Mandatory Assignment 2 AEF

ncx951 & dsc579

2023-04-16

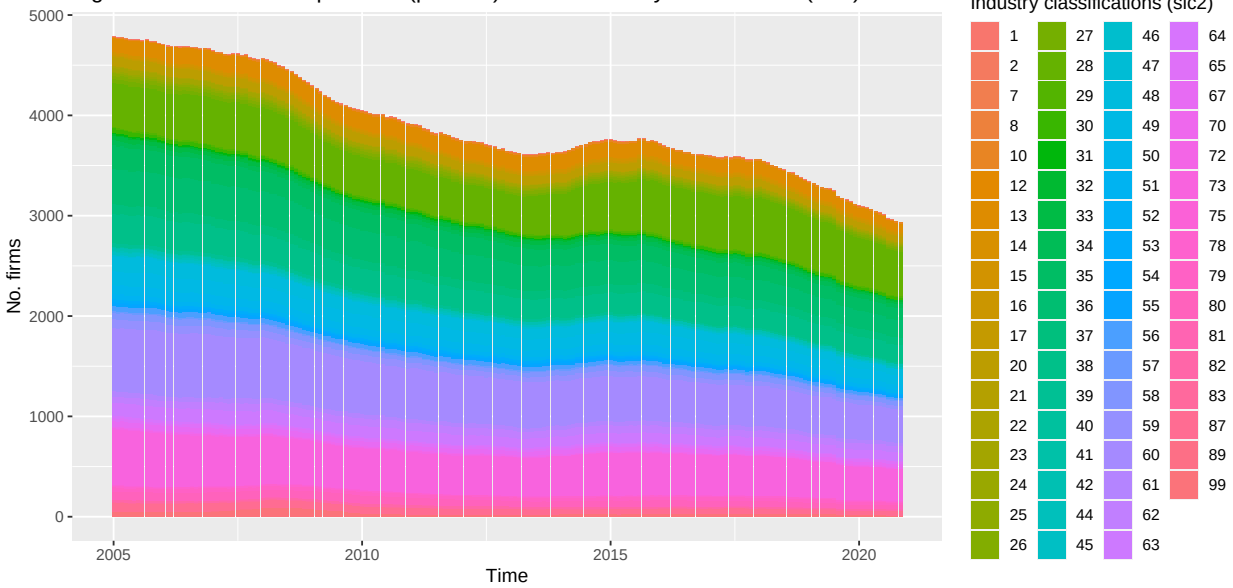
Excercise 1

We load in the dataset and select the following variables based on figure 5 from the Gu et al (2020) paper. We choose among the variables that contributes the most to overall model contribution from figure 5 from the paper. We end up using the financial/stock characteristic variables chmom, max_ret, mom12m, mom1m, mvel1. We also include the macroeconomic variables bm, dfy, ntis and tbl. These variables constitutes our predictor variables $z_{i,t}$. We also make interactions between our financial and macroeconomic variables and include them in the analysis. Lastly we generate dummies for each industy classifier contained in the sic2 variables. We do this by implementing it as a recipe. Our dataset also include permno (stock identifier), month, ret_excess (excess returns for stock i) and mktcap_lag (lagged marketcap for stock i). The financial and macroeconomic variables are already lagged by one month while the excess returns are not. Thus we don't need to make any lagging of the variables in order to make our analysis.

The dataset includes observations from January 1st 2005 and onward.

We perform EDA for the number of unique firms (permno) in each industry classification (sic2) for each month.

Figure 1: Number of unique firms (permno) in each industry classification (sic2) for each month



We notice that the total number of firms tend to decrease over time. While the ratio of industries somewhat seems to be the same over time.

Lastly we show a summary table of our chosen variables.

Table 1: Summary statistics of the predictors

Variables	mean	sd	min	median	max
Stock Characteristics					
c_chmom	0.00	0.59	-0.99	0.00	0.99
c_maxret	0.12	0.53	-0.99	0.14	0.99
c_mom12m	-0.01	0.59	-0.99	-0.03	0.99
c_mom1m	0.00	0.60	-0.99	-0.01	0.99
c_mvel1	0.03	0.59	-1.00	0.06	1.00
Macro Characteristics					
m_bm	0.30	0.05	0.22	0.31	0.44
m_dfy	0.01	0.00	0.01	0.01	0.03
m_ntis	-0.01	0.02	-0.06	-0.01	0.03
m_tbl	0.01	0.02	0.00	0.00	0.05
Initial Variables					
mktcap_lag	4908.72	24797.13	0.09	451.96	2206911.13
ret_excess	0.01	0.18	-1.00	0.00	19.88

One might note that there's a big difference between the minimum ret_excess and maximum ret_excess. While the median is only 0 and much closer to the minimum. This could indicate that high returns only occur rarely.

Excercise 2

Gu et al. (2020) describe an asset's excess return as an additive prediction error model the following way:

$$r_{i,t+1} = E_t(r_{i,t+1}) + \varepsilon_{i,t+1} \quad \text{where} \quad E_t(r_{i,t+1}) = g(z_{i,t})$$

Where $g(z_{i,t})$ is a function of the P-dimensional vector $z_{i,t}$ of predictor variables.

As stated in the paper this would be considered a very flexible model, however it imposes some important restrictions. First the $g(\cdot)$ function depends neither on the individual stock or time period. It thus leverages of the information from the entire dataset. The functional form thus doesn't adjust by time period or for specific stocks. For example one variable could have a larger explanatory power in the beginning of the time period but much lesser towards the end. The effect from the predictor would still remain constant, and could also relate to overfitting the model. The model also assumes that the prediction error $\varepsilon_{i,t+1}$ is additive and independent of the predictor variables, which may not hold in reality.

