Classify US (2018) stocks by Financial Indicators

2025-04-27

Context

Using over 200 financial indicators to classify whether a stock's price will increase during 2018. In other words, the goal is to determine whether an investor should buy the stock at the beginning of 2018 or not.

Data set

To set up the data realistically and avoid overfitting, removing the percentage price variation column is recommended to ensure that the model uses only information available at the time of prediction. This column is directly linked to the Class variable and would not be known at the start of 2018.

Example: If a stock's `X2019.PRICE.VAR...` value is positive, then the stock is labeled as `Class = 1`, meaning an investor should buy the stock at the start of 2018 and sell it at the end of the year. Conversely, if the value is negative, it is labeled as `Class = 0`, suggesting that the stock should not be bought under this model.

This setup better mirrors a real-world investment decision, where actions must be taken without future information. The models are trained to predict `Class`, which represents whether a stock would have been a profitable buy at the start of 2018.

Cleaning and preparing the dataset

```
data <- read.csv("2018_Financial_Data.csv")

#Drop stock % price variation, symbol (X) and Sector
data <- data |> select(-X, -Sector, -`X2019.PRICE.VAR....`, -operatingProfitMargin)
#-operatingProfitMargin because it is a col of just a constanst of 1

# Print out dropped cols
print(colnames(data[, colMeans(is.na(data)) > 0.80]))
```

```
## [1] "operatingCycle"      "cashConversionCycle"
```

```
# Drop columns with more than 20% NAs
data <- data[, colMeans(is.na(data)) <= 0.20]

# Fill missing vals with mean cols
data <- data |>
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))

# Set as factor
```

```r
data$Class <- as.factor(data$Class)

# Split train/test
set.seed(123)
index <- createDataPartition(data$Class, p = 0.7, list = FALSE)
train <- data[index, ]
test <- data[-index, ]

#Normalize all numeric cols
train_x <- train %>%
  select(-Class) %>%
  scale()

train_glm <- as.data.frame(train_x)
train_y <- train$Class
train_glm$Class <- train_y

# Scale the same with train set
test_x <- test %>%
  select(where(is.numeric)) %>%
  scale(
    center = attr(train_x, "scaled:center"),
    scale  = attr(train_x, "scaled:scale")
  )

test_glm <- as.data.frame(test_x)
test_y <- test$Class
test_glm$Class <- test_y
```

```r
set.seed(123)
# 10-folds cv
ctrl <- trainControl(method = "cv", number = 10)
```

Variable Selections

Due to multicollinearity among predictors, LASSO was used to reduce redundancy and retain only revelant features. This improves model stability, interpretability, and helps prevent overfitting in a high-dimensional financial dataset.

```r
# Setting up
x <- model.matrix(Class ~ ., data = train_glm)[, -1]

#LASSO
cvfit <- cv.glmnet(x, train_glm$Class, family = "binomial", alpha = 1)

# Get selected variables
selected_vars <- coef(cvfit, s = "lambda.min")
selected_vars <- rownames(selected_vars)[which(selected_vars != 0)][-1]

#Filter train/test

filtered_train <- train_glm[, c(selected_vars,"Class")]
filtered_test <- test_glm[,c(selected_vars,"Class")]
```

