing)<2>}$	-0.63	0.26	0.14	0.16	0.08	0.0
	$W_{15}^{(timing)<2>}$	-0.56	0.17	0.17	0.16	0.06	-0.01
	$W_{16}^{(timing)<2>}$	0.15	0.61	-0.51	-0.0	-0.18	-0.07
	$W_{17}^{(timing)<2>}$	0.2	0.1	-0.3	-0.0	0.04	-0.03
	$W_{18}^{(timing)<2>}$	0.41	0.28	-0.47	-0.04	-0.11	-0.07
	$W_{19}^{(timing)<2>}$	0.24	0.39	-0.38	-0.03	-0.14	-0.08
	$W_{20}^{(timing)<2>}$	0.23	0.53	-0.33	-0.04	-0.28	-0.11
	$W_{21}^{(timing)<2>}$	0.2	0.73	-0.45	-0.09	-0.27	-0.12
	$W_{22}^{(timing)<2>}$	-0.4	0.4	0.44	0.08	-0.38	-0.14
	$W_{23}^{(timing)<2>}$	-0.27	1.12	-0.27	-0.05	-0.39	-0.14
	$W_{24}^{(timing)<2>}$	-0.12	1.01	-0.21	-0.08	-0.45	-0.15
	$W_{25}^{(timing)<2>}$	-0.68	0.55	0.6	0.07	-0.41	-0.13
	$W_{26}^{(timing)<2>}$	-0.48	0.21	0.57	0.11	-0.29	-0.12
	$W_{27}^{(timing)<2>}$	-0.46	0.12	0.76	0.11	-0.39	-0.14
	$W_{28}^{(timing)<2>}$	-0.56	-0.1	1.05	0.16	-0.41	-0.14
	$W_{29}^{(timing)<2>}$	-0.2	-0.45	0.83	0.1	-0.18	-0.1
	$W_{30}^{(timing)<2>}$	-0.46	0.08	0.67	0.09	-0.27	-0.11
	$W_{31}^{(timing)<2>}$	-0.61	-0.59	1.19	0.17	-0.08	-0.07
	$W_{32}^{(timing)<2>}$	-0.49	-0.31	0.87	0.1	-0.09	-0.08
	$W_{33}^{(timing)<2>}$	-0.08	-0.54	0.68	0.03	-0.03	-0.06
	$W_{34}^{(timing)<2>}$	-0.14	-0.55	0.65	0.07	0.01	-0.04
	$W_{35}^{(timing)<2>}$	-0.14	-0.22	0.47	0.02	-0.07	-0.05
	$W_{36}^{(timing)<2>}$	-0.02	-0.26	0.16	-0.05	0.18	-0.0
	$W_{37}^{(timing)<2>}$	-0.28	-0.04	0.01	-0.04	0.32	0.03
	$W_{38}^{(timing)<2>}$	0.54	-0.14	-0.46	-0.14	0.19	0.0
	$W_{39}^{(timing)<2>}$	0.4	0.09	-0.47	-0.12	0.12	-0.01
	$W_{40}^{(timing)<2>}$	0.06	0.33	-0.44	-0.07	0.13	-0.01
	$W_{41}^{(timing)<2>}$	0.0	0.1	-0.26	0.03	0.13	-0.0
	$W_{42}^{(timing)<2>}$	-0.0	0.37	-0.32	0.02	-0.03	-0.04
	$W_{43}^{(timing)<2>}$	0.07	0.6	-0.64	-0.05	0.04	-0.02
	$W_{44}^{(timing)<2>}$	0.33	0.52	-0.69	-0.05	-0.08	-0.03
	$W_{45}^{(timing)<2>}$	0.25	0.38	-0.51	-0.01	-0.07	-0.03
	$W_{46}^{(timing)<2>}$	0.28	-0.46	0.02	0.08	0.08	0.0
	$W_{47}^{(timing)<2>}$	0.08	-0.26	-0.0	0.1	0.07	0.01
	$W_{48}^{(timing)<2>}$	-0.06	-0.31	0.09	0.13	0.13	0.02
Stationary Value $W_{i,c}^{(st)<2>}$	$W_j^{(st)<2>}$	-13.1	2.46	7.56	2.85	0.28	-0.04

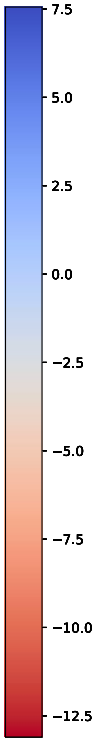


Table 6.5: The weights of layer 2 in the NN for a typical idle interval (12:30am-1:00am) capturing relative importance of the features.

has less feature importance. For example, for the idle-hours, class 1 has relative importance value of 6.2 from the $W_{2,1}^{(CU_c class)<2>}$ and class 5 has it as 0.85 from the $W_{5,5}^{(CU_c class)<2>}$. However, for the busy-hours, class 1 has relative importance value of 5.6 from the $W_{1,1}^{(CU_c class)<2>}$ and class 5 has it as 4.1 from the $W_{5,5}^{(CU_c class)<2>}$.

We conclude this part by presenting in Figure 6.5 the way the components of the NN build up a gradually a model that surpasses the naïve one. The first step to

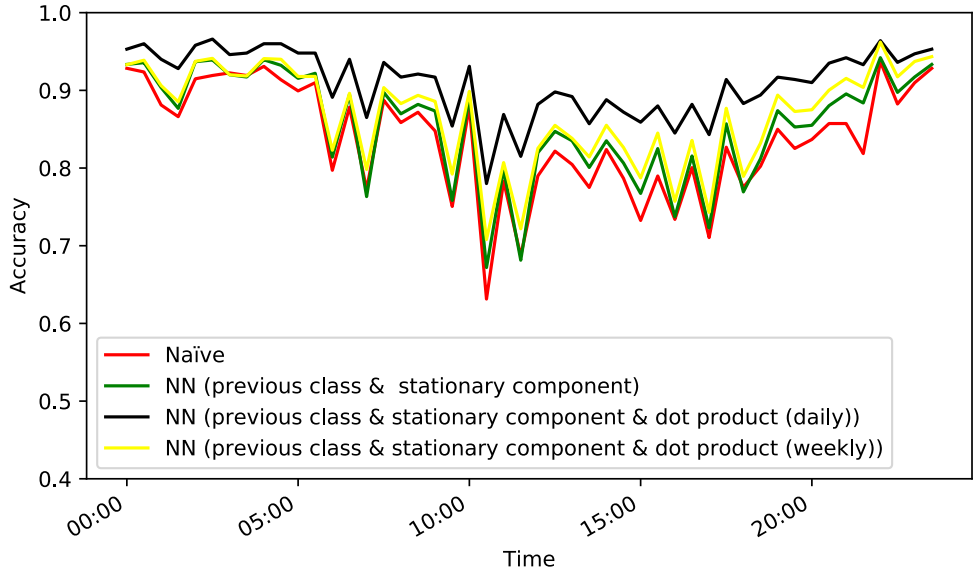


Figure 6.5: Building up the NN's accuracy vs. a naïve predictor baseline.

