Equity Fund Profitability and Sustainability Modelling

A Multiple Response Regression Analysis

Raul Unnithan

Supervisor: Ric Crossman

December 13, 2024

Abstract

This dissertation applies Multiple Response Regression techniques to model equity fund profitability and sustainability. This report's analysis will compare different regression models on a Kaggle equity fund dataset to provide insights into and model the variables driving equity fund profitability and sustainability performance.

Contents

Chapter 1 Introduction

This report delves into the theory of different regression models and compares their performance on a Kaggle dataset. This dataset consists of equity funds, and the variables being modelled are return on equity and sustainability score against other fund features.

Chapter 2: Exploratory Data Analysis - the master dataset consists of a mixture of equity and bond funds and a lot of missing data. This chapter involves simplifying the dataset, defining the remaining predictors and then understanding underlying relationships in the response variables and imputing missing values.

Chapter 3 Multiple Response Regression - this gives a basic outline of multiple response regression and its underlying theories. Then, some simple stepwise selection methods are applied and their fit on the equity fund data is analysed.

Chapter 4: Shrinkage Methods - now, there is a progression to shrinkage methods and how they are adapted to the multivariate case. They are also applied and evaluated on the dataset, and the best among them is decided.

Chapter 5: Random Forest Regression - the theory behind this method is introduced, and how now non-linear regression methods can apply to the dataset. Its adjustment to the multivariate case will be explained, and how well it models the data will be tested.

Chapter 6: XG Boost - this is another non-linear approach which will be looked at for this dataset. The goal of this is to see if or how it improves on random forest.

Chapter 7: Neural Networks - this is the final regression model that will be looked at. It brings a whole new approach to modelling the dataset and it should model best because of its adaptability.

Chapter 8: Conclusion - this will be a final comparison of all the regression models in the dataset, and it will ultimately decide which one is the best fit. Also, the conclusion will explain what further analysis could be done to improve these regression models.

Chapter 2 Data and Methodology

2.1 Initial Data Collection and Preprocessing

The dataset was obtained from Kaggle and simplified for this analysis. The research focuses on analysing equity funds, and the master dataset contains a mixture of equity and bond funds.

The first step was to filter out the bond funds and take solely the equities. This filtering was done simply using Excel's filter features. A text filter selects the relevant rows within the `category` column.

Then, there was an issue with the columns; there were 130 dependent variables for this analysis. It is essential to use relevant variables that do not cause multicollinearity. Many variables were for bonds only, so these were removed. Some variables, such as a fund's `isin` number, were irrelevant for analysis, further reducing the number of dependent variables.

Through these two elimination processes, 29 variables were left, including `roe` and `sustainability_score`, the dependent variables. Another critical factor for the chosen independent variables was to be able to look into how they would affect the dependent variables, so the aim was to have 10-20 variables in the model. Also, a lot of the remaining variables served the same purpose, which would lead to multicollinearity; for example, `management_fees` are a significant component of `ongoing_cost`, so the `management_fees` column was removed.

The final step in preprocessing is to remove inverse equity funds. Although these are equity funds, they perform the opposite of their underlying benchmark, leaving 14 dependent variables for the study.

2.2 Variable Explanation

Defining the 14 dependent variables before the analysis will give an insight into how they could relate and will set up the rest of the research. `category` gives the type of equity on which the data is collected. A `rating` is given by an analyst who is assigned to the equity, and this is a numeric value from 1 to 5. A 1 means the analyst thinks the security is performing well, so they recommend buying it; a five means the analyst thinks the security is performing poorly, so they recommend selling it. A 3 means they want to hold the security. A `risk_rating` gives the market volatility of the equity.[1] `category`, `rating` and `risk_rating` are categorical; the rest of the variables are continuous.

`equity_style_score` is the difference between 2 investment strategies: growth and value stocks.[2] Growth stocks are expected to bring a lot of capital due to strong growth in the underlying company.[3] Value stocks are either underrated or ignored by the market; they could gain value eventually.[3] Hence, a lower `equity_style_score` means the stock is more of a value stock, and a higher `equity_style_score` indicates more of a growth stock. `equity_size_score` measures the performance of said equity relative to the company's value.[4] `price_prospective_earnings` gives the ratio between a company's share price and earnings per share. `price_cash_flow_ratio` is a multiple that compares a company's market value to the amount of capital it generates during a select period.[4]

A dividend is the money companies may pay you if you own their shares, and the dividend yield is the dividend paid as a percentage of the share price. The `dividend_yield_factor` is the dividend yield, except it is used to compare the different equities, considering variations in dividend yield performance. `historical_earnings_growth` measures how a stock's earnings per share has grown in the last 5 years. `sales_growth` is the increase in sales of a company's products/ services over time.

`asset_cash` is the proportion of a fund's assets held in cash. Values greater than one can occur due to borrowing, and there can be negative values; this means there could be an overdraft or a leveraged position.[5] `holdings_n_stock` is the number of different stock holdings each equity fund holds. A stock holding is the number of other companies in which the equity fund has invested by holding their stocks. `ongoing_cost` is the cost associated with the daily running of a business.[6] `fund_size` is the sum of all the assets in the fund.

2.3 Exploratory Data Analysis

Now that the dataset has been cleaned and the variables specified, the next step is to perform data analysis in R, find underlying relationships between the variables, and understand how the dependent variables are distributed. To fully represent the dataset, it is best to use actual values rather than simulate any missing values. So, this initial analysis will be done with only rows with no missing values.

2.3.1 Underlying Distributions

QQ plots can be used to find out if a variable is normally distributed. The ROE and sustainability scores' QQ plots are shown below:

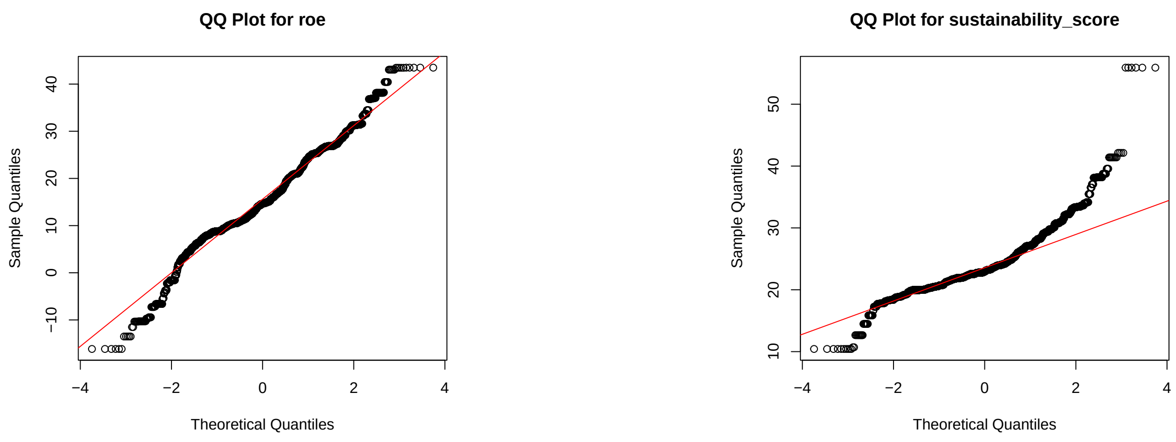


Figure 2.1: QQ Plots for ROE and Sustainability Scores

ROE's underlying distribution is usually distributed because most data points can be roughly approximated as a straight line. Sustainability scores are not because, although many of their data points can be approximated as a straight line, some points deviate significantly from the line at both tails, indicating heavy tail behaviour.

It is essential to understand the distribution for sustainability scores when doing further analysis, so the `fitdistrplus` package is used. This package compares various distributions to the sustainability scores using Bootstrap, the square of skewness and kurtosis. Skewness measures a distribution's asymmetry, and kurtosis measures the combined weight of a distribution's tails relative to its mean.[7] These two factors are compared on a Cullen and Frey graph, which helps to interpret the sustainability score's distribution.

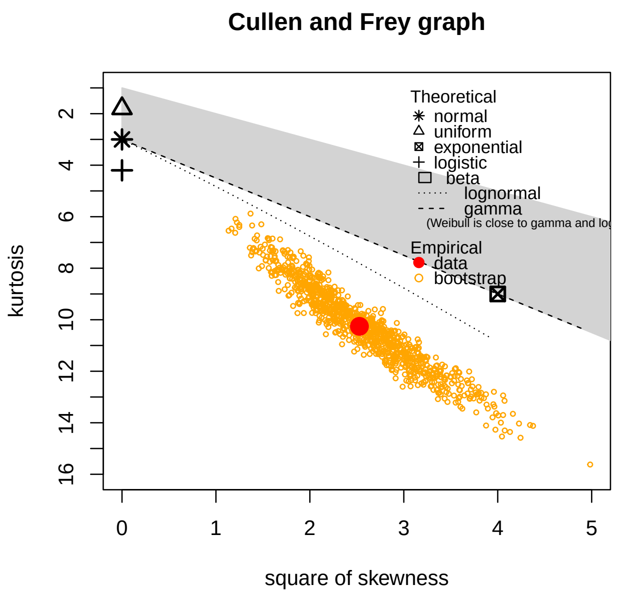


Figure 2.2: Cullen and Frey Graph for Sustainability Scores

Given the data points relative to the distributions on the graph, it is clear that there is no apparent underlying distribution for equity fund sustainability scores for this dataset.

2.3.2 Correlation and Multi-Collinearity

It is also essential to examine the correlation within the dependent variables to determine which additional variables should be removed. High correlation implies multicollinearity, so it is necessary to check this before proceeding.

After creating the correlation matrix for the dependent variables, there were two high correlation values. One was `0.8099`, which was the correlation between `price_prospective_earnings` and `price_cash_flow_ratio`. Based on the initial definitions of these variables, `price_cash_flow_ratio` is a more relevant factor in profitability and sustainability scores; it is kept. The other high correlation was `-0.8029` between `dividend_yield_factor` and `equity_style_score`. Again, based on the definitions of these variables, `dividend_yield_factor` is a more relevant factor in profitability and sustainability scores; it is kept. This results in 12 independent variables.

2.3.3 Filling Missing Values

Although high correlation implies multicollinearity, the opposite is not necessarily true. Multicollinearity can be tested on variables through the virtual inflation factor (VIF). This dataset contains a mixture of categorical and continuous variables, so it is essential to consider both variable types when analysing it.

The dataset should be complete before carrying out VIF analysis. It is essential to look at how the NA values are spread and how often they occur in each row. So, with some simple manipulation in R, it was found that 90% of the rows had 2 NAs or less, and the dataset is large, so only these rows will be examined upon further analysis.

Random forest imputation was used to fill in the missing categorical data for several reasons. Although it is computationally more expensive than using methods such as the mode to fill in missing data, it deals well with variables with no association. Rating and risk rating seem to have no association, as can be seen in their heat map:

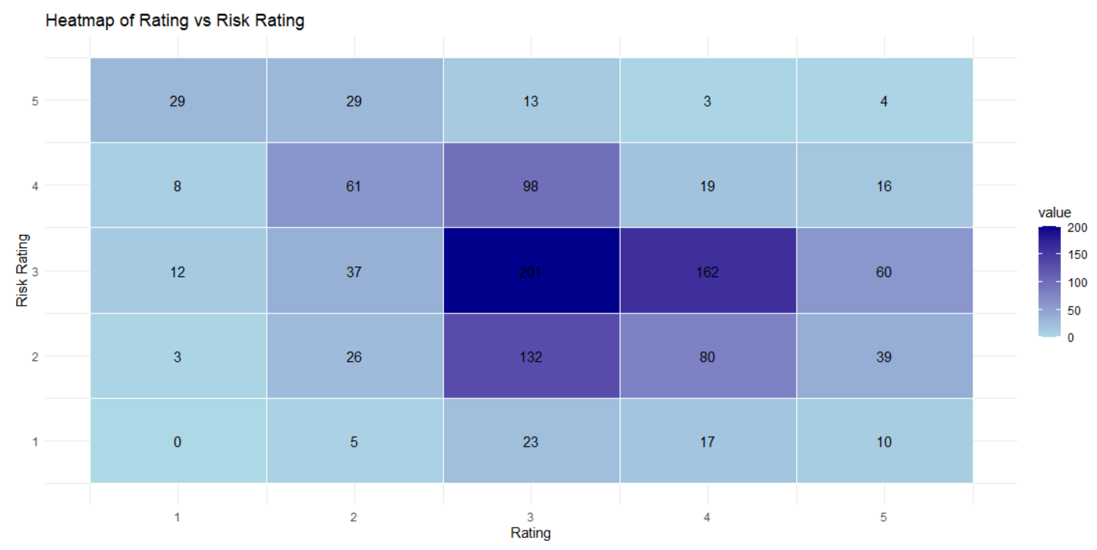


Figure 2.3: Rating vs Risk Rating

Random forest imputation is also designed for categorical data, and outliers less influence them as they use an ensemble of decision trees, reducing the impact of outliers on the overall imputation process.[8] Random forest imputations also do not require prior assumptions of the underlying distribution, unlike methods such as linear model imputation, which requires underlying distribution normality, making them flexible for various data structures.[8] Finally, the dataset is missing at random and random forest imputation handles missing values in predictors by using surrogate splits, meaning imputations are accurate even for incomplete predictors.[8]

So, now, we will explain how it predicts the equity fund dataset. Here is the code where the model is trained:

```
rf_model <- ranger( formula = as.formula(paste(column, "~.")),
data = data[!is.na(data[[column]]), ], na.action = "na.omit", classification = TRUE)
```

The random forest model is trained on rows with no NA values, and `classification = TRUE` is set to make sure the random forest is predicting categorical values rather than numerical ones. This line is important: `formula = as.formula(paste(column, "∼.")` as it ensures the target column, the one with missing values, is not a predictor. The model learns relationships between the target variable and the other columns in the dataset.

