Over 20% of Amazon’s North American retail revenue can be attributed to customers who first tried to buy the product at a local store but found it out-of-stock, [according to IHL group](https://www.ihlservices.com/product/oosoutofluck/) (a global research and advisory firm specializing in technologies for retail and hospitality. **In other words, many companies and local stores suck at forecasting.**

Accurate demand forecasts are necessary if you’re a retailer who has one of their competitors being Amazon. Want to lose business to Amazon? Then produce sh\*\*ty demand forecasts. It is also actually one of the “low-hanging fruits” of a new data science department at a company who’s just getting started on machine learning and AI initiatives. With accurate demand forecasts, you can boost profits by optimizing your labor, prices, and inventory.

What are some examples where several forecasts need to be created all at once?

Typically, when companies are creating forecasts, they’re creating forecasts on a time series basis. That is, they are generating daily, weekly, monthly, quarterly or yearly forecasts. Some real world business applications of time series forecasting are:

**Retail or B2B:**

* Weekly Sales Forecasts by Store. Many companies could have hundreds or thousands of stores. One of our previous employers **operated 1,500 pet stores** in the United States, Canada, and Puerto Rico.
* Daily or Monthly Units Sold by SKU (item). Many retail and B2B companies could have tens of thousands to hundreds of thousands of SKUs. At a previous employer of one of the Remix Institute co-founders, **they had over 650,000 SKUs**. Imagine having to do that many item demand forecasts.

**eCommerce:**

* Daily or Weekly Visits by Channel, Source, and/or Medium using Google Analytics data. This could have thousands of combinations depending on how complex your digital marketing operation is.
* Daily Customers, New Customers, Revenue, and Units Sold by Channel. This could also have hundreds of combinations to forecast for.

What are some challenges of creating automated forecasts for several stores, departments, SKUs, or channels?

**Lack of Automation**. Many current forecasting processes at companies require someone or multiple people to update an ugly, complicated Excel spreadsheet with multiple tabs and formulas. The process for doing this is error prone.

**Scalability**. Often, forecasting processes at companies are done by using someone’s own non-statistical methodology for forecasting, and that someone usually leaves no documentation for how to update it, reverse engineer it, or integrate it to current business processes.

**Computation and Turnaround Time**. Let’s face it. Doing thousands or hundreds of thousands of forecasts takes a long time to do. Especially if they’re manual. At past companies, we’ve seen this process takes several hours and sometimes days. The other thing is that the managers, VPs, and business stakeholders need it done yesterday and run around like it’s a big deal if it’s not done on their arbitrary deadlines.

**Lack of Resources and Personnel**. Several people could be involved in creating forecasts for thousands of stores or SKUs, and it becomes an even bigger challenge if those people need to be quantitative experts.

**Bias and Lack of Accuracy**. Oftentimes, there’s too much manual and human intervention giving “guard rails” to the forecasts with no documentation on why they were put in place. Any form of human intervention leads to what is called “error” in time series forecasts, which is the difference between the actual and the predicted value.

**AutoCatBoostCARMA()** from the [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) packages fixes all of these problems in just a single function (basically, just one line of R code) and would take minutes via GPU or only a few hours via CPU.

What is AutoCatBoostCARMA?

**AutoCatBoostCARMA** is a ***multivariate*** forecasting function from the [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) package in R that leverages the CatBoost gradient boosting algorithm. CARMA stands for **C**alendar, **A**uto**r**egressive, **M**oving **A**verage + time trend. AutoCatBoostCARMA really shines for multivariate time series forecasting. Most time series modeling functions can only build a model on a single series at a time. **AutoCatBoostCARMA can build any number of time series all at once.** You can run it for a single time series, but I have found that **AutoTS()** from [**RemixAutoML**](https://github.com/AdrianAntico/RemixAutoML) will almost always generate more accurate results.

The function **replicates** an ARMA process (autoregressive moving average) in that it will build the model utilizing lags and moving averages off of the target variable. It will then make a one-step ahead forecast, use the forecast value to regenerate the lags and moving averages, forecast the next step, and repeat, for all forecasting steps, just like an ARMA model does. However, there are several other features that the model utilizes.