Logistic Regression

```r
set.seed(123)
# Train with cv
log_model <- train(Class ~ ., data = filtered_train,
                   method = "glm", family = "binomial", trControl = ctrl)
summary(log_model)
```

```
##
## Call:
## NULL
##
## Coefficients: (7 not defined because of singularities)
##                                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)                               1.36272    0.29874   4.562 5.08e-06
## Income.Tax.Expense                        0.13083    0.11266   1.161  0.24554
## Net.Income...Non.Controlling.int         -0.12709    0.07773  -1.635  0.10204
## Preferred.Dividends                       0.06690    0.11652   0.574  0.56585
## EPS.Diluted                               0.41001    0.63816   0.642  0.52055
## Weighted.Average.Shs.Out                 -0.33910    0.10930  -3.102  0.00192
## Dividend.per.Share                        0.66611    0.10942   6.088 1.15e-09
## Gross.Margin                              0.15085    0.06716   2.246  0.02469
## Profit.Margin                             0.05182    0.06507   0.796  0.42576
## EBIT                                     -0.13917    0.17838  -0.780  0.43529
## Cash.and.cash.equivalents                 3.91090    2.34460   1.668  0.09531
## Cash.and.short.term.investments           0.07409    0.26163   0.283  0.77704
## Long.term.debt                            0.02543    0.08433   0.302  0.76296
## Deferred.revenue                          0.22335    0.17092   1.307  0.19129
## priceEarningsRatio                        0.38700    0.12412   3.118  0.00182
## priceToFreeCashFlowsRatio                 0.25861    0.11165   2.316  0.02055
## dividendYield                             0.32146    0.74383   0.432  0.66562
## payablesTurnover                         -0.99545    0.45974  -2.165  0.03037
## inventoryTurnover                        -0.31673    0.44425  -0.713  0.47588
## fixedAssetTurnover                        0.12504    0.29221   0.428  0.66871
## assetTurnover                            -0.13065    0.04462  -2.928  0.00341
## cashRatio                                -0.03845    0.04103  -0.937  0.34874
## debtEquityRatio                           0.05589    0.03973   1.407  0.15947
## longtermDebtToCapitalization              0.17977    0.09441   1.904  0.05687
## interestCoverage                         -0.13862    0.04308  -3.218  0.00129
## operatingCashFlowPerShare                -2.19824    2.03794  -1.079  0.28074
## freeCashFlowPerShare                     13.44747   23.65845   0.568  0.56976
## Operating.Cash.Flow.per.Share                 NA         NA      NA       NA
## Free.Cash.Flow.per.Share                      NA         NA      NA       NA
## Market.Cap                               -0.09158    0.19516  -0.469  0.63887
## Enterprise.Value                          0.33003    0.20131   1.639  0.10113
## PE.ratio                                      NA         NA      NA       NA
## Enterprise.Value.over.EBITDA             -0.06240    0.05083  -1.228  0.21957
## EV.to.Free.cash.flow                     -0.41256    0.17714  -2.329  0.01986
## Earnings.Yield                            1.99230    4.43854   0.449  0.65353
## Debt.to.Equity                                NA         NA      NA       NA
## Interest.Coverage                             NA         NA      NA       NA
## Dividend.Yield                           -0.30728    0.14996  -2.049  0.04046
```

```
## Intangibles.to.Total.Assets                0.03175    0.04473   0.710  0.47777
## Capex.to.Revenue                          -0.10630    0.10618  -1.001  0.31678
## Capex.to.Depreciation                      0.03271    0.03860   0.847  0.39683
## Graham.Net.Net                            -0.03973    0.05293  -0.751  0.45294
## Net.Current.Asset.Value                   -0.83109    0.47696  -1.742  0.08143
## Average.Inventory                          0.17384    0.17166   1.013  0.31120
## Payables.Turnover                               NA         NA      NA       NA
## Inventory.Turnover                              NA         NA      NA       NA
## Operating.Income.Growth                   -0.06180    0.04100  -1.507  0.13179
## EPS.Growth                                -0.09285    0.04645  -1.999  0.04562
## Weighted.Average.Shares.Diluted.Growth    -0.41999    0.14809  -2.836  0.00457
## Dividends.per.Share.Growth                -0.18083    0.05750  -3.145  0.00166
## Operating.Cash.Flow.growth                -0.14721    0.38340  -0.384  0.70101
## Free.Cash.Flow.growth                     -0.05682    0.06748  -0.842  0.39983
## X5Y.Revenue.Growth..per.Share.             0.05167    0.05004   1.032  0.30184
## X5Y.Operating.CF.Growth..per.Share.        0.10032    0.05085   1.973  0.04850
## X5Y.Net.Income.Growth..per.Share.          0.20464    0.06455   3.170  0.00152
## X3Y.Net.Income.Growth..per.Share.          0.15462    0.06731   2.297  0.02161
## X5Y.Dividend.per.Share.Growth..per.Share.  0.02690    0.05381   0.500  0.61714
## X3Y.Dividend.per.Share.Growth..per.Share.  0.05122    0.05959   0.860  0.39002
## Asset.Growth                               0.37639    0.44208   0.851  0.39454
## Debt.Growth                                0.10171    0.08988   1.132  0.25778
## SG.A.Expenses.Growth                      -0.18644    0.09167  -2.034  0.04198
##
## (Intercept)                               ***
## Income.Tax.Expense
## Net.Income...Non.Controlling.int
## Preferred.Dividends
## EPS.Diluted
## Weighted.Average.Shs.Out                  **
## Dividend.per.Share                        ***
## Gross.Margin                              *
## Profit.Margin
## EBIT
## Cash.and.cash.equivalents                 .
## Cash.and.short.term.investments
## Long.term.debt
## Deferred.revenue
## priceEarningsRatio                        **
## priceToFreeCashFlowsRatio                 *
## dividendYield
## payablesTurnover                          *
## inventoryTurnover
## fixedAssetTurnover
## assetTurnover                             **
## cashRatio
## debtEquityRatio
## longtermDebtToCapitalization              .
## interestCoverage                          **
## operatingCashFlowPerShare
## freeCashFlowPerShare
## Operating.Cash.Flow.per.Share
## Free.Cash.Flow.per.Share
## Market.Cap
```

```
## Enterprise.Value
## PE.ratio
## Enterprise.Value.over.EBITDA
## EV.to.Free.cash.flow                        *
## Earnings.Yield
## Debt.to.Equity
## Interest.Coverage
## Dividend.Yield                              *
## Intangibles.to.Total.Assets
## Capex.to.Revenue
## Capex.to.Depreciation
## Graham.Net.Net
## Net.Current.Asset.Value                     .
## Average.Inventory
## Payables.Turnover
## Inventory.Turnover
## Operating.Income.Growth
## EPS.Growth                                  *
## Weighted.Average.Shares.Diluted.Growth      **
## Dividends.per.Share.Growth                  **
## Operating.Cash.Flow.growth
## Free.Cash.Flow.growth
## X5Y.Revenue.Growth..per.Share.
## X5Y.Operating.CF.Growth..per.Share.         *
## X5Y.Net.Income.Growth..per.Share.           **
## X3Y.Net.Income.Growth..per.Share.           *
## X5Y.Dividend.per.Share.Growth..per.Share.
## X3Y.Dividend.per.Share.Growth..per.Share.
## Asset.Growth
## Debt.Growth
## SG.A.Expenses.Growth                        *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3791.6  on 3075  degrees of freedom
## Residual deviance: 3354.3  on 3022  degrees of freedom
## AIC: 3462.3
##
## Number of Fisher Scoring iterations: 12
```

The logistic regression model identifies several key financial predictors that significantly influence whether a stock is classified as "buy-worthy." Notably, dividend per share emerged as a highly significant variable, suggesting that investors favor companies providing consistent income returns. Similarly, price-to-earnings ratio and enterprise value to sales indicate that valuation metrics play a crucial role in stock selection.