gain an advantage over naïve prediction is to encode the previous class and stationary value (c.f. Previous Class and Stationary Value in Tables 6.4 and 6.5), hence bridging the prediction to the recent past dynamics. This refinement is shown by the green line in Figure 6.5. Yet the impact is relatively minor on accuracy compared to what happens when we add as feature the dot product (c.f. Dot Product in Tables 6.4 and 6.5) to the intra-day τ interval steady states (of which there are 48 in the current example since $\tau = 30$ minutes). Furthermore, it is the case that the comparison to the other intra-day steady states is an indirect way to encode the time of the day. If we were to, instead, replace the intra-day with the dot product against the intra-week library of steady states, the improvement would be only marginally better (yellow line in Figure 6.5).

Selection of τ

We previously claimed that by splitting time into τ minutes intervals we can remove some seasonality from the data, which leads to better performance of the learning algorithm. In this experiment, we use $\tau = 60$, 30, 20, and 10 minutes; and we establish a baseline with $\tau = 5$ which is the smallest realistic interval due to our experimental setup. Our baseline is to use the MI algorithm described in section 3.3.1 for merging intervals having similarity (dot product) higher than 0.98% or having at least 10 merged intervals. We run the MI algorithm on the whole data set. The fruit of merging $\tau = 5$ minutes intervals is 10 classes mixing similarly behaving $\tau = 5$ minutes intervals. To test the effect of removing seasonality by merging intervals, we lump states within each of the 10 clusters with the constraint of $C_l = 10\%$ and $C_u = 25\%$. The MI provides us with close to the best overall prediction accuracy. To this end, we compare the MI method against $\tau = 30$ minutes intervals, chosen in an ad-hoc manner, as well as with the global lumping case. For this experiment we use feature engineered NN prediction results for 6 lumped states (utilization ranges). Tables 6.6a, 6.3a, and 6.6b show, respectively, the results for the case of global lumping with overall prediction accuracy of 85.92%, $\tau = 30$ minutes interval and overall prediction accuracy of 91.16%, and MI with $\tau = 5$-minutes and overall prediction accuracy of 91.46%. While the difference with respect to global lumping is significant, the difference in accuracy of the model for when $\tau = 30$ minutes and MI $\tau = 5$ minute intervals is around 0.3% which is negligible. Thus $\tau = 30$ minutes has similar effect on reducing seasonality as MI with $\tau = 5$ minutes. One possible reason for the increased accuracy is that number of samples within data seen by the MI method is higher than the one with $\tau = 30$ minutes; so that the learning algorithm might do a better job in determining patterns. More importantly, the MI method is omniscient, i.e. it is base on knowledge of the entire data set, but the rest of the techniques only know their training data. For the sake of completeness,

we also considered the case of lumping into 5 states (utilization ranges). Then the difference becomes even less. The feature engineered NN for MI with $\tau = 5$-minutes gives overall accuracy of 91.96%; and the generic $\tau = 60$-minutes has the accuracy of 89.81%, $\tau = 30$-minutes has an accuracy of 91.06%, $\tau = 20$-minutes has 91.94%, and $\tau = 10$-minutes has 91.81%. As the values suggest, the difference in overall accuracy is small for $\tau \leq 30$ minutes. The results are shown in Figure 6.6. Not that in Figure 6.6, we represent the 144 intervals for $\tau = 10$ across a 24 hour day; we repeated each accuracy value in the plot for $\tau = 20$ 2 times, for $\tau = 30$ 3 times, and for $\tau = 60$ 6 times. The results suggest that the choice of τ has not much impact in overall performance of the model as far as it is small enough (30 minutes or less) to remove the intra-day seasonality.

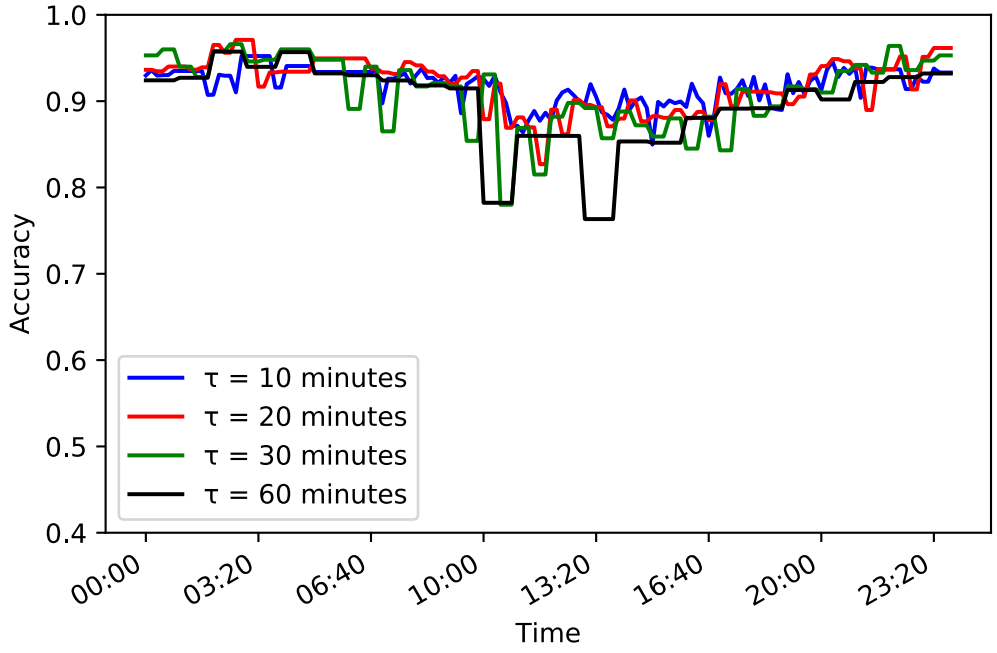


Figure 6.6: Accuracy for $\tau = 60$, $\tau = 30$, $\tau = 20$, and $\tau = 10$ minutes.

time	accuracy	precision	recall	f1-score	boundaries
00:00-23:59	0.86	0.86	0.86	0.86	[0% - 24.31% - 39.22% - 52.16% - 73.3% - 86.25% - 100.0%]

