Metro Interstate Traffic Volume Project

Josef Pishek and Chen Lou

MA5790

December 19, 2019

1. Abstract

Automatic traffic recorders (ATRs) are a vital municipal tool to model vehicle traffic volume along critical metro area interstates highways and roads. Traffic volume prediction is the objective for a dataset from the UCI Machine Learning Repository. The hourly data from ATR can be preprocessed alongside weather features to predict both a continuous and classification response variable, traffic volume. After chronological splitting into training and testing sets, several non-linear models were fit and tuned using rolling origin forecast resampling methods over a range of tuning parameters. Then, the optimal models were used to predict on a test set, and compared for predictive ability. K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) models performed the best for regression, while Neural Networks (NNET) and SVM models performed the best for classification.

Table of Contents

2. Background

An automatic traffic recorder (ATR) is a popular method for collecting data on traffic volume in metro areas. Permanent installations with varying levels of technology are being installed in large cities like Minneapolis and St. Paul to continuously monitor traffic volume and additional types of data, depending upon their equipment and sensors. This project's data is from one of 70+ active devices in Minnesota (30+ in the seven-county metro area and 35+ in greater Minnesota) [1]. More information about Traffic Forecasting and Analysis from the Minnesota Department of Transportation (MNDoT) is available online.

Our objective is to predict the response variable "traffic_volume" from a collection of numerical and categorical predictors. We will preprocess data from the UCI Machine Learning Repository dataset of hourly Minneapolis-St Paul, MN traffic volume for westbound I-94, including weather and holiday features from 2012-2018 [5]. We will fit and evaluate both regression and classification predictive models to explore the capabilities of non-linear models on time-series traffic volume data.

3. Variable Introduction and Definition

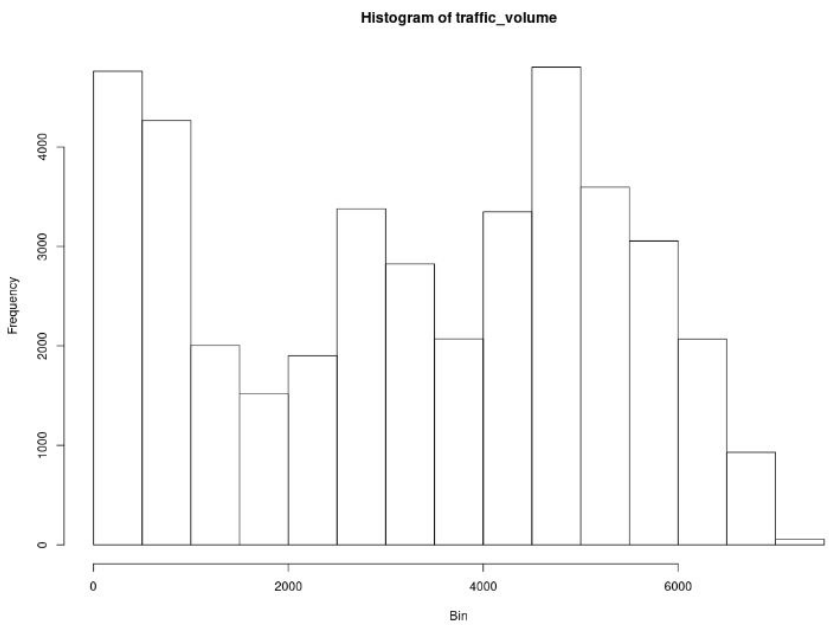
The initial Metro_Interstate_Traffic_Volume.csv file contained the 8 predictors and 1 response with 48,204 observations from MNDoT ATR station 301, located roughly midway between Minneapolis and St Paul, MN. The UCI kindly defined the variables shown in the table below [5].

Variable	Type	Description
holiday	Categorical	(12 levels) Categorical US National holidays plus regional holiday, Minnesota State Fair
temp	Numerical	Numeric Average temp in kelvin
rain_1h	Numerical	Numeric Amount in mm of rain that occurred in the hour
snow_1h	Numerical	Numeric Amount in mm of snow that occurred in the hour
clouds_all	Numerical	Numeric Percentage of cloud cover
weather_main	Categorical	(11 levels) Categorical Short textual description of the current weather
weather_description	Categorical	(38 levels) Categorical Longer textual description of the current weather
date_time	DateTime	DateTime Hour of the data collected in local CST time

traffic_volume (response)	Numerical	Numeric Hourly I-94 ATR 301 reported westbound traffic volume

Table 1. Metro Interstate Traffic Volume from the UCI Machine Learning Repository.

We felt that it would be important to train both regression and classification models to traffic data because of the diverse consumer of such predictions. We might have everyday commuters that simply care if the traffic will be heavy, normal, or light when they check their phones before their commute home from work. On the other hand, we might have a municipal customer that is interested in more specific volumes and making decisions based on continuous traffic estimations on the same roads that those commuters travel. The need to try classification became even more apparent after looking at the regression response variable below.



	traffic_volume
mean	3290.6505
median	3427
sd	1984.7729
0%	0
25%	1248.5
50%	3427
75%	4952
100%	7280
IQR	3703.5
skewness	-0.1074

Figure 1. Histogram and Summary of Regression Response Variable

The distribution of the response is certainly not normal, but it is not necessarily “skewed” in the typical sense, either. We decided that The distribution seemed to show three naturally occurring classes within itself. The illustration of how we split the categorical response from the regression response is shown in the figure below.

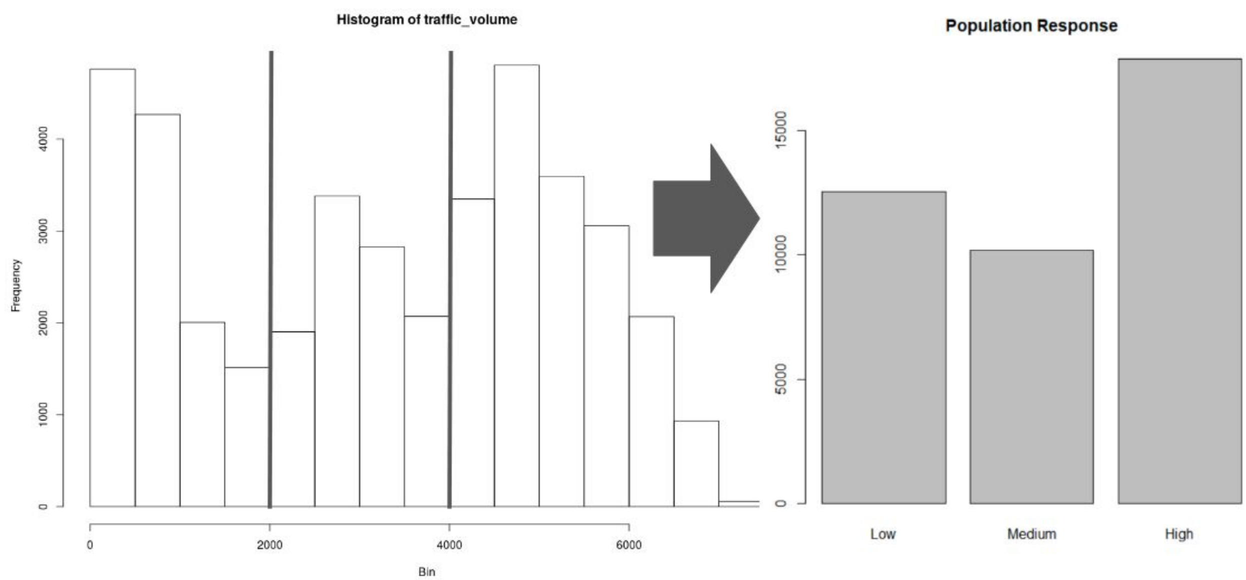


Figure 2. Binning of Classification Response Variable

Bin endpoints of 2,000 and 4,000 seemed to capture the three distinctive humps, low, medium and high traffic volume. And although the frequency distribution of the classification response is not perfectly balanced, it was sufficient for this analysis, and we will discuss further binning ideas in the summary.

4. Preprocessing

The preprocessing used a combination of visualization and thresholds to prepare our 8 initial variables for modeling. Detailed graphics can be found in the Appendix, while the full code can be found in the R Code. The high level steps were as follows:

1. Filter Duplicates
2. Remove “near-zero variance” predictors
3. Remove 10 observations for bad ‘temp’ values
4. Remove uninformative predictors
5. Dummy Variables
6. Remove “near-zero variance” predictors again
7. Remove “highly-correlated” predictors
8. Center and Scale as part of model training

The original data contained a large number of duplicate observations; therefore, we filtered them out and down-sized the data from 48,204 observations to 40,575 observations. These observations could have been added by the data compiler as a mock ‘testing’ set. We maintained a significant amount of the original observations and ensured that no observations were repeated in the date and time sequence.

We also suspected that there were 10 observations under the ‘temp’ predictor that did not make sense, since the temperature in Minnesota had never reached negative 273 degrees celsius. We eliminated these 10 observations to reach 40,565. Then, by applying the caret function “near-zero variance” on the predictors, we found that predictors ‘holiday’, ‘rain_1h’, ‘snow_1h’ were highly degenerate, and subsequently removed.

There is one predictor called ‘date_time’, which is a string-like predictor that contains the information of the year, day, month, and hour. By applying the lubridate package in R, we split them all as individual predictors. For example, one predictor as ‘year’, another predictor as ‘month’, ‘day’ of the week, and ‘hour’. At this point, the majority of our predictors in this dataset are categorical, so we transformed these predictors into binary representations using dummy variables. As a result, we ended up having a dataset with 40,565 observations and 98 predictors.

As we began fitting and tuning our initial models, we took further investigations into our predictors. Our results were far from what we would expect. We realized that we needed to reduce noise, and remove near-zero variance dummy predictors, and uninformative predictors. Predictors ‘year’ and ‘month’ have no variation with the response. The box plot shown below is a visual representation of this.

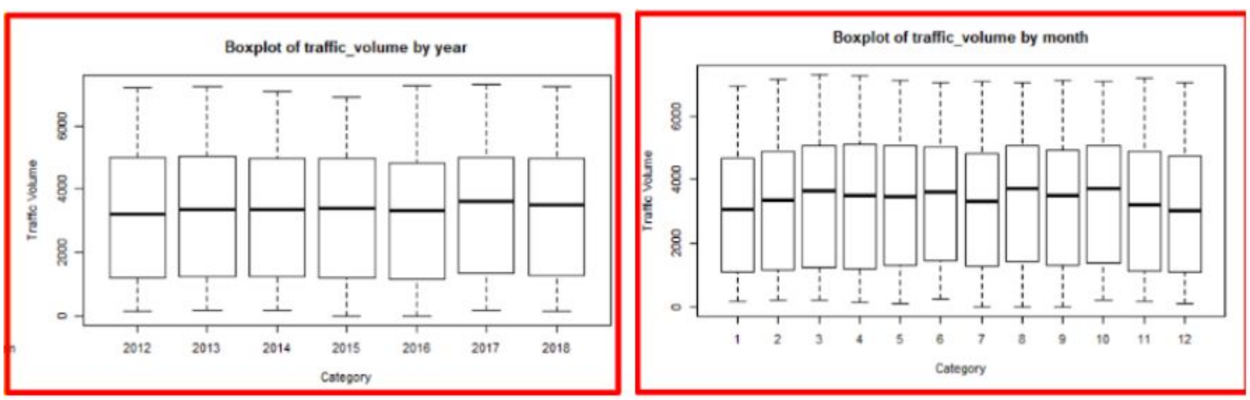


Figure 3. Uninformative Predictors Removed.

We further realized that predictors ‘weather_main’ and ‘weather_description’ describe similar information, so we removed ‘weather_description’ to avoid correlation and noise. We filtered correlated predictors to remove ‘clouds_all’ and ‘weather_main_Clear’ which are obviously negatively correlated. The algorithm kept the ‘weather_main_Clouds’ predictor to retain this information. We will also center and scale all predictors as part of our training control. So we finally have 40,565 observations and 36 predictors to split into training and test sets. The final predictors are essentially ‘temp’, ‘day’ (1-7), ‘hour’ (0-23), and ‘weather_main_Clouds’, ‘weather_main_Mist’, ‘weather_main_Rain’, and ‘weather_main_Snow’ to predict ‘traffic_volume’.

5. Data Splitting and Training Resampling

The first 80% of observations will be used for training, and the last 20% will be held-out for testing. This was done to maintain chronological order in the dataset sequence. We found that random splitting produces poor results for time-series data, and that a time-series method is a more realistic predictive approach. So we will essentially use traffic volume and associated predictors from 2012-2017 to predict traffic volume in 2018. Data has been split into the following dimensions:

- trainX has 32,452 observations, and 36 predictors
- trainY has 32,452 observations
- testX has 8,113 observations, and 36 predictors
- testY has 8,113 observations

We can check to make sure our splitting method did not create major differences between our training response and testing response. See the figure below and confirm the distributions.

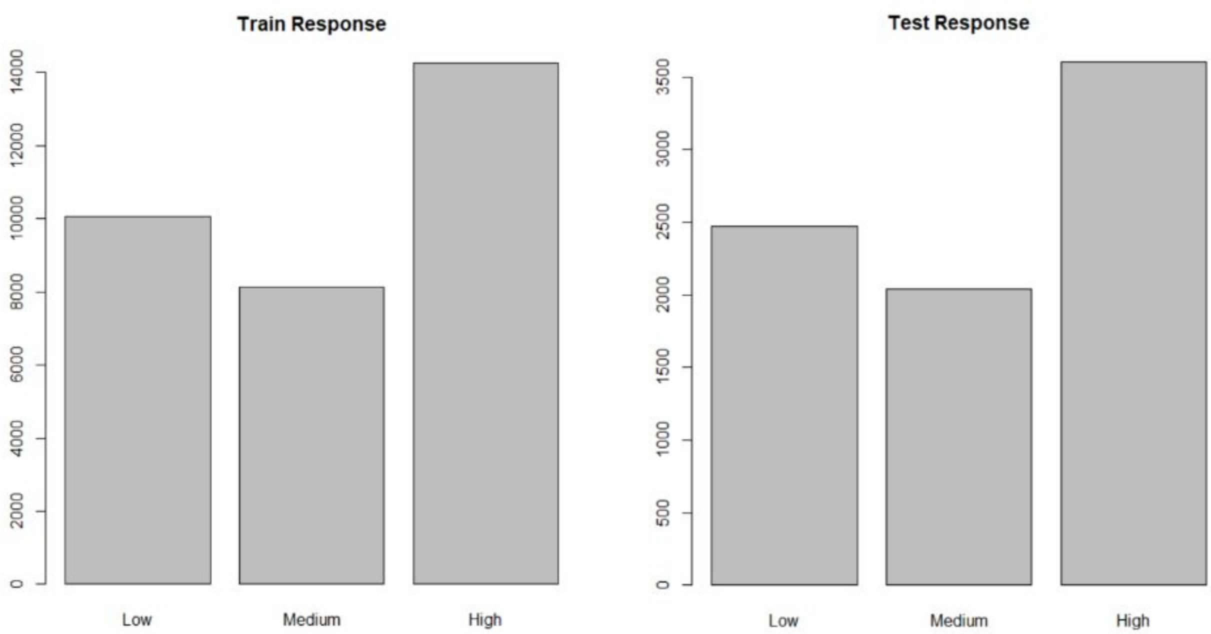


Figure 4. Training and Testing Response Frequency Distributions after 80/20 Train/Test Split

For training resampling, we used the rolling forecasting origin, “timeslice” method with the following parameters:

- fixedWindow = TRUE,
- horizon = 168,
- initialWindow = 672,
- skip = 167

We chose this method and parameters after reading documentation from caret and associated sources about how to best fit predictive models when the data is in time-series sequence [2][3]. The method we chose seeks to replicate the 80/20 split within the training set, except in much smaller, computationally efficient chunks. The figure below should illustrate the method.

