Investigating Distributed ARIMA Models for Ultra-long Time Series with Twin Cities Traffic Data

STAT 5825 Final Project

Ben Stockton

December 7, 2020

Better intro sentence. Try "Today,

Introduction

many TS

In the world today, we measure more continuous processes more carefully than ever before. This results in time series that have many observations that exist over very long time spans. These ultra-long time series present challenges to the traditional modeling methodologies since they inherently violate some of the

underlying these methods

s underlying the classical methods, such as constant variance, and even that the same process generates the entire time series. To provide a new solution to these assumption violations, Wang et al developed the Distributed ARIMA (DARIMA) framework (2020).

Make this part of the previous paragraph.

The new DARIMA method allows the data process to vary slowly and continuously over time while still using the entire time series to create a cohesive model that out performs the traditional ARIMA model. DARIMA models offer improvements in forecasting and in computation time, providing a clear and convincing advantage over using the ARIMA model.

data genera

alongside is one word?

For this project, I will use the code provided along side the DARIMA on GitHub (Wang 2020) to replicate their findings on improved performance and decreased computation costs relative to the ARIMA model, and to extend the DARIMA methodology to modeling with exogenous variables. For this purpose, I selected the Metro Interstate Traffic data set from the UCI Machine Learning Repository (*UCI Machine*

Find out when the data was posted to UCI MLR

s data set comes from measurements of hourly westbound traffic volume at Minnesota DoT ATR station 301 on I-95, which lies halfway between St. Paul and Minneapolis, hourly temperatures, holiday indicators, as well as a few weather variables[1]. The measurements were taken from October 2012 through September 2018 for a total length of 48,204.

[1]There appear to be issues with the percipitation measurements. I will discuss this further in the Data section.

To compare the DARIMA model to the traditional ARIMA model, I will use the Mean Absolute Scaled Error (MASE) and Mean Scaled Interval Score (MSIS), as were used in the original Distributed ARIMA paper. I will also follow the lead from Wang et al, and compare those metrics over various sub-series lengths and the compute times for the DARIMA vs ARIMA and the DARIMA at the selected sub-series lengths.

subseries find

The code for the DARIMA framework was posted to GitHub (Wang 2020), but has not been neatly packaged into either a Python nor an R package. At the moment this makes implementing their framework more difficult than usual. The framework requires Spark and is ideally employed on a distributed compute network, which could be set up on the Stats Department cluster, but I haven't taken the introduction "course" to get up to speed on using the cluster[2]. Instead, I set up an inferior single node computer on my personal computer to use Spark for the modeling process. This took a while to configure, but once set up the single node works well enough for the demonstration purposes of this project. My time series is also significantly shorter than the ultra-long time series (n = 124,171) used for demonstration in the DARIMA paper.

distributed

The parenthetic

Discussion of "Distributed ARIMA models for ultra-long t

More information shoul

series" (Wang et al. 2020)

arXiv.org

In the summer of 2020, researchers Xiaoqian Wang, Yanfei Kang, Rob Hyndman, and Feng Li posted their paper, Distributed ARIMA Models for Ultra-long Time Series (2020), to ArXiv in which they develop and demonstrate a methodology for modeling ultra-long time series based on the ARIMA process. Ultra-long time series arise from processes that are observed many, many times over a "long" period of time. Some of the examples discussed by Wang et al. in their paper include hourly electricity demand (which they use to demonstrate their new methodology), daily temperatures over hundreds of years, streaming data that is continuously generated, stock prices recorded hourly over several years (2020, p. 2).

and

The motivating hypothesis behind their research is that the data generating process doesn't remain constant over these ultra-long periods, but rather has small, gradual changes which is essentially constant on small enough time scales (2020, p. 2). To model this shifting behavior, they developed the DARIMA methodology. The DARIMA methodology splits the ultra-long time series into smaller, easier to model sub series to which ARIMA models are fit. The ARIMA estimates for the sub series are then combined together using a combination of the map reduce algorithm and weighted least squares. As demonstrated in their paper and my project, this results in models with better accuracy and reduced computation time compared to fitting a single ARIMA model for the entire ultra-long time series.

subseries here

[2] I could have also used Google Cloud Platform or a similar cloud compute engine, but I cnose not to pay for the compute time.

