STK2100: Solutions Week 9

Lars H. B. Olsen

10.03.2021

Textbook

Exercise 5.9

We are working with the following linear discriminant function: $d_k(x) = \log(\pi_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + x^T\Sigma^{-1}\mu_k$, for $k = 1, 2, \ldots, K$. Let $K = 2$ and assume equal prior probabilities $\pi_1 = \pi_2 = 0.5$. We want to show that $d_1(x) > d_2(x)$ also can be written as $(\mu_1 - \mu_2)^T\Sigma^{-1}(x - \mu) > 0$, where $\mu = \frac{1}{2}(\mu_1 + \mu_2)$. We have that

$$
\begin{aligned}
d_1(x) &> d_2(x) \\
d_1(x) - d_2(x) &> 0 \\
\big[\log(\pi_1) - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + x^T\Sigma^{-1}\mu_1\big] - \big[\log(\pi_2) - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + x^T\Sigma^{-1}\mu_2\big] &> 0 \\
\big[-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + x^T\Sigma^{-1}\mu_1\big] + \big[\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 - x^T\Sigma^{-1}\mu_2\big] &> 0 \\
\big[-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + x^T\Sigma^{-1}\mu_1\big] + \big[\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 - x^T\Sigma^{-1}\mu_2\big] - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2 &> 0 \\
x^T\Sigma^{-1}\mu_1 - x^T\Sigma^{-1}\mu_2 - \left[\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2\right] + \left[\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2\right] &> 0 \\
\mu_1^T\Sigma^{-1}x - \mu_2^T\Sigma^{-1}x - \left[\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_2\right] + \left[\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_1\right] &> 0 \\
\mu_1^T\Sigma^{-1}x - \mu_2^T\Sigma^{-1}x - \mu_1^T\Sigma^{-1}(\tfrac{1}{2}[\mu_1 + \mu_2]) + \mu_2^T\Sigma^{-1}(\tfrac{1}{2}[\mu_1 + \mu_2]) &> 0 \\
\mu_1^T\Sigma^{-1}x - \mu_2^T\Sigma^{-1}x - \mu_1^T\Sigma^{-1}\mu + \mu_2^T\Sigma^{-1}\mu &> 0 \\
(\mu_1^T - \mu_2^T)\Sigma^{-1}(x - \mu) &> 0 \\
(\mu_1 - \mu_2)^T\Sigma^{-1}(x - \mu) &> 0,
\end{aligned}
$$

which was what we were asked to show.

Exercise 5.10

We are asked to assume $p = 1$ and that $\sum_{i=1}^n x_i = 0$. Then the simple linear regression model takes the form $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, so $y = X\beta + \epsilon$, see section 5.4. Furthermore, assume that the $K = 2$ and that the two classes, 'zero' and 'one', are of equal size. In total, we have n observations. From this we get that $X^TX = \begin{bmatrix} n & 0 \\ 0 & \sum_{i=1}^n x_i^2 \end{bmatrix}$, and $X^Ty = \begin{bmatrix} n_1 \\ n_1\hat{\mu}_1 \end{bmatrix}$, where n_1 is the number of observations in class 1 and $\hat{\mu}_1$ is defined as in the book. Then $\hat{\beta} = \begin{bmatrix} n_1/n \\ n_1\hat{\mu}_1/\sum_{i=1}^n x_i^2 \end{bmatrix}$.

We then get the boundary $n_1/n + \frac{n_1\hat{\mu}_1}{\sum_{i=1}^n x_i^2}x > 1/2$. Since we have that the two classes are of equal size, we know that $n_1/n = 1/2$, thus we can subtract $-1/2$ from both sides of the inequality. This is the reason

for why the exercise states *show also that this statement does not hold if classes have different numbers of observations*. We then have that $\frac{n_1\hat{\mu}_1}{\sum_{i=1}^{n} x_i^2}x > 0$, which is equivalent to $x > 0$.

From the previous exercise, we see that the threshold value between the two classes is

$$x = \mu = \frac{1}{2}(\hat{\mu}_0 + \hat{\mu}_1) = \frac{1}{2n/2}(\sum_{i:c_i=0}^{n_0} x_i + \sum_{i:c_i=1}^{n_1} x_i) = \frac{1}{n}\sum_{i=1}^{n} x_i = 0,$$

as then $d_0(x) = d_1(x)$. Here c_i represent the class of the ith observation, and we have used the assumptions above of equal size $n_0 = n_1 = n/2$ and that $\sum_{i=1}^{n} x_i = 0$. The two threshold values coincide and we are done with the exercise.

