**Motivation**

We’ve read several studies and articles that claim Econometric models are still superior to machine learning when it comes to forecasting. In the article, [“Statistical and Machine Learning forecasting methods: Concerns and ways forward”](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889), the author mentions that,

*“After comparing the post-sample accuracy of popular ML methods with that of eight traditional statistical ones, we found that the former are dominated across both accuracy measures used and for all forecasting horizons examined.”*

In many business environments a data scientist is responsible for generating hundreds or thousands (possibly more) forecasts for an entire company, opposed to a single series forecast. While it appears that Econometric methods are better at forecasting a single series (which I generally agree with), how do they compare at forecasting multiple series, which is likely a more common requirement in the real world? Some other things to consider when digesting the takeaways from that study:

* Do the ML models benefit from building a single model to forecast all series at once, which most time series models cannot do?
* What are the run-time differences with both approaches?
* The author in the linked article above states that the Econometrics models outperform machine learning models across all forecast horizons but is that really the case?

**Approach**

In this article, We are going to show you an experiment We ran that compares machine learning models and Econometrics models for time series forecasting on an entire company’s set of stores and departments.

Before I kick this off, We have to mention that We’ve come across several articles that describe how one ***can*** utilize ML for forecasting (typically with deep learning models) but We haven’t seen any that truly gives ML the best chance at outperforming traditional Econometric models. On top of that, We also haven’t seen too many legitimate attempts to showcase the best that Econometric models can do either. That’s where this article and evaluation differ. The suite of functions We tested are near-fully optimized versions of both ML models and Econometric models (**list of models and tuning details are below**). The functions come from the R open source package **[RemixAutoML](https://github.com/AdrianAntico/RemixAutoML" \t "_blank)**, which is a suite of functions for automated machine learning (AutoML), automated forecasting, automated anomaly detection, automated recommender systems, automated feature engineering, and more. I provided the R script at the bottom of this article so you can replicate this experiment. You can also utilize the functions in **Python** via the [r2py package](https://rpy2.bitbucket.io/) and **Julia** via the [RCall package](https://github.com/JuliaInterop/RCall.jl" \t "_blank).

**The Data**

The data I’m utilizing comes from [Kaggle](https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting) — weekly Walmart sales by store and department. I’m only using the store and department combinations that have complete data to minimize the noise added to the experiment, which leaves me with a total of 2,660 individual store and department time series. Each store & dept combo has 143 records of weekly sales. I also removed the “IsHoliday” column that was provided.

Preview of Walmart Store Sales Kaggle Data Set

**The Experiment**

Given the comments from the article linked above, I wanted to test out several forecast horizons. The performance for all models are compared on n-step ahead forecasts, for n = {1,5,10,20,30}, **with distinct model builds used for each n-step forecast test**. For each run, I have 2,660 evaluation time series for comparison, represented by each store and department combination. In the Results section you can find the individual results for each of those runs.

**The Models**

In the experiment I used the **AutoTS()**function for testing out Econometric models and I used the **RemixAutoML** **CARMA** suite (**C**alendar-**A**uto-**R**egressive-**M**oving-**A**verage) for testing out Machine Learning. The **AutoTS()** function tests out every model from the list below in in several ways (similar to grid tuning in ML). The ML suite contains 4 different tree-based algorithms. As a side note, the Econometric models all come from the [forecast package in R](https://cran.r-project.org/web/packages/forecast/index.html). You can see a detailed breakdown of how each model is optimized below the **Results** section in this article.

**Econometrics Models used in AutoTS()**

1. DSHW — Double-Seasonal Holt-Winters
2. ARIMA — Autoregressive, integrated, moving average
3. ARFIMA — Fractionally differenced ARIMA
4. ETS — Exponential Smoothing State-Space Model
5. NN — Feed-forward neural network with a single hidden layer and lagged inputs. **I’m counting this towards Econometrics because it came from the Forecast package in R which is an Econometrics package along with the fact that it’s not as customizable as a TensorFlow or PyTorch model. (Besides, I’ve seen authors state that linear regression is machine learning which would imply that all the Econometrics methods are Machine Learning but I don’t want to debate that here). If you want to consider the NN as a Machine Learning model, just factor that into the results data below.**
6. TBATS (Exponential smoothing state-space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components)
7. TSLM — time series linear model with trend and seasonal components