In their paper, the researchers also discuss alternative methods for modeling ultra-long time series such as discarding early observations and only using later observations for model fitting and forecasting, and the model-free prediction method developed by Das and Politis [3] (Wang et al. 2020, p. 2). As mentioned in the paper, these methods have significant computational costs or require an inefficient use of the available data[4]. The DARIMA approach sidesteps both issues by using distributed computing to accomplish the computing in a more efficient fashion while employing the full data set.

no transition here

Distributed computing requires a network of computing nodes that coordinate with each other to process time consuming computations in a more efficient manner. In this distributed framework, there typically is one "master" node while the others are "worker" nodes[5]. The "master" node coordinates how tasks and how portions of the data get distributed to the "worker" nodes which then do the vast majority of the computation. The outcomes of the computations are then returned to the master node which uses an algorithm, such as distributed least squares (Zhu et al. 2019), to combine the outcomes from the "worker" nodes into a finalized output (Wang et al. 2020, p. 5). There are primary benefits of distributed computing as discussed by Wang et al.. The first is the reduced computational time via parallelization, but requires more computation overall since the independent results must be merged together rather than a single result being produced on a typical single node computer. The second primary benefit is the ability to scale computation by adding "worker" nodes to the network instead of upgrading the existing nodes (Wang et al. 2020, p. 5). A popular technology for working with distributed computing for machine learning and statistical tasks is the Spark system by Apache along with the Hadoop distributed file system (Foundation n.d.).

"master" c

two primary b

produced by a ty

got messed up

no transitionThis may n

MapReduce refers to the process of partitioning the data set for distributed computation and then recombining the outputs into a finalized output. The full data must first be partitioned into independent [illegible] ith a key for identification. These are then passed to the "worker" nodes for computation. [illegible] then produce outputs along with their keys. The "Map" portion of MapReduce acts as a [illegible] g the independent subsets and their keys to the independent outcomes and their keys. Then the "Reduce" portion of the framework takes the independent outcomes and their keys and processes them to reduce them down to a final output (Wang et al. 2020, p. 5)

Maybe this paragraph should repla

As noted by Wang et al., distributed computing does come with limitations for statistical computing, namely the requirement that the portions of the data that are sent to each "worker" node is independent from all other portions of the data since the individual "worker" nodes cannot communicate with each other to process dependencies (Wang et al. 2020, p. 7). This is less of a problem when the observations from the data set are independent of each other, such as in the traditional MLR framework. In the DARIMA paper, Wang

[3]"Predictive inference for locally stationary time series with an application to climate data" (2020)

[4]I can verify that the ARIMA model does come with significant computational costs when fitting to a long time series

[5]In the summer of 2020, the use of this analogy faced renewed scrutiny, albeit under the more odious phrasing as master-slave. The debate over this terminology is ongoing Eglash 2007 and several major technology companies and organizations including GitHub, Python, and Twitter have already or are replacing the terminology in their documentationLandau 2020.

et al., note that there have been a variety of attempts by researchers to implement forecasting with dependent data on distributed computing, but apparently none have tried to incorporate both distributed computing for the model fitting and for the forecasting as they have described and demonstrated (Wang et al. 2020, p. 7).

As we learned in our course, ARIMA models are fairly flexible and can be used to model time series with both time trends and seasonal trends by adding a differencing component into the ARMA process which combines the AR and MA processes (Shumway 2019, p. 99). The ARIMA model is specified as ARIMA(p, d, q) while the seasonal ARIMA is specified as ARIMA$(p, d, q) \times (P, D, Q)_s$. The coefficients p, P specify the order of the AR and seasonal AR processes, q, Q specify the order of the MA and seasonal MA processes, and d, D specify the order of the differencing. s specifies the seaonal period. The model is given below:

use "p and P", '

specify non-seasonal AR a

The seasonal ARIMA model is given b

$$\Phi_P(B^s)\phi(B)(1-B^s)^D(1-B)^d x_t = \alpha + \Theta_Q(B^s)\theta(B)w_t \tag{1}$$

$$\begin{array}{ll} \Phi_P(B^s) = 1 - (\sum_{j=1}^{P} \Phi_j B^{sj}) & \Theta_Q(B^s) = 1 + \sum_{j=1}^{Q} \Theta_j B^{sj} \\ \phi(B) = 1 - (\sum_{j=1}^{p} \phi_j B^j) & \theta(B) = 1 + \sum_{j=1}^{q} \theta_j B^j \end{array}$$

where B is the backshift operator and w_t is a white noise series.

where did phi, Phi, theta, and Theta come from?

