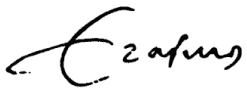

Erasmus University Rotterdam

Erasmus School of Economics

Bachelor Thesis Econometrics & Economics

Meta-strategies for Machine Learning Asset Pricing Predictions

Matteo Lengkeek

Abstract

This article presents a comparative analysis of the performance of forecasts combination techniques in the context of stock prices using the forecasts of OLS, dimension reduction models, coefficient penalizing models, regression trees and neural networks. The goal is to create a forecast combination technique which creates better forecasts than the stand alone machine learning models in terms of R^2, the Diebold and Mariano test and in terms of portfolio Sharpe ratio. I propose two new forecast combination techniques based on business cycle variables and past performance. I also include existing forecast combination techniques such as LASSO and simple average forecast combinations. The past performance rank weighted forecast combination can significantly improve forecasting performance in all three tested aspects. Furthermore, forecast combination generally leads to better forecasts and is recommended when multiple forecasts are available to make better decisions. (JEL C52, C55, C58, G0, G1, G17)

supervised by

Dr. X Xiao [1]

July 5, 2020

[1]My thanks go to Professor Xiao for her guidance, advice and feedback.

Contents

1 Introduction

In the field of empirical asset pricing and investing, measuring and estimating risk premiums remains one of the hardest problems faced in finance today. Gu, Kelly, and Xiu (2020) conduct a comparative analysis of several popular machine learning algorithms such as PCR, PLS, Elastic-net, Random Forests, and Neural Networks and demonstrate their performance in forecasting those risk premiums.

Gu et al. (2020) find that out of all models, neural networks and random forest significantly improve forecast predictions over standard linear models. At the end of their paper, they briefly mention two simple 'meta-strategies' which combine forecasts of all their models and find that the meta-models perform well but do not outperform their neural network.

In the field of economics it has been shown that combining multiple forecasts can lead to a smaller variance of predictions and lead to better forecasting results[2]. Another reason for forecast combination is that each of the models might explain a part of the complicated relations present in our economy. Combining those models leads to capturing more of those relations resulting in better forecasts. It is therefore interesting to extend the research of Gu et al. (2020) by implementing other forecast combining techniques to examine if they can improve forecasting performance.

While taking the simple average has proven itself in the literature, it seems counter-intuitive to assign equal weights to all forecasts when more information is readily available such as past performance and business cycles. I therefore, suggest a new forecast combination method which uses past performance to create a rank-weighted average, such that models which performed best in previous samples get more weight than models which performed relatively badly. Timmermann (2006) and Hendry and Clements (2004) suggest that model instability can help explain why forecast combining is so effective. Therefore, I suggest a forecast combining model which bases its weights on a neural network trained on business cycle variables to predict which forecasts work best each period.

This article extends the research done by Gu et al. (2020) by implementing existing forecasting combination techniques such as the simple average, LASSO, partial-egalitarian LASSO and suggest two new forecast combination techniques which use past performance and business cycle variables to create better forecasts. Using these meta-models, we test if asset-pricing forecast performance

[2]For overviews, see Diebold and Lopez (1996), Timmermann (2006) and Elliott and Timmermann (2008)

can be increased

The research question which will be examined in this article is:

Can we improve asset pricing forecasts by implementing meta-strategies that combine best-performing machine learning models?

The following sub-questions will back this research question:

- Can a meta-strategy outperform the stand-alone models in an out-of-sample R-squared metric?
- Can we achieve significantly better forecast performance in terms of the Diebold and Mariano test statistic using meta-strategies?
- Can we outperform the stand-alone models in a top-down decile spread Sharpe ratio?

The focus of this paper is to reproduce the results of (Gu et al., 2020) and to examine if their forecasts can be improved using more advanced model-combining techniques. The main finding is that meta-models can improve asset pricing forecasts. A simple rank-weighted ensemble outperforms all stand-alone models in terms of out-of-sample R^2, Diebold and Mariano pairwise model comparison test, and Sharpe Ratio. More generally, the meta-models perform better than the stand-alone models in portfolio creation, and simpler forecast combinations work better than complicated combinations. Forecast combination can thus improve our empirical understanding of asset returns by creating better forecasts.

In the asset pricing literature, machine learning models have not made many appearances yet; however, dimension reduction methods are used to estimate factor pricing models (Feng, Giglio, and Xiu and Kelly, Pruitt, and Su). Shrinkage and selection methods are applied to approximate a nonlinear function for expected returns by Freyberger, Neuhierl, and Weber (2020) and to approximate a stochastic discount factor by Kozak, Nagel, and Santosh (2020). LASSO is performed to predict global equity market returns using lagged returns by Han, He, Rapach, and Zhou (2018). Regression trees and tree-based models are used to predict credit card delinquencies and defaults by Khandani, Kim, and Lo (2010), Butaru et al. (2016), and portfolio sorting by Moritz and Zimmermann (2016). Neural networks are used to forecast derivatives prices (Hutchinson, Lo, and Poggio (1994), Yao, Li, and Tan (2000) among others), mortgage foreclosures, delinquencies and mortgage prepayment (Sirignano, Sadhwani, & Giesecke, 2016) and portfolio selection (Heaton, Polson, & Witte, 2016).

Machine learning models have been used to combine multiple forecasts in the literature. Diebold and Shin (2017) use LASSO and partial-egalitarian-LASSO to improve forecasts. Sigletos, Paliouras, Spyropoulos, and Hatzopoulos (2005) and Wolpert (1992) describe and perform stacking and voting methods based on machine learning to improve forecasts in the field of information extraction and text processing. Forecast combinations are proven to be successful in economic fields such as market volatility, money supply, inflation, check volume, gross national product, and non-economic fields such as city populations, political risks, and meteorological data (Clemen (1989)). Zhu, Xie, Wang, and Yan (2017) use ensemble models to predict the credit risk of China's small and medium-sized enterprise (SME) in supply chain finance and Diebold and Lopez (1996) discusses whether and how a set of forecasts may be combined to produce a superior composite forecast.

This article's contribution is that I introduce two new forecast combining models different from the existing literature. My first contribution takes the business cycle into account when assigning weights to the forecasts and, therefore, is able to assign time-varying weights based on the economic state. My second contribution is a forecasting technique that uses ranked past performance to assign weights to forecasts. The rank-weighted forecast combining model increased the forecasting performance of asset-pricing. The capability to better predict risk premiums is scientific relevant in the sense that it leads to more insights in which predictor variables and models play a role in explaining future returns and therefore, could support decision making. Therefore, this research partly solves the problem of risk premium measurement and enables more reliable research into the underlying economic mechanisms of stock prices. I demonstrate financial gains in terms of Sharpe ratio by using forecast-combining models, making it an attractive technique for investors. With the increasing popularity of machine learning and forecast combinations, this article might also serve as an interesting and practical example of how to perform such ensemble models and demonstrate its benefits.

The structure of the paper will be the following: first, I will describe the methodology used to reproduce Gu et al. (2020) and to implement my extensions in section 2. Results will be explained in section 3 and the conclusion of the research will be discussed in section 4.

2 Methodology

This section begins with discussing the data used for this research, followed by a description of the reproduction of machine learning methods used in Gu et al. (2020) and the implementation of the

meta-model extensions which combines the forecasts of the models and ends with the methodology used to answer the three sub-questions.

Some general objectives and concepts should first be declared. A general objective for each forecasting model is to minimize the prediction Mean Squared Error (MSE). Some models use regularization in the form of parameter penalization or outlier robustness adaptations to deal with over-fitting and therefore improve out-of-sample forecasting performance. An important definition is an asset's excess return ($r_{i,t+1}$) described as an additive prediction error model by Gu et al. (2020):

$$r_{i,t+1} = E_t(r_{i,t+1}) + \varepsilon_{i,t+1} \tag{1}$$

where

$$E_t(r_{i,t+1}) = g(\cdot) \tag{2}$$

We use this definition for the rest of this paper because it is a simple representation. The unspecified form of $g(\cdot)$ allows a wide variety of forecast models which can be specified individually. The excess return of stock i at time $t+1$ is denoted as $r_{i,t+1}$, the corresponding forecast errors are denoted as $\varepsilon_{i,t+1}$ and $g(\cdot)$ denote the model used to forecast the excess returns.

Another essential general concept used in the reproduction and especially in the extension of meta models, is ensembles. An ensemble model combines the prediction of multiple models into one. The purpose of an ensemble method is to cancel out biases from models by taking the average of predictions.

In equation 3 a simple ensemble is portrayed, which averages the prediction of multiple models. The ensemble outcome is denoted by y_{ensemble}, the set of models used for the ensemble is denoted as M, the models are again denoted as $g(\cdot)$.

$$y_{\text{ensemble}} = \frac{1}{|M|} \sum_{g \in M} g(\cdot) \tag{3}$$

2.1 Data

I use the same data used by Gu et al. (2020) to reproduce the stand-alone models. This data can be found at Xiu's website and contains a selection of 94 monthly stock-level explanatory variables which originate from papers of Kelly et al. (2019) and Freyberger et al. (2020). The variables selection procedure used is from Green, Hand, and Zhang (2013). We match this data with the monthly equity return obtained from the Center for Research in Security Prices (CRSP) database.

The stocks present in this data-set are from firms in the NASDAQ, AMEX, and NYSE from 1965 until 2016. Specifics of the variables are available internet appendix F of Gu et al. (2020).

In addition to the stock-level explanatory variables, we construct eight macroeconomic predictors following the definitions of Welch and Goyal (2008). The eight macroeconomic predictor variables are: stock variance (macro_svar), book-to-market ratio (macro_bm), default spread (macro_dfy), earnings-price ratio (macro_ep), dividend-price ratio (macro_dp), term spread (macro_tms), net equity expansion (macro_ntis) and Treasury-bill rate (macro_tbl). The independent variable *market risk premium* is proxied by subtracting the treasury bill rate from the stock level, as done in Gu et al. (2020).

Lastly we include an industry dummy based on the first two numbers of the SIC code. In summary, we have for 94 stock-specific variables, eight macro-economic variables, and 61 industry dummy variables. Throughout the reproduction, the explanatory variables used in the stand-alone models will be the following covariates as defined by Gu et al. (2020):

$$z_{i,t} = x_t \otimes c_{i,t}$$

where $c_{i,t}$ is the vector of 92 characteristics for each stock i, and x_t is the vector of macroeconomic predictors with an added constant. Thus, $z_{i,t}$ is a vector of features for predicting individual stock returns and includes interactions between macroeconomic state variables and stock-level characteristics. The total number of covariates is $94 \times (8 + 1) + 61 = 907$

The data sample of Gu et al. (2020) starts in March 1957 and ends in December 2016, spanning 60 years of monthly returns. The data provided on Xiu's website contains a lot of missing variables however, so after removing incomplete date [3] and outliers [4] we remain with a period starting in January 1985 and ending in December 2016. This data set contains 173416 monthly observations, averaging 452 companies per month, and 3009 companies in total.

For the extension, we consider a broader set of macroeconomic variables and use the FRED-MD database, which contains 124 macro-economic business cycle variables. This data-set has been set

[3] All data before 1985 was for at least one variable incomplete and it was therefore not possible to fill the values with the monthly cross-sectional median as done by Gu et al. (2020), more generally a lot of entries contained missing data, sometimes missing 50% of the variables. Filling with the median does not seem appropriate since it strongly deteriorates the explanatory power variables might have. We therefore choose to only consider complete data

[4] A few observations with excess risk premiums above 100% are discarded because they result in an extremely skewed distribution

up by McCracken and Ng (2016) to establish a starting point for empirical analysis that requires "big" macroeconomic data. The data-set covers variables frequently used in the literature and has the advantage that it is publicly available and monthly updated.

The data is split in 16 years of training, six years of validation and tuning, and ten years of testing. This is a slight deviation from traditional training and testing splits where a 50% - 25% - 25% split or 80% - 10% - 10% ratio is more common. The reason for increasing the test period at the cost of the validation period is that financial results should hold over a sufficiently long period. Gu et al. (2020) take a similar approach in splitting the data, but they consider more years at the cost of incomplete, cross-section median-filled data.

2.2 Reproduction: stand-alone models

This subsection describes the methods and models used by Gu et al. (2020) from which we need to obtain the forecasts used in the forecast combining meta-model extension. We start with the most simple model and end with the most complicated one. For each model we explain how it works and how they differ from each other. An important concept that comes back in multiple models is hyper tuning, which means tuning model-specific parameters to obtain better out-of-sample fits. We *tune* the hyperparameters until we find the best validation set forecasting performance. The test dataset never enters the training and hyper tuning processes. Another important concept is 'ensemble'. For the reproduction section, an ensemble means a combination of forecasts of the same type of model. A simple example would be to train a neural network ten times and average the forecasts. In the extension, we will go more in-depth details. All programming is done in Python, and code is attached in the appendix.

2.2.1 Linear models

The most simple form of machine learning is the ordinary least square (OLS) regression in the form of:

$$r_{i,t+1} = \alpha + \beta z_{i,t}$$

A problem with OLS regressions is that outliers can heavily affect the outcome of the regression. Therefore, we consider the Huber loss function as well, which assigns less weight to outliers and is considered a more robust approach (Huber (1992)). Because an OLS regressions with many variables can be prone to in-sample over-fitting, a more parsimonious version of OLS is also performed

based on only the three variables: size, momentum, and book-to-market based on the approach of Fama and French (1993).

2.2.2 Dimension reduction models

One approach to dealing with a large number of variables is to apply dimension reduction on the dataset and perform a regression on the reduced variables. We consider two dimension reduction techniques: Principal Component Analysis and Partial Least-squares.

When high correlations between variables are present (multi-collinearities), the variance of some of the coefficients can be very large, leading to unreliable regressions. To deal with this high correlation among predictors, we perform linear dimensional reduction using Singular Value Decomposition (SVD) to end up with a data set of transformed vectors which are orthogonal each other and therefore reduce the multi-collinearities. Using the principal components obtained, we perform a principal component regression (PCR) following the procedure of Jolliffe (1986).

As mentioned in (Gu et al., 2020), one drawback of PCR is that it does not take into account the goal of estimating risk premiums when creating the components but instead bases the components solely on the covariates of the predictors themselves. Partial Least Squares, however, takes the covariation of the explanatory variables and the risk premium into account when selecting the components. Instead of creating orthogonal vectors within the dataset, we take all variables and look at their univariate prediction coefficient. The coefficients found reflect the partial sensitivity of returns for each predictor variable. We use the partial sensitivities as weights to combine all predictor variables into a single aggregate component. After aggregating the variables, we orthogonalize the original dataset with respect to the aggregate components and repeat the same procedure until the desired number of components is reached. This approach performs dimension reduction while keeping the forecasting objective in mind. For both dimension reduction techniques, the number of components is decided by hyper tuning, and the Huber loss is also considered.

