Abdullah Farouk

HOURLY BIKE RENTAL PREDICTIONS

April 15, 2019

Student ID: 71244370
University of British Columbia
Department of Statistics

Contents

EXECUTIVE SUMMARY

In this report we predict the number of hourly bikes rides in Washington D.C. between November and December of 2012. The high frequency nature of ride count time creates a few interesting challenges. In this paper, we address some of these challenges and overcome them using a class of statistical models called GAMs. We find that GAMs provide a lot of flexibility in the modelling process and produce the highest accuracies.

Chapter 1

Introduction

Riding bikes are a popular mode of transport today. It is estimated that over 500 ride sharing systems exist around the world. Due to this popular mode of transport, people now use bikes to commute to and from work. Predicting hourly bike ride counts is useful for a variety of reasons. Many companies use ride sharing data to accurately predict traffic flows and identify congested routes in cosmopolitan areas of a city, on a daily basis. Predicting ride counts accurately is challenging. These counts are often recorded over small intervals of time over many years. The high frequency and long length of these time series makes it difficult to use traditional time series models on them. In this paper we compare a variety of statistical and machine learning models in their abilities to accurately predict hourly bike ride counts.

1.1 DATA DESCRIPTION

The data contains a mix of continuous and categorical variables. There are no missing values in the data set. Initial exploration of the data revealed no outliers either. A comprehensive table with descriptions of the variables in the data set can be found in the appendix.

1.2 DATA TRANSFORMATION

The goal in this paper is to predict hourly ride counts. Since counts are positive numbers, we cannot use its raw values as inputs in any of the regression methods discussed in the upcoming chapters. This is because, most regression models assume the dependent variable can take any real number. To circumvent this problem, we seek to use a power transformation. Box-Cox analysis of a suitable power transformation reveals that 0 is inside the 95% confidence interval of its estimate of lambda. Hence our counts are transformed using:

$$T(Y_t) = log(Y_t) + C \qquad C = 1 \tag{1.1}$$

The constant is added to ensure all counts are > 0. This prevents our models from producing negative ride count predictions. Taking logs transforms our model from a multiplicative to an additive model. This makes it easier to work with time series whose variance changes proportionately with the trend (e.g. ride counts series).

Reversing the log transformation is not so straightforward. Exponentiation returns the median of the forecast densities. An adjustment is required to get back the mean. Hatko et al. [1] outlines a simple method of moments based adjustment. Its only assumption is that residuals be identically distributed. We present a variant of this correction below. Hyndman et al. [2] recommends an alternative correction based on out of sample errors. We however prefer Hatko et al.'s correction as it does not require the errors be normally distributed.

$$\begin{gathered} \text{Let D} = \frac{\sum_i^n Y_i}{\sum_i^n exp(Y_i - C)} \\ Y_t = T^{-1}(Y_t) = D \times exp(T_{Y_t} - C) \qquad C = 1 \end{gathered} \tag{1.2}$$

Chapter 2

Exploratory Data Analysis

In this chapter, we analyze the variable in our data set. In particular we seek to identify temporal features of the ride count time series. These features give us insights into the nature of the time series we are working with and enable us to identify appropriate models to use to predict future values. Additionally, we identify other variables to incorporate into our models to improve prediction accuracy.

2.1 Features of Rental count Time Series

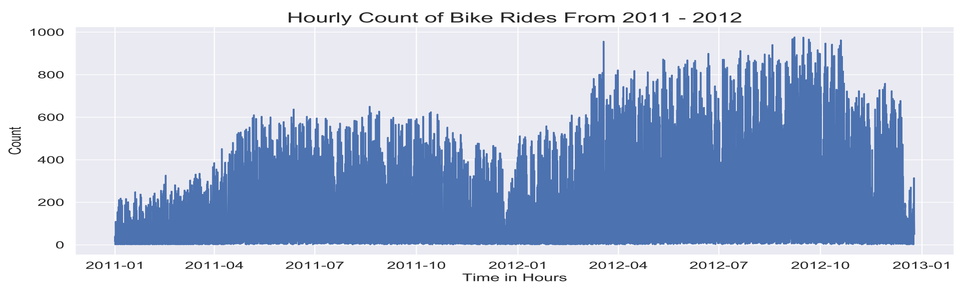

Figure 2.1: Hourly ride counts from January 2011 to December 2012

The image above displays the values of ride counts over time. Two two things stand out in this picture. One is a clearly increasing trend in ride counts. We notice that these counts are much higher in 2012 compared with 2011. Furthermore, we also observe that counts exhibit annual seasonal variation (i.e they are higher in the summer, compared to other months in the year). However is this the only type of seasonality we observe in our data?