Example Plot from AutoTS(): Single Store Forecast with 80% & 95% Prediction Intervals

**Machine Learning Algorithms**

1. AutoCatBoostCARMA() — CatBoost
2. AutoXGBoostCARMA() — XGBoost
3. AutoH2oGBMCARMA() — H2O Gradient Boosting Machine
4. AutoH2oDRFCARMA() — H2O Distributed Random Forest

Example Plot from AutoCatBoostCARMA(): Aggregated Forecast

**Results**

The table outputs below shows the ranks of 11 models (7 Econometric and 4 Machine Learning) when it comes to lowest mean absolute error (MAE) for every single store and department combination (2,660 individual time series) across five different forecast horizons.

For example, in the 1-step ahead forecast table below, NN was the most accurate model on 666 of the 2,660 time series. TBATS was the most accurate 414 times out of the 2,660.

Still looking at the 1-step ahead forecast table below, the NN was the second most accurate on 397 out of 2,660 time series. TBATS was the second most accurate on 406 out of the 2,660 time series. TBATS ranked last place (11th) 14 times.

* **Winner in Single Model Accuracy — TBATS is the winner** of the competition (Econometrics model) with a mean rank of 1.6 across all five forecast horizons.
* **Runner-Up in Single Model Accuracy** — Catboost is the runner up of the competition (Machine Learning model) with a mean rank of 3.6 across all five forecast horizons.
* **Winner in Run time — ML is winner**: For a single run (there were 5 total, 1 for each forecast horizon) the Econometrics automated forecasting took an average of **33 hours**! to run while the automated ML models took an average of**3.5**hours, where each run included a grid tune of 6 comparisons, (1 hour for CatBoost, 1 hour for XGBoost, 30 minutes for H2O GBM and 1 hour for H2O Distributed Random Forest).
* **Winner in Shorter-Horizon Accuracy**— The Econometrics models dominate on shorter forecast horizons.
* **Winner in Longer Horizon Accuracy** — The Machine Learning models dominate on longer forecast horizons.

**Aggregate Summaries**

Mean Rank by Model

Ranks: Long Term = {20,30} Period-Ahead & Short Term = {1,5,10} Period-Ahead

The histograms below were derived from selecting the best Econometrics models for each individual store and department time series (essentially the ensemble results) and the best Machine Learning models for each individual store and department time series (ensemble). You can see that as the forecast horizon grows, the Machine Learning models catch up and overcome (slightly) the Econometrics models. With the shorter forecast horizon, the Econometrics models outperform the Machine Learning models by a larger amount than the converse.

Forecast MAE by Econometrics(Blue) and Machine Learning(Gold): 1-Period, 5-Period, 20-Period, 30-Period

**Individual Forecast Horizon Summaries by Model**

1- Period Ahead Forecast Model Counts by Rank (based on lowest MAE)

5- Period Ahead Forecast Model Counts by Rank (based on lowest MAE)

10- Period Ahead Forecast Model Counts by Rank (based on lowest MAE)

20- Period Ahead Forecast Model Counts by Rank (based on lowest MAE)

30- Period Ahead Forecast Model Counts by Rank (based on lowest MAE)

**Conclusion**

***While the short term horizon forecasts are more accurate via the Econometrics models I tend to have a greater need for longer term forecasts for planning purposes and the Machine Learning models exceed the Econometrics in that category. On top of that, the run-time is a pretty significant factor for me.***

***If your business needs are the opposite, the Econometrics models are probably your best bet, assuming the run times are not a concern.***

***If I had enough resources available I’d run both functions and utilize the individual models that performed best for each series, which means I’d be utilizing all 11 models.***

**Algorithm Tuning Details**

**Econometrics Model Details:**

Each of the individual Econometrics models in **AutoTS()** are optimized based on the following treatments.

**Global Optimizations (applies to all models):**

A) Optimal **Box-Cox Transformations** are used in every run where data is strictly positive. The optimal transformation could be no transformation (artifact of Box-Cox).