2.2.3 Parameter penalizing and selecting models

Another approach in dealing with a large number of variables is to apply a variable selection method. We do this by an Elastic Net (ENet) regression. ENet is a combination of LASSO and RIDGE regressions. LASSO adds the sum of absolute coefficients to the MSE loss function such that it is natural for the coefficients to shrink to zero when they have little to low explanatory power. RIDGE does the same but with squared coefficients. Elastic Net combines both methods

by taking a linear combination. Equation 4 shows how a penalty is added to the MSE loss function and equation 5 shows the ENet loss and its relation to LASSO and RIDGE.

$$\mathcal{L}(\theta;\cdot) = \underbrace{\mathcal{L}(\theta)}_{\text{MSE loss}} + \underbrace{\phi(\theta;\cdot)}_{\text{penalty}} \tag{4}$$

$$\phi(\theta;\lambda,\rho) = \underbrace{\lambda(1-\rho)\sum_{j=1}^{P}|\theta_j|}_{\text{LASSO}} + \underbrace{\frac{1}{2}\lambda\rho\sum_{j=1}^{P}\theta_j^2}_{\text{RIDGE}} \tag{5}$$

ENet can take the form of both LASSO or RIDGE by adjusting to ρ to 0 or 1, respectively. The size of ρ determines the magnitude of penalizing the coefficients. Both hyperparameters ρ and γ are hyper tuned in the validation set.

Up until now we have only considered linear relations. Economic relations are often complicated an not always easy to capture in linear models. We therefore also include a Generalized Linear Model which can capture such non-linear relations. We use a spline series defined as in equation 6 of order 2 just as (Gu et al., 2020) to end up with the transformation: $(1, z, (z-c_1)^2, (z-c_2)^2, \ldots, (z-c_{K-2})^2)$, where $c_1, c_2, \ldots c_{K-2}$ are knots.

$$g(z;\theta,p(\cdot)) = \sum_{j=1}^{P} p\left(z_j\right)'\theta_j \tag{6}$$

Knots are based on the variables' distribution and placed at quantiles such that the knots are divided evenly over the variable range. One knot would be at the 50% quantile, and two knots are at the 33.3% and 66.7% quantile. More generally, the knots are placed at the quantiles $\frac{i}{K+1}$ with $1 < i < K$ where i is the i_{th} knot and K is the total amount of knots. Using the splined data, we perform a group LASSO in order to select the same variables in each spline using the penalty function shown in equation 7. The penalty represents the sum of Euclidean norms for each variable and its transformations, which enables grouped variable selection.

$$\phi(\theta;\lambda,K) = \underbrace{\lambda\sum_{j=1}^{P}\left(\sum_{k=1}^{K}\theta_{j,k}^2\right)^{1/2}}_{\text{Group LASSO}} \tag{7}$$

2.2.4 Regression Forests

A more advanced machine learning technique is the regression forest. Because Regression Forests are highly vulnerable to in-sample over-fitting, we apply two types of ensembles on the regression

forests: Random Forest (RF) and Gradient Boosted Regression Trees (GBRT).

The first ensemble is a Random Forest. This algorithm creates multiple regression forests using random subsets of the training data. The random subsets are drawn with replacements and with different seeds for each forest. The Random Forest is obtained by averaging all forests' predictions, leading to a better out-of-sample forecast performance. Another measure to prevent in sample over-fitting is setting the maximum depth of the trees, that is, the maximum consecutive branches. The number of trees and the maximum depth is tuned using the validation sample. We test combinations of ensembles of 10 to 200 forests and depths between 1 and 10.

Gradient Boosted Regression Trees is another form of an ensemble where previous models of the ensemble are taken into account when creating a new model. GBRT exists of shallower trees (1 2 branches) but selects subsets of the data with adjusted probabilities. The probabilities of selection increase if previous trees could not explain this observation very well and decreased when the observation is already quite well explained using the existing trees. This ensemble can be seen as a recursive form of regression forest, which takes the explanatory power of previous trees into account when selecting a random set of data for the new tree. The GBRT trees are also shallower than RF, but research has shown that a combination of a lot of shallow learners can outperform single complicated models in out-of-sample performance.

2.2.5 Neural Networks

The final model implemented is the neural network. The neural network that we use is a sequential neural network. This sequential network has multiple layers where the output of each layer is used as an input for a deeper layer until we reach the final node of the network where the prediction is made. We use the RElu activation function just like (Gu et al., 2020), but also consider other widespread activation functions such as sigmoid and softmax. We consider multiple layer setups, the approach of Gu et al. (2020) is followed, where we implement the pyramid schedule as suggested by (Masters, 1993). Because our data contains many variables, some sort of variable selection is desired. Therefore L1 (LASSO) regularization is applied; that is, we add the sum of absolute weights of each layer to the loss function. In addition to L1 regularization, L2 (RIDGE) and L1L2 (ENet) is tested. The learning rate (LR) and L1/L2/L1_L2 parameters are tuned in the validation period. Because of the stochastic nature of the neural network training we get the final predicition of an ensemble of size 10.

2.3 Extension: meta-models

Now that all stand alone models from our reproduction have been introduced we can start to combine the forecasts. The motivation for forecast combination is that the literature has shown that linear combinations of forecasts often reduces variance and can even outperform individual models (Granger (1989), (Newbold & Harvey, 2002) and Aiolfi and Timmermann (2006)). A possible explanation for the improved performance might be that combinations of models mitigate structural breaks and model misspecification and thus lead to more accurate forecasts ((Pesaran & Timmermann, 2007)). Another explanation is that the models process the information differently, thus leading to a variety of forecasts. If each model provide a partial, incomplete overlapping explanations, then a combination of the models might perform better than either stand-alone model because more aspects of the underlying process is explained (Hendry and Clements (2004)). (Timmermann, 2006) finds in his review of forecast combinations that no single model can be expected to dominate over time because individual models are coarse approximations to complex processes with biases that shift over time. This explains the success of forecast combination in a lot of fields because biases can cancel each out and variance is often reduced. Furthermore Dietterich (2000) argues that forecast combination is less prone to over-fitting, although some of our models already contain ensembles (regression trees and neural networks), combining different models might still improve out-of-sample performance.

As shown in the literature forecast combination can be very advantageous. Gu et al. (2020) implement a meta-model which simply averages the forecasts of all models. Although this 'simple-average' has been proven useful in the literature, it seems counter-intuitive to give all models equal weights when there are clear differences in forecasting performance.

Up until now we have only covered same model ensembles with equal weights. For the remainder of this paper we will use a more advanced form known as the stacked generalization to combine forecasts. We will introduce the concept of stacked generalization for the rest of this paper because it allows more complicated forecast combinations. A stacked generalization is a type of ensemble which uses a high-level model to combine lower-level models to achieve a greater predictive accuracy. In our case that implies an over-arching model selecting a combination of forecasts produced by the stand alone machine learning models.

The formula we use for our stacked generalization is described in equation 8 and the flexible form enables model and time varying weights ($w(\cdot)$). The input for the weights may depend on

performance, macro-economic variables or on auxiliary regressions. $\widehat{r}^{(m)}_{i,t+1|t}$ represent the forecasted one-month-ahead return of model m in period t and $\widehat{M}$ denotes all models considered for the stacked ensemble. When the weights add up to one and can only be one or zero this stacked generalization can be seen as a voting model giving all weight to one model for a certain month.

$$E_t(r_{i,t+1}) = \sum_{m \in \widehat{M}} w(\cdot) \widehat{r}^{(m)}_{i,t+1|t} \tag{8}$$

Wolpert (1992) have used such an stacked generalization approach and showed that the combined model outperformed the stand-alone models. Sigletos et al. (2005) also used stacked generalization and also found that stacked generalization outperformed every stand alone model. It would therefore be interesting to perform a similar approach to the best performing models in asset-pricing where the expertise of each type of model combined might lead to better overall forecasts.

The motivation for this extension is that the simple average usually works well when the forecast errors are of relative similar size. As found by (Gu et al., 2020) there is quite some difference in forecasting performance and it may therefore be fruitful to consider other types of forecast combination which take this information into account. Supporting this hypothesis are the findings of Gupta and Wilton (1987), who find that the performance of equal weighted forecasts strongly depend on the relative sizes of the explantory power and when the differences are sufficiently large, different weights are generally desired. Because we have information about past performance (validation set) it might be fruitful to use this information in setting up the weights.

Using OLS in order to estimate the weights for each model is highly vulnerable for over-fitting. Timmermann (2006) and Diebold and Pauly (1990) find that shrinkage methods have performed quite well in empirical studies and that it improves the forecasting performance over single models or least squares combination weights. We therefore implement a LASSO meta-model to take care of this advantage. We also implement partial egalitarian LASSO (peLASSO) which first selects the best models and than shrinks the selection to average weights. The advantage of peLASSO is that it first removes worst performing models and than shrinks best to the known 'simple-average' and its advantages. Nevertheless, Timmermann (2006), Clemen and Winkler (1986), Dunis, Laws, and Chauvin (2001), Figlewski and Urich (1983), and Winkler and Makridakis (1983) find that generally simple combination schemes are difficult to beat. This is explained by the parameter estimation error in the combination weights. We therefore also introduce a new simple non-parametric forecast combination scheme based on past ranked weighted performance. The benefit of the model is that

little to no estimation has to be done and it is simple and intuitive to implement.

Another explanation for the increased performance of combining is that individual models are differently affected when deterministic shifts and structural breaks of the underlying data generating process occur. Pooling the forecasts mitigates those breaks and shifts, leading to increased performance (Diebold and Pauly (1987), Aiolfi and Timmermann (2006), Bates and Granger (1969), Makridakis (1989), and Figlewski and Urich (1983)).

Because the economy suffers from those structural changes and shifts, it may be fruitful to construct a meta-model which selects forecasts based on business cycle variables and tries to capture those changes. We therefore suggest an over-arching model which selects for each time period how much weight to assign to each model based on business cycle variables.

The meta-models just described will be further explained in the following subsections.

2.3.1 Simple Average

The most simple approach and implemented by (Gu et al., 2020) would be to perform an equally weighted ensemble using $\frac{1}{|M|}$ as weights in equation 8. This ensemble simply gives equal weights to all forecasts but has proven itself in the literature and will therefore also be included as benchmark.

2.3.2 LASSO ensemble

In order to select the models for our ensemble we can of course use the techniques applied before for variable selection. We implemented this by regressing the real risk premium on the stand-alone forecasts. Such a representation is shown in equation 9. We expect this however to be highly vulnerable for in-sample over-fitting. We therefore apply the LASSO penalty as seen in equation 5 and hypertune this penalty using the original train set. This LASSO model will then select which models to include. The meta-LASSO forecast can be calculated by using the models' coefficients as weights in equation 8. Models which are not selected have a coefficient of 0 and are therefore 'ignored' in this meta-model.

$$y_{RP} = \beta_1 \widehat{y}_{OLS3} + \cdots + \beta_{11} \widehat{y}_{NN5} \tag{9}$$

Because simple average does has proven itself often in the literature it might also be fruitful to consider a model which takes advantage of both LASSO and the simple average, that is: a model which first selects models and then shrinks them towards their average. This is done by (Diebold

& Shin, 2017) who tweak the original LASSO and shrink the selected models to an average and the not selected models to 0. The implementation is quite easy and described by Diebold and Shin as follows: We can obtain the egalitarian LASSO regression:

$$r_{i,t} \rightarrow_{\text{EgalLASSO}} g_{1,i,t}, \ldots, g_{K,i,t}$$

by running the standard LASSO regression on the forecast demeaned excess returns

$$(r_{i,t} - \bar{g}_t) \rightarrow_{\text{LASSO}} g_{1,i,t}, \ldots, g_{K,i,t}$$

That is, for each month and stock we subtract the average of forecasts of the excess return and regress this residual on the original forecasts using ordinary LASSO. Afterwards we fit this LASSO model and add back the subtracted average of forecasts. The weights of each model can be obtained by adding the coefficient of forecast demeaned excess returns and $\frac{1}{|M|}$ (weight of each element to obtain average). Now having obtained all weights of meta-peLASSO we can again calculate the meta forecast using formula 8.

2.3.3 Rank weighted average

The implementation of our rank weighted average is the following: we first obtain the R^2 of all models in the validation set and rank them from best to worse. Using the ranking obtained in the validation set we assign weights for each model using the formula in equation 10. Using the weights obtained we can then get our rank weighted forecast by equation 8

$$w_m(\cdot) = \frac{M + 1 - rank}{M(M+1)/2} \tag{10}$$

In equation 10, $w_m(\cdot)$ denotes the weight assigned to model m based on its $rank$, M denotes the total number of models considered. There is an inverse relation between rank and weight, that is models which perform best (rank 1,2,3,..) have higher weights and vice versa. The advantage of this method is that is both simple to implement and we don't need to estimate the weights using regressions.

2.3.4 Business cycle based ensemble

(Timmermann, 2006) show that ensemble models perform well because it reduces the effects of model specifications, parameter instability and model changes. We can imagine that underlying economic mechanisms shift from time to time and therefore also cause parameter instability for our

models. It may therefore be fruitful to construct a model which uses a gating network which uses the information on these economics states to assign weights for our stacked generalization. This can be done by creating a gating network which decides how much weight each model should receive in the ensemble based on some state variables. Because the original dataset only contains 8 macro-economic variables and already incorporated in the forecasts of the models we use the FRED-MD data set to train our Business cycle based ensemble which contains 106 monthly updated macro economic variables. The forecast are generated in a two step method:

1. Compute weights for models using macro economic variables
2. Compute weighted average of model predictions using weights obtained in step 1.

An illustration of this new method is shown in figure 1

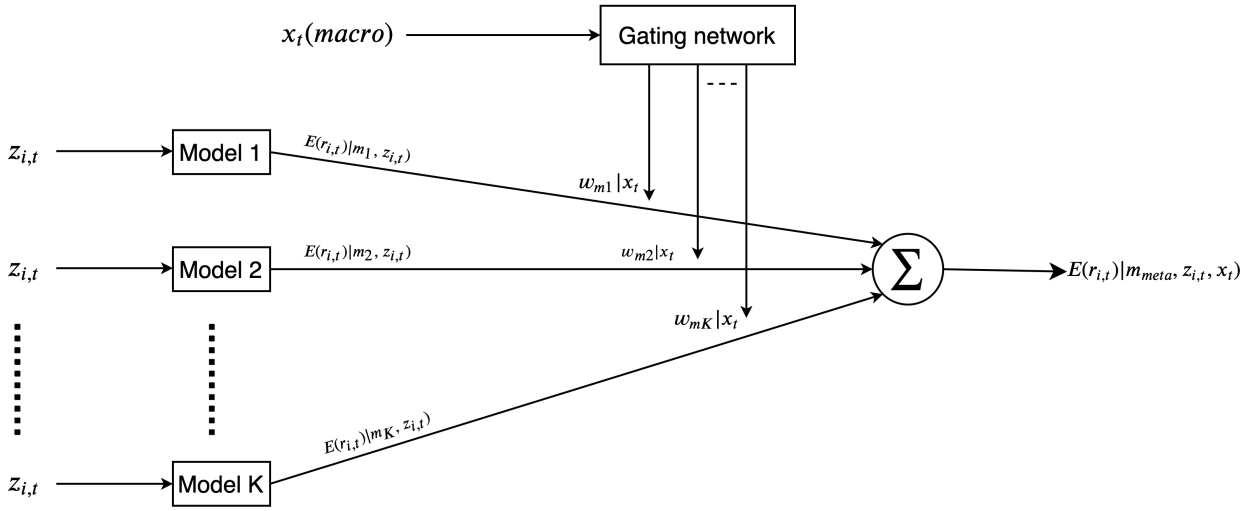

Figure 1
Non generative ensemble model.
Visualization of how a gating network assigns weights to forecasts of models based on macro economic variables and how those are combined into one forecast.

