HarvardX Data Science Capstone Project

Predicting bankruptcy with machine learning

Stefan Kälin

Introduction

Bankruptcy prediction is a valuable skill in in the financial industry. Suppose you are in charge of a bank lending out credits to companies. Knowing in advanced which companies will default and not be able to pay back the loan, is an information of enormous value. Up until now, to evaluate which businesses might go bankrupt, a bank has to employ credit experts. They check the financial statements and based on their financial understanding and their experience decided which companies get a credit and which are likely to default and thus do not receive the funds. However, this credit experts need a formal education and a proper understanding of the financial reporting to make their decisions. This is costly and they do not work for free. So, wouldn't it be nice to have a computer doing this job? The goal of this project is to use different machine leaning algorithms to predict whether a company will go bankrupt. It is not the aim of this project to build a perfect model which predicts all bankruptcies perfectly, but instead to find out if it is in general possible and if machine learning can provide an added value.

The following report is structured in different sections:

- Set up
- Data exploration
- Analysis
- Evaluation
- Conclusion

Set up

The data used throughout this project is publicly available at:

https://www.kaggle.com/fedesoriano/company-bankruptcy-prediction

According to the data source: "The data were collected from the Taiwan Economic Journal for the years 1999 to 2009. Company bankruptcy was defined based on the business regulations of the Taiwan Stock Exchange."

The data set contains one output feature (Y) and 95 input features (X). It has the following variables:

- Y - Bankrupt?: Class label
- X1 - ROA(C) before interest and depreciation before interest: Return On Total Assets(C)
- X2 - ROA(A) before interest and % after tax: Return On Total Assets(A)
- X3 - ROA(B) before interest and depreciation after tax: Return On Total Assets(B)
- X4 - Operating Gross Margin: Gross Profit/Net Sales
- X5 - Realized Sales Gross Margin: Realized Gross Profit/Net Sales
- X6 - Operating Profit Rate: Operating Income/Net Sales
- X7 - Pre-tax net Interest Rate: Pre-Tax Income/Net Sales
- X8 - After-tax net Interest Rate: Net Income/Net Sales

- X9 - Non-industry income and expenditure/revenue: Net Non-operating Income Ratio
- X10 - Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales
- X11 - Operating Expense Rate: Operating Expenses/Net Sales
- X12 - Research and development expense rate: (Research and Development Expenses)/Net Sales
- X13 - Cash flow rate: Cash Flow from Operating/Current Liabilities
- X14 - Interest-bearing debt interest rate: Interest-bearing Debt/Equity
- X15 - Tax rate (A): Effective Tax Rate
- X16 - Net Value Per Share (B): Book Value Per Share(B)
- X17 - Net Value Per Share (A): Book Value Per Share(A)
- X18 - Net Value Per Share (C): Book Value Per Share(C)
- X19 - Persistent EPS in the Last Four Seasons: EPS-Net Income
- X20 - Cash Flow Per Share
- X21 - Revenue Per Share (Yuan ¥): Sales Per Share
- X22 - Operating Profit Per Share (Yuan ¥): Operating Income Per Share
- X23 - Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share
- X24 - Realized Sales Gross Profit Growth Rate
- X25 - Operating Profit Growth Rate: Operating Income Growth
- X26 - After-tax Net Profit Growth Rate: Net Income Growth
- X27 - Regular Net Profit Growth Rate: Continuing Operating Income after Tax Growth
- X28 - Continuous Net Profit Growth Rate: Net Income-Excluding Disposal Gain or Loss Growth
- X29 - Total Asset Growth Rate: Total Asset Growth
- X30 - Net Value Growth Rate: Total Equity Growth
- X31 - Total Asset Return Growth Rate Ratio: Return on Total Asset Growth
- X32 - Cash Reinvestment %: Cash Reinvestment Ratio
- X33 - Current Ratio
- X34 - Quick Ratio: Acid Test
- X35 - Interest Expense Ratio: Interest Expenses/Total Revenue
- X36 - Total debt/Total net worth: Total Liability/Equity Ratio
- X37 - Debt ratio %: Liability/Total Assets
- X38 - Net worth/Assets: Equity/Total Assets
- X39 - Long-term fund suitability ratio (A): (Long-term Liability+Equity)/Fixed Assets
- X40 - Borrowing dependency: Cost of Interest-bearing Debt
- X41 - Contingent liabilities/Net worth: Contingent Liability/Equity
- X42 - Operating profit/Paid-in capital: Operating Income/Capital
- X43 - Net profit before tax/Paid-in capital: Pretax Income/Capital
- X44 - Inventory and accounts receivable/Net value: (Inventory+Accounts Receivables)/Equity
- X45 - Total Asset Turnover
- X46 - Accounts Receivable Turnover
- X47 - Average Collection Days: Days Receivable Outstanding
- X48 - Inventory Turnover Rate (times)
- X49 - Fixed Assets Turnover Frequency
- X50 - Net Worth Turnover Rate (times): Equity Turnover
- X51 - Revenue per person: Sales Per Employee
- X52 - Operating profit per person: Operation Income Per Employee
- X53 - Allocation rate per person: Fixed Assets Per Employee
- X54 - Working Capital to Total Assets
- X55 - Quick Assets/Total Assets
- X56 - Current Assets/Total Assets
- X57 - Cash/Total Assets
- X58 - Quick Assets/Current Liability
- X59 - Cash/Current Liability
- X60 - Current Liability to Assets
- X61 - Operating Funds to Liability
- X62 - Inventory/Working Capital