(a) Global lumping

grouped interval #	accuracy	precision	recall	f1-score	boundaries
1.0	0.95	0.93	0.95	0.94	[0% - 24.31% - 39.22% - 56.1% - 77.25% - 88.25% - 100.0%]
2.0	0.95	0.95	0.95	0.95	[0% - 10.195% - 34.12% - 54.12% - 64.3% - 75.3% - 100.0%]
3.0	0.95	0.93	0.94	0.93	[0% - 10.195% - 33.34% - 52.16% - 72.2% - 82.4% - 100.0%]
4.0	0.93	0.92	0.93	0.92	[0% - 10.195% - 28.23% - 45.1% - 67.06% - 78.06% - 100.0%]
5.0	0.93	0.91	0.93	0.91	[0% - 11.375% - 36.1% - 51.38% - 67.06% - 81.2% - 100.0%]
6.0	0.96	0.94	0.95	0.95	[0% - 12.16% - 36.1% - 46.28% - 63.12% - 78.06% - 100.0%]
7.0	0.86	0.84	0.86	0.84	[0% - 13.336% - 38.03% - 52.16% - 63.12% - 75.3% - 100.0%]
8.0	0.83	0.83	0.83	0.83	[0% - 12.16% - 29.02% - 53.34% - 78.06% - 88.25% - 100.0%]
9.0	0.91	0.9	0.91	0.9	[0% - 11.375% - 36.1% - 60.4% - 78.06% - 89.0% - 100.0%]
10.0	0.89	0.88	0.89	0.88	[0% - 13.336% - 38.03% - 54.12% - 64.3% - 82.4% - 100.0%]

(b) Merged Intervals (MI)

time	accuracy	precision	recall	f1-score	boundaries
00:00-00:30	0.95	0.96	0.95	0.96	[0% - 24.31% - 35.28% - 51.38% - 65.1% - 77.25% - 100.0%]
00:30-01:00	0.96	0.96	0.96	0.96	[0% - 24.31% - 38.03% - 48.25% - 59.22% - 78.06% - 100.0%]
01:00-01:30	0.94	0.95	0.94	0.94	[0% - 23.14% - 37.25% - 48.25% - 59.22% - 76.06% - 100.0%]
01:30-02:00	0.93	0.92	0.93	0.92	[0% - 10.195% - 29.02% - 52.16% - 68.25% - 79.2% - 100.0%]
02:00-02:30	0.96	0.96	0.96	0.96	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]
02:30-03:00	0.96	0.96	0.96	0.96	[0% - 24.31% - 47.06% - 57.25% - 68.25% - 83.1% - 100.0%]
03:00-03:30	0.95	0.95	0.95	0.95	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]
03:30-04:00	0.95	0.95	0.95	0.95	[0% - 23.14% - 34.12% - 45.1% - 56.1% - 77.25% - 100.0%]
04:00-04:30	0.96	0.96	0.96	0.96	[0% - 24.31% - 38.03% - 49.03% - 61.2% - 84.3% - 100.0%]
04:30-05:00	0.95	0.95	0.95	0.95	[0% - 24.31% - 38.03% - 48.25% - 63.12% - 82.4% - 100.0%]
05:00-05:30	0.94	0.95	0.94	0.95	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 87.06% - 100.0%]
05:30-06:00	0.94	0.95	0.94	0.95	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 87.06% - 100.0%]
06:00-06:30	0.9	0.9	0.9	0.9	[0% - 10.195% - 23.14% - 33.34% - 54.12% - 65.1% - 100.0%]
06:30-07:00	0.94	0.94	0.94	0.94	[0% - 24.31% - 36.1% - 50.2% - 60.4% - 76.06% - 100.0%]
07:00-07:30	0.87	0.87	0.87	0.87	[0% - 17.25% - 39.22% - 50.2% - 61.2% - 75.3% - 100.0%]
07:30-08:00	0.92	0.92	0.92	0.92	[0% - 11.375% - 36.1% - 48.25% - 59.22% - 80.4% - 100.0%]
08:00-08:30	0.93	0.93	0.93	0.93	[0% - 10.195% - 30.2% - 52.16% - 63.12% - 75.3% - 100.0%]
08:30-09:00	0.91	0.9	0.91	0.9	[0% - 13.336% - 37.25% - 48.25% - 60.4% - 84.3% - 100.0%]
09:00-09:30	0.92	0.91	0.92	0.91	[0% - 12.16% - 36.1% - 47.06% - 67.06% - 87.06% - 100.0%]
09:30-10:00	0.86	0.86	0.86	0.85	[0% - 19.22% - 43.12% - 53.34% - 67.06% - 79.2% - 100.0%]
10:00-10:30	0.92	0.92	0.92	0.91	[0% - 11.375% - 36.1% - 49.03% - 59.22% - 81.2% - 100.0%]
10:30-11:00	0.82	0.83	0.82	0.82	[0% - 23.14% - 39.22% - 56.1% - 67.06% - 86.25% - 100.0%]
11:00-11:30	0.89	0.89	0.89	0.89	[0% - 10.195% - 33.34% - 56.1% - 70.2% - 81.2% - 100.0%]
11:30-12:00	0.83	0.83	0.83	0.83	[0% - 10.195% - 22.36% - 45.1% - 69.0% - 79.2% - 100.0%]
12:00-12:30	0.89	0.88	0.89	0.88	[0% - 11.375% - 36.1% - 54.12% - 76.06% - 89.0% - 100.0%]
12:30-13:00	0.88	0.88	0.88	0.88	[0% - 13.336% - 38.03% - 48.25% - 62.34% - 77.25% - 100.0%]
13:00-13:30	0.88	0.87	0.88	0.87	[0% - 13.336% - 37.25% - 61.2% - 72.2% - 82.4% - 100.0%]
13:30-14:00	0.88	0.88	0.88	0.87	[0% - 12.16% - 36.1% - 57.25% - 68.25% - 80.4% - 100.0%]
14:00-14:30	0.9	0.9	0.9	0.89	[0% - 11.375% - 36.1% - 57.25% - 69.0% - 89.0% - 100.0%]
14:30-15:00	0.88	0.87	0.88	0.87	[0% - 13.336% - 38.03% - 53.34% - 71.4% - 89.0% - 100.0%]
15:00-15:30	0.86	0.86	0.86	0.85	[0% - 13.336% - 35.28% - 59.22% - 71.4% - 82.4% - 100.0%]
15:30-16:00	0.89	0.88	0.89	0.89	[0% - 11.375% - 36.1% - 60.4% - 76.06% - 87.06% - 100.0%]
16:00-16:30	0.86	0.87	0.86	0.86	[0% - 24.31% - 46.28% - 58.03% - 77.25% - 88.25% - 100.0%]
16:30-17:00	0.9	0.9	0.9	0.89	[0% - 10.195% - 33.34% - 58.03% - 74.1% - 89.0% - 100.0%]
17:00-17:30	0.85	0.85	0.85	0.85	[0% - 14.12% - 28.23% - 52.16% - 63.12% - 75.3% - 100.0%]
17:30-18:00	0.9	0.9	0.9	0.9	[0% - 11.375% - 36.1% - 48.25% - 63.12% - 82.4% - 100.0%]
18:00-18:30	0.89	0.9	0.89	0.89	[0% - 19.22% - 42.34% - 56.1% - 66.25% - 77.25% - 100.0%]
18:30-19:00	0.9	0.9	0.9	0.9	[0% - 17.25% - 41.2% - 52.16% - 62.34% - 79.2% - 100.0%]
19:00-19:30	0.92	0.91	0.92	0.91	[0% - 10.195% - 34.12% - 51.38% - 68.25% - 79.2% - 100.0%]
19:30-20:00	0.91	0.91	0.91	0.91	[0% - 13.336% - 37.25% - 52.16% - 64.3% - 88.25% - 100.0%]
20:00-20:30	0.91	0.92	0.91	0.91	[0% - 24.31% - 49.03% - 59.22% - 70.2% - 80.4% - 100.0%]
20:30-21:00	0.92	0.92	0.92	0.92	[0% - 10.195% - 32.16% - 56.1% - 67.06% - 78.06% - 100.0%]
21:00-21:30	0.93	0.93	0.93	0.93	[0% - 10.195% - 34.12% - 47.06% - 62.34% - 80.4% - 100.0%]
21:30-22:00	0.93	0.93	0.93	0.93	[0% - 10.195% - 32.16% - 50.2% - 60.4% - 75.3% - 100.0%]
22:00-22:30	0.96	0.96	0.96	0.96	[0% - 10.195% - 32.16% - 50.2% - 60.4% - 75.3% - 100.0%]
22:30-23:00	0.95	0.95	0.95	0.95	[0% - 24.31% - 47.06% - 57.25% - 70.2% - 80.4% - 100.0%]
23:00-23:30	0.94	0.94	0.94	0.94	[0% - 24.31% - 35.28% - 47.06% - 58.03% - 82.4% - 100.0%]
23:30-00:00	0.94	0.95	0.94	0.94	[0% - 23.14% - 37.25% - 48.25% - 59.22% - 76.06% - 100.0%]