Profitability indicators such as pre-tax profit margin and EPS diluted growth reflect strong earnings potential and efficiency, further enhancing a stock's appeal. The model also highlights the importance of financial structure: companies with lower net debt and strong interest coverage are perceived as less risky, making them more attractive to investors. Additionally, operational efficiency metrics such as SG&A to revenue ratio and days of inventory outstanding reveal that cost control and inventory management are relevant to stock performance. The significance of tangible book value per share and 5-year net income per share growth emphasizes the market's preference for companies with both real asset backing and sustained long-term growth.

In terms of statistical significance, several expected financial predictors are significant at the 0.05 level, including `Revenue`, `Gross.Profit`, `Operating.Expenses`, and `Net.Debt`, which reflect a company's core financial health and operational efficiency. Additionally, `Tangible.Book.Value.per.Share` and `Dividend.per.Share` are highly significant at the 0.001 level, reinforcing the importance of tangible value and direct shareholder returns in identifying stocks with strong buy potential.

```
#Predict
log_probs <- predict(log_model, newdata = filtered_test, type = "prob")[,2]
# Covert to class labels
log_pred <- ifelse(log_probs > 0.5, "1", "0")
log_pred <- factor(log_pred, levels = levels(test_glm$Class))
#Confusion matrix
log_cm <- confusionMatrix(log_pred, test_glm$Class)
log_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  61  48
##          1 342 865
##
##                Accuracy : 0.7036
##                  95% CI : (0.6782, 0.7282)
##     No Information Rate : 0.6938
##     P-Value [Acc > NIR] : 0.2279
##
##                   Kappa : 0.1241
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.15136
##             Specificity : 0.94743
##          Pos Pred Value : 0.55963
##          Neg Pred Value : 0.71665
##              Prevalence : 0.30623
##          Detection Rate : 0.04635
##    Detection Prevalence : 0.08283
##       Balanced Accuracy : 0.54940
##
##        'Positive' Class : 0
##
```

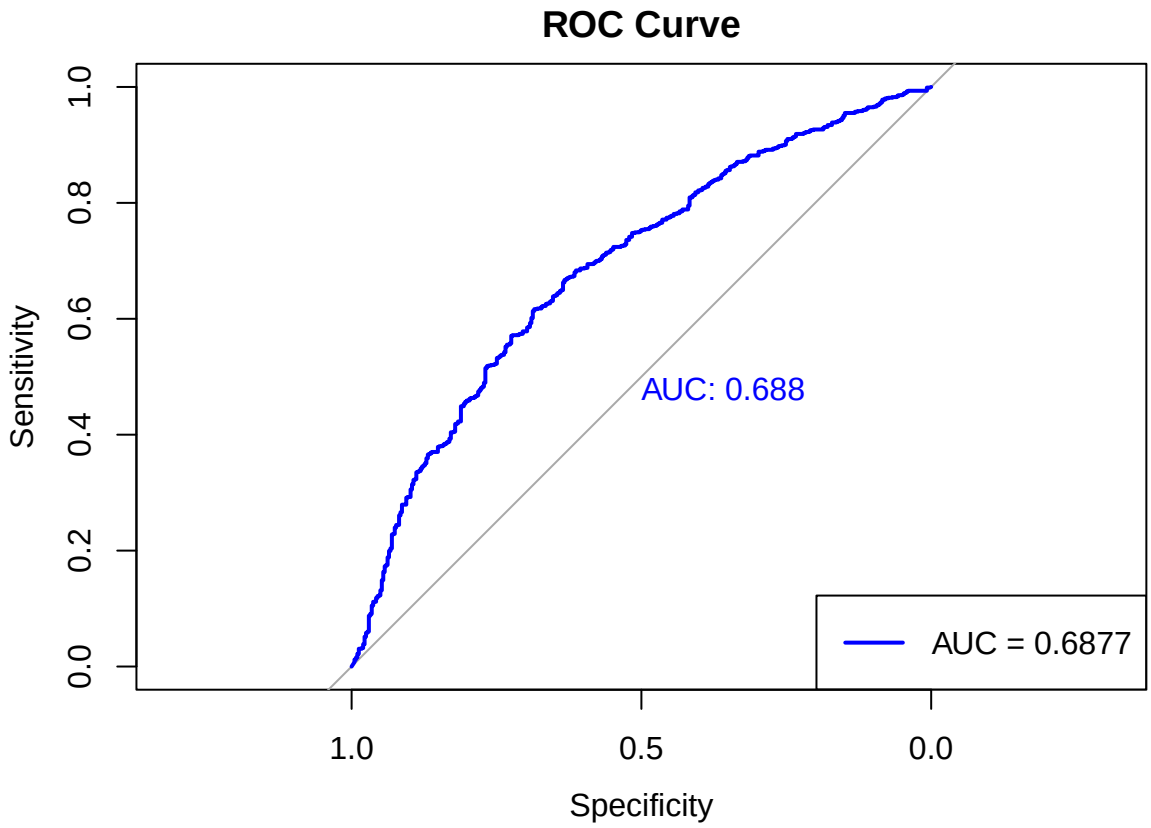
The logistic regression model achieved an accuracy of 70.36%, only slightly higher than the No Information Rate (NIR) of 69.38%, which represents the accuracy of always predicting the majority class. The associated p-value (0.2279) suggests that this improvement is not statistically significant, indicating that the model may not perform much better than a naive baseline.

```
log_probs_1 <- predict(log_model, newdata = filtered_test, type = "prob")[,2]
#ROC curves
roc_log <- roc(filtered_test$Class, log_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_log,col = "blue", main = "ROC Curve", print.auc = TRUE)
auc_value_log <- auc(roc_log)
legend("bottomright", legend = paste("AUC =", round(auc_value_log, 4)), col = "blue", lwd = 2)
```



