Group-based Term Paper

FIE453 – Big Data with Applications to Finance

Candidates: 27, 37, 40

Contents

Term Paper - Big Data with Application to Finance

Generative AI Statement

The use of generative AI in this project is limited to troubleshooting R code and resolve errors we couldn't fully understand.

1. Introduction

Predicting the future to make good investments is generally known to be difficult. Predicting change in stock prices is challenging as the efficient market theory ensures that prices adjust almost instantaneously to new information. Net income is used to calculate earnings per share (EPS) and can, therefore, influence the stock price. However, many other factors affect the stock price, e.g., investor sentiment, macroeconomic indicators, risk, etc. (Pohl, Lecture: Crash Course in Financial Markets, 2024).

Prior research by Ou and Penman (1989) demonstrates that net income and its changes influence stock prices and can also serve as robust predictors of stock returns. Their study shows that financial indicators derived from net income, alongside other financial statement attributes, provide valuable insights into future stock performance. By analyzing patterns in net income changes, investors can uncover opportunities for designing profitable investment strategies.

Building on this foundation, we aim to explore whether such predictive relationships can be further enhanced using modern machine-learning techniques. In this paper, we use different supervised and unsupervised machine learning techniques to predict the change in net income. This leads us to the following research question;

> *"Can we predict whether a company's net income will increase or decrease, compared to the previous period, using financial indicators and macroeconomic factors?"*

The objectives of this study are to build predictive models, evaluate their performance, identify the most influential predictors of net income changes, and finally explore if the models we train can result in a profitable investing strategy.

To address the research question, we utilize different machine-learning methods such as the generalization technique Lasso, Decision tree, Random Forest, XGBoost, and Artificial Neural Networks. Inspired by prior financial statement analyses, such as the approach outlined by Ou and Penman (1989), we integrate lagged financial and macroeconomic

indicators as predictive variables to align with our predictions of the future change in net income. To provide the time-series aspect of the data, we used time-slice validation. A feature selection using Lasso regularization was used to ensure the models focused on the most important predictors.

2. Data

2.1 Compustat

Compustat is the main dataset in this research, providing detailed company-specific data, including balance sheets, cash flow- and income statements. The dataset is made by Standard and Poor's (S&P) and was established in 1950, making it one of the oldest datasets of its kind. Compustat is also one of the most used company-specific datasets among researchers and investors, making it a suitable and comprehensive dataset for this research. The data is structured both yearly and quarterly and is gathered from financial reports from thousands of companies in the US and the rest of the world. (Hayes, 2022).

For this research, the subset of Compustat utilized includes 239,642 observations across 375 variables, focusing on the period from 2010 through 2019. In 2020, the outbreak of the global COVID-19 pandemic occurred. The pandemic had a major impact on the economy and made many companies struggle financially. To avoid the models to train and test on extreme and unusual situations caused by the pandemic, the cutoff was set at the end of 2019 (World Bank Group, 2022).

2.2 Macro data

As macroeconomic factors are highly correlated with financial distress and stock returns for companies (Tinoco & Wilson, 2013) (Tangjitprom, 2012), we hypothesize that including such variables in the machine-learning model should yield higher accuracy in predicting whether net income will increase or decrease in the next period.

The macroeconomic data in the United States for the relevant period (2010-2019) were collected from the National Income and Product Accounts tables issued by the Bureau of Economic Analysis (BEA Interactive Data Application)[1], the Data Viewer tool issued by the Bureau of Labor Statistics (BLS) (BLS Data Viewer)[2], and the Federal Reserve via the

[1] Tables 1.1.1, 2.3.1, 3.9.1, 4.2.1, 5.3.1 (Quarterly data, percentage change from one quarter to next)

[2] Stat retrieved: "All employees, thousands, total nonfarm, seasonally adjusted"

Federal Reserve Bank of St. Louis (Market Yield on U.S. Treasury Securities at 10-Year Constant Maturity, Quoted on an Investment Basis (DGS10) | FRED | St. Louis Fed)[3].

Although one may argue that the best approach should be only to include commonly regarded leading indicators as they tend to predict future economic cycles better, the impact of macroeconomic variables is complicated and sometimes acts as both (Lingard, Hallowell, Salas, & Pirzadeh, 2017). Thus, we decided to use a broad set of macro variables and let the machine-learning techniques decide whether they are helpful for predicting future changes in net income or not.

Although one may argue that the best approach should be only to include commonly regarded leading indicators as they tend to predict future economic cycles better, the impact of macroeconomic variables is complicated and sometimes acts as both (Lingard, Hallowell, Salas, & Pirzadeh, 2017). Thus, we decided to use a broad set of macro variables and let the machine-learning techniques decide whether they are helpful for predicting future changes in net income or not.

2.3 Beta

As a final variable, company betas were acquired from Wharton Research Data Services (WRDS) using their Backtester tool. WRDS offers several asset pricing signals derived from datasets such as Compustat and CRSP (The Wharton School, 2018).

Beta is a measure of a stock's price fluctuation compared to the market's fluctuation. A positive beta implies that the stock, on average, is positively correlated with the return of the market portfolio, and vice versa. If the beta is higher than 1 or less than -1, the stock’s price will, on average, fluctuate more than the market, reflecting a volatile stock (The Economic Times, 2024).

We included betas to explore the relationship between company data from Compustat and macroeconomic indicators. By integrating financial data, macroeconomic indicators, and beta values, we aim to investigate whether both strained and stable economic environments, combined with varying levels of volatility, can help predict changes in net income.

[3] Stat retrieved: “Market Yield on U.S. Treasury Securities at 10-Year Constant Maturity, Quoted on an Investment Basis (DGS10)”

2.4 Data Cleaning and Preprocessing

Merging and Cleaning

To prepare the data for analysis, our initial step was to merge Compustat with the macroeconomic data. The values were merged on the calendar year and quarter column, so each company got the same macroeconomic values. After merging, the first removal of NA-values was made. All the columns that contained a higher percentage of NA-values than the overall median of NA-values was removed, yielding a total of 188 variables removed.

The next step in the data-cleaning process was introducing the dependent variable *net_income_change.* It was created from *niq*, representing a company's quarterly net income. Every company was calculated individually, and if the net income from the previous observation were lower than the given observation, *net_income_change* would yield a value of 1. If there were a reduction in net income, the value would be 0. We experimented with including an ordinal variable for the net income change, where -1 represented a decrease, 0 represented movement less than 2%, and 1 represented an increase in net income. However, the number of observations for stable companies (0) was low, and the harder interpretability of the ordinal variable made the choice of a binary variable suitable for this research.

In this step, lag variables were applied to macroeconomic and company-specific data to align features with their respective prediction dates. The target variable, *net_income_change*, remained untouched to ensure the model could make accurate predictions for future periods without introducing bias or data leakage. Missing values resulting from the lagging process in the first period for each company were removed, and the columns included non-lagged variables. Next, columns with more than 5% missing values were dropped (Complete Dissertation by STATISTICS SOLUTIONS, 2024), and companies with any remaining missing values were excluded, leaving us with 67 063 observations. This procedure ensured a complete panel dataset without gaps, preserving data quality and the time-series structure necessary for future predictions.

Inclusion of Beta

We added the beta value for each company by merging the data on company name, year, and quarter. Only the most recent beta value within each quarter was included. To ensure consistency with the prediction of *net_income_change,* as previously discussed for the

other variables, the beta variable was adjusted by creating a lagged version. The late merging was due to a later imputation of betas, making the data easier to handle when the only missing values were affiliated with beta.

To know what to do with the missing values of beta, we first analyzed whether there were any patterns of what year the data was from. However, it showed us an even distribution, and we could not notice any clear patterns. Further, we analyzed whether any specific companies had many missing values. The approach was to calculate the percentage of missing values for each company and remove the companies with more than 50% missing values. The number of companies exceeding this threshold ended up at 1193, all of which were removed from our dataset. The thought behind removing these companies was that they would create problems when imputing new values. Before the imputations were made, the data was divided into train and test splits to avoid leaking out-of-sample data from the "future" into the training data (Pohl, Lecture: Supervised Learning Pipeline, 2024).

Train, Validation and Test Split

Further, the dataset is split into a training set covering the period up to 2018Q2, and a test set covering the remaining quarters through the end of 2019. This split preserves the time-series aspect of the data, ensuring that the model is trained on past data and evaluated on out-of-sample, future data.

2010Q3 -> 2018Q1	2018Q2 -> 2019Q4
Train / Validation	Test

For most classification methods, cross-validation is used to tune hyperparameters and prevent overfitting. Cross-validation is generally preferred as it provides a more robust evaluation of model performance across multiple subsets of the training data (GeeksForGeeks, 2024). Initially, a rolling window approach was considered, but differences between the quarters caused macroeconomic data to introduce bias into the out-of-sample validation sets (Pohl, Lecture: Validation and Testing, 2024). This occurred because the quarters did not have an equal number of observations, leading to macroeconomic data identical for each company within a given quarter being leaked into the training set. The first three quarters, as shown in table 1, highlight the differences in the number of observations. A single validation set was used to address this issue and ensure greater control over the split. The validation set spans from 2016Q2 to 2018Q2 and does not overlap with the training set, ensuring a practical and consistent approach to hyperparameter tuning.