We choose a neural network as gating network with a 32-16-8 layer setup, following the pyramid schedule as suggested earlier ((Masters, 1993)). This model recombines the base learners (models) and select the most appropriate model(s) based on the macro-economic state. The gating network is trained on the validation set and hypertuned on the train set[5]. The model is trained as a

[5]Because we already used the original training set for training the individual model we do not use it for training

classification model where we classify for each period (month) which model performs best based in terms of monthly R^2, since we are dealing with classification we will use the Cross-Entropy loss instead of MSE as loss function. The variable to be explained and 'class' is therefore the type of model. We apply L1 regularization and take an ensemble of 10 of these models to get better out-of-sample performance. The hyperparameters: learning rate and L1 penalty are tuned in the train set. The business cycle variables are lagged with 1-month to prevent the forward looking bias.

The result of this classification model is that is a $M * 1$ vector with M the number of stand-alone models and the elements representing the probabilities that each corresponding stand-alone model will perform best this month based on the lagged business cycle variables. Using these probabilities we will create two stacked generalizations. Our first implementation of is to interpret the classification probabilities (which sum up to 1) as weights ($w(\cdot)$) and use those in the stacked generalization described in equation 8. This method is called Mixture of Experts (MoE) because it is a time varying mixture of expert models.

The other approach is to not consider a combination of all models but only the model expected to perform best. This transforms the mixture of experts to an expert selection model. The predicted class is the class with the highest probability. We use the transformation shown in equation 11 to make the output compatible with the general form of equation 8.

$$w_m(\cdot) = \begin{cases} 1, & \text{if } p_m = max(p_{OLS3}, \ldots, p_{NN5}) \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

2.4 Forecasting performance

In order to test the out-of-sample (OOS) performance, the adjusted R^2_{OOS} of Gu et al. (2020) is used which is shown in equation 12.

$$R^2_{OOS} = 1 - \frac{\sum_{(i,t)\in\tau_3}(r_{i,t+1} - \widehat{r_{i,t+1}})^2}{\sum_{(i,t)\in\tau_3} r^2_{i,t+1}} \tag{12}$$

This is an adjustment of the original formula of R^2 because in this form the denominator (sum of square risk premiums) is not demeaned. (Gu et al., 2020) argue that this form is better for

our gating network because it would then select the models with the best in-sample fit, we therefore train on the validation set which is out-of-sample for the individual models and use the original train dataset as validation set. The test set still never enters training and validation

measuring forecasting performance of individual stocks because the historical mean stock returns is quite noisy and therefore masks the individual forecast performance.

2.5 Statistical model comparison

Statistical comparison across models is done by using the Diebold Mariano test statistic ((Diebold & Mariano, 2002)) which enables pairwise performance testing. The null hypothesis of the test is that both models have equal performance. The test statistic is shown in equation 13 and represents the average difference of squared prediction error.

$$d_{m1,m2} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \left(\left(\widehat{e}_i^{(m1)} \right)^2 - \left(\widehat{e}_i^{(m2)} \right)^2 \right) \tag{13}$$

$d_{m1,m2}$ represent the Diebold and Mariano test statistic for model 1 (m1) and model 2 (m2), $\widehat{e}_i$ is the one-month ahead forecast error and n_{test} is the number of element in the test period.

2.6 Economical model comparison

In addition to statistical model comparisons which is interesting for the literature, we will also compare the meta-models in a economical framework, making it interesting for investors who look for models which can forecast their future returns better. We do so by creating a monthly resorted equally weighted top-bottom decile long-short spread.

The portfolios are compared using their Sharpe ratios (Sharpe (1994)). Because we don't have information about the mean and standard deviation of the Sharpe ratios we have to estimate them in order to make statements about significant differences. A popular approach is the test of Jobson and Korkie (1981) which is corrected by (Memmel, 2003). As mentioned by Ledoit and Wolf (2008) however, this test does not hold with fat tails. Because our stock data has fat tails I use the robust bootstrap inference method suggested by (Ledoit & Wolf, 2008).

The procedure to test if the difference of Sharpe ratios (Δ_{Sharpe}) is significant is the following: we test

$$H_0 : \Delta_{\text{Sharpe}} = 0$$

by inverting a bootstrap confidence interval. I construct this two-sided confidence interval by re-sampling the pairwise portfolio returns with replacement and same-size, following the procedure of Efron (1982). Using the re-sampled portfolios returns, I calculate the difference of Sharpe ratios and repeat this 10,000 times in order to get inference over the distribution of Δ_{Sharpe}.

I test the null hypothesis by a two-sided bootstrap 95% confidence interval for Δ. If zero lies outside this interval we reject the null hypothesis of same Sharpe ratio. The 95% confidence interval is given by:

$$\widehat{\Delta} \pm z_{95\%} s(\widehat{\Delta})$$

In this notation, $\widehat{\Delta}$ is the expected difference, and $s(\widehat{\Delta})$ is a standard error for $\widehat{\Delta}$, both computed from the bootstrap data. Ledoit and Wolf (2008) argue that when data are heavy-tailed, then $z_{1-\alpha}$ will typically be somewhat larger than $z_{1-\frac{\alpha}{2}}$ for small to moderate samples, resulting in conservative inferences. Since we have a small sample of only 10 years of monthly data (120 observations) we follow their approach and use $z_{1-\alpha}$ to construct the two-sided confidence interval. I perform this pairwise Sharpe ratio test for all possible model combinations in order to make statements about significant differences in portfolio selection performance.

3 Results

3.1 Machine learning models

performance for OLS with all predictors. Similar to Gu et al. (2020), we find that the neural networks perform best, and plain OLS performs badly. On the contrary, we find that the forests perform worse than dimension reduction and penalizing models, which are still quite competitive with the machine learning models. Overall we find a much higher explanatory power for almost all models compared to (Gu et al., 2020). This might be explained because we only use complete data and therefore have higher explanatory power. We find that the Neural Network with five layers performs better than NN3 and NN4, which is not in line with Gu et al. (2020) who find that NN3 performs best. This may also be caused because we only consider complete data and, therefore, end up with a neural network with more layers. (Gu et al., 2020) fill missing data with the cross-sectional monthly median, making NN5 possibly too complicated and causes in-sample overfitting whereas NN3 works better for their median filled data. We did not find that NN6 and NN7 performed better than NN5 and therefore left them out.

Table 1
Monthly out of sample R^2.

Models' out-of-sample one month ahead forecasting performance (test period) in terms of R2 for all stocks and for top and bottom 100 stocks per month in terms of size

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5
All	-0.009	-3.964	0.053	0.057	0.055	0.010	0.050	0.034	0.059	0.059	0.064
Top 100	-0.017	-13.519	0.077	0.042	0.065	0.015	0.062	0.032	0.079	0.069	0.091
Bottom 100	-0.003	-2.707	0.034	0.032	0.036	0.006	0.035	0.027	0.031	0.035	0.037

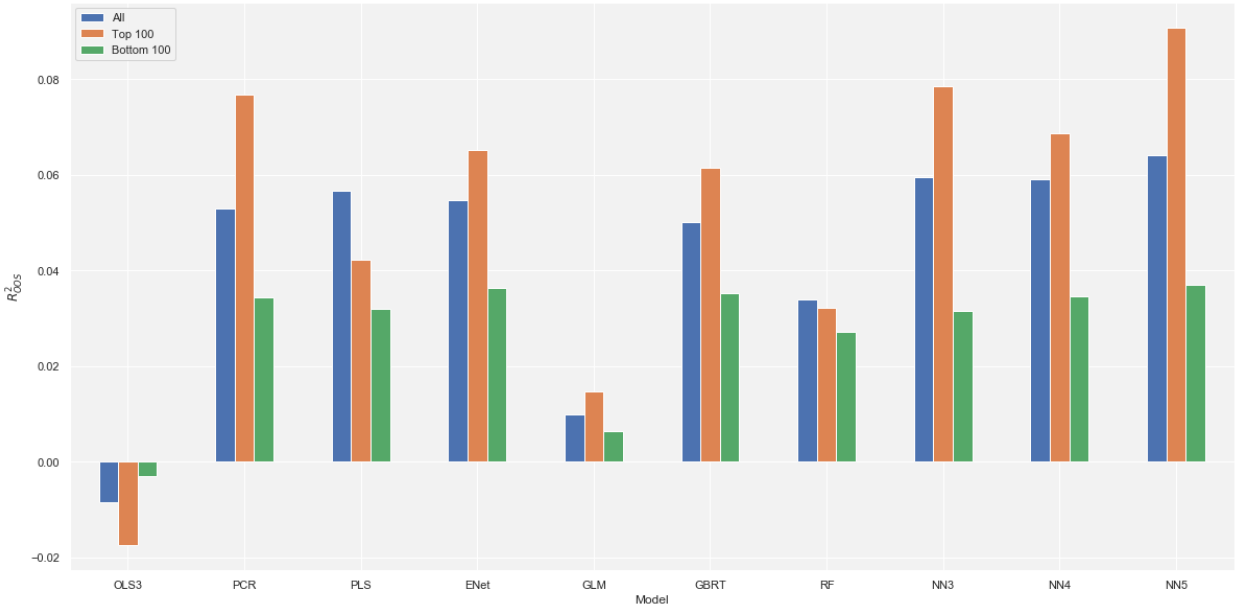

Figure 2
Models' R^2 for different sizes.

Models' one month out-of-sample ahead forecasting performance (test period) in terms of R^2 for all stocks and for top and bottom 100 stocks per month in terms of size.

In figure 3 the model's R^2 is plotted on a yearly basis over the full data range. The vertical black lines indicate the splits of train, validate and test set, respectively. The vertical shaded bands indicate times of recession. The graph shows clearly that OLS with Huber loss performs quite well in the train set (in-sample) but immediately plummets down in the validation set, indicating strong in-sample overfitting.

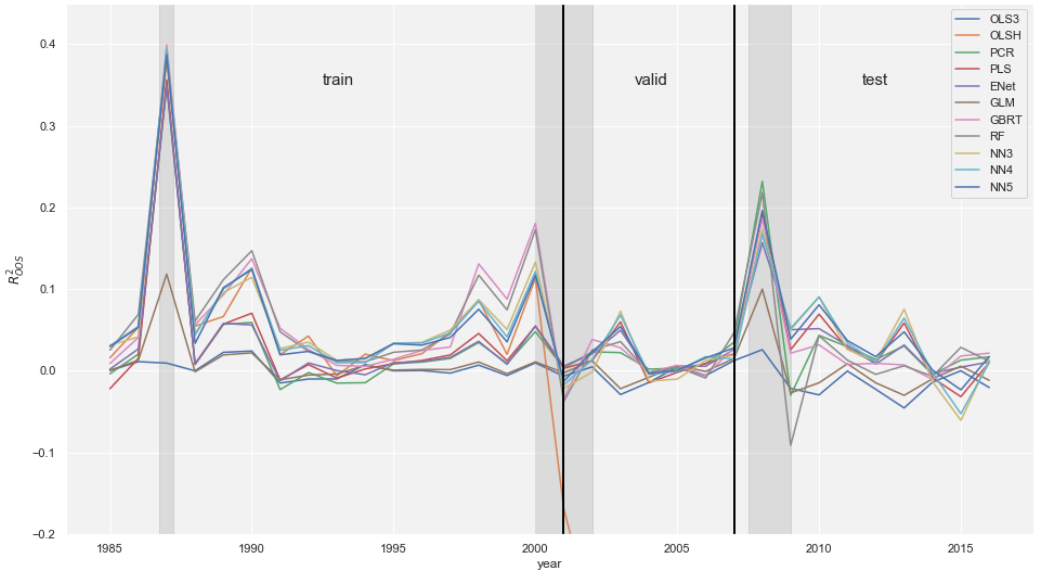

Figure 3
Models' yearly R^2.

Monthly R^2 plotted for each stand alone model during the train, validation and test period. Financial crisis are marked with vertical shaded areas and the train, validation and test period are split by the vertical black lines. We find a interesting spike of explanatory power at the beginning of each of the three recessions.

Another thing that stands out is the relation between explanatory power and recessions. When inspecting the yearly mean risk premiums (Figure 4) we find that the mean expected risk premiums alongside the real risk premiums move in negative territory just before and at the beginning of the crisis moves back up at the end of the recession to its normal value. This could be explained because the expected reward of taking risks in the financial crisis may be negative. More interesting is that during all three recessions (Black Friday 1987, Dot-com bubble and Financial crisis 2007-2008), a sharp increase of explanatory power is realized before and at the beginning of the crisis and is followed by a downward correction during and after the crisis. More generally, we see an inverse relationship between real equity premium and explanatory value, which is confirmed by checking the correlations which is in line with Garcia (2013). This shows how business cycles can really affect forecasting performance and is one more reason to consider the business cycle based stacked generalization.

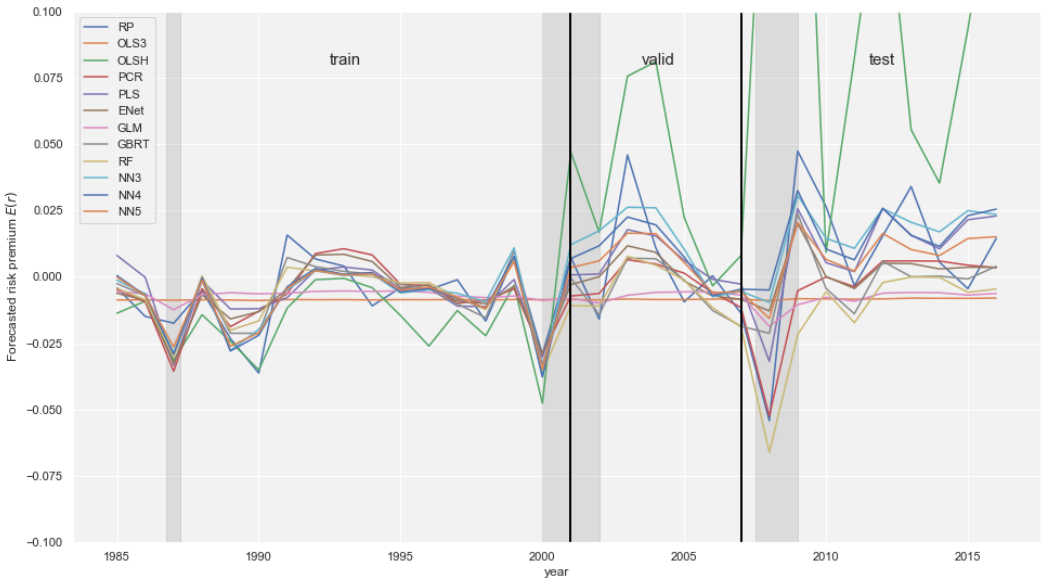

Figure 4
Models' yearly mean risk premium forecasts.

All yearly mean risk premium forecasts plotted alongside the real yearly mean risk premium denoted as RP in the legend. OLSH shows clear signs of overfitting because in the validation and testing period is moves far away from the real mean risk premium. Shaded areas indicate financial crisises and the vertical lines indicate the data splits; train, valid and test.

