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 it’s 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.

2 - the modified model progressively denoises the data. 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. This is partially treated by the modification I propose as well.

The Algorithm:

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 in time) and B is the vector of parameters to estimate. For a SARIMA model, we would 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 == SUMi,n(X\*B-y)^2

In the modified algorithm, for the first stage, we’d minimize:

L’ == SUMi,n(X\*B\*a-y)^2

, where a == SUMi,n (c\*t^i)

,with respect to the vector of coefficients B, but also the coefficients of the high degree polynomial we’ve chosen to be a function of time index t (t=1 for the first observation, t=2 for the second one, etc.). I wanted to make a parallel here with measures of node centrality from network theory (such as Katz Centrality), so I call this additional component the attenuation factor a. For every point in time, it t tells us how much we need to transform the X\*B space to achieve a loss function of 0. The datapoint that perfectly contributes to the model would have an attenuation factor of 1, i.e. would be left unchanged.

Stage 2:

We estimate params in a normal way

Estimate the next few points

Calc generalization error

Transform space to give the ‘correct’ parameters

Reweight

Do again till end

Stage 1:

Determine the minimum amount of datapoints necessary to fit your model. Let’s say for now that this number is 35.

* Take the points from 36 to 70 and minimize the loss function L == SUMi,n(X\*B-y)^2 on them, yielding the optimal value vector B\_optimal
* Take the previous 35 datapoints, with B\_optimal already determined,and minimize

L’ == SUMi,n(X\*B\_optimal\*a-y)^2

, where a == SUMi,n (c\*t^i)

, with respect to the vector c

This yields values for the attenuation factor a - by the factor of how much did a particular datapoint (with corresponding entry in the vector a) need to be transformed to achieve a the minimum generalization error.

Stage 2:

We want to learn from the attenuation vector a, which aims to estimate the importance of every point in the previous series. At the same time, we don’t want to give the 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)). 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 shift. The more an entry of a is different from 1, the more error the corresponding point generated. So the points that required the greatest transformation by the polynomial(which gives the attenuation factor) will be given the least weight.

Stage 3:

Now our dataset consists of the 35 reweighted points X\*p\*(1/1-a) and 25 weighted 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. If the task is to provide a model of this kind every day, we can now estimate the linear model on this modified space 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 the attenuation factor by one. 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 corresponding entry in the vector a =0, and this is happens in a stable way over many iterations of this algorithm, we have successfully denoised the dataset.

As we move across time, we can expect the generalization error to decreasefurther, as a larger proportion of our dataset will 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 B\_optimal and are left with 280 weighted datapoints, which makes roughly 89% of the dataset.

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.