Date	Observations
2010Q3	1694
2010Q4	1697
2011Q1	1684

Table 1 Observations Per Quarter in Validation Set

Imputation and Standardization

After splitting the data into train and test sets, missing beta values were imputed. To avoid leaking out-of-sample information into the training set, the mean beta for each company was calculated using only the training data. These imputed values were then applied to both the train and test sets (Pohl, Lecture: Supervised Learning Pipeline, 2024). However, some beta values could not be imputed in either set. This issue may arise when a company has missing beta values in the train set but valid beta values in the test set. Another possibility is that these companies were established during the period represented by the test set, leaving no historical data for imputing their betas. To handle the missing beta values, their respective companies were removed from both sets to maintain complete panel data.

The next step in the pipeline to prepare the dataset for analysis is to perform standardization using the “caret” package and the preprocess function. The method applied was center and scale, which involves subtracting the mean and dividing by the standard deviation for each variable (Pohl, Lecture: Supervised Learning Pipeline, 2024). We excluded variables such as the dependent variable net_income_change, non-numeric, and other variables that did not need scaling, like macroeconomic data and beta. Standardization ensures that features are scaled appropriately, which is critical for models like Artificial Neural Networks (ANNs) that are sensitive to the magnitude of input variables. Although standardization is not strictly necessary for tree-based models like Random Forests or Decision Trees (Filho, 2023), it ensures consistency across the models and allows for later comparisons. In the end, the dataset used for analysis contained 57 489 observations of 267 variables, divided into a train set with 47 892 observations and a test set with 9597 observations.

2.5 Feature Selection with Lasso

To improve the performance and efficiency of our predictive models, we implemented Lasso regression as a feature selection technique. This helped us reduce the number of columns and, thus, the complexity of the data set before the subsequent machine-learning methods used in our task without compromising too much on the solidity and explanatory power.

As one could expect with the broad set of macro variables included, the majority were discarded, resulting in only 26 explanatory variables. Eighteen of these were macro variables, seven were company-specific variables from Compustat, and the last one was the beta downloaded from WRDS. This cleaner, more focused dataset was then ready for further analysis, facilitating the use of more complex parameters in the other machine learning models (for example, the number of trees in the random forest model).

3. Methodology and analysis

3.1 Dumb Model

In our initial approach to predicting net income change, we developed a baseline "dumb" model that serves as a fundamental benchmark for future predictive efforts. This model employs the most straightforward possible prediction strategy: always predict net income change to equal one. By consistently predicting the majority class, the model achieves an accuracy of 51.4%, demonstrating minimal predictive power as it barely surpasses the expected performance of random guessing (50%). The training data shows a slight imbalance, with 23,149 instances of no increase and 24,743 instances where there was a positive change in net income in the training set. Similarly, the validation set reflects this pattern, with 4,664 no-increase instances and 4,933 increase instances.

The dumb model achieved an Area Under the Curve (AUC) of 0.5. The AUC is a performance measurement for classification problems and tells us how capable the model is of distinguishing between classes. The AUC ranges from 0-1 where 1 is a perfect model. (Narkhede, 2018). This will serve as a benchmark for evaluating the performance of more advanced machine learning models. This simple approach gives us a starting point for analysis and highlights the need for more advanced models to improve prediction accuracy.

Metric	Value
Accuracy	51.4%
AUC	0.5

Table 2 Results Dumb Model

3.2 Lasso Regularization

Regularization in the context of machine learning refers to techniques used to balance the problem of overfitting (too much variance of the model) and underfitting (too much bias of the model) by trading “a marginal decrease in training accuracy for an increase in generalizability” (Murel, 2023). Generally, adding variables to the model lowers bias and increases variance (Pohl, Lecture: Regularization, 2024), meaning a model with high predictive power relies on finding the sweet spot in the number of explanatory variables. In our case, regularization also helps sort out potential problems of multicollinearity that would be present with many similar macro- and company-specific variables included.

Finding the right balance between overfitting and underfitting is crucial in machine learning. This is because a model with too many parameters captures the underlying noise of the data, while a model with too few parameters won’t pick up the critical patterns in the data needed to predict accurately, both resulting in models with bad out-of-sample prediction accuracy.

The three main regularization techniques used in machine learning are Lasso Regularization (L1), Ridge Regularization (L2), and Elastic Net (a combination of L1 and L2). How these methods work is that they employ a penalty function to ordinary least squares regression (OLS), penalizing each additional variable added to the model which forces unimportant variables (or redundant in the case of multicollinearity) towards 0 (all the way to 0 in Lasso regression). The lambda parameter acts as a weight on how much the model should penalize added features, meaning a lambda of 0 would result in a standard OLS regression. Elastic net incorporates an alpha parameter acting as a relative weight of how much the Ridge and Lasso penalties should weigh, where an alpha of 0 means only Ridge and an alpha of 1 means only Lasso.

Lasso is generally a very efficient method when there are more explanatory variables than observations in the dataset (Farbahari, Dehesh, & Gozashti, 2019), but is known to struggle handling multicollinearity effectively (Kamkar, Gupta, Phung, & Venkatesh, 2015). Saleh and Shalabh point out that Ridge on the other hand performs better in this situation (Saleh & Shalabh, 2014).

Model Description

The dataset was first split into a train and a test set as described in the data section. We implemented the timeslice validation method from the caret package to maintain the time-series aspect. When implementing the validation function, the training window was set to 35 963 observations, representing the observations until the second quarter of 2016. Simultaneously, the horizon parameter was set to 11 929 observations, representing the remaining observations of the initial train set spanning from Q2 2016 to Q2 2018. In practice, this split implies that one validation set was used. As explained in the data section, the implementation of cross-validation would result in leaking out-of-sample data, thus justifying the use of one validation set.

Our initial approach involved using Elastic Net, as the dataset was very complex with many variables and likely contained a degree of multicollinearity, making the selection between Lasso and Ridge unclear. Through hyperparameter tuning, we found, contrary to our beliefs about possible multicollinearity, that the best alpha value for our data was 1, turning the model into a regular Lasso.

Using the glmnet method within the caret framework, we experimented with a wide range of lambdas, where the model, after some trial and error, found the optimal lambda to be 0.05. The evaluation metric for model tuning was the area under the ROC curve (AUC), and the package we used for this was the caret package in R.

Finally, we tested the predictions using the Lasso model and its coefficients on the external test dataset to check the validity and accuracy. For assessing the classification performance, we generated a confusion matrix and calculated the AUC score. Our strong results gave us confidence in the predicting power of Lasso's 26 selected explanatory variables.

3.3 Classification Trees

Classification trees offer a powerful approach to predictive modeling. Unlike regression techniques, classification trees predict that each observation belongs to the most commonly occurring class of training observations within its specific region (James, Witten, Hastie, & Tibshirani, 2021). This method is particularly valuable when seeking an interpretable model that can segment data based on key characteristics.

The process of growing a classification tree involves recursive binary splitting, a technique that differs from regression trees primarily in its splitting criteria. While regression trees use Residual Sum of Squares (RSS), classification trees cannot employ the same metric. Instead, they use the classification error rate, which represents the fraction of training observations in a region that do not belong to the most common class (James, Witten, Hastie, & Tibshirani, 2021).

Model Description

Our implementation followed a rigorous methodology for developing a classification tree model to predict net income changes. The dataset preprocessing included removal of irrelevant columns and missing values, with the target variable, n*et_income_change*, binary-encoded as "Negative_Change" or "Positive_Change" to facilitate classification.

We implemented a time series cross-validation strategy using the caret package's timeslice method in R. When implementing the validation function, the training window was set to 35 963 observations, representing the observations until the second quarter of 2016. Simultaneously, the horizon parameter was set to 11 929 observations, representing the remaining observations of the initial train set spanning from Q2 2016 to Q2 2018.

We created a complexity plot to understand the relationship between the tree's complexity and predictive performance. This plot helps visualize how the model's error changes with different complexity parameters (CP) and tree sizes. We systematically explored different complexity parameters by varying the CP values from 0.000001 to 0.001. The plot in Figure 1 shows the relative error of the model against the number of splits in the tree and the corresponding complexity parameter.