Exercise 5.10

ISLR

Include necessary packages:

```
library(tidyverse)
library(ISLR)  # 'Weekly' data
library(caret) # train(), confusionMatrix()
library(MASS)  # lda(), qda(), `Boston` data
select <- dplyr::select # MASS 'select' clashing with dplyr
library(class) # knn()
library(gridExtra)

theme_set(theme_light())
```

Exercise 4.9 Odds vs Probability

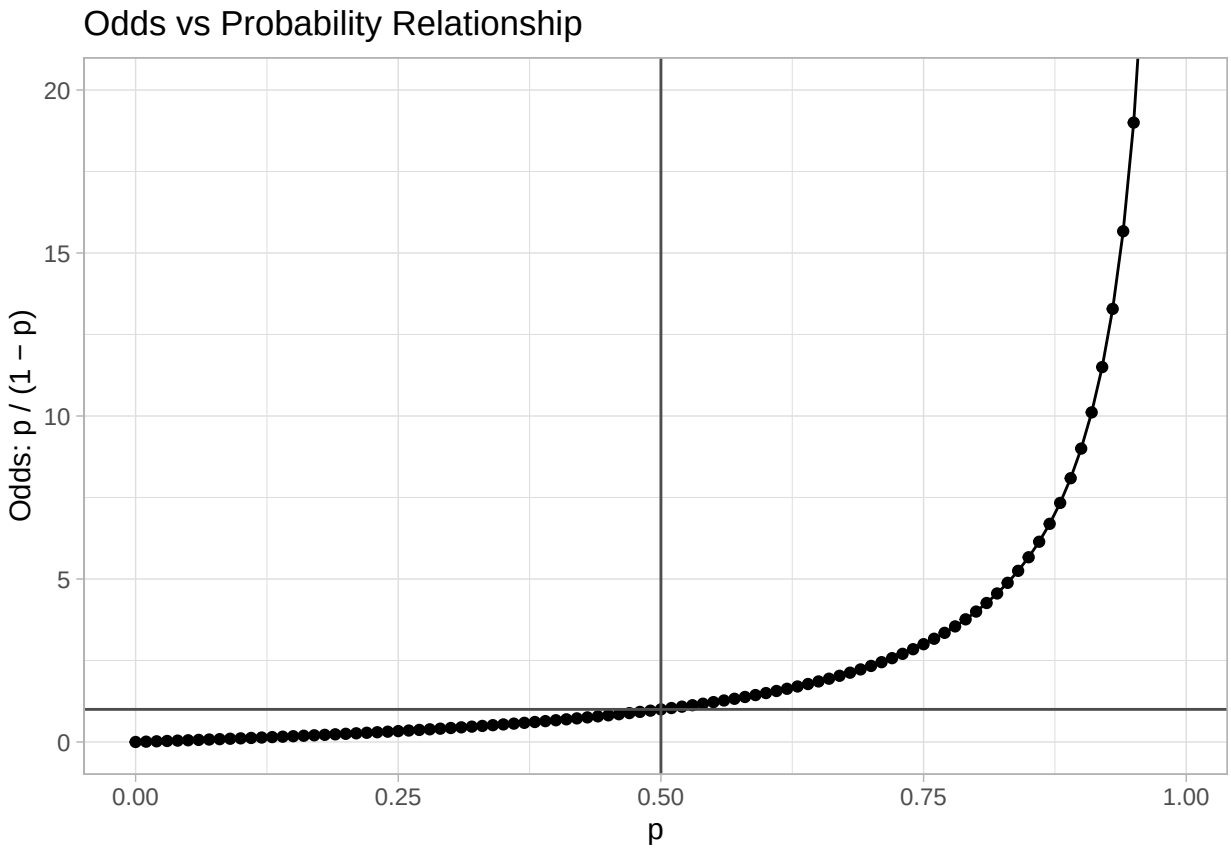
This problem has to do with *odds*.