**The full set of model features includes:**

* 1. Lagged and seasonal lagged features (user defined, does not need to be contiguous)
  2. Moving average and seasonal moving average features (user defined, does not need to be contiguous)
  3. Calendar features from the **CreateCalendarVariables()** function, which include relevant values from weekday, day-of-month, day-of-year, week of year, ISO-week, month, quarter, and year
  4. Time trend feature for capturing a trend component
  5. Automatic target variable transformation, along with the lags and moving averages, where the best transformation is chosen from YeoJohnson, BoxCox, arcsinh, arcsin(sqrt(x)), and logit (the last two are only used for proportion data)
  6. You can also specify to truncate data that is incomplete from the lags and moving averages that are created. Otherwise, the missing values are automatically imputed with a value of -1

AutoTS() versus AutoCatBoostCARMA()

[A recent study](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889) evaluated the performance of many classical and modern machine learning and deep learning methods on a large set of more than 1,000 univariate time series forecasting problems.

The results of this study suggest that simple classical methods, such as ARIMA and exponential smoothing, outperform complex and sophisticated methods, such as decision trees, Multilayer Perceptrons (MLP), and Long Short-Term Memory (LSTM) network models.

We decided to do a similar experiment by comparing **AutoTS** ([also from the RemixAutoML package](https://www.remixinstitute.com/blog/automate-your-kpi-forecasts-with-only-1-line-of-r-code-using-autots)) versus **AutoCatBoostCARMA** on the [Walmart store sales data set from Kaggle](https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data). Here are the overall highlights of that experiment:

**Overall highlights:**

* 1. AutoCatBoostCARMA() had more accurate forecasts than AutoTS() on store / dept’s **41%** of the time
  2. AutoCatBoostCARMA average MAPE by store / dept of **14.1%** vs **12%** for AutoTS
  3. AutoCatBoostCARMA had a lower MAPE than AutoTS by store / dept **47%** of time for the top 100 grossing Walmart store departments
  4. Basically, we found that time series methods will beat machine learning models most of the time but not all of the time

R Code For Running the AutoTS versus AutoCatBoostCARMA Experiment on Walmart Store Sales Data

**NOTE: WE DON’T RECOMMEND YOU RUNNING THIS R CODE UNLESS YOU’VE CONFIGURED A GPU, OTHERWISE IT WILL TAKE 3-5 HOURS. WITH GPU, YOU SHOULD BE ABLE TO COMPLETE ~2,660 FORECASTS IN 15 MINUTES USING AN NVIDIA GeForce 1080 Ti GPU**

library(RemixAutoML)

library(data.table)

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

# Prepare data for AutoTS()----

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

# Load Walmart Data from Remix Institute's Box Account----

data <- data.table::fread("https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv")

# 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: Keep Run Times and AutoTS() Results----

# NOTES:

# 1. SkipModels: run everything

# 2. StepWise: runs way faster this way (cartesian check otherwise, but parallelized)

# 3. TSClean: smooth outliers and do time series imputation

# over the data first

# 4. ModelFreq: algorithmically identify a series frequency to build

# your ts data object

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 = 52,

HoldOutPeriods = 30,

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.R"))

save(OutputList, file = paste0(getwd(),"/OutputList.R"))

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

# Prepare data for AutoCatBoostCARMA()----

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

# Load Walmart Data from Remix Institute's Box Account----

data <- data.table::fread("https://remixinstitute.box.com/shared/static/9kzyttje3kd7l41y1e14to0akwl9vuje.csv")