3.2 Ensembles

Now that all single model predictions are created, we can start combining them in cross-model ensembles. We start with the plain average ensemble and then build on to more complex models.

3.2.1 Simple average

When taking the simple average of predictions, we end up with an OOS R^2 of 0.03723, which is lower than most stand-alone models. This low R^2 could be explained by the OLS+H prediction, which has an OOS R^2 of -3.96377 but is given the same weight as the best performing model. This shows that taking the simple average could be improved with some form of prior model selection and confirms the observation of (Gupta & Wilton, 1987) that the R^2 should be of the same order and that otherwise different weights are desired.

3.2.2 LASSO ensemble

One way to estimate the weights is by using a regression approach. Using plain OLS, we find an OOS R^2 of 0.01595, lower than the simple average. The coefficients are shown in table 2, and coefficients like 5.164 and -3.538 indicate strong in-sample overfitting.

Table 2
Plain OLS ensemble weights.
Weights obtained when regressing the excess returns on OLS without any penalization. It becomes immediately clear that there is a lot of overfitting when we inspect the weights. Penalizing is therefore necessary.

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5
$w(\cdot)$	-3.538	0.024	0.177	0.136	1.123	5.164	0.501	-1.546	-0.191	1.162	-0.675

When incorporating the LASSO penalty, we find better results. We find an OOS R^2 of 0.06, which is much higher than the simple average. Moreover, compared to the stand-alone models it outperforms all except NN5. This is quite impressive since we never know ex-ante which model is going to perform well. The LASSO ensemble selects the following five models: OLSH, PLS, ENet, GBRT, and NN4. The LASSO model is trained on the validation period (2001 - 2006) of which the R^2 is shown in table 3, we would expect the models' weights to be related to the explanatory value. As expected we find that the models with the highest explanatory are included with higher weights and models with less explanatory value are assigned 0 or smaller weights. Interestingly it does not select NN5 instead of NN4 while NN5 has a higher explanatory power in the train set. Apparently not only the highest R^2 is important. A possible explanation is that NN4 explains a unique aspects of the asset returns which is not captured by the other models and that most of NN5's explanatory power is already covered by the other models, supporting the theory Hendry and Clements (2004).

Table 3
Lasso weights & validation R^2.
Meta-LASSO model coefficients and the models' R^2 during the period 2001 - 2006 on which the meta-LASSO model is trained. A positive relation between R^2 and the weights is found.

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5
$w(\cdot)$	0	0.03	0	0.22	0.22	0	0.22	0	0	0.15	0
R^2 valid	-0.0081	-0.4141	0.0115	0.0136	0.0162	-0.0026	0.0043	0.0014	0.0062	0.0122	0.0129

When we perform peLASSO and shrink the coefficients to equality we find similar weights, shown in table 4. As a result, we find a similar, but slightly lower R^2 of 0.05747. Since all weights

are so similar, the same inferences hold as for LASSO. (Diebold & Shin, 2017) also argue that when the LASSO weights of step 1 are already quite similar peLASSO produces similar results. The reason for slightly worse performance might be the increase of weight assigned to GLM, which performs not that well compared to the other models.

Table 4
Partial egalitarian LASSO weights.
Meta-peLASSO coefficient and the resulting weight after adding the subtracted average. the peLASSO coefficients are result of the forecast demeaned excess premium LASSO regression, step 1 as described by (Diebold & Shin, 2017). $w(\cdot)$ is obtained by adding back the mean weight.

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5
$w(\cdot)$	0	0.04	0	0.2	0.2	0	0.2	0	0	0.2	0

3.2.3 Rank weighted average

The nonparametric rank weighted average is easily obtained using the rank of explanatory power in the validation set and the formula described in equation 5. This results in the following ensemble weights:

Table 5
Rank weighted average.
Ranks of predicting performance in the validation period (2001 - 2006) and the corresponding weights obtained by equation 10. Rank 1 being the best model and 11 the worst in terms of R^2. An inverse relation between rank and weight exist due to the specification of equation 10

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5
$w(\cdot)$	0.03	0.015	0.106	0.152	0.167	0.045	0.076	0.061	0.091	0.121	0.136
rank	10	11	5	2	1	9	7	8	6	4	3

Using the weights obtained in table 5 we find an out-of-sample performance of 0.06665, beating all stand-alone and meta-models up until now. This result shows that the rank-weighted forecast combination can better cancel out biases than the other meta-models. Another explanation is that the rank weighted average gives enough weight to models with unique explanatory power but sufficiently low to models with very low explanatory power. This is illustrated by the difference of magnitude of the weights of the worst (0.015) and best (0.167) ranked models. Furthermore it remains one of the most intuitive and easy to implement ensembles up until now making it even more attractive, and also in line with the theory that simple ensembles often perform better than complicated ones (Timmermann (2006), Clemen and Winkler (1986), Dunis et al. (2001), Figlewski and Urich (1983), and Winkler and Makridakis (1983)).

3.2.4 Business cycle based ensemble

Because the weights of our business cycle based ensemble are time-varying, we can best depict them in an area plot shown in figure 5. We see that in the validation and test set (flipped from the original train and test set), all models have kind of similar weights except for NN3 and OLS3. In the train set, we find that the weights of many models shrink while the weights of NN3 and OLS3 become even more dominant. We see the same pattern in the test set around 2009. Both disturbances of weight occur right after a crisis suggesting that NN3 and OLS3 are good predictors at the end of financial distress. It is surprising to see a combination of one of the most sophisticated models as NN3 with the most simple model using only three variables. The combination does make sense however because those models occurred often as monthly best forecasts in the training set. After the financial crisis of 2007 all weights stabilize which can be explained by the fact that the business cycle variables do not vary a lot over this period.

The performance of the Mixture-of-Experts model is quite low with an R^2 of only 0.0481, easily beaten by some of the stand-alone models. An explanation for this low explanatory power is that over the whole test period all weights are almost equal, only deviating right after the financial crisis of 2007 where NN3 gets some more weight, When we interpolate between the R^2 of NN3 and meta-average we can understand how we get at this number.

The expert-selecting meta-model select almost exclusively NN3 as best model and achieves an R^2 of around 0.0593. This value makes sense because it is close to the R^2 of NN3. This model however does not have any gain of variance reduction or bias canceling properties, hence has no real advantage over any of the neural networks.

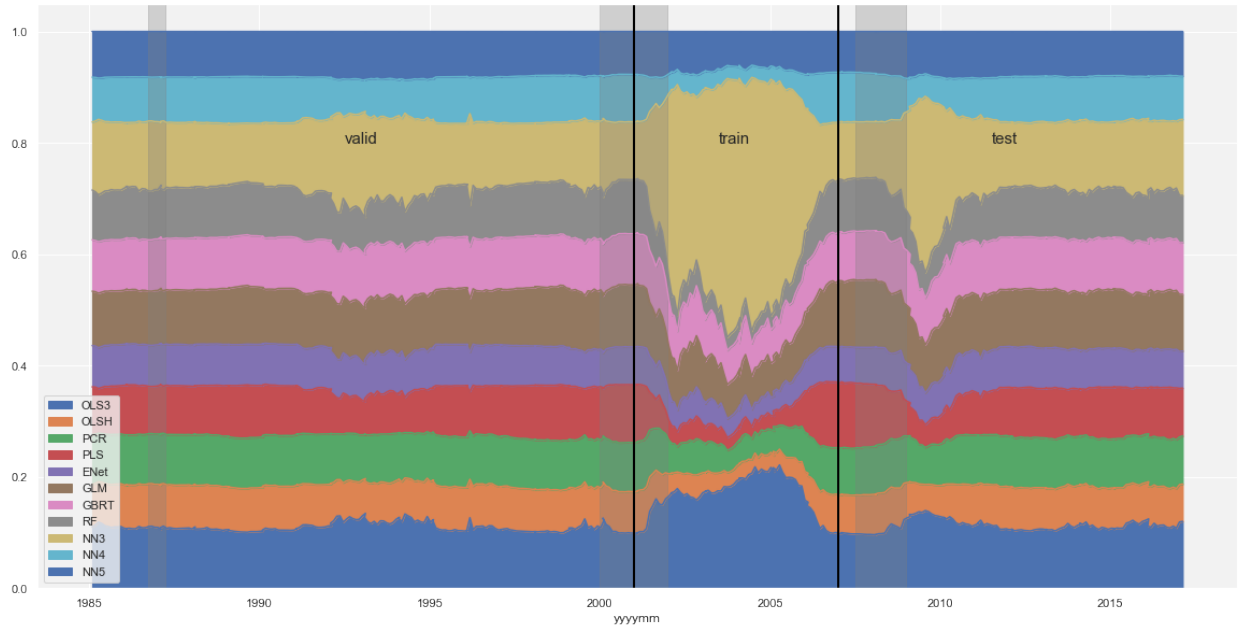

Figure 5
Mixture of expert weights.

This figure shows the weights decided by the gating network which uses FRED-MDs macroeconomic variables to decide how much weight to give to each model. For each time period the vertical height of each of the model correspond to the weight given in the Mixture of experts model. With the voting model we take the prediction of the model with the most vertical height (weight) given a time period. The original validation period and train period have been swapped in order to prevent training on in-sample fit.

Overall we find that the rank meta-model performs better than the individual models with ex-ante decided weights. LASSO and peLASSO achieve similar performance similar to the best performing stand alone models. The business cycle based ensemble performs worse than the best stand-alone models. All meta models outperform the simple-average ensemble.

Table 6
Overview of out-of-sample R^2 of stand-alone models and meta-models.

Table with explanatory power in terms of R^2 over the testing period Januari 2007 until December 2016

Model:	OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5	meta_average	meta_lasso	meta_pel	meta_ranked	meta_moe	meta_expert
R^2	-0.0086	-3.9638	0.0529	0.0567	0.0547	0.0099	0.0502	0.034	0.0595	0.0591	0.0641	0.0372	0.06	0.0575	0.0667	0.0481	0.0593

3.3 Statistical gains: Diebold Mariano

Following the methodology of (Gu et al., 2020) we not only compare the models R^2 but also formally test the significance of pairwise differences in out-of-sample predictive accuracy between each combination of models using the Diebold and Mariano test (Diebold & Mariano, 2002). The results are shown in table 7. Positive numbers indicate that the row model outperforms the column

model and vice versa. Asterisks indicate significant differences. The table shows clearly that of all stand-alone models, the NN5 model performs best, having significantly better performance over all others. NN3 and NN4 also show significantly better performance than all remaining stand-alone models except for PLS. The regression trees do not seem to produce significant better forecasts than the other stand-alone models. This result is in line with the relatively low R^2 we find for our regression trees but difference from the result found in (Gu et al., 2020).

When we also consider the meta-models, we find that the meta-LASSO model outperforms all stand-alone models except NN5, but its performance is not significantly better than NN3 and NN4. The meta-peLASSO model shares the same pairwise results which makes sense because of the similar weights. Meta-ranked significantly outperforms all stand-alone and meta models, in line with its superior R^2. The meta-MoE model performs mediocre like meta-average which is also expected with their below average R^2. Meta-expert outperforms all non neural-networks just like NN3 because it primarily picks NN3 as the best model. Hence, we are able to outperform the stand-alone models in terms of the Diebold-Mariano statistic, answering our second sub-question: *Could we achieve statistically significant better performance using a meta-strategy?*

	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5	meta_average	meta_lasso	meta_pel	meta_ranked	meta_moe	meta_expert
OLS3	46.93*	-20.07*	-16.76*	-31.25*	-27.06*	-24.86*	-13.33*	-22.54*	-22.67*	-28.97*	-14.96*	-29.03*	-26.55*	-30.28*	-21.62*	-22.6*
OLSH		-47.55*	-47.73*	-47.72*	-47.12*	-47.66*	-47.31*	-47.79*	-47.8*	-47.82*	-48.07*	-47.88*	-47.91*	-47.87*	-47.97*	-47.79*
PCR			-2.07*	-0.7	16.82*	1.07	12.41*	-2.74*	-2.56*	-5.55*	5.01*	-3.1*	-1.83	-7.86*	2.28*	-2.8*
PLS				0.72	13.35*	2.21*	7.78*	-1.33	-1.14	-3.42*	6.46*	-1.4	-0.31	-5.11*	3.64*	-1.4
ENet					25.72*	3.03*	7.06*	-2.85*	-2.82*	-7.62*	8.3*	-5.21*	-2.38*	-10.18*	4.92*	-2.91*
GLM						-19.71*	-9.12*	-18.33*	-18.42*	-25.6*	-9.57*	-24.48*	-21.57*	-27.44*	-16.12*	-18.39*
GBRT							5.77*	-4.33*	-4.39*	-8.17*	5.6*	-7.32*	-4.88*	-10.71*	1.72	-4.37*
RF								-7.81*	-7.75*	-11.12*	-0.89	-8.94*	-7.58*	-13.11*	-4.1*	-7.84*
NN3									0.62	-4.96*	10.94*	-0.4	1.57	-6.62*	7.97*	-0.32
NN4										-6.11*	11.17*	-0.78	1.44	-7.59*	8.13*	-0.72
NN5											13.65*	4.52*	6.31*	-4.33*	12.52*	4.75*
meta_average												-16.46*	-17.6*	-16.75*	-13.47*	-10.97*
meta_lasso													9.46*	-10.69*	16.07*	0.32
meta_pel														-11.31*	16.56*	-1.64
meta_ranked															17.1*	6.53*
meta_moe																-8.02*

Table 7
Pairwise Diebold and Mariano statistics.

Pairwise Diebold and Mariano statistics, positive numbers indicate that the column model outperforms the row model in terms of forecasts. Asterisks indicate significant differences according to the Diebold and Mariano test.

3.4 Financial gains: top-bottom decile spread

Attractive to investors is not only the statistical advantages but also the financial advantages of machine learning models and meta models. The annualized Sharpe ratios are reported in table 8. Monthly average and standard deviation of returns of the portfolios are denoted with AVG and SD

respectively, and the annual Sharpe ratio is denoted as SR. The annual Sharpe ratio is calculated by dividing the yearly average return by the yearly volatility obtained by the square-root-of-time rule, as done by Gu et al..

We find the meta models perform best in this category. The meta-peLASSO performs best with a Sharpe ratio of 1.30 and is closely followed by the meta-MoE (1.27) and Rank weighted average (1.22) meta models. Generally we find that the forecast combination models work better than the stand alone model. The expert selecting meta model performs again similar to NN3 by construction. Even the worse performing meta-average performs similar to the best performing stand-alone models. The prediction variance reduction argument of Granger (1989), (Newbold & Harvey, 2002) and Aiolfi and Timmermann (2006) might explain why all meta-models perform so well in this test because the meta-models create portfolios with relatively lower volatility.

Table 8
Portfolio's Sharpe ratios.

Monthly average returns and volatility, and annualized Sharpe ratio calculated with the square-root-of-time rule. Statistics for the lowest decile portfolio (decile 1), highest decile portfolio (decile 10) and of the high minus low portfolio (going long in decile 10 and short in decile 1). Decile 10 being the highest expected returns and decile 1 being the lowest expected returns. H-L is the long short portfolio going long in decile-10 and going short in decile-1. The portfolios are equally re-weighted every month.