Here's a quick bonus graph showing the relationship between odds ($\frac{p(X)}{1-p(X)}$) and probability ($p(X)$), where we can see:

- $\frac{p(X)}{1-p(X)} \geq 0 \quad (\forall p(X) \in [0, 1))$
- Odds $= 1$ when $p(X) = 0.5$:

The graph has been truncated at y $= 20$, since $\lim_{p\to 1^-} \frac{p}{1-p} = \infty$

```
data.frame(prob = seq(0, 0.99, 0.01)) %>%
  mutate(odds = prob / (1 - prob)) %>%
  ggplot(aes(x = prob, y = odds)) +
  geom_point() +
  geom_line() +
  geom_vline(xintercept = 0.5, col = "grey30") +
  geom_hline(yintercept = 1, col = "grey30") +
  coord_cartesian(ylim = c(0, 20)) +
  labs(x = "p",
       y = "Odds: p / (1 - p)",
       title = "Odds vs Probability Relationship")
```



(a) Odds → Probability

Q: *On average, what fraction of people with an odds of 0.37 of defaulting on their credit card payment will in fact default?*

A:

$$\begin{aligned} \frac{p(X)}{1-p(X)} &= 0.37 \\ \implies p(X) &= 0.37 - 0.37 \cdot p(X) \\ \implies p(X) &= \frac{0.37}{1.37} \end{aligned}$$

```
round(0.37/1.37, 3)
```

```
## [1] 0.27
```

(b) Probability → Odds

Q: *Suppose that an individual has a 16% chance of defaulting on her credit card payment. What are the odds that she will default?*

A:

$$p(X) = 0.16 \implies \frac{p(X)}{1-p(X)} = \frac{0.16}{1-0.16}$$

```r
round(0.16/0.84, 3)
```

```
## [1] 0.19
```

Exercise 4.10 The `Weekly` Dataset (Logistic, LDA, QDA, KNN)

This question should be answered using the `Weekly` data set, which is part of the `ISLR` package. This data is similar in nature to the `Smarket` data from this chapter's lab, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

```r
glimpse(Weekly)
```

```
## Rows: 1,089
## Columns: 9
## $ Year      <dbl> 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 199...
## $ Lag1      <dbl> 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0.807...
## $ Lag2      <dbl> 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.372...
## $ Lag3      <dbl> -3.936, 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, 1.178...
## $ Lag4      <dbl> -0.229, -3.936, 1.572, 0.816, -0.270, -2.576, 3.514, 0.71...
## $ Lag5      <dbl> -3.484, -0.229, -3.936, 1.572, 0.816, -0.270, -2.576, 3.5...
## $ Volume    <dbl> 0.1549760, 0.1485740, 0.1598375, 0.1616300, 0.1537280, 0....
## $ Today     <dbl> -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0.807, 0.041...
## $ Direction <fct> Down, Down, Up, Up, Up, Down, Up, Up, Up, Down, Down, Up,...
```

The variables are:

- `Year`: The year that the observation was recorded
- `Lag1`: Percentage return for previous week
- `Lag2`: Percentage return for 2 weeks previous
- `Lag3`: Percentage return for 3 weeks previous
- `Lag4`: Percentage return for 4 weeks previous
- `Lag5`: Percentage return for 5 weeks previous
- `Volume`: Volume of shares traded (average number of daily shares traded in billions)
- `Today`: Percentage return for this week
- `Direction`: A factor with levels `Down` and `Up` indicating whether the market had a positive or negative return on a given week

