Mandatory Assignment 2

Niels Nygreen Bonke and Johan Kielgast Ladelund

2022-04-07

Question 1

Based on Figure 5 and Table 4 in (Gu et al, 2020), we choose the following characteristics and macro variables (we refer to (Gu et al, 2020) for definitions of the variables):

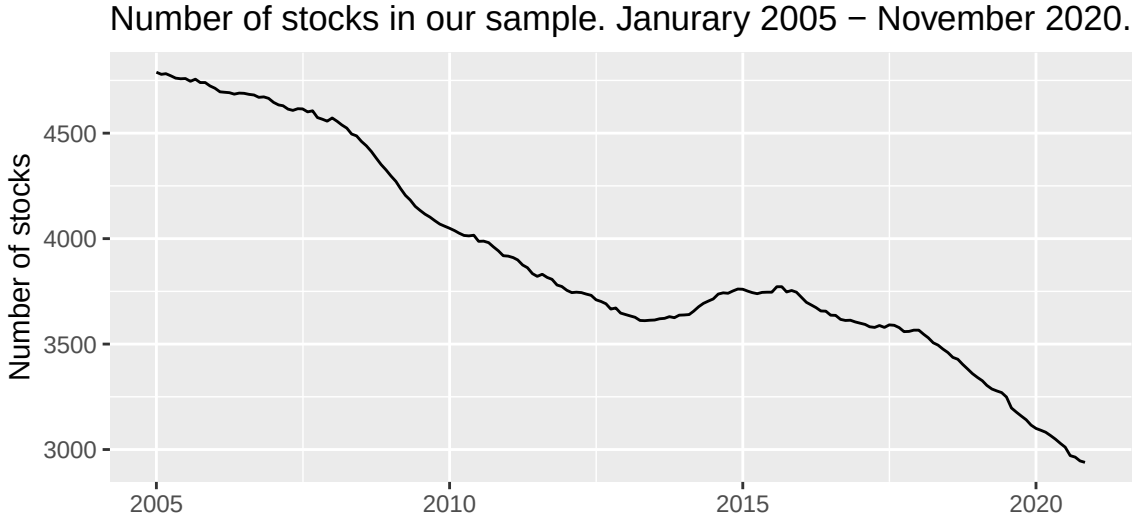
- **Characteristics:** `mom1m`, `mvel1`, `mom12m`, `chmom`, `maxret`
- **Macro variables:** `bm`, `dfy`, `ntis`, `tbl`
- **Other variables:** `permno`, `month`, `ret_excess`, `mktcap_lag`, `sic2`

These variables are chosen as they appear to be the most relevant variables across all the models used in (Gu et al, 2020). Interestingly, `dfy` seems to be very important for all models except the Neural Networks (NNs). We are eager to find whether our analysis replicates this discrepancy. Further, it might be include both momentum (`mom12m`) and momentum change (`chmom`), but from table 5 of (Gu et al, 2020) it seems that both these variables are simultaneously important across all the models.

From the below graph and table, we see that the number of stocks has been steadily declining. Specifically, we see that the decline in number of stocks is from 4789 to 2939 stocks.

Table 1: Summary statistics of the number of stocks (monthly data)

Avgerage	Std. dev.	Min	Max
3896	500	2939	4789



As for the macro-economic variables, we provide the following summary statistics:

Table 2: Summary statistics of the included macro variable (monthly data)

Variable	Min	Median	Mean	Max	Standard deviation
Book-to-market ratio	0.2164	0.3088	0.3013	0.4411	0.0481
Default spread	0.0055	0.0095	0.0108	0.0338	0.0046
Net equity expansion	-0.0560	-0.0102	-0.0092	0.0267	0.0169
Treasury-bill rate	0.0001	0.0027	0.0124	0.0503	0.0158

Finally, we make the the following recipe:

```
rec <- recipe(ret_excess ~ ., data = data ) %>%
  step_rm(month, permno) %>% #Remove Month --> not a feasible predictor
  step_interact(terms = ~ contains("characteristic"):contains("macro")) %>% #Interactions
  step_dummy(sic2, one_hot = TRUE) %>% #One_hot=TRUE --> no reference category (dummy trap)
  step_normalize(all_predictors()) %>%
  step_center(ret_excess, skip = TRUE)
```

```
## # A tibble: 744,088 x 101
##    mktcap_lag characteristic_mom1m characteristic_mvel1 characteristic_mom12m
##         <dbl>                <dbl>                <dbl>                 <dbl>
##  1     -0.197                1.30                -1.59                 -0.535
##  2     -0.187                1.30                -0.145                 1.03
##  3     -0.197                1.64                -1.51                 -1.63
##  4     -0.193                1.57                -0.707                 0.250
##  5     -0.180                0.273                0.211                 0.726
##  6     -0.197                1.16                -1.64                 -0.143
##  7     -0.175               -1.32                 0.364                -1.11
##  8     -0.196                0.521               -1.36                 -1.63
##  9     -0.195               -0.290               -1.10                  1.54
## 10     -0.191                0.168               -0.453                -1.53
## # ... with 744,078 more rows, and 97 more variables:
## #   characteristic_chmom <dbl>, characteristic_maxret <dbl>, macro_bm <dbl>,
## #   macro_dfy <dbl>, macro_ntis <dbl>, macro_tbl <dbl>, ret_excess <dbl>,
## #   characteristic_mom1m_x_macro_bm <dbl>,
## #   characteristic_mom1m_x_macro_dfy <dbl>,
## #   characteristic_mom1m_x_macro_ntis <dbl>,
## #   characteristic_mom1m_x_macro_tbl <dbl>, ...
```