Arbitrage Pricing Theory assumes that in a competitive and frictionless market, the return of an asset can be described by the equation:

$$R_i = a_i + b_i' f + \varepsilon_i$$

Where R_i is the return of asset i, a_i is the intercept of the model, b_i is the $(K \times 1)$ vector of factor loadings and f is a $(K \times 1)$ vector of common factor realizations. And ε_i is white-noise i.e. $E(\varepsilon_i) = 0$. Thus we can translate the APT model into an additive model as following:

$$E(R_i) = a_i + b_i' f = g(z_i)$$

Where we have that $z_i \equiv [a_i, f]'$ is the $(K + 1 \times 1)$ vector of factors. For the APT model we have a linear function (i.e the form of $g(\cdot)$ is linear), and we could thus estimate the parameters with a regression. In

this case of time series analysis we also have a time aspect, where we want to estimate excess returns: $r_i = R_i - R_{rf}$ at time t and we can then translate the equation into:

$$r_{i,t+1} = E(r_{i,t+1}) + \varepsilon_{i,t+1} = g(z_{i,t}) + \varepsilon_{i,t+1}$$

This model shares some of the same issues as stated above.

Excercise 3

Hyperparameter tuning

Hyperparameter selection, also known as hyperparameter tuning or optimization, is a crucial step in the machine learning process. The primary purpose of this procedure is to find the optimal set of hyperparameters that improve the model's performance on unseen data. Hyperparameters are external configurations of the model that cannot be learned during the training process. They directly affect the behavior of the model and its ability to generalize. And they're used to counterfeit overfitting.

The objective function in hyperparameter selection measures the performance of the model with a specific set of hyperparameters on the validation data. The goal is to minimize (or maximize) this objective function, depending on the context, by searching through various hyperparameter combinations. For instance consider the L2 regularization objective function. Where the goal is to minimese the squared residuals of predictions (MSPE).

$$\mathcal{L}(\theta) = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} (r_{i,t+1} - g(z_{i,t}; \theta))^2$$

Fitting a model based on the entire dataset without a separate validation or test set can be unwise in certain circumstances, especially when the primary goal is to achieve good generalization performance. Using the entire dataset for training can lead to overfitting, where the model performs well on the training data but poorly on unseen data. This occurs because the model has learned the noise in the data and fails to capture the underlying pattern.

1. Limitations or alternatives to selecting tuning parameters from a validation sample:
2. Limited data: If the dataset is small, setting aside a portion of it for validation may result in insufficient data for training, limiting the model's ability to learn the underlying pattern effectively.
3. Overfitting to the validation set: If the hyperparameter optimization process is too extensive, the model may overfit to the validation set, resulting in poor generalization performance on unseen data.

Computationally expensive: Exhaustive search methods like grid search can be computationally expensive, especially when the search space is large, and the model takes a long time to train.

Making the train-validation-test split with 20% of the last observations as test data and make 20% of the train data into validation. That is we keep data points from 2017-03-01 and onward as test data. We keep from 2014-07-01 until 2017-02-01 as validation data. While we keep the remaining data starting from 2005-01-01 as training data. We keep the splits constant for this assignment.

Table 2: Neural network performance for different hyperparameters (evaluated on the validation data)

No. neurons at each layer	5.00000	5.00000	5.00000	10.00000	10.00000	10.00000	15.00000	15.0000	15.00000
lambda for L2-regularization	0.00010	0.00100	0.01000	0.00010	0.00100	0.01000	0.00010	0.0010	0.01000
MSE	0.03332	0.02883	0.02844	0.03425	0.02873	0.02846	0.03201	0.0286	0.02862

Excercise 4

Neural Network

Neural networks are a class of machine learning models inspired by the structure and function of the human brain. They consist of interconnected layers of artificial neurons, also called nodes, with each layer transforming the input data into more abstract representations. A typical neural network consists of an input layer, one or more hidden layers, and an output layer. Each connection between neurons has a weight that determines the strength of the connection. The input data is fed into the input layer, and the output layer provides the final prediction. Activation functions, such as ReLU (Rectified Linear Unit) or sigmoid, are used in the neurons to introduce non-linearity, which allows the network to learn complex patterns.

The objective function measures the difference between the model's predictions $\hat{y}_t$ and the actual target values y_t. In a regression problem, a common objective function is the Mean Squared Error (MSE), which calculates the average of the squared differences between the predicted and actual values, $MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_t - y_t)^{"}$. The goal of training the neural network is to minimize the objective function by adjusting the weights and biases through a process called backpropagation.

The hyperparameters to be chosen before training the model in Neural Networks include:

- Number of hidden layers
- Number of neurons at each hidden layer
- Activation function used at each layer
- Learning rate for backpropagation
- Regularization such as L1 or L2
- Epochs
- Bath size

A major drawback of Neural Network is their computational load. In this paper we oncly consider a NN with 2 layers and relatively few neurons at each layer. Since the dataset is quite vast and optimization takes a very long time. Below we provide a summary table of the different hyperparameters we have tuned on.

Our chosen network is structured as following:

- 2 hidden layers with 5 neurons each
- Sigmoid activation function in the hidden layers with a linear in the output.
- L2 regularization with $\lambda = 0.01$

Excercise 5

We predict our final fitted models on our test data. Then we calculate the predicted excess returns for each month into deciles, and then each month reconstitute portfolios using value weights. We then construct a "zero-net-investment" portfolio that buys the highest decile and sells the lowest.

Table 3: Portfolio perfomances

	Mean	SD	Sharpe Ratio	cummulative returns
NN	-0.004	-0.315	0.012	0.232
Market	0.014	0.822	0.051	1.605

Below we summarize the perfomance of the portfolio

Figure 2: Monthly Returns Performance