		OLS3	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5	meta_average	meta_lasso	meta_pel	meta_ranked	meta_moe	meta_expert
Decile 1 (L)	AVG	0.58	0.12	0.02	-0.21	-0.21	0.62	-0.01	-0.07	-0.22	-0.45	-0.04	-0.28	-0.32	-0.35	-0.27	-0.45	-0.20
	SD	7.98	5.41	7.23	6.91	6.10	5.19	6.57	7.32	6.55	6.45	6.82	5.99	6.17	6.11	6.45	6.06	6.55
	SR	0.25	0.08	0.01	-0.11	-0.12	0.42	-0.01	-0.03	-0.12	-0.24	-0.02	-0.16	-0.18	-0.20	-0.15	-0.26	-0.11
Decile 10 (H)	AVG	0.64	0.58	0.98	1.09	0.64	0.01	0.93	1.20	1.24	1.24	1.30	0.80	1.10	1.17	1.28	0.81	1.21
	SD	5.30	6.80	6.12	5.98	7.06	6.03	6.63	6.89	7.68	7.62	7.67	6.65	6.58	6.70	6.65	6.63	7.70
	SR	0.42	0.30	0.56	0.64	0.32	0.01	0.49	0.61	0.56	0.56	0.59	0.42	0.58	0.60	0.67	0.42	0.55
H-L	AVG	0.03	0.23	0.48	0.65	0.43	-0.30	0.47	0.64	0.73	0.84	0.67	0.54	0.71	0.76	0.78	0.63	0.71
	SD	2.31	1.74	2.14	2.03	2.51	2.13	1.86	2.35	2.43	2.53	2.33	1.75	2.09	2.02	2.21	1.72	2.45
	SR	0.04	0.46	0.78	1.12	0.59	-0.49	0.87	0.94	1.04	1.16	1.00	1.08	1.17	1.30	1.22	1.27	1.00

To test if the differences in Sharpe ratios are significant, we perform the bootstrap procedure of Ledoit and Wolf (2008). The results are in table 9. Positive signs indicate that the row model outperforms the column model. Asterisks indicate significant differences according to the bootstrap inference. We do not find any of meta-models which significantly outperforms the stand-alone models. More generally, we do not find a lot of significant differences. This is most likely due to the small sample (120) used for re-sampling, leading to conservative intervals. In summary, we find that the meta-ranked, meta-LASSO, meta-peLASSO, and meta-MoE all outperform the stand-alone models, but not significantly, answering our third and final sub-question: *Can we outperform the stand-alone models in a top-down decile portfolio Sharpe ratio?*

Table 9
Difference of Sharpe ratios and significance.

Share ratio differences and significance based on bootstrap inference using 10,000 re-samples with replacement. Differences shown in table are $Sharpe_{row} - Sharpe_{column}$. Asterisks indicate significant differences of Sharpe ratio based on the bootstrap inference.

	OLSH	PCR	PLS	ENet	GLM	GBRT	RF	NN3	NN4	NN5	meta_average	meta_lasso	meta_pel	meta_ranked	meta_moe	meta_expert
OLS3	-0.41	-0.74*	-1.07*	-0.55	0.54	-0.83*	-0.9*	-1.0*	-1.12*	-0.96*	-1.03*	-1.13*	-1.26*	-1.18*	-1.23*	-0.96*
OLSH		-0.33	-0.66	-0.13	0.95*	-0.42	-0.48	-0.59	-0.71*	-0.54	-0.62*	-0.72*	-0.85*	-0.77*	-0.82*	-0.54
PCR			-0.33*	0.19	1.28*	-0.09	-0.16	-0.26	-0.38	-0.22	-0.29	-0.39	-0.52	-0.44*	-0.49	-0.22
PLS				0.53	1.61*	0.24	0.18	0.08	-0.05	0.12	0.04	-0.06	-0.19	-0.11	-0.16	0.12
ENet					1.09*	-0.28	-0.35	-0.45*	-0.57*	-0.41*	-0.49	-0.58*	-0.71*	-0.63*	-0.68*	-0.41
GLM						-1.37*	-1.44*	-1.54*	-1.66*	-1.5*	-1.57*	-1.67*	-1.8*	-1.72*	-1.77*	-1.5*
GBRT							-0.07	-0.17	-0.29	-0.13	-0.2	-0.3	-0.43	-0.35	-0.4	-0.13
RF								-0.1	-0.22	-0.06	-0.14	-0.23	-0.36	-0.28	-0.33	-0.06
NN3									-0.12	0.04	-0.04	-0.13	-0.26	-0.18	-0.23	0.04
NN4										0.16	0.09	-0.01	-0.14	-0.06	-0.11	0.16
NN5											-0.08	-0.17	-0.3	-0.22	-0.27	-0.0
meta_average												-0.1	-0.23	-0.15	-0.2*	0.08
meta_lasso													-0.13	-0.05	-0.1	0.17
meta_pel														0.08	0.03	0.3
meta_ranked															-0.05	0.22
meta_moe																0.27

The accumulated returns of the top-bottom decile spread portfolio over time are shown in figure 6. Interestingly we find quite some models making profits during the financial crisis. The profits from the short positions explain this rise, however. We find that after 2012 almost all models stagnate in their position, this could be explained by the relatively low R^2 of this period, which could be seen in figure 3 where many models drop in negative territory or shrink towards zero in the last year(s). One explanation is that the last years are furthest away from the train data set, and because of model instabilities, the models do not produce as accurate forecasts anymore, confirming the hypothesis of time-varying trends.

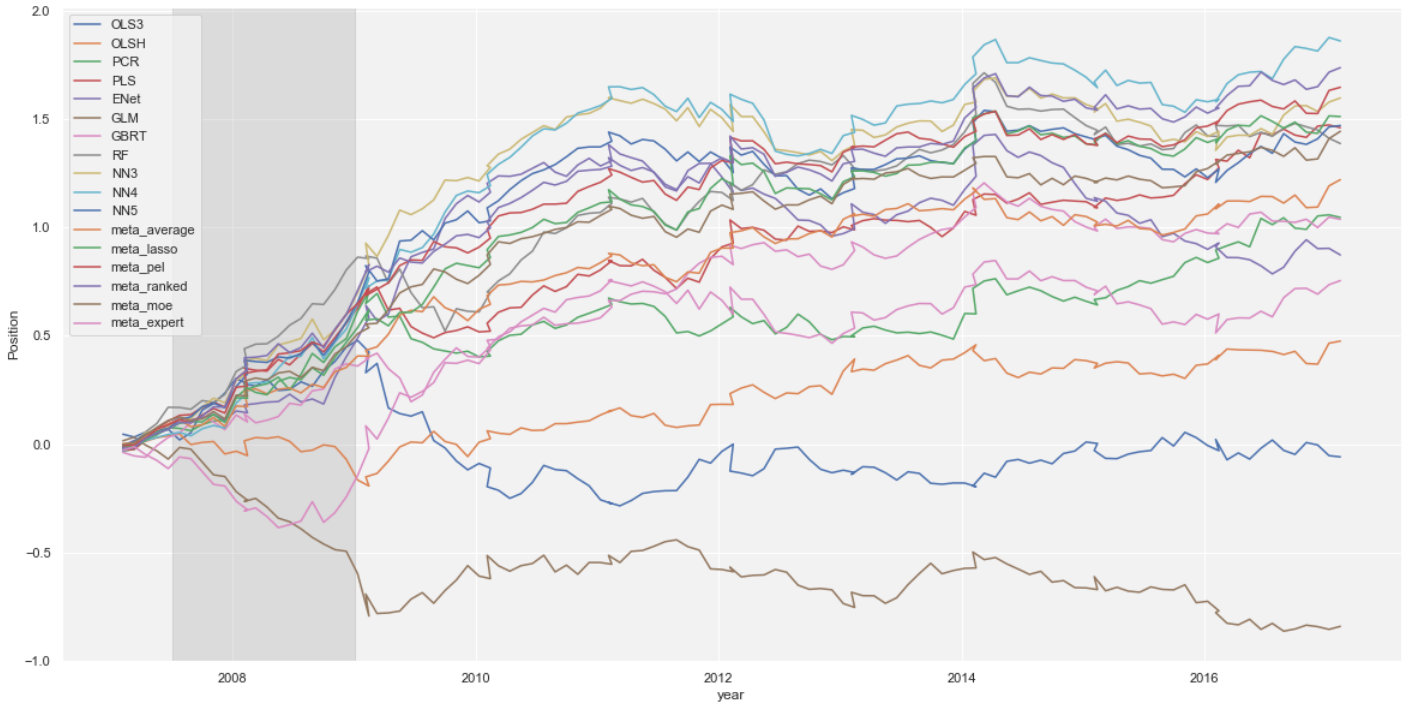

Figure 6
Decile spread position over time.

This figure shows the result of accumulated returns of a top minus bottom decile long-short portfolio. Stocks are equally weighted and re-selected each month. The position indicates the accumulated returns. Since we start with a netto zero investment we start at 0, over time we accumulate the returns. A end position of 1.74 thus indicates 174% return on the initial invested and shorted amount of money.

4 Conclusion

The goal of this research was to find an answer to the research question: *Can we improve asset pricing forecasts by implementing meta-strategies that combine best-performing machine learning models?*

We suggest two novel approaches in the field of asset pricing and ensembles. The first is a forecast combining model which picks and combines forecasts of the stand-alone models using monthly business cycle data. The other is a rank-weighted average, giving more weight to the past best performing models. Furthermore, we test LASSO and peLASSO and the simple-average forecast combinations.

In order to answer our research question, we first answer our three sub-questions. We find that the rank weighted average ensemble outperforms all models in terms of R^2. LASSO, peLASSO, and the expert-selecting model are among the best stand-alone models. Hence, improving the R^2 using a meta-model is possible.

Secondly, We test if the improvements made over the stand-alone models are significant. We find that the rank-weighted meta-model creates significantly better forecasts than all other stand-alone and meta-models according to the pairwise Diebold and Mariano test.

Finally, we test the financial performance in terms of Sharpe ratios. We find that in a top-bottom decile spread portfolio, several meta-models obtain higher Sharpe ratios than all individual models. According to the bootstrap inference however, we find that the improvements are not significant.

Considering the outcome of the three sub-questions, we find that meta-models can outperform stand-alone machine learning models in asset-pricing forecasts. Outperforming is done by using weights based on past performance. A meta-model that combines forecasts based on business-cycle variables has not proven to be of much use, probably because most of the models already incorporate the essential macro-economic variables.

5 Appendix

Code used in this thesis is provided on the next pages.

REPRODUCTION

Imports

In [ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import json
import math
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from tensorflow.keras.models import load_model
from scipy import stats
import seaborn as sns

sns.set()
```

Options

In [ ]:
```
pd.set_option('max_columns', 100)
```

Data collecting & matching

Macros

- including dividend-price ratio (dp) = the difference between the log of dividends and the log of prices
- earnings-price ratio (ep) = is the difference between the log of earnings and the log of prices
- book-to-market ratio (bm)
- net equity expansion (ntis)
- Treasury-bill rate (tbl)
- term spread (tms) = the difference between the long term yield on government bonds and the Treasury-bill.
- default spread (dfy) = the difference between BAA and AAA-rated corporate bond yields
- variance (svar)

In [ ]:

```
# Load DataShare characteristics Gu et al.
datashare = pd.read_csv('datashare.csv')
print("datashare loaded")
datashare.dropna(inplace=True)
datashare.rename(columns={'permno':'PERMNO'}, inplace=True)
datashare['sic2'] = datashare[['sic2']].astype(int)
datashare = pd.get_dummies(datashare, columns=['sic2'], prefix_sep=
'_dummy_')

# Load CRSP returns
crsp = pd.read_csv('crsp.csv')
print("crsp loaded")
str_cols = ['RET']
crsp.rename(columns={'date':'DATE'}, inplace=True)
crsp[str_cols] = crsp[str_cols].apply(pd.to_numeric, errors='coerce
')
df = crsp.merge(datashare, how = 'inner', on = ['PERMNO', 'DATE'])
df['yyyymm'] = df.DATE.apply(lambda x: int(x/100))
df.drop(columns=['DATE'], inplace = True)

macros = pd.read_excel('macro_predictors.xlsx')
print("macros loaded")
macros['macro_dp'] = np.log(macros.D12) - np.log(macros.Index)
macros['macro_ep'] = np.log(macros.E12) - np.log(macros.Index)
macros['macro_tms'] = macros.lty - macros.tbl
macros['macro_dfy'] = macros.BAA - macros.AAA
macros.drop(columns=['Index', 'D12', 'E12', 'AAA', 'BAA', 'lty', 'R
free', 'infl', 'ltr', 'corpr', 'CRSP_SPvw','CRSP_SPvwx', 'csp'], in
place = True)
macros.rename(columns={'b/m':'macro_bm', 'ntis':'macro_ntis', 'svar
':'macro_svar', 'tbl':'macro_tbl'}, inplace=True)
df = pd.merge(df, macros, how='left', on='yyyymm')
print("all data merged")