2.1.1 Multiple Seasonalities

Initial exploration of the data reveals that ride counts three significant patterns of seasonal variations. They are variations due to the time of day (daily), day of the week (weekly) and month of the year (annual). More details can be found in our presentation slides

2.1.2 Effects of Environmental Variables

The data set contains multiple environmental variables . Some of them are categorical, like weathersit, whilst others take real values (e.g. temp & hum). It is quite plausible that people would prefer to ride their bikes when it is sunny outside and are less likely to do so when it's raining outside. The plots below highlight our findings from investigating the effect of environmental variables discussed above on ride counts.

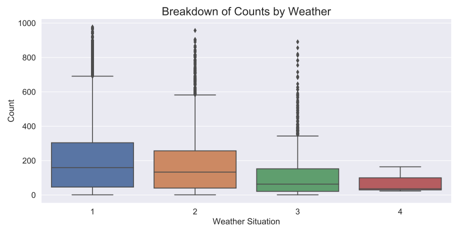

Figure 2.2: Distribution of ride counts under different weather conditions

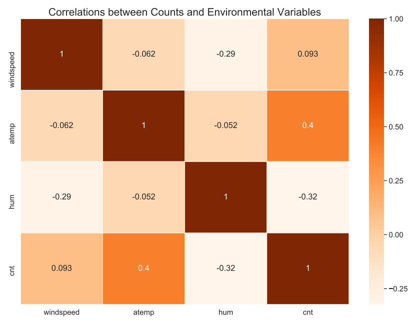

Figure 2.3: Strength of linear relationship between environmental variables and counts

It is clear from the picture above that people weather situation affects people's decision to ride bikes. Incorporating such information into our models would therefore improve our predictions. Other environmental condition (EC) variables like humidity, wind speed and temperature also have a noticeable effect on ride counts. The heat map above indicates that whilst there is a linear relationship between ride counts and EC variables, it is modest at best.

From our exploratory analyses above, we notice that traditional time series techniques like ARIMA models cannot be used by themselves. This is because they are not designed to work on time series with multiple seasonalities. Furthermore, we seek to use models that can include additional regressors with ease. This stems from our observation of the significant effects of environmental variables on ride counts. Hence we expect prediction accuracy to go up when information from all of these variables are included.

In the next section we discuss two types of models. The first is a hybrid. It uses a combination of classical uni variate time series techniques to predict counts. The second is an alternative class of models that have recently gained popularity in the machine learning literature. They are general additive models (GAM). We make extensive use of these models as they allow us to incorporate multiple regressors in a flexible manner.

Chapter 3

Models

To predict rental counts, we use a very popular class of models from the time series literature. They are decomposable models. These models take the form:

$$Y_t = \text{Trend}_t + \text{Seasonal}_t + \text{Remainder}_t \tag{3.1}$$

The decomposition specified above breaks down variation in a natural and intuitive manner. Models built upon this premise provide us with great flexibility to model sources of both predictable and random variation. Each of the models discussed in the next sections vary in how they capture the underlying trend and seasonality in the data.

3.1 Forecasting using STL Decomposition & ARIMA

To forecast time series with multiple seasonal patterns, we employ a simple forecasting procedure outlined by Hyndman et al [2, 3]. This technique works as follows. A time series is decomposed as described in equation [4.1]. This decomposition is performed iteratively since ride counts exhibit multiple seasonalities. Predictions of each seasonal components' values are computed in a naive manner. The trend and residuals are then combined to form a seasonally adjusted series.

Visual analysis of the seasonally adjusted component shows it to be non-stationary. This is also verified via an ADF test (details in code scripts). We find it plausible that ride counts in the next hour may depeond on the ride count in the previous hour. Thus an ARIMA model seems appropriate to model the seasonally adjusted component. The predictions from each of these methods are then added together to obtain a final prediction.