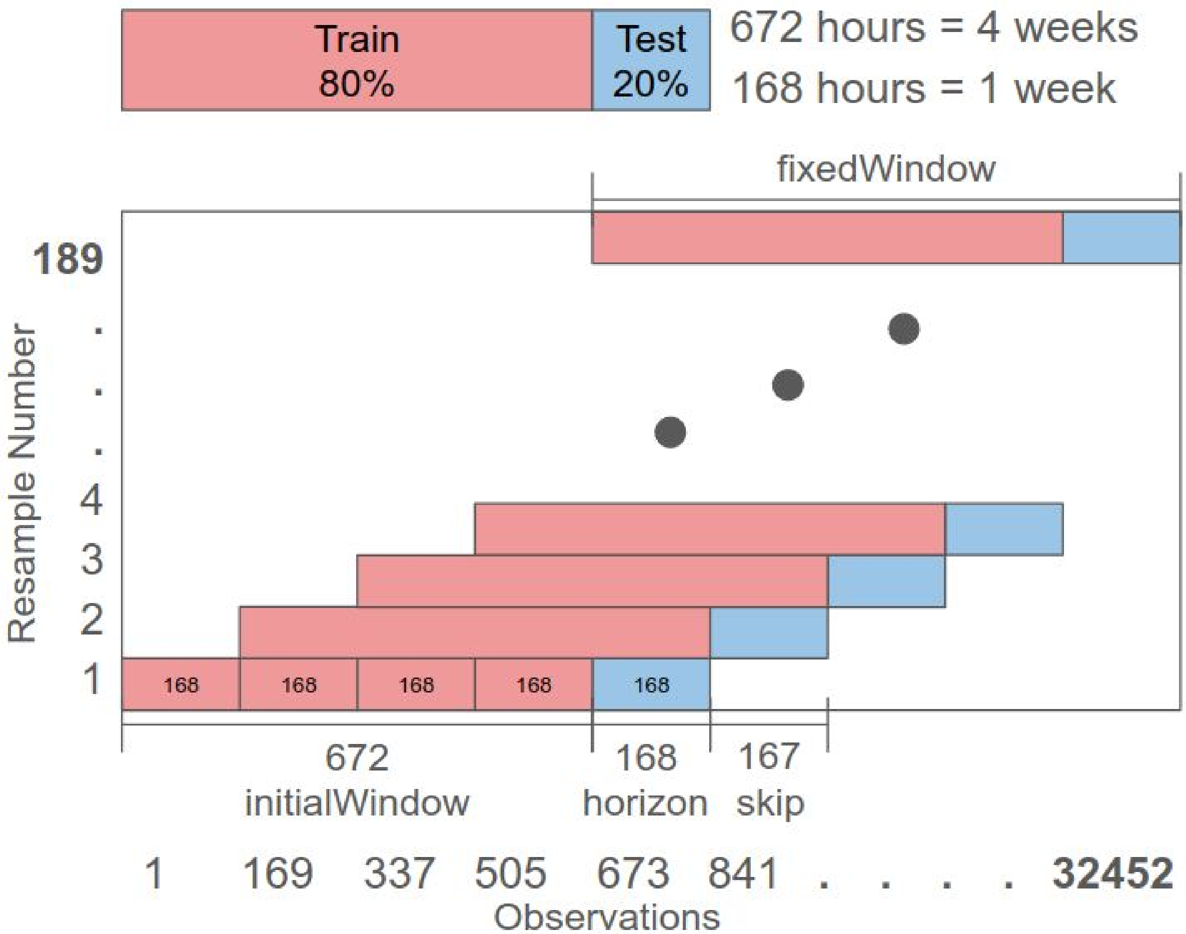


Figure 5. Resampling Method Illustration for Training Control

Using an initial window of 4 weeks (672 hours) we will train, and then test on 1 week (168 hours), before the moving window (fixedWindow=False) skips to the next week to predict on. For the 32,452 training observations, we will resample 189 times to train each model and parameters. This may seem like it would take a long time, but because the resampled train test set is so small, “timeslice” is much faster than training much larger folded or bootstrapped models, with even better training predictions. All models, classification and regression were split and trained in the same way, just with a different response.

6. Model Tuning and Selection

As we began to train and test models, it became abundantly clear that the nature of our data was better suited to non-linear models, rather than linear models. For this reason, we will not cover them in this report. For more information on training partial least squares regression, elastic net regression, and least angle regression spline models see the R Code section.

Using our training data and rolling forecasting origin resampling method we fit and tuned the following non-linear models, and predicted each response on the appropriate testing data:

- Regression Models
  - Neural Network (NNET)
  - Multivariate Adaptive Regression Spline (MARS)
  - Support Vector Machine (SVM)
  - K-Nearest Neighbor (KNN)
- Classification Models
  - K-Nearest Neighbor (KNN)
  - Flexible Discriminant Analysis (FDA)
  - Neural Network (NNET)
  - Support Vector Machine (SVM)

The results for each type of model are summarized in the subsections below, while full tuning plots, model fit summaries, and optimal tuning parameters are available in the Appendix for the eight models discussed.

Regression Model Results

For regression, we chose to minimize root mean square error (RMSE) to find the optimal model in training (not to say we don't want to see a high coefficient of determination, R^2). The results for training and testing are summarized in the table below.

Model	Train RMSE	Train R^2	Train Time (sec)	Test RMSE	Test R^2
NNET	673.14	0.89	13797	569.76	0.92
MARS	617.58	0.90	843	548.61	0.92
SVM	820.23	0.84	2412	476.93	0.94
KNN	584.25	0.91	57	544.60	0.92

Table 2. Regression Model Performance

We can see that KNN (K = 3) performed the best in training, while SVM (sigma = 0.01136232 and C = 512) had the best predictive ability on the testing set. MARS (nprune = 36 and degree = 2) is not far behind either, while NNET (size = 6, decay = 8 and bag = FALSE) seems to lag behind. SVM may have performed better, but KNN had similar performance in a fraction of the computational time. Additionally, SVM is inherently not parallelizable, while KNN must load the entire model so its predictions typically take a little longer. Training and testing combined, however, KNN far surpasses the rest of the models. We should continue to investigate the fit by looking at the testing predicted versus observed charts for each model below.

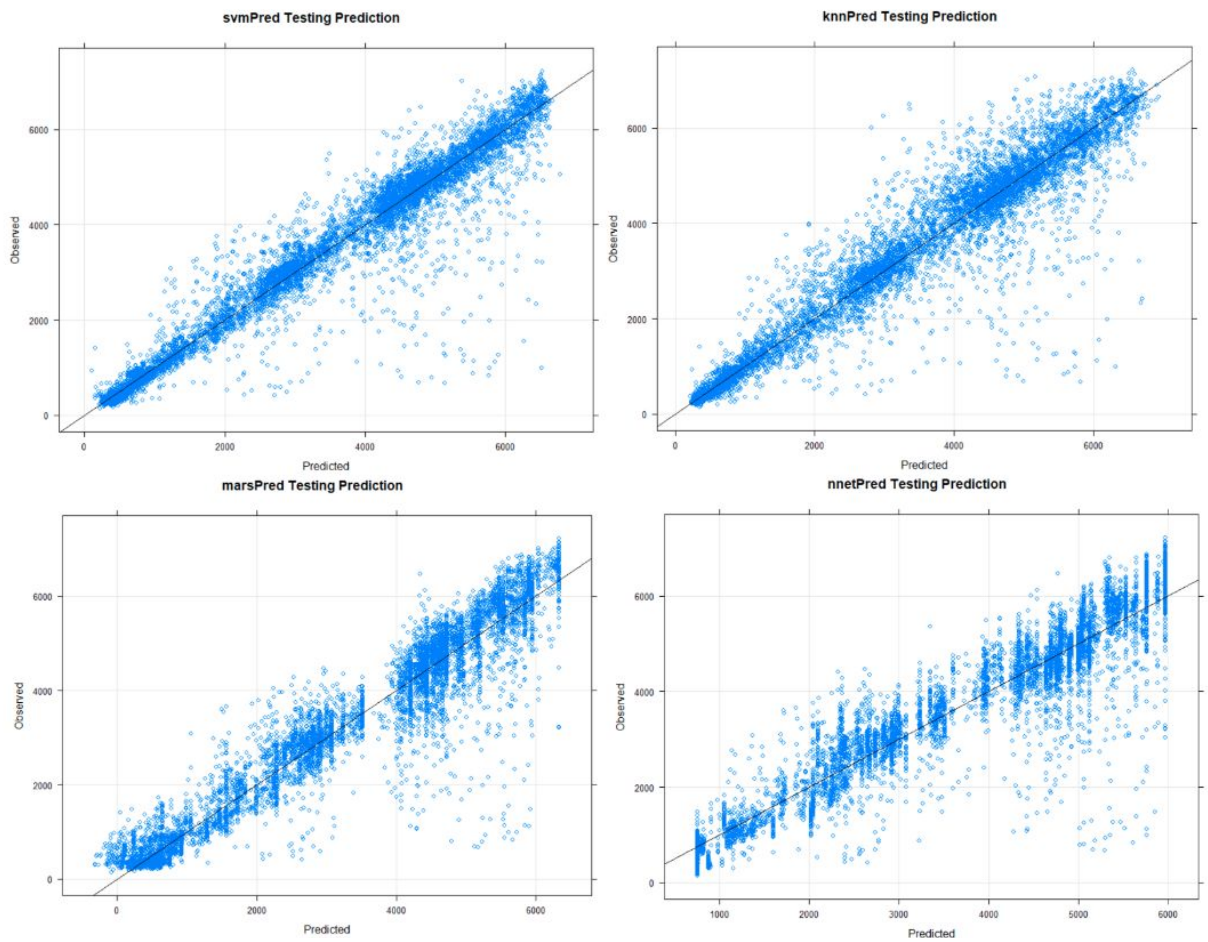


Figure 6. Testing Predicted Versus Observed Plot for each Regression Model

We can see from the plots that SVM and KNN (top) have much better looking residuals than MARS and NNET (bottom). Each model has several observations in the lower right corner, suggesting that we overpredicted low observations, more than we underpredicted high observations. We can look at the testing predictions alongside our actual test response for a single week in the figure below.

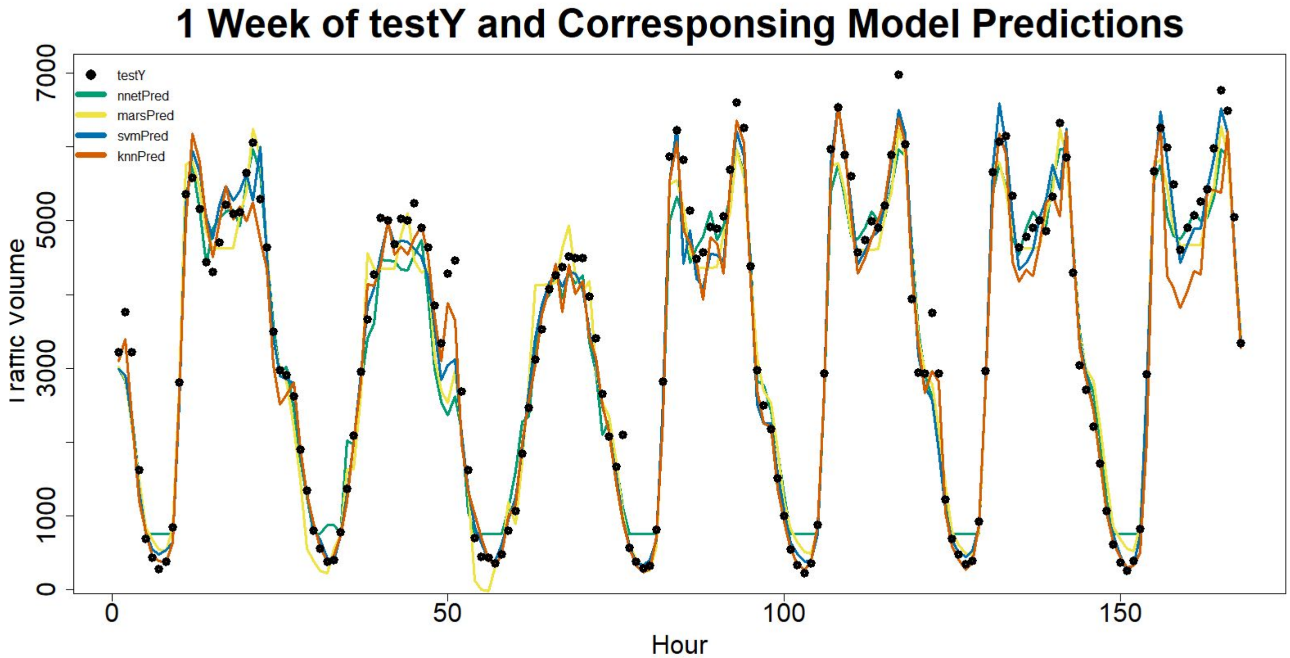


Figure 7. Hours 0 to 168 of Test Predictions and Test Response

We can see in the plot that the blue SVM line and the orange KNN line follow the black response dots better than NNET in green, and MARS in yellow. From this graph, I would say that KNN is the best because the prediction is so similar, for much less computational effort.