Question 2

The linear factor model may be seen as on possible specification of an Arbitrage Pricing Theory (APT) model. In the APT framework, $z_{i,t}$ should be matrix consisting selected *factors* that we consider important in explaining the the stock returns. These will typically be portfolios (often times self-financing) which we belive the correlate significantly with the average stock return. If we, for instance, take inspiration from the [Fama-French three factor model]{https://bit.ly/3xgs5dF} and look at factors such as `Market (Mkt)`, `Small-minus-big (SMB)`, and `High-minus-low (HML)`, the functional form of $g(\cdot)$ would be

$$g(z_{i,t}) = \beta_i^{Mkt} E_t(r_{Mkt} - r_f) + \beta_i^{SMB} E_t(r_{SMB}) + \beta_i^{HML} E_t(r_{HML})$$

where we subtract the the risk free rate, `r_f`, only from the Market return, as the other two portfolios are typically defined in a self-financing manner. Note that one may add t-subscripts to the beta-parameters in order to make these time-variant. Hence $z_{i,t} = (r_{Mkt} - r_f \quad r_SMB \quad r_HML)'$.

Question 3

Hyperparameters (or tunings parameters) are parameters which values determines and controls the learning process/model complexity in a Machine Learning (ML) algorithm. Examples of hyperparameters could be the number of random trees in a forest or the train-test split ratio. Since these are crucial factors of the learning outcome of the algorithm, they must be chosen according to the objective. In our case the objective is to maximize out-of-sample (OOS) performance, and hence we should perform hyperparameter tuning according to minimizing the discrepancy between predicted OOS and actual returns.

As explained by Gu et al (2020) this can be done by splitting the full data set in to three different chronologically ascending parts: a training sample, a validation sample and a test sample.

The training sample estimates the chosen model subject to a constraint that the hyperparameters are given by some specific values, i.e. best guesses for initial values. Subsequently the validation sample tunes the hyperparameters by first producing forecasts for the validation sample, using the parameters estimated in the test sample, and then compute the values for a specific objective function (e.g. associated with SLS, WLS, elastic net, PCR, PLS or NNs) using the forecast errors. These first two steps are repeated a large number of times for different hyperparameter values (i.e. a random or grid search is applied), and hence the model is iteratively re-estimated to find the hyperparameter values that induces the estimation of the model parameters with the smallest forecast errors (using a loss function like (R)MSE, MAE or QLIKE) in relation to the objective function. At last the test sample is used to perform OOS performance testing of the model with the optimal hyperparameters chosen and the remaining model parameters as estimated in the training sample.

Generally if one estimates (trains) the model on the entire data set, the possibility to tune model hyperparameters and evaluate actual OOS performance is lost. The trade-off is the fact that in general (or to a certain extent) more data can increase the accuracy or fit of the model. However, if one intends to evaluate OOS forecasting performance it is unwise not to leave at least some part of the data set to validation and testing.

Another risk - that is also impacted by the choice of objective function - is the risk associated with either over- or underfitting the model. That is to either fit the model so well to the training sample that it performs poorly when forecasting OOS because dynamics might be slightly different (overfitting) or to not fit the model well enough (for example due to the risk of overfitting) such that it neither performs well e.g. in an in-sample forecast nor an OOS forecast (underfitting). In accordance with Gu et al (2020) we note that linear models (OLS, WLS) are at risk of overfitting noise in the training sample, why we might choose objective function that are not (only) linear (e.g. elastic net, PCR, PLS, NNs).

One simple alternative to performing hyperparameter tuning could be instead to use the default hyperparameter settings which are chosen to fit most models moderately well. This would free-up more data for training and testing the model, by not allocating a portion of the data set solely to tuning hyperparameters.