This decomposition is performed using STL (seasonal and trend decomposition using locally weighted regression - (LOESS)). STL in a nutshell applies LOESS sequentially to obtain a smooth decomposition of the seasonal and trend components of a time series. STL has 6 parameters. Of these the parameters of interest to us are the trend (n_t) and seasonal window (n_s) sizes. Small values allow for rapid changes whilst large values allow for smoother changes along the decomposition. n_t is computed after the annual seasonality is removed from the data. Thus it's smoothness is estimated using a much larger window size. We reduced its window size as its default setting created remainder components that had a very irregular shape. We considered altering the number of robustness iterations but did not as we did not find any outliers in our data. We found the rest of the pre-defined settings specified by Cleveland et al. [4] sensible for our problem at hand and chose to keep them as is.

We vary n_s according to the type of seasonality we seek to capture. n_s is set to 24 (daily), 24x$\frac{330}{52}$ (weekly) and 24x330 (annual) respectively. We use 330 days as we do not have enough data points to capture 2 cycles of annual seasonality. Whilst this does not capture the annual seasonality effect completely, it provides a reasonable approximation. This ensures that our results are comparable across models.

We prefer STL over other decomposition techniques for multiple reasons. The first is that can handle any type of seasonality. Second, it allows us to specify the degree of smoothness of the trend and seasonal components. Third, it is able to decompose long time series quickly. However there are some limitations from predicting ride counts using this method. The first is that the decomposition is not great. The plot below displays the residuals after fitting our hybrid model to the data. We see that there are a lot of correlations that it does not pick up. The second is that additional regressors cannot be incorporated to account for variation in ride count values. This is problematic as we know that environmental conditions affect people's decision to ride bikes. In the next section we present a model that overcomes these limitations.

3.2 FACEBOOK PROPHET

Facebook Prophet (Prophet) is a new time series model developed by Facebook. It was developed to accurately predict time series with multiple seasonalities. It is formulated as:

$$Y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{3.2}$$

The equation above uses non-linear functions of time, to model trend (g(t)), seasonality (h(t)) and holiday effects (h(t)). Applying non-linear smoothing functions to the regressors allows

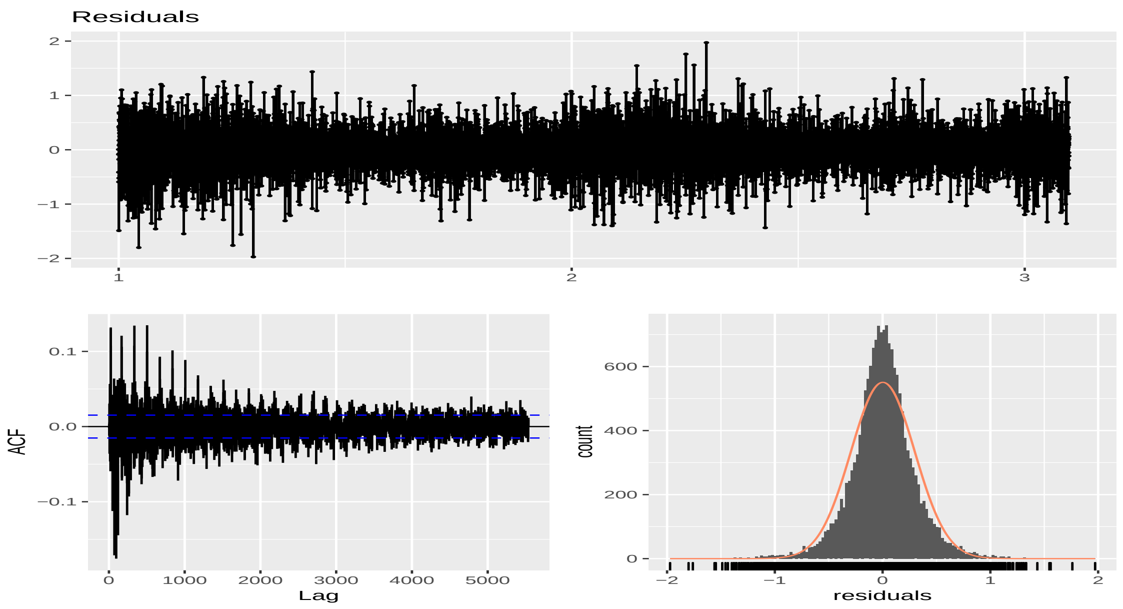

Figure 3.1: STLF is unable to capture all the correlation in ride counts

Prophet to be formulated as a General Additive Model (GAM). This is useful as information from additional regressors can be incorporated by simply adding them to the model.