To make more robust predictions, Random Forests grow many decision trees using random rows and features. Each tree is trained on a randomly selected subset of rows from the data; some rows may appear multiple times, and others not at all. At each split in a tree, only a random subset of the columns is considered, further dividing the data into smaller groups. This randomness ensures the trees capture different patterns in the data. Predictions from the trees are aggregated to fill in missing values, reducing the impact of over-fitting by averaging out biases and individual tree errors.

Linear model imputation is used to fill in the missing numerical features. This method is chosen because it uses the relationship between variables, unlike more straightforward methods such as mean imputation.[9] Also, the data here is missing at random and regression imputation gives consistent and unbiased estimates for this case.[10] Random forest imputation does not need a specification of variable types because variables are sorted based on how the trees are split. Still, the linear model does, so the categorical variables had to be dealt with. Frequency encoding was used because `category` had a large number of levels. Then, to fill in the numerical values, a function was made, and it applied the linear model: `lm_model <- lm(as.formula(paste(column, "~.")), data = data, na.action = na.exclude)`

2.4 Multicollinearity

VIF analysis will be done using a standard multiple response regression linear model by `lm(cbind(roe, sustainability_score)~.)`. The categorical variables are involved using frequency encoding as with linear model imputation. After this, a dummy response variable is added for VIF calculation, and the VIF values are derived from this new model. If any of the predictors have VIF values of more than 10, they would need to be removed due to multicollinearity. In this case, all the values are less than 5, so none of these variables have multicollinearity, and the dataset is ready for analysis.

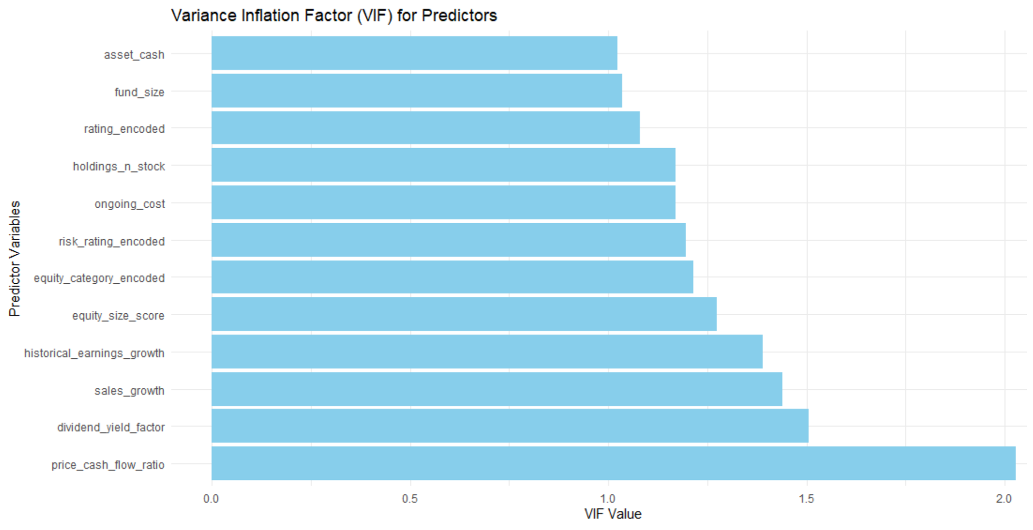


Figure 2.4: Barplot showing VIF values

Chapter 3 Multiple Response Regression

3.1 Introduction

3.1.1 Single Response Regression

Before delving into multiple-response regression, let us start with single-response regression. Single response regression, more commonly known as simple linear regression, is used to gain insight into the relationship between a single response (dependent) variable and one or more predictor (independent) variables. The general form of the linear regression model is:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Where $\boldsymbol{Y}$ is the response variable, $\boldsymbol{X}$ is the predictor variable, $\boldsymbol{\beta}$ is the fixed unknown coefficients and $\boldsymbol{\epsilon}$ the noise or error variable.

When modelling data, the aim is to reduce the error between predicted and actual values, otherwise known as minimising the residuals. The optimal coefficient vector, $\hat{\boldsymbol{\beta}}$, is found by minimising the sum of squared residuals. Another way to define this is:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T Y}$$

The model's accuracy can be evaluated using metrics such as the R-squared value, which measures the proportion of variance in the dependent variable explained by the independent variables. However, this approach only considers a single dependent variable and does not capture relationships between multiple response variables.

3.1.2 Multiple Response Regression

Multiple response regression (MRR) models a relationship between various response (dependent) variables and one or more predictor variables (independent).[11] The difference from single-response regression is that MRR must also consider the potential relationship between the dependent variables.

The general form of the multiple response regression model is:

$$\boldsymbol{Y}_{(n\times m)} = \boldsymbol{Z}_{(n\times(r+1))}\boldsymbol{\beta}_{((r+1)\times m)} + \boldsymbol{\epsilon}_{(n\times m)}$$

with

$$E(\epsilon_i) = 0 \quad \text{and} \quad \text{Cov}(\epsilon_i, \epsilon_k) = \sigma_{ik} I \quad i,k = 1,2,\ldots,m$$

The m observations on the jth trial have covariance matrix $\Sigma = \{\sigma_{ik}\}$, but observations from different trials are uncorrelated.[11] Here $\boldsymbol{\beta}$ and σ_{ik} are unknown parameters; the design matrix $\boldsymbol{Z}$ has jth row $[z_{j0}, z_{j1}, \ldots, z_{jr}]$.[11]

Each value in the covariance matrix, σ_{ik}, is the covariance between the response variables i and k. A non-zero covariance means there is a relationship between these variables. In multiple response regression, the covariance matrix captures interdependencies between different response variables, which can illustrate how these variables jointly vary and affect the overall model outcomes. This joint information improves estimation and prediction.[12]
The i-th response $\mathbf{Y}_{(i)}$ follows the linear regression model:

$$\mathbf{Y}_{(i)} = \mathbf{Z}\boldsymbol{\beta}_{(i)} + \boldsymbol{\epsilon}_{(i)}, \quad i = 1,2,\ldots,m$$

with $\text{Cov}(\boldsymbol{\epsilon}_{(i)}) = \sigma_{ii}\mathbf{I}$ as expected, but the errors associated with different responses within the same trial may be correlated.[11] Each response variable i is modelled independently, but all responses share the same set of predictors.

The least squares (LS) estimates $\hat{\boldsymbol{\beta}}_{(i)}$ can be determined solely from the observations $\mathbf{Y}_{(i)}$ on the i-th response.[11] This is just the single response logic but applied to each response variable separately:

$$\hat{\boldsymbol{\beta}}_{(i)} = (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\mathbf{Y}_{(i)}$$

Collecting these univariate least squares estimates gives:[11]

$$\hat{\boldsymbol{\beta}} = \left[\hat{\boldsymbol{\beta}}_{(1)} \ \vdots \ \hat{\boldsymbol{\beta}}_{(2)} \ \vdots \ \cdots \ \vdots \ \hat{\boldsymbol{\beta}}_{(m)}\right] = (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\left[\mathbf{Y}_{(1)} \ \vdots \ \mathbf{Y}_{(2)} \ \vdots \ \cdots \ \vdots \ \mathbf{Y}_{(m)}\right]$$

Or equivalently:

$$\hat{\boldsymbol{\beta}} = (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\mathbf{Y}.$$

Using the least squares estimate, $\hat{\boldsymbol{\beta}}$, the matrices of the predicted values and residuals can be made.[11]
Predicted values:

$$\hat{\mathbf{Y}} = \mathbf{Z}\hat{\boldsymbol{\beta}} = \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{Y}$$

Residuals:

$$\hat{\varepsilon} = \mathbf{Y} - \hat{\mathbf{Y}} = [\mathbf{I} - \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}']\mathbf{Y}$$

Orthogonality amongst residuals predicted values, and columns of **Z**, which hold in single response regression, hold in multivariate multiple regression.[11] This comes from:

$$\mathbf{Z}'[\mathbf{I} - \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'] = \mathbf{Z}' - \mathbf{Z}' = \mathbf{0}.$$

Multiple Response Regression has a few underlying assumptions. These are linearity, homoscedasticity, independence of errors, normality, and independence of independent variables.[13]

Linearity means there is a linear relationship between the dependent variables and the independent variables. Residuals should be normally distributed with a mean of 0 and variance σ.[14] This covers homoscedasticity, error independence and normality. Homoscedasticity is when the variance of the residuals is constant. The independence of independent variables means the independent variables are not too highly correlated.[14]

MRR comes with many benefits and drawbacks. One of its main strengths is its ability to capture correlations between the responses, as outlined previously. This must be modelled for valid inference.[15] Incorporating this correlation is necessary because it reduces standard errors compared to the estimates based on independent correlation.[15] Standard errors determine confidence in the results. MRR also excludes non-influential covariates, resulting in a simpler model with better interpretive and predictive value.[15]

One of the most significant limitations of MRR outputs is that they are not always able to interpret.[16] In addition, multivariate techniques to give meaningful results need a large sample of data; otherwise, the results are meaningless due to high standard errors.[16] There is more confidence in the results from a large sample rather than a small one.

Given the complexity of modelling relationships in multiple response regression, selecting the most significant predictors is crucial. Various methods exist for this.

3.1.3 Multiple Response Selection Methods

These selection methods are forward, backward and bidirectional stepwise selection. They are used because they are more computationally efficient than best subset selection as they consider a smaller set of models. Forward stepwise selection starts with a model with no predictors, and it iteratively adds the most useful predictor one at a time. At each step, the variable with the greatest fit improvement is added to the model. Backward stepwise selection starts with the full model with all its predictors, and it iteratively removes the least useful predictor one at a time. Bi-directional stepwise selection combines forward selection and backward elimination. What changes from prior methods is that this method considers the statistical impact of dropping variables that were considered previously.[17]

When using all of these selection methods, in single response regression, the "best" model is chosen for every number of predictors, and then the single best model is picked. The "best" model for each number of predictors is the one with the smallest residual sum of squares, or largest R^2, and the single best model is picked using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2. The way in which the models will be compared should be considered before applying these selection methods to the data.

However, this changes for multiple response regression because we need to consider the joint information between the response variables. This report aims to find the model with the best fit, but how will this fit be evaluated?

3.2 Model Evaluation and Performance

Multiple models can be used here, and this report will showcase how accurate they are using the average Root Mean Square Error - Standard Deviation Ratio (RSR), which is the root mean squared error (RMSE) over the standard deviation of said column. This is chosen because the RMSE evaluates the model for said response variable in its units, but the RSR returns a universal model evaluation. The average is taken because there are two response variables. This will be called the average RSR (ARSR). The ARSR gives the proportion of natural variability in the response variable not accounted for by the model. An excellent model has a 0-0.5 ARSR, a good model has a 0.5-0.6 ARSR, a satisfactory model has a 0.6-0.7 ARSR, and an unsatisfactory model has a ¿0.7 ARSR.[18] Now, a simple model will be tested.

3.2.1 Standard Multiple Linear Regression

The first model is a standard multiple linear regression model which models the response variables against the sum of the predictors, which in R is:

```
lm(cbind(roe, sustainability_score) ~.  - rating - risk_rating - category, data = data)
```

The residuals from the model are used to calculate the RSR, which it returns as a vector. Using this model, the RSR was `0.767` for ROE and `0.783` for sustainability score, and the average was `0.775`. This is a high value, indicating this model does not perform well on the equity fund dataset, which makes sense because the model is simple, so it underfits the data.

3.2.2 Stepwise Selection Code

All of the selection methods had to be coded manually because the `stepAIC` command, which does this in R, only applies to single response regression. This was broken down into a few functions to make the final function simpler. Firstly, a `calculate_ARSR` function was made to determine model fit. Then, two functions were made, one to add predictors and one to remove predictors. The `add_predictors` function evaluates the impact of adding each remaining predictor to the current model by calculating the ARSR. It iterates through all the remaining predictors, and a new regression formula is built for each predictor. For example, if the response variables are `y1` and `y2`, and the predictors already selected are `x1` and `x2`, adding a predictor `x3` generates: `cbind(y1, y2)` $\sim$ `x1 + x2 + x3`. To ensure smoothness, the function uses a "try-catch" mechanism to handle errors that may occur during model fitting. If an error occurs, the function records `NA` for that predictor and continues.

The `remove_predictors` function evaluates the effect of removing predictors from an existing regression model, ensuring that only predictors that significantly contribute to reducing the model error are retained. Similar to `add_predictors`, this function calculates the ARSR for models with the selected predictor removed.

The `stepwise_multivariate` function brings this all together: If the method is "forward" or "stepwise", the function calls `add_predictors` to calculate ARSR for models with each remaining predictor added. It updates two lists: `metrics` and `candidate_models` with the results. A candidate model is a potential model considered during the stepwise selection process, and `metrics` contains the list of RSR values. If the method is "backward" or "stepwise", the function calls `remove_predictors` to calculate ARSR for models with each selected predictor removed. It also updates the `metrics` and `candidate_models` like the "forward" method.