(c) Lumping separate $\tau = 30$ minute intervals

Table 6.6: Accuracy for feature engineered NN for global vs. MI (with $\tau = 5$ minutes) vs. separate $\tau = 30$-minute interval lumping using 6 lumpted states (utilization ranges) in all cases.

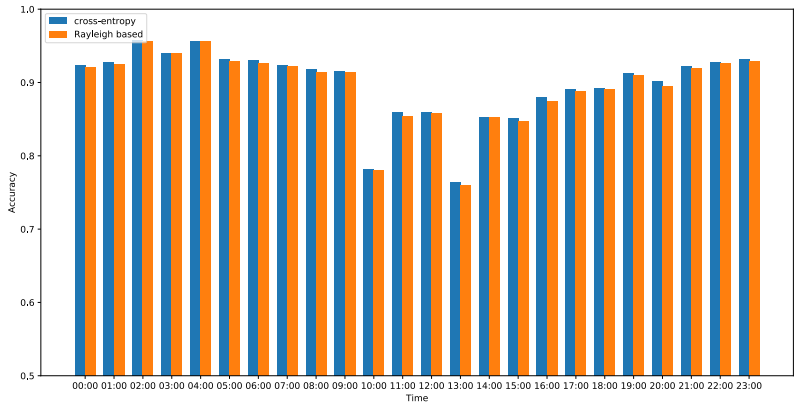
The impact of Loss Function

We run experiments for 5 and 6 lumped states with $\tau = 60$ minutes to compare the impact of a cross-entropy and the Rayleigh-based cost function. The Figure 6.7 compares the two penalty values for all intervals with 5 and 6 states. The accuracy results of the proposed loss function shows a very slight decrease at some of the intervals for the asymmetric penaly but this comes at a huge reduction in penalty values in almost all intervals. The increase is caused by the impact of penalizing under-estimation of the utilization when it is likely to be high (during busy hours), which could cause congestion if the intention is to use all, but no more than, the unutilized capacity. More importantly, looking at the peak penalty differences between the cross-entropy and the asymmetric cost function, they appear to be frequently at busy times of the day, or when the dynamics change as people leave the workplace. The asymmetric cost function avoids the "costly" underpreductions of utilization that could cause congestion when a significant number of users are stil present – and could have been impact by adverse effects of congestion.

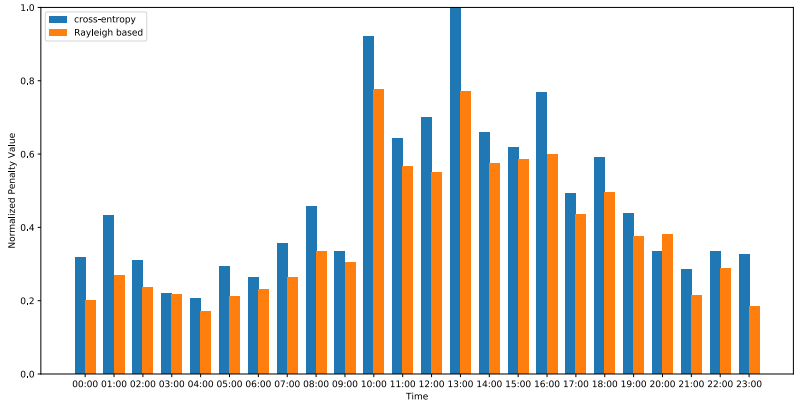
A closer inspection of what is the wrong predictions of the asymmetric cost function can be seen in Figure 6.10 where, e.g., between 3pm and 4pm, it mispredicts utilization range 1 (the lowest) more often than the entropy-based cost function. This means that the state is 1 but the asymmetric cost function considers it to be higher, hence is cautious of utilizing the leftover capacity. At higher utilizations, the two methods have less distinct behavior.