There are definite peaks on what looks to be weekdays perhaps before and after working hours. Tuning further does not look like it would help these models, as they already follow the predicted line very well. Perhaps dissolving 'day' of the week into a two-class, 'weekday' or 'weekend' would help reduce noise and improve general prediction. Looking at the variable importance in the appendix shows that the top predictor are a selection of 'hour' and 'day' of the 'week' with 'temp' and a few 'weather_mains' bringing up the middle. It is doubtful that adding predictors back on that were removed in preprocessing would help, but that is certainly another possible action to try and improve RMSE. These regression results are certainly satisfactory, however.

Classification Model Results

We chose the Kappa statistics to evaluate how great the models were fitted. The reason is, our classification model contains 3 levels in the response, the accuracy statistics only work well on a binary classification problem, which would poorly describe how the accuracy breaks down across multiple classes. Table 3 lists all the training and testing results along with the time spent for our classification analysis.

Model	Train AUC	Train Kappa	Train Time (sec)	Test AUC	Test Kappa

KNN	0.9631	0.8430	74	0.9666	0.8799
FDA	0.9611	0.7817	1787	0.9606	0.7834
NNET	0.9727	0.8641	3117	0.9784	0.8828
SVM	0.9695	0.8654	517	0.9656	0.8850

Table 3: Classification Model Performance

According to table 3, we found out the SVM model and NNET model performed the best. However, the NNET model required a large amount of time to train. If we take the time efficiency into account, the KNN model would replace the NNET model. The best performed tuning parameter for those models are:

- KNN: k = 6
- FDA: degree = 1, nprune = 21
- NNET: size = 6, decay = 0.5
- SVM: sigma = 0.011366, cost = 4

Confusion Matrices — **Reference**

Prediction

KNN	L	M	H
L	2337	113	0
M	104	1677	129
H	33	248	3472

FDA	L	M	H
L	1995	87	0
M	276	1425	20
H	203	526	3581

NNET	L	M	H
L	2357	134	0
M	82	1630	85
H	35	274	3516

SVM	L	M	H
L	2355	129	0
M	84	1627	68
H	35	282	3533

Table 4. Confusion Matrices for each set of Test Predictions and Observations

As Figure 8 shown below, this is a traffic volume density plot over a week. We originally classified the traffic volumes below 2000 as low, and above 4000 as high. But, it would be better if we raise the threshold bar higher to fit the overall traffic volume distribution. The result is to be further explored.

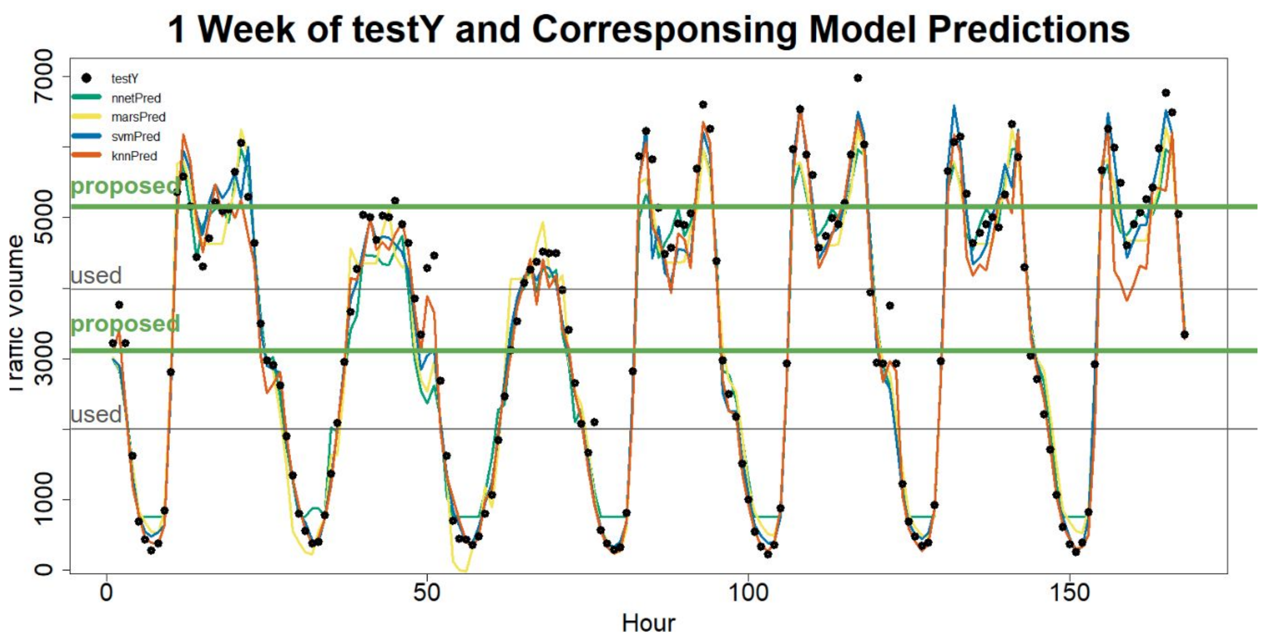


Figure 8. Proposed Categorical Response Bins

7. Summary

The results attained in both classification and regression are promising for the methods used to build predictive models. An excellent next step would be to take these preprocessing and modeling methods and apply them on a   , new dataset. How could we apply this to a network of ATRs to provide predictive results in a production environment? Hopefully, neither of these models are over-tuned, or under-tuned, and would be able to easily plug and play on new data.

8. Appendix

Preprocessing

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
11612	Martin Luther King Jr Day	271.79	0	0	64	Clouds	broken clouds	2014-01-20 00:00:00	480
30081	Martin Luther King Jr Day	266.08	0	0	1	Mist	mist	2017-01-16 00:00:00	698
30082	Martin Luther King Jr Day	266.08	0	0	1	Haze	haze	2017-01-16 00:00:00	698
40656	Martin Luther King Jr Day	262.54	0	0	90	Snow	light snow	2018-01-15 00:00:00	600
40657	Martin Luther King Jr Day	262.54	0	0	90	Mist	mist	2018-01-15 00:00:00	600
40658	Martin Luther King Jr Day	262.54	0	0	90	Haze	haze	2018-01-15 00:00:00	600

Figure 9. Example of Duplicate Observations

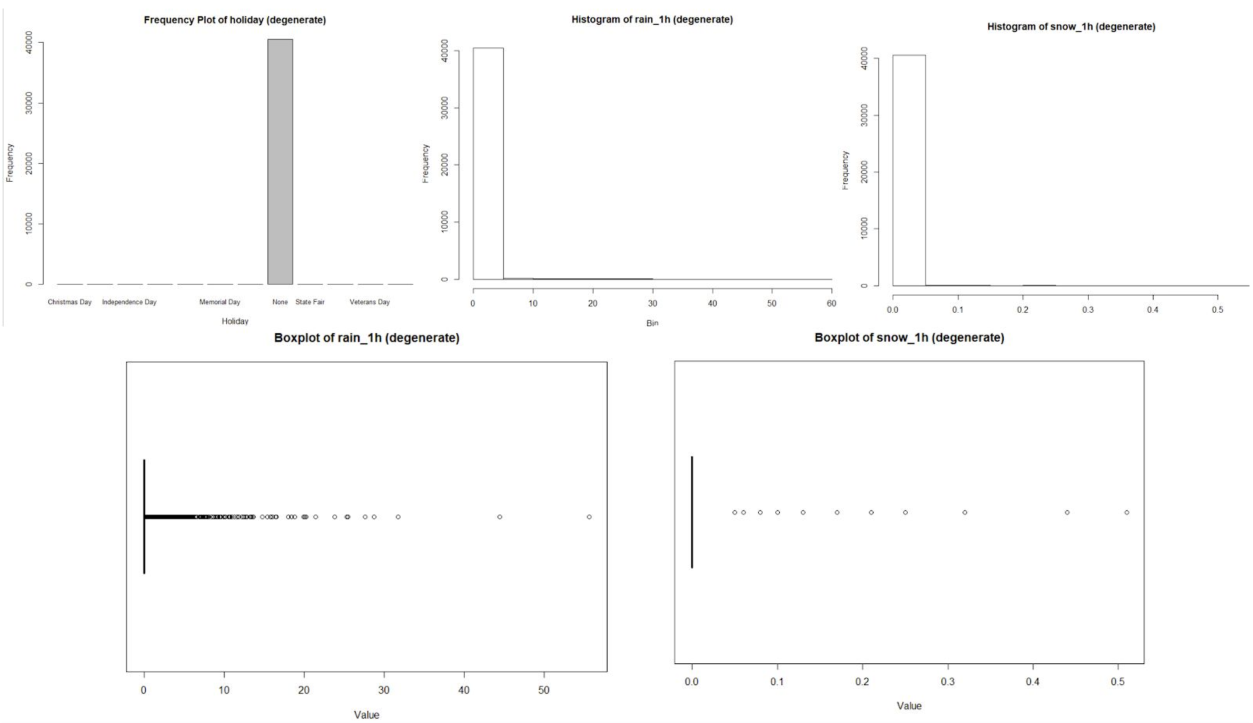


Figure 10. Predictors Removed for “Near-zero Variance”

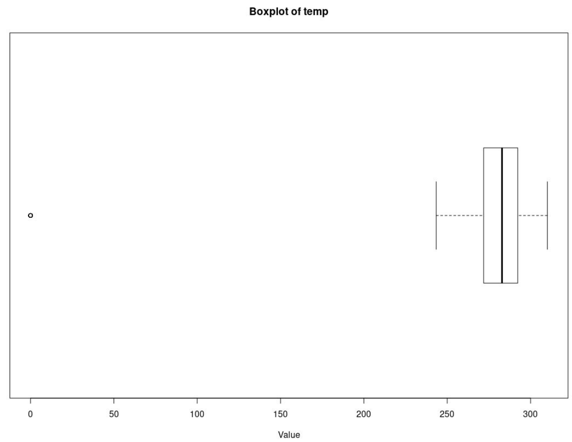
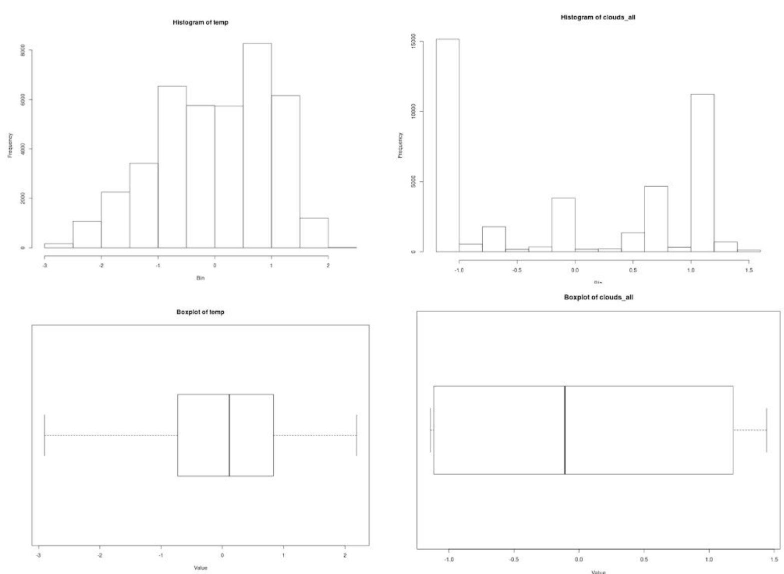


Figure 11. Unreasonable Values in ‘temp’ Predictor



	temp	clouds_all
mean	5e-04	0
median	0.1133	-0.1086
sd	1.0003	1
0%	-2.902	-1.1426
25%	-0.7283	-1.1167
50%	0.1133	-0.1086
75%	0.8328	1.184
100%	2.1907	1.4425
IQR	1.5611	2.3007
skewness	-0.3818	0.0296

temp clouds_all

Figure 12. Numerical Predictor Summaries

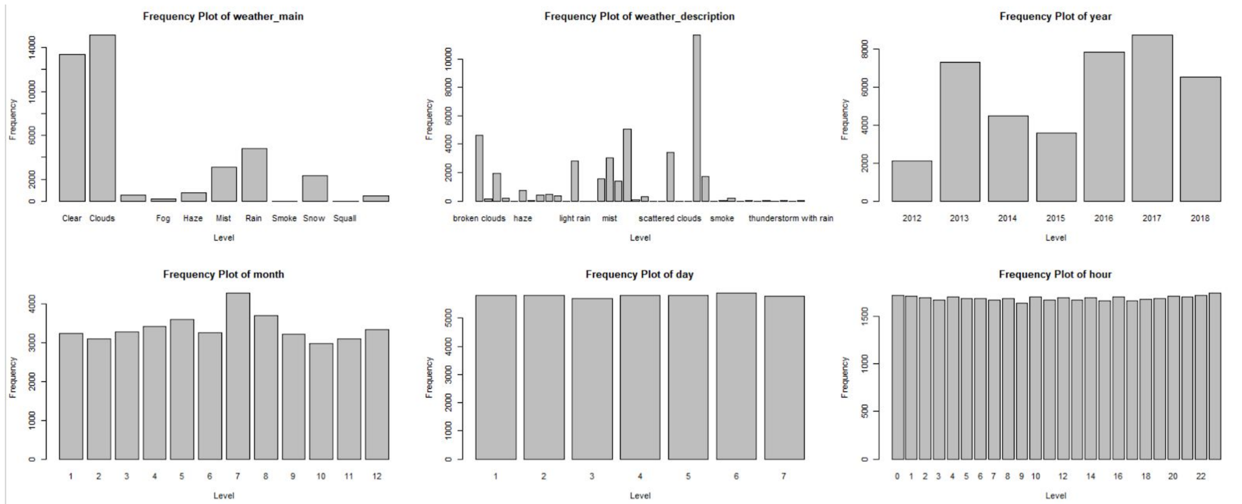


Figure 13. Categorical Predictor Frequency Plots. Notice degenerates in both weather predictors.