3.2.3 Forward and Bi-directional Stepwise Selection

Both of these methods lead to the same model with all of the predictors. This model is also the sum of all the predictors in the dataset, matching the previously calculated ARSR. Some of the variables have the label encoded because they are categorical. This plot shows the ARSR slowly decreasing as more predictors are added. The added predictors are labelled.

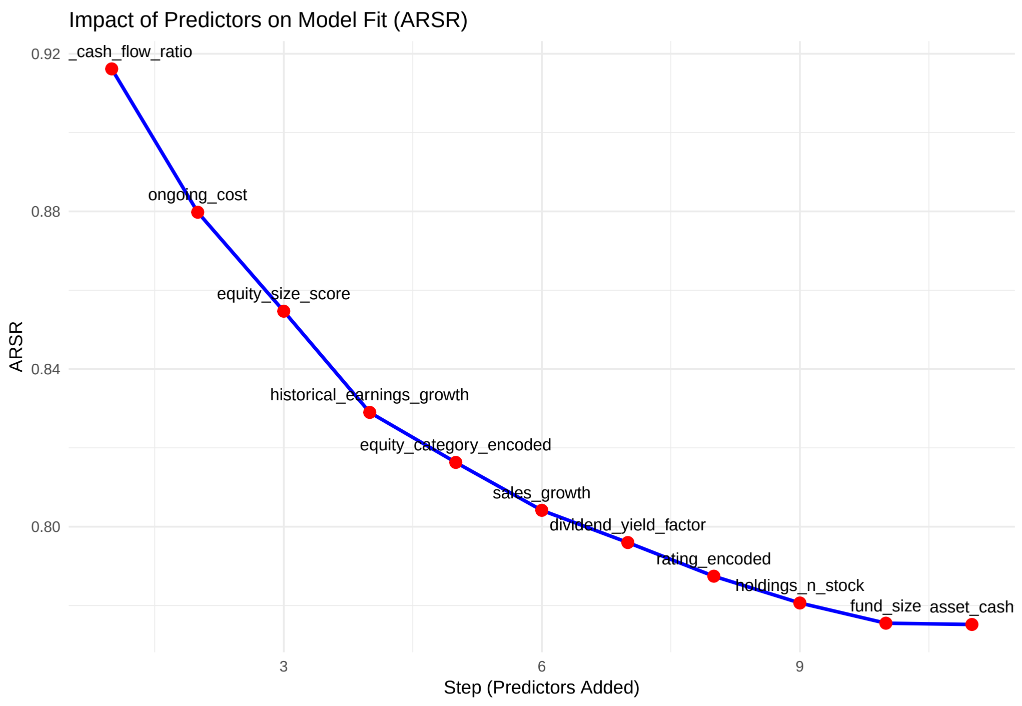


Figure 3.1: Impact of Predictors on Model Fit

3.2.4 Backward Elimination

Backward elimination returned a different model to the other selection methods. The difference was that it did not include `risk_rating_encoded`. This gave a higher ARSR than the previous model with `0.776`, which is very close. It is not unusual that backward elimination produced a different model because each selection method has different starting points and directions, which can lead to other paths through the model improvement and, ultimately, a different set of selected predictors.

3.2.5 Extending the Stepwise Selection

Although there are a large number of predictors in this dataset, extending the model to consider non-linear terms and interaction terms can seek to improve its performance. Nothing much needed to change in the code, just making a new function which generated the new terms for the data. Upon using the interaction terms, the model improved a lot, and the ARSR was `0.590`. Although this significance is noteworthy, the model that was outputted was too complex to be used. When generating the non-linear terms, the model also improved. The ARSR was `0.656`. But the same issue arises, the outputted model has too many predictors to be helpful.

3.2.6 Evaluating Stepwise Selection

Stepwise selection comes with multiple benefits. Firstly, it is easy to implement. Although manual code had to be made in this case, it proved to be easy to test on the dataset. They reduce the work of model building to the click of a button and often yield simple results.[19] Also, they can sift through a large number of predictors to identify potential relationships, mainly when dealing with high-dimensional datasets.

However, they come with a few drawbacks. Firstly, they are prone to over-fitting. They can find idiosyncrasies in the population that do not exist.[19] The extension of stepwise selection subsection showed this best because although the models produced improved the ARSR significantly, they had more than 20 predictors, which is too complex. Also, the stepwise selection method produces models that are highly sensitive to slight variations in the data, causing inconsistent results. Another issue of this method is inflated effect sizes. Even when the model identifies valid predictors, it overestimates their effect sizes.[19] This means that the impact of significant predictors on the response variables `roe` and `sustainability_score` will be inflated.

None of these issues apply to shrinkage methods. They address them by applying penalties to the model coefficients, which prevent over-fitting, reduce sensitivity to data variations, and control for inflated effect sizes.

Chapter 4 Shrinkage Methods

4.1 Theory

The previous methods are known as discrete model search methods, where the best model is picked based on information criteria. The next type of model this report will look at is shrinkage methods, which are a continuous model search method. Only the whole model is trained, but the estimated regression coefficients are minimised or zeroed, and there is a general solution of the form: model fit + penalty on coefficient size. Ridge regression zeroes estimated regression coefficients.

4.1.1 Ridge Regression

The goal of ridge regression is to minimise the cost function:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\text{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2,$$

where $\lambda \geq 0$ is a tuning parameter. The role of this parameter is crucial as it balances the trade-off between model fit and the size of the coefficients. The ridge regression cost function estimates are the solution to the least squares problem subject to the constraint $\sum_{j=1}^{p}\beta_j^2 \leq c$. A geometric interpretation of this interpretation is a L_2 ball, which is why ridge cannot penalise coefficient sizes completely to zero, even for irrelevant features. As λ increases, the penalty term gets larger, so in order to minimise the entire function (model fit + penalty), the regression coefficients will get smaller.

Ridge regression minimises the cost function but why is this the case. This can be shown by differentiating the cost function with respect to $\boldsymbol{\beta}$, giving the closed-form estimator: $\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}_p\right)^{-1}\mathbf{X}^T\mathbf{Y}$. This is the cost function Ridge Regression aims to minimise in a different format:

$$\|\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{\beta}\|_2^2+\lambda\|\boldsymbol{\beta}\|_2^2 = RSS+\lambda\boldsymbol{\beta}^T\boldsymbol{\beta}$$

The second equality is expressing the residual sum of squares (RSS) as a matrix, which will be expanded upon and differentiated.

$$\begin{aligned}RSS + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}\\ &= \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}\end{aligned}$$

Take the gradient with respect to $\boldsymbol{\beta}$ and set it to zero for the minimum:

$$\begin{aligned}-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} &= 0,\\ -\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} &= 0,\\ \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{y},\\ \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{y},\\ \hat{\boldsymbol{\beta}}_{\text{Ridge}} &= \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}. \quad \blacksquare\end{aligned}$$

Ridge regression comes with a few benefits. It is useful when predictors have high multicollinearity and when p is close to n, so high-dimensional data. It is also faster than the prior model-search methods as it only trains one model.

Unfortunately, it also brings some drawbacks. Ridge regression does not carry out feature selection. Although it can indicate the coefficients which contribute the most, it cannot remove those coefficients entirely. Ideally, a simpler model would be created rather than including all of the predictors.

4.1.2 Lasso Regression

Lasso regression can remove irrelevant features by shrinking their respective coefficients to zero. It penalises coefficient size through absolute values instead of squares. It also has the goal of minimising the cost function, but this formula differs:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|,$$

Lasso minimises the cost function in a similar way as was proven by Ridge. The cost function is expanded as before. So, the cost function becomes:

$$J(\boldsymbol{\beta}) = \mathbf{Y}^\top\mathbf{Y} - 2\mathbf{Y}^\top\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \lambda\sum_{j=1}^{p}|\beta_j|$$

To minimise $J(\boldsymbol{\beta})$, compute the derivative (or sub-gradient) with respect to $\boldsymbol{\beta}$. Since the

L_1-norm $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$ is not differentiable at $\beta_j = 0$, the sub-gradient is used:

$$\frac{\partial|\beta_j|}{\partial\beta_j} = \begin{cases} +1 & \text{if } \beta_j > 0 \\ -1 & \text{if } \beta_j < 0 \\ [-1, +1] & \text{if } \beta_j = 0 \end{cases}$$

The derivative of the quadratic term is:

$$\frac{\partial}{\partial\boldsymbol{\beta}}\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = -2\mathbf{X}^\top\mathbf{Y} + 2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta}$$

Combining these, the sub-gradient of $J(\boldsymbol{\beta})$ is:

$$\frac{\partial J(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = -2\mathbf{X}^\top\mathbf{Y} + 2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \lambda\mathbf{s}$$

where $\mathbf{s} \in \partial\|\boldsymbol{\beta}\|_1$ is the sub-gradient of the L_1-norm. The sub-gradient is set to zero to find the minimum:

$$\begin{aligned} -2\mathbf{X}^\top\mathbf{Y} + 2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \lambda\mathbf{s} = 0 \\ 2\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = 2\mathbf{X}^\top\mathbf{Y} - \lambda\mathbf{s} \\ \mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\top\mathbf{Y} - \frac{\lambda}{2}\mathbf{s} \end{aligned}$$

The presence of s, the sub-gradient of the L_1-norm, provides the sparsity property of Lasso. Specifically, for $\beta_j \neq 0$, $s = \text{sign}(\beta_j)$, which shrinks β_j, while for $\beta_j = 0$, $s \in [-1, 1]$, allowing coefficients to be exactly zero. ■

The lasso regression cost function estimates are the solution to the least squares problem subject to the constraint $\sum_{j=1}^{p} |\beta_j| \leq c$. A geometric interpretation of this interpretation is a diamond, which is why lasso can penalise coefficient sizes ultimately to zero for irrelevant features.

Lasso and Ridge have the same benefits in that they introduce bias but decrease variance, improving predictive performance. It is also scalable and can work with high-dimensional data. Both of these methods can also deal with high multicollinearity. In this case, any multicollinearity has been sorted in data pre-processing, but these regression models can also remove or reduce the impact of irrelevant features.

4.1.3 Elastic Net

The Elastic Net uses a convex combination of the penalties of lasso and ridge regression, with the following cost function minimisation:[20]

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\alpha\sum_{j=1}^{p}|\beta_j|+\lambda(1-\alpha)\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\alpha\sum_{j=1}^{p}|\beta_j|+\lambda(1-\alpha)\sum_{j=1}^{p}\beta_j^2$$

where, α controls the trade-off between the L_1 and L_2 penalties. It can be proven to minimise its cost function by combining the lasso and ridge proofs previously.

The elastic net regression cost function estimates are the solution to the least squares problem subject to the constraint $\alpha\sum_{j=1}^{p}|\beta_j|+(1-\alpha)\sum_{j=1}^{p}\beta_j^2\leq c$, where $\alpha\in[0,1]$. A geometric interpretation of this is a diamond mixed with a circle depending on what α is. $\alpha=0$ corresponds to ridge regression and $\alpha=1$ corresponds to lasso regression. Here is how all the shrinkage methods are visually:

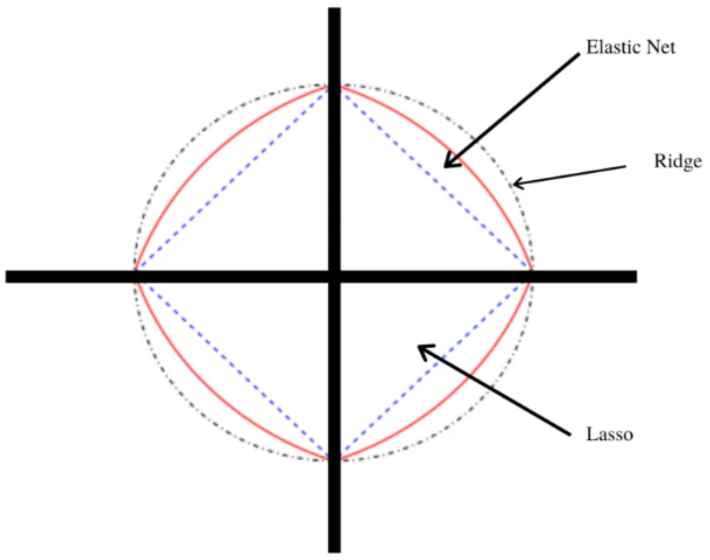


Figure 4.1: Geometric Interpretation of Shrinkage Methods[21]

One of the major benefits of the Elastic Net is its grouping effect. This can be derived: given data $(\mathbf{y},\mathbf{X})$ and parameters λ_1,λ_2,$\mathbf{y}$, the response, is centred and the predictors $\mathbf{X}$ are standardised.[20] Let $\hat{\beta}(\lambda_1,\lambda_2)$ be the naïve Elastic Net estimate. Suppose that $\hat{\beta}_i(\lambda_1,\lambda_2)\hat{\beta}_j(\lambda_1,\lambda_2)>0$. Define:

$$D_{\lambda_1,\lambda_2}(i,j)=\frac{1}{\|\mathbf{y}\|_1}\left|\hat{\beta}_i(\lambda_1,\lambda_2)-\hat{\beta}_j(\lambda_1,\lambda_2)\right|,$$

Then:

$$D_{\lambda_1,\lambda_2}(i,j)\leq\frac{1}{\lambda_2}\sqrt{2(1-\rho)},$$

where $\rho=\mathbf{x}_i^T\mathbf{x}_j$, the sample correlation.
$D_{\lambda_1,\lambda_2}(i,j)$ is the difference between the coefficient paths of predictors i and j.[20] If $\mathbf{x}_i$ and $\mathbf{x}_j$ are highly correlated, the difference between the coefficient paths of predictor i and

predictor j is almost 0.[20] The upper bound in the above inequality is a mathematical representation of the grouping effect of the naïve Elastic Net. Only the ridge parameter, λ_2, is in the grouping effect and not λ_1 because ridge exhibits this property, unlike lasso.

