A weighting algorithm for time-series models – a first draft /work in progress report

How this document is structured:
Section one gives the overview of what issues I see with currently popular time-series prediction algorithms for small datasets. Section 2 provides the details of the algorithm – you may want to skip directly to that part or start reading it first. Finally, section 3 provides a very brief discussion about extending this modification to algorithms other than SARIMA.

(1) The background

For traditional time-series prediction algorithms such as (S)ARIMA, but also other popular predictive algorithms such as gradient boosting models, I found that once characteristics of a time series change, we're left with limited options to address that change. Some of those are simply not using the older part of a dataset which is not as relevant to future predictions anymore, or somehow semi-manually assigning relative weights to observations. Further, I thought about that while training(estimating) parameters on the time series, we often effectively use more recent observations to estimate parameters of a time series further away in the past, which seemed to me as an illogical approach. The algorithm proposed in section 2 addresses the first of these concerns, and partially resolves the second issue as well, in part reformulating the traditional approach for minimizing a loss function for a time series problem to that it is 'forward looking'

Two main existing concepts/algorithms inspired me here, which are node centrality measures from Network Theory and forget cells in recurrent neural networks. In particular, Katz Centrality, as a measure of node centrality contains a (hyper)parameter alpha, considering not only the set of nodes directly connected to another node to determine it's centrality, but also connections to this directly connected set of nodes, which become gradually less important as we move further away from the node (increase the minimum number of steps needed to get to it). This parameter is called attenuation factor and was the main inspiration for this algorithm, so I call the weighting vector I compute attenuation factor as well.
Secondly, with memory cells in RNNs, a neural network is able to learn to systematically 'forget' past data that is no longer relevant for future predictions, increasing the network's flexibility. Inspired by this, I was looking to create an analogous method of lower complexity that can be applied to small datasets (on which a neural network typically wouldn't perform very well).

(2) – The algorithm

There are two main ways I see the algorithm adding value to popular time-series prediction algorithms such as SARIMA:

1 - this modified model can learn to 'forget' past components of a time series when they're no longer relevant. For instance, when predicting sales for a gas station with two competitors, the time series for sales will change after one of the competitors drops out. Rather than solving this problem by estimating the parameters (of SARIMA) on only the portion of the dataset that includes the reaction of gas station sales to this important event, i.e. deleting the data from before it, it might be advantageous to have a mechanism in place for the algorithm to determine this on its own. I see two main reasons for this:

- Not all changes in business conditions for the gas station are this easily identifiable
- It might be the case that the original idea - the deletion of the part of the gas station sales time series might be a prudent solution for one of the features of the model, such as the number of customers, which is likely to both shift and behave differently, but not for others, such as advertising costs, which are left unaffected. So if both advertising costs and number of customers are important for the model, we're left with a tradeoff between deleting the portion of the dataset that is no longer relevant and brings unnecessary noise to the model, and deleting useful information provided by the other feature.

2 - the modified model progressively denoises the data. If we formulate our problem in the way I'm proposing, we can imagine noise in data being very similar to an outdated portion of a dataset – something that no longer contributes with predictive power. Additionally, a datapoint might be considered a useful observation at one point, and to be bringing unnecessary noise to the dataset later on, when the characteristics of the time series change. As it will hopefully become clear later, my solution is 'dynamic' in this respect – it will keep the datapoint roughly until it is useful for future predictions and deweight/remove it later on.

The Algorithm (explained as a modification to the traditional OLS estimation):

I aim to train a high degree polynomial that transforms the predictions space X*B, where X is the feature matrix of a model (every row being one observation at a point in time) and B is the vector of parameters to estimate. We would ordinarily minimize a loss function L(X*B,y) by choosing the B vector, where y is the regressand and L captures the difference between estimates and target data. If we aim to minimize the L2 norm (sum of squared residuals):

$$L = \sum_{n=1}^{N}(X * B - y)\text{^}2$$

Stage 1:

For a time series of any (arbitrary) length, determine the minimum amount of datapoints necessary to fit your model. Let's say for now that this number is 35. Also construct a simple time index **t** – a vector which gives an observation number for every corresponding observation(row) in matrix X