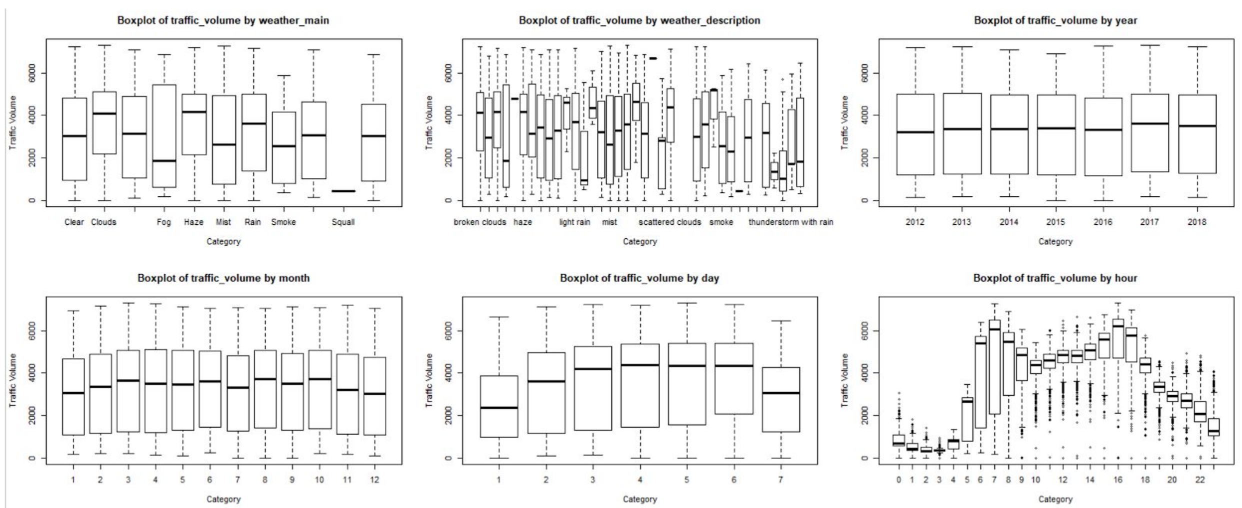


Figure 14. Categorical Predictor Box Plots Split by Response. Notice uninformative ‘year’ and ‘month’.

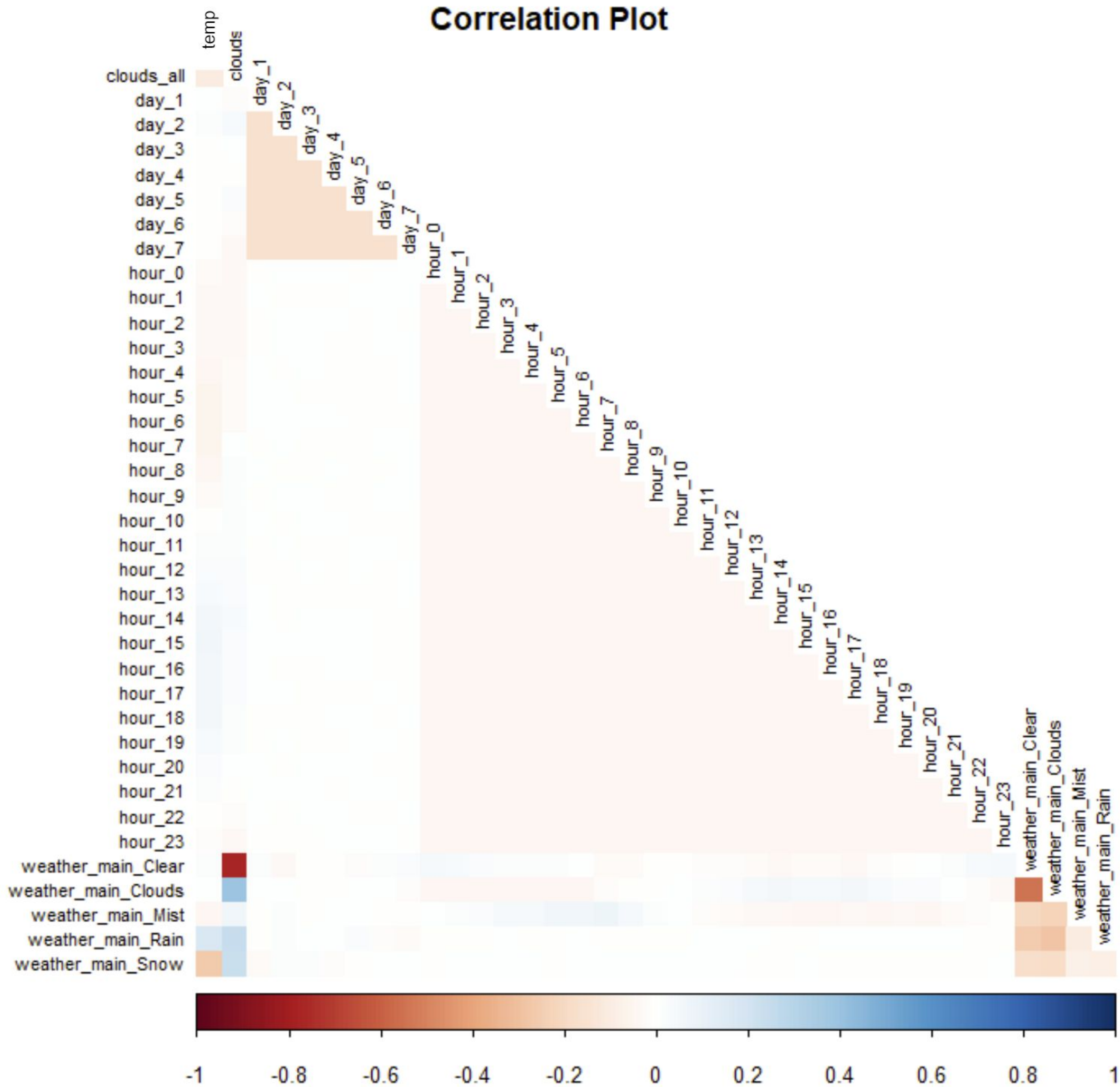


Figure 15. 38 Predictors Before Filtering for Highly Correlated Predictors.

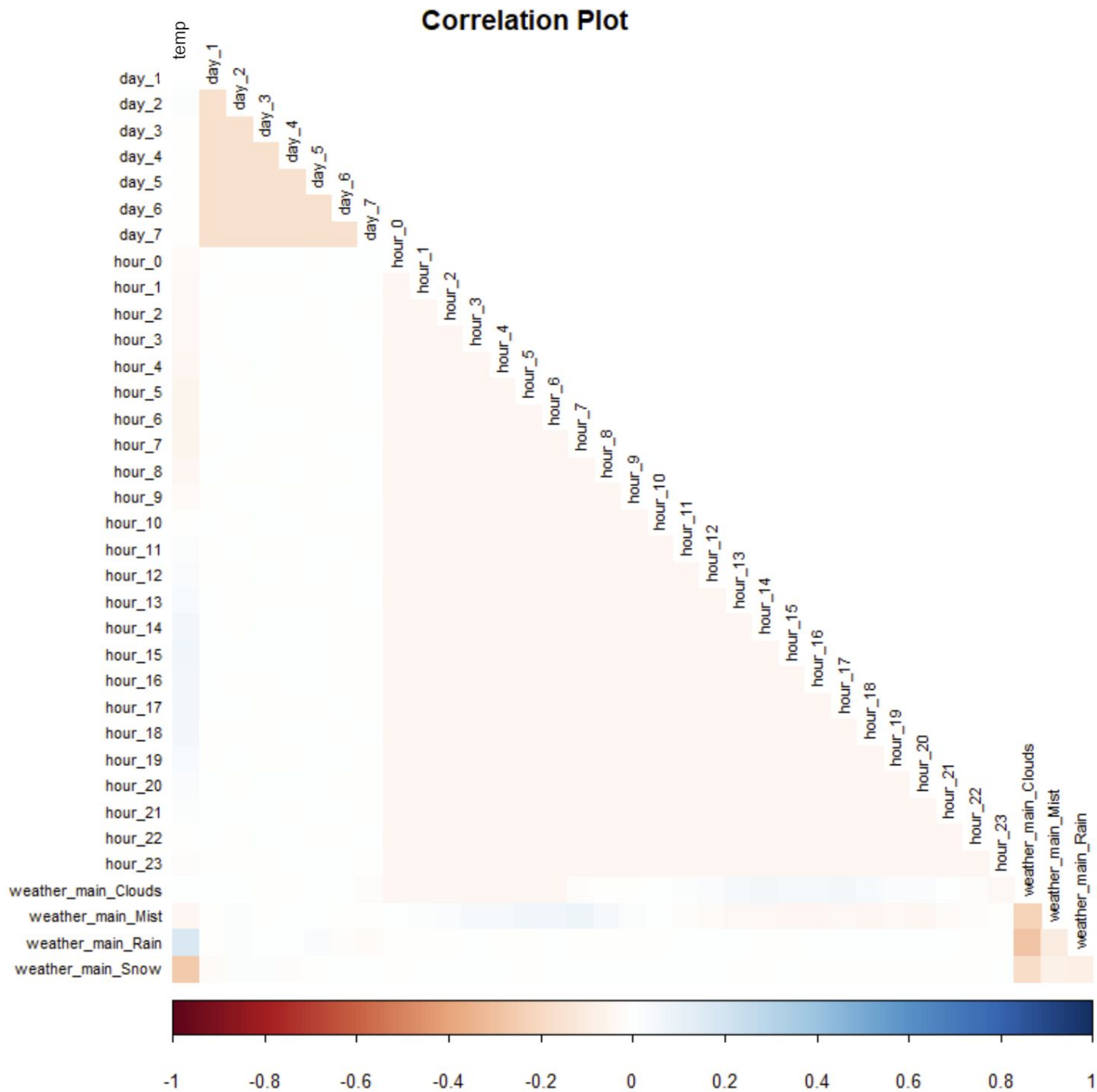
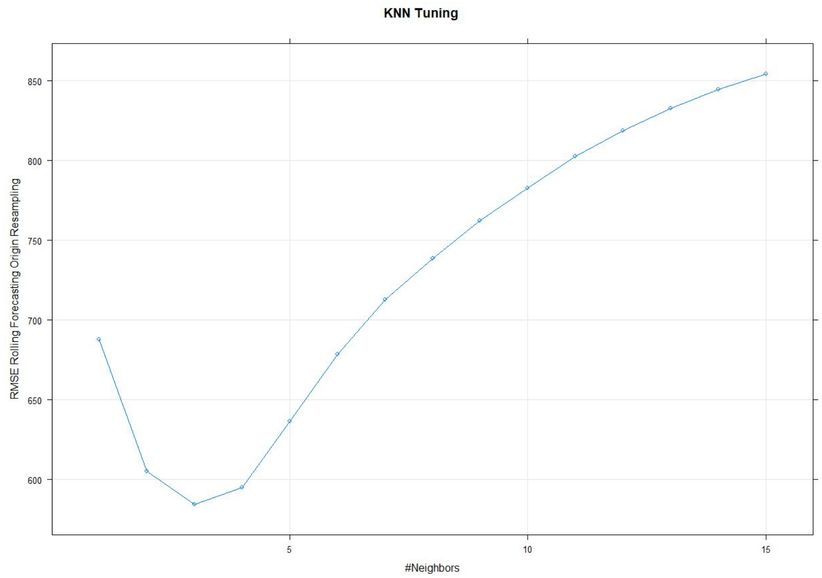


Figure 16. 36 Predictors After Filtering for Highly Correlated Predictors.

Regression Models

A. KNN Model Tuning:

Optimal model parameters using k =3



Fit Summary

k-Nearest Neighbors

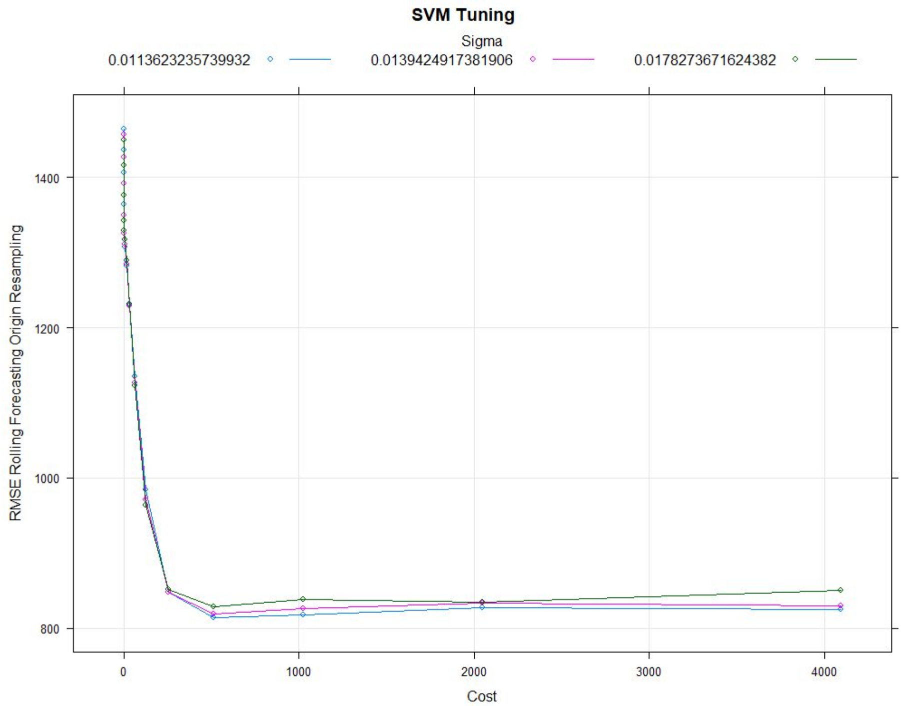
32452 samples
36 predictor

Pre-processing: centered (36), scaled (36)
Resampling: Rolling Forecasting Origin Resampling (168 held-out with a fixed window)
Summary of sample sizes: 672, 672, 672, 672, 672, 672, ...
Resampling results across tuning parameters:

k	RMSE	Rsquared	MAE
1	687.6352	0.8765053	381.1908
2	605.2612	0.9023880	353.8062
3	584.2548	0.9096409	348.6127
4	594.7507	0.9078032	363.9458
5	636.3525	0.8980972	403.6100
6	678.3299	0.8866051	439.6416
7	712.5787	0.8756902	467.4562
8	738.3807	0.8666732	489.9183
9	762.1219	0.8580774	508.4747
10	782.7342	0.8501034	524.9070
11	802.4574	0.8419946	539.4306
12	818.5165	0.8352868	551.6481
13	832.7596	0.8293353	562.2092
14	844.5620	0.8243107	571.3017
15	854.3385	0.8202251	579.2358

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 3.