As we did in our course, Wang et al., use the **forecast** package's `auto.arima()` function to fit their overall ARIMA model and the localized ARIMA models for each subseries in the DARIMA framework. For the DARIMA framework, they use the stepwise option for the model selection. As described by Hyndman, the algorithm for selecting the most appropriate ARIMA order follows two main steps (Rob J. Hyndman and Yeasmin Khandakar 2008, p. 10-12). The first step is using unit root tests to determine the order of the differencing[6]. The second step in the automatic selection is to use an information criterion (AIC, AICc, BIC) to traverse the space of possible models in a stepwise manner to select the best model given the upper limits on the orders of p, q, P and Q. At each step, we select the model with the lowest information criterion to move forward with and consider the next possible set of models with ± 1 to each of the p, q, P and Q parameters. When no models can beat our current selection, the process ends.

Wang et al., discuss several issues that come along with the automatic ARIMA fitting when in the

ARIMA fitting being a time-consuming process for ultra-long time series,...

these are (1) the unrealistic assumption of a constant data generating process, (2) ARIMA fitting is time-consuming, (3) multiple models can be considered in the model refitting process, (4) Conditional Sum of Squares (CSS) must be used instead of CSS-ML or ML because it is the least inefficient, (5) single node systems may not be able to fit an ARIMA to an ultra-long time series due to resource limitations, and (6) the upper limit on the orders restricts the

[6]KPSS unit root test for the non-seasonal differencing and Canova-Hansen unit root test for the seasonal differencing. D is selected before d.

potential models that could be fit.

The preceding paragraph should probably be merged with the one before it.

No transition The DARIMA modeling framework consists of five main steps: prepocessing, modeling, linear transformation, estimator combination, and forecasting (Wang et al. 2020, p. 13-14). The preprocessing step subdivides the the full time series into the K subseries that will be modeled individually and these subseries are passed onto the "worker" nodes. In the modeling step, the "worker" nodes fit the `auto.arima()` model to the subseries. Then the linear transformation step converts the ARIMA representations of the fitted models into AR infinity transformations (which are actually AR(2000) processes) (Wang et al. 2020, p. 16). This completes the Map portion of the MapReduce framework. The Reduce portion then picks up with the estimator combination step, which uses distributed weighted least squares to combine the estimates from each subseries into estimates for the entire time series in a final AR(2000) process. This process is then used for forecasting completing the DARIMA framework. This process is explained in detail in Figures 3 & 4 and Algorithms 1 & 2 in the DARIMA paper (Wang et al. 2020, p. 13-15).

end senten

AR(\infty) representations

via invertibili

they're estimates

global estimates

The invertibility of ARMA processes is well-established and we covered the topic in our course and in *Time Series: A Data Analysis Approach Using R* (2019, p. 73). The time series x_t is invertible if the roots of its $\phi(B)$ polynomial have modulus greater than 1. In the ARMA context this means that the process:

$$\phi(B)(x_t - \mu) = \theta(B) w_t \tag{2}$$

can be expressed in the linear representation

AR(\infty) linear representation

$$w_t = \sum_{j=0}^{\infty} \pi_j x_{t-j} = \pi(B) x_t = \theta^{-1}(B)\phi(B) x_t \tag{3}$$

where $\sum_{j=0}^{\infty} \pi^2 < \infty$.

Wang et al. extend invertibility to seasonal ARIMA models by equating the seasonal ARIMA (1) to an ARMA model (2) by setting $\phi'(B) = \phi(B)\Phi(B)$ and $\theta'(B) = \theta(B)\Theta(B)$ where $\phi(B) = (1 - \sum_{j=1}^{p} \phi_j B^j)(1-B)^d$ and $\Phi(B) = (1 - \sum_{j=1}^{P} \Phi_i B^j)(1-B)^D$ and $\theta(B)$ and $\Theta(B)$ have the typical definitions (2020, p. 16-17). They describe this re-expressed process as a ARMA(u, v) process where $u = p + P$ and $v = q + Q$. Having re-expressed the process as an ARMA, it is a simple matter to transform it to the linear representation (3) where $\pi(B) = \theta'^{-1}(B)\phi'(B)$ to which a linear trend is added[7] (Wang et al. 2020, p. 17). They also take the extra step of defining an approximation AR$(p >> 1)$ to the AR(∞) representation

(eqn. (1))(

while instead of and

ARMA pro

straightforward n

AR(p(>>1))

$$x_t = \beta_0 + \beta_1 t + \sum_{j=1}^{p} \pi_j x_{t-j} + \epsilon_t \tag{4}$$