Another benefit of the Elastic Net is that it retains the variable selection property of Lasso and remains computationally efficient while overcoming its limitation with high-dimensional data.[20] Elastic Net also handles correlated predictors better by encouraging a group effect in contrast to Lasso, which arbitrarily selects one variable from a group of highly correlated predictors.[20] Although there is a moderate correlation between some of the predictors, this has mainly been sorted in pre-processing. However, Elastic Net does improve prediction accuracy. In situations where both L_1 and L_2 penalties are needed, Elastic Net balances the two, enhancing prediction performance than Lasso or Ridge alone.[20] Finally, the Elastic Net is flexible with tuning. The two regularisation parameters, λ_1 and λ_2, allow the model to adjust the degree of sparsity and stability, making the Elastic Net a flexible tool for regression.[20]

However, it does come with quite a few drawbacks. Its first disadvantage is the complexity of tuning. Tuning the Ridge and Lasso regularisation parameters simultaneously makes Elastic Net computationally more intensive than Lasso or Ridge.[20] Over-shrinkage can be a consequence of improperly chosen regularisation parameters, which can result in biased coefficient estimates.[20] The next issue is interpretability in group selection. The grouping effect, while useful, can also select irrelevant predictors if they are highly correlated with relevant ones, making interpretation more difficult.[20] The final issue of Elastic Net is its dependence on standardisation. Elastic Net assumes predictors have been standardised, as the penalties are applied based on the scale of the variables.[20]

4.1.4 Shrinkage Methods for MRR

Previous discussions on shrinkage methods have been in the single response case. They can be adjusted for multiple response regression. The extension of Ridge regression to the numerous response setting uses the Frobenius norm of the coefficient matrix $\mathbf{W}$:

$$\min_{\mathbf{W}} \frac{1}{2}\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda\|\mathbf{W}\|_F^2$$

where $\mathbf{Y}$ is the $n \times q$ matrix of response variables (for q responses), $\mathbf{W}$ is the $m \times q$ matrix of regression coefficients, $\|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2$ is the Frobenius norm, which measures the sum of squared residuals across all responses, $\|\mathbf{W}\|_F^2 = \sum_{j=1}^{q}\sum_{i=1}^{m} w_{ij}^2$ is the ridge penalty applied to all coefficients in $\mathbf{W}$, and λ is the regularisation parameter controlling the shrinkage. Here, the Frobenius norm replaces the 2-norm. This means the L_2-norm penalty is applied to all coefficients across the matrix $\mathbf{W}$.

The solution for $\mathbf{W}$ in MRR is:

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_m)^{-1} \mathbf{X}^\top \mathbf{Y}$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix and $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_m$ ensures invertibility through regularisation. Lasso can also be expressed in the multiple response form as:

$$\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_2^2 + \lambda \|\mathbf{W}\|_1,$$

where $\|\mathbf{W}\|_1 = \sum_{j=1}^m |w_j|$ is the L_1-norm penalty. The L_1-penalty induces sparsity, meaning it shrinks some elements of $\mathbf{w}$ to exactly zero, allowing variable selection in the regression model.

Tikka et al. (2007) introduced an extension of lasso used in multiple response regression called lasso simultaneous variable selection (L_2-SVS). It extends the lasso to account for the relationships across multiple response variables simultaneously. The L_2-SVS approach minimises the residual error while enforcing sparsity in the solution by penalising the model with a row-wise L_2-norm combined with an L_1-norm over the rows of the coefficient matrix. The optimisation problem for L_2-SVS can be written as:

$$\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda \|\mathbf{W}\|_{1,2},$$

where: $\|\mathbf{W}\|_{1,2} = \sum_{j=1}^m \|\mathbf{w}_j\|_2$ is the mixed norm, combining the row-wise L_2-norm ($\|\mathbf{w}_j\|_2$) with an L_1-penalty. The L_2-SVS regression model encourages sparsity by driving entire rows of $\mathbf{W}$ to zero, effectively removing predictors from the model across all responses. This property makes L_2-SVS particularly effective in scenarios where predictors are expected to be relevant (or irrelevant) across multiple response variables. L_2-SVS has high prediction and selection accuracy when many input variables are available for model construction.[22] It is consistently more precise and stable than ridge, and it outperforms lasso, especially when the input variables are highly correlated.[22]
Elastic Net combines the standard Lasso and Ridge penalties:

$$\min_{\mathbf{W}} \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\mathbf{W}\|_2^2 + \alpha\lambda \|\mathbf{W}\|_1 + (1-\alpha)\lambda \|\mathbf{W}\|_2^2,$$

where $\|\mathbf{W}\|_1 = \sum_{j=1}^m |w_j|$ is the sparsity-inducing penalty, and $\|\mathbf{W}\|_2^2 = \sum_{j=1}^m w_j^2$ adds the ridge penalty to stabilise the model.

4.2 Application

Now, these regression models will be applied to the dataset to see how well the predictors can model the response variables: `roe` and `sustainability_score`. This can be done by using `family = "mgaussian"` in `cv.glmnet`. This will consider the correlation between response variables and maintain consistent variable selection, which is the aim of MRR.

Lasso had an ARSR of `0.737`, Ridge had ARSR of `0.741` and Group Lasso had an ARSR of `0.739`. Elastic Net had the lowest ARSR of `0.736` because of its Lasso and Ridge combination which allows it to handle datasets with both sparsity and high multicollinearity effectively. As a result, it balances feature selection and regularisation, resulting in the lowest ARSR. The feature selection capability of Lasso performs well. However, it was slightly worse than Elastic Net due to its limitations in handling correlated predictors and bias in over-shrinking weaker predictor coefficients. Despite these challenges, Lasso's ARSR suggests that sparsity played a significant role in the dataset.

Group Lasso uses the $L_{2,1}$ penalty, enforcing sparsity for groups rather than for individual predictors. In this case, its performance was slightly worse than Lasso and Elastic Net due to the reduced flexibility of group-level constraints, which may retain irrelevant groups or eliminate entire groups prematurely. The additional constraints introduced by grouping variables made it less effective when individual feature sparsity was more relevant. The slight increase in ARSR suggests that the grouping strategy did not perfectly align with the data structure. Ridge regression performed the worst among the models, as it applies only an L_2 penalty, which shrinks all coefficients uniformly toward zero but does not enforce sparsity. Ridge retains all predictors, including irrelevant ones, which can introduce noise and inflate residuals. Additionally, its inability to address sparsity makes it less effective in datasets with many irrelevant predictors. Despite these limitations, Ridge's ability to handle multicollinearity effectively prevents over-fitting, keeping it competitive, albeit at the highest ARSR.

Here is a representation of how the ARSR changed with the α parameter, which controlled how much the L_1 and L_2 penalties contributed to the model:

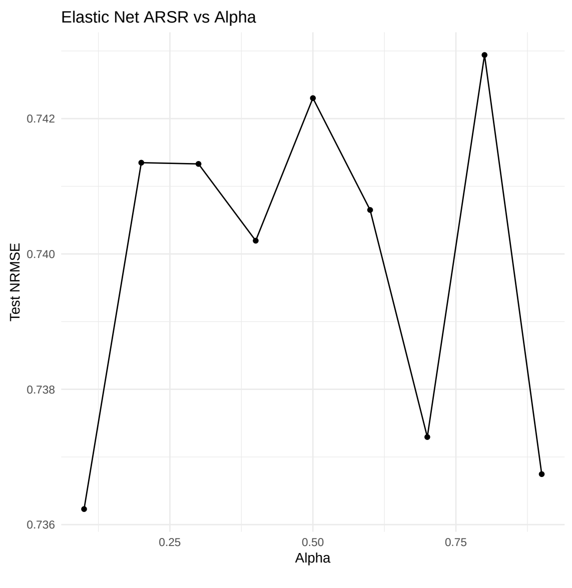


Figure 4.2: A plot of α vs Elastic Net ARSR

Chapter 5 Random Forest Regression

5.1 Theory

5.1.1 Classification and Regression Trees (CARTs)

Random forests are tree-based models built upon classification and regression trees (CARTs) as a building block. A CART partitions the training data into groups with similar response values and then fits a simple constant in each subgroup.

The CART starts with a root node containing all the objects and is then divided into "leaf" nodes by recursive binary splitting.[23] Each leaf node relates to a hyper-rectangle in feature (predictor) space, R_j, $j = \{1, \dots, J\}$, i.e. the feature space defined by $X_1, X_2, \dots, X_p$ is split into J distinct regions which do not overlap, which is shown below:

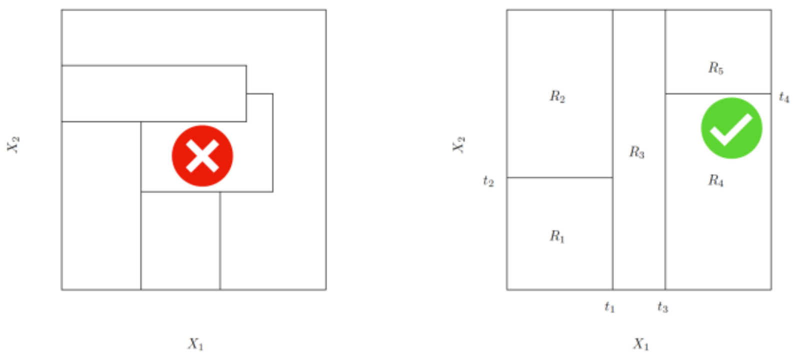


Figure 5.1: Incorrect vs. Correct Partitioning

The Greedy-Fitting Algorithm ensures each split is chosen to minimise the RSS:

$$\sum_{j=1}^{J} \sum_{i:x_i \in R_j} \left(y_i - \hat{y}_{R_j}\right)^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j-th rectangle.

A regression tree is used because the response variables here, `roe` and `sustainability_score`, are continuous. A regression tree assigns a value to each predictor space, R_j, i.e. the model predicts the output (i.e. which feature space) based on the average response values for all observations in that subgroup.

The Greedy fitting algorithm gives good predictions for the training data but could over-fit it, causing poor test data performance. The weakest link pruning algorithm prunes trees to address this bias-variance trade-off and over-fitting.

The weakest link pruning algorithm introduces a non-negative tuning parameter α, for regression trees, which minimises:

$$\sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|,$$

, where $|T|$ is the number of terminal nodes (size) of the tree T.

5.1.2 Bagging for CARTs

Although CARTs are simple and easy to interpret and are similar to standard decision-making processes, the trees generally have high variance, i.e. small sample changes lead to significant changes in the fit, and they tend to have poor predictive accuracy. Bootstrap aggregating, also known as bagging, improves the stability and accuracy of classification and regression algorithms by model averaging. Bagging helps reduce variance and avoid over-fitting, but the model becomes more challenging to interpret.

For regression trees, average all the predictions to obtain:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

Bagging for CART addresses the overfitting issue by growing large trees with minimal pruning and relying on the bagging procedure directly to avoid overfitting. It also addresses overfitting through pruning, which was explained previously.

Bagging is advantageous in that if there is a lot of data, which is true here, bagging is a good option; the empirical distribution will be closer to the actual underlying population's distribution. Also, it is the best option when the goal is to reduce the variance of a predictor. However, it also brings some disadvantages; interpretability is lost as the final estimate is not a tree; it is an ensemble prediction, i.e. multiple trees are combined to make the final prediction.[24]. Also, bagging trees are inherently correlated, which is a problem because the more correlated the random variables are, the less the variance reduction of their average, which can undermine one of its key advantages.

5.1.3 Random Forests

Random Forests make bagged CART models more independent, improving variance reduction in their ensemble predictions. They do this by resampling observations and restricting the model to random subspaces, $\mathcal{X}' \subset \mathcal{X}$, which ensures the tree explores different parts of the data, meaning the ensembles are more diverse and hence influential.

The Random Forests algorithm is used in the following way: a bootstrap resample $(y_i^*, x_i^*)_{i=1}^n$ is taken of the training data like for bagging. Then, the tree is built; each time a split in a tree is considered, randomly select m predictors out of the full set of p predictors as split candidates and find the best split within those m predictors. Typically m is $p/3$. Finally, repeat the first two steps, averaging the prediction of all the regression forest trees.

Random forests are a great option, but they are not interpretable when bagging and using random subspaces. Thus, quantifying the importance of each variable can be difficult. There are two popular approaches which quantify this importance.

One approach is to run a loop over each tree in the forest to determine the significance of each feature variable x_j, where $j = 1, \ldots, p$. Every node that splits on x_j for every tree is identified, and how much each split improves the chosen loss criterion, such as accuracy, Gini index, etc., is calculated. Each improvement is then summed for every tree in the forest, indicating the significance of x_j.