The ROC curve suggests that the logistic model performs only moderately better than random guessing, with an AUC of 0.688. While it shows some predictive power, its ability to separate classes is limited, indicating room for improvement through further tuning or more flexible models.

```
log_plot_data <- data.frame(TrueClass = filtered_test$Class,
                        PredictedClass = log_pred)

log_plot_data$Correct <- ifelse(log_plot_data$TrueClass ==
                                  log_plot_data$PredictedClass, "Correct", "Incorrect")

# plot
ggplot(log_plot_data, aes(x = TrueClass, y = PredictedClass, color = Correct)) +
  geom_jitter(width = 0.2, height = 0.2, alpha = 0.7) +
  labs(title = "Scatter Plot of Predictions (Logistic Model)",
       x = "True Class", y = "Predicted Class") +
  theme_light() +
  scale_color_manual(values = c("green", "red"))
```

As shown in the plot, Logistic mode perform consistently well with `Class = 1` but fail to do the same for `Class = 0`.

LDA (Linear Discriminant Analysis)

```
set.seed(123)
lda_model <- train(Class ~ ., data = filtered_train,
                   method = "lda",
                   preProcess = "pca",# Reduce collinearity
                   trControl = ctrl)
lda_model
```

```
## Linear Discriminant Analysis
##
## 3076 samples
##   60 predictor
##    2 classes: '0', '1'
##
## Pre-processing: principal component signal extraction (60), centered
##  (60), scaled (60)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2768, 2769, 2769, 2769, 2768, 2769, ...
## Resampling results:
```

```
##
##   Accuracy   Kappa
##   0.6986336  0.06917519
```

After applying PCA to reduce multicollinearity, the LDA model with 10-fold CV achieved 69.86% accuracy and a Kappa of 0.069. While accuracy is decent, the low Kappa indicates that the model might be struggle to classify the minority class, suggesting LDA may not capture the data's complexity well.

```
lda_probs <- predict(lda_model,filtered_test)
lda_cm <- confusionMatrix(lda_probs,filtered_test$Class)
lda_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  34  26
##          1 369 887
##
##                Accuracy : 0.6998
##                  95% CI : (0.6743, 0.7245)
##     No Information Rate : 0.6938
##     P-Value [Acc > NIR] : 0.328
##
##                   Kappa : 0.0733
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.08437
##             Specificity : 0.97152
##          Pos Pred Value : 0.56667
##          Neg Pred Value : 0.70621
##              Prevalence : 0.30623
##          Detection Rate : 0.02584
##    Detection Prevalence : 0.04559
##       Balanced Accuracy : 0.52794
##
##        'Positive' Class : 0
##
```