Constraints on Lumping Utilization Range

So far, in all of the experiments, meta-states produced from lumping states have constraints of $C_l = 10\%$ and $C_u = 25\%$. However, constraints need to adapt to the needs of the application. Previously, we claimed that given constraints on utilization range, the clustering coming from lumping, best fits data. In this subsection, we run

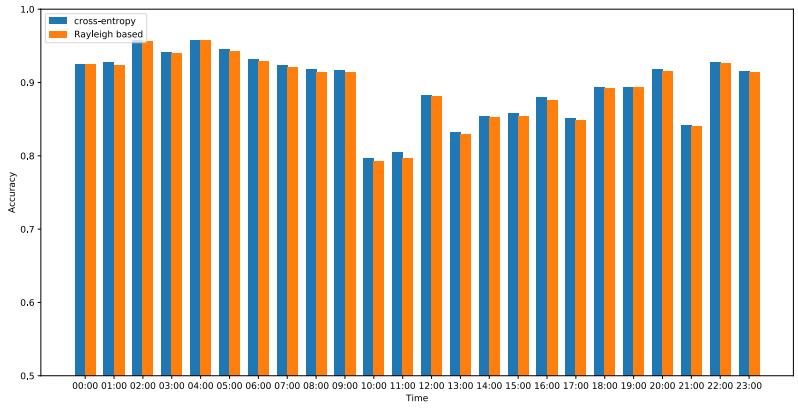


(a) Prediction accuracy

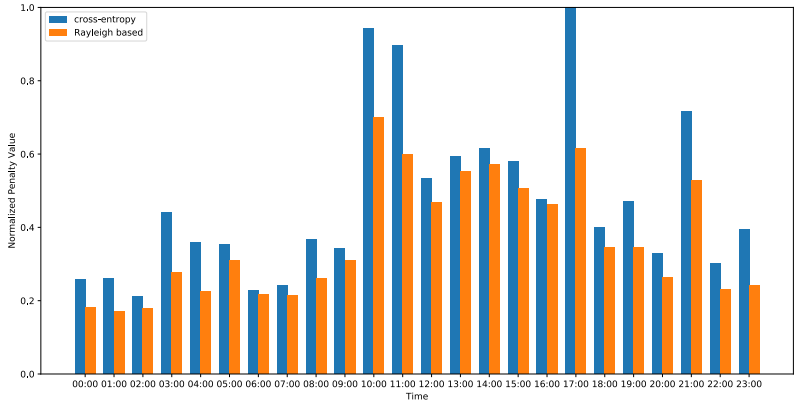


(b) Normalized penalty (highest = 1.0)

Figure 6.7: Cross-entropy vs. asymmetric (Rayleigh-based) loss function performance for 6 lumped states (utilization ranges).



(a) Prediction accuracy



(b) Normalized penalty (highest = 1.0)

Figure 6.8: Cross-entropy vs. asymmetric (Rayleigh-based) loss function performance for 5 lumped states (utilization ranges).

three experiments, one with $C_u = 35\%$, one with $C_u = 25\%$, and the last one with $C_l = 10\%$ and $C_u = 25\%$. Figure 6.10 shows the cumulative accuracy results over time-intervals with the aforementioned constraints. We used cumulative accuracy (summing up accuracy over time) since the changes on some of the intervals were not detectable by eye. But the cumulative accuracy shows the overall accuracy increases. It shows, as expected, as we loosen the constraints, we observe a boost in overall accuracy. To this end, the red line with $C_u = 35\%$ outperforms the rest. Also, the blue with $C_u = 25\%$ in overall performs better than the green line with $C_l = 10\%$ and $C_u = 25\%$ as the constraints are looser for the latter.

Multi-Step Ahead Prediction

So far, we investigated the capabilities of our model and extracted features in predicting 1-step ahead class of utilization. In these set of experiments, we aim to pursue the capabilities of our model in predicting several steps ahead. For the purposes of these experiments, we use the same set of features and the same Neural Network architecture shown in Figure 6.1. In contrast to other experiments that we incorporated 48 previously seen stationary values as feature, in this experiment we only use 24 of them. The reason behind this decision is that for our dataset, a larger number of lags spanning a larger stride simply do not exist which would lead to fewer data for training and testing. To this end, changing 48 features to 24 causes less information from past to be incorporated in prediction of the future values; hence, increasing number of training and testing samples from our dataset. In a larger dataset, there is no reason to prohibit us from using 48 features. In this experiment, we predict the class of utilization for 1, 2, 4, and 8 steps ahead for $\tau = 60$-minutes. We also use the default constraints of $C_l = 10\%$ and $C_u = 25\%$. The Figures 6.11a and 6.11b show the changes in prediction accuracy of multi steps-ahead for respectively 5 and 6 lumped states (utilization ranges). As the results suggest, the accuracy for multi-

step ahead prediction does not differ significantly as the number of steps increase but it does deteriorate as we predict for more steps into the future.

Deep Sequence Modeling Architectures

Finally, evaluating our Neural Network model with deep networks can aid us to determine how good our proposed network is. However, we are facing the issue of having not enough data for data hungry deep networks. With the amount of data collected over the past few months, the deep networks appears to overfit. Among possible deep networks we could use, sequence modeling networks such as Long Short Term Memory (LSTM), would be beneficial for time-series predictions. It is worth to mention that, as we discussed before, in some steps of our work (such as lumping), we converted an unsupervised time-series problem into a supervised clustering problem in which the goal is to predict the right class of utilization given some non time dependent features, extracted from time-series. Although we changed the objective function of the prediction, the goal stays the same, predicting utilization of the channel over time. To this end, comparing our model with a deep sequence modeling architecture makes sense. Two of the drawbacks of the deep networks, other than their data hunger, are their inability to provide us information regarding the importance and relevance of features extracted; and high computational cost in performing prediction. To this end, we used LSTM only for the sake of evaluating our model and provided features we used in our original proposed network. Using different machine learning and deep learning techniques, such drop out and adding regularizer, aided us to overcome overfitting; however, due to insufficient data, we cannot claim that the deep model is performing its best in predicting the utilization of the channel for a certain time step ahead. With these caveats, we implemented an LSTM with 2 hidden layers, for $\tau = 20$ minutes and with constraint of $C_l = 10\%$ and $C_u = 25\%$. For this setup, LSTM gave us the overall accuracy around 92.3%.

Our NN model with the same constraints and τ value had an accuracy of 91.94%. It suggests the task of prediction having the features provided, does not need a very deep network. Providing more data would most likely increase the accuracy of LSTM to a more meaningful difference against our NN, but by how much is a just a matter of speculation at this point.