We now split our data set into the training, validation and test samples, where the latter is assigned 20% of our total data set. We choose to use 75% of the remaining data to train our model and 25% for hyperparameter tuning (i.e. validation) - hence the actual splits are 60-20-20. We choose these fractions since it is our understanding that they are duly considered by the literature on the axis from underfitting to overfitting, as explained previously.

Table 3: Summary statistics of the number of stocks (monthly data)

	Training sample	Validation sample	Testing sample
% of total dataset	60	20	20
Number of data points	446452	148818	148818

Question 4

We now choose two machine learning methods to proceed with in the analysis of excess return prediction. We choose the elastic net and neural network methods. We do this to explore both one fully transparent yet possibly less successful method (i.e. elastic net) and one opaque and at times "black-box"-like yet very accurate and widely used method (i.e. neural network).

Elastic Net

Elastic Net (ENet) is a hybrid of the Ridge and Lasso regressions. Both the Lasso and Ridge regressions involve shrinkage factors, where the former penalizes the sum of the absolute value of the weights in the regression, and the latter penalizes the sum of the squared value of the weights. To reap the benefits of both methods (including the higher penalization of extreme values in the Ridge regression i.a.) we use ENet. The ENet penalty function is literally a linear combination of the Lasso and Ridge regressions, where $\rho = 0$, in the following, is equivalent to the Lasso penalty function and $\rho = 1$ to the Ridge penalty function:

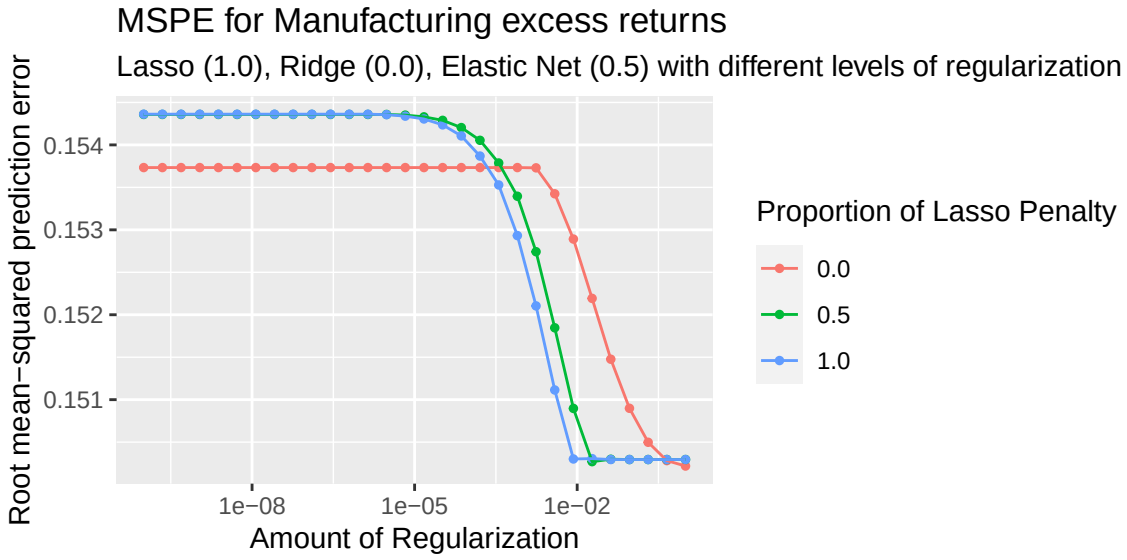
$$\hat{\beta}^{ENet} = \arg\min\ (Y - X\beta)'(Y - X\beta) + \lambda(1-\rho)\sum_{j=1}^{P}|\beta_j| + \frac{1}{2}\lambda\rho\sum_{j=1}^{P}\beta_j^2$$

We now set up and fit the model using the elastic net method on the training sample, and specify some initial guesses for the optimal λ and ρ. These are not crucial since we will optimize them with hyper parameter tuning subsequently. Next we tune the model hyperparameters by forecasting the validation sample created previously and evaluating Mean Squared Prediction Errors (MSPE)

Next we tune the model hyperparameters by forecasting the validation sample created previously and evaluating Mean Squared Prediction Errors (MSPE)

```
## # A tibble: 1 x 2
##   train_months val_months
##          <dbl>      <dbl>
## 1         110.       36.8
```

```
## Overlapping Timestamps Detected. Processing overlapping time series together using sliding windows.
```



We have limited the ENet to only have 3 different candidate values ($\rho \in \{0, 0.5, 1\}$) in order to accommodate our limited computational power. However, a more finely meshed grid would have been preferable.

Neural Network

Neural networks is one of the (if not the) most dominant method in ML. Gu et al (2020) emphasizes that besides being widely used, it is also the preferred method to use in complex ML problems, including i.a. the computer science branch of Natural Language Processing (NLP). One drawback of this method (as we will come to see) is that it is much less obvious and more opaque, exactly what is going on in the computational parts of the method (i.e specific parts of the network).