B) Four different treatments are tested for each model:

* user-specified time frequency + no outlier smoothing & no imputation
* model-based time frequency + no outlier smoothing & no imputation
* user-specified time frequency + outlier smoothing & imputation
* model-based time frequency + outlier smoothing & imputation

The treatment of outlier smoothing and imputation sometimes has a beneficial effect on forecasts; sometimes it doesn’t. You really need to test out both to see what generates more accurate predictions out-of-sample. Same goes with manually defining the frequency of the data. If you have daily data, you specify “day” in the AutoTS arguments. Alternatively, if specified, spectral analysis is done to find the frequency of the data based on the dominant trend and seasonality. Sometimes this approach works better, sometimes it doesn’t. That’s why I test all the combinations for each model.

**Individual Model Optimizations:**

C) For the **ARIMA**and **ARFIMA**, I used up to 25 lags and moving averages, algorithmically determined how many differences and seasonal differences to use, and up to a single difference and seasonal difference can be used, all determined in the stepwise procedure *(all combinations can be tested and run in parallel but it’s too time consuming for my patience).*

D) For the **Double Seasonal Holt-Winters** model, alpha, beta, gamma, omega, and phi are determined using least-squares and the forecasts are adjusted using an AR(1) model for the errors.

E) The **Exponential Smoothing State-Space** model runs through an automatic selection of the error type, trend type, and season type, with the options being “none”, “additive”, and “multiplicative”, along with testing of damped vs. non-damped trend (either additive or multiplicative). Alpha, beta, and phi are estimated.

F) The **Neural Network** is set up to test out every combination of lags and seasonal lags (25 lags, 1 seasonal lag) and the version with the best holdout score is selected.

G) The **TBATS**model utilizes 25 lags and moving averages for the errors, damped trend vs. non-damped trend are tested, trend vs. non-trend are also tested, and the model utilizes parallel processing.

H) The **TSLM**model utilizes simple time trend and season depending on the frequency of the data.

**Machine Learning Model Details:**

The **CARMA**suite utilizes several features to ensure proper models are built to generate the best possible out-of-sample forecasts.

A) **Feature engineering**: I use a time trend, calendar variables, holiday counts, and 25 lags and moving averages along with 51, 52, and 53-week lags and moving averages (all specified as arguments in the CARMA function suite). *Internally, the CARMA functions utilize several* [***RemixAutoML***](https://github.com/AdrianAntico/RemixAutoML) *functions, all written using* ***data.table****for fast and memory efficient processing:*

* ***DT\_GDL\_Feature\_Engineering() —****creates lags and moving average features by grouping variables (also creates lags and moving averages off of time between records)*
* ***Scoring\_GDL\_Feature\_Engineering() —****creates lags and moving average features for a single record by grouping variables (along with the time between features)*
* ***CreateCalendarVariables() —****creates numeric features identifying various time units based on date columns*
* ***CreateHolidayFeatures()****— creates count features based on the specified holiday groups you want to track and the date columns you supply*

B) **Optimal transformations:** the target variable along with the associated lags and moving average features were transformed. This is really useful for regression models with categorical features that have associated target values that significantly differ from each other. The transformation options that are tested (using a Pearson test for normality) include:

* YeoJohnson, Box-Cox, Arcsinh, Identity,
* arcsin(sqrt(x)), logit(x) — for proportion data, not used in experiment

The functions used to create the transformations throughout the process and then back-transform them after the forecasts have been generated come from [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) :

* **AutoTransformationCreate()**
* **AutoTransformationScore()**

C) **Models:** there are four CARMA functions and each use a different algorithm for the model fitting. The models used to fit the time series data come from [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) and include:

* **AutoCatBoostRegression()**
* **AutoXGBoostRegression()**
* **AutoH2oDRFRegression()**
* **AutoH2oGBMRegression()**

You can view all of the 21 process steps in those functions on my [GitHub page README](https://github.com/AdrianAntico/RemixAutoML/blob/master/README.md) under the section titled, “Supervised Learning Models” in the “Regression” sub-section (you can also view the source code directly of course).

D) **GPU**: With the CatBoost and XGBoost functions, you can build the models utilizing GPU (I ran them with a GeForce 1080ti) which results in an average 10x speedup in model training time (compared to running on CPU with 8 threads). I should also note, the lags and moving average features by store and department and pretty intensive to compute and are built with data.table exclusively which means that if you have a CPU with a lot of threads then those calculations will be faster as data.table is parallelized.