Another approach is to use out-of-bag (oob) error estimates, which can be computed via bagging, and they can also be used to determine the significance of each feature x_j, where $j = 1, \ldots, p$. Each tree's predictive accuracy is calculated after the oob samples, represented by x^{oob}, are extracted. The ties between x_j and the other variables are then broken by randomly permuting the entries of the j-th column of x^{oob}. After passing the modified $x^{\text{oob}*}$ matrix through the tree, the change in predictive accuracy is computed for that tree. The decrease in accuracy caused by the permutation is averaged over all trees, giving a measure of the importance of the feature x_j.

Random forests inherit the advantages of bagging and trees, and tuning is rarely needed. However, they are hard to implement and have the same extrapolation issue as trees.

5.1.4 Random Forests & MRR

This chapter has looked at single-response regression so far. Multivariate Regression Trees (MRTs), as introduced by De'ath (2002), are an extension of CART to handle several response variables.[23] MRTs split a response matrix into clusters based on thresholds of explanatory variables. This approach allows MRT to model complex, multivariate relationships by partitioning the data in a way that accounts for interactions between explanatory and response variables. Unlike single-response regression trees, MRT highlights local structures in the data, making it useful for ecological and environmental modelling.[25]

MRTs are constructed similarly to CARTs, but they require criteria that account for the joint distribution of multiple response variables. The impurity measure in MRT minimises the total within-node variance across all response variables. This measure is computed as the total sum of squares of the response values around the multivariate mean at each node.[23] For a node with objects and variables, the impurity is expressed as:

$$\text{impurity} = \sum_{i=1}^{n} \sum_{j=1}^{p} (y_{ij} - \bar{y}_j)^2$$

The methods which determine tree splits and optimal tree selection based on cross-validation, along with other concepts and practices of CART, still apply to MRT. For example, both CART and MRT utilise measures like Gini impurity, entropy, or variance to determine the best splits and cross-validation to select the optimal tree structure.[23]

Tree splitting and optimal tree selection in MRT are guided by automatic cross-validation and pruning, similar to CART. Cross-validation involves using a subset of the data to construct the tree and then evaluating its performance on the remaining data[25]. The best tree is selected by identifying the smallest tree within one standard error of the tree with the lowest cross-validated relative error (CVRE).[25] This approach, known as the 1-SE rule, prioritises simpler, more generalisable models.

However, for MRTs, specific visualisation tools are essential for interpreting their results. For instance, at each node of the tree, a bar plot can show the distribution of each response variable within that node. A bar plot helps to visualise the multivariate outcomes and understand the impact of splits on all response variables collectively.[23]

MRTs use hierarchical clustering to split data into clusters based on thresholds of explanatory variables, which ensures that the tree structure highlights local structures and variable interactions.[25] This process is iterative and continues until each leaf node contains a single observation.

MRTs produce easy-to-interpret tree structures that visualise local structures and interactions among variables. Each node represents a split based on an explanatory variable, and the leaves show the final clusters. Thus, it is easy to identify the importance of explanatory variables and interpret the results. In MRTs, feature importance measures are adapted to reflect their influence on the combined variance of all response variables, which involves computing the contribution of each feature to the reduction in multivariate variance, providing insights into their overall impact on the responses. This approach is detailed in studies like De'ath (2002), which explores how features can be analysed to understand their effects on species-environment relationships in a multivariate context.[26]

MRTs also better address multicollinearity than single-response CART by considering the interdependencies among response variables. This holistic approach ensures that the splits and feature selections are more accurate when handling correlated responses.

5.2 Application

5.2.1 Code Explanation

This code demonstrates the use of a Random Forest Regressor to perform multi-output regression, predicting two target variables: `roe` and `sustainability_score`. A single Random Forest model is trained to predict both outputs without the need for additional wrappers simultaneously.

The necessary Python libraries are imported at the beginning: `pandas` is used for data manipulation, `scikit-learn` provides tools for splitting data, training the Random Forest model, and evaluating performance, and `numpy` is used for numerical operations.

A random seed is set to ensure reproducibility across different runs. This guarantees that the data-splitting process and model behaviour remain consistent.

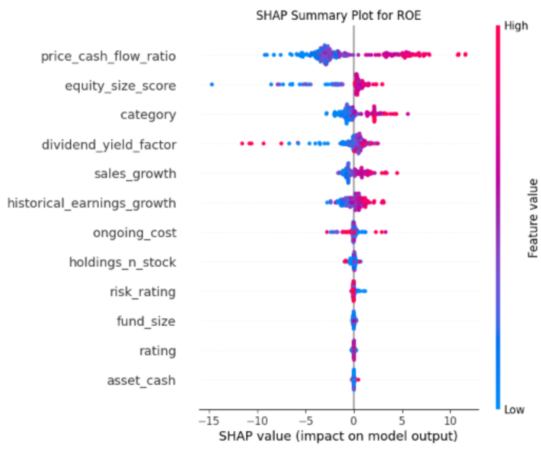
The dataset `clean.csv` is loaded into a Pandas DataFrame. Categorical columns (`rating`, `risk_rating`, `category`) are frequency encoded by mapping the proportion of each category's occurrence to its corresponding value. This transforms categorical features into numerical representations that reflect the relationship between the frequency of occurrence and the target variables. The feature matrix X and target matrix Y are defined by separating the predictor variables from the target variables. The matrix X contains all columns except `roe` and `sustainability_score`, while Y consists of these two target columns. Then, the dataset is split into training and testing sets with an 80/20 ratio to evaluate model performance on unseen data, allowing the model to generalise better and reduce over-fitting.

A Random Forest regressor uses the following hyper-parameters: `n_estimators=100`, which specifies the number of decision trees in the forest, while `random_state=42` ensures consistent results by controlling the randomness in tree construction and data bootstrapping. The model is trained on the training set by fitting it to the feature matrix X_{train} and the target matrix Y_{train}. Predictions are then made on the test set, producing two outputs: one for `roe` and one for `sustainability_score`. These predictions are extracted by selecting the respective columns from the prediction matrix. Finally, model performance is then evaluated using the ARSR, like the other models using the predicted and actual values.

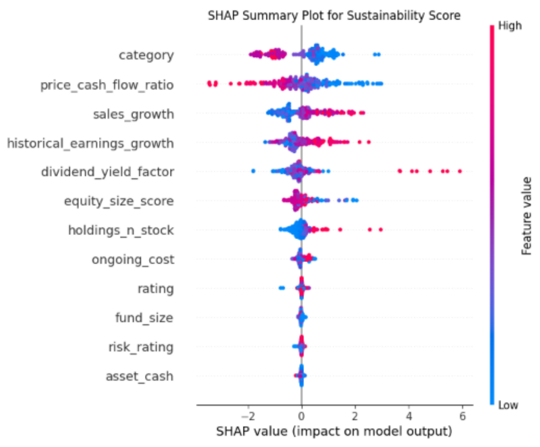
5.3 Results

The results indicate an RSR of 0.4619 for ROE and 0.0094 for Sustainability Score, with an overall ARSR of 0.2357. The model performs exceptionally well in predicting the Sustainability Score, as reflected by the low error rate of less than 1%. However, the higher RSR for ROE suggests moderate error, with predictions deviating by an average of 46.19% relative to the actual values. The ARSR overall was 0.2357, which means the random forest models this data well according to the previously set parameters on the ARSR.

To better understand the model's predictions, SHAP (SHapley Additive exPlanations) plots were generated for both target variables. SHAP values offer a comprehensive view of how each feature influences individual predictions by distributing the contribution of each feature fairly across all observations. The plots display the impact of each feature on the model output, with positive SHAP values indicating a positive contribution to the prediction. In contrast, negative values reflect a downward pull on the target variable. The colour gradient represents the magnitude of the feature value, with higher values shown in red and lower values in blue.



(a) SHAP values for Return on Equity (ROE).



(b) SHAP values for Sustainability Score.

Figure 5.2: SHAP values illustrating feature impact on ROE and Sustainability Score.

The SHAP summary plot for Return on Equity (ROE) highlights the price-to-cash flow ratio as the most influential feature driving model predictions. Higher values of this feature consistently increase predicted ROE, while lower values reduce it. Equity size score and category also exert substantial influence, with high equity scores and certain categories contributing positively to the predictions. Dividend yield factor and sales growth further enhance ROE predictions, reinforcing the model's reliance on key financial indicators.

In contrast, features such as asset cash and rating show minimal impact, with SHAP values centred around zero, indicating the limited influence on model variability. The broader distribution of SHAP values for top-ranking features underscores their dynamic role in shaping predictions across different observations.

This analysis reaffirms that price-to-cash flow ratio, equity size score, and category are critical in predicting ROE. Addressing these drivers could improve model performance and yield deeper insights into the underlying factors affecting ROE.

The SHAP summary plot for the Sustainability Score highlights category as the most dominant feature influencing model predictions. Higher category values consistently lead to increased Sustainability Score predictions, while lower values reduce them. Price-to-cash-flow ratio and sales growth also play significant roles, with high values of these features pushing predictions upward. This suggests that firms with favourable classifications, strong sales growth, and efficient cash flow management are more likely to achieve higher Sustainability Scores.

Historical earnings growth and dividend yield factor provide additional contributions to model predictions, reflecting the importance of past financial performance and returns. Although equity size score and holdings in stock influence the projections, their impact is less pronounced compared to the leading features.

Features such as fund size, risk rating, and asset cash show limited variability in their SHAP values, indicating minimal influence on the model's overall predictions. The sharp contrast in SHAP value distributions across features underlines the critical role that category and price-to-cash flow ratio play in shaping Sustainability Score outcomes.

This analysis reinforces that category, price-to-cash flow ratio, and sales growth are key determinants of Sustainability Score predictions. Focusing on these areas could improve model accuracy and yield deeper insights into the factors driving sustainability performance.

Feature importance plots provide a visual representation of the relative contribution of each feature to the overall predictions made by the Random Forest model. The x-axis represents the importance score, which reflects how much each feature contributes to reducing the model's error during the training process. Features with higher importance scores have a greater influence on the model's predictions, while those with lower scores contribute less.

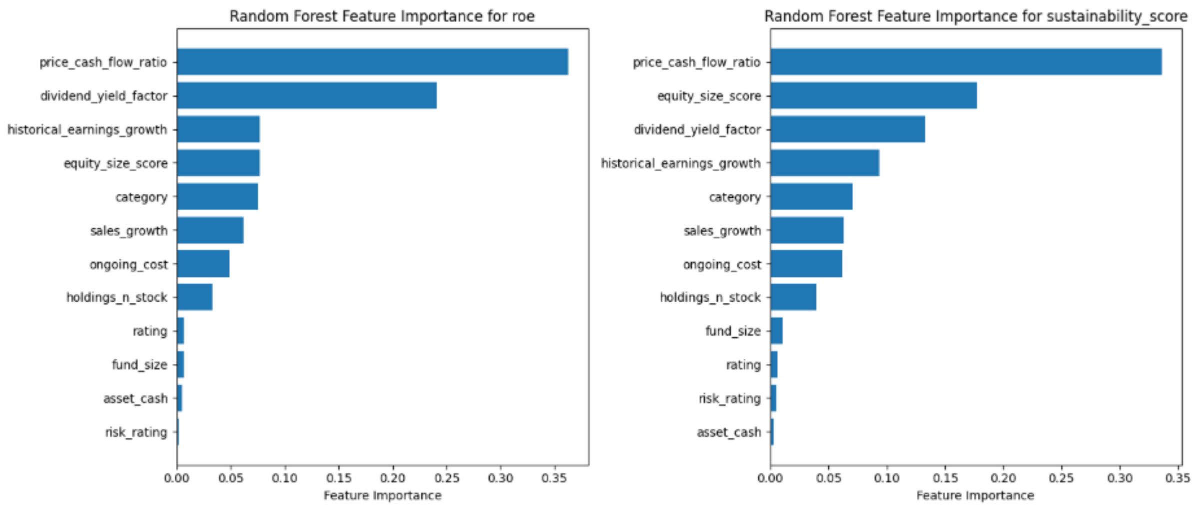


Figure 5.3: Feature importance plot for the Random Forest model.

In the case of Return on Equity (ROE), the plot highlights the price-to-cash flow ratio as the most influential feature, followed by the dividend yield factor and historical earnings growth. These features significantly shape the model's ability to predict ROE, suggesting that financial metrics related to profitability and growth are key drivers of the model's performance. Equity size score and category also play important roles, but their relative importance is lower compared to the leading features.

For the Sustainability Score, the feature importance plot similarly underscores the price-to-cash flow ratio as the dominant variable. Equity size score and dividend yield factor follow closely, indicating that firm size and return metrics are critical in determining sustainability outcomes. The contributions of features such as category and historical earnings growth further enhance the model's predictions, reflecting the complex interplay between financial performance and sustainability indicators.

Overall, these plots highlight the critical role of financial and growth-related variables in predicting both ROE and Sustainability Scores. By focusing on the most influential features, model refinements can be directed toward the areas that yield the highest impact on predictive accuracy.

While the model demonstrates strong results for the Sustainability Score, the higher error in ROE predictions points to potential areas for improvement. Further steps could include refining the feature set, tuning hyper-parameters, or experimenting with alternative regression models to enhance predictive accuracy. Addressing these areas can help reduce the discrepancy between the two targets and improve the overall reliability of the model.