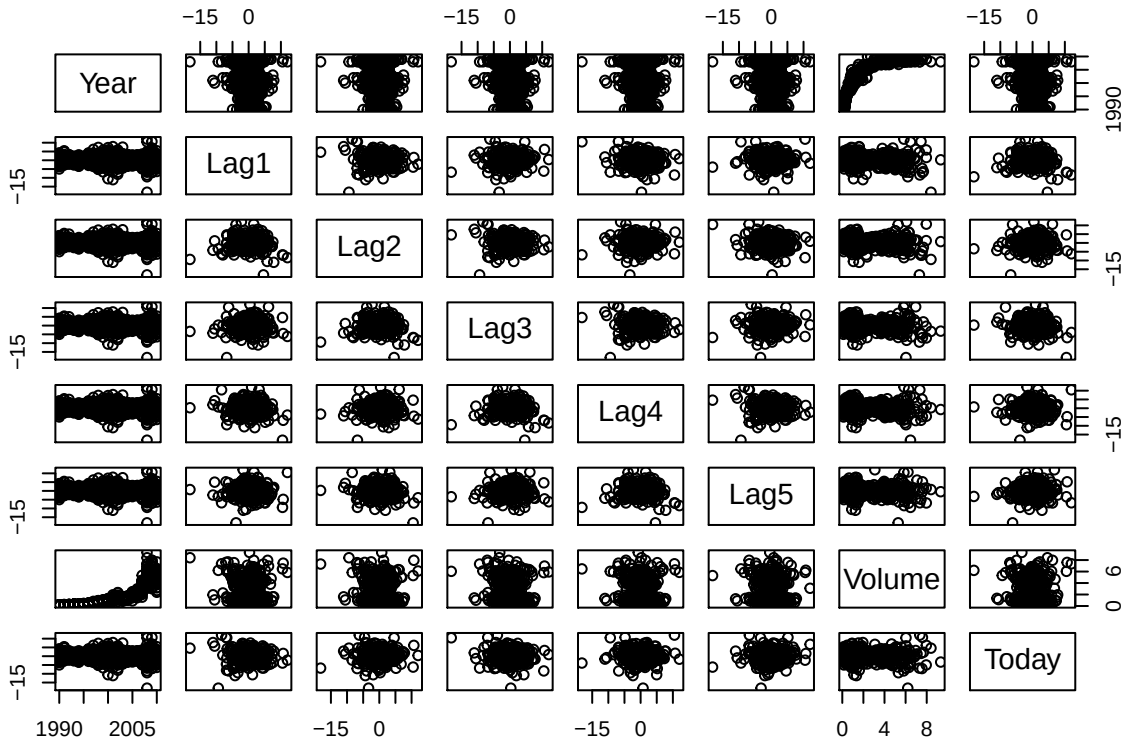
(a) Data Summary

Q: *Produce some numerical and graphical summaries of the `Weekly` data. Do there appear to be any patterns?*

A:

An initial glance at the numeric variables of the dataset:

```r
pairs(Weekly[ ,-9])
```




```
abs(cor(Weekly[ ,-9]))
```

```
##              Year        Lag1       Lag2       Lag3        Lag4        Lag5
## Year   1.00000000 0.032289274 0.03339001 0.03000649 0.031127923 0.030519101
## Lag1   0.03228927 1.000000000 0.07485305 0.05863568 0.071273876 0.008183096
## Lag2   0.03339001 0.074853051 1.00000000 0.07572091 0.058381535 0.072499482
## Lag3   0.03000649 0.058635682 0.07572091 1.00000000 0.075395865 0.060657175
## Lag4   0.03112792 0.071273876 0.05838153 0.07539587 1.000000000 0.075675027
## Lag5   0.03051910 0.008183096 0.07249948 0.06065717 0.075675027 1.000000000
## Volume 0.84194162 0.064951313 0.08551314 0.06928771 0.061074617 0.058517414
## Today  0.03245989 0.075031842 0.05916672 0.07124364 0.007825873 0.011012698
##            Volume       Today
## Year   0.84194162 0.032459894
## Lag1   0.06495131 0.075031842
## Lag2   0.08551314 0.059166717
## Lag3   0.06928771 0.071243639
## Lag4   0.06107462 0.007825873
## Lag5   0.05851741 0.011012698
## Volume 1.00000000 0.033077783
## Today  0.03307778 1.000000000
```

As we would expect with stock market data, there are no obvious strong relationships between the `Lag` variables. However, there do appear to be some interesting trends over time. I create the `Week` variable below, allowing for easier plotting of trends, since there is a chronology to the rows that is not shown fully through the `Year` variable.

I first do a quick sense-check that the rows are in the correct order, based on the definition of `Today` and `Lag1`:

```
Weekly %>%
filter(lead(Lag1) != Today) %>%
nrow()
```