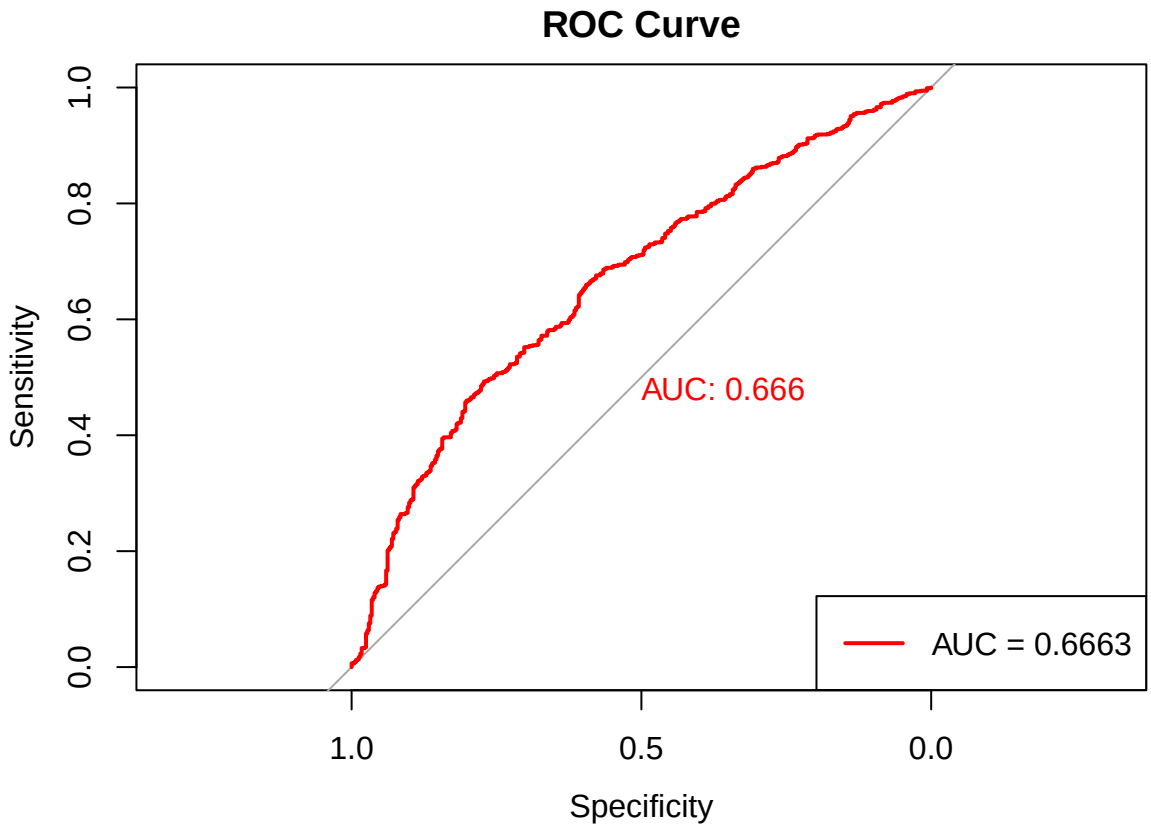
The LDA model achieved a test accuracy of 69.98%, which is only marginally better than the No Information Rate of 69.38%. The p-value of 0.328 further suggests that this improvement is not statistically significant. Overall, LDA performs only slightly better than the logistic regression model, which had an accuracy of 70.36%

```
lda_probs_1 <- predict(lda_model, filtered_test, type = "prob")[,2]
roc_lda <- roc(filtered_test$Class, lda_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_lda,col = "red", main = "ROC Curve", print.auc = TRUE)
auc_value_lda <- auc(roc_lda)
legend("bottomright", legend = paste("AUC =", round(auc_value_lda, 4)), col = "red", lwd = 2)
```

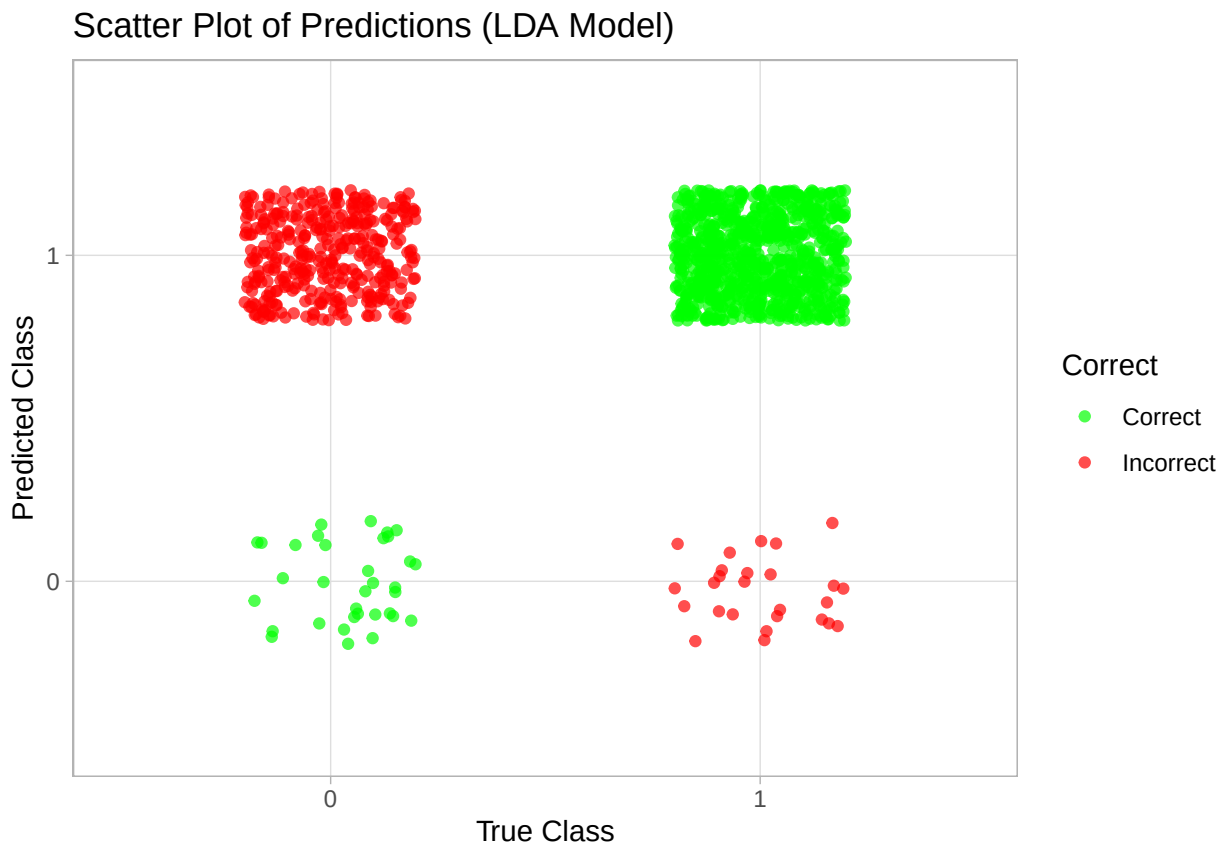


Again, the AUC of 0.666 suggests that the LDA model only slightly better than random guessing.

```r
lda_plot_data <- data.frame(TrueClass = filtered_test$Class,
                            PredictedClass = lda_probs)

lda_plot_data$Correct <- ifelse(lda_plot_data$TrueClass == lda_plot_data$PredictedClass, "Correct", "In

# Scatter plot
ggplot(lda_plot_data, aes(x = TrueClass, y = PredictedClass, color = Correct)) +
  geom_jitter(width = 0.2, height = 0.2, alpha = 0.7) +
  labs(title = "Scatter Plot of Predictions (LDA Model)",
       x = "True Class", y = "Predicted Class") +
  theme_light() +
  scale_color_manual(values = c("green", "red"))
```