3.2.1 Modelling Trend

Prophet offers two options to models trends in time series. The first option is to use a non-linear saturating growth model. This allows users to specify a growth ceiling (e.g. population cap) at which growth will saturate. The second is a linear constant growth model whose slope is allowed to change at specific change points. This is similar to a kernel regression with local linear fits. We prefer the second option. This is because we only have data on two years. Furthermore, without broader context about competition and transportation preferences, we are unable to specify a realistic saturation capacity. The trend is formulated as follows:

$$g(t) = (K + a(t)^{\intercal}\delta)t + (m + a(t)^{\intercal}\rho) \tag{3.3}$$

where k is the growth rate and m is an offset parameter. Let s_j, j = 1,...,S denote the points at which the trend's growth rate is allowed to change. Let δ_j denote the change in rate that occurs at time j. The growth rate at any time t is then given by $k + \sum_{j:t>s_j} \delta_j$. The summation can be concisely expressed as $a_t = 1$ if $t > s_j$. Thus the growth rate at any time t is given by $(K + a(t)^{\intercal}\delta)t$ (vector form).

$(m + a(t)^\intercal \gamma)$ are the adjustments made to the offset parameter to correctly join the different line segments. More details on this adjustment can be found in [5]. Change points are automatically specified. This is done by choosing a large number of points uniformly using 80% of the data. A sparse prior is then used on δ i.e. $\delta_j \sim \text{Laplace}(0, \tau)$. This prior regularizes the fitted trend model by setting most of the δs to 0. This is easy to achieve with a Laplace prior as its mass is always concentrated at its location parameter, for any value of τ.

3.2.2 Modelling Seasonality

Prophet uses Fourier terms (sine and cosine pairs) to model seasonality. Due to it's additive nature, multiple Fourier terms, with varying cycle lengths can be included as follows:

$$r(t) = \sum_{i}^{N} [sin(\frac{2\pi i t}{P}) + cos(\frac{2\pi i t}{P})] \tag{3.4}$$

where P is the period of the seasonal pattern. This allows the model to incorporate multiple seasonal patterns. Better fits of the seasonal pattern can be obtained by varying the number of Fourier terms used. N can then be thought of as a smoothing parameter. We favor modelling seasonality in this manner for a few reasons. Since we only have two years of data, we cannot obtain a reasonable estimate of the annual variation. Furthermore it may not be appropriate to use differencing to estimate seasonal patterns when dealing with high frequency time series (differencing compares the ride count at 1 am in January 2011 with the count at 1 am in January 2012. A lot of daily random variation is unaccounted for by examining the difference between these two values.)

Prophet uses priors to control the effects of each seasonal pattern on predictions as follows:

$$s(t) = r(t)\beta \tag{3.5}$$

A normal prior is applied to the βs to regularize their effects on the final prediction.

3.2.3 Modelling Holidays and Other Exogenous Variables

Holidays often cause unexpected shocks that are hard to account for by modelling seasonal variation. As seen in the EDA chapter, weather conditions have a considerable effect on ride counts. Incorporating information about these events could improve our predictions. These predictors are easy to incorporate into Prophet due to its additive nature. Assuming holidays are independent of one and another, we can use the following equation to model their effects:

$$h(t) = I(t)\theta \tag{3.6}$$

Here $I(t) = 1(t \in D_1, \ldots, 1(t \in D_L$ is a set of indicator variables that indicate if the day in question is a holiday. A normal prior is applied to θ to regularize its effects. Additional regressors are included into the model in a similar fashion.

As we will see in the next section shortly, Prophet predicts hourly ride counts well. Furthermore its model of seasonal variations line up well with those seen in the EDA phase. However it does have a few limitations. It's trend confidence intervals do not have exact coverage. This is because it assumes the trend will continue to change with the same frequency and magnitude as observed in the past. Second, its far too time consuming to generate confidence intervals for each seasonal pattern. This is because it uses MCMC sampling to build these intervals. The third limitation is its use of priors as a form of regularization. Unlike regular regularization parameters these parameters can take any value > 0. This makes it hard to specify their values even with reasonable prior knowledge of the process being modelled, without significant trial and error.

