Write Up for Forecast Conditional Factor Model

(note: the title and name for this model is not great)

Starting with Fama and French’s three factor model (Fama and French 1993), there is an idea that certain qualities in equities are rewarded or penalized in the stock market. For example, companies with low market capitalization have increased average excess return perhaps because large institutions cannot buy these assets. Since the original Fama and French paper, more than 300 factors have been developed (Cochrane, 2011). The presence of so many mispricings has led to thinking of ways to algorithmically construct portfolios of abnormally returning stocks using stock market data and statistical tools.

The dominant way of doing this in the asset pricing literature is Principal Components Analysis (PCA). The factors derived are constructed without any relationship to the quantity they should be predicting, i.e. next periods excess return, Sharpe ratio, and other predictive performance indicators. Instead PCA of a set of data is constructed in an unsupervised manner to maximize coverage of its own variance. However, from a statistical perspective, building factors that are guided by predicting its own variance should have less predictive accuracy than constructing factors guided by trying to predict whatever you want to predict, be that 1 period ahead returns, future Sharpe ration, etc.

In fact, before my PhD, I did a study about using big data predictors at this link: <https://quantonomics.wordpress.com/2017/10/22/embedding-layers-autoencoders-and-high-dimensional-forecasting/> (it’s not a great academic piece of work and not necessary to read but it will highlight my point). This article aims to perform forecasting when the number of covariates is large compared to the number of time steps. It’s not exactly the factor regression problem, but there are similarities. The point is dimensionality reduction must be done. When comparing PCA regression (i.e. predicting by regressing with factors from PCA models) with other big data techniques like LASSO and Ridge regression, PCA regression performs the worst by far. The intuition why this is, is because the factors aren’t constructed with maximizing forecast information, they are contrasted with the intention of maximizing reconstruction variance-covariance. In this task, the RMSE of the three techniques are LASSO: 0.00257, Ridge: 0.00223 and PCA: 0.00346. PCA regression is about 50% worse than the other big data techniques. Now LASSO and Ridge don’t produce factors, so even though they have improved performance, they cannot be used in factor analysis asset pricing exercise. However, the rest of the above article is devoted to pointing out that if you construct a supervised form of PCA, where the construction of the factors is also directly optimized by the data you wish to predict, you will have more success.

With PCA, one takes data and attempts to find the components that provide the most information on how to reconstruct the original input data. However, one really should construct factors so that they provide the most information on how to construct the data you would like to forecast which in asset pricing could be taking the current set of returns and predicting the one step ahead return. Ultimately, using this logic in what I term Supervised PCA, one gets a RMSE of 0.00224, in line with the Ridge and LASSO regressions. I imagine there is some Sharpe ratio type theory on why matching its own variance may forecast future returns better in the asset pricing literature than the generic big data picture, so perhaps the difference in improvement will be less stark than this. However, it seems likely that moving from an unsupervised PCA approach to a supervised PCA approach will yield increases in predictive accuracy, as predicting itself certainly shouldn’t be a stronger predictor than using the object you wish to predict to construct the factors. Of course, one should still do this out of sample and evaluate the factors on both returns data and one step ahead data that hasn’t been used to estimate the model.

What is supervised PCA and how can one construct such an object? First, I will introduce the autoencoder, which is a deep learning tool used for unsupervised learning. It is similar and as I will show, at times, identical to PCA. Here is a picture of an autoencoder:

A diagram of a network

Description automatically generated

From: Gu, Kelly and Xiu 2021

An autoencoder is a particular type of neural network. A neural network is, in its simplest form, a recursive stack of basis functions one on top of another. For example, given an input, , the first layer of a neural network is , where is a matrix and is a vector and they both map the dimensionality of to . Likewise, a second layer would be , and you can have , , and so on to build your network. However, if is the output layer, then would be the only hidden layer. is a nonlinearity or link function. It is important because if one didn’t have it, writing recursively: . In this case is only a linear function of because matrices are closed under multiplication. Thus, the addition of which is some nonlinear transformation (a line with a kink in it, a logistic transform, or many other transformations) gives the neural network the ability not only to approximate nonlinear functions, but, due to results in the literature, to approximate any continuous function arbitrarily well (Hornik et. al. 1989).

Let’s get back to the autoencoder. I’ll give this autoencoder two layers and . is the output. Despite cautioning against it, I’ll make everything linear: and . I’ll also define as the reconstruction term (ie ). Obviously, this is not interesting if has the same or larger dimensionality as , as it can store all the information of to reconstruct at . However, if the dimensionality of is less than , the autoencoder must figure out the information that is most important to save in , so that given this bottleneck, it can reconstruct as as efficiently as possible.