The plot shows that while the LDA model performs well on `Class = 1`, it consistently struggles to identify `Class = 0`.

QDA (Quadratic Discriminant Analysis)

```
set.seed(123)
qda_model <- train(Class ~ ., data = filtered_train,
                   method = "qda",
                   preProcess = "pca",
                   trControl = ctrl)
qda_model
```

```
## Quadratic Discriminant Analysis
##
## 3076 samples
##   60 predictor
##    2 classes: '0', '1'
##
## Pre-processing: principal component signal extraction (60), centered
##  (60), scaled (60)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2768, 2769, 2769, 2769, 2768, 2769, ...
## Resampling results:
```

```
##
##   Accuracy   Kappa
##   0.6911693  0.07603995
```

The QDA model achieved an accuracy of 69.11% and a kappa of 0.070 based on the training set. This performance is slightly lower than the LDA model (accuracy = 69.99%), suggesting that the added flexibility of QDA may not improve predictive power significantly for this dataset.

```
qda_probs <- predict(qda_model,filtered_test)
qda_cm <- confusionMatrix(qda_probs,filtered_test$Class)
qda_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  42  49
##          1 361 864
##
##                Accuracy : 0.6884
##                  95% CI : (0.6626, 0.7134)
##     No Information Rate : 0.6938
##     P-Value [Acc > NIR] : 0.6742
##
##                   Kappa : 0.0645
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.10422
##             Specificity : 0.94633
##          Pos Pred Value : 0.46154
##          Neg Pred Value : 0.70531
##              Prevalence : 0.30623
##          Detection Rate : 0.03191
##    Detection Prevalence : 0.06915
##       Balanced Accuracy : 0.52527
##
##        'Positive' Class : 0
##
```

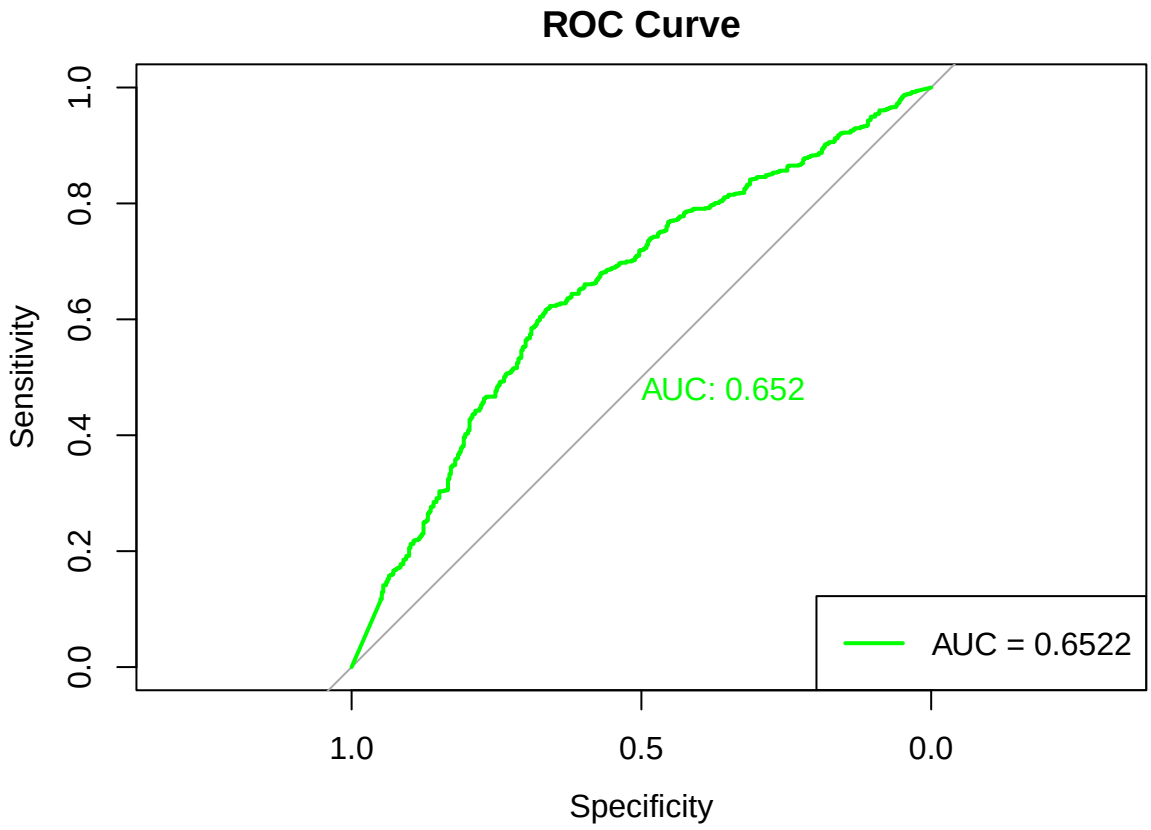
The QDA model achieved an accuracy of 68.84%, slightly lower than the No Information Rate of 69.38%, with a p-value of 0.6742 on test set. Similar to LDA, it struggles to classify the minority class, as indicated by a low Kappa value and high p-value, and offers no significant improvement over baseline guessing.

```
qda_probs_1 <-  predict(qda_model, filtered_test, type = "prob")[,2]
roc_qda <- roc(filtered_test$Class, qda_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_qda,col = "green", main = "ROC Curve", print.auc = TRUE)
auc_value_qda <- auc(roc_qda)
legend("bottomright", legend = paste("AUC =", round(auc_value_qda, 4)), col = "green", lwd = 2)
```



