MA 751 Final Project

Michael Djaballah

April 2020

1 Introduction

When time is a factor for data, it can be leveraged to find significantly more complex relationships in the data than without. The process of building a model with this data, however, requires more care. When time is a factor, the process of cross-validation (CV) is more complicated because only so much of the data is available at a given time. Thus, a random split of the data does not suffice and can cause something called "data leakage". This occurs when information about the target prediction value (either classification or regression) is inadvertently given to the model. In a random CV split, this occurs when a test value happens before a trained value. Since the outcome of the test value is a trained value, the model is given more information than a real life inference test would give. As a result, method of splitting where all train observations take place before test observations. This is called *forward chaining*. This issue of CV with time series has been solved, but other problems still remain.

A lot of time series models are statistically rigorous but are not aimed for machine learning. In other words they are mathematically sound, but may not have the same emphasis on prediction ability that a discipline like machine learning demands. To this end, there are time series models which focus on prediction ability such as recurrent neural networks (RNNs). RNNs also have their pitfalls. They are relatively hard to interpret with or without expert knowledge and have a tendency towards overfitting as a result of their complexity. Other, more statistical, machine learning models close these gaps by having regularization and being more interpretable. These models, such as ridge and lasso regression, regression trees, and random forests are lacking the time dependency aspect.

2 Problem and Data

To incorporate time into these models, set the response to be the next observation in time of the feature that you wish to predict. This method, however, leads to shallow associations analyzed between observations because of the memory of the model having only one observation. Expanding to more observations allows more depth, but loses the relationship of observations to each other and cause overfitting due to the high number of features: given data X with n observations, p features, and a target y for each observation, each memory observation adds p features and takes an observation away. For a memory of one observation, we have p features and $n-1$ observations because the final observation was removed due to not having a successor. We can add observations in this way, let m be the size of the model's memory ($m \geq 1$): $p * m$ features and $n - m$ observations. This large amount of features as m grows and the fewer observations to train on can quickly lead to overfitting. This is the core issue.

The data can be manipulated to tap into the power of these models, but issues still persist. This overfitting can be remedied by using a regularization method, but the time aspect may be obstructed by the high

amount of features. This method does not differentiate between observations, it is simply a pool of features with no sense of time distance from the desired response. Aside from creating a model that takes in tensor (2 dimensional input for a single value output) input which would differentiate between observations, I propose a solution that does not require an edit to the model besides regularization.

The data being used is cryptocurrency price data scraped by me. It consists of over 1.7 million observations of over 200 cryptocurrencies. The data is very high resolution (5 minute intervals) and the response variable is the next price observation. I will also limit it to the most popular cryptocurrency, Bitcoin (BTC).

3 Models and Methods

In order to preserve the time distance inside of the models memory, we will use the parameter reducing aspect of regularization. To describe the process with ridge regression: standardize the data (centering the data and scaling to unit variance) and impose a penalty:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} ||y - X\beta||_2^2 + \lambda ||\beta||_2^2$$

The penalty prevents coefficient from becoming too large and the normalization ensures that all features are on a level playing field to predict the response. This is where I look to take advantage.

Within the memory of model, it is possible to simply increase the penalty λ to reduce overfitting, but this still leaves the memory as a pool of features without relationship to one another. It also has some features present being from the same original feature without any sort sense of time. I propose a secondary regularization technique that is applied before training: weighing the observations.

The general concept is simple, let s be the series creation function from X such that $T = s(X)$ and let m be the size of memory so T is $(n - m)$ x $(m * p)$ and y is $(n - m)$ x 1. Let α be a weighing such that α is m by 1 and α_i is the weight assigned to the ith observation in memory. In order to not undermine the normalization, we will constrain $0 < \alpha_i \leq 1$, $\forall i$. It is also suggested to let the sequence $\alpha_1, \alpha_2, \alpha_3, ...\alpha_m$ be descending.

The idea behind this method is that the weighting combined with the penalty will reduce the power of past observations to predict the next. With a larger memory, applying these weights after normalizing causes the penalty parameter to impact past observations higher than more recent ones. Our new ridge looks like this:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^{p*m}}{\arg\min} ||y - \boldsymbol{\alpha}(s(X))\beta||_2^2 + \lambda ||\beta||_2^2$$

Where $\boldsymbol{\alpha}$ is the weighing function.

which is $(m * p)$ by $(m * p)$ with each column having m of each α in order. This accomplishes the desired weighting. $\alpha(\beta)$ actually gives us the *pseudo coefficients* when a penalty is not applied because it "undoes" the weighting. Lower weighted things will have a higher coefficient and applying α does this. With a penalty, however, the coefficients are constrained, and the model values more recent observations.