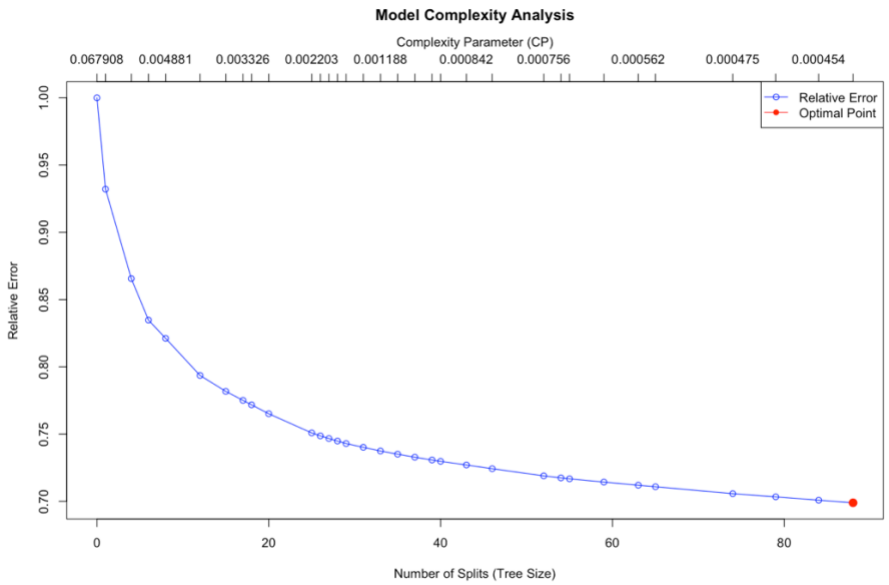

Figure 1 Model Complexity Analysis Classification Tree

Our analysis identified an optimal complexity parameter of 0.00045, which resulted in a tree with 88 splits. The red point on the plot indicates the configuration that minimizes the model's relative error, balancing model complexity with predictive accuracy. This approach allows us to select a tree structure that captures the underlying patterns in the data without overfitting the training set.

3.4 Random Forest

Random Forest is an ensemble method in which many classification trees are constructed and combined to make a powerful model. Each separate tree is called a weak-learner and does not need to make good predictions on its own, but when there is an ensemble of trees, the final random forest will use each tree to learn and make the optimal model. Random forest is a form of bagging but differs from regular bagged trees. When selecting the next variable to split from, regular bagging chooses from all the predictors, while random forest can choose from a random subset of predictors. An effect of choosing from the entire set of variables is that the most influential variables will be selected first, making the trees correlate. While dividing the predictors into subsets, the trees are decorrelated, making the model more reliable. As random forest methods normally provide better results than regular bagging, the technique is preferred in our report (James, Witten, Hastie, & Tibshirani, 2021).

Model Description

To evaluate the classification of *net_income_change*, we made a pipeline leveraging time-slice validation and parameter optimization using the grid search method. As in previous models, the same train and test split with time-slice validation was used for Random Forest.

While random forest is mostly self-regularizing and prevents overfitting by using random subsets of features, it can still benefit from tuning the hyperparameters (Pohl, Lecture: Validation and Testing, 2024). Specifically, tuning the parameters mtry and ntrees, can make the model perform better. Mtry refers to the number of random predictors considered when evaluating a new split. In a standard bagging model, this number represents the entire set of predictors, but random forest allows tuning. A commonly used value is the square root of the total number of predictors (James, Witten, Hastie, & Tibshirani, 2021). The hyperparameter ntrees represent the total number of trees to grow. (Rdocumentation, 2024).

Further on, the grid search method was implemented using the base function in R, *expand.grid*. In the grid search, the model was fitted with different sets of hyperparameters, and the parameters that yielded the best score based on the AUC metric were selected. Three values for mtry (square root of total predictors, half of the total predictors, and all predictors) and five values for ntrees (100, 250, 500, 1000, and 1500) were tested. A function iterated through each value of ntrees and applied the values of mtry to the model, saving the results in a data frame. To accelerate computation, the R library *Ranger* was used as the models method instead of *randomForest*. (Rdocumentation, 2024)

All available predictors were initially included in the model to evaluate its performance. This approach achieved an AUC score of 0.7113 on the validation set with mtry equal 132 and ntrees equal 500. However, the model was computationally expensive. To address these limitations, the variables acquired from the Lasso feature selection were applied to select a subset of predictors, which reduced the feature set while retaining the most significant variables. The model trained with Lasso-selected variables achieved a slightly higher AUC score of 0.7115; however, this required more trees. Given the comparable performance and reduced computational complexity, the Lasso-selected predictors were used in this paper.

After tuning the parameters with a grid search and validating the models with the timeslice method, the optimal hyperparameters were applied to the entire training set to fit the final model. Ranger was again chosen as the modeling method, with the importance parameter set to "impurity" to enable later feature importance analysis. “Impurity” represents the use of Gini index, which is preferred for classification problems (Rdocumentation, 2024) (James, Witten, Hastie, & Tibshirani, 2021). Although permutation importance was considered, the faster computation time of the Gini index made it the preferred choice (Pohl, Lecture: Interpretability, 2024). After fitting the model, measures like AUC-score, accuracy, and feature importance were calculated. This will be discussed under section 4. Results and Discussion.

3.5 XGBoost

XGBoost or extreme gradient boosting is like random forest an ensemble method. To understand extreme gradient boosting, we need first to address what gradient boosting (GBM) is. Boosting models are closely related to random forests as they both fit separate trees and use each tree to make a model. While random forest considers subsets of predictors at each split, GBM uses the whole sample and grows the tree sequentially. This means that for each new tree, they use the previous trees’ information to grow (James, Witten, Hastie, & Tibshirani, 2021). However, XGBoost differs from regular gradient boosting (GBM) as the trees are built parallel to each other and not sequentially (Nvidia, 2024). This allows for scaling and makes XGBoost a suitable method for this paper.

Model Description

The methods used to evaluate the classification of *net_income_change* using XGBoost are closely related to the methods used in the Random Forest model. Similarly, we used a pipeline incorporating time-slice validation and grid search for parameter tuning. As with the previous models, a single validation set was utilized to preserve the time-series structure of the data. The training window comprised the initial 35,963 observations (up to Q2 2016), while the validation horizon consisted of the subsequent 11,929 observations (from Q2 2016 to Q2 2018).

Boosting models, including XGBoost, can sometimes overfit when the number of trees is excessively high. However, the risk of overfitting can be reduced by employing validation and tuning appropriate hyperparameters (James, Witten, Hastie, & Tibshirani, 2021). In this paper, we performed a grid search to optimize the selected hyperparameters,

including the number of trees (nrounds), the shrinkage parameter (eta), interaction depth (max_depth), and the subsample ratio.

The number of trees (nrounds) controls the number of boosting iterations, and while increasing it can improve performance, it may also lead to gradual overfitting if too large. The shrinkage parameter (eta) regulates the learning rate, with smaller values requiring more trees to achieve optimal results but allowing the model to converge more carefully. Interaction depth (max_depth) controls the complexity of individual trees, with shallow trees focusing on simple patterns and deeper trees capturing more complex interactions (James, Witten, Hastie, & Tibshirani, 2021). Finally, the subsample ratio determines the proportion of randomly selected training data used for each tree, helping to reduce overfitting and variance. To capture most of the patterns in the data, this parameter should be above 0.5, or the samples can include insufficient information (DMLC XGBoost, u.d.).

As with the Random Forest model, *expand.grid* was used to perform the grid search, and the AUC score was used to evaluate the best hyperparameters to optimize the model. The model was tested with five different values for nrounds (100, 250, 500, 750, and 1000), three different values for max_depth (3, 6, and 9), three different values for eta (0.01, 0.1, and 0.3) and finally two distinct values for subsample (0.8 and 1). The remaining hyperparameters were set to the default value.

Regarding variable selection, the variables obtained through Lasso regularization were used in this model. As demonstrated previously with the Random Forest model, using these variables ensures a faster runtime while still capturing the underlying patterns in the data. Although the XGBoost model could produce different results when using all available variables, we assume that the performance would be similar based on the Random Forest results. Therefore, only the variables selected through Lasso were utilized in this analysis.

After tuning the parameters through a grid search and validating the models using the timeslice method, the optimal hyperparameters were applied to the full training set to fit the final XGBoost model. Feature importance was assessed using XGBoost’s built-in importance metrics, with "Gain" serving as the primary indicator. "Gain" measures the improvement in accuracy or reduction in error contributed by each feature at every split in the decision trees, making it a reliable way to assess variable impact (Rdocumentation, 2024). After fitting the final model, key performance metrics, including AUC, accuracy, sensitivity, specificity, and feature importance, were calculated and will be discussed in section 4, Results and Discussion.

3.6 Artificial Neural Networks

Artificial Neural Networks (ANN) are a subset “of machine learning models inspired by the structure and function of the human brain” (Marques, Santos, André, & Silva, 2024). It consists of interconnected nodes, or neurons, organized in layers. These nodes are connected by arrows, representing the flow of information (Pohl, Lecture: Deep Learning, 2024). The structure of the topology of the network defines how the nodes are connected.