# 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]

# Run AutoCatBoostCARMA()----

# NOTES:

# 1. GroupVariables get concatenated into a single column but returned back to normal

# 2. Lags and MA\_Periods cover both regular and seasonal so mix it up!

# 3. CalendarVariables:

# seconds, hour, wday, mday, yday, week, isoweek, month, quarter, year

# 4. TimeTrendVariable: 1:nrow(x) by group with 1 being the furthest back in time

# no need for quadratic or beyond since catboost will fit nonlinear relationships

# 5. DataTruncate: TRUE to remove records with imputed values for NA's created by the

# DT\_GDL\_Feature\_Engineering

# 6. SplitRatios - written the way it is to ensure same ratio split as AutoTS()

# 7. TaskType - I use GPU but if you don't have one, set to CPU

# 8. I did not set GridTune to TRUE because I didn't want to wait

# 9. GridEvalMetric and ModelCount only matter if GridTune is TRUE

# 10. NTrees - Yes, I used 15k trees and I could have used more since the best model

# performance utilized all trees (hit upper boundary)

# 11. PartitionType - "timeseries" allows time-based splits by groups IF you have equal sized

# groups within each series ("random" is well, random; "time" is for transactional data)

# 12. Timer - Set to TRUE to get a print out of which forecasting step you are on when the

# function hits that stage

# \*13. TargetTransformation is a new feature. Automatically choose the best transformation for

# the target variable. Tries YeoJohnson, BoxCox, arcsinh, along with

# asin(sqrt(x)) and logit for proportion data

Results <- RemixAutoML::AutoCatBoostCARMA(

data,

TargetColumnName = "Weekly\_Sales",

DateColumnName = "Date",

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

FC\_Periods = 52,

TimeUnit = "week",

TargetTransformation = TRUE,

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

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

CalendarVariables = TRUE,

TimeTrendVariable = TRUE,

DataTruncate = FALSE,

SplitRatios = c(1 - 2\*30/143, 30/143, 30/143),

TaskType = "GPU",

EvalMetric = "MAE",

GridTune = FALSE,

GridEvalMetric = "mae",

ModelCount = 1,

NTrees = 20000,

PartitionType = "timeseries",

Timer = TRUE)

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

Results$TimeSeriesPlot

# Metrics for every store / dept. combo----

# NOTES:

# 1. Can also pull back other AutoCatBoostRegression() info such as

# Variable Importance, Evaluation Plots / BoxPlots, Partial

# Dependence Plots / BoxPlots, etc.

ML\_Results <- Results$ModelInformation$EvaluationMetricsByGroup

# Transformation info:

# ColumnName = Variable Modified

# NethodName = Transformation Method

# Lambda = lambda value for YeoJohnson or BoxCox; NA otherwise

# NormalizedStatistic = pearson statistic

# Note: value of 0.0000 is a filler value for prediction values

# and it's included to show that the correct transformation was done

TransformInfo <- Results$TransformationDetail

# ColumnName MethodName Lambda NormalizedStatistics

# 1: Weekly\_Sales YeoJohnson 0.6341344 532.3125

# 2: Predictions YeoJohnson 0.6341344 0.0000

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

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

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

# Load AutoTS outputs we saved earlier----

load(paste0(getwd(), "/TimerList.R"))

load(paste0(getwd(), "/OutputList.R"))

# 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: format results----

results <- list()

for(i in 1:2660) {

results[[i]] <- tryCatch({

OutputList[[i]]$EvaluationMetrics[1,]},

error = function(x)

return(data.table::data.table(

ModelName = "NONE",

MeanResid = NA,

MeanPercError = NA,

MAPE = NA,

MAE = NA,

MSE = NA,

ID = 0)))

}

# AutoTS() Results----

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

# AutoTS() Model Winners by Count----

print(

data.table::setnames(

Results[, .N, by = "ModelName"][order(-N)],

"N",

"Counts of Winners"))

# ModelName Counts of Winners

# 1: TBATS 556

# 2: TSLM\_TSC 470

# 3: TBATS\_TSC 469

# 4: ARIMA 187

# 5: ARIMA\_TSC 123

# 6: TBATS\_ModelFreq 117

# 7: ARFIMA 86

# 8: NN 74

# 9: ETS 69

# 10: ARIMA\_ModelFreq 68

# 11: NN\_TSC 66

# 12: NN\_ModelFreqTSC 63

# 13: NN\_ModelFreq 60

# 14: ARFIMA\_TSC 52

# 15: ETS\_ModelFreq 51

# 16: TBATS\_ModelFreqTSC 38

# 17: TSLM\_ModelFreqTSC 29

# 18: ARFIMA\_ModelFreqTSC 27

# 19: ETS\_ModelFreqTSC 23

# 20: ARIMA\_ModelFreqTSC 15

# 21: ARFIMA\_ModelFreq 11

# 22: NONE 6

# ModelName Counts of Winners

# AutoTS() Run Times----

User <- data.table::data.table(data.table::transpose(TimerList)[[1]])

data.table::setnames(User,"V1","User")