# Fill na with cross sectional month median
# months = list(set(df['yyyymm']))
# months.sort()
# excludecols = ['PERMNO', 'DATE', 'RET']
# columns = [c for c in list(df.columns) if c not in excludecols]
# monthdfs = []
# for m in months:
#     if m % 500 == 1:
#         print('filling nans of', int(m/100))
#     monthdf = df.loc[df['yyyymm'] == m]
#     fillna_dict = {}
#     for c in columns:
#         cross_sec_month_median = np.nanmedian(monthdf[c])
#         fillna_dict[c] = cross_sec_month_median
#     monthdfs.append(monthdf.fillna(value=fillna_dict))
# df = pd.concat(monthdfs, ignore_index=True)
# print("all nans filled with cross sectional month median")

df['RP'] = df['RET'] - ((1+df['macro_tbl'])**(1/3)-1)

df.dropna(inplace=True)

stock_characteristics = ['mvel1', 'beta', 'betasq', 'chmom', 'dolvo
```

```
l', 'idiovol', 'indmom',
       'mom1m', 'mom6m', 'mom12m', 'mom36m', 'pricedelay', 'turn',
'absacc',
       'acc', 'age', 'agr', 'bm', 'bm_ia', 'cashdebt', 'cashpr', 'c
fp',
       'cfp_ia', 'chatoia', 'chcsho', 'chempia', 'chinv', 'chpmia',
'convind',
       'currat', 'depr', 'divi', 'divo', 'dy', 'egr', 'ep', 'gma',
'grcapx',
       'grltnoa', 'herf', 'hire', 'invest', 'lev', 'lgr', 'mve_ia',
'operprof',
       'orgcap', 'pchcapx_ia', 'pchcurrat', 'pchdepr', 'pchgm_pchsa
le',
       'pchquick', 'pchsale_pchinvt', 'pchsale_pchrect', 'pchsale_p
chxsga',
       'pchsaleinv', 'pctacc', 'ps', 'quick', 'rd', 'rd_mve', 'rd_s
ale',
       'realestate', 'roic', 'salecash', 'saleinv', 'salerec', 'sec
ured',
       'securedind', 'sgr', 'sin', 'sp', 'tang', 'tb', 'aeavol', 'c
ash',
       'chtx', 'cinvest', 'ear', 'nincr', 'roaq', 'roavol', 'roeq',
'rsup',
       'stdacc', 'stdcf', 'ms', 'baspread', 'ill', 'maxret', 'retvo
l',
       'std_dolvol', 'std_turn', 'zerotrade']
macro_columns = ['macro_bm', 'macro_tbl', 'macro_ntis', 'macro_svar
','macro_dp', 'macro_ep', 'macro_tms', 'macro_dfy']

for mc in macro_columns:
    for sc in stock_characteristics:
        df['{}_{}'.format(mc, sc)] = df[mc] * df[sc]

df.drop(columns=macro_columns, inplace=True)
df.sort_values(by='yyyymm')
df.reset_index(inplace=True, drop=True)
```

Inspect Data

In [ ]:
```
df.drop(columns=macro_columns, inplace=True)
df.columns
```

In [ ]:
```
print(len(stock_characteristics) * (len(macro_columns) + 1) + 61)
```

In [ ]:
```
for c in df.columns:
    print(c)
```

Transform & split data

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import feather

df.sort_values('yyyymm', inplace=True)
df.reset_index(inplace=True, drop=True)

testidx = df.loc[df['yyyymm'] > 200900].index[0] #X.loc[X['yyyymm']
> 201300].index[0]
valididx = df.loc[df['yyyymm'] > 200100].index[0]

y = df['RP']
X = df.drop(['PERMNO','yyyymm','RET','RP'], axis=1)
xcols = X.columns

scaler = StandardScaler()
scaler.fit(X)

X_trainvalid = X.iloc[:testidx]
X_trainvalid = pd.DataFrame(scaler.transform(X_trainvalid))
X_trainvalid.columns = xcols
y_trainvalid = y.iloc[:testidx]

X_train = X_trainvalid.iloc[:valididx]
y_train = y.iloc[:valididx]

X_valid = X_trainvalid.iloc[valididx:]
y_valid = y.iloc[valididx:testidx]

X_test = X.iloc[testidx:]
X_test = pd.DataFrame(scaler.transform(X_test))
X_test.columns = xcols
y_test = y.iloc[testidx:]


def OOS_R2(y, ypred):
    y = np.array(y)
    ypred = np.array(ypred)
    R2 = 1 - (np.sum((y-ypred)**2) / np.sum(y**2))
    return round(R2,ndigits=5)
```

In [ ]:
```
X_trainvalid
```

Inspect Data

correlations

In [ ]:
```
corr = df.drop(columns=['PERMNO','DATE','RP']).corr()
# corr = corr[['RET']]

fig, ax = plt.subplots(figsize=(22,22))

ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=9),
    square=True,
    xticklabels=1,
    yticklabels=1
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```

plots

In [ ]:
```
df['RP'].hist(bins=100, figsize=(20,10))
```

In [ ]:
```
macro_time = macros['yyyymm']/100 + (macros['yyyymm']%100)/12
macros[['macro_bm', 'macro_tbl', 'macro_ntis', 'macro_svar', 'macro
_dp', 'macro_ep', 'macro_tms', 'macro_dfy']].plot(figsize=(20,10),
title='Macros-economic variables')
```

Models

Splitting data: 80-10-10, 70-20-10, 70-15-15 Splitting data: seed Layers: 32-16, 64-32-16, 128-64-32-16

OLS size, book-to-market, and momentum

In [ ]:
```
import statsmodels.api as sm

X3_train = X_trainvalid[['mvel1', 'bm','mom1m']]
X3_test = X_test[['mvel1', 'bm','mom1m']]

OLS3H_model = sm.RLM(y_trainvalid, X3_train, M=sm.robust.norms.Hube
rT())
OLS3H_result = OLS3H_model.fit()
print(OLS3H_result.summary2())
print(OOS_R2(y_test, OLS3H_result.predict(X3_test)))
```

OLS ALL

In [ ]:
```
import statsmodels.api as sm

OLS_model = sm.OLS(y_trainvalid, sm.add_constant(X_trainvalid))
OLS_result = OLS_model.fit()
print(OLS_result.summary2())
print(OOS_R2(y_test, OLS_result.predict(sm.add_constant(X_test))))

OLSH_model = sm.RLM(y_trainvalid, sm.add_constant(X_trainvalid), M=
sm.robust.norms.HuberT())
OLSH_result = OLSH_model.fit()
print(OLSH_result.summary2())
print(OOS_R2(y_test, OLSH_result.predict(sm.add_constant(X_test))))
```

PCA

In [ ]:

```
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
import statsmodels.api as sm


components = range(50,85)

#Hyper tuning
for c in components:
    pca = PCA(c)

    pcaX_train = pca.fit_transform(X_train)
    pcaX_train = pd.DataFrame(pcaX_train)
    pcaX_train = sm.add_constant(pcaX_train)
    pcaX_train.index = y_train.index

    pcaX_valid = pca.transform(X_valid)
    pcaX_valid = pd.DataFrame(pcaX_valid)
    pcaX_valid = sm.add_constant(pcaX_valid)

    pcaX_test = pca.transform(X_test)
    pcaX_test = pd.DataFrame(pcaX_test)
    pcaX_test = sm.add_constant(pcaX_test)

    pca_model = sm.OLS(y_train, pcaX_train).fit()

    y_valid_pred = pd.DataFrame(pca_model.predict(pcaX_valid))
    y_valid_pred.index = y_valid.index
    valid_ols_model = sm.OLS(y_valid, sm.add_constant(y_valid_pred)
).fit()
    valid_r2 = round(valid_ols_model.rsquared, 5)

    y_pca_first_pred = pca_model.predict(pcaX_test)
    y_pca_pred = valid_ols_model.predict(sm.add_constant(y_pca_firs
t_pred))

    print("components:",c,valid_r2,OOS_R2(y_test, y_pca_pred))


# Huber
pca = PCA(82)
huber_pca_model = sm.RLM(y_train, pcaX_train, M=sm.robust.norms.Hub
erT()).fit()
#Hyper tuning
y_valid_pred = pd.DataFrame(huber_pca_model.predict(pcaX_valid))
y_valid_pred.index = y_valid.index
valid_ols_model2 = sm.OLS(y_valid, sm.add_constant(y_valid_pred)).f
it()
valid_r2 = round(valid_ols_model2.rsquared, 5)

y_pca_h_first_pred = huber_pca_model.predict(pcaX_test)
y_pca_h_pred = valid_ols_model2.predict(sm.add_constant(y_pca_h_fir
st_pred))

print(OOS_R2(y_test, y_pca_h_pred))
```

PLS

In [ ]:
```
from sklearn.cross_decomposition import PLSRegression

for c in range(5,51,5):
    pls = PLSRegression(n_components=c)
    pls.fit(X_train, y_train)

    y_valid_pred_pls = pls.predict(X_valid).flatten()
    y_test_pred_pls = pls.predict(X_test).flatten()

    print('Components: {} R2 Valid: {} R2 Test: {}'.format(c, OOS_R
2(y_valid, y_valid_pred_pls), OOS_R2(y_test, y_test_pred_pls)))
```

In [ ]:
```
pls.predict(X_valid).flatten()
```

In [ ]:
```
print(pls_aux_model.summary())
```

In [ ]:
```
pls_aux_model.predict(sm.add_constant(y_pred1_pls))
```

Elastic Net

In [ ]:
```
from sklearn.linear_model import ElasticNet
from sklearn.datasets import make_regression

# hyper tuning
for a in [0.1, 0.01, 0.001, 0.0001]:
    for l in range(1,10,1):
        l = l/10
        regr = ElasticNet(alpha=a, l1_ratio=l)
        regr.fit(sm.add_constant(X_train), y_train)
        print("alpha: {0: <6}, l1: {1: <3}, Valid R2: {2} {3}".form
at(a,l,OOS_R2(y_valid,regr.predict(sm.add_constant(X_valid))), OOS_
R2(y_test,regr.predict(sm.add_constant(X_test)))))

# alpha: 0.001 l1: 0.7

regr = ElasticNet(alpha=0.001, l1_ratio=0.7)
regr.fit(sm.add_constant(X_train), y_train)
y_elasticnet = regr.predict(sm.add_constant(X_test))
OOS_R2(y_test,y_elasticnet)
```

Spline GMM

In [ ]:

```
from sklearn import linear_model
import statsmodels.api as sm

# returns transformed data (knots on quantiles and take square)
def getSplineData(X, knots):

    X = X.copy()

    cols = list(X.columns)
    for c in cols:
        for k in range(1, knots+1):
            # quantile
            q = k/(knots+1)
            X['{}-q{:.0f}_sq'.format(c,100*q)] = (X[c] - X[c].quant
ile(q))**2
    return X

# Hyper tuning
for k in [1,2,3,4,5]:
    for a in [0.1, 0.01, 0.001, 0.0001]:
        X_train_spline = getSplineData(X_train, k)
        X_valid_spline = getSplineData(X_valid, k)
        X_test_spline = getSplineData(X_test, k)
        glm = linear_model.Lasso(alpha=a).fit(sm.add_constant(X_tra
in_spline), y_train)

        y_valid_pred = glm.predict(sm.add_constant(getSplineData(X_
valid, k)))
        y_test_pred = glm.predict(sm.add_constant(getSplineData(X_t
est, k)))

        print("knots: {0: <3} alpha: {1: <5}, {2: <5}, {3: <5}".for
mat(k,a,OOS_R2(y_valid, y_valid_pred), OOS_R2(y_test, y_test_pred))
)
```

Neural Net ꩜꩜꩜

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.regularizers import l1,l2,l1_l2
from keras.optimizers import RMSprop

for labda in [0.5,0.3,0.2,0.1, 0.01, 0.001]:
'''neural nets with differnt layer setips'''

    layersetups = [[32, 16, 8], [32, 16, 8, 4], [32, 16, 8, 4, 2]]

    for layersetup in layersetups:

        modelname = '{}_{}'.format(str(layersetup), labda)

        modelsetup = []
        regulizer = [tf.keras.regularizers.l1(labda), tf.keras.regu
larizers.l2(labda)]

        for r in regulizer:
            for idx, layernodes in enumerate(layersetup):
                if idx == 0:
                    modelsetup.append(layers.Dense(layernodes, acti
vation='relu', input_shape=(X_train.shape[1],), kernel_regularizer=
tf.keras.regularizers.l1(labda)))
                else:
                    modelsetup.append(layers.Dense(layernodes, acti
vation='relu', kernel_regularizer=tf.keras.regularizers.l1(labda)))
            modelsetup.append(layers.Dense(1, kernel_regularizer=tf
.keras.regularizers.l1(labda)))

            model = keras.Sequential(modelsetup)
    #         model.summary()

            optimizer = keras.optimizers.SGD(learning_rate=labda, )
            model.compile(optimizer=optimizer, loss='mse')

            early_stopping_monitor = EarlyStopping(monitor='val_los
s', mode='min', verbose=1, patience=5)
            model.fit(X_train, y_train, batch_size=1024, validation
_data = (X_valid, y_valid), epochs=100, callbacks=[early_stopping_m
onitor], verbose=0)

            y_pred_valid = model.predict(X_valid)
            R2_valid = OOS_R2(pd.DataFrame(y_valid), y_pred_valid)

            y_pred_test = model.predict(X_valid)
            R2_test = OOS_R2(pd.DataFrame(y_valid), y_pred_test)
            # model.save("models/nn/nn3/nn3_{}.h5".format(R2))
            print('\nMODEL: {0: <30} R2 Valid:{1: <10} R2 Test:{2:
<10} {3}'.format(modelname, R2_valid, R2_test, r.get_config()))
```

In [ ]:

```
print("Num GPUs Available: ", len(tf.config.experimental.list_physi
cal_devices('GPU')))
```

Trees 🌲🌳🌴🎄

GradientBoostingRegressor

In [ ]:
```
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor

## Hyper tuning
for ne in [10,20,50,100,150,200]:
    reg = GradientBoostingRegressor(n_estimators=ne,random_state=7)
    reg.fit(X_train, y_train)

    y_pred_valid = reg.predict(X_valid)
    y_pred_test = reg.predict(X_test)


    print("estimators {}, R2 Valid {} R2 Test ".format(ne,OOS_R2(y_
valid, y_pred_valid)), OOS_R2(y_test, y_pred_test))
```

Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

## Hyper tuning
for ne in [300]:
    for d in [6]:
        reg = RandomForestRegressor(n_estimators = ne, max_depth=d,
random_state=7)
        reg.fit(X_train, y_train)

        y_pred_valid = reg.predict(X_valid)
        y_pred_test = reg.predict(X_test)
        print("estimators {}, depth: {}, R2 Valid {}, R2 Test ".for
mat(ne,d,OOS_R2(y_valid, y_pred_valid)), OOS_R2(y_test, y_pred_test
))
```

EXTENSION

Imports

In [ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import json
import math
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from tensorflow.keras.models import load_model
from scipy import stats
import seaborn as sns

sns.set()
sns.set_style("darkgrid", {"axes.facecolor": ".95"})
```

Load Data

In [ ]:
```
import feather
df = feather.read_dataframe('predictions-6.ftr')
df['year'] = (df['yyyymm']/100).astype(int)
size = feather.read_dataframe('size.ftr')
```

Inspect Data

Table 1

In [ ]:
```
size = feather.read_dataframe('size.ftr')
sizedf = pd.concat([df,size],axis=1)
sizedf = sizedf.loc[sizedf['year'] >= 2007]

def get_models_r2(d):
    '''
    calculates models r2
    '''
    d = d.copy()
    xcols = [c for c in d.columns if c not in ['year','RP', 'lasso_
eq', 'yyyymm', 'mvel1']]

    for c in xcols:
        d[c] = (d[c] - d['RP'])**2 / sum(d['RP']**2)
    # table1 = 1 - table1.sum(axis=0)
    d = pd.DataFrame((1-d.sum(axis=0))).T[xcols]
    return d

def bigsize_getter(d):
    '''
    retrieves top 10% stocks per month in size
    '''
    return d.sort_values(by='mvel1', ascending = False).iloc[:100]

def smallsize_getter(d):
     '''
    retrieves bottom 10% stocks per month in size
    '''
    return d.sort_values(by='mvel1', ascending = True).iloc[:100]
# sizedf.apply(sizegetter, args=(False, 0.1))
bigsizes = sizedf.groupby(by=['yyyymm']).apply(bigsize_getter).rese
t_index(drop=True)
smallsizes = sizedf.groupby(by=['yyyymm']).apply(smallsize_getter).
reset_index(drop=True)

table1 = pd.concat([get_models_r2((df.loc[df['year']>=2007])), get_
models_r2(bigsizes),get_models_r2(smallsizes)])
table1.index = ['All', 'Top 100', 'Bottom 100']
table1 = table1.T
table1.index.name = 'Model'
table1.drop('OLSH', axis = 0, inplace=True)
table1.round(4).to_excel('table1.xlsx')
print(table1.round(3).T.to_latex())
```

In [ ]:
```
## Plot ALL, TOP and BOTTOM R2 OOS
table1.plot(kind='bar', figsize=(20,10), width = 0.5)
plt.ylabel(r"$R^2_{OOS}$")
plt.xticks(rotation='horizontal')
```