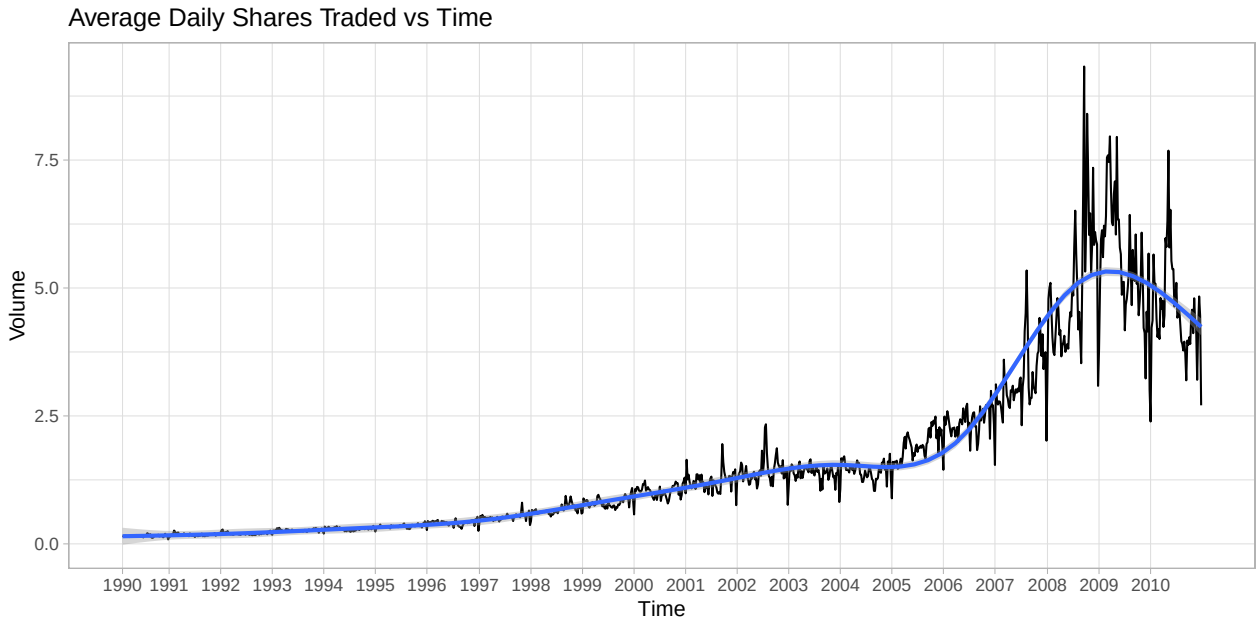
```
## [1] 0
```

Since there are no rows out of order, the dataset appears to be correctly ordered in ascending weeks, so I create `Week` (basically a row counter):

```
Weekly$Week <- 1:nrow(Weekly)
```

Looking at `Volume` over time, there has been a significant increase in the volume of shares traded since the 90's. This appears to have peaked around 2009, starting to decrease in 2010. it would be interesting to see the S&P 500 stats since then.

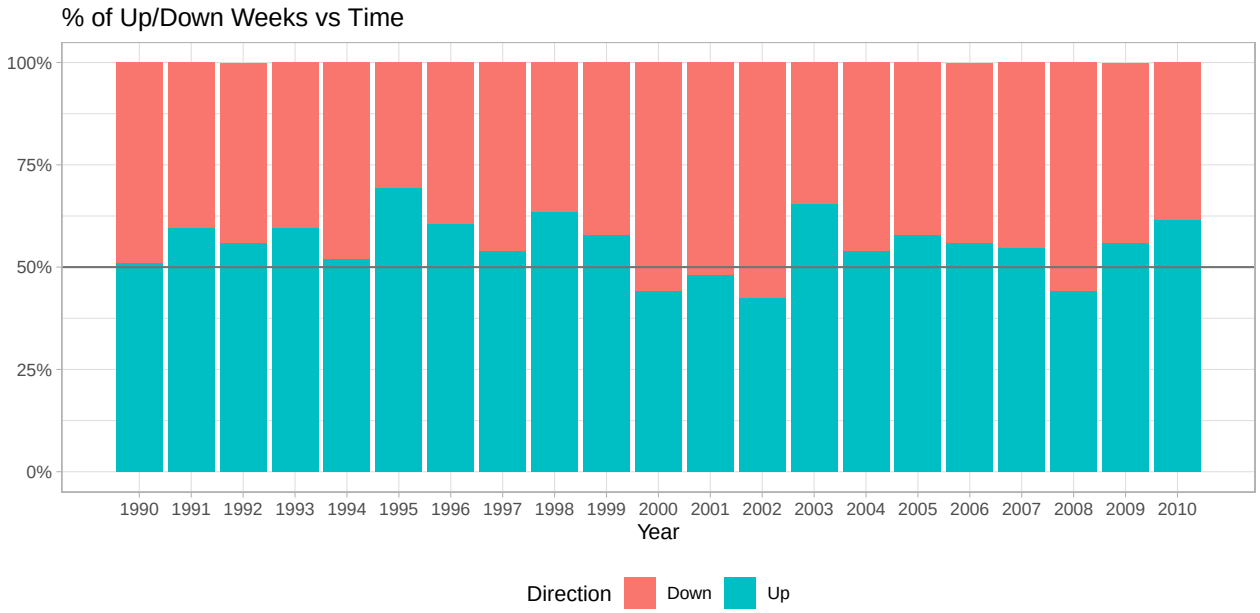
```
year_breaks <- Weekly %>%
  group_by(Year) %>%
  summarize(Week = min(Week))

ggplot(Weekly, aes(x = Week, y = Volume)) +
  geom_line() +
  geom_smooth() +
  scale_x_continuous(breaks = year_breaks$Week,
                     minor_breaks = NULL,
                     labels = year_breaks$Year) +
  labs(title = "Average Daily Shares Traded vs Time",
       x = "Time") +
  theme_light()
```