E) **One model for all series**: I built the forecasts for all the store and department combinations with a single model by simply specifying c(“Store”,”Dept”) in the **GroupVariables**argument, which provides superior results compared to building a single model for each series. The group variables are used as categorical features and do not require one-hot-encoding before hand as CatBoost and H2O handle those internally. The **AutoXGBoostCARMA()** version utilizes the **DummifyDT()** function from [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) to handle the categorical features.

F) The **max**number of trees used for each model was (early stopping is used internally):

* **AutoCatBoostCARMA() = 20,000**
* **AutoXGBoostCARMA() = 5,000**
* **AutoH2oDRFCARMA() = 2,000**
* **AutoH2oGBMCARMA() = 2,000**

G) **Grid tuning**: I ran a 6 model random hyper-parameter grid tune for each algorithm. Essentially, a baseline model is built and then 5 other models are built and compared with the lowest MAE model being selected. This is all done internally in the **CARMA**function suite.

H) **Data partitioning:** for creating the training, validation, and test data, the CARMA functions utilize the **RemixAutoML::AutoDataPartition()**function and utilizes the “timeseries” option for the **PartitionType**argument which ensures that the train data reflects the furthest data points back in time, followed by the validation data, and then the test data which is the most recent data points in time. For the experiment, I used 10/143 as the percent holdout for validation data. The test data varied by which n-step ahead holdout was being tested, and the remaining data went to the training set.

I) **Forecasting:** Once the regression model is built, the forecast process replicates an ARIMA process. First, a single step-ahead forecast is made. Next, the lags and moving average features are updated, making use of the predicted values from the previous step. Next, the other features are updated (trend, calendar, holiday). Then the next forecast step is made; rinse and repeat for remaining forecasting steps. This process utilizes the [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) functions:

* **AutoCatBoostScoring()**
* **AutoXGBoostScoring()**
* **AutoH2oMLScoring()**

**Contact**

If anyone is interested in testing out other models, utilizing different data sets, or just need to set up automated forecasts for their company, [contact me on LinkedIn](https://www.linkedin.com/in/adrian-antico/).

If you’d like to learn how to utilize the [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) package check out the free course on [Remyx Courses](https://www.remyxcourses.com/" \t "_blank).

**P.S.**

I have plans to continue enhancing and adding capabilities to the automated time series functions discussed above. For example, I plan to:

* Test Fourier features for both AutoTS() and the CARMA suite
* Test other ML algorithms
* Ensemble methods for combining forecasts
* Add Croston Econometric model for intermittent demand forecasting
* Create machine learning based intermittent demand forecasting models (in a similar fashion to the Croston method) utilizing [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) **Generalized Hurdle Models**

**R Script**

[Code to reproduce: https://gist.github.com/AdrianAntico/8e1bbf63f26835756348d7c67a930227](https://gist.github.com/AdrianAntico/8e1bbf63f26835756348d7c67a930227)

library(RemixAutoML)

library(data.table)

###########################################

# Prepare data for AutoTS()----

###########################################

# Load Walmart Data ----

# link to manually download file: <https://remixinstitute.app.box.com/v/walmart-store-sales-data/>

data <- data.table::fread("<https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv>", header = T, stringsAsFactors = FALSE)

# Subset for Stores / Departments with Full Series Available: (143 time points each)----

data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----

keep <- c("Store","Dept","Date","Weekly\_Sales")

data <- data[, ..keep]

# Group Concatenation----

data[, GroupVar := do.call(paste, c(.SD, sep = " ")), .SDcols = c("Store","Dept")]

data[, c("Store","Dept") := NULL]

# Grab Unique List of GroupVar----

StoreDept <- unique(data[["GroupVar"]])

###########################################

# AutoTS() Builds----

###########################################

for(z in c(1,5,10,20,30)) {

TimerList <- list()

OutputList <- list()

l <- 0

for(i in StoreDept) {

l <- l + 1

temp <- data[GroupVar == eval(i)]

temp[, GroupVar := NULL]