Our approach focuses on feedforward networks where information flows only from input to output, without any cycles or loops. This ensures that the output of a node does not eventually flow back into itself. Figure 2 illustrates a simple visualization of an ANN model, including the input layer, hidden layer, and output layer.

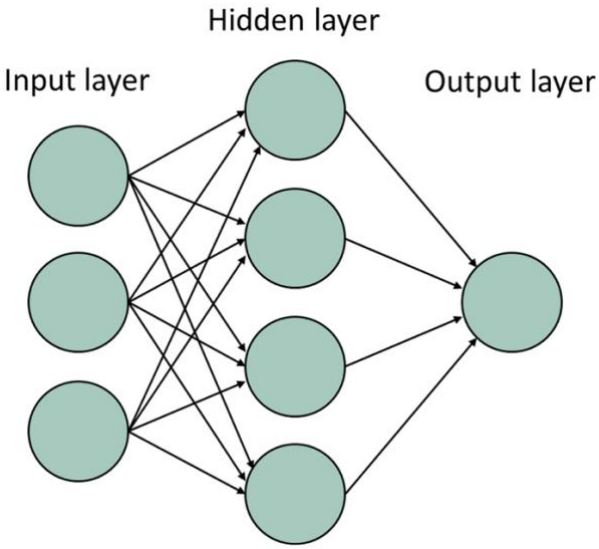

Figure 2 A Simple Visualization of an ANN Model

Model Description

The ANN model implemented in this project aims to predict changes in net income. The model was built using the Keras library in R, which provides an intuitive interface for developing and training deep learning models. The architecture of our ANN consists of multiple dense layers with ReLU (Rectified Linear Unit) activation functions, batch normalization, and dropout layers to prevent overfitting. The final layer uses a sigmoid activation function to produce binary classifications, indicating whether the net income change is positive or negative.

For hidden layers, experience has shown that the best choice of activation function is Rectified Linear Unit (RELU). ReLU helps the network to learn complex patterns in the

data by introducing non-linearity, which is crucial for capturing intricate relationships. For the output layer in binary classification tasks, the sigmoid activation function is used to model probabilities and output a value between 0 and 1 (Pohl, Lecture: Deep Learning, 2024).

The final structure of our neural network model is shown in Table 3 below.

Layer type	Neurons	Activation Function	Additional Technique
Input Layer	26	ReLU	
Hidden Layer 1	40	ReLU	Dropout = 0.1
Hidden Layer 2	20	ReLU	Dropout = 0.1
Hidden Layer 3	10	ReLU	Dropout = 0.2
Output Layer	1	Sigmoid	

Table 3 Structure of Our Neural Network Model

The target variable, *net_income_change*, was converted to a numeric format to ensure compatibility with the neural network model. The data was then split into features and a target variable. To create training and validation sets, we implemented the same time-based split as previously. This approach ensures that the model is tested on data that comes after the training period, simulating a real-world scenario where future data is predicted based on past data.

The ANN model was trained using different batch sizes to identify the optimal configuration. We evaluated batch sizes of 32, 64, and 128 and found that a batch size of 32 provided the best balance between model performance and generalization. Specifically, the batch size of 32 achieved the highest train accuracy and a favorable trade-off between training and validation metrics.

We implemented several techniques to prevent overfitting and enhance the model's generalizability. Dropout regularization was applied with dropout rates set to 0.1 for initial hidden layers and 0.2 for deeper layers. Inspired by the concept of random forests, this method randomly deactivates a fraction of neurons during training. This prevents neurons

from becoming overly specialized and forces the network to learn more robust, generalized features (James, Witten, Hastie, & Tibshirani, 2021).

Batch normalization is another technique we utilized. Applied after each hidden layer, this method normalizes each layer's inputs to stabilize the learning process, reduce internal covariate shift, and enable higher learning rates and faster convergence.

Additionally, early stopping was employed with patience set to 15 epochs. This means the training process ends if the validation loss fails to improve for 15 consecutive epochs, preventing the model from overfitting by memorizing the training data rather than learning generalizable patterns.

The training process used the Adam optimizer to update the weights and minimize the loss function iteratively (Pohl, Lecture: Deep Learning, 2024). We experimented with different learning rates to find the most effective one. Lower learning rates generally take longer to converge but can result in better overall performance. Our experiments found that a learning rate of 0.0003 provided a good balance between convergence speed and model stability, making it sufficient for our needs.

Figure 3 Training and Validation Loss ANN Model

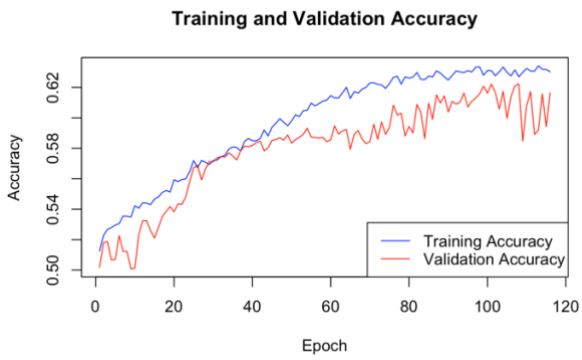

Figure 4 Training and Validation Accuracy ANN Model

4. Results and Discussion

4.1 Lasso Classification

The confusion matrix in the table below shows how well the classification tree predicted the outcomes. The columns represent the actual outcomes, and the rows represent the predicted outcomes. The model correctly predicted 2510 instances as *Negative Change* and 3039 instances as *Positive Change*, resulting in an accuracy of **58.65%.**

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	2510	1894
Positive Change	2074	3039

Table 4 Confusion Matrix Lasso Classification

The Area Under the Curve (AUC) quantifies the overall ability of the model to discriminate between the classes. It quantifies the model's ability to predict more nuanced by scoring predictions on a spectrum, giving value to almost right predictions, rather than a binary right/wrong where all wrong predictions are scored the same. The AUC score for the Lasso regression was 0.6286, with its ROC curve in Figure 5 below. We can observe a significant improvement from the random guessing (50%) represented by the gray line and how far off it is from the theoretical maximum, where it would have hugged the left corner perfectly (Pohl, Lecture: Nonlinear Classification, 2024).

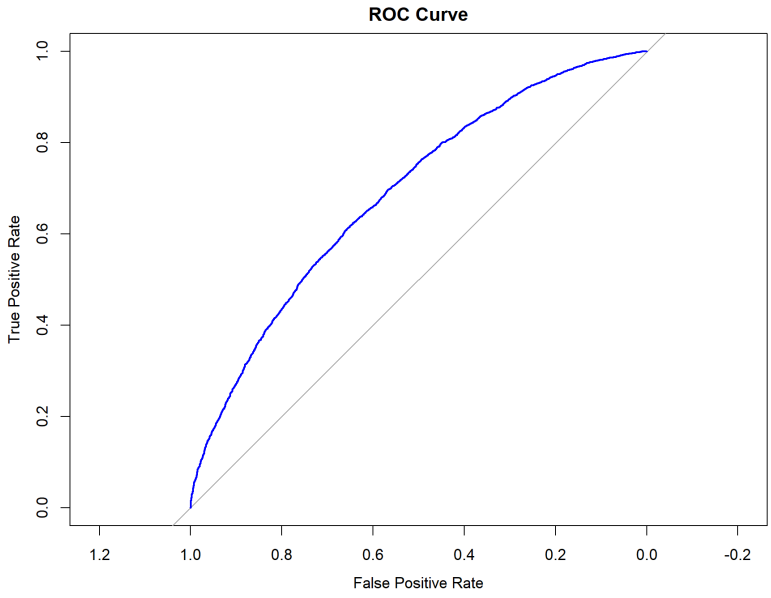

Figure 5 ROC Curve Lasso

Metric	Value
Accuracy	58.65%
AUC	62.86%

Table 5 Results Lasso Classification

4.2 Classification Trees

The classification tree correctly classified 2867 instances as *Negative Change* and 3055 instances as *Positive Change*, resulting in an accuracy of **61.71%.**

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	2867	1878
Positive Change	1797	3055

Table 6 Confusion Matrix Classification Trees

“The ROC curve is a graphical representation of the trade-off between the True Positive Rate and the False Positive Rate” (Truera, n.d.) at various thresholds for a classification model. The curve illustrates how well the model distinguishes between the two classes *Positive Change* and *Negative Change.* A perfect model would have an ROC curve that hugs the top left corner of the plot, while a random model would produce a diagonal line as we could see from the Dumb Model. The AUC in this model is 0.652, which suggests the model has moderate discriminatory power.