3.3 XGBoost

In this section we use decision trees to predict hourly ride counts. Decision trees are often used in machine learning to predict many outcomes of interest. They are able to model complex non-linear relationships between a dependent variable and several independent variables well. Additionally they are interpret-able and can be used to predict continuous outcomes (regression trees). Aggregating predictions from multiple trees (i.e. creating an ensemble) has shown to significantly improve prediction accuracy. The ensemble method we use is called Boosting. In this technique a weak learner (e.g. trees with one or two nodes) is fit to the data, with all points weighed equally. Next, the residuals from this fit are computed. Points predicted correctly (small residuals) are given low weights, whilst those that were not (large residuals) are given higher weights. Another weak learner is then fit onto the re-weighted data set. This process is repeated until the gain in prediction accuracy becomes negligible. A weighted (according to their accuracy) average of each learner's predictions are then computed to create a final prediction. It can be formally expressed as:

$$\hat{Y}_t = \sum_{i}^{k} f_k(X_t), \quad f_k \in \mathscr{F} \tag{3.7}$$

where X_t is a set of predictors, f_k are simple independent trees and $\mathscr{F}$ is the space of weak learners. A generalization of the procedure mentioned above is to boost the loss function used to evaluate prediction accuracy. These models are referred to as gradient boosting models (GBM). It is immediately clear that boosting can be formulated as a GAM. XGBoost provides a quick and efficient implementation of the gradient boosting algorithm discussed above. The loss function used by XGBoost to learn the set of models learned (f_k) is:

$$\mathscr{L}(\Phi) = \sum_i l(\hat{Y}_i, Y) + \sum_k \Omega(f_k) \tag{3.8}$$

$\mathscr{L}(\Phi)$ is a regularized convex loss function the user seeks to minimize. $\Omega(f_k)$, described in detail in [6] penalizes model complexity (i.e. the set of f_k. It is included to prevent over fitting. Chen et al. [6] provides a detailed description of how to tune each parameter in XGBoost. We chose to tune only a few of these parameters due to time constraints. Thus we only vary values of parameters that directly affect model complexity. This includes the step size used to update weights (i.e. the learning rate), the number of trees included in the ensemble and finally the number of columns used in each split.

Chapter 4

Results

In this section we compare the predictive performance of the models discussed above. All three models under consideration have multiple parameters. To make it a fair comparison, we compare the best version each model against one and another. This is achieved by splitting the data into a training (data from January-01-2011 to November-30-2012) and test (data from December-31-2012) set. The models are trained and tuned on subsets of the training set to identify model variants with the highest accuracy. This process is described in further detail in the next section. Once the optimal parameters have been selected for each model under study, they are trained on the entire training set, using the optimal parameters found. The final step involves using these tuned models to predict values in the test set and comparing their predictive performance thereafter.

4.1 Expanding Window Validation Sets

We study a medium data set in this paper. Thus a single validation set will not provide a robust estimate of a model's generalizability. Instead we use a popular back testing strategy for medium sized data sets from the time series literature. It is called expanding window validation. The idea is simple. Set aside k ~ equal length portions of data as validation sets. We set k = 3, where each month from September 2012 to November 2012 is used as validation sets. Next, train all models on data up until the first split (August 31 2012). Compute each model's prediction accuracy on the held out test set (i.e.September 2012).

Once this is done, add the validation split back into the training set and repeat the process above until all folds of the validation set have been iterated over. Once the optimal variant of each model is found, train these optimal variants on all available data in the training set (i.e. until November 30 2012).

Optimal parameters are chosen after back transforming predicted values. These parameters minimize the mean absolute error (MAE), across all parameter values under consideration.

The final step is to compare the predictive accuracy of each model on the held out test set.

4.2 Comparing prediction accuracy

Model	Training Set			Test Set		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE
STLF	0.296	0.215	0.052	1.29	0.99	0.238
Prophet - D	0.661	0.492	0.118	1.02	0.734	0.190
Prophet - T	0.376	0.256	0.066	0.667	0.445	0.117
XGB	0.339	0.227	0.059	0.519	0.355	0.088

Table 4.1: Table displays log accuracy scores of each model. We use the following shorthands Prophet - D: Default FB GAM, Prophet - T: Tuned FB GAM, XGB - Tuned XGBoost

Model	Training Set			Test Set		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE
STLF	62.81	33.35	22.44	179.7	114.6	2.027
Prophet - D	105.4	69.79	0.753	127.8	85.89	1.557
Prophet - T	58.28	34.27	0.303	65.03	44.23	0.691
XGB	42.79	26.57	0.268	58.85	38.38	0.521

Table 4.2: Accuracy measures were computed after applying bias adjusted transformation.