It turns out that if you use squared loss and is d dimensional, and that is equal to or less in dimensionality compared to , the optimal values will learn, span the first d PCA principal components (see here: <https://pvirie.wordpress.com/2016/03/29/linear-autoencoders-do-pca/>). A linear autoencoder described in this way is analogous to performing PCA. Looking at equation 1 in <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4344837> (Bryzgalova, Svetlana, et al., 2023):

A group of letters and numbers

Description automatically generated

Here is the factor loading equation. I’ll assume, for now and for illustration, that R is not some variable you want to predict but is the returns of the assets used to create the factors, F. Then in this case in this equation, F is the factors, is the loadings and e are the errors. Since F is constructed out of R, then F is analogous to in the autoencoder description above and maps to (and ). Likewise, F is constructed in this way . Thus, the first layer in the autoencoder constructs the factors as a linear combination of the return time series -- and are both R -- and the second layer of the autoencoder learns the factor loadings to reconstruct the return. This begs the question, I don’t need to have to output of the autoencoder reconstruct the input return, I could for example instead predict the next period return. This is what equation 1 in Bryzgalova et. al is really saying, I get the factors from PCA and then use them to predict some other variable than R, maybe R one step ahead. I will call this R’ to differentiate from the R that is the input to the factor. What I am arguing is why should only the second stage regression be optimized to match the output R’, using the autoencoder framework the first stage should be optimized to convert R to factors also with guidance from R’. The benefit of doing this in the autoencoder structure is not only are the parameters constructed based on the variable you want to predict, but the first stage regression that constructs the factors is also constructed based on the variables you want to predict, rather than self-prediction like PCA.

As the linear autoencoder is equivalent to PCA, this could be considered the natural extension of PCA to construct factors that are guided by supervised predictive targets. So your autoencoder takes R as input, but will match a different variable like 1-step ahead returns, R’, as its output. Because the autoencoder is trained end to end, the first stage factor construction equations will construct factors that provide the most information for forecasting the next period return, rather than the reconstruction error if you use PCA, just as the second stage will learn factor loadings that minimize forecast errors. In this case, still has a factor interpretation, but it’s factors that are most effective at recovering a forecast rather than recovering a reconstruction loss. This autoencoder has the same intuition as PCA since these factors are still linear combinations of assets (ie portfolios) and they affect the returns linearly or if you want to create a more complex autoencoder, nonlinearly. The difference is that the way these factors are formed, is guided by predictive power of future returns that one wants to predict, rather than just the ability to reconstruct the input variable.

Following Bryzgalova, Svetlana, et al. (2023) you can add other objectives to forecast as well as shape constraints like monotonicity, but notably, you aren’t wasting effort adding in a reconstruction loss that doesn’t do anything to improve the predictive power of your factors. Additionally, following Gu, Kelly and Xiu (2021), you can make the autoencoder nonlinear and functions of other covariates outside of return data. But more importantly, your factors are now optimized also to maximize predictive accuracy rather than learning PCA components. Additionally, even if you argue that you want factors that can produce forecasts of expected return across a wide variety of return periods, I imagine building factors that can predict one step ahead would be an improvement over factors that are constructed to predict reconstruction loss. You can also just append additional objective functions into , so that you can predict all returns from one step ahead to ten steps head for instance.

This is the outline of this idea. I am not an asset pricing person, so I am not sure if it has been done before. However, given Bryzgalova, Svetlana et al. (2023) and Gu, Kelly and Xiu (2021) are roughly state-of-the-art in the literature and this paper is a strict improvement on both of these papers (at least in my mind), there is a good chance this hasn’t been done yet. I also have a couple of other asset pricing ideas as well. Happy to share and discuss those as well. But happy to start with this first. I’m unsure if I explained it well and was clear, so let me know if you have questions. Thanks for taking the time to read this,

Cameron

Fama, Eugene F., and Kenneth R. French. "Common risk factors in the returns on stocks and bonds." *Journal of financial economics* 33.1 (1993): 3-56.

Cochrane, John H., 2011, Presidential address: Discount rates, The Journal of Finance 66, 1047– 1108.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. "Autoencoder asset pricing models." *Journal of Econometrics* 222.1 (2021): 429-450.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366.

Bryzgalova, Svetlana, et al. "Asset-Pricing Factors with Economic Targets." *Available at SSRN 4344837* (2023).