These AR(p) representations are fit to each of the subseries and will then be combined for global estimates for the whole time series in a global AR(p) process. As Wang et al. describe, the linear representation is used

[7]The linear trend term takes care of the non-stationarity of the time series. The differencing terms of the ARIMA model are absorbed into β_0 and β_1.

because the DLSA method requires that each subseries model estimation must have the same form (2020, p. 18).

use distributed least squares approximation here

The researchers selected Zhu's Distributed Least Squares Approximation (DLSA) for their combination method to estimate the global parameters (Wang et al. 2020, p. 18)(Zhu et al. 2019). The DLSA method takes the subseries estimates and calculates a weighted average of the estimates to create the global estimates. To calculate the weighted average, Wang et al., minimize a global loss function which is the mean of the subseries loss functions. The global estimate for the AR(p) parameters is a weighted average of the minimizers of the local loss functions and the estimated covariance matrices of the observations of each subseries

To calculate the

$$\tilde{\theta} = \left(\sum_{k=1}^{K} T_k \hat{\Sigma}_k^{-1}\right)^{-1} \left(\sum_{k=1}^{K} T_k \hat{\Sigma}_k^{-1} \hat{\theta}_k\right).$$

estimates

The global covariance of the observations estimated as

$$\tilde{\Sigma} = T \left(\sum_{k=1}^{K} T_k \hat{\Sigma}_k^{-1}\right)^{-1}$$

where K is the number of subseries, T_k is the length, $\hat{\Sigma}_k$ is the covariance estimate of the observations, and the vector $\hat{\theta}_k$ are the estimates of the AR(p) process of the k^{th} subseries. cite here too

Point forecasts for the DARIMA process are straightforwardly computed using the global estimates of the AR(p) process, $\tilde{\theta}$ (Wang et al. 2020, p. 19-20).

$$\hat{y}_T(h) = \tilde{\beta}_0 + \tilde{\beta}_1(T+h) + \begin{cases} \sum_{j=1}^{p} \tilde{\pi}_j y_{T+1-j}, & h = 1 \\ \sum_{j=1}^{h-1} \tilde{\pi}_j y_T(h-j) + \sum_{j=1}^{p} \tilde{\pi}_j y_{T+h-j}, & 1 < h < p \\ \sum_{j=1}^{p} \tilde{\pi}_j y_T(h-j), & h \geq p \end{cases}$$

Similarly, the prediction intervals for the DARIMA process are constructed similarly to the prediction interavls for the ARMA process as discussed in Shumway & Stoffer (2019, p. 93-94) (Wang et al. 2020, p. 20-21). The prediction interval of the h-step ahead forecast is

$$\hat{y}_T(h) \pm \Phi^{-1}(1-\alpha/2)\tilde{\sigma}_h$$

where the global estimate of the standard error is

$$\tilde{\sigma}^2 = \begin{cases} \tilde{\sigma}^2, & h = 1 \\ \tilde{\sigma}^2(1 + \sum_{j=1}^{h-1} \tilde{\theta}_j^2), & h > 1 \end{cases}$$

and $\Phi^{-1}(\alpha)$ is the standard normal inverse CDF.

no transition

To demonstrate their new DARIMA framework, the researchers chose to model the publicly available hourly electricity demands of New England (Massachussets, Connecticut, Rhode Island, Vermont, New

goes in a footnote

Hampshire, and Maine) spanning March 1, 2003 to April 30, 2017[8] (Wang et al. 2020, p. 21). This data set contains eight time series of electricity demands: one for each of CT, RI, VT, NH, and ME, and three for northeastern, western-central, and southeastern Massachussets. There are also weather and holiday variables included, but these were ignored for the purposes of their paper. They chose to split their data such that they would have four months of demands for forecast evaluation.

In their experimental set up, they chose to have a baseline of 150 sub-series for their DARIMA model, resulting in sub-series lengths of around 800. They then set parameters along which the sub-series' models and the comparison ARIMA model should be fit using the auto.arima() function from the **forecast** package in R (Wang et al. 2020, p. 23). They set max values for p and q at 5, max values for P and Q at 2, the max order $(p+q+P+Q)$ at 5, and then fit the models with Conditional Sum of Squares (CSS). For their ARIMA model, they fit the model with parallelization enabled to get the best possible computation time, but this also requires disabling stepwise selection. Conversely, the DARIMA sub-series models are selected using stepwise selection, but without parallelization. The models are then evaluated using MASE and MSIS (with 95% prediction intervals)[9]. Their computing environment differs drastically from mine. They used one master node and two worker nodes, which each had 32 logical cores, 64 GB RAM, and 160 GB total SSD storage. My set up uses a single computer node with 4 cores, 16 GB RAM, and 256 GB total SSD storage.