The methods for selecting our α can vary widely. One of the simplest is choosing some value $0 < d < 1$ and subtracting it for each consecutive observation, ensuring that it does not violate the lower bound of α_i. This is linearly decaying dependence. The slope of this dependence (d) can be as steep or as flat within the bounds of 0 and 1. Since it is linear, each observation is not that much less important than the last. This allows further our observations to have a significant yet decreasing weight on the predicted response.

A variation on linearly decaying dependence is now we subtract by a larger and larger multiple of d. This causes the dependence to fall of in a quadratic manner. It is important to choose a d which does not violate

the lower bound of α_i. This allows closer observations to matter almost as much as the closest (which always has weight 1) while minimizing the effect of very far away observations.

Another form of dependence is exponentially decaying dependence. This involves d as previously defined, but instead of subtracting d at each iteration, you multiply by d at each iteration. This decays exponentially and so does the effect of past observations on the predicted response. It also guarantees that for any d we will get a valid weight assignment (unless it becomes too small to remain numerically stable).

The final method that will be discussed here involves autocorrelation. Autocorrelation is a function that gives the correlation of a series to itself at a given lag time. Correlation lies between 1 and -1 and works well for our purposes. Only the sign needs to be changed. We could allow for negative α values in this context, but it would only effect the coefficient β, so we remain within our parameters and require the absolute value. Given an autocorrelation function f, such a weighting looks like: $[|f(0)|, |f(1)|, ...|f(m)|]$. Notice that the first element is the autocorrelation with no lag. This accomplishes our weighting of the closest observation being 1.

4 Analysis

Each of the previous methods proposed was run on around ten thousand observations. Series were created with memory ten, twenty, and thirty. This was to begin to push the boundaries of normal series creation methods. All methods of α creation were tested, including a control where all α were equal to 1. The series were split into 80% train and 20% test. An MSE was found for each:

Memory	None	Linear	Quadratic	Exponential	Auto Corr
10	1.62e-4	9.15e-2	9.15e-2	3.02e-1	2.97e-1
20	1.67e-4	4.08e-6	5.50e-8	3.19e-6	2.13e-5
30	1.70e-4	4.92e-7	5.79e-5	1.54e-6	2.72e-5
40	1.72e-4	3.87e-6	3.88e-6	1.59e-6	7.67e-5

These results fall directly in line with what was expected with this weighting technique. The accuracy of all weighting methods increased as memory size increased and most methods eventually levelled off or even became worse. At a memory of 10, the control method of equal weighting severely outpaced all other methods, but as the size increased, we could see what is either continuous slight overfitting or results that are not statistically significant. This contrasts with the weighting methods that only seem to overfit deep into memory expansion. The linear and quadratic also seemed to be the most erratic and this makes sense. Autocorrelation and exponential did not need to be changed for this experiment since the same value works for exponential and autocorrelation has nothing to tune. With other methods, it is necessary to recheck to ensure that our values are above 0.

5 Conclusion and Discussion

This method proves that there are more ways to use regularization, especially in analyzing data with more intricate interactions. The results show a clear improvement over themselves as more memory is desired and a clear advantage over non weighting. It is important to remember that there was no feature extraction and both methods were tested on the exact same data and split. It is clear that with no weighting, the ridge regression overfit on data that did not contribute to the target value, and this showed in its performance when memory was increased.

There is also much to be developed with such a weighting scheme. The generation of α has also yet to be fully realized. This parameter (which it ultimately is) has many degrees of freedom. Perhaps removing our constraint of being between 0 and 1 would allow for even more flexible models. Negative values are present in the autocorrelation and we make them positive, perhaps there is greater power in allowing these weights to be negative. Even working where more than one weighting is applied is possible, such as data that has two axes of dependence (spatial). Such application to spatial data could use euclidean distance to be a measure of weighting instead of just giving the coordinates. More work in time series is also necessary where more complex α generation is certainly possible.

6 Code

For all code used in this project, see my Github.