TimerList[[i]] <- system.time(

OutputList[[i]] <- tryCatch({

RemixAutoML::AutoTS(

temp,

TargetName = "Weekly\_Sales",

DateName = "Date",

FCPeriods = 1,

HoldOutPeriods = z,

EvaluationMetric = "MAPE",

TimeUnit = "week",

Lags = 25,

SLags = 1,

NumCores = 4,

SkipModels = NULL,

StepWise = TRUE,

TSClean = TRUE,

ModelFreq = TRUE,

PrintUpdates = FALSE)},

error = function(x) "Error in AutoTS run"))

print(l)

}

# Save Results When Done and Pull Them in After AutoCatBoostCARMA() Run----

save(TimerList, file = paste0(getwd(),"/TimerList\_FC\_",z,"\_.R"))

save(OutputList, file = paste0(getwd(),"/OutputList\_FC\_",z,".R"))

rm(OutputList, TimerList)

}

###########################################

# Prepare data for AutoCatBoostCARMA()----

###########################################

# Load Walmart Data----

# link to manually download file: <https://remixinstitute.app.box.com/v/walmart-store-sales-data/>

data <- data.table::fread("<https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv>", header = T, stringsAsFactors = FALSE)

# Subset for Stores / Departments With Full Series (143 time points each)----

data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----

keep <- c("Store","Dept","Date","Weekly\_Sales")

data <- data[, ..keep]

# Build AutoCatBoostCARMA Models----

for(z in c(1,5,10,20,30)) {

CatBoostResults <- RemixAutoML::AutoCatBoostCARMA(

data,

TargetColumnName = "Weekly\_Sales",

DateColumnName = "Date",

GroupVariables = c("Store","Dept"),

FC\_Periods = 10,

TimeUnit = "week",

TargetTransformation = TRUE,

Lags = c(1:25,51,52,53),

MA\_Periods = c(1:25,51,52,53),

CalendarVariables = TRUE,

TimeTrendVariable = TRUE,

HolidayVariable = TRUE,

DataTruncate = FALSE,

SplitRatios = c(1 - 60/143, 30/143, 30/143),

TaskType = "GPU",

EvalMetric = "RMSE",

GridTune = FALSE,

GridEvalMetric = "r2",

ModelCount = 2,

NTrees = 1500,

PartitionType = "timeseries",

Timer = TRUE)

# Output----

CatBoostResults$TimeSeriesPlot

CatBoost\_Results <- CatBoostResults$ModelInformation$EvaluationMetricsByGroup

data.table::fwrite(CatBoost\_Results, paste0(getwd(),"/CatBoost\_Results\_",30,".csv"))

rm(CatBoost\_Results,CatBoostResults)

}

###########################################

# Prepare data for AutoXGBoostCARMA()----

###########################################

# Load Walmart Data ----

# link to manually download file: <https://remixinstitute.app.box.com/v/walmart-store-sales-data/>

data <- data.table::fread("<https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv>", header = T, stringsAsFactors = FALSE)

# Subset for Stores / Departments With Full Series (143 time points each)----

data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----

keep <- c("Store","Dept","Date","Weekly\_Sales")

data <- data[, ..keep]

for(z in c(1,5,10,20,30)) {

XGBoostResults <- RemixAutoML::AutoXGBoostCARMA(

data,

TargetColumnName = "Weekly\_Sales",

DateColumnName = "Date",

GroupVariables = c("Store","Dept"),

FC\_Periods = 2,

TimeUnit = "week",

TargetTransformation = TRUE,

Lags = c(1:25, 51, 52, 53),

MA\_Periods = c(1:25, 51, 52, 53),

CalendarVariables = TRUE,

HolidayVariable = TRUE,

TimeTrendVariable = TRUE,

DataTruncate = FALSE,

SplitRatios = c(1 - (30+z)/143, 30/143, z/143),

TreeMethod = "hist",

EvalMetric = "MAE",

GridTune = FALSE,

GridEvalMetric = "mae",

ModelCount = 1,

NTrees = 5000,

PartitionType = "timeseries",

Timer = TRUE)

XGBoostResults$TimeSeriesPlot

XGBoost\_Results <- XGBoostResults$ModelInformation$EvaluationMetricsByGroup

data.table::fwrite(XGBoost\_Results, paste0(getwd(),"/XGBoost\_Results",z,".csv"))

rm(XGBoost\_Results)