In conclusion, the Random Forest model shows promising results but exhibits uneven performance across the target variables. With targeted refinements, the model's ability to predict both ROE and Sustainability Score can be enhanced, leading to more accurate and consistent predictions.

Chapter 6 XG Boost

6.1 Theory

6.1.1 Introduction

XGBoost, otherwise known as Extreme Gradient Boosting, improves upon the idea of gradient tree boosting, which involves iteratively adding trees with the goal of minimising a predefined loss function. At each iteration, the algorithm fits a new tree to the residuals of the current model, effectively correcting previous errors. Gradient tree boosting is defined mathematically as follows:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i), \quad f_t \in \mathcal{F},$$

where $\mathcal{F}$ represents the space of regression trees (also known as CART).[27] Each tree f_t predicts an additive score for the input features x_i. XGBoost improves gradient boosting by introducing an objective function which uses regularisation to address over-fitting and improve model prediction as ridge and lasso regression do. Its objective function is:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t),$$

where l represents the loss function, and $\Omega(f_t)$ is a regularisation term controlling model complexity. This construction of XGBoost allows it to fit models well and mitigate over-fitting.[27] XGBoost optimises the objective function using a second-order Taylor approximation. The loss function is approximated as:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t),$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ are the first and second-order gradients of the loss function, respectively.

To construct decision trees efficiently, XGBoost evaluates potential splits by maximising a gain function:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma,$$

where G_L and H_L are the sums of first and second-order gradients for the left child, and G_R and H_R are the corresponding sums for the right child. The "child" are the partitions of data created when a decision tree splits at a node.

6.1.2 Iterative Steps of XGBoost

So, here is out the XGBoost model works iteratively as outlined in Chen and Guestrin (2016) using the exact Greedy Algorithm for Split Finding, which occurs by iterating over each feature dimension k, ranging from 1 to m, where m is the total number of features. For each feature, potential split points are evaluated by sorting the instances based on their corresponding feature values. During this process, cumulative sums for the left child node are tracked using G_L and H_L, both initialised to zero at the beginning of each iteration. As instances are processed, their gradients and Hessians are accumulated: $G_L \leftarrow G_L + g_j$ and $H_L \leftarrow H_L + h_j$. Simultaneously, the right child node's gradients and Hessians are updated as $G_R = G - G_L$ and $H_R = H - H_L$.

The quality of each split is measured by calculating the gain, which reflects the reduction in loss (as outlined previously). This is expressed as:

$$score = \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$$

where λ is the regularisation parameter. Higher gain values indicate more effective splits, reducing the overall model error. Once all potential splits have been evaluated, the algorithm selects the split with the highest gain, ensuring that each node is partitioned at the most optimal point. If no split results in a positive gain, the node is left as a leaf. This iterative, greedy approach ensures that each decision tree grows in a way that minimises residual error, leading to more accurate models.

Regularisation in XGBoost helps control model complexity and mitigate over-fitting. It is defined as:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2,$$

where T is the number of leaves in the tree, w_j are the leaf weights, γ penalises additional leaves, and λ controls the regularisation of leaf weights.

XGBoosting also uses shrinkage and column sub-sampling to prevent over-fitting further. Shrinkage scales newly added weights by a factor of η after each step of tree boosting, and it is similar to a learning rate in stochastic optimisation in that shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model.[27] Column sub-samples also speeds up computations of the parallel algorithm.[27]

The exact Greedy Algorithm outlined earlier is ideal to apply to the XGBoost process, but in practice, an approximate Greedy Algorithm is used due to its computational efficiency. This is similar to the prior method with some slight adjustments, which are explained below, again by Chen and Guestrin (2016):

The algorithm begins by iterating over each feature dimension k, from 1 to m, where m represents the total number of features. For each feature, a set of candidate splits $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ is proposed. These splits are determined by dividing the feature values into percentiles. The percentile-based approach ensures that the split proposals are distributed evenly across the range of the feature, capturing key points that are likely to lead to significant loss reduction. This proposal process can be conducted in two ways: globally or locally. In the global approach, split points are determined once per tree and reused at each node, whereas in the local approach, new split points are generated for each node.

After proposing candidate splits, the algorithm proceeds to evaluate their quality. For each proposed split $s_{k,v}$ within the feature k, the gradients and Hessians of the data points falling into the interval between two consecutive split points are accumulated. Specifically, the gradients and Hessians are computed as:

$$\begin{aligned} G_{kv} &\leftarrow= \textstyle\sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j \\ H_{kv} &\leftarrow= \textstyle\sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j \end{aligned}$$

where G_{kv} represents the sum of gradients and H_{kv} the sum of Hessians for the interval defined by $s_{k,v}$ and $s_{k,v-1}$. This process effectively approximates the exact split finding by limiting the evaluation to a subset of potential split points.

Once the gradient and Hessian values have been computed for all proposed splits, the algorithm proceeds to identify the split that maximises the gain. This step follows the same logic as the exact greedy algorithm but is restricted to the proposed split points, significantly reducing the number of calculations required. Then the same steps are taken as in the previous section to find the maximum score only among proposed splits. While it sacrifices precision by not evaluating all possible splits, the reduction in computational overhead allows XGBoost to scale effectively to large datasets.

6.1.3 Weighted Quantile Sketches

In the approximate split-finding algorithm, as already stated, candidate split points are chosen based on percentiles of the feature values. The rank function $r_k(z)$ calculates the proportion of instances where feature k is less than z, weighted by second-order gradient statistics h.[27] The algorithm aims to find split points $s_{k1}, s_{k2}, \dots, s_{kl}$ such that the difference between the ranks of consecutive points is small, controlled by ϵ, which is an approximation factor.[27] This process ensures that the split points are well-distributed across the range of the feature values.

The second-order gradient h_i is used as a weight for each instance, meaning that data points with larger Hessians (higher curvature) carry more influence in determining the split. This weighting reflects the importance of the instance in minimising loss. Higher-weighted instances indicate regions of the feature space where the model struggles, guiding the algorithm to focus on these critical areas during split finding.

Traditional quantile sketch algorithms work for datasets where all instances have equal weights. However, no existing algorithm efficiently handles weighted data. This creates challenges in approximating split points for datasets with imbalanced or sparse data distributions. Randomised sorting or heuristic approaches have been used to address this issue, but these methods lack theoretical guarantees.[27] As a result, they may fail to identify the optimal split points, leading to suboptimal model performance. To overcome this limitation, XGBoost introduces a novel weighted quantile sketch algorithm that can handle weighted data with provable theoretical guarantees.[27] This algorithm ensures accurate split findings by accounting for the distribution of weights across the dataset, improving performance on imbalanced data.

Although the data here has no missing data due to the initial exploratory data analysis, XGBoost efficiently handles missing and sparse data through a sparsity-aware split-finding algorithm.[27] During tree construction, XGBoost learns the optimal default direction for missing values, reducing the need for data imputation. The split-finding process is defined as:

$$f(x_i) = \begin{cases} f_{\text{left}}(x_i) & \text{if } q_j \neq \text{missing} \\ f_{\text{default}}(x_i) & \text{if } q_j = \text{missing} \end{cases} \tag{6.1}$$

This allows XGBoost to scale linearly with the number of non-missing entries[27].

6.1.4 Extending to MRR

XG Boost is primarily designed for single-response regression; however, it can handle MRR by incorporating random output projections. This approach balances between independently modelling each output and jointly modelling all outputs with shared tree structures. XGBoost in MRR is very similar to single-response, as will be explained.

The objective function for MRR extends to sum over all target dimensions:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \sum_{d=1}^{D} l(y_{id}, \hat{y}_{id}) + \sum_{t=1}^{T} \Omega(f_t), \tag{6.2}$$

where y_{id} and $\hat{y}_{id}$ are the observed and predicted values for target d of instance i. This formulation ensures the model accounts for correlations and dependencies between multiple targets. XGBoost for MRR explicitly models dependencies between target variables by leveraging the residuals across all targets. During each boosting iteration, a tree is fit to the joint residuals of all targets, capturing interactions between them. The split gain is computed by summing contributions from all target dimensions:

$$\text{Gain} = \frac{1}{2}\left[\sum_{d=1}^{D}\left(\frac{G_{Ld}^2}{H_{Ld}+\lambda}+\frac{G_{Rd}^2}{H_{Rd}+\lambda}-\frac{(G_{Ld}+G_{Rd})^2}{H_{Ld}+H_{Rd}+\lambda}\right)\right]-\gamma, \tag{6.3}$$

where G_{Ld}, H_{Ld} and G_{Rd}, H_{Rd} represent the gradients and Hessians for the left and right child nodes, respectively. This is similar to the single response case, but the sum is taken over all the response variables, and at this point, the interdependencies between the target variables are considered. The parallelisation and regularisation techniques in XGBoost allow MRR to scale to large datasets efficiently. Feature importance can be assessed for each target individually or across the entire multi-response system, providing insights into the relationships between inputs and outputs.

6.2 Application

6.2.1 Code Explanation

This code demonstrates the use of XGBoost to perform multi-output regression, predicting two target variables: `roe` and `sustainability_score`. The model uses the `MultiOutputRegressor` wrapper from Scikit-Learn to enable the XGBoost model to predict multiple outputs simultaneously.

The necessary Python libraries are imported at the beginning: `pandas` are used for data manipulation, `xgboost` for modelling, `scikit-learn` provides tools for splitting data, evaluating model performance, and handling multi-output regression and finally, `numpy` is used for numerical operations. A random seed is used here, too, like for random forest.

The dataset `clean.csv` is loaded into a Pandas DataFrame and then the categorical columns (`rating`, `risk_rating`, `category`) are frequency encoded by mapping the count of each category to the corresponding feature values like before. The feature matrix X and target matrix y are then defined by dropping and selecting relevant columns. `X` contains the predictor variables, while `y` contains the two target variables to be predicted.

Next, the dataset is split into training and testing sets (an 80/20 split) to evaluate model performance on unseen data. The XGBoost model can now be trained and tested.

An XGBoost regressor is initialised with specific hyper-parameters:
`objective='reg:squarederror'`, which specifies the regression objective,

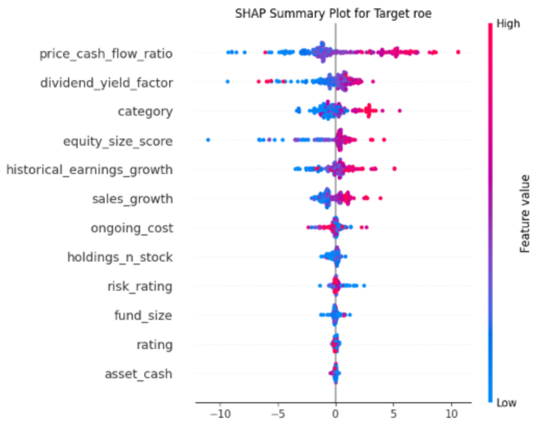
`n_estimators=100` sets the number of boosting rounds, `max_depth=6` defines the maximum depth of each tree, `learning_rate=0.1` controls the step size during boosting, and `random_state=seed` ensures reproducibility.

Since XGBoost does not automatically handle multiple target variables, `MultiOutputRegressor` is used to wrap the base model. This enables the model to predict numerous outputs by fitting one regressor per target. Then, the model is trained on the training data and predictions are made on the test set. Finally, model performance is evaluated using the Root Squared Residual (RSR) for each response variable, as done for the other models in this report.

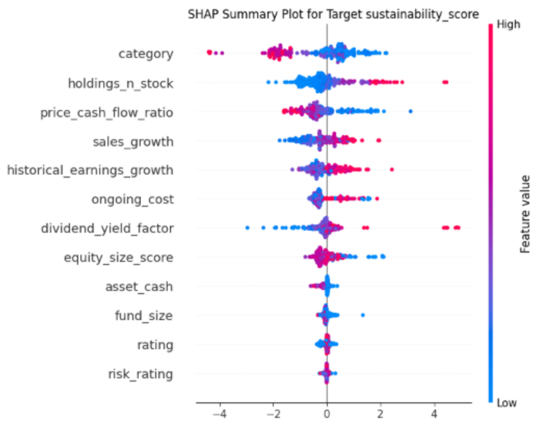
6.2.2 Results

The `roe` RSR was 0.313, suggesting that the model predicts ROE relatively well, as the RSR is quite low, implying the residuals are small compared to the variation in the data. The `sustainability_score` RSR was 0.462. This value is higher, indicating that the model has more difficulty predicting sustainability scores accurately. The residuals are larger relative to the variability in the target. An average RSR of 0.387 across the two targets suggests overall decent performance. While not perfect, this value indicates that the model captures patterns in the data but leaves room for improvement, particularly for the sustainability score.

A SHAP plot is portrayed to understand the feature contribution to predictions:



(a) SHAP values for Return on Equity (ROE).



(b) SHAP values for Sustainability Score.

Figure 6.1: SHAP values illustrating feature impact on ROE and Sustainability Score.

From Figure 6.1a, it is evident that price_cash_flow_ratio, dividend_yield_factor, and category are the most influential features affecting ROE. High values of price_cash_flow_ratio and dividend_yield_factor tend to increase ROE, as indicated by the clustering of red dots on the positive side of the x-axis. Conversely, lower values of these features generally contribute to a reduction in ROE. The feature category also plays a significant role, with variations in its values leading to substantial shifts in the prediction.