- X63 - Inventory/Current Liability
- X64 - Current Liabilities/Liability
- X65 - Working Capital/Equity
- X66 - Current Liabilities/Equity
- X67 - Long-term Liability to Current Assets
- X68 - Retained Earnings to Total Assets
- X69 - Total income/Total expense
- X70 - Total expense/Assets
- X71 - Current Asset Turnover Rate: Current Assets to Sales
- X72 - Quick Asset Turnover Rate: Quick Assets to Sales
- X73 - Working capitcal Turnover Rate: Working Capital to Sales
- X74 - Cash Turnover Rate: Cash to Sales
- X75 - Cash Flow to Sales
- X76 - Fixed Assets to Assets
- X77 - Current Liability to Liability
- X78 - Current Liability to Equity
- X79 - Equity to Long-term Liability
- X80 - Cash Flow to Total Assets
- X81 - Cash Flow to Liability
- X82 - CFO to Assets
- X83 - Cash Flow to Equity
- X84 - Current Liability to Current Assets
- X85 - Liability-Assets Flag: 1 if Total Liability exceeds Total Assets, 0 otherwise
- X86 - Net Income to Total Assets
- X87 - Total assets to GNP price
- X88 - No-credit Interval
- X89 - Gross Profit to Sales
- X90 - Net Income to Stockholder's Equity
- X91 - Liability to Equity
- X92 - Degree of Financial Leverage (DFL)
- X93 - Interest Coverage Ratio (Interest expense to EBIT)
- X94 - Net Income Flag: 1 if Net Income is Negative for the last two years, 0 otherwise
- X95 - Equity to Liability

First of all I am checking and installing the required libraries as well as downloading the data set. The CSV file is saved on my personal GitHub for convenience reasons, so I do not have to fiddle around with the kaggle API. As a first step afterwards, I do some clean up of the data. Namely, I rename the variables according to the above list to Y, X1, X2, etc. Moreover, the output feature Y has to be converted to a factor in order for the machine learning algorithms to work correctly. Finally, I also check whether there are any missing values / NAs in the data set. But fortunately there are none.

With the raw data ready, as a next step I split the data into a training and a testing data set. As a ratio I chose 70/30, meaning the training set will be 70% of the raw data, while the test set is 30%. Why exactly this ratio? I took inspiration from the following analysis:

https://hrcak.srce.hr/file/375100

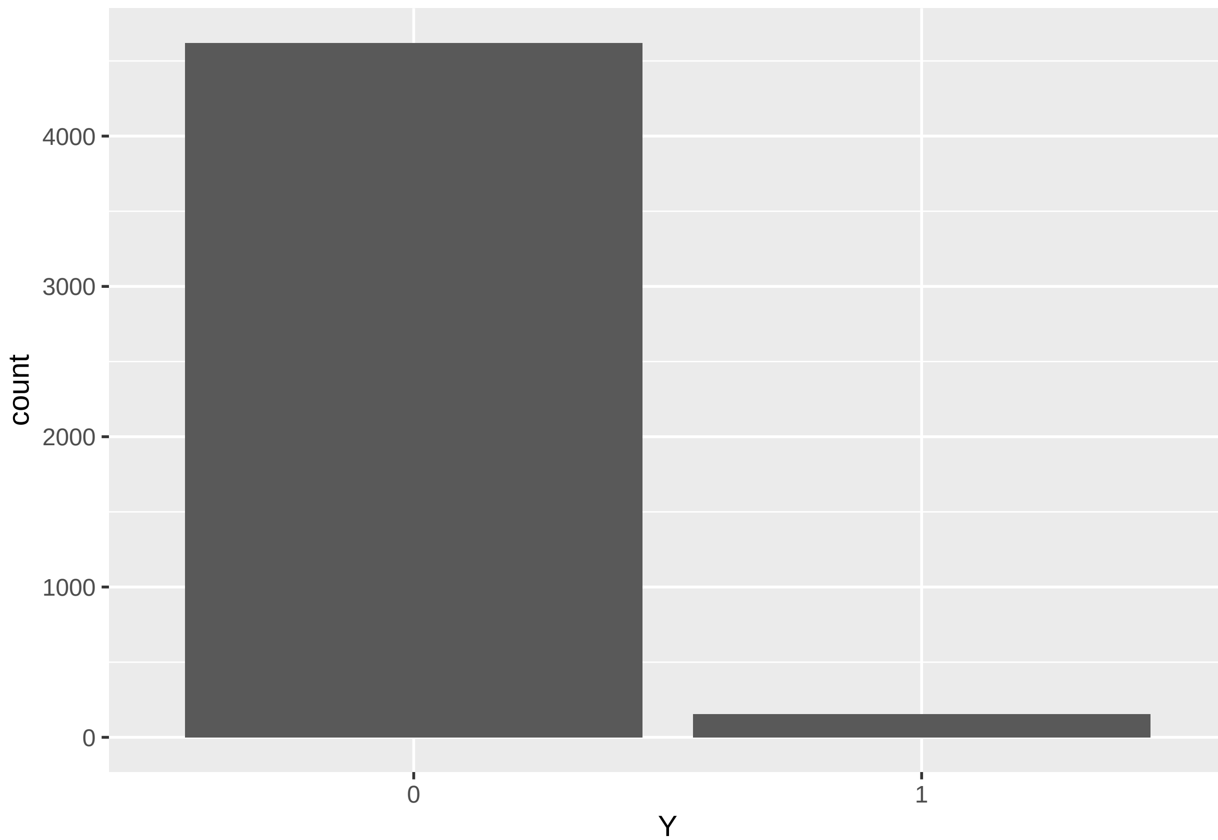
The author of the article, Borislava Vrigazova tested out different ratios for splitting data into training and test set in classification problems regarding the performance and the accuracy. In conclusion, a train/test splitting ratio of 70/30 is suitable.

Data exploration

As part of the data exploration I take a look at the variable Y in the training set. A value of 1 means that the company went bankrupt, while a value of 0 means it survived. By calculating the mean of Y I receive the proportion of bankrupt companies in the data set:

```
## [1] 0.03225806
```

Moreover, to visualize the distribution, I create a bar chart:

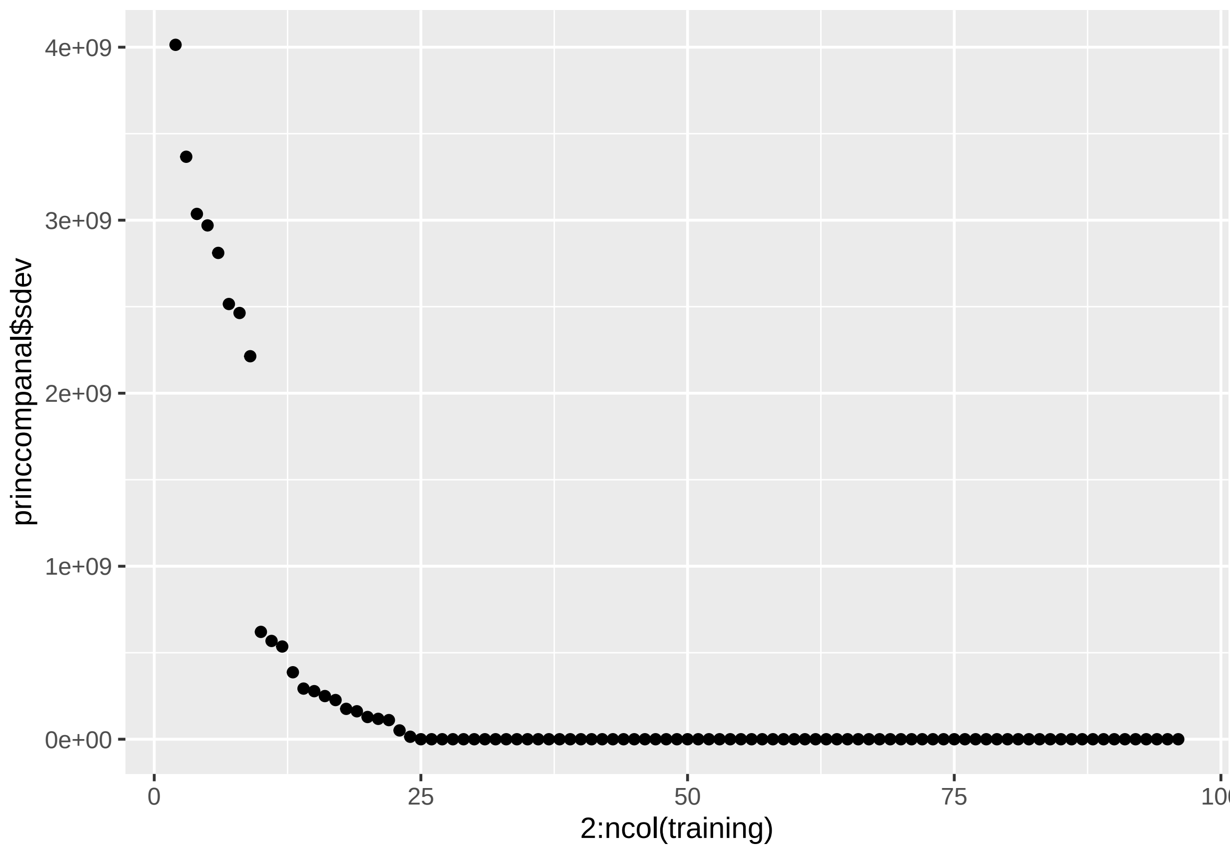


As one can see, most companies do in fact survive, only about 3% of the companies go bankrupt.

Taking a look at all the input features X1 to X95 individually would go beyond the scope of this project and would not contribute significantly to the analysis. Instead, I do a principal component analysis (PCA) as learned in the course:

https://rafalab.github.io/dsbook/large-datasets.html#pca

The following plot shows how much variability is captured by adding more features:



According to the plot, after 8 features the significance of each additional feature drops greatly.

Analysis

Now I am ready to start the actual analysis and apply the different machine learning algorithms to the training data set. The first model is a naive approach. As shown in the previous data exploration section, only about 3% of companies go bankrupt. Thus the simplest model possible is to predict, that no company goes bankrupt. Therefore, in most cases this guess would be correct. By applying this to the test set, I achieve the following:

```
predBase <- factor(array(0, c(length(testing$Y),1)), levels = c(0,1))
confusionMatrix(testing$Y, predBase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1979    0
##          1   66    0
##
##                Accuracy : 0.9677
##                  95% CI : (0.9591, 0.975)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
```

```
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : 1.235e-15
##
##            Sensitivity : 0.9677
##            Specificity :     NA
##         Pos Pred Value :     NA
##         Neg Pred Value :     NA
##             Prevalence : 1.0000
##         Detection Rate : 0.9677
##   Detection Prevalence : 0.9677
##      Balanced Accuracy :     NA
##
##       'Positive' Class : 0
##
```

The accuracy is 0.9677. This is the baseline on which I have to improve with the following models. If I am not able to achieve this accuracy with the machine learning algorithms, then machine learning is useless for predicting bankruptcies.

Learning Vector Quantization (LVQ)

In the data exploration section it was shown by using a principal component analysis (PCA) that not all features are equally important. Instead of using all 95 variable I narrow it down to the most important ones. An expert in financial analysis or financial reporting would probably know by experience which of the financial indicators are relevant for the bankruptcy. Credit experts can select the features conceptually. On the other hand, I will also rely on machine learning to select the appropriate features. I took some inspiration from the following article:

https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/

According to the author, one can use the learning vector quantization algorithm to train a model using all 95 features. Then, to check which variables are the most important ones, I use the following function:

```r
importance <- varImp(fitlvq)
```

By printing it one receives a sorted list of the most important variables. The most important ones are on top in descending order. Here are the top 20:

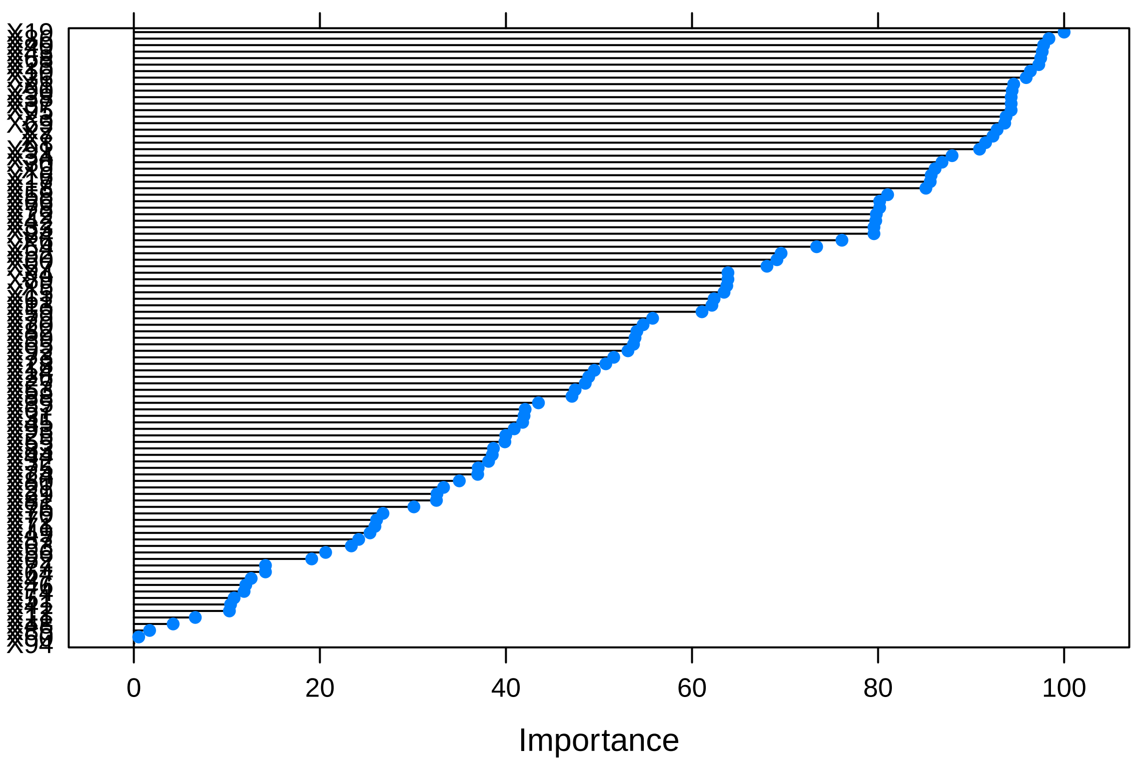
```r
print(importance)
```

```
## ROC curve variable importance
##
##   only 20 most important variables shown (out of 95)
##
##     Importance
## X19     100.00
## X86      98.37
## X40      97.79
## X43      97.64
## X68      97.48
## X23      97.28
## X10      96.37
```

```
## X36      95.92
## X1       94.58
## X90      94.41
## X38      94.31
## X37      94.31
## X95      94.31
## X3       93.75
## X69      93.62
## X2       92.79
## X7       92.35
## X8       91.56
## X91      90.91
## X34      87.96
```

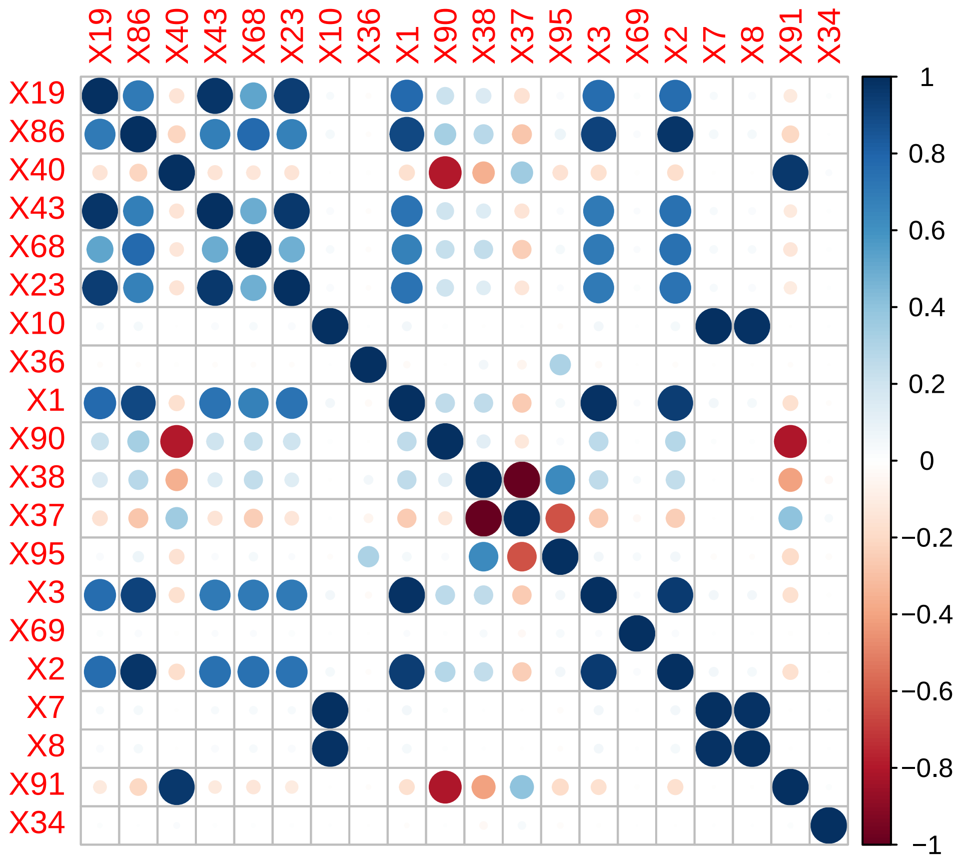
One can also plot the importance to visualize it:

```
plot(importance)
```