}

###########################################

# Prepare data for AutoH2oDRFCARMA()----

###########################################

# Load Walmart Data ----

# link to manually download file: <https://remixinstitute.app.box.com/v/walmart-store-sales-data/>

data <- data.table::fread("<https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv>", header = T, stringsAsFactors = FALSE)

# Subset for Stores / Departments With Full Series (143 time points each)----

data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----

keep <- c("Store","Dept","Date","Weekly\_Sales")

data <- data[, ..keep]

for(z in c(1,5,10,20,30)) {

H2oDRFResults <- AutoH2oDRFCARMA(

data,

TargetColumnName = "Weekly\_Sales",

DateColumnName = "Date",

GroupVariables = c("Store","Dept"),

FC\_Periods = 2,

TimeUnit = "week",

TargetTransformation = TRUE,

Lags = c(1:5, 51,52,53),

MA\_Periods = c(1:5, 51,52,53),

CalendarVariables = TRUE,

HolidayVariable = TRUE,

TimeTrendVariable = TRUE,

DataTruncate = FALSE,

SplitRatios = c(1 - (30+z)/143, 30/143, z/143),

EvalMetric = "MAE",

GridTune = FALSE,

ModelCount = 1,

NTrees = 2000,

PartitionType = "timeseries",

MaxMem = "28G",

NThreads = 8,

Timer = TRUE)

# Plot aggregate sales forecast (Stores and Departments rolled up into Total)----

H2oDRFResults$TimeSeriesPlot

H2oDRF\_Results <- H2oDRFResults$ModelInformation$EvaluationMetricsByGroup

data.table::fwrite(H2oDRF\_Results, paste0(getwd(),"/H2oDRF\_Results",z,".csv"))

rm(H2oDRF\_Results)

}

###########################################

# Prepare data for AutoH2OGBMCARMA()----

###########################################

# Load Walmart Data ----

# link to manually download file: <https://remixinstitute.app.box.com/v/walmart-store-sales-data/>

data <- data.table::fread("<https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv>", header = T, stringsAsFactors = FALSE)

# Subset for Stores / Departments With Full Series (143 time points each)----

data <- data[, Counts := .N, by = c("Store","Dept")][Counts == 143][, Counts := NULL]

# Subset Columns (remove IsHoliday column)----

keep <- c("Store","Dept","Date","Weekly\_Sales")

data <- data[, ..keep]

for(z in c(1,5,10,20,30)) {

H2oGBMResults <- AutoH2oGBMCARMA(

data,

TargetColumnName = "Weekly\_Sales",

DateColumnName = "Date",

GroupVariables = c("Store","Dept"),

FC\_Periods = 2,

TimeUnit = "week",

TargetTransformation = TRUE,

Lags = c(1:5, 51,52,53),

MA\_Periods = c(1:5, 51,52,53),

CalendarVariables = TRUE,

HolidayVariable = TRUE,

TimeTrendVariable = TRUE,

DataTruncate = FALSE,

SplitRatios = c(1 - (30+z)/143, 30/143, z/143),

EvalMetric = "MAE",

GridTune = FALSE,

ModelCount = 1,

NTrees = 2000,

PartitionType = "timeseries",

MaxMem = "28G",

NThreads = 8,

Timer = TRUE)

# Plot aggregate sales forecast (Stores and Departments rolled up into Total)----

H2oGBMResults$TimeSeriesPlot

H2oGBM\_Results <- H2oGBMResults$ModelInformation$EvaluationMetricsByGroup

data.table::fwrite(H2oGBM\_Results, paste0(getwd(),"/H2oGBM\_Results",z,".csv"))

rm(H2oGBM\_Results)

}

##################################################

# AutoTS() and AutoCatBoostCARMA() Comparison----

##################################################

# Gather results----

for(i in c(1,5,10,20,30)) {

load(paste0("C:/Users/aantico/Desktop/Work/Remix/RemixAutoML/TimerList\_",i,"\_.R"))

load(paste0("C:/Users/aantico/Desktop/Work/Remix/RemixAutoML/OutputList\_",i,"\_.R"))

# Assemble TS Data

TimeList <- names(TimerList)

results <- list()

for(j in 1:2660) {

results[[j]] <- cbind(

StoreDept = TimeList[j],

tryCatch({OutputList[[j]]$EvaluationMetrics[, .(ModelName,MAE)][

, ModelName := gsub("\_.\*","",ModelName)