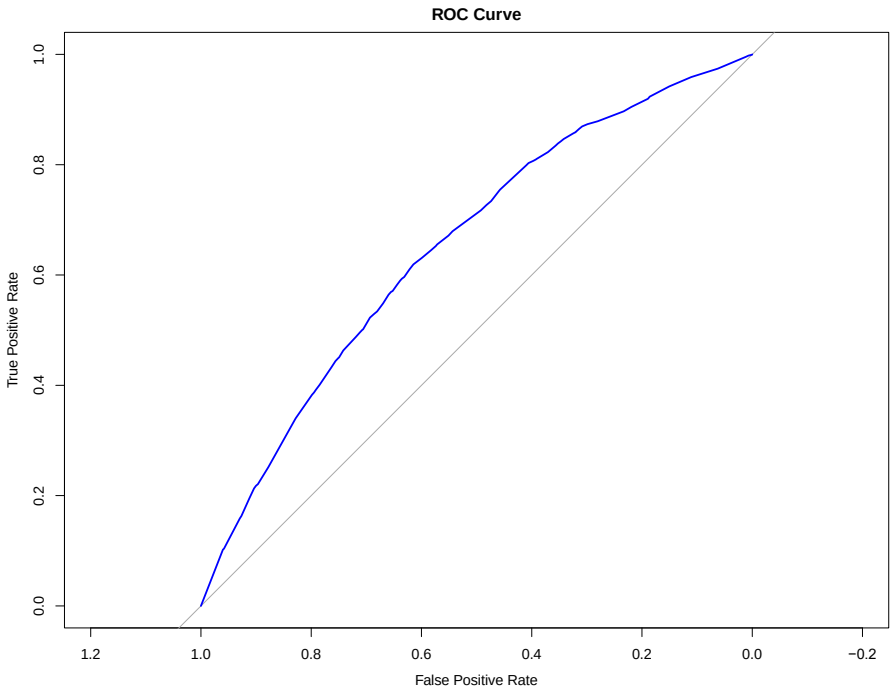

Figure 6 ROC Curve Classification Trees

Metric	Value
Accuracy	61.71%
AUC	65.2%

Table 7 Results Classification Trees

4.3 Random Forest

Results

To determine the optimal hyperparameters for the random forest model, the grid search method introduced in part 3: Methodology and Analysis, was used. Figure 7 illustrates the results from the grid search and shows the relationship between AUC, the number of trees (ntress), and the number of predictors used for splitting (mtyr). The highest AUC score, 0.7115, on the validation set was achieved when mtry was 13 and ntrees was 1500. As the graph in Figure 7 shows, the AUC-score seems to settle with a higher number of trees, making 1500 a sufficient amount of trees for this model. (James, Witten, Hastie, & Tibshirani, 2021).

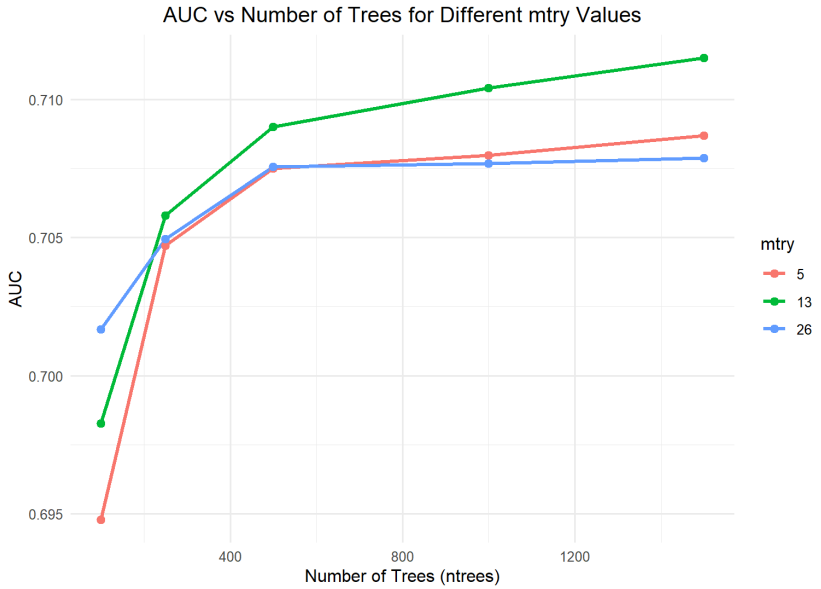

Figure 7 AUC vs Number of Trees for Different mtry Values

The best-performing model, with the hyperparameters acquired from the grid search, was fitted with the test data to evaluate the model's performance. From the confusion matrix in Table 8 below, we notice that the model predicted a decrease in net income correct 2822 times and an increase in net income correct 3212 times. The model correctly classified **62.87%** of the observations in the test set, reflecting the accuracy score.

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	2822	1721
Positive Change	1842	3212

Table 8 Confusion Matrix Random Forest

As visualized in Figure 8, an AUC score of 0.6889 suggests room for improvement when discriminating between classes.

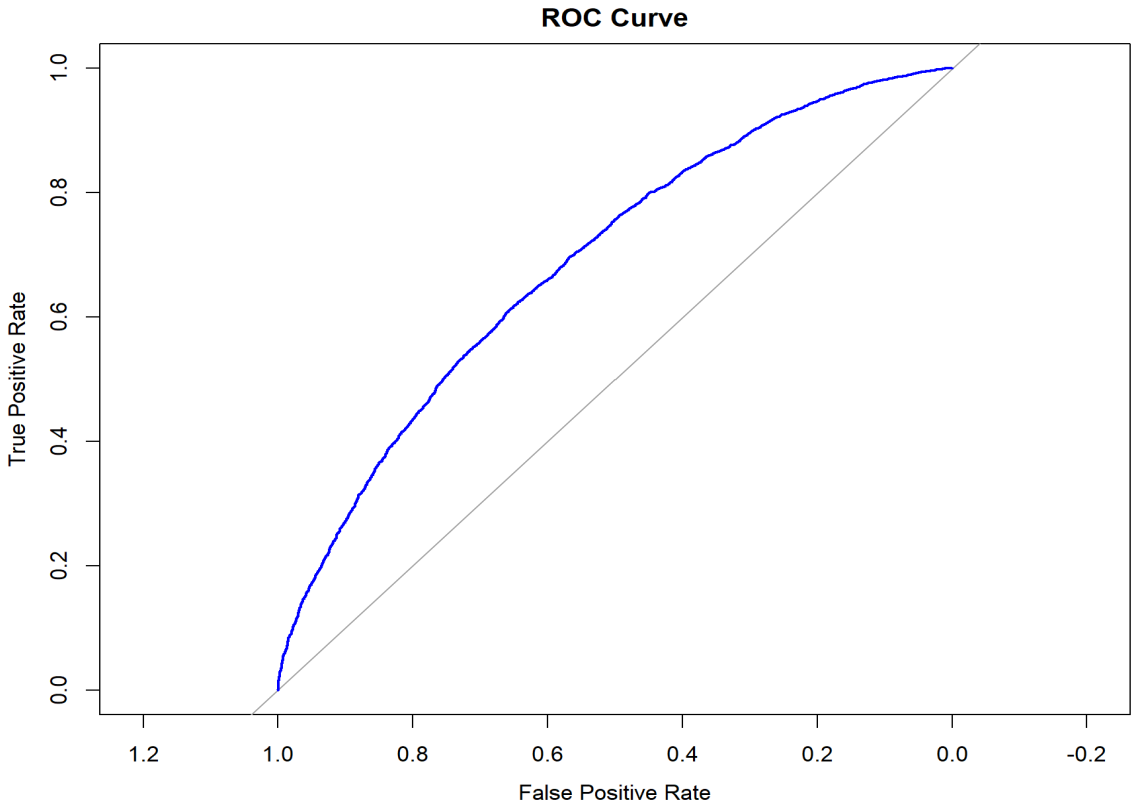

Figure 8 ROC Curve Random Forest

A feature importance plot (Figure 9) was included to discuss the model further. It shows the relative contribution of each predictor to our model. As mentioned in the methodology section, the Gini Index is used to measure the importance, and the variables with the largest mean decrease in the Gini Index are the most important variables. From Figure 9, we notice that the financial indicators play a crucial role in predicting change in net income, with earnings per share (oepf12_lag and epspiq_lag), non-operating income(nopiq_lag), and beta (beta_lag) being the most influential. The macroeconomic variables are calculated to be less contributive to the model than the financial variables.