We use the feed-forward neural network that is composed of an input layer, hidden layers and and output layer.

The input layer consists of the first set of so-called "neurons" in the network. It is in this stage that the raw data is brought into the system. Gu et al (2020) refers to this layers as consisting of "raw predictors"

The hidden layer(s) is the second (third, fourth and so on) layer(s) of neurons in the network. It is in these interactive layers that the computation is performed, and hence where the "raw predictors" are altered. This is done such that each neuron in each hidden layer in the network uses a so-called "activation function" that decides whether or not the neuron in the specific layer should be activated. This is done prior to sending its output to the next hidden layer of neurons. Hence the objective function is generally used to determine whether or not the specific neuron's input to the network is important for predictions.

Finally there is the output layer, which is the last set of neurons in the network. It is here that the chain of interaction between the hidden layers is collected to a single set of predictions. We note that while the outputlayer might also use an activation function, it is ofen times a different one than that of the hidden layers, depending on the required prediction outcome

Gu et al (2020) describes the various important choices that should be made before creating a neural network. These i.a. include choosing the number of hidden layers and the number of neurons. We rely on the fact that when having to do with less complex data sets such as ours, one hidden layer is typically enough. Hence we choose to set up the model with a single hidden layer.

```
## Loaded Tensorflow version 2.8.0
```

Table A.5

The table below describes the hyperparameters used in the ENet and NN1 model. Throughout our tuning regimes we choose the hyperparameters that yields the lowest RMSPE (or equivalently MSPE).

Table 4: Hyperparameters for both models

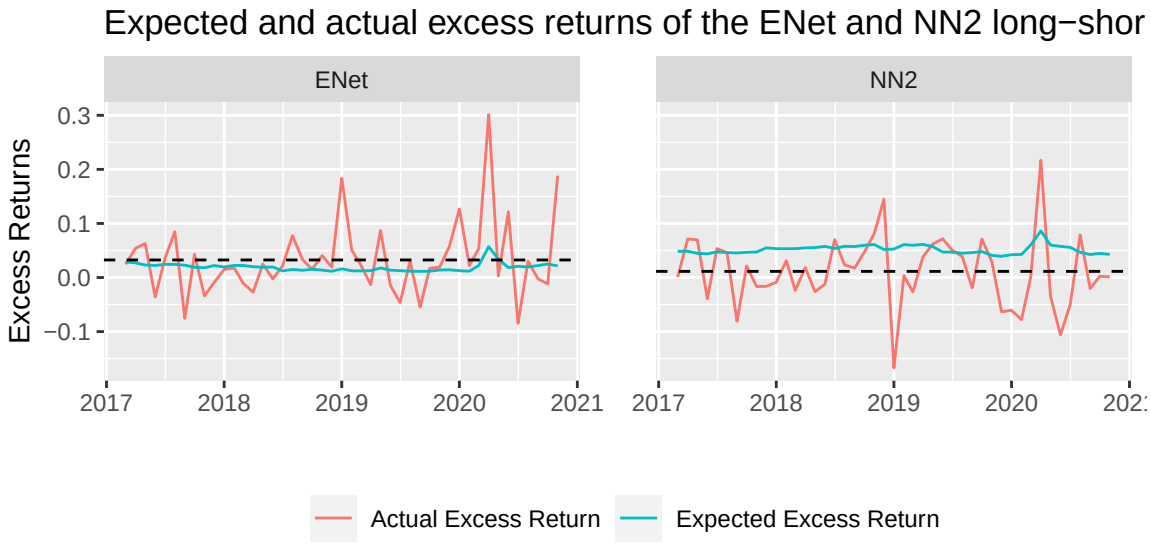
	ENet	NN1
MSPE	$\checkmark$	$\checkmark$
L1 (lambda)	$\lambda\in[10^{-10},1]$	$\lambda\in[10^{-10},1]$
L2 (rho)	$\rho\in\{0,0.5,1\}$	-
Others	Data folds = 1	Epochs = 100, Hidden units = 20

Question 5

```
## Loaded Tensorflow version 2.8.0
```

In the figure below we have evaluated return predictions for the long-short portfolio predicted by the tuned ENet and NN2 models as described in the question. We observe that our NN2 models performs best in terms of predicting returns. This might be why it's preferred over most oster ML methods. It is however very opaque exactly what is going on in the two hidden layers, aside from the fact that we know the activation function applied.

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```



At last we compare the results inside the CAPM framework by computing α's of the long short portfolios produced by each model.

Table 5: CAPM Alpha estimates

	ENet	NN2
Alpha	0.0273	0.0110
Std.dev	0.0102	0.0101
t-test statistic	2.6683	1.0900