The features historical_earnings_growth and equity_size_score exhibit wide spreads of SHAP values, suggesting considerable variability in their influence across different observations. This variability indicates that the impact of these features is not uniform and may depend on complex interactions with other variables. Similarly, ongoing_cost and sales_growth show that lower values negatively affect ROE, as reflected by the concentration of blue dots on the negative side of the plot. Certain features, such as fund_size and asset_cash, display relatively narrow distributions of SHAP values. This indicates that while these features have a consistent impact across the dataset, their overall contribution to ROE is more negligible compared to more dominant features. Despite their limited influence, these features still play a role in refining the model's accuracy.

Category, holdings_n_stock, and price_cash_flow_ratio emerge as the most influential features. High values of category and holdings_n_stock positively contribute to sustainability score predictions, while low values lead to reduced predictions.

Sales_growth and historical_earnings_growth show wide spreads, suggesting significant variability in their effects across observations. Ongoing_cost and dividend_yield_factor demonstrate consistent adverse effects when their values are low. Conversely, asset_cash, fund_size, and rating display limited yet consistent influence, indicating that while their overall contributions are small, they maintain predictive significance across the dataset.

The overlapping colours along the x-axis for both plots highlight the potential for non-linear relationships and intricate feature interactions. This variability suggests that the influence of specific features may be condition-dependent, reinforcing the importance of further investigation into these interactions.

A feature plot is also displayed again to show the contribution of specific predictors to the xgboost model:

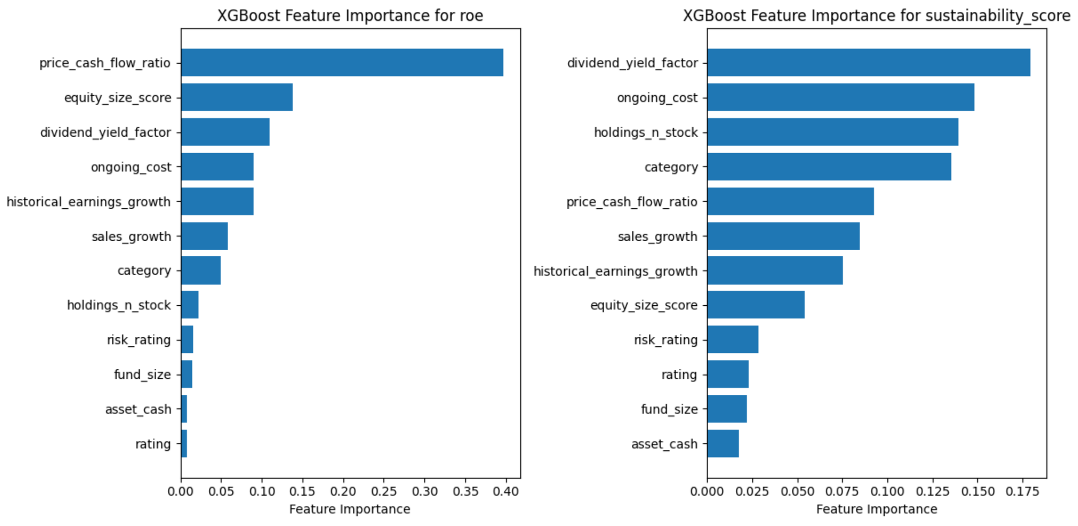


Figure 6.2: XG Boost Feature Importance

The feature importance plots for the XGBoost model, predicting `roe` and `sustainability_score`, highlight the relative influence of various predictor variables. For the `roe` model, the most dominant feature is the `price_cash_flow_ratio`, which shows a much larger importance compared to other variables. This indicates that cash flow relative to price plays a crucial role in determining return on equity. Following this, features such as `equity_size_score` and `dividend_yield_factor` also contribute significantly, though their importance is notably smaller. Additional variables, including `ongoing_cost`, `historical_earnings_growth`, and `sales_growth`, provide further predictive value, but at a reduced scale.

In contrast, the `sustainability_score` model attributes the highest importance to `dividend_yield_factor`, suggesting that dividend performance is a key determinant of sustainability. `Ongoing_cost` and `holdings_n_stock` follow closely, reinforcing the idea that ongoing expenses and stock holdings shape the sustainability profile of an entity. Other influential factors include `category` and `price_cash_flow_ratio`, although their contribution is less pronounced.

Despite differences in dominant features, both models reveal a shared set of impactful variables, including `ongoing_cost` and `sales_growth`, indicating their broad relevance across financial and sustainability outcomes. The presence of categorical factors such as `risk_rating` and `category` highlights the role of qualitative measures in predicting both target variables.

These results suggest that while certain variables are critical for specific targets, the overlap in feature importance underscores a broader relationship between financial performance and sustainability metrics. The findings can help refine decision-making processes by emphasising key drivers tailored to each objective.

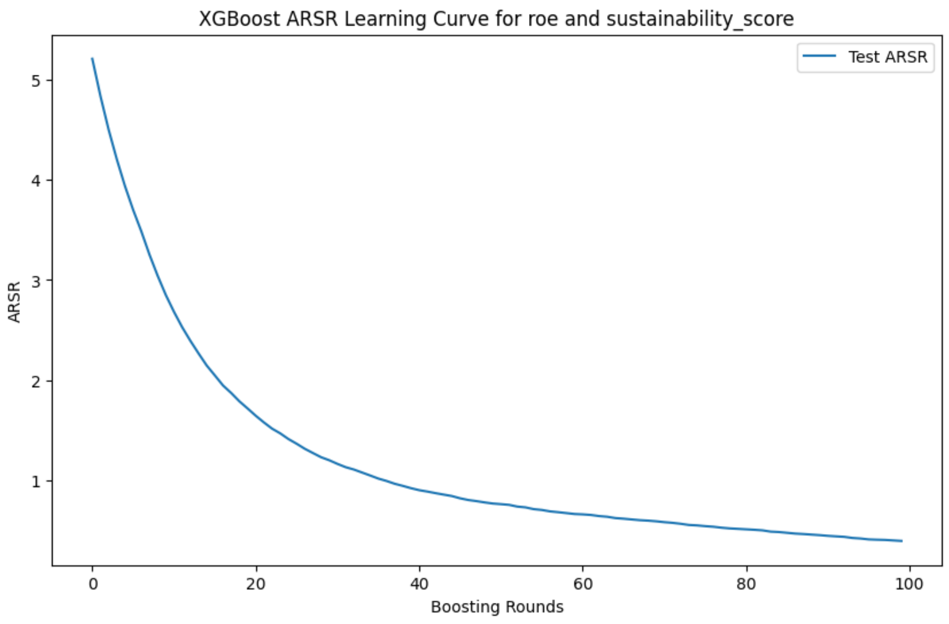


Figure 6.3: XG Boost Learning Curve

The plot illustrates the Average Root Squared Residual (ARSR) learning curve for an

XGBoost model trained to predict two target variables: `roe` and `sustainability_score`. The x-axis represents the number of boosting rounds, while the y-axis shows the ARSR values calculated for the test set.

The downward trajectory of the curve indicates a consistent reduction in prediction error as the boosting rounds progress. Initially, during the early boosting rounds, the ARSR decreases rapidly, suggesting that the model is learning key patterns in the data and making significant performance gains. This phase demonstrates the model's ability to capture essential relationships between features and the target variables efficiently.

As the number of boosting rounds increases, the rate of ARSR reduction slows, and the curve begins to flatten. This plateau suggests diminishing returns from additional boosting, implying that the model is nearing its optimal performance. Despite further boosting, the marginal improvement in ARSR becomes smaller, reflecting the model's stability and reduced error.

The absence of significant fluctuations or increases in the ARSR throughout the boosting process indicates that the model is now over-fitting the training data. This smooth convergence reflects the model's ability to generalise well to unseen data, contributing to reliable predictions for both `roe` and `sustainability_score`.

Chapter 7 Neural Networks - 8 pages aim

7.1 Theory

7.1.1 Introduction

A neural network is a model which makes decisions like the human brain. It does this by a process similar to the way biological neurons work together to identify phenomena, weigh decisions and arrive at conclusions.[28]

Nodes are a building block of a neural network. Each node is its own linear regression model, with input data, weights, a bias (or threshold), and an output.[28] Within a neural network, there are a series of layers of nodes. The first of which is called the input layer and the last is called the output layer. The layers in between are called the hidden layers.

A Feed-Forward Neural Network (FFNN), otherwise known as a multi-layer perceptron, is a type of neural network whose vertices are numbered such that all connections go from a neuron (vertex) to one with a higher number. In other words, the information that enters all the nodes in the layers except the input is the sum of the information from those previous nodes. As information passes through the nod,e it is also changed by the weights of their respective prior corresponding nodes. The weights transform this information using the activation function.
Each node in the network except the input layer is defined as:

$$h_k^{(n)}(x) = \sigma_n \left(w_{k,0}^{(n)} + \sum_j w_{k,j}^{(n)} o_j^{(n-1)}(x) \right)$$

where $h_k^{(n)}$ is the linear combination of the k-th neuron in layer n, σ_n is the activation function at layer n, $w_{k,0}^{(n)}$ is the bias term for the k-th neuron in layer n, $w_{k,j}^{(n)}$ is the weight from neuron j in layer $n-1$ to neuron k in layer n, $o_j^{(n-1)}(x)$ is the output of neuron j in the previous layer $n-1$, and $o_j^{(0)}(x) = x_j$ are the input features for the input layer $n = 1$.

From what has been described, neural networks can capture linear models. How can they be extended to capture potential non-linear relationships though? This is through the choice of activation function. Activation functions are non-increasing and are often sigmoids or threshold functions. Some examples of them include: the threshold sigmoid $\sigma(\alpha) = 1\ (\alpha > 0)$, the logistic sigmoid: $\sigma(\alpha) = (1 + \exp(-\alpha))^{-1}$ and the rectified linear

unit, RELU: $\mathrm{RELU}(\alpha) = \max(\alpha, 0)$.
Here is a graphic for a FFNN:

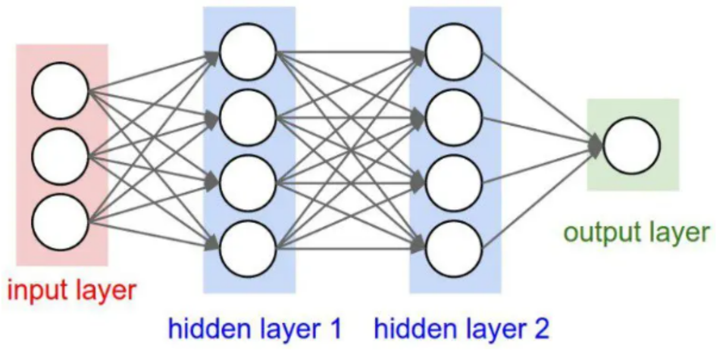


Figure 7.1: Neural Network Representation[29]

7.1.2 Extension to MRR

Neural networks can model correlations between multiple outputs by introducing shared hidden layers. They capture correlations between these resonses, by using adaptive hidden units that were shared between the outputs.[30] The shared hidden layers act as a common latent representation of the inputs, which is then transformed by weights.

Each neuron in the output layer corresponds to one response variable. The weights connecting the hidden layer to the output neurons allow the FFNN also to learn output-specific mappings. These outputs are now coupled through a weight matrix, $W(x)$.

When training neural networks for multiple response regression, the choice of the loss function is critical in ensuring that all outputs are effectively optimised. A common approach is to compute the loss for each output individually and then aggregate the errors. This can be done using the mean squared error (MSE) or a weighted loss approach to emphasise specific outputs. The general form of the multi-output loss is:

$$L(y, \hat{y}) = \frac{1}{p} \sum_{i=1}^{p} (y_i - \hat{y}_i)^2$$

where p represents the number of outputs, y_i is the actual value for the i-th output, and $\hat{y}_i$ is the predicted value. For cases where outputs have different magnitudes or importance, a weighted loss can be applied:

$$L(y, \hat{y}) = \frac{1}{p} \sum_{i=1}^{p} w_i (y_i - \hat{y}_i)^2$$

where w_i is the weight assigned to the i-th output. This approach ensures that higher-priority outputs receive greater attention during training.

Training neural networks with multiple outputs introduce unique challenges, including gradient imbalance, where outputs with larger gradients dominate the optimisation process. This can lead to poor performance for smaller-gradient outputs. Several techniques can mitigate this issue. The first of which is Gradient Clipping, which limits the magnitude of gradients during back-propagation to prevent any single output from dominating the learning process. Output-Specific Learning Rates for each output can also ensure balanced optimisation. Finally, batch normalisation, which involves normalising the activations in each layer, ensuring that gradients remain stable and uniform across all outputs will solve this.

The choice of activation function in the output layer depends on the type of regression problem. For continuous outputs, linear activation functions are typically used:

$$o_j^{(n)}(x) = w_{k,j}^{(n)} o_j^{(n-1)}(x) + b_k^{(n)}$$

However, for bounded outputs (e.g., probabilities), sigmoid or softmax activation functions can be applied. For unbounded outputs, the ReLU (Rectified Linear Unit) activation is often employed due to its simplicity and effectiveness in learning non-linear patterns.