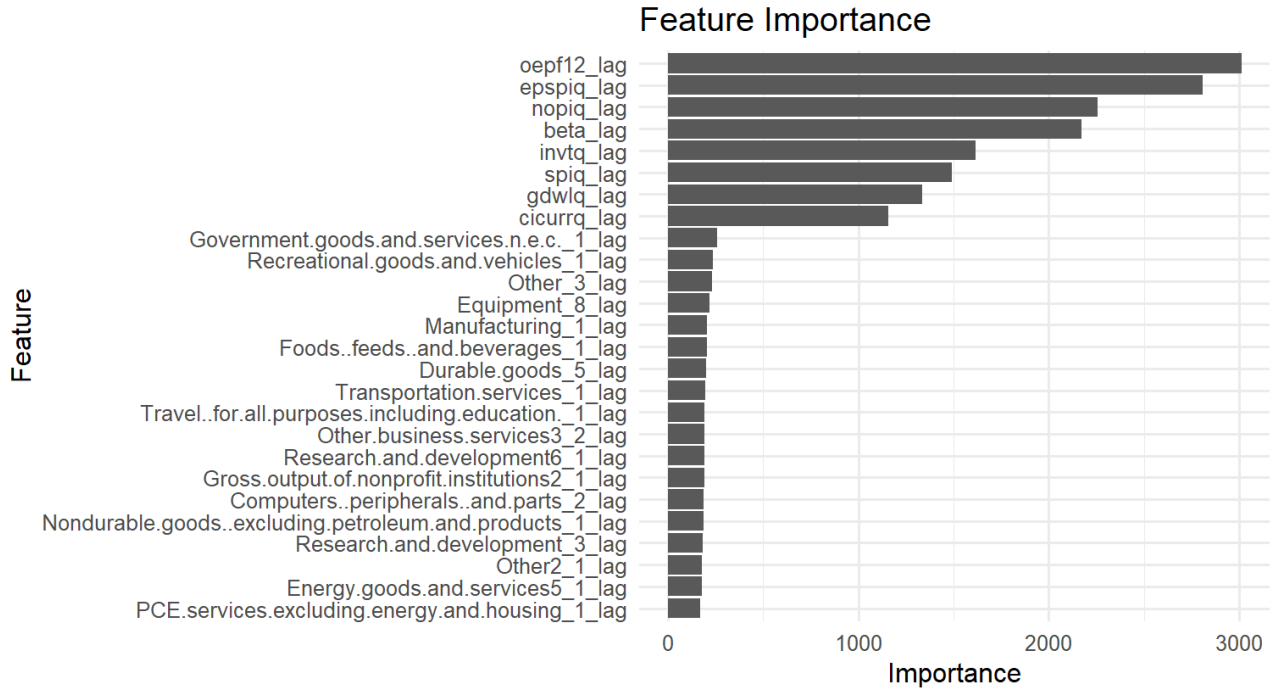

Figure 9 Feature Importance Random Forest

Metric	Value
Accuracy	62.87%
AUC	68.89%

Table 9 Results Random Forest

4.4 XGBoost

As described in the Methodology and Analysis section, a grid search method was used to determine the optimal hyperparameters for the XGBoost model. Figure 10 illustrates the results from the grid search, showing the relationship between AUC, the number of boosting rounds (nrounds), the learning rate (eta), and the tree depth (max_depth). The highest AUC score on the validation set was achieved with the following hyperparameters: nrounds = 500, max_depth = 9, eta = 0.01, and subsample = 0.8. As shown in the figure, smaller eta values generally required more boosting rounds to achieve higher AUC scores, especially for deeper trees, confirming the trade-off between learning rate and boosting iterations. Tree depth also had a noticeable impact, with deeper trees (6 and 9) outperforming shallower ones (3).

Notably, in Figure 10, two points appear at the same number of boosting rounds (nrounds). This is because the grid search included multiple values of the subsample parameter (0.8 and 1) for the same combinations of nrounds, eta, and max_depth. These overlapping points reflect the slight variations in AUC due to differences in the proportion of rows used to build each tree.

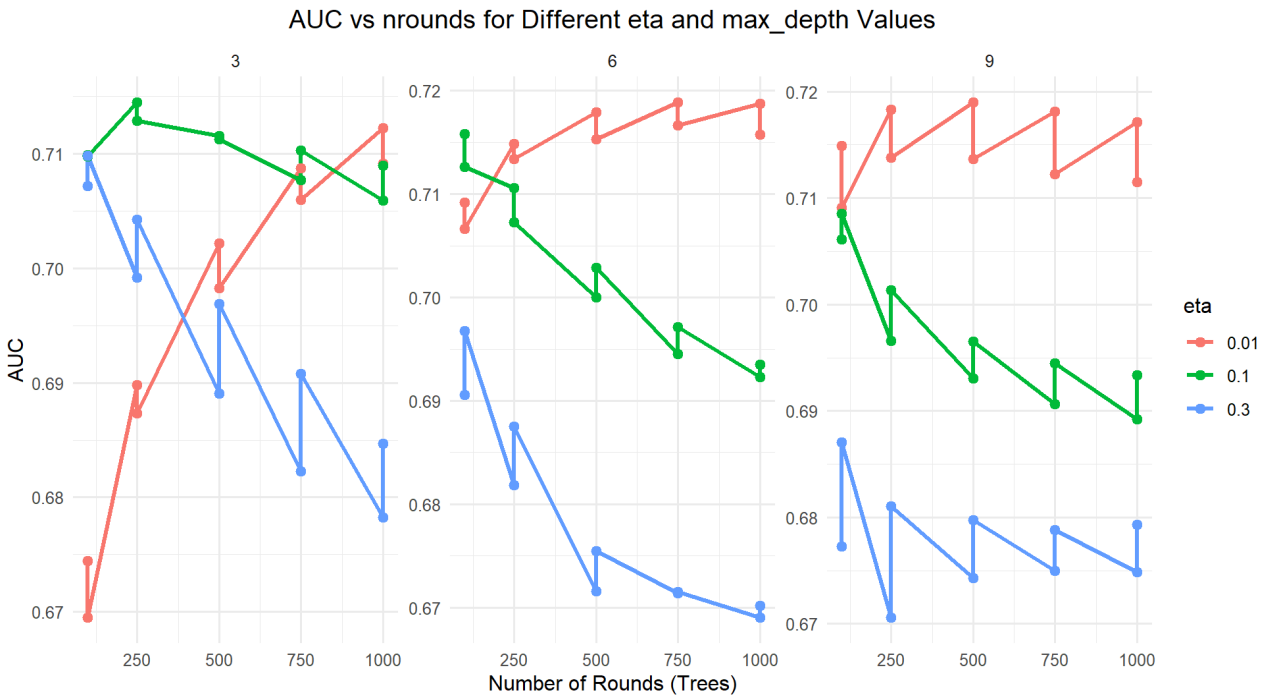

Figure 10 AUC vs nrounds for Different eta and max_depth Values

The model with these optimal hyperparameters was then fitted to the test set to evaluate its performance. From the confusion matrix (Table 10), the XGBoost model correctly predicted a decrease in net income 2889 times and an increase in net income 3142 times. Overall, the model correctly classified **62.95%** of observations in the test set, reflected in the accuracy score.

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	2889	1791
Positive Change	1765	3142

Table 10 Confusion Matrix XGBoost

For the test set, the AUC was calculated at 0.6952, which reflects moderate performance. Nonetheless, it outperforms random guessing, represented by the diagonal grey line.

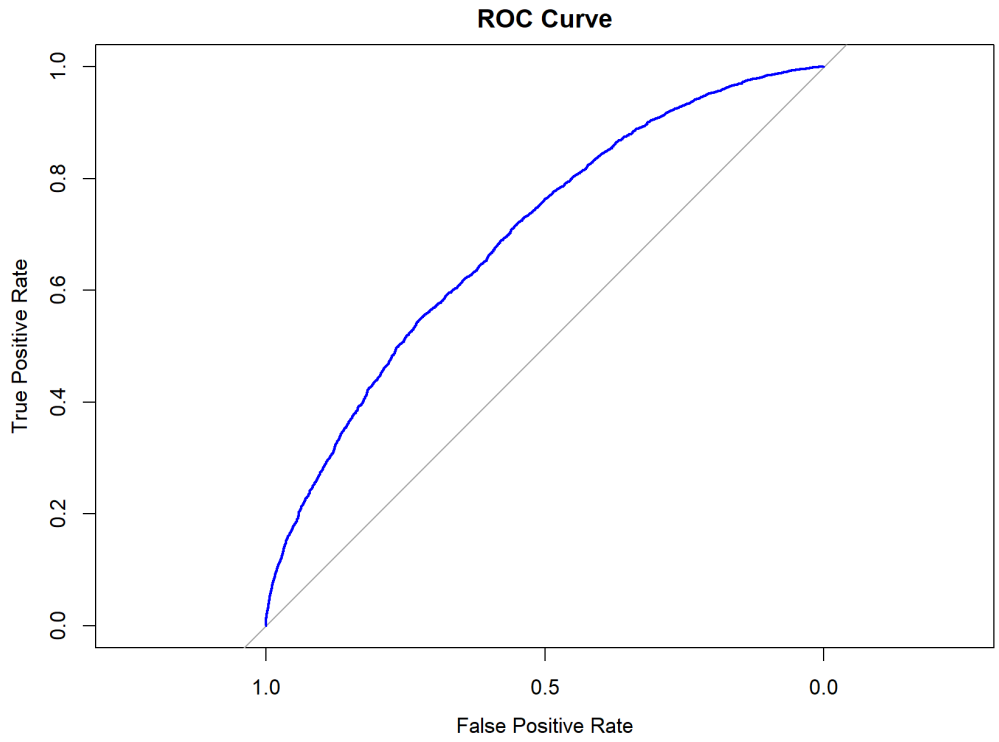

Figure 11 ROC Curve XGBoost

To better understand the XGBoost model's behavior, a feature importance plot (Figure 12) was included. This plot measures the relative contribution of each predictor using the "Gain" metric, which quantifies how much each feature reduces the loss function. We notice that financial indicators played a dominant role in predicting changes in net income. Among the most important predictors were oepf12_lag, epspiq_lag, nopiq_lag, and spiq_lag, highlighting the relevance of earnings and profitability metrics. Macroeconomic variables and sector-specific features contributed less to the model but still added nuance to the overall predictions.