The limitation of the QDA model is evident in the ROC curve, with an AUC of 65.22%, only slightly above the 50% baseline of random guessing.

```r
qda_plot_data <- data.frame(TrueClass = filtered_test$Class,
                            PredictedClass = qda_probs)

qda_plot_data$Correct <- ifelse(qda_plot_data$TrueClass == qda_plot_data$PredictedClass, "Correct", "In

# Scatter plot
ggplot(qda_plot_data, aes(x = TrueClass, y = PredictedClass, color = Correct)) +
  geom_jitter(width = 0.2, height = 0.2, alpha = 0.7) +
  labs(title = "Scatter Plot of Predictions (QDA Model)",
       x = "True Class", y = "Predicted Class") +
  theme_light() +
  scale_color_manual(values = c("green", "red"))
```

This plot shows that QDA model perform extremely well with `Class = 1` while struggling to classify `Class = 0` correctly.

```
set.seed(123)
knn_model <- train(Class ~ ., data = filtered_train, method = "knn",
      tuneLength = 20,
      trControl = ctrl)
knn_model
```

```
## k-Nearest Neighbors
##
## 3076 samples
##   60 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2768, 2769, 2769, 2769, 2768, 2769, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.6869506  0.2159463
##    7  0.6957263  0.2301666
##    9  0.6940871  0.2217222
##   11  0.6966961  0.2248223
##   13  0.6996246  0.2293216
```

```
##   15  0.7018952  0.2352599
##   17  0.7012501  0.2346671
##   19  0.6999503  0.2318601
##   21  0.7022230  0.2347357
##   23  0.6976755  0.2211177
##   25  0.6999471  0.2244808
##   27  0.7025435  0.2275894
##   29  0.7025477  0.2276029
##   31  0.7048215  0.2303861
##   33  0.7012532  0.2208095
##   35  0.7015768  0.2183364
##   37  0.7032055  0.2213774
##   39  0.7041753  0.2236716
##   41  0.7054814  0.2259274
##   43  0.7041795  0.2206046
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 41.
```

```
knn_model$bestTune
```

```
##     k
## 19 41
```

The best k is 41 which produce the highest accuracy of 70.55% with Kappa of 0.226 thus then used in KNN model.

```
knn_probs <- predict(knn_model,filtered_test)
knn_cm <- confusionMatrix(knn_probs,filtered_test$Class)
knn_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 135 147
##          1 268 766
##
##                Accuracy : 0.6847
##                  95% CI : (0.6588, 0.7097)
##     No Information Rate : 0.6938
##     P-Value [Acc > NIR] : 0.7731
##
##                   Kappa : 0.1899
##
##  Mcnemar's Test P-Value : 3.849e-09
##
##             Sensitivity : 0.3350
##             Specificity : 0.8390
##          Pos Pred Value : 0.4787
##          Neg Pred Value : 0.7408
##              Prevalence : 0.3062
##          Detection Rate : 0.1026
##    Detection Prevalence : 0.2143
```

```
##       Balanced Accuracy : 0.5870
##
##        'Positive' Class : 0
##
```