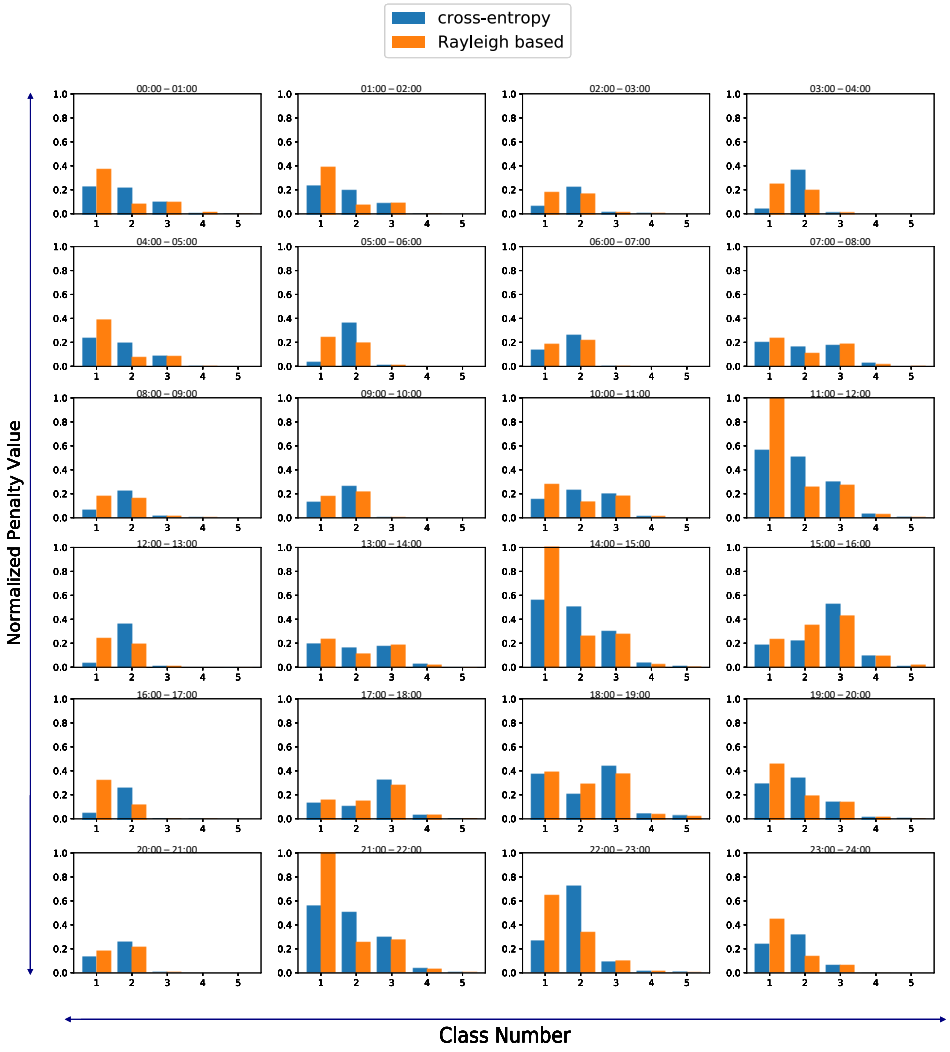


Figure 6.9: Comparison of wrongly predicted classes, at various times of the day, for normal cross-entropy and proposed asymmetric loss function for 5 lumped states (utilization ranges) and $\tau = 60$ minutes.

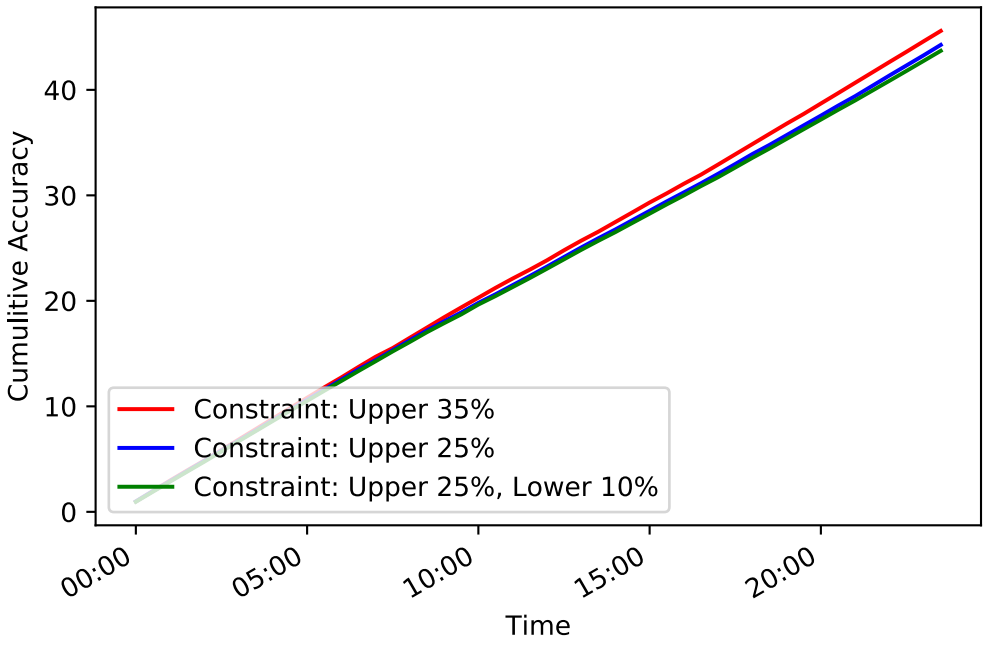
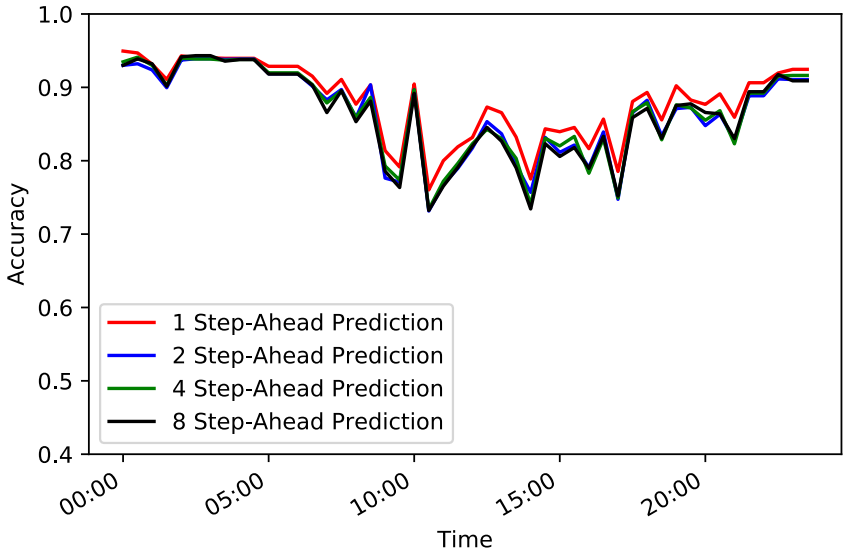
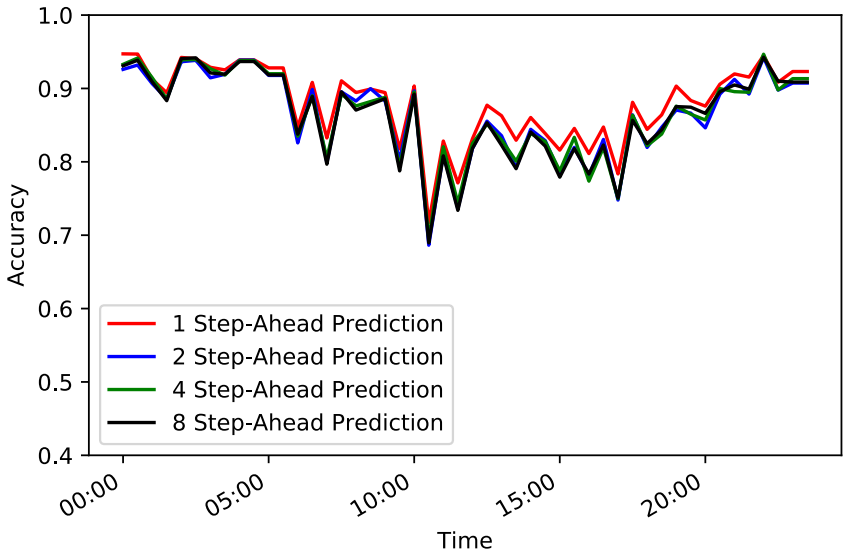