We did not tune STLF, for reasons mentioned in section 4.1. Prophet - B was not tuned either as we wanted to test the performance of its default parameters. The table shows us some interesting trends. First, it is clear that Prophet and XGBoost have the best predictive performance. This is because these models are able to capture seasonal variations much better than STLF. Second STLF has the highest training set accuracy. However its test set performance is the worst - a clear indication that it is over-fitting.

Plots of back transformed counts can be found in Appendix A.2. Here we observe that all of our models consistently under predict observed ride counts. This is verified by the large discrepancy between log accuracy scores and their back transformed counter-parts.

The plot above shows residuals of Prophet - T and XGB. These plots are clearly non-stationary and therefore unlikely to be identically distributed. Thus the underlying assumption of our bias adjustment is violated. This is confirmed by the ADF test statistic. Its p-values

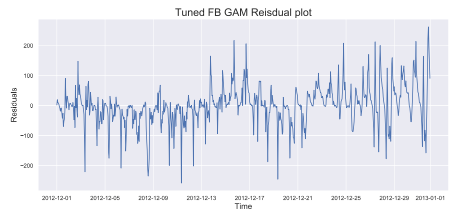

Figure 4.1: Tuned Prophet model's residuals look non-stationary

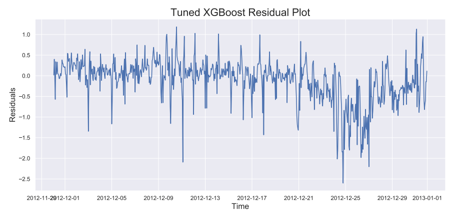

Figure 4.2: XGB Residuals seem to have a non-constant mean and variance

for each residual series is large. Whilst our bias adjustment may not have captured all the bias, we are fairly confident in our ranking of each model's performance. This is because log transformations are monotonic. Under such transformations ranks are preserved.

4.3 Conclusion

High frequency time series are challenging to model accurately. In this paper we identify several complexities that arise with such data. We find that incorporating additional regressors into our models greatly improves predictions. This is because high frequency signals are very noisy. Therefore traditional models, designed to work on smoother low frequency signals, are not able to capture all of the variation in these time series. In this sense, use of non-linear smoothing models like GAMs are preferred over classical time series methods (e.g. STL decomposition) for these kinds of problems. Our results show that they produce more accurate predictions over a short to medium term prediction window.

Appendix A

Appendix

A.1 DETAILED DESCRIPTION OF COLUMNS IN DATA SET

Variable	**Description**
Instant	Record Index.
dteday	Date-time.
season	4 level categorical variable.
yr	Binary variable that indicates year.
mnth	Categorical variable that identifies month observation was recorded in.
hr	Categorical variable that identifies hour observation was recorded in.
holiday	Binary variable that indicates if date is a holiday in Washington D.C.
weekday	Categorical variable that identifies the day of week observation was recorded on.
working day	Binary variable that identifies if day is a weekend or weekday.
weathersit	4-level categorical variable that describes weather condition outside.
temp	Normalized temperatures in Celsius (all values divided by 41 (max)).
atemp	Normalized real feel temperatures in Celsius (all values divided by 50 (max)).
hum	Normalized humidity (all values divided by 100 (max)).
windspeed	Normalized wind speed (all values divided by 67 (max)).
casual	Number of casual users.
registered	Number of registered users.
cnt	Total number of users.

A.2 Back transformed prediction figures

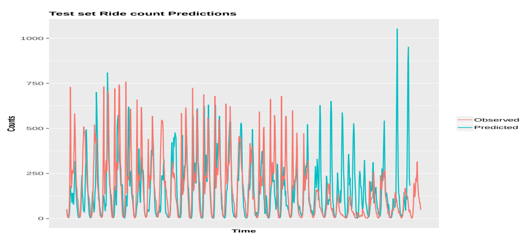

Figure A.1: STLF's long term prediction quality deteriorates rapidly

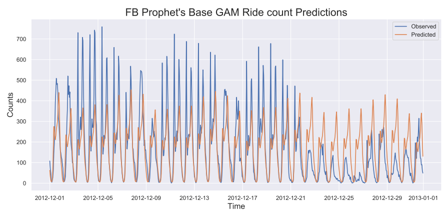

Figure A.2: Prophet's base model is able to predict end of year counts better than STLF