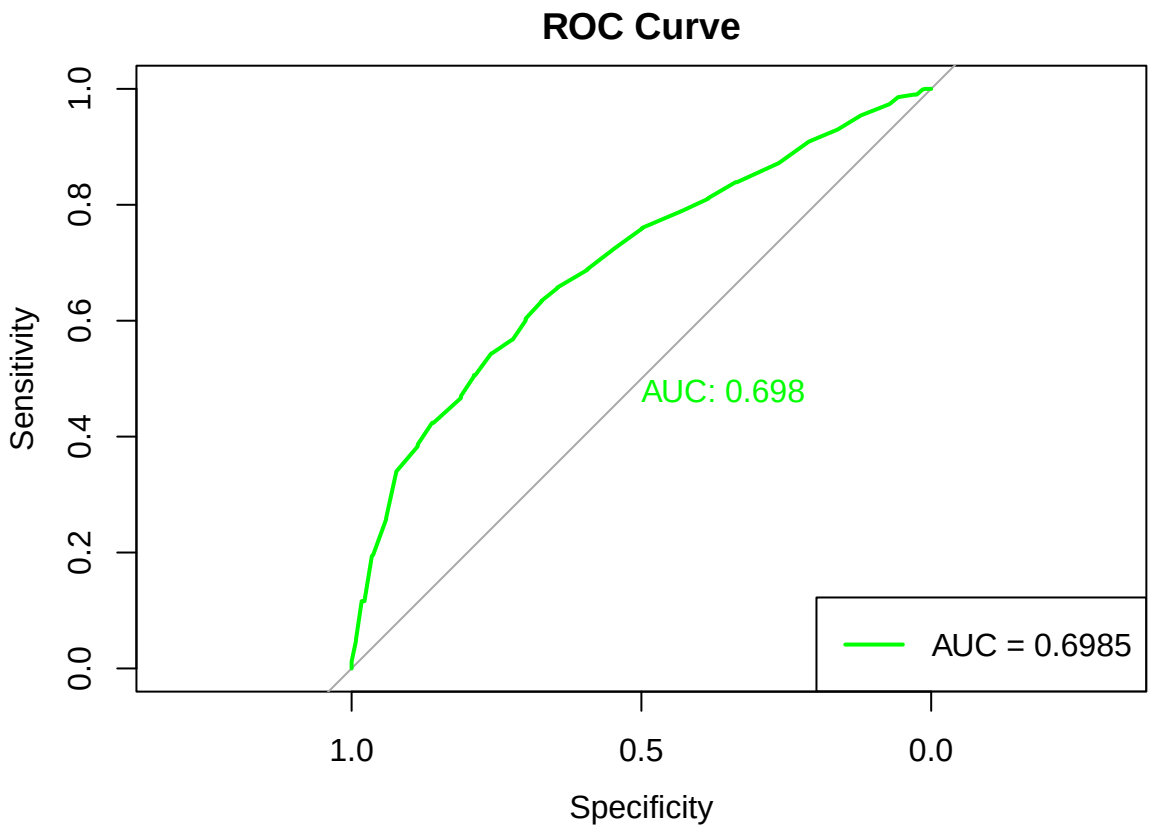
Even with the filtered dataset reduced to 45 predictors from over 200, the kNN model likely still suffers from the curse of dimensionality. Its test accuracy of 68.47% slightly higher than the No Information Rate of 69.38%, and the high p-value (0.7731) indicates that the model performs no better than random guessing.

```
knn_probs_1 <- predict(knn_model, filtered_test, type = "prob")[,2]
roc_knn <- roc(filtered_test$Class, knn_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_knn,col = "green", main = "ROC Curve", print.auc = TRUE)
auc_value_knn <- auc(roc_knn)
legend("bottomright", legend = paste("AUC =", round(auc_value_knn, 4)), col = "green", lwd = 2)
```



The ROC curve also shows that the KNN model, AUC of 0.699, is only a bit above the baseline of random guessing.

```
knn_plot_data <- data.frame(TrueClass = filtered_test$Class,
                            PredictedClass = knn_probs)
```

```
knn_plot_data$Correct <- ifelse(knn_plot_data$TrueClass == knn_plot_data$PredictedClass, "Correct", "In

# Scatter plot
ggplot(knn_plot_data, aes(x = TrueClass, y = PredictedClass, color = Correct)) +
  geom_jitter(width = 0.2, height = 0.2, alpha = 0.7) +
  labs(title = "Scatter Plot of Predictions (KNN Model)",
       x = "True Class", y = "Predicted Class") +
  theme_light() +
  scale_color_manual(values = c("green", "red"))
```

The scatter plot shows that while the KNN model reliably predicts `Class = 1`, it performs poorly on `Class = 0`, misclassifying many observations. This indicates a strong bias toward predicting the majority class and highlights the model's weakness in handling class imbalance.

```
# Comparing all models accuracy and Kappa
model_results <- data.frame(
  Model = c("Logistic Regression", "LDA", "QDA", "KNN"),
  Accuracy = c(log_cm$overall["Accuracy"], lda_cm$overall["Accuracy"], qda_cm$overall["Accuracy"], knn_
  Kappa = c(log_cm$overall["Kappa"], lda_cm$overall["Kappa"], qda_cm$overall["Kappa"], knn_cm$overall["
    P_Value = c(log_cm$overall["AccuracyPValue"], lda_cm$overall["AccuracyPValue"], qda_cm$overall["Accu
)
model_results <- model_results[order(model_results$Accuracy,decreasing = TRUE), ]
model_results
```

```
##                 Model  Accuracy      Kappa   P_Value
```

```
## 1 Logistic Regression 0.7036474 0.12407115 0.2278806
## 2                 LDA 0.6998480 0.07331874 0.3280057
## 3                 QDA 0.6884498 0.06449499 0.6742219
## 4                 KNN 0.6846505 0.18990519 0.7731485
```

Since Logistic and LDA models perform slightly better than KNN and QDA models, linear models would be more appropriate in this case but non perform strongly overall.

Conclusion

The models attempt to predict future stock movements based solely on financial indicators available at the beginning of 2018, making the analysis realistic and economically meaningful. Correctly identifying stocks likely to rise would enable investors to make profitable buy decisions, directly impacting portfolio returns.

However, the relatively low AUC and low Kappa values across models suggest that predictive power is limited, meaning financial indicators alone may not be sufficient for reliable investment decisions. This reflects real-world challenges, as markets are influenced by unpredictable factors beyond what financial ratios capture.

In addition, the models' struggles to classify the minority class highlight a major limitation. Using models better suited for imbalanced data may improve prediction accuracy and better capture important stock volatility and movements

Overall, while the current models provide some guidance, their practical economic benefit is limited without further model improvements or incorporating more economic signals.