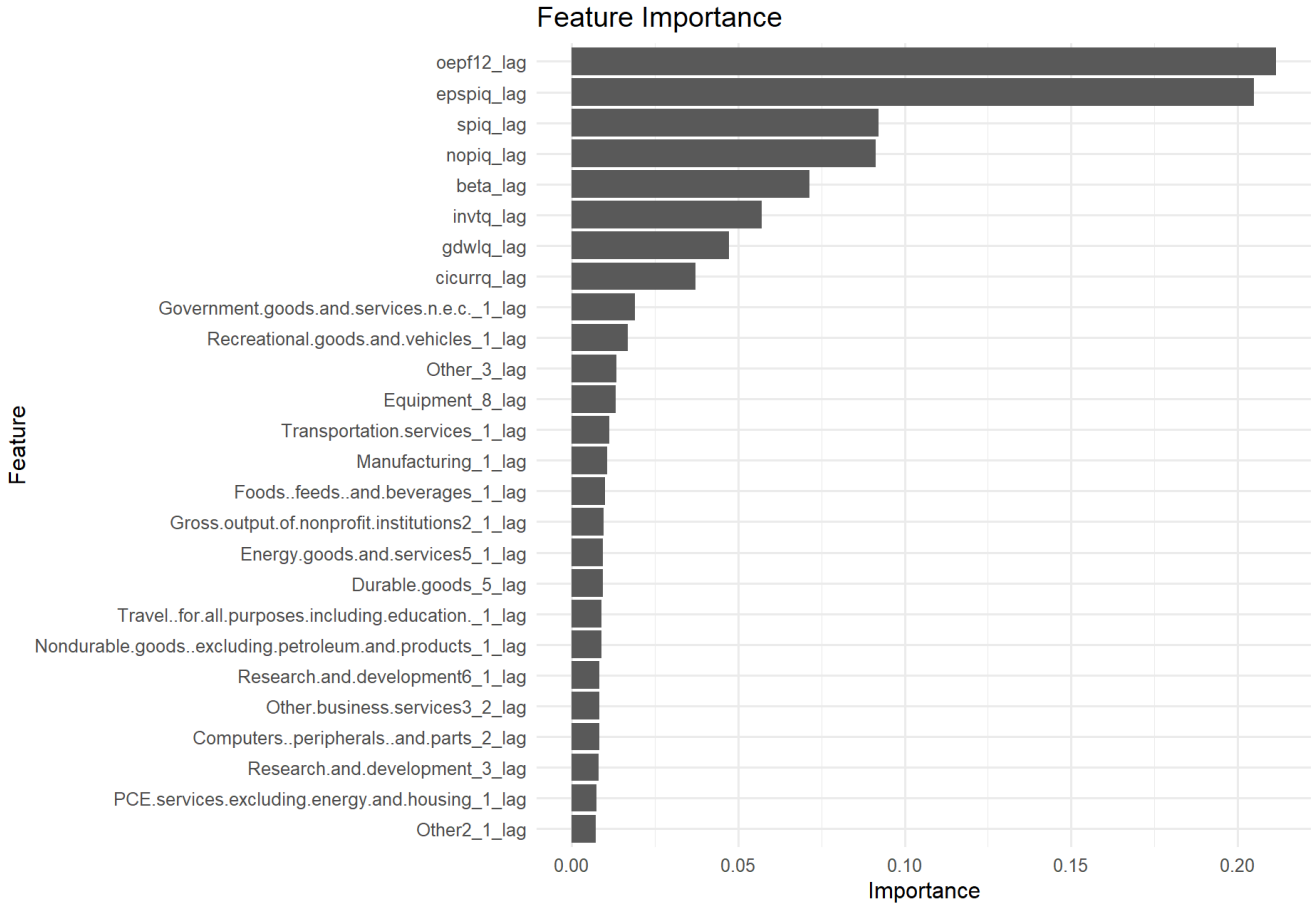

Figure 12 Feature Importance XGBoost

Metric	Value
Accuracy	62.95%
AUC	69.52%

Table 11 Results XGBoost

4.5 Artificial Neural Networks

The ANN model's performance was evaluated using the same metrics to assess its predictive capability as our other machine learning methods. The confusion matrix below revealed the following classification outcomes on the test set.

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	2241	1355
Positive Change	2423	3578

Table 12 Test Set Confusion Matrix Artificial Neural Networks

From the results, the ANN model achieved an accuracy of **60.63%.** It shows that the model has learned some patterns in the data and performs better than our Dumb model. We examined the model's performance on the training data to further validate it. The confusion matrix for the training set is shown below:

	Reference	
Prediction	Negative Change	Positive Change
Negative Change	8848	4409
Positive Change	8708	13998

Table 13 Training Set Confusion Matrix Artificial Neural Networks

For the training data, the accuracy is 63.52%. Compared to the test set, this accuracy on the training set suggests that the measures we implemented to prevent overfitting, such as dropout regularization and early stopping, were effective.

In addition to accuracy, we also evaluated the model using the AUC metric. The AUC value of 0.6501 indicates that the model can moderately discriminate between positive and negative changes in net income. This metric provides a richer view of the model's performance to balance sensitivity and specificity.

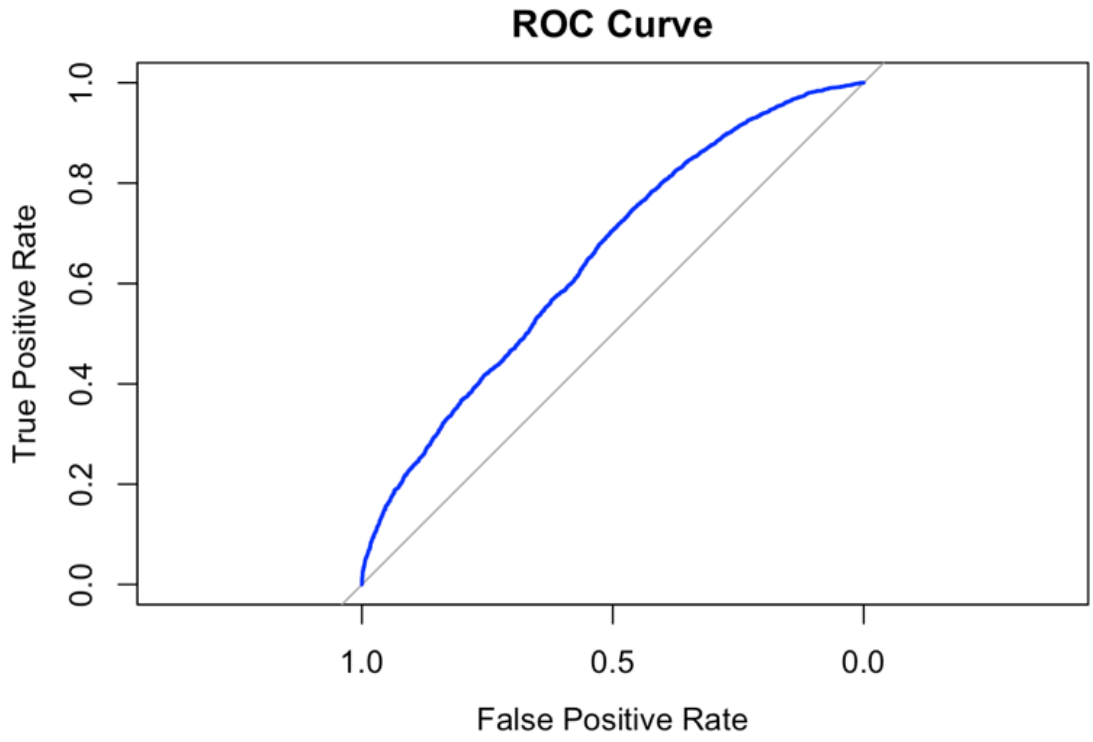

Figure 13 ROC Curve Artificial Neural Network

Metric	Value
Accuracy	60.63%
AUC	65.01%

Table 14 Results Artificial Neural Network

4.6 Comparison

The analysis compared several machine learning models against a baseline Dumb Model. Table 15 shows that all advanced models significantly improved predictive performance compared to the Dumb Model. The XGBoost model had the highest accuracy and AUC scores, followed closely by Random Forest.