][

, ID := 1:.N, by = "ModelName"

][

ID == 1

][

, ID := NULL

]},

error = function(x) return(

data.table::data.table(

ModelName = "NONE",

MAE = NA))))

}

# AutoTS() Results----

Results <- data.table::rbindlist(results)

# Remove ModelName == NONE

Results <- Results[ModelName != "NONE"]

# Average out values: one per store and dept so straight avg works----

Results <- Results[, .(MAE = mean(MAE, na.rm = TRUE)), by = c("StoreDept","ModelName")]

# Group Concatenation----

Results[, c("Store","Dept") := data.table::tstrsplit(StoreDept, " ")][, StoreDept := NULL]

data.table::setcolorder(Results, c(3,4,1,2))

##################################

# Machine Learning Results----

##################################

# Load up CatBoost Results----

CatBoost\_Results <- data.table::fread(paste0(getwd(),"/CatBoost\_Results\_",i,".csv"))

CatBoost\_Results[, ':=' (MAPE\_Metric = NULL, MSE\_Metric = NULL, R2\_Metric = NULL)]

data.table::setnames(CatBoost\_Results, "MAE\_Metric", "MAE")

CatBoost\_Results[, ModelName := "CatBoost"]

data.table::setcolorder(CatBoost\_Results, c(1,2,4,3))

# Load up XGBoost Results----

XGBoost\_Results <- data.table::fread(paste0(getwd(),"/XGBoost\_Results",i,".csv"))

XGBoost\_Results[, ':=' (MAPE\_Metric = NULL, MSE\_Metric = NULL, R2\_Metric = NULL)]

data.table::setnames(XGBoost\_Results, "MAE\_Metric", "MAE")

XGBoost\_Results[, ModelName := "XGBoost"]

data.table::setcolorder(XGBoost\_Results, c(1,2,4,3))

# Load up H2oDRF Results----

H2oDRF\_Results <- data.table::fread(paste0(getwd(),"/H2oDRF\_Results",i,".csv"))

H2oDRF\_Results[, ':=' (MAPE\_Metric = NULL, MSE\_Metric = NULL, R2\_Metric = NULL)]

data.table::setnames(H2oDRF\_Results, "MAE\_Metric", "MAE")

H2oDRF\_Results[, ModelName := "H2oDRF"]

data.table::setcolorder(H2oDRF\_Results, c(1,2,4,3))

# Load up H2oGBM Results----

H2oGBM\_Results <- data.table::fread(paste0(getwd(),"/H2oGBM\_Results",i,".csv"))

H2oGBM\_Results[, ':=' (MAPE\_Metric = NULL, MSE\_Metric = NULL, R2\_Metric = NULL)]

data.table::setnames(H2oGBM\_Results, "MAE\_Metric", "MAE")

H2oGBM\_Results[, ModelName := "H2oGBM"]

data.table::setcolorder(H2oGBM\_Results, c(1,2,4,3))

##################################

# Combine Data----

##################################

# Stack Files----

ModelDataEval <- data.table::rbindlist(

list(Results, CatBoost\_Results, XGBoost\_Results, H2oGBM\_Results, H2oDRF\_Results))

data.table::setorderv(ModelDataEval, cols = c("Store","Dept","MAE"))

# Add rank----

ModelDataEval[, Rank := 1:.N, by = c("Store","Dept")]

# Get Frequencies----

RankResults <- ModelDataEval[, .(Counts = .N), by = c("ModelName","Rank")]

data.table::setorderv(RankResults, c("Rank", "Counts"), order = c(1,-1))

# Final table----

FinalResultsTable <- data.table::dcast(RankResults, formula = ModelName ~ Rank, value.var = "Counts")

data.table::setorderv(FinalResultsTable, "1", -1, na.last = TRUE)

# Rename Columns----

for(k in 2:ncol(FinalResultsTable)) {

data.table::setnames(FinalResultsTable,

old = names(FinalResultsTable)[k],

new = paste0("Rank\_",names(FinalResultsTable)[k]))

}

# Print

print(i)

print(knitr::kable(FinalResultsTable))

}