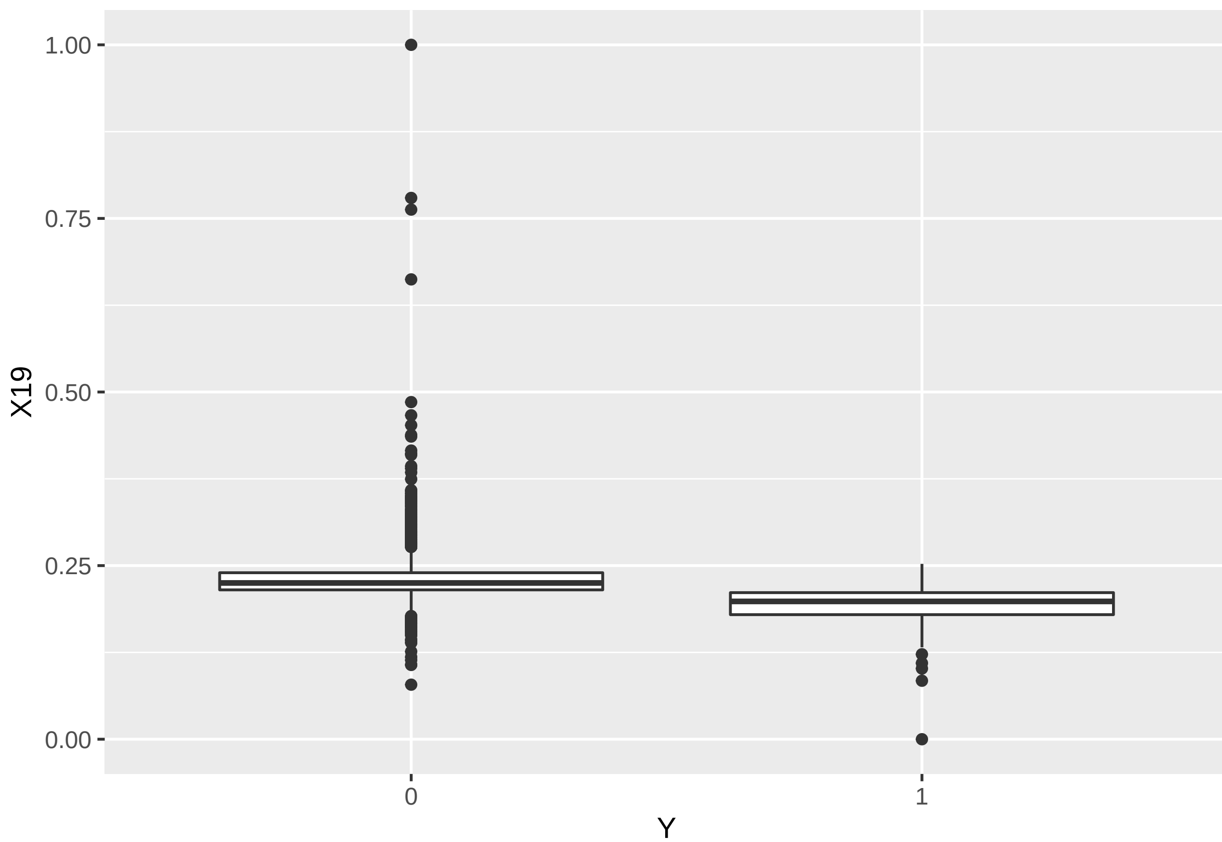


The correlation of the top 20 most important features can be shown like this:

```
corrplot(varCor)
```

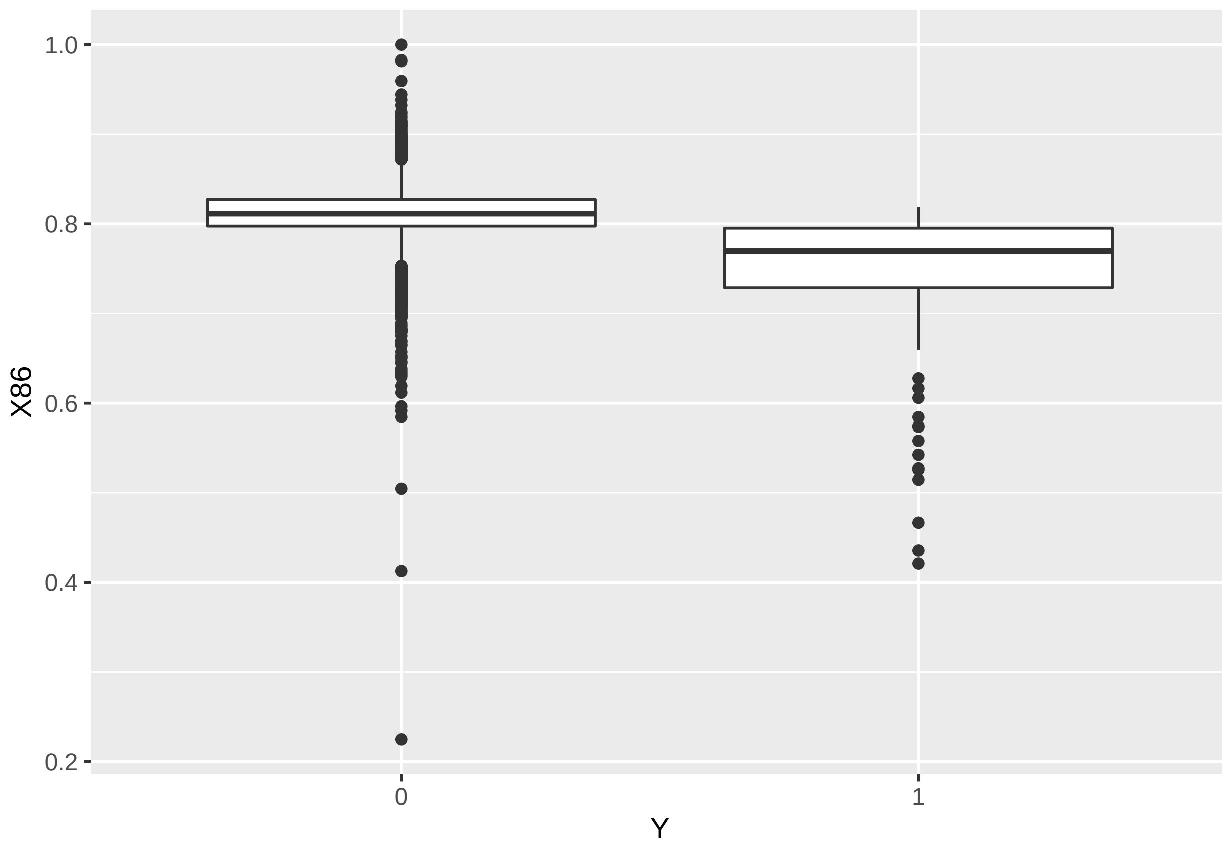


Let's examine some of the of the most significant features more in detail. The most important feature is X19. By plotting X19 against Y one can see the following picture:



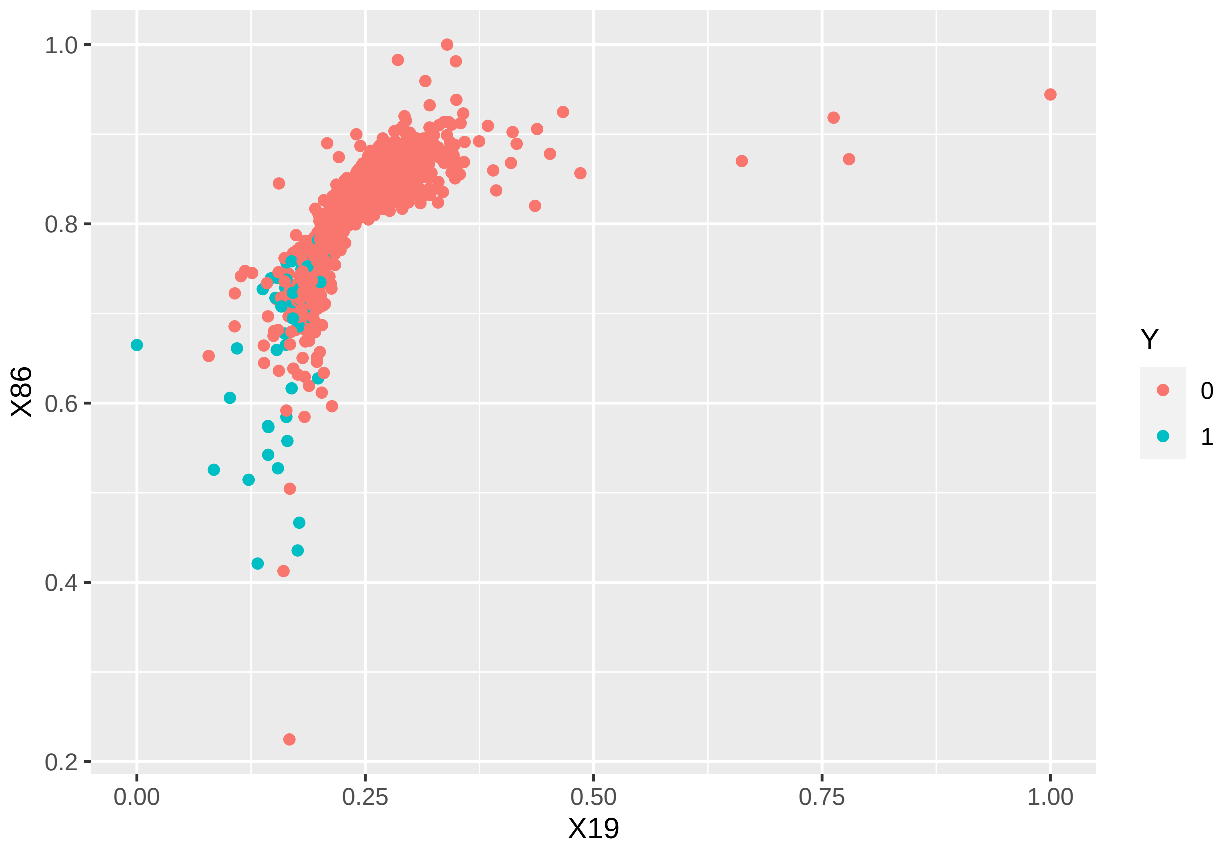
All companies that went bankrupt had an X19 value lower than 0.25. Vice versa, for the prediction algorithm this means that a company with an X19 value greater than 0.25 is unlikely to default.

The second most important feature is X86. I create the same plot as above but with X86 instead:



Here, we see a similar picture. All defaulting companies had an X86 value around 0.8 or less. Thus, if a company has a value greater than 0.8 one would predict that it does not go bankrupt.

Moreover, I create a scatter plot of the two most significant features X19 and X86 and color the points with the bankruptcy value Y. This looks like this:



This wonderful plot shows, that all companies that went bankrupt are in the bottom left corner, where X86 is smaller than 0.8 and where X19 is smaller than 0.25, while most surviving companies are in the top right corner.

For estimating the following models not all 95 features are used but only the top 20 most important ones as shown above.

The Learning Vector Quantization (LVQ) algorithm itself did not perform really well, as shown by the confusion matrix:

```
confusionMatrix(testing$Y, predlvq)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1979    0
##          1   66    0
##
##                Accuracy : 0.9677
##                  95% CI : (0.9591, 0.975)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.235e-15
```

```
##
##             Sensitivity : 0.9677
##             Specificity :     NA
##          Pos Pred Value :     NA
##          Neg Pred Value :     NA
##              Prevalence : 1.0000
##          Detection Rate : 0.9677
##    Detection Prevalence : 0.9677
##       Balanced Accuracy :     NA
##
##        'Positive' Class : 0
##
```

It missed all bankruptcies in the test set and thus performed equally to the naive approach with an accuracy of 0.9677.

Random Forest (rf)

To improve performance of estimating the models, I run the following models in parallel using the "doParallel" library. The description on how to set it up can be found in the caret documentation:

https://topepo.github.io/caret/parallel-processing.html

After estimating all models on the training set, I use the test set to predict the values and to evaluate the performance of each model. This is done by inspecting the confusion matrix.

The confusion matrix for the Random Forest (rf) looks following:

```
confusionMatrix(testing$Y, predrf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1974    5
##          1   53   13
##
##                Accuracy : 0.9716
##                  95% CI : (0.9635, 0.9784)
##     No Information Rate : 0.9912
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2998
##
##  Mcnemar's Test P-Value : 6.769e-10
##
##             Sensitivity : 0.9739
##             Specificity : 0.7222
##          Pos Pred Value : 0.9975
##          Neg Pred Value : 0.1970
##              Prevalence : 0.9912
##          Detection Rate : 0.9653
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.8480
```

```
##
##        'Positive' Class : 0
##
```

k-Nearest Neighbors (knn)

According to the following article, the most popular machine learning algorithms for classification problems are:

- K-Nearest Neighbors
- Decision Tree
- Logistic Regression
- Naive Bayes
- Support Vector Machines