and stepwise se

From their modeling, they demonstrate superior performance by the DARIMA method across the board (Wang et al. 2020, p. 25-31). The DARIMA method produces shorter compute times, lower MASE and MSIS overall, and as lower or lower MASE and MSIS at nearly all forecast horizons. They also note that the performance gap increases as the forecast horizon increases and as the confidence level of the prediction interval increases from 50% to 95%. To conclude their experiment, they also compared the DARIMA method to itself when different numbers of sub-series are used and when the maximum model order is changed. They found that the DARIMA method performed best when the number of sub-series was between 150 and 200, for a sub-series length of around 800. Increasing the maximum model order resulted in marignally higher MASE and MSIS values for the DARIMA model while slightly decreasing them for the ARIMA models. The worst performing DARIMA model out performed the best performing ARIMA model in both metrics. The more significant point here is that increasing the model order from (5,2,5) to (8,4,10) almost doubled the DARIMA compute time from 1.219 minutes to 2.224 while it increased the ARIMA compute time by more than 20-fold from 4.596 to 117.875.

20x

In my data analysis, (comma)

In my data analysis I will try to replicate their findings regarding increased computing efficiency and better performing models compared ARIMA. I will also try to extend the method to include ARIMA modeling with exogenous regression variables.

better performance

[8] Roach 2020

[9] See appendix for the formulas for MASE and MSIS.

Make this section past tense.

Data

Analysis

Discussion

Conclusion

References

Das, S., & Politis, D. N. (2020). Predictive Inference for Locally Stationary Time Series With an Application to Climate Data. *Journal of the American Statistical Association*, 1–16. https://doi.org/10.1080/01621459.2019.1708368

Eglash, R. (2007). Broken Metaphor: The Master-Slave Analogy in Technical Literature [Publisher: [The Johns Hopkins University Press, Society for the History of Technology]]. *Technology and Culture*, *48*(2), 360–369. Retrieved December 6, 2020, from http://www.jstor.org/stable/40061475

Foundation, A. S. (n.d.). Apache Spark™ - Unified Analytics Engine for Big Data. Retrieved December 6, 2020, from https://spark.apache.org/

Landau, E. (2020). Tech Confronts Its Use of the Labels 'Master' and 'Slave'. *Wired*. Retrieved December 6, 2020, from https://www.wired.com/story/tech-confronts-use-labels-master-slave/

Roach, C. (2020). camroach87/gefcom2017data [original-date: 2018-12-13T01:46:20Z]. Retrieved December 6, 2020, from https://github.com/camroach87/gefcom2017data

Rob J. Hyndman, & Yeasmin Khandakar. (2008). Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, *27*(3), 1–22. https://doi.org/10.18637/jss.v027.i03

Shumway, R. H. (2019). *Time Series: A Data Analysis Approach Using R* (1st). CRC Press, Taylor & Francis Group.

UCI Machine Learning Repository. (n.d.). Retrieved November 30, 2020, from https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume#

Wang, X. (2020). xqnwang/darima [original-date: 2020-06-30T07:33:48Z]. Retrieved December 1, 2020, from https://github.com/xqnwang/darima

Wang, X., Kang, Y., Hyndman, R. J., & Li, F. (2020). Distributed ARIMA Models for Ultra-long Time Series [arXiv: 2007.09577]. *arXiv:2007.09577 [stat]*. Retrieved November 29, 2020, from http://arxiv.org/abs/2007.09577

Zhu, X., Li, F., & Wang, H. (2019). Least Squares Approximation for a Distributed System [arXiv: 1908.04904]. *arXiv:1908.04904 [cs, stat]*. Retrieved December 6, 2020, from http://arxiv.org/abs/1908.04904

Appendix

Formula for MASE:

$$MASE = \frac{\frac{1}{H}\sum_{t=T+1}^{T+H}|y_t - \hat{y}_{t|T}|}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|}$$

Formula for MSIS:

$$MSIS = \frac{\frac{1}{H}\sum_{t=T+1}^{T+H}\left[(U_{t|T} - L_{t|T}) + \frac{2}{\alpha}(L_{t|T} - y_t)I(y_t < L_{t|T}) + \frac{2}{\alpha}(y_t - U_{t|T})I(y_t > U_{t|T})\right]}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|}$$