Figure 6.10: The impact on (cumulative) accuracy of different utilization range constraints imposed during lumping.



(a) 5 lumped states (utilization ranges)



(b) 6 lumped states (utilization ranges)

Figure 6.11: Multi-step prediction accuracy ($\tau = 60$ minutes).

Chapter 7

Conclusions and Future Work

In this thesis we focused on techniques for predicting utilization of a wireless channel. The framework assumed is one where there exist ways to collect utilization observations, the measurements are stored and models are generated based on them, and finally the models are used to form on-demand predictions. Practicality concerns, both for the development of this thesis, as well as for a potential deployment, suggest that the data collection and model production should probably take place near the "edge" of the network, i.e., as a form of *edge computing*. After all, we expect that a model fit for a particular location is not necessarily good for another location. Future work should explore whether this assertion is true or if models developed in one location are suitable at other locations. We should add that the environment in which the data collection was performed, is covered by a centrally managed controller for most WiFi APs (although several user-owned APs also exist, but do not account for a lot of traffic). Data collection in a residential environment is expected to behave differently, both in terms of differences are different times of the day, and due to the lack of coordination of how channels are (re)used across space.

A significant amount of time was spent in developing the data collection infrastructure. It involved the deployment of various sniffers and their observations were sent to a backbone computing infrastructure. The data collected have only been

partially studied in this thesis. More data, for the "popular" channels (1, 6, and 11) as well as for different locations are still waiting analysis. To this end, future work should look into the correlations of the utilization across channels; eventually leading to a corresponding study of channel switching strategies.

Our modelling efforts started with splitting the utilization time-series into intervals, each one of them approximated as a stationary process modeled as Markov chain in order to capture its short-term behavior. We successfully demonstrated in this thesis that the similarity between the short-term behavior of each interval and a "library" of prior behaviors benefit the model. So much so that this similarity is a means to capture the time of the day (since when we explicitly added the time of the day, it did not improve the prediction). Additionally, we exploited the lagged correlation to incorporate, what we called the stationary component, after seasonality was eliminated from the time series. We demonstrated that these features helped the learning algorithm to better capture the dynamics of the environment.

Given that very few applications have the finesse to decide on actions for minuscule differences in predicted utilization, we resorted to a means of "coarsening" utilization ranges by applying lumping technique to the state transition matrices of the first-order Markovian models of each time interval. The experiments suggest feeding the features into a shallow Neural Network (NN) empowers it to predict utilization ranges with an overall promising accuracy, even if called to perform multiple step prediction. Looking at the weights of the trained network, it is evident that the class previously observed has a strong influence on the next prediction, indirectly validating why naïve predictors can work very well in a number of cases. Nevertheless, the additional features build up an advantage over naïve.

Even though we attempted to remain application-agnostic, we tried to capture a plausible concern for certain applications, namely the concern that it is "better" to underutilize the channel rather than create congestion (and impacting other traffic

flows as well) if we overutilize it. For this goal, we demonstrated the beneficial impact of an asymmetric cost function following a Rayleigh function, and comparing it against a classic application of cross-entropy. Whle it is possible that the approach requires fine tuning, it was evident that for the same overall accuracy, the asymmetric function was avoiding to call the channel underutilized and therefore would help an application avoiding overutilizing it.

We did not concern ourselves with the architecture of a future system that provides wireless utilization prediction services. To be useful it should be able to adopt in real time, using powerfull processing close to the observations. Therefore, future work could be an online updating of this model. One example update is that of making the lumping into ranges of utilization more flexible. Lumping assumes noise is involved in the Markov chain transitions as derived by the observations, so that reducing the state space is not possible without introducing error. At the same time, lumping is computationally expensive; hence, updating it without starting the calculation from scratch could be challenging. Future work is needed to find the best online algorithms to revise the lumping algorithms.

Another future work would be finding correlations among Access Points' (AP) signals. Part of our main focus in this project was to find features which enhance the ability of the learner to predict the utilization of the channel. One of the features that we never discussed due to the scope of the project was the correlations among same channels of different APs. We observed in some of our experiments that there are some correlations among utilization of the same channels of APs. The reason behind this is the fact that the traffic from a user connected to an AP, using a specific channel, keeps the channel busy and it sensed as such by other APs (unrelated to the user) which operate nearby on the same channel – a straightforward consequence of wireless being a shared medium. Therefore, the utilization of the channel of all APs covering the same area bears strong correlations because of the number of users

"seen" as keeping the medium active overlaps among APs. We see this as a source for two potential directions. One is to develop a model to predict which channel to switch to at a *given location* – i.e., add the location as one of the dimensions of the model. This will be in contrast to previous channel-switching work where location was not taking part in the model. Equally, the correlations across AP utilization could be used as the basis for developing models that can inform us if two APs are near or far from each other. In an institutional deployment as the one where the data collection happened, the AP locations are known. The same cannot be said about residential environments though where the AP deployment is arbitrary.