In [ ]:
```
print(get_models_r2(df.loc[(df['year']>=2007)]).round(4).to_latex()
)
```

In [ ]:
```
def OOS_R2(y, ypred):
    '''
    adjusted R2 following gu et al. 2019
    '''
    y = np.array(y)
    ypred = np.array(ypred)
    R2 = 1 - (np.sum((y-ypred)**2) / np.sum(y**2))
    return round(R2,ndigits=5)

def df_R2(X):
    '''
    dataframe edition R2 following gu et al. 2019
    '''
    y = X['RP']

    models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5']

    for m in models:
        X[m] = (X[m] - y)**2
    X = X.sum()
    for m in models:#['OLS3H', 'OLS', 'OLSH', 'PCR', 'PCRH', 'PLS']
:
        X[m] = 1 - X[m]/sum(y**2)
#     X.drop(['year', 'yyyymm', 'RP'], inplace=True)
    return X[['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5']]



# cols = [c for c in preds.columns if c not in ['class', 't', 'year
']]
```

In [ ]:
```
df.groupby(by=['year']).apply(df_R2).plot(figsize=(16,9), ylim=[-0.
2,0.45])
plt.axvline(linewidth=2,c='black', x=2007)
plt.axvline(linewidth=2,c='black', x=2001)
plt.axvspan(2007.5, 2009, alpha=0.2, color='grey')
plt.axvspan(1986.75, 1987.25, alpha=0.2, color='grey')
plt.axvspan(2000, 2002, alpha=0.2, color='grey')
plt.text(x=1992.5,y=0.35,s='train', size = 15)
plt.text(x=2003.5,y=0.35,s='valid', size = 15)
plt.text(x=2011.5,y=0.35,s='test', size=15)
plt.ylabel(r"$R^2_{OOS}$")
plt.savefig('r2_oos.png')
```

In [ ]:
```
df.drop(columns=['yyyymm']).groupby(by=['year']).mean().plot(figsiz
e=(16,9), ylim=[-0.1,0.1])
plt.axvline(linewidth=2,c='black', x=2007)
plt.axvline(linewidth=2,c='black', x=2001)
plt.axvspan(2007.5, 2009, alpha=0.2, color='grey')
plt.axvspan(1986.75, 1987.25, alpha=0.2, color='grey')
plt.axvspan(2000, 2002, alpha=0.2, color='grey')
plt.text(x=1992.5,y=0.08,s='train', size = 15)
plt.text(x=2003.5,y=0.08,s='valid', size = 15)
plt.text(x=2011.5,y=0.08,s='test', size=15)
plt.ylabel(r"Forecasted risk premium $E(r)$")
plt.savefig('Er_oos.png')
```

split data

In [ ]:
```
import statsmodels.api as sm
from sklearn.linear_model import Lasso

xcols = [c for c in df.columns if c not in ['year','RP', 'lasso_eq'
, 'yyyymm']]

y = df['RP']
X = df[xcols]

testidx = df.loc[df['year'] >= 2007].index[0]
valididx = df.loc[df['year'] >= 2001].index[0]

X_train = X.iloc[:valididx]
X_valid = X.iloc[valididx:testidx]
X_test = X.iloc[testidx:]

y_train = y.iloc[:valididx]
y_valid = y.iloc[valididx:testidx]
y_test = y.iloc[testidx:]
```

Meta Model

Simple average

In [ ]:
```
y_pred_simpleavg = X_test.sum(axis=1)/len(xcols)
OOS_R2(y_test, y_pred_simpleavg)

y_pred_simpleavg = X.sum(axis=1)/len(xcols)
```

Rank selection

In [ ]:
```
validr2 = get_models_r2(df.loc[(df['year']<2007) & (df['year'] >= 2
001)].copy())
print(validr2)
array = validr2.iloc[0]
order = array.argsort()
ranks = order.argsort() + 1
print(pd.DataFrame(12-ranks).T.to_latex())
rank_distribution = ranks/sum(ranks)
print(pd.DataFrame(rank_distribution).round(3).T.to_latex())
y_pred_rank = X_test.dot(rank_distribution)
OOS_R2(y_test, y_pred_rank)

y_pred_rank = X.dot(rank_distribution)
```

Meta OLS

In [ ]:
```
import statsmodels.api as sm

meta_ols = sm.OLS(y_valid, X_valid).fit()
y_pred_meta_ols = meta_ols.predict(X_test)

print(pd.DataFrame(meta_ols.params).T)
print(OOS_R2(y_test, y_pred_meta_ols))
```

LASSO and peLASSO 🎗

In [ ]:
```
import statsmodels.api as sm
from sklearn.linear_model import Lasso

## LASSO
def doLASSO(a):
    lassoreg = Lasso(alpha=a, max_iter=1e5, fit_intercept=False).fi
t(X_valid, y_valid)
    y_pred_train_lasso = lassoreg.predict(X_train)
    r2_train = OOS_R2(y_train, y_pred_train_lasso)
    return r2_train, lassoreg

# Hyper tuning
hyper_a = None
best_r2 = -999

for i in range(1,10):
    a = i/1e5
    lasso_r2, lasso_coef = doLASSO(a)
    if lasso_r2 > best_r2:
        best_r2 = lasso_r2
        hyper_a = a

print(f"best pel a: {hyper_a}")
a = hyper_a
_, lassoreg = doLASSO(a)
```

```
y_pred_lasso = lassoreg.predict(X_test)


models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT', 'RF'
, 'NN3', 'NN4', 'NN5']
print(OOS_R2(y_test, y_pred_lasso))
print(pd.DataFrame(lassoreg.coef_, index = models).round(2).T.to_la
tex())

y_pred_lasso = lassoreg.predict(X)


### peLASSO
selection = [1 if x > 0 else 0 for x in lassoreg.coef_]
X_valid_pel = X_valid.iloc[:,np.where(selection)[0]]
# X_test_pel = X_test.iloc[:,np.where(selection)[0]]
y_pel = df['RP'] - X.iloc[:,np.where(selection)[0]].mean(axis=1)
y_valid_pel = y_pel.iloc[valididx:testidx]
# y_test_pel = y_pel.iloc[testidx:]

# Step 2 LASSO
pelassoreg = Lasso(alpha=hyper_a, max_iter=1e5, fit_intercept=False
).fit(X_valid_pel, y_valid_pel)
pelasso_coef = pelassoreg.coef_ + 1/len(pelassoreg.coef_)
pelasso_coef_idx = 0

for idx, i in enumerate(selection):
    if i == 1:
        selection[idx] = pelasso_coef[pelasso_coef_idx]
        pelasso_coef_idx += 1

pelasso_coef = selection

print(OOS_R2(y_test, X_test.dot(pelasso_coef)))
print(pd.DataFrame(pelasso_coef, index = models).round(2).T.to_late
x())


y_pred_pel = X.dot(pelasso_coef)
```

Neural Net ensemble / selection

In [ ]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import Sequential
```

```
from keras.layers import Dense, Activation
from keras.regularizers import l1,l2,l1_l2
from functools import reduce


macros = pd.read_csv('fred.csv')
macrocols = macros.columns.drop('yyyymm')
print("macros loaded")
macro_yyyymm = macros['yyyymm']
scaler = preprocessing.StandardScaler()
macros = pd.DataFrame(scaler.fit_transform(macros[macrocols]), colu
mns = macrocols)
macros['yyyymm'] = macro_yyyymm

macros.set_index('yyyymm', inplace=True)

## Split data in train, valid, test

macros['bestmodel'] = np.array(df.groupby(by=['yyyymm']).apply(df_R
2).apply(lambda x: np.argmax(np.array(x)), axis = 1))
print("Data labeled")
y_nn = pd.DataFrame(np_utils.to_categorical(macros['bestmodel']))
y_nn.index = macros.index

X_nn = macros[macrocols]
X_nn = scaler.fit_transform(X_nn)
X_nn = pd.DataFrame(X_nn, columns=macrocols)
X_nn.index = macros.index

X_nn_train = X_nn.loc[:200012]
X_nn_valid = X_nn.loc[200101:200612]
X_nn_test = X_nn.loc[200701:]

y_nn_train = np.array(y_nn.loc[:200012])
y_nn_valid = np.array(y_nn.loc[200101:200612])
y_nn_test = np.array(y_nn.loc[200701:])




models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT', 'RF'
, 'NN3', 'NN4', 'NN5']

def make_nn_model_selection_ensemble(labda, ensembles):
    '''
    creates neural network with ensembles
    '''

    nnweights_ensembles = []

    for i in range(ensembles):
        print('ensemble',i)
        model = Sequential()
        model.add(Dense(32, activation='relu', input_dim=len(macroc
ols), kernel_regularizer=tf.keras.regularizers.l1(labda)))
        model.add(Dense(16, activation='relu', kernel_regularizer=t
```

```
f.keras.regularizers.l1(labda)))
        model.add(Dense(8, activation='relu', kernel_regularizer=tf
.keras.regularizers.l1(labda)))
        model.add(Dense(11, activation='softmax', kernel_regularize
r=tf.keras.regularizers.l1(labda)))

        # Compile the model
        model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
        # build the model
        model.fit(X_nn_valid, y_nn_valid, epochs=100, validation_da
ta = (X_nn_train, y_nn_train), verbose = 0)

        nnweights = pd.DataFrame(model.predict(X_nn))
        nnweights.columns = models
        nnweights['yyyymm'] = macros.index

        nnweights = pd.merge(df[['yyyymm']], nnweights, how='left',
on='yyyymm').drop('yyyymm', axis=1)
        nnweights_ensembles.append(nnweights)

    return nnweights_ensembles


nn_ensembles = make_nn_model_selection_ensemble(0.01, 10)
nn_ensemble = reduce(lambda a, b: a.add(b, fill_value=0), nn_ensemb
les)/len(nn_ensembles)

nn_ensemble.index = df.yyyymm/100 + df.yyyymm%100/12
nn_ensemble.plot(kind='area', figsize=(20,10))
plt.axvline(linewidth=2,c='black', x=2007)
plt.axvline(linewidth=2,c='black', x=2001)
plt.axvspan(2007.5, 2009, alpha=0.3, color='grey')
plt.axvspan(1986.75, 1987.25, alpha=0.3, color='grey')
plt.axvspan(2000, 2002, alpha=0.3, color='grey')
plt.text(x=1992.5,y=0.8,s='valid', size = 15)
plt.text(x=2003.5,y=0.8,s='train', size = 15)
plt.text(x=2011.5,y=0.8,s='test', size=15)
plt.savefig('moe_weights.png')

nn_ensemble.reset_index(inplace=True, drop=True)

testa = 57000
traina = 72000

y_pred_moe = (df[models] * nn_ensemble).sum(axis=1)


nn_ensemble.columns=[0,1,2,3,4,5,6,7,8,9,10]
nn_select = pd.DataFrame(np_utils.to_categorical(nn_ensemble.idxmax
(axis=1)))
# fill model 10 and 11 with zeros since they are not included in ma
trix because they are never selected
for i in range(max(nn_select.columns)+1, 11):
    nn_select[i] = 0
nn_select.columns = models
```

```
y_pred_expert = (df[models] * nn_select).sum(axis=1)
```

Diebold Marino

In [ ]:
```
def dm_test(actual_lst, pred1_lst, pred2_lst, h = 1, crit="MSE", po
wer = 2):
    # Routine for checking errors
    def error_check():
        rt = 0
        msg = ""
        # Check if h is an integer
        if (not isinstance(h, int)):
            rt = -1
            msg = "The type of the number of steps ahead (h) is not
an integer."
            return (rt,msg)
        # Check the range of h
        if (h < 1):
            rt = -1
            msg = "The number of steps ahead (h) is not large enoug
h."
            return (rt,msg)
        len_act = len(actual_lst)
        len_p1  = len(pred1_lst)
        len_p2  = len(pred2_lst)
        # Check if lengths of actual values and predicted values ar
e equal
        if (len_act != len_p1 or len_p1 != len_p2 or len_act != len
_p2):
            rt = -1
            msg = "Lengths of actual_lst, pred1_lst and pred2_lst d
o not match."
            return (rt,msg)
        # Check range of h
        if (h >= len_act):
            rt = -1
            msg = "The number of steps ahead is too large."
            return (rt,msg)
        # Check if criterion supported
        if (crit != "MSE" and crit != "MAPE" and crit != "MAD" and
crit != "poly"):
            rt = -1
            msg = "The criterion is not supported."
            return (rt,msg)
#         # Check if every value of the input lists are numerical v
alues
#         from re import compile as re_compile
#         comp = re_compile("^\d+?\.\d+?$")
#         def compiled_regex(s):
#             """ Returns True is string is a number. """
#             if comp.match(s) is None:
#                 return s.isdigit()
#             return True
```

```
#         for actual, pred1, pred2 in zip(actual_lst, pred1_lst, pr
ed2_lst):
#             is_actual_ok = compiled_regex(str(abs(actual)))
#             is_pred1_ok = compiled_regex(str(abs(pred1)))
#             is_pred2_ok = compiled_regex(str(abs(pred2)))
#             if (not (is_actual_ok and is_pred1_ok and is_pred2_ok
)):
#                 msg = "An element in the actual_lst, pred1_lst or
pred2_lst is not numeric."
#                 rt = -1
#                 return (rt,msg)
        return (rt,msg)

    # Error check
    error_code = error_check()
    # Raise error if cannot pass error check
    if (error_code[0] == -1):
        raise SyntaxError(error_code[1])
        return
    # Import libraries
    from scipy.stats import t
    import collections
    import pandas as pd
    import numpy as np

    # Initialise lists
    e1_lst = []
    e2_lst = []
    d_lst  = []

    # convert every value of the lists into real values
    actual_lst = pd.Series(actual_lst).apply(lambda x: float(x)).to
list()
    pred1_lst = pd.Series(pred1_lst).apply(lambda x: float(x)).toli
st()
    pred2_lst = pd.Series(pred2_lst).apply(lambda x: float(x)).toli
st()

    # Length of lists (as real numbers)
    T = float(len(actual_lst))

    # construct d according to crit
    if (crit == "MSE"):
        for actual,p1,p2 in zip(actual_lst,pred1_lst,pred2_lst):
            e1_lst.append((actual - p1)**2)
            e2_lst.append((actual - p2)**2)
        for e1, e2 in zip(e1_lst, e2_lst):
            d_lst.append(e1 - e2)
    elif (crit == "MAD"):
        for actual,p1,p2 in zip(actual_lst,pred1_lst,pred2_lst):
            e1_lst.append(abs(actual - p1))
            e2_lst.append(abs(actual - p2))
        for e1, e2 in zip(e1_lst, e2_lst):
            d_lst.append(e1 - e2)
    elif (crit == "MAPE"):
        for actual,p1,p2 in zip(actual_lst,pred1_lst,pred2_lst):
            e1_lst.append(abs((actual - p1)/actual))
```

```
            e2_lst.append(abs((actual - p2)/actual))
        for e1, e2 in zip(e1_lst, e2_lst):
            d_lst.append(e1 - e2)
    elif (crit == "poly"):
        for actual,p1,p2 in zip(actual_lst,pred1_lst,pred2_lst):
            e1_lst.append(((actual - p1))**(power))
            e2_lst.append(((actual - p2))**(power))
        for e1, e2 in zip(e1_lst, e2_lst):
            d_lst.append(e1 - e2)