B. SVM Model Tuning:
Optimal model parameters using sigma=0.01136, Cost=512



Fit Summary
Support Vector Machines with Radial Basis Function Kernel

32452 samples
36 predictor

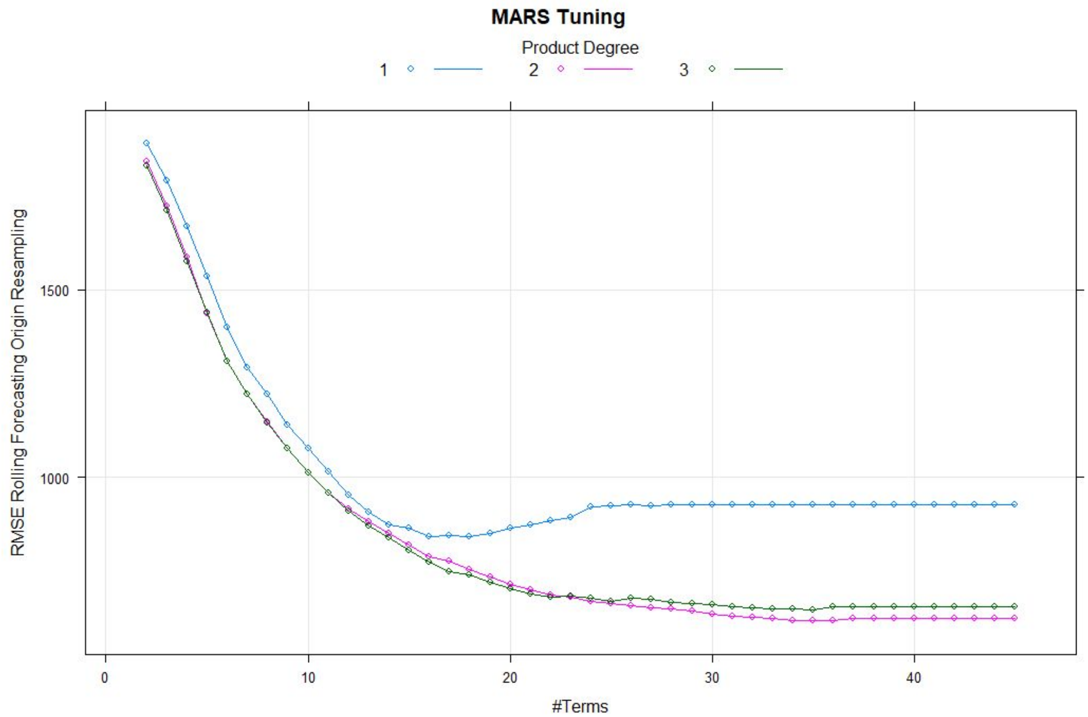
Pre-processing: centered (36), scaled (36)
Resampling: Rolling Forecasting Origin Resampling (168 held-out with a fixed window)
Summary of sample sizes: 672, 672, 672, 672, 672, 672, ...
Resampling results across tuning parameters:

sigma	C	RMSE	Rsquared	MAE
0.01136232	0.25	1464.4735	0.7501980	1205.3230
0.01136232	0.50	1436.3326	0.7567575	1167.2613
0.01136232	1.00	1405.7662	0.7668309	1142.1048
0.01136232	2.00	1364.0901	0.7789692	1118.0219
0.01136232	4.00	1325.7203	0.7876315	1094.5386
0.01136232	8.00	1307.8098	0.7888303	1076.5705
0.01136232	16.00	1281.8121	0.7899805	1047.8927
0.01136232	32.00	1230.6676	0.7944333	992.6385
0.01136232	64.00	1134.9514	0.8072850	894.5037

0.01136232	128.00	983.6130	0.8285925	751.4524
0.01136232	256.00	847.5982	0.8403365	611.4428
0.01136232	512.00	814.0615	0.8395167	548.0961
0.01136232	1024.00	818.1544	0.8355518	517.6126
0.01136232	2048.00	826.8587	0.8333945	516.7418
0.01136232	4096.00	824.5965	0.8330344	525.3532
0.01394249	0.25	1457.3957	0.7532113	1196.9586
0.01394249	0.50	1427.2881	0.7612139	1161.1685
0.01394249	1.00	1391.8796	0.7724859	1135.4637
0.01394249	2.00	1349.4455	0.7838843	1111.7773
0.01394249	4.00	1325.5129	0.7882306	1093.8179
0.01394249	8.00	1310.5516	0.7883868	1077.1918
0.01394249	16.00	1284.1106	0.7892319	1047.1528
0.01394249	32.00	1229.2714	0.7939755	987.5456
0.01394249	64.00	1126.7811	0.8080166	883.5007
0.01394249	128.00	970.9506	0.8284701	736.9249
0.01394249	256.00	847.9528	0.8379863	605.3303
0.01394249	512.00	818.4071	0.8363741	542.7631
0.01394249	1024.00	826.7255	0.8320300	520.4685
0.01394249	2048.00	833.2397	0.8299862	523.2434
0.01394249	4096.00	830.1281	0.8291564	534.4791
0.01782737	0.25	1449.7998	0.7568522	1189.5372
0.01782737	0.50	1416.0414	0.7664568	1154.8793
0.01782737	1.00	1376.7053	0.7782290	1129.4099
0.01782737	2.00	1341.8960	0.7866399	1108.0432
0.01782737	4.00	1329.4989	0.7877241	1095.9689
0.01782737	8.00	1317.0372	0.7869828	1080.5601
0.01782737	16.00	1289.1092	0.7877869	1048.2731
0.01782737	32.00	1231.6379	0.7926439	985.2892
0.01782737	64.00	1123.0931	0.8077244	876.6688
0.01782737	128.00	964.0597	0.8267581	725.9386
0.01782737	256.00	851.5455	0.8346454	600.1820
0.01782737	512.00	828.3460	0.8315961	540.3077
0.01782737	1024.00	837.8586	0.8273212	526.1222
0.01782737	2048.00	834.2160	0.8275752	529.0601
0.01782737	4096.00	850.3719	0.8220609	545.6880

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were sigma = 0.01136232 and C = 512.

C. MARS Model Tuning:
Optimal model parameters using nprune=36, degree=2



Fit Summary
Multivariate Adaptive Regression Spline

32452 samples
36 predictor

Pre-processing: centered (36), scaled (36)
Resampling: Rolling Forecasting Origin Resampling (168 held-out with a fixed window)
Summary of sample sizes: 672, 672, 672, 672, 672, 672, ...
Resampling results across tuning parameters:

degree	nprune	RMSE	Rsquared	MAE
1	2	1888.7890	0.1163909	1614.6134
1	3	1790.7610	0.1977256	1497.4816
1	4	1668.2641	0.2973433	1359.6602
1	5	1536.6251	0.4027945	1221.2903
1	6	1399.6216	0.5062174	1091.0057
1	7	1292.6342	0.5801247	987.2415
1	8	1222.3685	0.6249750	926.4793
1	9	1140.3363	0.6742511	854.2476
1	10	1076.4442	0.7102899	798.6244
1	11	1014.6407	0.7432850	738.7564
1	12	952.6639	0.7739168	692.0468
1	13	907.1155	0.7956457	657.7298
1	14	873.6661	0.8105913	644.7006
1	15	863.7850	0.8167980	642.8503
1	16	842.1235	0.8245964	628.6313

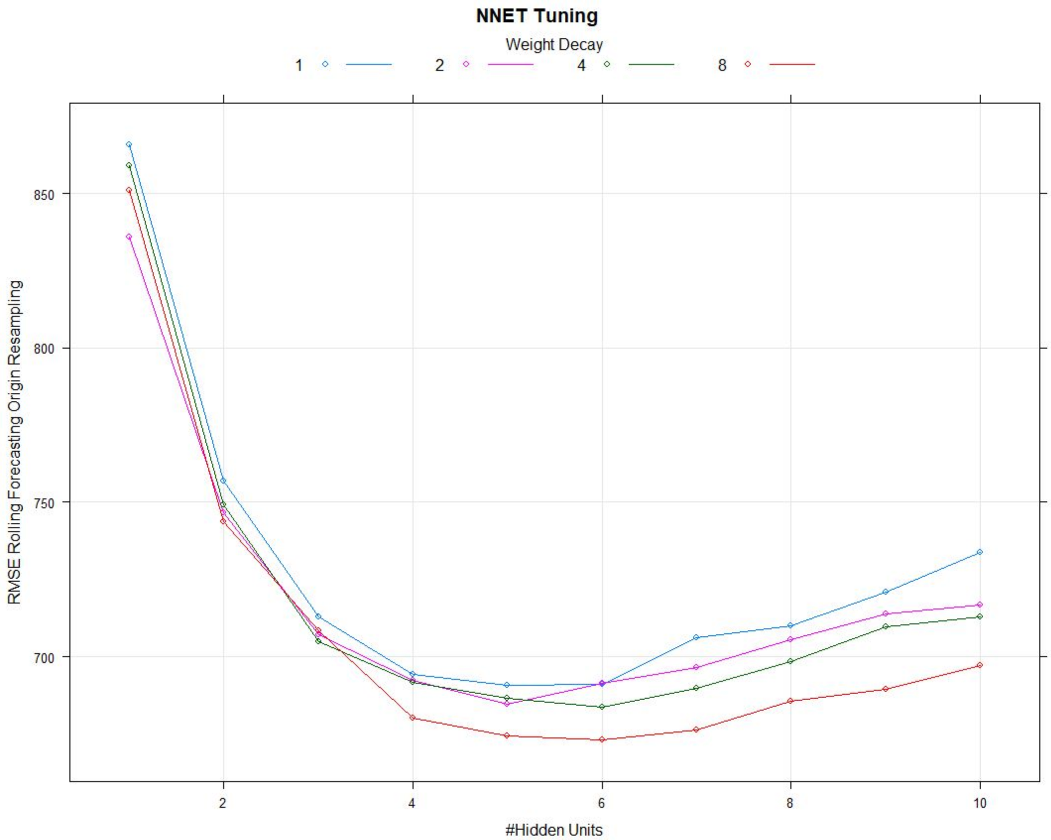
1	17	845.5996	0.8237503	626.6130
1	18	843.6954	0.8248603	623.6283
1	19	850.7222	0.8236342	626.6841
1	20	865.7849	0.8200972	631.8081
1	21	874.5780	0.8172517	634.7043
1	22	885.8060	0.8145583	640.3419
1	23	893.4496	0.8122680	644.3849
1	24	922.0087	0.8078011	665.0628
1	25	923.7664	0.8081119	665.1309
1	26	926.3281	0.8071626	666.2144
1	27	925.3896	0.8072950	665.4372
1	28	927.4255	0.8071303	666.9498
1	29	926.4131	0.8073369	665.8677
1	30	926.8294	0.8072571	666.2527
1	31	926.9474	0.8072381	666.2981
1	32	927.0608	0.8071809	666.3687
1	33	926.9650	0.8072131	666.2909
1	34	926.9669	0.8072133	666.3219
1	35	926.9393	0.8072290	666.3067
1	36	926.9549	0.8072223	666.3098
1	37	926.9549	0.8072223	666.3098
1	38	926.9549	0.8072223	666.3098
1	39	926.9549	0.8072223	666.3098
1	40	926.9549	0.8072223	666.3098
1	41	926.9549	0.8072223	666.3098
1	42	926.9549	0.8072223	666.3098
1	43	926.9549	0.8072223	666.3098
1	44	926.9549	0.8072223	666.3098
1	45	926.9549	0.8072223	666.3098
2	2	1842.5989	0.1455618	1556.1145
2	3	1722.4376	0.2511592	1415.7520
2	4	1587.0711	0.3626469	1273.6100
2	5	1436.8506	0.4765893	1128.3685
2	6	1309.0405	0.5672070	1009.9071
2	7	1220.5558	0.6246520	927.6423
2	8	1147.5633	0.6685239	862.0400
2	9	1077.1351	0.7085415	796.0229
2	10	1013.6026	0.7426703	741.2573
2	11	957.5253	0.7709525	697.0037
2	12	914.9517	0.7913636	667.1445
2	13	881.9205	0.8060107	641.6879
2	14	850.2303	0.8196783	617.5825
2	15	818.5319	0.8324428	596.1706
2	16	787.5313	0.8448720	575.8927
2	17	777.4423	0.8529705	561.0803
2	18	753.4233	0.8619253	545.1134
2	19	733.5541	0.8690547	531.3903
2	20	714.5875	0.8749004	518.7723

2	21	699.6361	0.8799314	508.0701
2	22	686.5132	0.8841756	498.3386
2	23	679.8840	0.8860703	491.4095
2	24	668.9602	0.8895135	480.8420
2	25	663.6946	0.8910734	474.7454
2	26	658.1899	0.8927332	468.8658
2	27	652.9687	0.8943548	463.0366
2	28	648.8631	0.8949637	457.2004
2	29	643.5038	0.8963459	450.8463
2	30	636.9368	0.8980352	445.3320
2	31	630.6084	0.8997842	439.0067
2	32	627.8729	0.9006217	436.4620
2	33	624.3671	0.9015069	432.3405
2	34	619.9736	0.9027047	428.1166
2	35	619.4257	0.9029328	425.5584
2	36	617.5752	0.9033766	424.0558
2	37	624.2265	0.9004390	426.7153
2	38	624.1767	0.9004336	426.7901
2	39	623.8759	0.9004813	426.4074
2	40	623.8759	0.9004813	426.4074
2	41	623.8759	0.9004813	426.4074
2	42	623.8759	0.9004813	426.4074
2	43	623.8759	0.9004813	426.4074
2	44	623.8759	0.9004813	426.4074
2	45	623.8759	0.9004813	426.4074
3	2	1831.1679	0.1555071	1546.8573
3	3	1710.4619	0.2611273	1406.1333
3	4	1574.6478	0.3721510	1263.2216
3	5	1438.4635	0.4769386	1128.6934
3	6	1309.7877	0.5663274	1010.1534
3	7	1222.7755	0.6231225	931.3745
3	8	1146.3955	0.6692766	861.3525
3	9	1076.3475	0.7091750	798.3611
3	10	1012.2131	0.7431645	744.1519
3	11	957.6282	0.7707822	701.2222
3	12	909.5762	0.7935228	665.6387
3	13	869.8434	0.8109327	635.2393
3	14	840.7362	0.8228932	612.4887
3	15	805.9735	0.8370065	587.0529
3	16	774.0852	0.8495240	566.5657
3	17	748.1213	0.8597419	548.6084
3	18	740.0655	0.8655421	536.1506
3	19	720.6384	0.8720745	521.5554
3	20	704.5129	0.8773863	509.3990
3	21	689.9372	0.8823105	498.2038
3	22	679.6285	0.8855730	489.0608
3	23	683.9069	0.8844742	484.9186
3	24	677.4474	0.8865418	476.8198

3	25	670.6504	0.8884209	468.9604
3	26	677.3520	0.8848395	464.5992
3	27	676.4984	0.8851504	459.1291
3	28	666.1942	0.8882962	451.3099
3	29	664.7606	0.8887183	446.5578
3	30	661.3860	0.8891985	442.5705
3	31	656.6872	0.8905651	437.8560
3	32	653.6426	0.8909557	434.9688
3	33	651.0745	0.8915370	431.4463
3	34	650.6110	0.8918704	429.0293
3	35	648.0138	0.8921732	426.6649
3	36	654.5896	0.8900666	428.8259
3	37	655.3774	0.8898167	428.9617
3	38	655.9525	0.8895869	429.0826
3	39	655.7799	0.8895729	428.7433
3	40	655.7799	0.8895729	428.7433
3	41	655.7799	0.8895729	428.7433
3	42	655.7799	0.8895729	428.7433
3	43	655.7799	0.8895729	428.7433
3	44	655.7799	0.8895729	428.7433
3	45	655.7799	0.8895729	428.7433

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were nprune = 36 and degree = 2.