Source: https://monkeylearn.com/blog/classification-algorithms/

Therefore, for each of the above mentioned algorithms I use a similar R / caret counterpart.

```
confusionMatrix(testing$Y, predknn)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1972    7
##          1   53   13
##
##                Accuracy : 0.9707
##                  95% CI : (0.9624, 0.9775)
##     No Information Rate : 0.9902
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2917
##
##  Mcnemar's Test P-Value : 6.267e-09
##
##             Sensitivity : 0.9738
##             Specificity : 0.6500
##          Pos Pred Value : 0.9965
##          Neg Pred Value : 0.1970
##              Prevalence : 0.9902
##          Detection Rate : 0.9643
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.8119
##
##        'Positive' Class : 0
##
```

Random Forest (Rborist)

For the decision tree I use a different Random Forest library namely "Rborist".

```
confusionMatrix(testing$Y, predRborist)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1975    4
##          1   52   14
##
##                Accuracy : 0.9726
##                  95% CI : (0.9646, 0.9792)
##     No Information Rate : 0.9912
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.324
##
##  Mcnemar's Test P-Value : 3.372e-10
##
##             Sensitivity : 0.9743
##             Specificity : 0.7778
##          Pos Pred Value : 0.9980
##          Neg Pred Value : 0.2121
##              Prevalence : 0.9912
##          Detection Rate : 0.9658
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.8761
##
##        'Positive' Class : 0
##
```

Boosted Logistic Regression (LogitBoost)

Instead of the "Normal" Logistic Regression I use a Boosted Logistic Regression.

```
confusionMatrix(testing$Y, predLogitBoost)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1957   22
##          1   41   25
##
##                Accuracy : 0.9692
##                  95% CI : (0.9608, 0.9762)
##     No Information Rate : 0.977
##     P-Value [Acc > NIR] : 0.99023
##
##                   Kappa : 0.4271
##
##  Mcnemar's Test P-Value : 0.02334
##
```

```
##             Sensitivity : 0.9795
##             Specificity : 0.5319
##          Pos Pred Value : 0.9889
##          Neg Pred Value : 0.3788
##              Prevalence : 0.9770
##          Detection Rate : 0.9570
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.7557
##
##        'Positive' Class : 0
##
```

Naive Bayes (naive_bayes)

```
confusionMatrix(testing$Y, prednaive_bayes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1915   64
##          1   21   45
##
##                Accuracy : 0.9584
##                  95% CI : (0.9489, 0.9667)
##     No Information Rate : 0.9467
##     P-Value [Acc > NIR] : 0.008548
##
##                   Kappa : 0.4939
##
##  Mcnemar's Test P-Value : 5.225e-06
##
##             Sensitivity : 0.9892
##             Specificity : 0.4128
##          Pos Pred Value : 0.9677
##          Neg Pred Value : 0.6818
##              Prevalence : 0.9467
##          Detection Rate : 0.9364
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.7010
##
##        'Positive' Class : 0
##
```

Least Squares Support Vector Machine with Radial Basis Function Kernel (ssvmRadial)

For the Support Vector Machine I used the "Least Squares Support Vector Machine with Radial Basis Function Kernel" algorithm.

```
confusionMatrix(testing$Y, predlssvmRadial)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1975    4
##          1   60    6
##
##                Accuracy : 0.9687
##                  95% CI : (0.9602, 0.9758)
##     No Information Rate : 0.9951
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1507
##
##  Mcnemar's Test P-Value : 6.199e-12
##
##             Sensitivity : 0.97052
##             Specificity : 0.60000
##          Pos Pred Value : 0.99798
##          Neg Pred Value : 0.09091
##              Prevalence : 0.99511
##          Detection Rate : 0.96577
##    Detection Prevalence : 0.96773
##       Balanced Accuracy : 0.78526
##
##        'Positive' Class : 0
##
```

Evaluation

In total there are now seven different machine learning algorithms used to predict the bankruptcy outcome. Instead of just relying on one model, I use ensembles. The idea of ensembles is to combining predictions from different algorithms to obtain a better estimate of the true outcome. As learned throughout the course, in machine learning, one can usually greatly improve the final results by combining the results of different algorithms. Source: https://rafalab.github.io/dsbook/machine-learning-in-practice.html#ensembles

The ensemble, or as I call it the voting system, is created by firstly gathering all predictions in a data frame. Then, I take the average of all predictions. If it is larger than 0.5 it means four of the seven models predicted a bankruptcy. Therefore the collective vote is a 1 for bankruptcy. On the other hand, if the mean is less than 0.5 the outcome is a 0.

The code looks like this:

```r
# Create a voting system
preds <- data.frame(predlvq) %>%
  mutate(predrf, predknn, predRborist, predLogitBoost, prednaive_bayes, predlssvmRadial)

indx <- sapply(preds, is.factor)
preds[indx] <- lapply(preds[indx], function(x) as.numeric(as.character(x)))
preds <- preds %>% mutate(vote=if_else(rowMeans(preds)>=0.5,1,0))
```

By printing the first ten rows of the outcome one sees following:

```r
head(preds,10)
```

```
##    predlvq predrf predknn predRborist predLogitBoost prednaive_bayes
## 1        0      0       1           0              1               1
## 2        0      0       0           0              0               0
## 3        0      0       0           0              0               0
## 4        0      0       0           0              0               0
## 5        0      0       0           0              0               0
## 6        0      0       0           0              0               0
## 7        0      0       0           0              0               0
## 8        0      0       0           0              0               0
## 9        0      0       0           0              0               0
## 10       0      1       1           1              1               1
##    predlssvmRadial vote
## 1                0    0
## 2                0    0
## 3                0    0
## 4                0    0
## 5                0    0
## 6                0    0
## 7                0    0
## 8                0    0
## 9                0    0
## 10               0    1
```

In the first row, three models predicted a bankruptcy. Four models have a contrary opinion. Therefore, the result in the "vote" column is a 0. On the other hand, the majority in row ten, namely five out of the seven algorithms predicted a bankruptcy. Thus the result is also a 1.