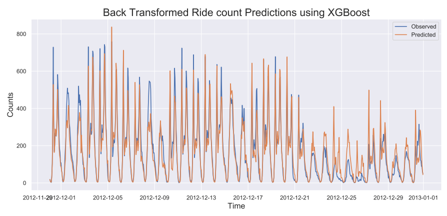

Figure A.3: Tuned FB model is able to capture trend and seasonality better

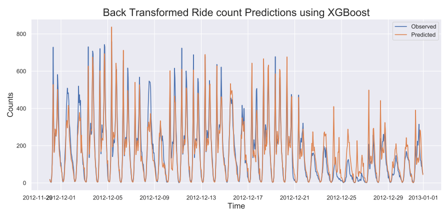

Figure A.4: Tuned XGBoost model outperforms Prophet and STLF

A.3 Importance Plots

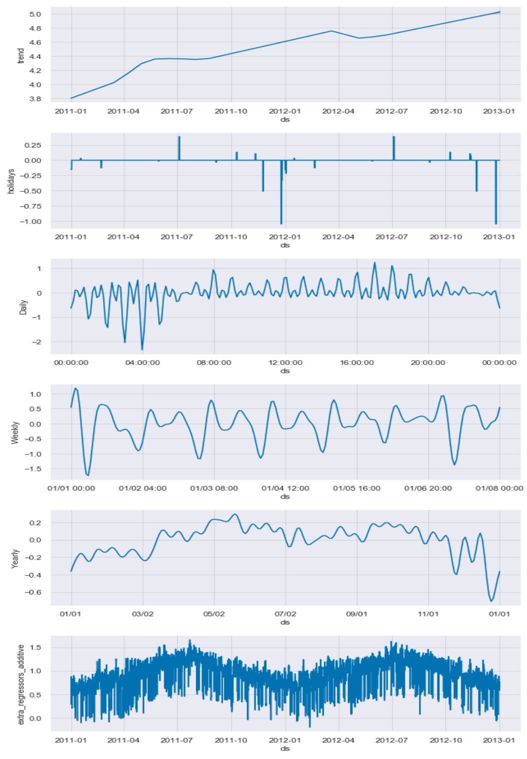

Figure A.5: Tuning Prophet by increasing the number of fourier terms used to capture daily seasonality significantly improves its predictions

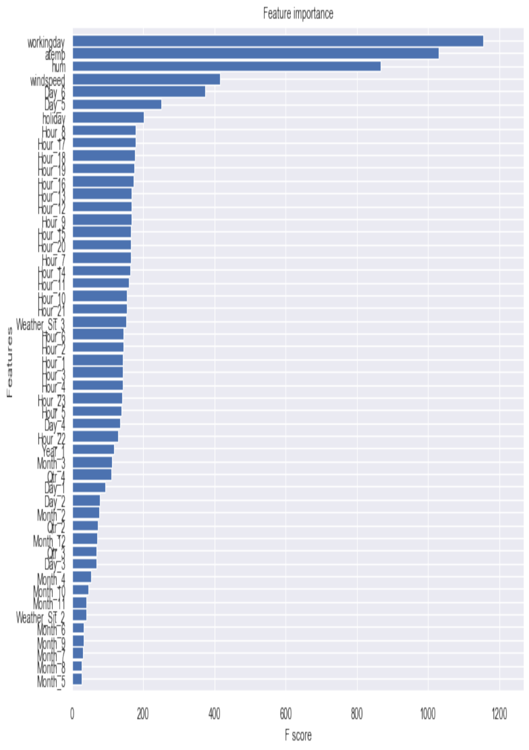

Figure A.6: XGBoost is able to learn from a variety of time variables and environmental conditions to improve its prediction accuracy.

Bibliography

[1] Stan Hatko et al. *The Bank of Canada 2015 Retailer Survey on the Cost of Payment Methods: Nonresponse.* Bank of Canada= Banque du Canada, 2017.

[2] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice.* OTexts, 2018.

[3] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

[4] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition. *Journal of official statistics*, 6(1):3–73, 1990.

[5] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.