- Take the points from 36 to 70 and minimize the loss function $L = \sum_{n=1}^{N}(X * B - y)\text{^}2$ on them, yielding the 'optimal' value vector $\boldsymbol{B_{optimal}}$
- Take the previous 35 datapoints, with $\boldsymbol{B_{optimal}}$ already determined, and minimize $L' = \sum_{n=1}^{N}(X * \boldsymbol{Boptimal} * \mathbf{a} - y)\text{^}2$, where $\mathbf{a} = \sum_{i=0}^{m}(c * t\text{^}i)$, with respect to vector **c**, which gives the coefficients for an m-th degree polynomial.

Due to the slightly more complex nature of this minimization, I used stochastic gradient descent to minimize the function. This yields values for the attenuation vector **a** - by the factor of how much did a particular datapoint (with corresponding entry in **a**) need to be transformed to achieve the minimal error for future predictions. This is the reason why we estimate $\boldsymbol{B}_{optimal}$ on 'future' data first – it is to see how important the past points are to future errors- something that I attempt to make a proxy for the generalization error (error on unseen data).

Stage 2:

We want to learn from the attenuation vector **a**, which aims to estimate the importance of every point in the previous series. The more different the value of an entry in this vector is from 1, the more we needed to transform the dot product of that observation with the parameter vector. At the same time, we don't want to give the attenuation vector too much influence on all future predictions, since this may progressively change. Thus, a hyperparameter **p** (pace) will be chosen and will be used to reweight the first datapoints by **p***(1/abs(**a**)). Note that if a time series wouldn't need this reweighting, all entries of **a** would be equal to one. Choosing p to be any constant value across time, this process would generate the original (unmodified) model, since relative observation weights would remain the same. I take the reciprocal of the absolute value of **a**, since the direction in which a point was transformed by the attenuation factor doesn't matter, only the size of the required shift.

Stage 3:

Now our dataset consists of the 35 reweighted points X***p***(1/1-**a**) and 35 unweighted points (points from 36 to 70). This gives us a space that is partially reweighted, and the reweighting reflects how much (the first half of) datapoints contribute to the future generalization error. The idea is that we can now use this modified dataset to generate better forecasts, as it partially takes into account how much each of the data points contributes to the future generalization error. This is what I meant by wanting the optimization to be 'forward-looking' in section 1.

If the task is to provide a model of this kind every day, we can now estimate the linear model on this modified space X***p***(1/1-**a**) tomorrow, with the benefit of having a 'forward looking' minimization function. The next day, a new data point will be generated and we can increase the sample size for estimating our parameters by one, with estimating the attenuation factor always on the last few (35 in this example) datapoints. As we progress through time, some of the points will be partially or fully forgotten (having corresponding entries in vector a close to 0), thus achieving the first objective I set out. It simultaneously achieves point 2 as well, since noise in the time series can be thought of as something that has no future predictive power, so if the entry $\mathbf{a}_i$ in the vector **a** is zero, and we can see this happening in a stable way in the future (retraining the model as new data becomes available, keeping track of $\mathbf{a}_i$), we have successfully forgotten the ith row of feature matrix X.

As we move across time, I hypothesize that we can expect the generalization error to decrease further, as a larger proportion of our dataset can be reweighted. Only in the beginning does one half of the dataset need to be left unweighted, due to there being a minimum size of a dataset to estimate our model. So if we have a time series of 315 datapoints, we still only need to take 35

points to estimate $\boldsymbol{B}_{optimal}$ and are left with 280 weighted datapoints, which makes roughly 89% of the dataset.

(3) - Final thoughts

This idea can be also applied in an analogous way on the feature level (determining the attenuation factor for every predictor separately), further decreasing the expected generalization error and meeting the second part of our goal 2. This being said, we would then need to estimate a separate attenuation vector for every feature. Although this idea is described here in the context of a regression model such as SARIMA, it can be applied in an analogous way to other algorithms, such as XGBoost. While the loss function to be minimized there is different, the same principle of dividing the dataset and estimating the attenuation vector can be applied.