SystemT <- data.table::data.table(data.table::transpose(TimerList)[[2]])

data.table::setnames(SystemT,"V1","System")

Elapsed <- data.table::data.table(data.table::transpose(TimerList)[[3]])

data.table::setnames(Elapsed,"V1","Elapsed")

Times <- cbind(User, SystemT, Elapsed)

# AutoTS Run time Results----

MeanTimes <- Times[, .(User = sum(User),

System = sum(System),

Elapsed = sum(Elapsed))]

# AutoTS() Run Time In Hours----

print(MeanTimes/60/60)

# User System Elapsed

# 1: 29.43282 0.3135111 33.24209

# AutoTS() Results Preparation----

Results <- cbind(StoreDept, Results)

GroupVariables <- c("Store","Dept")

Results[, eval(GroupVariables) := data.table::tstrsplit(StoreDept, " ")][

, ':=' (StoreDept = NULL, ID = NULL)]

data.table::setcolorder(Results, c(7,8,1:6))

# Merge in AutoCatBoostCARMA() and AutoTS() Results----

FinalResults <- merge(ML\_Results,

Results,

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

all = FALSE)

# Add Indicator Column for AutoCatBoostCARMA() Wins----

FinalResults[, AutoCatBoostCARMA := ifelse(MAPE\_Metric < MAPE, 1, 0)]

# Percentage of AutoCatBoostCARMA() Wins----

print(paste0("AutoCatBoostCARMA() performed better on MAPE values ",

round(

100 \* FinalResults[!is.na(MAPE), mean(AutoCatBoostCARMA)],

1),

"% of the time vs. AutoTS()"))

# [1] "AutoCatBoostCARMA() performed better on MAPE values 41% of the time vs. AutoTS()"

# AutoCatBoostCARMA() Average MAPE by Store and Dept----

print(paste0("AutoCatBoostCARMA() Average MAPE of ",

round(

100 \* FinalResults[!is.na(MAPE), mean(MAPE\_Metric)],

1),

"%"))

# [1] "AutoCatBoostCARMA() Average MAPE of 14.1%"

# AutoTS() Average MAPE by Store and Dept----

print(paste0("AutoTS() Average MAPE of ",

round(

100 \* FinalResults[!is.na(MAPE), mean(MAPE)],

1),

"%"))

# [1] "AutoTS() Average MAPE of 12%"

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

# AutoTS() by top 100 Grossing Departments----

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

temp <- data[, .(Weekly\_Sales = sum(Weekly\_Sales)), by = "GroupVar"][

order(-Weekly\_Sales)][1:100][, "GroupVar"]

GroupVariables <- c("Store","Dept")

temp[, eval(GroupVariables) := data.table::tstrsplit(GroupVar, " ")][

, ':=' (GroupVar = NULL, ID = NULL)]

temp1 <- merge(FinalResults, temp, by = c("Store","Dept"), all = FALSE)

# Percentage of AutoCatBoostCARMA() Wins----

print(paste0("AutoCatBoostCARMA() performed better on MAPE values ",

round(

100 \* temp1[!is.na(MAPE), mean(AutoCatBoostCARMA)],

1),

"% of the time vs. AutoTS()"))

# [1] "AutoCatBoostCARMA() performed better than AutoTS() on MAPE values 47% of the time"

# AutoCatBoostCARMA() Average MAPE by Store and Dept----

print(paste0("AutoCatBoostCARMA() Average MAPE of ",

round(

100 \* temp1[!is.na(MAPE), mean(MAPE\_Metric)],

1),

"%"))

# [1] "AutoCatBoostCARMA() Average MAPE of 5.6%"

# AutoTS() Average MAPE by Store and Dept----

print(paste0("AutoTS() Average MAPE of ",

round(

100 \* temp1[!is.na(MAPE), mean(MAPE)],

1),

"%"))

# [1] "AutoTS() Average MAPE of 5.6%"