    # Mean of d
    mean_d = pd.Series(d_lst).mean()

    # Find autocovariance and construct DM test statistics
    def autocovariance(Xi, N, k, Xs):
        autoCov = 0
        T = float(N)
        for i in np.arange(0, N-k):
              autoCov += ((Xi[i+k])-Xs)*(Xi[i]-Xs)
        return (1/(T))*autoCov
    gamma = []
    for lag in range(0,h):
        gamma.append(autocovariance(d_lst,len(d_lst),lag,mean_d)) #
0, 1, 2
    V_d = (gamma[0] + 2*sum(gamma[1:]))/T
    DM_stat=V_d**(-0.5)*mean_d
    harvey_adj=((T+1-2*h+h*(h-1)/T)/T)**(0.5)
    DM_stat = harvey_adj*DM_stat
    # Find p-value
    p_value = 2*t.cdf(-abs(DM_stat), df = T - 1)

    # Construct named tuple for return
    dm_return = collections.namedtuple('dm_return', 'DM p_value')
    rt = dm_return(DM = DM_stat, p_value = p_value)

    DM_stat = np.round(DM_stat,2)

    return DM_stat if p_value > 0.05 else str(DM_stat)+'*' #'\textb
f{'+str(DM_stat)+'}' if p_value < 0.05 else DM_stat
```

In [ ]:
```
df['meta_average'] = y_pred_simpleavg
df['meta_lasso'] = y_pred_lasso
df['meta_pel'] = y_pred_pel
df['meta_ranked'] = y_pred_rank
df['meta_moe'] = y_pred_moe
df['meta_expert'] = y_pred_expert

# df.to_feather('finaldraft_df.ftr')
```

In [ ]:
```
df = feather.read_dataframe('finaldraft_df.ftr')
```

Diebold and Mariano pairwise test

In [ ]:
```
db_df = df.loc[df.year >= 2007]
models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT', 'RF'
, 'NN3', 'NN4', 'NN5', 'meta_average', 'meta_lasso', 'meta_pel', 'm
eta_ranked', 'meta_moe', 'meta_expert']

q = np.empty([len(models)-1, len(models) - 1], dtype="<U10")
for i in range(len(models)-1):
    for j in range(i, len(models)-1):
        modelA = models[i]
        modelB = models[j+1]
        dmstat = dm_test(np.array(db_df['RP']), np.array(db_df[mode
lB]), np.array(db_df[modelA]), h=1)
        q[i,j] = dmstat

dm_table = pd.DataFrame(q, index=models[:-1], columns=models[1:])
print(dm_table.to_latex())
dm_table
```

Portfolio returns

In [ ]:
```
def topminbottom_return(X):

    X = X.copy()
    num_elem = int(len(X)/10)


    models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5', 'meta_average', 'meta_lasso', 'meta_pel'
, 'meta_ranked', 'meta_moe', 'meta_expert']

    ret = X[models].iloc[0].copy()

    for m in models:

        X.sort_values(by=m, ascending=False, inplace=True)
        returns = np.mean(X['RP'][:num_elem]) - np.mean(X['RP'][-nu
m_elem:])
        ret[m] = returns

    return ret

def get_decile10_return(X):
    X = X.copy()
    num_elem = int(len(X)/10)
    models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5', 'meta_average', 'meta_lasso', 'meta_pel'
, 'meta_ranked', 'meta_moe', 'meta_expert']
    res = []
    for m in models:
        X.sort_values(by=m, ascending=False, inplace=True)
        res.append(list(X['RP'][:num_elem]))
    res = pd.DataFrame(res).T
    res.columns = models
    return res.mean()
```

```
def get_decile1_return(X):
    X = X.copy()
    num_elem = int(len(X)/10)
    models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5', 'meta_average', 'meta_lasso', 'meta_pel'
, 'meta_ranked', 'meta_moe', 'meta_expert']
    res = []
    for m in models:
        X.sort_values(by=m, ascending=True, inplace=True)
        res.append(list(X['RP'][:num_elem]))
    res = pd.DataFrame(res).T
    res.columns = models
    return res.mean()


def get_decile10_1_return(X):
    X = X.copy()
    num_elem = int(len(X)/10)
    models = ['OLS3', 'OLSH', 'PCR', 'PLS', 'ENet', 'GLM', 'GBRT',
'RF', 'NN3', 'NN4', 'NN5', 'meta_average', 'meta_lasso', 'meta_pel'
, 'meta_ranked', 'meta_moe', 'meta_expert']
    res = []
    for m in models:
        X.sort_values(by=m, ascending=False, inplace=True)
        res.append(list(X['RP'][:num_elem])+list(-X['RP'][-num_elem
:]))
    res = pd.DataFrame(res).T
    res.columns = models
    return res.mean()
```

In [ ]:
```
topbottom = df.loc[df.year >= 2007].drop(columns='year').groupby(by
=['yyyymm']).apply(topminbottom_return)#.plot(figsize=(16,9))
topbottom.index = topbottom.index/100 + topbottom.index%100/12
topbottom = np.log(1+topbottom)
topbottom.cumsum().plot(figsize=(20,10))

plt.axvspan(2007.5, 2009, alpha=0.2, color='grey')
plt.ylabel("Position")
plt.xlabel("year")
plt.savefig('fig_topbottom_pos.png')

print(pd.DataFrame(topbottom.sum()).T.round(2).to_latex())
```

In [ ]:

```
def get_annual_sharpe(x):
    '''
    calculates annual sharpe
    '''
    return ((np.mean(x)*12) / (np.sqrt(12)*np.std(x)))

df_decile1 = df.loc[df.year >= 2007].drop(columns='year').groupby(b
y=['yyyymm']).apply(get_decile1_return)
df_decile10 = df.loc[df.year >= 2007].drop(columns='year').groupby(
by=['yyyymm']).apply(get_decile10_return)
df_decile10_1 = df.loc[df.year >= 2007].drop(columns='year').groupb
y(by=['yyyymm']).apply(get_decile10_1_return)

sharpe_df = pd.DataFrame([100*df_decile1.mean(),100*df_decile1.std(
),df_decile1.apply(get_annual_sharpe),
                          100*df_decile10.mean(),100*df_decile10.st
d(),df_decile10.apply(get_annual_sharpe),
                          100*df_decile10_1.mean(),100*df_decile10_
1.std(),df_decile10_1.apply(get_annual_sharpe)])

sharpe_df.insert(0, 'avg_sd_sr', 3*['AVG','SD','SR'])
sharpe_df.index = ['Decile 1 (L)','','','Decile 10 (H)','','','H-L'
,'','']
print(sharpe_df.round(2).to_latex())
sharpe_df.round(2)
```

In [ ]:

```
bootstrap_df = df_decile10_1.copy()
models = list(bootstrap_df.columns)


def do_sharpe_bootstrap(portfolios):
    '''
    Calculate differences between portfolios and examine statistica
l difference by bootstrap inference
    '''
    dSharpe = np.round(get_annual_sharpe(portfolios.iloc[:,0]) - ge
t_annual_sharpe(portfolios.iloc[:,1]),2)

    z_alpha = 1.645

    bootstrap_dSharpes = []
    BOOTSTRAPS = 10000
    for i in range(BOOTSTRAPS):
        bootstrap_sample_Sharpes = portfolios.sample(n=len(portfoli
os), replace=True).apply(get_annual_sharpe)
        bootstrap_dSharpes.append(bootstrap_sample_Sharpes[0] - boo
tstrap_sample_Sharpes[1])

    bootstrap_dSharpe_std = np.std(bootstrap_dSharpes)
    sign_difference = not(np.mean(bootstrap_dSharpes) - z_alpha * b
ootstrap_dSharpe_std < 0 < np.mean(bootstrap_dSharpes) + z_alpha *
bootstrap_dSharpe_std)
    print(sign_difference, np.mean(bootstrap_dSharpes) - z_alpha *
bootstrap_dSharpe_std, np.mean(bootstrap_dSharpes) + z_alpha * boot
strap_dSharpe_std)

    return f'{dSharpe}*' if sign_difference else dSharpe


res = np.empty([len(models)-1, len(models) - 1], dtype="<U10")
for i in range(len(models)-1):
    for j in range(i, len(models)-1):
        modelA = models[i]
        modelB = models[j+1]
        print(modelA, modelB)
        res[i,j] = do_sharpe_bootstrap(bootstrap_df[[modelA,modelB]
])

sharpe_sign_df = pd.DataFrame(res, index=models[:-1], columns=model
s[1:])
print(sharpe_sign_df.to_latex())
sharpe_sign_df
```

References

Aiolfi, M., & Timmermann, A. (2006). Persistence in forecasting performance and conditional combination strategies. *Journal of Econometrics*, *135*(1-2), 31–53.

Bates, J. M., & Granger, C. W. (1969). The combination of forecasts. *Journal of the Operational Research Society*, *20*(4), 451–468.

Butaru, F., Chen, Q., Clark, B., Das, S., Lo, A. W., & Siddique, A. (2016). Risk and risk management in the credit card industry. *Journal of Banking & Finance*, *72*, 218–239.

Clemen, R. T. (1989). Combining forecasts: A review and annotated bibliography. *International journal of forecasting*, *5*(4), 559–583.

Clemen, R. T., & Winkler, R. L. (1986). Combining economic forecasts. *Journal of Business & Economic Statistics*, *4*(1), 39–46.

Diebold, F. X., & Lopez, J. A. (1996). 8 forecast evaluation and combination. *Handbook of statistics*, *14*, 241–268.

Diebold, F. X., & Mariano, R. S. (2002). Comparing predictive accuracy. *Journal of Business & economic statistics*, *20*(1), 134–144.

Diebold, F. X., & Pauly, P. (1987). Structural change and the combination of forecasts. *Journal of Forecasting*, *6*(1), 21–40.

Diebold, F. X., & Pauly, P. (1990). The use of prior information in forecast combination. *International Journal of Forecasting*, *6*(4), 503–508.

Diebold, F. X., & Shin, M. (2017). Beating the simple average: Egalitarian lasso for combining economic forecasts.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems* (pp. 1–15).

Dunis, C. L., Laws, J., & Chauvin, S. (2001). The use of market data and model combination to improve forecast accuracy. *Developments in forecast combination and portfolio choice*, 45–80.

Efron, B. (1982). *The jackknife, the bootstrap and other resampling plans*. SIAM.

Elliott, G., & Timmermann, A. (2008). Economic forecasting. *Journal of Economic Literature*, *46*(1), 3–56.

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of*.

Feng, G., Giglio, S., & Xiu, D. (2017). Taming the factor zoo. *Chicago Booth research paper*(17-04).

Figlewski, S., & Urich, T. (1983). Optimal aggregation of money supply forecasts: Accuracy, profitability and market efficiency. *The Journal of Finance*, *38*(3), 695–710.

Freyberger, J., Neuhierl, A., & Weber, M. (2020). Dissecting characteristics nonparametrically. *The Review of Financial Studies*, *33*(5), 2326–2377.

Garcia, D. (2013). Sentiment during recessions. *The Journal of Finance*, *68*(3), 1267–1300.

Granger, C. W. (1989). Invited review combining forecasts—twenty years later. *Journal of Forecasting*, *8*(3), 167–173.

Green, J., Hand, J. R., & Zhang, X. F. (2013). The supraview of return predictive signals. *Review of Accounting Studies*, *18*(3), 692–730.

Gu, S., Kelly, B., & Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, *33*(5), 2223–2273.

Gupta, S., & Wilton, P. C. (1987). Combination of forecasts: an extension. *Management Science*, *33*(3), 356–372.

Han, Y., He, A., Rapach, D., & Zhou, G. (2018). What firm characteristics drive us stock returns? *Available at SSRN 3185335*.

Heaton, J., Polson, N. G., & Witte, J. (2016). Deep portfolio theory. *arXiv preprint arXiv:1605.07230*.

Hendry, D. F., & Clements, M. P. (2004). Pooling of forecasts. *The Econometrics Journal*, *7*(1), 1–31.

Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics* (pp. 492–518). Springer.

Hutchinson, J. M., Lo, A. W., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, *49*(3), 851–889.

Jobson, J. D., & Korkie, B. M. (1981). Performance hypothesis testing with the sharpe and treynor measures. *Journal of Finance*, 889–908.

Jolliffe, I. T. (1986). Principal components in regression analysis. In *Principal component analysis* (pp. 129–155). Springer.

Kelly, B. T., Pruitt, S., & Su, Y. (2019). Characteristics are covariances: A unified model of risk and return. *Journal of Financial Economics*, *134*(3), 501–524.

Khandani, A. E., Kim, A. J., & Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, *34*(11), 2767–2787.

Kozak, S., Nagel, S., & Santosh, S. (2020). Shrinking the cross-section. *Journal of Financial*

Economics, *135*(2), 271–292.

Ledoit, O., & Wolf, M. (2008). Robust performance hypothesis testing with the sharpe ratio. *Journal of Empirical Finance*, *15*(5), 850–859.

Makridakis, S. (1989). Why combining works? *International Journal of Forecasting*, *5*(4), 601–603.

Masters, T. (1993). *Practical neural network recipes in c++*. Morgan Kaufmann.

McCracken, M. W., & Ng, S. (2016). Fred-md: A monthly database for macroeconomic research. *Journal of Business & Economic Statistics*, *34*(4), 574–589.

Memmel, C. (2003). Performance hypothesis testing with the sharpe ratio. *Finance Letters*, *1*(1).

Moritz, B., & Zimmermann, T. (2016). Tree-based conditional portfolio sorts: The relation between past and future stock returns. *Available at SSRN 2740751*.

Newbold, P., & Harvey, D. I. (2002). Forecast combination and encompassing. *A companion to economic forecasting*, *1*, 620.

Pesaran, M. H., & Timmermann, A. (2007). Selection of estimation window in the presence of breaks. *Journal of Econometrics*, *137*(1), 134–161.

Sharpe, W. F. (1994). The sharpe ratio. *Journal of portfolio management*, *21*(1), 49–58.

Sigletos, G., Paliouras, G., Spyropoulos, C. D., & Hatzopoulos, M. (2005). Combining information extraction systems using voting and stacked generalization. *Journal of Machine Learning Research*, *6*(Nov), 1751–1782.

Sirignano, J., Sadhwani, A., & Giesecke, K. (2016). Deep learning for mortgage risk. *arXiv preprint arXiv:1607.02470*.

Timmermann, A. (2006). Chapter 4 forecast combinations. , *1*, 135 - 196. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1574070605010049` doi: https://doi.org/10.1016/S1574-0706(05)01004-9

Welch, I., & Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, *21*(4), 1455–1508.

Winkler, R. L., & Makridakis, S. (1983). The combination of forecasts. *Journal of the Royal Statistical Society: Series A (General)*, *146*(2), 150–157.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241–259.

Yao, J., Li, Y., & Tan, C. L. (2000). Option price forecasting using neural networks. *Omega*, *28*(4), 455–466.

Zhu, Y., Xie, C., Wang, G.-J., & Yan, X.-G. (2017). Comparison of individual, ensemble and integrated ensemble machine learning methods to predict china's sme credit risk in supply

chain finance. *Neural Computing and Applications*, *28*(1), 41–50.