The confusion matrix of the ensemble / my voting system looks like this:

```r
confusionMatrix(testing$Y, factor(preds$vote, levels = c(0,1)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1975    4
##          1   53   13
##
##                Accuracy : 0.9721
##                  95% CI : (0.964, 0.9788)
##     No Information Rate : 0.9917
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3041
##
##  Mcnemar's Test P-Value : 2.047e-10
##
##             Sensitivity : 0.9739
##             Specificity : 0.7647
##          Pos Pred Value : 0.9980
##          Neg Pred Value : 0.1970
```

```
##              Prevalence : 0.9917
##          Detection Rate : 0.9658
##    Detection Prevalence : 0.9677
##       Balanced Accuracy : 0.8693
##
##        'Positive' Class : 0
##
```

To compare all this models easily to each other, I extracted “Accuracy”, “Sensitivity” and “Specificity” and saved them in a separate data frame:

```
confInfo
```

```
##        Method  Accuracy Sensitivity Specificity
## 1         lvq 0.9677262   0.9677262          NA
## 2          rf 0.9716381   0.9738530   0.7222222
## 3         knn 0.9706601   0.9738272   0.6500000
## 4     Rborist 0.9726161   0.9743463   0.7777778
## 5  LogitBoost 0.9691932   0.9794795   0.5319149
## 6 naive_bayes 0.9584352   0.9891529   0.4128440
## 7 lssvmRadial 0.9687042   0.9705160   0.6000000
## 8        vote 0.9721271   0.9738659   0.7647059
```

As one can see by the accuracy, the worst model is the Naive Bayes (naive_bayes) one. Judging by the accuracy it performed even worse than my simplistic approach by always predicting a 0. While the Learning Vector Quantization (LVQ) is on par with my simplistic approach, all other machine learning algorithms performed better and thus delivered an added value. The best model judged by the accuracy is the Random Forest (Rborist) which achieved 0.9726161. This is slightly but not much better than my voting system which delivered 0.9721271. My voting system was the second best. The theory that ensembles performe better than a single algorithms could in this case not be confirmed. One would be better of by just using the Rborist model.

Lets' take a look at the sensitivity. The sensitivity describes the true positive rate and refers to the proportion of those who received a positive result on the test out of those who actually have the condition.

The best in class model is here the Naive Bayes (naive_bayes) one with a sensitivity of 0.9891529. My voting system places somewhere in the middle. Following table shows how many actual bankruptcies were correctly predicted by each model:

```
colSums(predbank)
```

```
##             Y         hitlvq          hitrf        hitlknn     hitRborist
##            66              0             13             13             14
##  hitLogitBoost hitnaive_bayes hitlssvmRadial        hitvote
##            25             45              6             13
```

Out of the 66 actual bankruptcies, the Naive Bayes (naive_bayes) model predicted with 45 by fare the most ones correctly. Taking a look at how many bankruptcies were missed by each model, one sees that it missed the fewest:

```
colSums(missedbank)
```

```
##                  Y         missedlvq         missedlrf        missedlknn
##                 66                66                53                53
##     missedRborist  missedLogitBoost missednaive_bayes missedlssvmRadial
##                 52                41                21                60
##        missedvote
##                 53
```

This explains the high sensitivity. However, by looking at how many false positives were predicted, meaning where the model predicted a bankruptcy where there was not, the picture looks completely different:

```
colSums(falsepos)
```

```
##                 Y         falselvq         falselrf        falselknn
##                66                0                5                7
##     falseRborist  falseLogitBoost falsenaive_bayes falselssvmRadial
##                 4               22               64                4
##        falsevote
##                 4
```

The high sensitivity of the Naive Bayes (naive_bayes) was achieved at the expense of many false positives, namely it predicted 64 false positives. The voting system, and the two Random Forest models performed about equally well. The Random Forest (rf) and the voting system predicted both 13 bankruptcies correctly. The Random Forest (Rborist) was slightly better by predicting 14 correctly. The false positives are also similar. The voting system and the Random Forest (Rborist) both predicted 4 false, while the Random Forest (rf) did classify 5 incorrectly.

Next I evaluate the specificity. The specificity describes the true negative rate and refers to the proportion of those who received a negative result on the test out of those who do not actually have the condition.

Here the ranking is similar to the accuracy. The best one is also the Random Forest (Rborist), followed by my voting system and then the Random Forest (rf). Here is the table of the correctly predicted surviving companies.

```
colSums(survive)
```

```
##                   Y         survivelvq         survivelrf        survivelknn
##                  66               1979               1974               1972
##     surviveRborist  surviveLogitBoost survivenaive_bayes survivelssvmRadial
##               1975               1957               1915               1975
##        survivevote
##               1975
```

Obviously the best model was the Learning Vector Quantization (LVQ) one, since it never predicted a bankruptcy and always predicted a survival. Thus, here it predicted all survivals 100% correct. The second best ones in this regard is again the Random Forest (Rborist) and my voting system.

In conclusion, out of the eight models, seven machine learning algorithms and the voting system, the best models were the Random Forest ones. While the Rborist algorithm was the best, also the standard one (rf) performed really well. My voting system, the ensemble, worked also quite good, however it did not outperform the Rborist one, thus it did not add much additional value.

Conclusion

Throughout this project, I used different machine learning algorithms to predict company bankruptcies. By starting with a quite simplistic approach, namely by assigning no bankruptcy (all companies survive) to the entire set, I created a baseline. After analyzing and selecting the most important features in the data set, I applied seven different machine learning algorithms. Additionally, I used the ensembles technique to create a voting system. While one model performed worse than the baseline in terms of accuracy, it was shown that all other outperformed it. This means, provided that one has no idea on how to conceptually differentiate bankruptcy vulnerable companies from sound firms, instead of handing out credits to everyone, machine learning algorithms can indeed provide an added value in the distinction.

In future work I could try two things: Firstly, the selection of the important features was done also using machine learning. Maybe the results can be improved if one selects the relevant features base on financial theory. If conceptually meaningful variables were use for the following machine learning tasks, the predictions might be also more accurate. Secondly, while it was explicitly not the aim of this project to create a perfect model which predicts all bankruptcies perfectly, I could nevertheless try to improve the models using tuning parameters. Without tuning, the Random Forest (Rborist) algorithms was the best one. Maybe by tuning the model I might get even better and more accurate predictions of the bankruptcies.