D. NNET Model Tuning:
Optimal model parameters using size=6, decay=8



Fit Summary
Model Averaged Neural Network

32452 samples
36 predictor

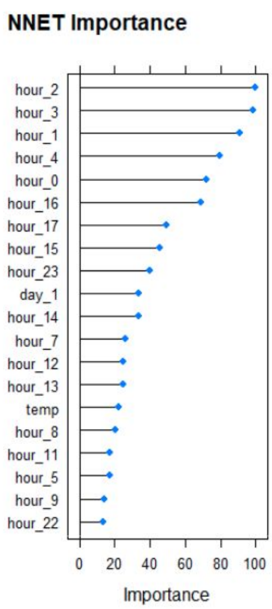
Pre-processing: centered (36), scaled (36)
Resampling: Rolling Forecasting Origin Resampling (168 held-out with a fixed window)
Summary of sample sizes: 672, 672, 672, 672, 672, 672, ...
Resampling results across tuning parameters:

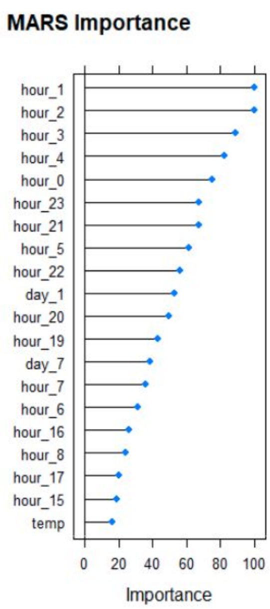
size	decay	RMSE	Rsquared	MAE
1	1	865.7320	0.8268260	695.5474
1	2	835.9969	0.8361610	666.6311
1	4	858.9165	0.8309962	691.6091
1	8	851.0217	0.8347016	681.1698
2	1	756.8913	0.8675805	582.6361
2	2	746.6201	0.8718055	573.8824
2	4	749.0797	0.8712473	576.6520
2	8	743.6155	0.8747629	571.4442
3	1	712.9225	0.8812473	535.9615
3	2	707.0232	0.8829617	531.5490
3	4	704.7605	0.8841095	529.8571
3	8	708.3674	0.8837047	531.9804

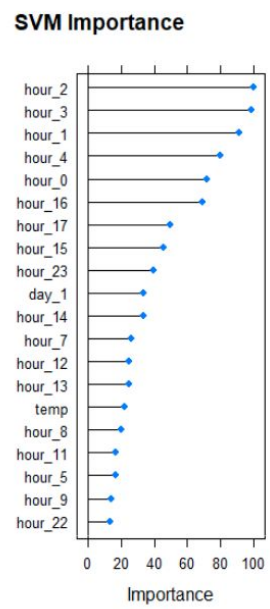
4	1	694.1738	0.8852568	513.7818
4	2	692.4270	0.8851753	513.5405
4	4	691.6395	0.8867727	513.9763
4	8	680.1195	0.8907474	505.1962
5	1	690.6909	0.8849727	507.3950
5	2	684.7087	0.8885246	503.3431
5	4	686.4135	0.8876909	503.2800
5	8	674.3261	0.8917616	496.8621
6	1	691.2241	0.8847433	507.3874
6	2	691.3688	0.8859225	506.0072
6	4	683.7515	0.8883021	499.5676
6	8	673.1382	0.8924923	491.8767
7	1	706.0279	0.8810094	516.1078
7	2	696.3954	0.8835899	510.7229
7	4	689.8542	0.8859442	502.4931
7	8	676.1776	0.8910964	494.2594
8	1	709.9521	0.8793444	519.9551
8	2	705.5874	0.8810627	516.3578
8	4	698.6038	0.8833928	511.4432
8	8	685.7586	0.8882351	501.0593
9	1	721.0865	0.8765603	530.7381
9	2	713.9553	0.8784409	522.6598
9	4	709.7429	0.8807181	521.8485
9	8	689.5571	0.8864036	503.7032
10	1	733.6938	0.8719320	541.0659
10	2	716.6894	0.8777075	526.6443
10	4	712.9077	0.8791903	521.5847
10	8	697.2149	0.8851862	509.8092

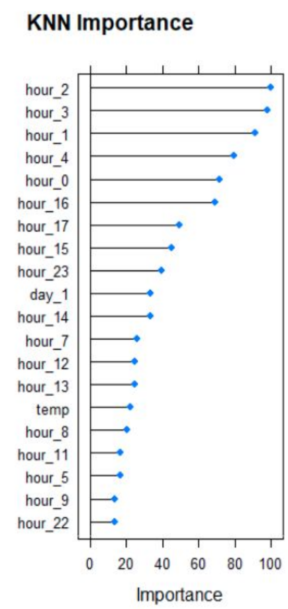
Tuning parameter 'bag' was held constant at a value of FALSE
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 6, decay = 8 and bag = FALSE.

E. Variable Importance:



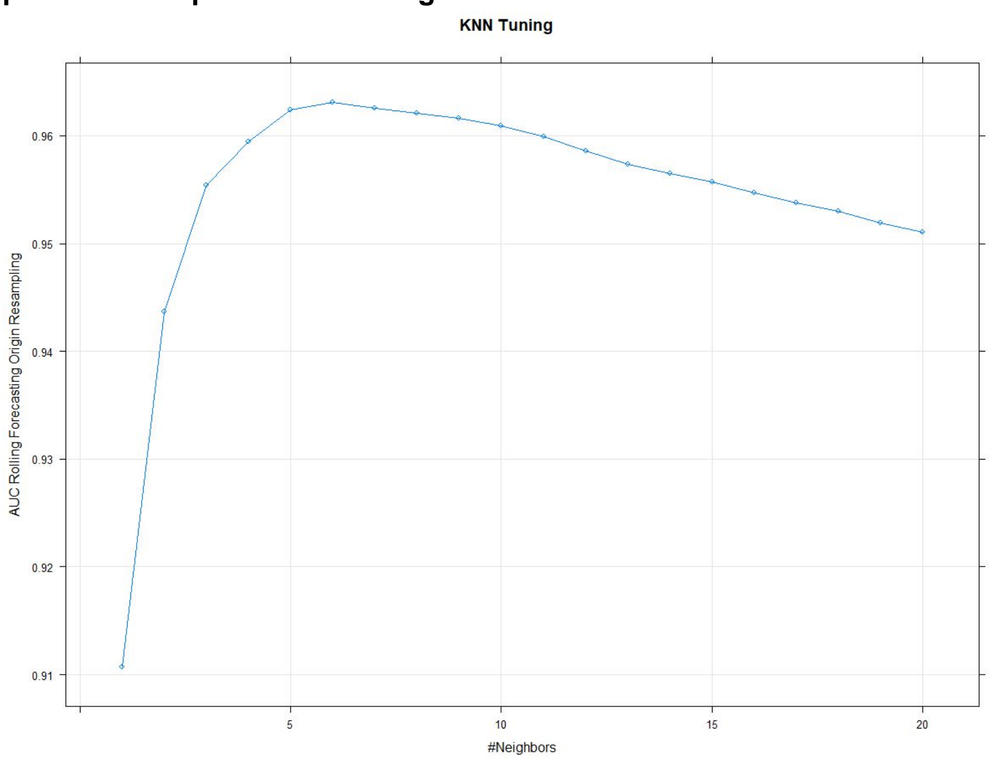






Classification Models

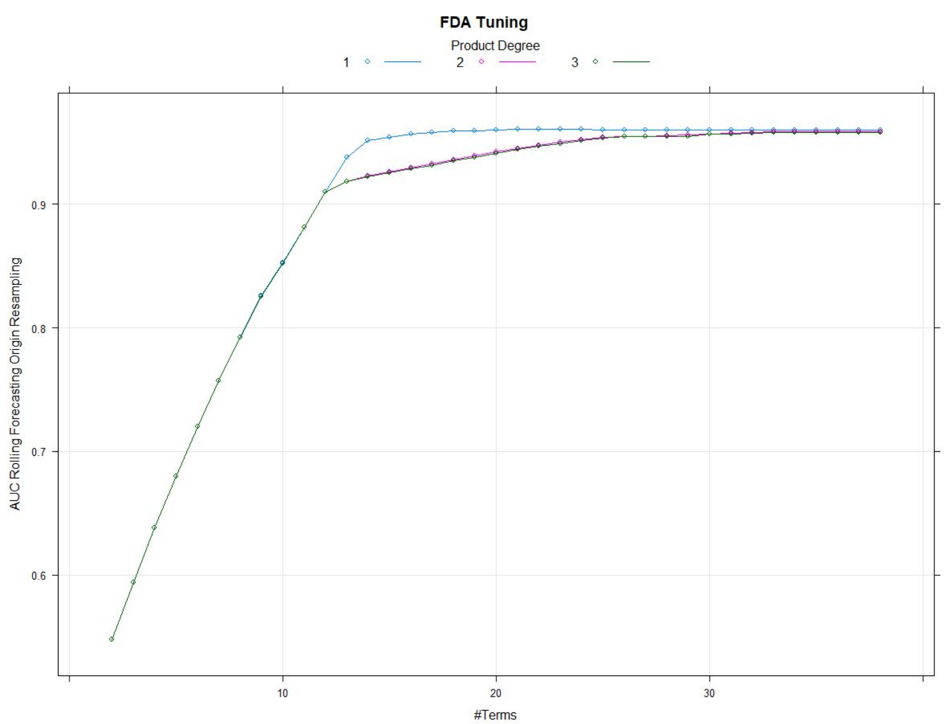
A. KNN Model tuning:
Optimal model parameters using k=6



KNN Tuning Summary:

```
Confusion Matrix and Statistics

          Reference
Prediction  Low Medium High
    Low    2337    113    0
    Medium  104   1677  129
    High     33    248 3472

Overall Statistics

               Accuracy : 0.9227
                 95% CI : (0.9167, 0.9284)
    No Information Rate : 0.4439
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8799

 Mcnemar's Test P-Value : 2.691e-15

Statistics by Class:

AUC was used to select the optimal model using the largest value.
The final value used for the model was k = 6.
```

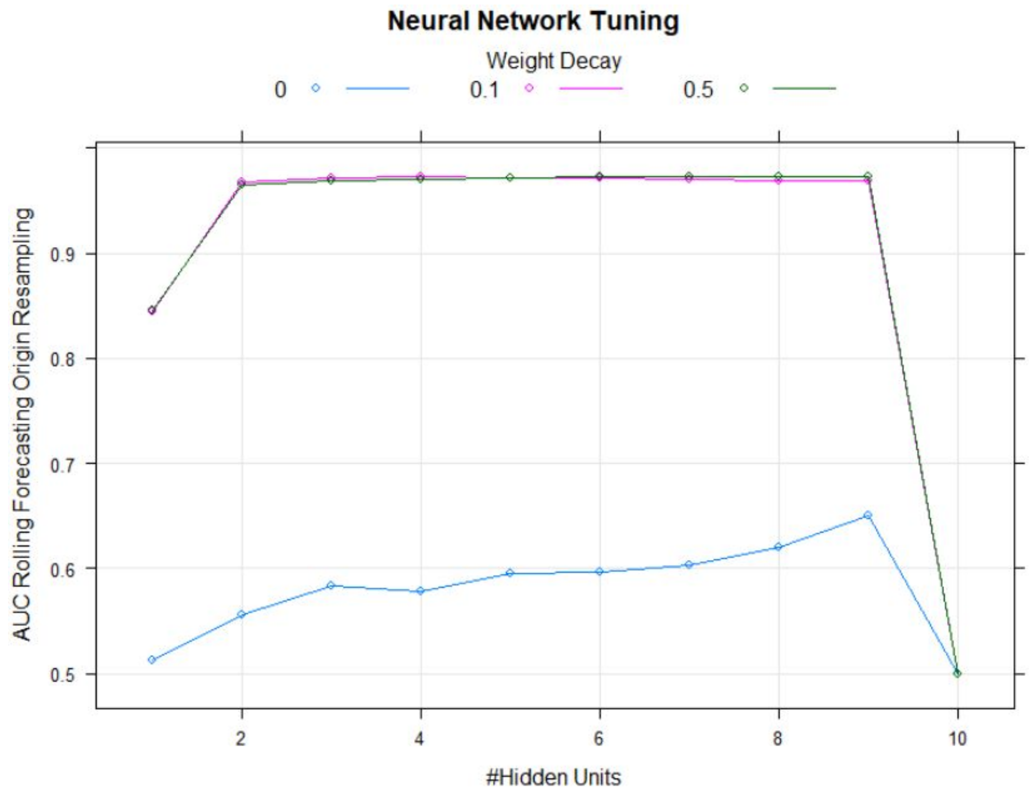
B. FDA Model Tuning:
Optimal model parameters using degree=1, nprune=21



FDA Tuning Summary:

```
Confusion Matrix and Statistics

          Reference
Prediction  Low Medium High
    Low    1995     87    0
    Medium  276   1425   20
    High    203    526 3581

Overall Statistics

               Accuracy : 0.8629
                 95% CI : (0.8553, 0.8703)
    No Information Rate : 0.4439
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7834

 Mcnemar's Test P-Value : < 2.2e-16

AUC was used to select the optimal model using the largest value.
The final values used for the model were degree = 1 and nprune = 21.
```