Finally, the work in this thesis is solely focused on the assumption that a device taking advantage of various prediction techniques is interested to utilize the channel as much as possible. What if multiple devices predict the same and decide to do so at the same time, leading to congestion? There are many ways to prevent this from happening by having a controller (possibly co-located with APs – announcing its predictions with possible extensions to the Beacon payload) performing the predictions and apportioning (explicitly or implicitly) the predicted available capacity to the various devices. Load and call admission concerns have made it into the 802.11 standard as various amendments but we think the real value is if this is done in a coordinated way across APs that *do not necessarily* have a common network manager, i.e., in a decentralized way. It might even be possible to announce the model parameterizations the APs have used to form the predictions as well as the predictions on the number of client devices which could attempt access to the channel in the next step. For an analysis of this kind of an approach, data would need to be collected that do not just preserve the utilization measurements, but also identify the individual devices present in the vicinity. Given the expanding use of MAC randomization, this could be a challenging task.

References

[1] B. Abraham and J. Ledolter, *Statistical methods for forecasting*. John Wiley & Sons, 2009, vol. 234.

[2] R. Akl and A. Arepally, "Dynamic channel assignment in IEEE 802.11 networks," in *2007 IEEE international conference on portable information devices*, IEEE, 2007, pp. 1–5.

[3] N. Arianpoo and V. C. Leung, "How network monitoring and reinforcement learning can improve TCP fairness in wireless multi-hop networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, p. 278, 2016.

[4] B. Balaji, J. Xu, A. Nwokafor, R. Gupta, and Y. Agarwal, "Sentinel: Occupancy based HVAC actuation using existing WiFi infrastructure within commercial buildings," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2013, p. 17.

[5] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[6] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[7] P. Buchholz, "Exact and ordinary lumpability in finite Markov chains," *Journal of applied probability*, vol. 31, no. 1, pp. 59–75, 1994.

[8] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[9] Cisco, "The dynamic channel assignment white paper," [Online]. Available: `https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-3/b_RRM_White_Paper/b_RRM_White_Paper_chapter_0100.pdf` (visited on 06/29/2020).

[10] T. Dayar and W. J. Stewart, "Quasi lumpability, lower-bounding coupling matrices, and nearly completely decomposable markov chains," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 2, pp. 482–498, 1997.

[11] S. Derisavi, H. Hermanns, and W. H. Sanders, "Optimal state-space lumping in Markov chains," *Information Processing Letters*, vol. 87, no. 6, pp. 309–315, 2003.

[12] F. Fainelli, "The OpenWrt embedded development framework," in *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[13] J. Froehlich and J. Krumm, "Route prediction from trip observations," SAE Technical Paper, Tech. Rep., 2008.

[14] C. Ghosh, C. Cordeiro, D. P. Agrawal, and M. B. Rao, "Markov chain existence and hidden Markov models in spectrum sensing," in *2009 IEEE International Conference on Pervasive Computing and Communications*, IEEE, 2009, pp. 1–6.

[15] J. Gong, X. Zhong, and C.-Z. Xu, "Maximizing rewards in wireless networks with energy and timing constraints for periodic data streams," *IEEE Transactions on Mobile Computing*, vol. 9, no. 8, pp. 1187–1200, 2010.

[16] E. J. Hannan, *Multiple time series*. John Wiley & Sons, 2009, vol. 38.

[17] "IEEE Standard for Information technology–Local and metropolitan area networks–specific requirements–part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications - amendment 8: Medium access control (MAC) quality of service enhancements," *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003)*, pp. 1–212, Nov. 2005, ISSN: null. DOI: 10.1109/IEEESTD.2005.97890.

[18] "IEEE Standard for Information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec. 2016. DOI: 10.1109/IEEESTD.2016.7786995.

[19] Y. Jiang, D. C. Dhanapala, and A. P. Jayasumana, "Tracking and prediction of mobility without physical distance measurements in sensor networks," in *2013 IEEE International Conference on Communications (ICC)*, IEEE, 2013, pp. 1845–1850.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[21] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.

[22] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic: Observations and initial models," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, IEEE, vol. 2, 2003, pp. 1178–1188.

[23] S. J. Tarsa, M. Comiter, M. B. Crouse, B. McDanel, and H. Kung, "Taming wireless fluctuations by predictive queuing using a sparse-coding link-state model," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2015, pp. 287–296.

[24] A. Vattani, "The hardness of k-means clustering in the plane," *Manuscript*, vol. 617, 2009. [Online]. Available: https://cseweb.ucsd.edu/~avattani/papers/kmeans_hardness.pdf (visited on 06/29/2020).

[25] A. M. Voicu, L. Simic, and M. Petrova, "Survey of spectrum sharing for inter-technology coexistence," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 2, pp. 1112–1144, 2019. DOI: 10.1109/COMST.2018.2882308. [Online]. Available: https://doi.org/10.1109/COMST.2018.2882308.

[26] W. Wanalertlak, B. Lee, C. Yu, M. Kim, S.-M. Park, and W.-T. Kim, "Behavior-based mobility prediction for seamless handoffs in mobile wireless networks," *Wireless Networks*, vol. 17, no. 3, pp. 645–658, 2011.

[27] W. Wang, J. Chen, T. Hong, and N. Zhu, "Occupancy prediction through markov based feedback recurrent neural network (M-FRNN) algorithm with WiFi probe technology," *Building and Environment*, vol. 138, pp. 160–170, 2018.

[28] W. Wang, J. Chen, and X. Song, "Modeling and predicting occupancy profile in office space with a Wi-Fi probe-based dynamic Markov time-window inference approach," *Building and Environment*, vol. 124, pp. 130–142, 2017.

[29] Y. Wang and L. Shao, "Understanding occupancy pattern and improving building energy efficiency through Wi-Fi based indoor positioning," *Building and Environment*, vol. 114, pp. 106–117, 2017.

[30] X. Xing, T. Jing, W. Cheng, Y. Huo, and X. Cheng, "Spectrum prediction in cognitive radio networks," *IEEE Wireless Communications*, vol. 20, no. 2, pp. 90–96, 2013.

[31] X. Xing, T. Jing, Y. Huo, H. Li, and X. Cheng, "Channel quality prediction based on bayesian inference in cognitive radio networks," in *2013 Proceedings IEEE INFOCOM*, IEEE, 2013, pp. 1465–1473.

[32] J. Yang and Z. Fei, "Broadcasting with prediction and selective forwarding in vehicular networks," *International Journal of Distributed Sensor Networks*, vol. 9, no. 12, p. 309 041, 2013.

[33] Z. R. Zaidi and B. L. Mark, "Real-time mobility tracking algorithms for cellular networks based on Kalman filtering," *IEEE Transactions on Mobile Computing*, vol. 4, no. 2, pp. 195–208, 2005.