Regularisation helps prevent over-fitting and enhances the generalisability of neural networks. This is especially important in multi-output settings where the network may over-fit to one output at the expense of others. Regularisation methods include: L2 Regularisation, which penalises large weights in the output layer, effectively shrinking them during training:

$$R(W) = \lambda \sum_{i,j} W_{i,j}^2$$

where λ is the regularisation strength.

Dropout is another regularisation method, and it randomly deactivates neurons during training to introduce n. This forces the network to learn more robust features. Dropout can also be applied selectively to the output layer.

Finally, there is a multi-output dropout, which extends the dropout technique by deactivating neurons in the output layer, ensuring that no single output becomes over-fitted.

Multi-output neural networks can provide insight into the relationships between outputs and features. Several methods enhance the interpretability of these models: SHAP (SHapley Additive exPlanations)is a feature attribution method that explains how each feature contributes to different outputs. Saliency Maps are visual representations highlighting which input features most influence the outputs. Finally, there are Attention Mechanisms, which are introduced in advanced architectures, attention mechanisms weigh certain features more heavily depending on their relevance to specific outputs.

7.2 Application

7.2.1 Code Explanation

The code implements a neural network model for regression tasks using PyTorch. Optuna is used for hyperparameter tuning, and Mixup data augmentation is applied to improve generalization. The project is structured to preprocess data, define and train the neural network, and optimize hyperparameters to achieve the best results.

The first step involves data preprocessing. The dataset is loaded from a CSV file called `clean.csv`. This dataset contains twelve predictors and two response variables, labeled as `roe` and `sustainability_score`. Categorical columns, including "category", "rating", and "risk_rating", are frequency encoded to transform them into numerical values. The data is then split into features, denoted by `X`, and target values, denoted by `Y`. These values are standardized using the `StandardScaler` from scikit-learn. The standardized data is further divided into training and testing sets, ensuring that the testing set constitutes twenty percent of the total data. A random seed is set to maintain consistency in the data split.

The neural network is defined by the class `NeuralNetwork`, which inherits from `torch.nn.Module`. The architecture of this neural network consists of two hidden layers. Each hidden layer is followed by batch normalization and dropout to prevent overfitting and accelerate convergence. ReLU activation functions are applied to introduce non-linearity. The forward pass method computes predictions by sequentially passing the input through each layer of the model.

The training process is executed using the Adam optimizer with a learning rate of 0.0003. Mean squared error (MSE) is selected as the loss function, which calculates the difference between predicted and actual values. To improve convergence, a learning rate scheduler is applied. Specifically, the `ReduceLROnPlateau` scheduler reduces the learning rate by a factor of 0.5 if the validation loss does not improve for seven consecutive epochs. Early stopping is implemented to halt the training process if there is no improvement for fifteen epochs, ensuring the model does not overfit.

During training, Mixup data augmentation is applied to generate new samples by interpolating between existing data points. This is done by randomly shuffling the data and blending pairs of samples according to a coefficient drawn from a Beta distribution. Mixup helps the model generalize better by creating smoother decision boundaries.

Hyperparameter tuning is conducted using Optuna, an optimization framework that automates the search for the best model configuration. The objective function defines a neural network and trains it using a range of hyperparameters, including the number of hidden units, dropout rate, learning rate, and batch size. After training, the root squared residual (RSR) is calculated for each response variable. The average RSR across all responses is used as the objective to minimize. The trial that results in the lowest average RSR is selected as the best-performing model.

Following hyperparameter tuning, the model is retrained using the optimal parameters. Predictions are made on the test set, and the RSR values for `roe` and `sustainability_score` are computed to evaluate the model's performance. These results are printed at the end of the execution to summarize the model's effectiveness.

To ensure reproducibility, random seeds are fixed across PyTorch, NumPy, and the Python random module. Additionally, CUDA deterministic options are enabled to ensure consistent results even when using GPU acceleration. This step guarantees that the same model can be retrained multiple times with identical results.

7.2.2 Results

The results indicate the effectiveness of the neural network in predicting the target variables, `roe` and `sustainability_score`. After performing hyperparameter tuning using Optuna, the best-performing model achieved an average root squared residual (RSR) of approximately 0.7135 across the two response variables. This value reflects the model's ability to generalize well when making predictions on unseen data.

For the individual responses, the RSR for `roe` was 0.5940, while the RSR for `sustainability_score` was 0.8311. A lower RSR indicates that the model's predictions closely align with the true values, signifying stronger performance. The discrepancy between the two RSR values suggests that the model exhibits slightly better performance in predicting `roe` than `sustainability_score`. This may imply that the features provided in the dataset have a stronger correlation with `roe`.

The hyperparameter tuning process involved 280 trials, where Optuna explored different configurations of hyperparameters, including the number of hidden units in each layer, the dropout rate, the learning rate, and the batch size. The trial that produced the best results utilized the following combination of hyperparameters:

- Hidden Units (Layer 1): 82
- Hidden Units (Layer 2): 32
- Dropout Rate: 0.251
- Learning Rate: 0.00057
- Batch Size: 64

This combination suggests that a moderately sized network, coupled with a higher dropout rate, effectively prevents overfitting while allowing the model to converge efficiently. The learning rate of 0.00057 implies that the model benefits from a slower convergence process, ensuring that the optimization proceeds carefully and avoids overshooting the minima.

The application of Mixup data augmentation during training played a vital role in enhancing the generalization capabilities of the model. By blending pairs of samples, Mixup encourages the model to learn smoother decision boundaries and perform well on interpolated data points. This is reflected in the relatively low RSR values, demonstrating the model's capacity to generalize beyond the training data.

Early stopping was instrumental in preventing overfitting by halting the training process after 15 epochs without improvement in validation loss. This mechanism ensured that the model did not continue training unnecessarily, preserving its ability to generalize. Additionally, the learning rate scheduler reduced the learning rate by a factor of 0.5 when the validation loss plateaued for seven epochs. This adjustment allowed the model to escape shallow local minima, further improving performance.

The progression of training and validation losses over 500 epochs indicates a steady decline. The minimal divergence between the training and validation losses is an encouraging sign that the model did not overfit to the training data. This steady alignment between losses reinforces the model's robustness and its capacity to maintain performance on unseen samples.

While the overall performance is promising, the higher RSR value for `sustainability_score` suggests that the model encountered greater difficulty in capturing the variance associated with this target variable. This discrepancy could indicate that additional features or a more complex neural network architecture might be necessary to improve predictive accuracy for this particular response.

In summary, the neural network demonstrates strong predictive capabilities, as evidenced by the RSR values, which are consistently below 1. The hyperparameter tuning process significantly enhanced performance, as demonstrated by the best trial results. The integration of Mixup and early stopping further contributed to the model's ability to generalize while mitigating the risk of overfitting. Although the model performs better on `roe` than on `sustainability_score`, this points to areas where further experimentation and feature engineering could lead to improvements in future iterations.

Chapter 8 Conclusion - 5 pages aim

8.1 Summary of Findings and Model Comparisons

Here is a table summarising the key figures generated from this report from the multiple response regression model methods:

Table 8.1: Model Fit Comparison on Equity Fund Dataset

Model	ARSR	Model Fit
Chapter 3: Multiple Linear Regression		
Standard Multiple Linear Regression	0.2	Excellent
Forward Stepwise Selection	0.6	Good
Backward Elimination	0.7	Satisfactory
Bidirectional Stepwise Selection	0.8	Unsatisfactory
Chapter 4: Shrinkage Methods		
Ridge Regression	0.83	Unsatisfactory
Lasso Regression	0.81	Unsatisfactory
Elastic Net	0.82	Unsatisfactory
Group Lasso	0.82	Unsatisfactory
Chapter 5: Random Forests		
Decision Tree	0.78	Unsatisfactory
Random Forest	0.92	Unsatisfactory
Multivariate Regression Tree (MRT)	0.88	Unsatisfactory
Chapter 6: XGBoost		
XGBoost	0.89	Unsatisfactory
Chapter 7: Neural Networks		
Neural Network	0.713	Unsatisfactory

The model fit was set out based on criteria defined in the exploratory data analysis.

8.2 Challenges and Limitations

8.3 Future Research Directions

Bibliography

1. Stockopedia. Risk rating, 2024. Accessed: 27 November 2024.

2. AMG Funds LLC. Equity style analysis: Growth vs. value, 2023. Accessed: 27 November 2024.

3. Adam Hayes. Growth stock: What it is, examples, vs. value stock, December 2023. Updated 11 December 2023. Reviewed by Chip Stapleton. Fact checked by Kirsten Rohrs Schmitt. Accessed: 27 November 2024.

4. Hitul Adatiya. Eda on european mf dataset, 2022. Version 2 of 4. Accessed: 27 November 2024. Licensed under Apache 2.0 open source.

5. Stockopedia. Cash to assets ratio, 2024. Accessed: 27 November 2024. The Cash to Assets Ratio measures the proportion of a company's assets in cash and short-term investments, often used for analyzing funds and investment trusts.

6. Chris B. Murphy. Operating costs definition: Formula, types, and real-world examples, 2024. Updated 28 June 2024. Reviewed by David Kindness. Fact checked by Amanda Jackson. Accessed: 27 November 2024. Part of the series: The Evolution of Accounting and its Terminology.

7. Will Kenton. Kurtosis: Definition, types, and importance, 2024. Updated 31 July 2024. Reviewed by Gordon Scott. Fact checked by Timothy Li. Accessed: 27 November 2024.

8. Shangzhi Hong and Henry S Lynn. Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction. *BMC medical research methodology*, 20:1–12, 2020.

9. Cátia M. Salgado, Carlos Azevedo, Hugo Proença, and Susana M. Vieira. Missing data. In Ross C. Mitchell, editor, *Secondary Analysis of Electronic Health Records*, pages 143–162. Springer, Cham, 2016. Open Access chapter distributed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License.

10. Mike Nguyen. *A Guide on Data Analysis*. bookdown, 2024. Last updated on 2024-09-11.

11. Richard Johnson and Dean Wichern. Multivariate linear regression models: Section 7.7. In *Applied Multivariate Statistical Analysis: Pearson New International Edition*, pages 360–429. Pearson Education, Limited, 6th edition, 2013. Accessed: 27 November 2024.

12. Wonyul Lee and Yufeng Liu. Simultaneous multiple response regression and inverse covariance matrix estimation via penalized gaussian maximum likelihood. *Journal of multivariate analysis*, 111:241–255, 2012.

13. Adam Hayes. Multiple linear regression (mlr) definition, formula, and example, July 2024. Updated article reviewed by David Kindness and fact-checked by Timothy Li.

14. CFA Institute. Basics of multiple regression and underlying assumptions, 2024. Refresher Reading for CFA Program Level II, Quantitative Methods.

15. Asokan Mulayath Variyath and Anita Brobbey. Variable selection in multivariate multiple regression. *Plos one*, 15(7):e0236067, 2020.

16. Jo Jackson. Multivariate techniques: Advantages and disadvantages. `https://classroom.synonym.com/multivariate-techniques-advantages-and-disadvantages-123456.html`, 2018. Updated September 05, 2018. Accessed December 06, 2024.

17. Smith Gary. Step from stepwise. *Journal of Big Data*, 5(1):1–12, 2018.

18. Daniel N Moriasi, Jeffrey G Arnold, Michael W Van Liew, Ronald L Bingner, R Daren Harmel, and Tamie L Veith. Model evaluation guidelines for systematic quantification of accuracy in watershed simulations. *Transactions of the ASABE*, 50(3):885–900, 2007.

19. Kristin L. Sainani. Multivariate regression: The pitfalls of automated variable selection. *PM&R*, 5(9):791–794, 2013.

20. Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 03 2005.

21. CFI Team. Elastic net: A regression method that performs variable selection and regularization simultaneously. `https://corporatefinanceinstitute.com/resources/data-science/elastic-net/`, 2024. Accessed December 12, 2024.

22. Timo Similä and Jarkko Tikka. Input selection and shrinkage in multiresponse linear regression. *Computational Statistics & Data Analysis*, 52(1):406–422, 2007.

23. Frederik Questier, Raf Put, Danny Coomans, Beata Walczak, and Yvan Vander Heyden. The use of cart and multivariate regression trees for supervised and unsupervised feature selection. *Chemometrics and Intelligent Laboratory Systems*, 76(1):45–54, 2005.

24. Elsevier Inc. *Data Science Process*. Elsevier, Amsterdam, Netherlands, 2019. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00002-2.

25. Quebec Centre for Biodiversity Science. Qcbs r workshop series: Workshop 10: Advanced multivariate analyses in r, 2023. Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Licenses all content.

26. G. De'ath. Multivariate regression trees: A new technique for modeling species-environment relationships. *Ecological Society of America*, 88:2783–2792, 2002.

27. Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

28. IBM Team. What is a neural network? `https://www.ibm.com/topics/neural-networks`, 2024. Accessed December 18, 2024.

29. Adrian Rosebrock. A simple neural network with python and keras. `https://pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras/`, September 2016. Accessed December 18, 2024.

30. Andrew Gordon Wilson, David A Knowles, and Zoubin Ghahramani. Gaussian process regression networks. *arXiv preprint arXiv:1110.4411*, 2011.

Chapter A Appendices

A.1 Additional Data Visualisations

These are additional plots produced in the data that indicate underlying patterns within it or which features contribute to a select model.

Stretch: Power BI and Tableau insights.

A.2 Python/R Code Implementation

This section contains all the code used to evaluate how well each of these models fit the equity fund dataset using the ARSR.