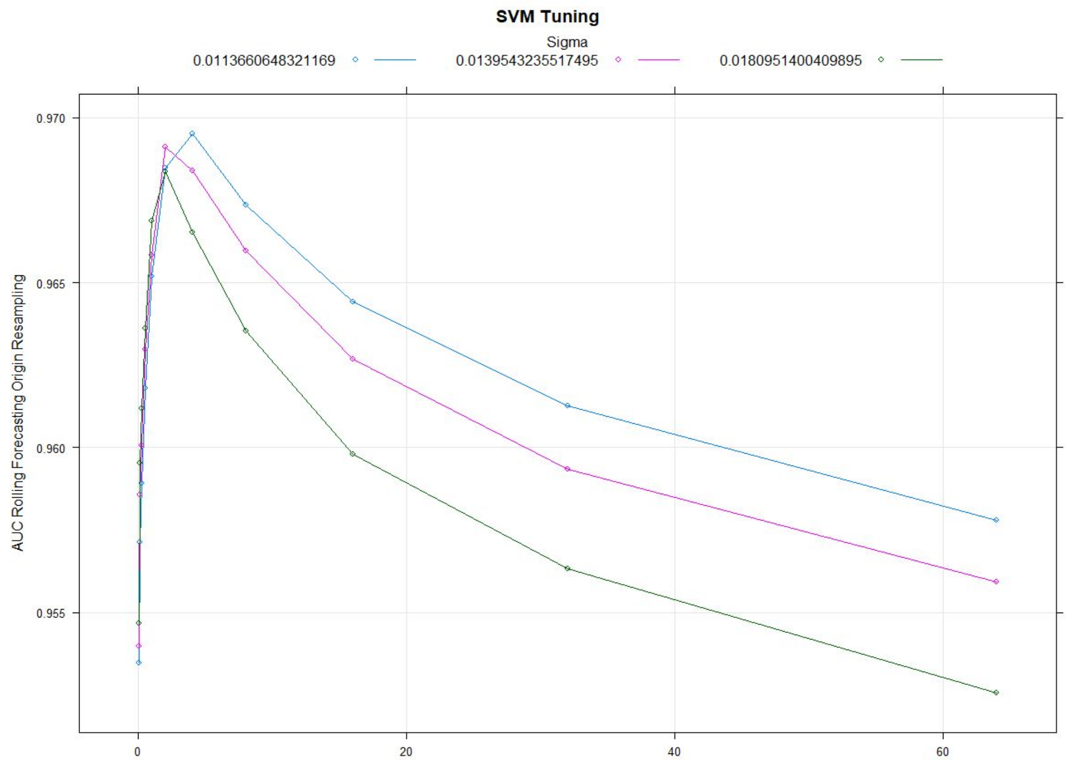
C. **NNET Model Tuning:**
   Optimal model parameters using size=6, decay=0.5



Neural Network Tuning Summary:

```
Confusion Matrix and Statistics

          Reference
Prediction  Low Medium High
    Low    2357    134    0
    Medium   82   1630   85
    High     35    274 3516

Overall Statistics

               Accuracy : 0.9248
                 95% CI : (0.9189, 0.9305)
    No Information Rate : 0.4439
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8828

 Mcnemar's Test P-Value : < 2.2e-16

Tuning parameter 'bag' was held constant at a value of FALSE
AUC was used to select the optimal model using the largest value.
The final values used for the model were size = 6, decay = 0.5 and bag = FALSE.
```

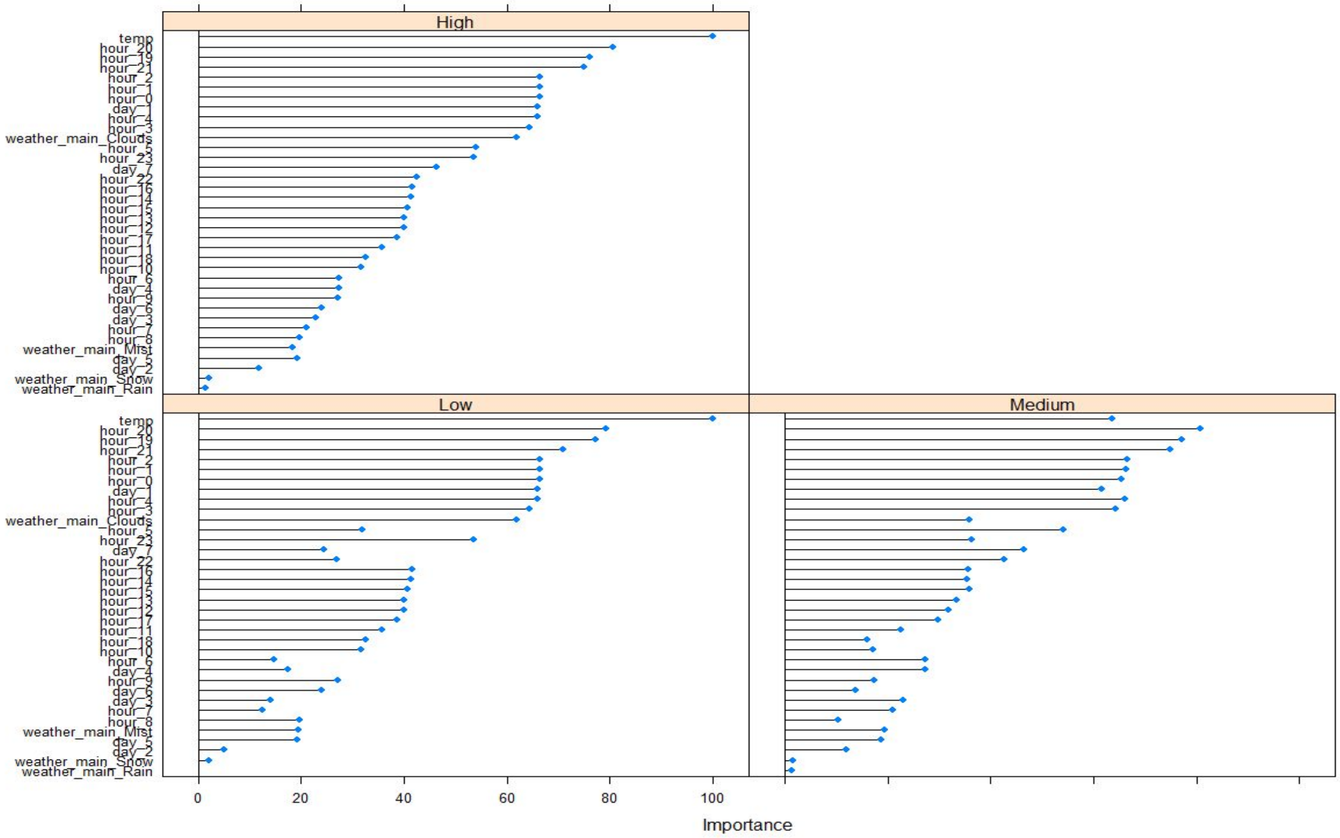
D. SVM Model Tuning:
Optimal model parameters using sigma=0.011366, Cost=4



SVM Tuning Summary:

```
Confusion Matrix and Statistics

          Reference
Prediction  Low Medium High
    Low    2355    129    0
    Medium   84   1627   68
    High     35    282 3533

Overall Statistics

               Accuracy : 0.9263
                 95% CI : (0.9204, 0.9319)
    No Information Rate : 0.4439
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.885

 Mcnemar's Test P-Value : < 2.2e-16

AUC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.01136606 and C = 4.
```

E. Variable Importance:


High
Low
Medium
temp
hour_20
hour_19
hour_21
hour_2
hour_1
hour_0
day_1
hour_4
hour_3
weather_main_Clouds
hour_5
hour_23
day_7
hour_22
hour_16
hour_14
hour_15
hour_13
hour_12
hour_17
hour_11
hour_18
hour_10
hour_6
day_4
hour_9
day_6
day_3
hour_7
hour_8
weather_main_Mist
day_5
day_2
weather_main_Snow
weather_main_Rain
0
20
40
60
80
100
Importance

9. R Code

Regression Models

```
## Parallel Computing Setup for 4 CPU Cores
library(doParallel)
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

#### Pre-processing ####
df<- read.csv("~/MTU/MA5790 - Predictive Modeling/MA5790 Predictive Modeling
Project/Metro_Interstate_Traffic_Volume.csv.gz")
#Filter Duplicates
df_filt_dup <- df[!duplicated(df$date_time),]
#Datetime conversion to categorical
library(lubridate)
df.new <- data.frame(df_filt_dup,
                year = year(df_filt_dup$date_time),
                month = month(df_filt_dup$date_time),
                day = wday(df_filt_dup$date_time),
                hour = hour(df_filt_dup$date_time))
df.new <- subset(df.new, select = -date_time)
#Near Zero Variance, NA's, split response
library(caret)
nearZeroVar(df.new)
df.new <- subset(df.new, select = -c(nearZeroVar(df.new)))
df.new$temp[df.new$temp==0]<-NA
df.new = na.omit(df.new)
df.newnew <- df.new[,-c(5)]
df.response <- df.new[,5] #Response
#Dummy Variables for Categorical
facts <- c('year', 'month', 'day', 'hour')
df.newnew[facts] <- lapply(df.newnew[facts], factor)
dummies <- dummyVars("~+day+hour+weather_main",
                sep = '_', data = df.newnew, fullRank = F)
df.dummy <- data.frame(predict(dummies, df.newnew))
df.dummy <- cbind(df.newnew, df.dummy)
df.dummy <- subset(df.dummy, select = -c(year, month,
                               day, hour,
                               weather_main,
                               weather_description))
```

```
# Near Zero Variance Again, or plus hours, or not at all
nzColumns=nearZeroVar(df.dummy,freqCut = 25/1)
df.dummy_nzv = df.dummy[,-nzColumns]
#df.dummy_nzv_hrs = cbind(df.dummy_nzv,df.dummy[,29:52])
names(df.dummy_nzv)
#names(df.dummy_nzv_hrs)
names(df.dummy)
rm(df.new);rm(df);rm(df.newnew);rm(df_filt_dup)

library(corrplot)
correlation = cor(df.dummy_nzv)
corrplot(correlation, method = "color", diag = F, type = "lower",
        tl.pos = "ld",tl.cex=.666, tl.col = "black",
        title = "Correlation Plot",
        mar=c(0,0,1,0))

hcor = findCorrelation(correlation,cutoff = .3, exact = T)
df.dummy_nzv_hcor = df.dummy_nzv[,-hcor]

correlation = cor(df.dummy_nzv_hcor)
corrplot(correlation, method = "color", diag = F, type = "lower",
        tl.pos = "ld",tl.cex=.666, tl.col = "black",
        title = "Correlation Plot",
        mar=c(0,0,1,0))
names(df.dummy_nzv_hcor)

## Time Slice Method
timesplitdata = function(x,y,trainpercentage){
  ## Data Splitting
  trainingRows <- floor(length(y)*trainpercentage)
  testingRows <-
  # Subset the data into objects for training
  trainX <<- x[1:trainingRows,]
  trainY <<- y[1:trainingRows]
  # Do the same for the test set using negative integers.
  testX <<- x[-c(1:trainingRows), ]
  testY <<- y[-c(1:trainingRows)]
  cat(paste("Data has been split into the following dimensions:\n"),
      paste("trainX has",dim(trainX)[1],"observations, and",
            dim(trainX)[2],"predictors\n"),
      paste("testX has",dim(testX)[1],"observations, and",
            dim(testX)[2],"predictors\n"),
      paste("trainY has",length(trainY),"observations\n",
```

```r
          "testY has",length(testY),"observations\n\n"))
  return(
    print("trainX,trainY,testX, and testY are now global variables"))
}

timesplitdata(df.dummy_nzv_hcor,df.response,0.8)


myTimeControl <- trainControl(method = "timeslice",
                              initialWindow = 672,
                              horizon = 168, skip = 167,
                              fixedWindow = TRUE,
                              preProcOptions = c("center", "scale"))

#myControl <- trainControl(method = "cv", number = 5, repeats = 3,
#                          preProcOptions = c("center", "scale"))
# PLS
set.seed(1)
plsModel <- train(trainX, trainY,
                  method = "pls",
                  preProc = c("center", "scale"),
                  tuneLength = 19,
                  trControl = myTimeControl)
plsModel
plsPred <- predict(plsModel, testX)
postResample(pred = plsPred, obs = testY)[1:2]

# Lars
set.seed(1)
larsModel <- train(trainX, trainY,
                   method = "lars",
                   preProc = c("center", "scale"),
                   tuneLength = 40,
                   trControl = myTimeControl)
larsModel
larsPred <- predict(larsModel, testX)
postResample(pred = larsPred, obs = testY)[1:2]

# ENET
set.seed(1)
enetGrid <- expand.grid(
  .lambda = seq(0, .2, length = 25),
  .fraction = seq(.01, 1, length = 30))
```

```
enetModel <- train(trainX, trainY,
                method = "enet",
                preProc = c("center", "scale"),
                tuneGrid = enetGrid,
                trControl = myTimeControl)
enetModel
enetPred = predict(enetModel, testX)
postResample(pred = enetPred, obs = testY)[1:2]

# MARS
marsGrid <- expand.grid(.degree = c(1:3),
                        .nprune = c(2:45))
set.seed(1)
marsModel <- train(trainX, trainY,
                method = "earth",
                preProc = c("center", "scale"),
                tuneGrid = marsGrid,
                #tuneLength = 10,
                trControl = myTimeControl)
marsModel
marsPred <- predict(marsModel, testX)
postResample(pred = marsPred, obs = testY)[1:2]


# KNN
set.seed(1)
knnModel <- train(trainX, trainY,
               method = "knn",
               preProc = c("center", "scale"),
               tuneGrid = data.frame(.k=c(1:15)),
               trControl = myTimeControl)
knnModel
knnPred = predict(knnModel, testX)
postResample(pred = knnPred, obs = testY)[1:2]
set.seed(1000)
val<-sample(trainY,1000)
postResample(predict(knnModel,testX[val,]),testY[val])
# Neural Network Model
nnetGrid <- expand.grid(.size = 1:10, .decay = c(1, 2, 4, 8), .bag = FALSE)
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (ncol(trainX) + 1) + (maxSize+1)*2)
set.seed(1)
nnetModel <- train(trainX, trainY,
```

```
              method = "avNNet",
              tuneGrid = nnetGrid,
              preProc = c("center", "scale"),
              #tuneLength = 20,
              trControl = myTimeControl,
              linout = TRUE,
              trace = FALSE,
              MaxNWts = numWts,
              maxit = 500)
nnetModel
nnetPred = predict(nnetModel, testX)
postResample(pred = nnetPred, obs = testY)[1:2]

# SVMradial
library(kernlab)
set.seed(1)
sigmaRangeReduced <- sigest(as.matrix(trainX))
svmRGridReduced <- expand.grid(.sigma = c(sigmaRangeReduced[1],
sigmaRangeReduced[2],
                               sigmaRangeReduced[3]), .C = 2^(seq(-2, 12)))
svmModel <- train(trainX, trainY,
             method = "svmRadial",
             preProc = c("center", "scale"),
             #tuneLength = 15,
             tuneGrid = svmRGridReduced,
             trControl = myTimeControl)
svmModel
svmPred <- predict(svmModel, testX)
postResample(pred = svmPred, obs = testY)[1:2]

# Model Comparison
trainPerformance = data.frame()
testPerformance = data.frame()
trainPerformance = rbind(pls = getTrainPerf(plsModel)[1:2],
                   enet = getTrainPerf(enetModel)[1:2],
                   lars = getTrainPerf(larsModel)[1:2],
                   nnet = getTrainPerf(nnetModel)[1:2],
                   mars = getTrainPerf(marsModel)[1:2],
                   svm = getTrainPerf(svmModel)[1:2],
                   knn = getTrainPerf(knnModel)[1:2])
testPerformance = rbind(pls = postResample(pred = plsPred,
                               obs = testY)[1:2],
                  enet = postResample(pred = enetPred,
```

```
                          obs = testY)[1:2],
                  lars = postResample(pred = larsPred,
                          obs = testY)[1:2],
                  nnet = postResample(pred = nnetPred,
                          obs = testY)[1:2],
                  mars = postResample(pred = marsPred,
                          obs = testY)[1:2],
                  svm = postResample(pred = svmPred,
                          obs = testY)[1:2],
                  knn = postResample(pred = knnPred,
                          obs = testY)[1:2])
colnames(testPerformance) = c("TestRMSE","TestRsquared")
trainPerformance;testPerformance

#Plot Predictions
testPredictions <- cbind.data.frame(plsPred,enetPred,larsPred,
                              nnetPred,marsPred,svmPred,knnPred)
colnames(testPredictions) <- c("plsPred","enetPred","larsPred",
                        "nnetPred","marsPred","svmPred","knnPred")
cbPalette <- c("#999999", "#CC79A7", "#56B4E9", "#009E73",
          "#F0E442", "#0072B2", "#D55E00", "#E69F00")
iStart=0;iStop=168
plot(testY[iStart:iStop],type="p",col="black",lwd = 5,
   lty=2,xlab="Hour",ylab="Traffic Volume",
   main = "1 Week of testY and Corresponsing Model Predictions")
for (m in 4:ncol(testPredictions)) {
lines(testPredictions[iStart:iStop,m],type="l",
    col=cbPalette[m],lwd = 3)
}
points(testY[iStart:iStop],type="p",col="black",lwd = 5)
legend("topleft",
     legend = c("testY",names(testPredictions)[4:ncol(testPredictions)]),
     col=c("black",cbPalette[4:ncol(testPredictions)]),
     lty = c(NA,rep(1,8)), lwd = c(7,rep(7,8)),
     pch = c(1,rep(NA,9)),
     ncol = 1, bty = 'n')


# plot predictions and observed
library(grid)
library(gridExtra)
trainPredictions <- cbind.data.frame(predict(plsModel),predict(enetModel),
                              predict(larsModel),predict(nnetModel),
```

```
                              predict(marsModel),predict(svmModel),
                              predict(knnModel))
colnames(trainPredictions) <- c("plsPred","enetPred","larsPred",
                          "nnetPred","marsPred","svmPred","knnPred")

xyplot(testY ~ svmPred, type = c("p", "g"),
      xlab = "Predicted", ylab = "Observed",
      main = "svmModel Testing Prediction")

xyplot(trainY ~ predict(svmModel), type = c("p", "g"),
      xlab = "Predicted", ylab = "Observed",
      main = "svmModel Training Prediction",
      panel = function(...) {
        panel.xyplot(...)
        panel.abline(1,1)})
m=4
xyplot(testY ~ testPredictions[,m], type = c("p", "g"),
      xlab = "Predicted", ylab = "Observed",
      main = paste(colnames(testPredictions[m]),"Testing Prediction"),
      panel = function(...) {
        panel.xyplot(...)
        panel.abline(1,1)})



## Save Models
library(caretEnsemble)
timemodels <- c(pcrModel,plsModel,glmnModel,larsModel,
             nnetModel,marsModel,svmModel,knnModel)



## Variable Importance
#https://stackoverflow.com/questions/48055259/extracting-more-than-20-variables-by-importanc
#e-via-varimp/48056842
library(dplyr)
library(tibble)
plot(varImp(nnetModel),main = "NNET Importance")
plot(varImp(marsModel),main = "MARS Importance")
plot(varImp(svmModel),main = "SVM Importance")
plot(varImp(knnModel),main = "KNN Importance")
varImp(knnModel)$importance %>%
  as.data.frame() %>%
```

```
  rownames_to_column() %>%
  arrange(Overall) %>%
  mutate(rowname = forcats::fct_inorder(rowname )) %>%
  ggplot()+
  geom_col(aes(x = rowname[1:5], y = Overall[1:5]))+
  coord_flip()+
  theme_bw()

## Training Time
knnModel$times$everything['elapsed']

##
## Stop Parallel Computing:
stopCluster(cl)

registerDoSEQ()
```