Each model demonstrated unique strengths, with ensemble methods like XGBoost and Random Forest performing particularly well. The consistent improvement over the baseline model highlights the effectiveness of these advanced machine learning techniques in predicting changes in net income.

Model	Accuracy (%)	AUC (%)
XGBoost	62.95	69.52
Random Forest	62.87	68.89
Classification Tree	61.71	65.2
ANN	60.63	65.01
Lasso	58.65	62.86
Dumb Model	51.4	50

Table 15 Comparison of Different Machine Learning Methods

5. Trading Strategy

This study implements a trading strategy inspired by Ou and Penman (1989), which focuses on selecting stocks with a high probability of experiencing an increase in net income. Specifically, a threshold of 0.55 is applied to the predicted probability of net income increase to identify stocks for inclusion in the portfolio. The predictions were generated using the XGBoost model, which yielded the highest accuracy score.

The trading strategy consists of selecting stocks at the beginning of each quarter, holding them throughout the quarter, and selling the stocks at the end of the quarter. The strategy's performance is evaluated based on average quarterly and cumulative returns and compared to the S&P 500 Index as a benchmark.

To prepare the data for the strategy, we merged the prediction probabilities with the CRSP dataset to include the stock price of each company. Since the probabilities are based on quarterly data, we obtained the last recorded price of each company at the end of each quarter. The quarterly return was then calculated using the following formula: $Return = \frac{Price_t - Price_{t-1}}{Price_t}$. Afterward, all rows containing NA values were removed to ensure data quality.

To initialize the trading strategy, we saved the companies with predicted probabilities above the threshold for each quarter, assuming an equal investment in each selected

company. The mean portfolio return and cumulative portfolio return were then calculated. The cumulative portfolio return represents the compounded growth of the portfolio over time.

The strategy yielded a final cumulative return of 12.2%, as shown in Table 16. However, if we were to invest in the S&P 500 Index instead while holding the investment for the entire period, it would have yielded a return of 18.14%. While the strategy resulted in a positive return, the statement in the introduction regarding the difficulty of predicting the future holds true, as investing in the S&P 500 Index would have resulted in a higher overall return.

	Return	Cumulative Return
2018Q2	16.40 %	16.40 %
2018Q3	-1.05 %	15.20 %
2018Q4	-25.10 %	-13.7 %
2019Q1	19.30 %	2.9 %
2019Q2	-2.58 %	2.6 %
2019Q3	-3.10 %	-0.5 %
2019Q4	12.90 %	12.2 %

S&P500	
2018Q2	2726.71
2019Q4	3221.29
Return	18.14 %

Table 16 Results from Trading Strategy

6. Conclusion

6.1 Limitations

One of the downsides of our approach regarding the validation of the different models is that we only used one validation set. When we included macroeconomic factors we included the same values for each company, given the same quarter. This, and an uneven distribution of observations per quarte,r made it difficult to control where the train and validation were split. We chose the one validation split to avoid leaking out-of-sample data from the validation set when training, but the implementation of cross-validation could have yielded better results. However, some bias was introduced when training the models.

6.2 Key Results

Key results from our term paper highlight the effectiveness of machine learning models in predicting net income changes. The XGBoost model stood out with the highest accuracy of 62.95% and an AUC score of 69.52%, closely followed by the Random Forest model with similarly strong performance. These advanced models significantly outperformed the baseline Dumb Model, demonstrating the power of sophisticated machine learning techniques in financial forecasting. The analysis revealed that financial indicators, especially earnings per share and non-operating income, were the most crucial predictors of net income fluctuations.

The trading strategy developed from the XGBoost model showed promise but ultimately fell short of market benchmarks, generating a 12.2% cumulative return compared to the S&P 500 Index's 18.14%. This finding illustrates the challenges of transforming predictive models into profitable trading strategies. The research process involved substantial data refinement, transforming the initial dataset from 239,642 observations across 375 variables to a more manageable 57,489 observations and 267 variables through careful cleaning and Lasso regularization.

References

Complete Dissertation by STATISTICS SOLUTIONS. (2024). Retrieved from Managing Missing Data: https://www.statisticssolutions.com/managing-missing-data/

DMLC XGBoost. (n.d.). Retrieved from https://xgboost.readthedocs.io/en/stable/parameter.html#parameters-for-tree-booster

Farbahari, A., Dehesh, T., & Gozashti, M. H. (2019). THE USAGE OF LASSO, RIDGE, AND LINEAR REGRESSION TO EXPLORE THE MOST INFLUENTIAL METABOLIC VARIABLES THAT AFFECT FASTING BLOOD SUGAR IN TYPE 2 DIABETES PATIENTS. *Rom J Diabetes Nutr Metab Dis. 26(4)*, 371-379.

Filho, M. (2023, June 29). *Does Random Forest Need Feature Scaling or Normalization?* . Retrieved from Forecastegy: https://forecastegy.com/posts/does-random-forest-need-feature-scaling-or-normalization/

GeeksForGeeks. (2024, August 07). *Cross Validation in Machine Learning*. Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/cross-validation-machine-learning/

Hayes, A. (2022, June 30). *Investopedia*. Retrieved from https://www.investopedia.com/terms/c/compustat.asp

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: with Applications in R. Second Edition.* New York: Springer.

Kamkar, I., Gupta, S. K., Phung, D., & Venkatesh, S. (2015). Stable feature selection for clinical prediction: Exploiting ICD tree structure using Tree-Lasso. *Journal of Biomedical Informatics, Volume 53*, 277-290.

Lingard, H., Hallowell, M., Salas, R., & Pirzadeh, P. (2017). Leading or lagging? Temporal analysis of safety indicators on a large infrastructure construction project. *Safety Science, Volume 91*, 206-220.

Marques, R., Santos, J., André, A., & Silva, J. (2024). Ultrasound Versus Elastography in the Diagnosis of Hepatic Steatosis: Evaluation of Traditional Machine Learning Versus Deep Learning. *Sensors, 24(23), 7568.*

Murel, J. (2023, November 16). *What is regularization?* Retrieved from IBM: https://www.ibm.com/think/topics/regularization

Narkhede, S. (2018). *Medium*. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Nvidia. (2024). Retrieved from https://www.nvidia.com/en-us/glossary/xgboost/

OECD. (2012, April). *OECD SYSTEM OF COMPOSITE LEADING INDICATORS.* Retrieved from https://www.oecd.org/content/dam/oecd/en/data/methods/OECD-System-of-Composite-Leading-Indicators.pdf

Ou, J. A., & Penman, S. H. (1989). FINANCIAL STATEMENT ANALYSIS AND THE PREDICTION. *Journal of Accounting and Economics 11 (1989)*.

Pohl, W. (2024). *Lecture: Crash Course in Financial Markets.* Bergen.

Pohl, W. (2024). *Lecture: Deep Learning.* Bergen.

Pohl, W. (2024). *Lecture: Interpretability.* Bergen.

Pohl, W. (2024). *Lecture: Nonlinear Classification.* Bergen.

Pohl, W. (2024). *Lecture: Regularization.* Bergen.

Pohl, W. (2024). *Lecture: Supervised Learning Pipeline.*

Pohl, W. (2024). *Lecture: Trees, Bagging, Boosting.* Bergen.

Pohl, W. (2024). *Lecture: Validation and Testing.* Bergen.

Rdocumentation. (2024). *Rdocumentation*. Retrieved from https://www.rdocumentation.org/packages/ranger/versions/0.16.0/topics/ranger

Saleh, E., & Shalabh. (2014). A ridge regression estimation approach to the measurement error model. *Journal of Multivariate Analysis, Volume 123*, 68-84.

Tangjitprom, N. (2012). The Review of Macroeconomic Factors and Stock Returns. *International Business Research; Vol. 5, No. 8*, 107-115.

The Economic Times. (2024). Retrieved from https://economictimes.indiatimes.com/definition/beta

The Wharton School. (2018, October). *WRDS BACKTESTER User Manual.* Retrieved from wrds-web.wharton.upenn.edu: https://wrds-www.wharton.upenn.edu/documents/1109/Backtest_Manual_v2.pdf

Tinoco, M. H., & Wilson, N. (2013). Financial distress and bankruptcy prediction among listed companies. *International Review of Financial Analysis 30*, 394-414.

Truera. (n.d.). *Introduction to ML Model Performance*. Retrieved from https://truera.com/ai-quality-education/performance/what-is-ml-model-performance/

World Bank Group. (2022). *World Development Report 2022: Chapter 1. The economic impacts of the COVID-19 crisis*. Retrieved from https://www.worldbank.org/en/publication/wdr2022/brief/chapter-1-introduction-the-economic-impacts-of-the-covid-19-crisis