Classification Models

```
## Parallel Computing Setup for 4 CPU Cores
library(doParallel)
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

#### Pre-processing ####
df<- read.csv("C:\\Users\\mrkhs\\OneDrive\\Desktop\\ORI_data.csv")
# Corece the response variable to categorical
df$traff_vol <- cut(df$traffic_volume, include.lowest = TRUE,
                    breaks = c(0, 2000, 4000, 100000000),
                    labels = c("Low","Medium" ,"High"))
names(df)[names(df) == "traff_vol"] <- "traffic_volume"
df <- df[, -c(9)]
#Filter Duplicates
df_filt_dup <- df[!duplicated(df$date_time),]
#Datetime conversion to categorical
library(lubridate)
df.new <- data.frame(df_filt_dup,
                 year = year(df_filt_dup$date_time),
                 month = month(df_filt_dup$date_time),
                 day = wday(df_filt_dup$date_time),
                 hour = hour(df_filt_dup$date_time))
df.new <- subset(df.new, select = -date_time)
#Near Zero Variance, NA's, split response
library(caret)
nearZeroVar(df.new)
```

```
df.new <- subset(df.new, select = -c(nearZeroVar(df.new)))
df.new$temp[df.new$temp==0]<-NA
df.new = na.omit(df.new)
df.newnew <- df.new[,-c(5)]
df.response <- df.new[,5] #Response
#Dummy Variables for Categorical
facts <- c('year', 'month', 'day', 'hour')
df.newnew[facts] <- lapply(df.newnew[facts], factor)
dummies <- dummyVars("~+day+hour+weather_main",
              sep = '_', data = df.newnew, fullRank = F)
df.dummy <- data.frame(predict(dummies, df.newnew))
df.dummy <- cbind(df.newnew, df.dummy)
df.dummy <- subset(df.dummy, select = -c(year, month,
                            day, hour,
                            weather_main,
                            weather_description))
# Near Zero Variance Again, or plus hours, or not at all
nzColumns=nearZeroVar(df.dummy,freqCut = 25/1)
df.dummy_nzv = df.dummy[,-nzColumns]
#df.dummy_nzv_hrs = cbind(df.dummy_nzv,df.dummy[,29:52])
names(df.dummy_nzv)
#names(df.dummy_nzv_hrs)
names(df.dummy)
rm(df.new);rm(df);rm(df.newnew);rm(df_filt_dup)

library(corrplot)
correlation = cor(df.dummy_nzv)
corrplot(correlation, method = "color", diag = F, type = "lower",
      tl.pos = "ld",tl.cex=.666, tl.col = "black",
      title = "Correlation Plot",
      mar=c(0,0,1,0))

hcor = findCorrelation(correlation,cutoff = .3, exact = T)
df.dummy_nzv_hcor = df.dummy_nzv[,-hcor]

correlation = cor(df.dummy_nzv_hcor)
corrplot(correlation, method = "color", diag = F, type = "lower",
      tl.pos = "ld",tl.cex=.666, tl.col = "black",
      title = "Correlation Plot",
      mar=c(0,0,1,0))
names(df.dummy_nzv_hcor)
barplot(table(df.response), main = 'Population Response')
## Time Slice Method
timesplitdata = function(x,y,trainpercentage){
  ## Data Splitting
  trainingRows <- floor(length(y)*trainpercentage)
  testingRows <-
  # Subset the data into objects for training
```

```
  trainX <<- x[1:trainingRows,]
  trainY <<- y[1:trainingRows]
  # Do the same for the test set using negative integers.
  testX <<- x[-c(1:trainingRows), ]
  testY <<- y[-c(1:trainingRows)]
  cat(paste("Data has been split into the following dimensions:\n"),
     paste("trainX has",dim(trainX)[1],"observations, and",
          dim(trainX)[2],"predictors\n"),
     paste("testX has",dim(testX)[1],"observations, and",
          dim(testX)[2],"predictors\n"),
     paste("trainY has",length(trainY),"observations\n",
          "testY has",length(testY),"observations\n\n"))
  return(
   print("trainX,trainY,testX, and testY are now global variables"))
}

timesplitdata(df.dummy_nzv_hcor,df.response,0.8)
barplot(table(trainY), main = 'Train Response')
barplot(table(testY), main = 'Test Response')

myTimeControl <- trainControl(method = "timeslice",
                              summaryFunction = multiClassSummary,
                              initialWindow = 672,
                              horizon = 168, skip = 167,
                              fixedWindow = TRUE, classProbs = T,
                              preProcOptions = c("center", "scale"))

myTimeControl_fda <- trainControl(method = "timeslice",
                              summaryFunction = multiClassSummary,
                              initialWindow = 672,
                              horizon = 168, skip = 168,
                              fixedWindow = TRUE, classProbs = T,
                              preProcOptions = c("center", "scale"))

#===== KNN =====#
set.seed(1)
library(MLmetrics)
knnModel <- train(trainX, trainY,
                  method = "knn", metric = "AUC",
                  preProc = c("center", "scale"),
                  tuneGrid = data.frame(.k = 1:20), trControl = myTimeControl)
knnModel
knnPred = predict(knnModel, testX)
postResample(pred = knnPred, obs = testY)[1:2]
knnModel$times$everything['elapsed']
# plot KNN
plot(knnModel, main = "KNN Tuning")
# confusion matrix
```

```
confusionMatrix(knnPred, testY)
#===== FDA =====#
set.seed(1)
marsGrid <- expand.grid(.degree = 1:3, .nprune = 2:38)

fdaTuned <- train(x = trainX,
              y = trainY,
              method = "fda", metric = "AUC",
              # Explicitly declare the candidate models to test
              tuneGrid = marsGrid,
              trControl = myTimeControl_fda)
fdaTuned
fdaPred = predict(fdaTuned, testX)
postResample(pred = fdaPred, obs = testY)[1:2]
fdaTuned$times$everything['elapsed']
# plot FDA
plot(fdaTuned, main = "FDA Tuning")
# confusion matrix
confusionMatrix(fdaPred, testY)

#===== Neural Network Model =====#
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 0.5), .bag = FALSE)
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (ncol(trainX) + 1) + (maxSize+1)*2)
set.seed(1)
nnetModel <- train(trainX, trainY,
              method = "avNNet",
              metric = "AUC",
              tuneGrid = nnetGrid,
              tuneLength = 20,
              trControl = myTimeControl,
              linout = TRUE,
              trace = FALSE,
              MaxNWts = numWts,
              maxit = 500)
nnetModel
nnetPred = predict(nnetModel, testX)
postResample(pred = nnetPred, obs = testY)[1:2]
nnetModel$times$everything['elapsed']
plot(nnetModel, main = "Neural Network Tuning")
# confusion matrix
confusionMatrix(nnetPred, testY)
#===== SVM =====#
library(kernlab)
sigmaRangeReduced <- sigest(as.matrix(trainX))
svmRGridReduced <- expand.grid(.sigma = c(sigmaRangeReduced[1],
sigmaRangeReduced[2],
                              sigmaRangeReduced[3]), .C = 2^(seq(-4, 6)))
```

```
set.seed(1)
svmRModel <- train(x = trainX,
                y = trainY,
                method = "svmRadial",
                metric = "AUC",
                preProc = c("center", "scale"),
                tuneGrid = svmRGridReduced,
                fit = FALSE,
                trControl = myTimeControl)
svmRModel
svmPred = predict(svmRModel, testX)
postResample(pred = svmPred, obs = testY)[1:2]
svmRModel$times$everything['elapsed']
plot(svmRModel, main = "SVM Tuning")
# confusion matrix
confusionMatrix(svmPred, testY)



#===== Naive Bayes (Try)=====#
library(klaR)
set.seed(1)
nbFit <- train( x = trainX,
              y = trainY,
              method = "nb",
              metric = "AUC",
              preProc = c("center", "scale"),
              tuneGrid = data.frame(.fL = 0:10,.usekernel = TRUE,.adjust = seq(0, 10, by = 1)),
              trControl = myTimeControl)
nbFit
nbPred <- predict(nbFit, testX)
nbPred_train <- predict(nbFit, trainX)
# plot the nb model
plot(nbFit, main = "Naive Bayes Tuning")
# confusion matrix
confusionMatrix(nbPred, testY)
nbFit$times$everything['elapsed']

# Model Comparison
trainPerformance = data.frame()
testPerformance = data.frame()
trainPerformance = rbind(knn = getTrainPerf(knnModel)[4:5],
                    fda = getTrainPerf(fdaTuned)[4:5],
                    svm = getTrainPerf(svmRModel)[4:5],
                    nnet = getTrainPerf(nnetModel)[4:5],
                    nb = getTrainPerf(nbFit)[4:5])
testPerformance = rbind(knn = postResample(pred = knnPred, obs = testY)[1:2],
                   fda = postResample(pred = fdaPred, obs = testY)[1:2],
```

```
                    svm = postResample(pred = svmPred, obs = testY)[1:2],
                    nnet = postResample(pred = svmPred, obs = testY)[1:2],
                    nb = postResample(pred = nbPred, obs = testY)[1:2])
colnames(testPerformance) = c("TestAccuracy","TestKappa")
trainPerformance;testPerformance
#importance of svm
plot(varImp(svmRModel))
## Stop Parallel Computing:
stopCluster(cl)

registerDoSEQ()
```

10. References

1. Hogue, John. (2019). “Metro Interstate Traffic Volume Data Set”. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Available from: https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume
2. Hyndman, R.J. & Athanasopoulos, G (2018). Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. Available from: https://otexts.com/fpp2/accuracy.html
3. Kuhn, M (2019). “Data Splitting for Time Series”. The caret package. Available from: https://topepo.github.io/caret/data-splitting.html#data-splitting-for-time-series
4. Kuhn, M & Johnson, K (2013). Applied Predictive Modeling. Springer Science + Business Media.
5. MNDoT (2019). “Collection Methods”. Traffic Forecasting and Analysis. St. Paul, MN. Available from: http://www.dot.state.mn.us/traffic/data/coll-methods.html