Here is `Direction` over time, which is less interesting. There appear to only be 4 years in which >= 50% of the weeks *didn't* see a positive return (2000, 2001, 2002, 2008).

```
ggplot(Weekly, aes(x = Year, fill = Direction)) +
  geom_bar(position = "fill") +
  geom_hline(yintercept = 0.5, col = "grey45") +
  scale_x_continuous(breaks = seq(1990, 2010),
                     minor_breaks = NULL) +
  scale_y_continuous(labels = scales::percent_format()) +
  theme_light() +
  theme(axis.title.y = element_blank(),
        legend.position = "bottom") +
```

```
  ggtitle("% of Up/Down Weeks vs Time")
```



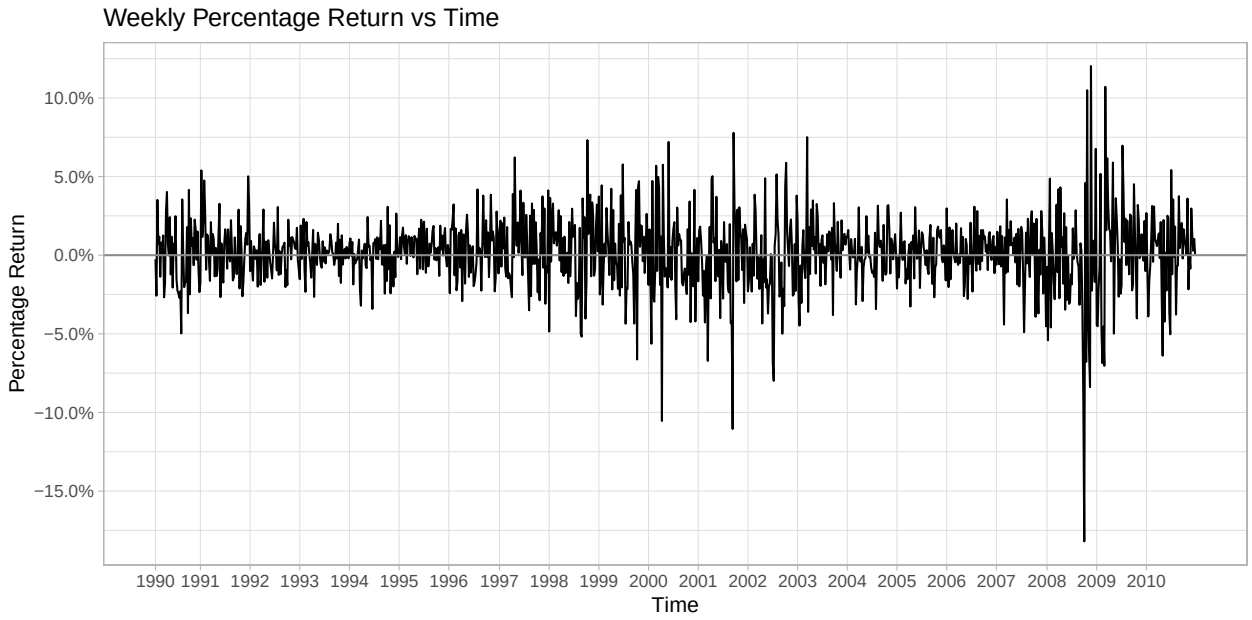
The split of the *weeks* into `Down` & `Up` can be seen in the table below. We could get a classifier with 55.56% accuracy simply by predicting the S&P 500 return will be positive every week.

```
prop.table(table(Weekly$Direction))
```

```
##
##      Down        Up
## 0.4444444 0.5555556
```

We can also see that the market seems to go through periods of higher variance/instability. Crashes (e.g. Sept. 2008) stand out here.

```
ggplot(Weekly, aes(x = Week, y = Today / 100)) +
  geom_line() +
  scale_x_continuous(breaks = year_breaks$Week,
                     minor_breaks = NULL,
                     labels = year_breaks$Year) +
  scale_y_continuous(labels = scales::percent_format(), breaks = seq(-0.2, 0.2, 0.05)) +
  geom_hline(yintercept = 0, col = "grey55") +
  theme_light() +
  labs(title = "Weekly Percentage Return vs Time",
       x = "Time",
       y = "Percentage Return")
```



(b) Logistic Regression (predict market `Direction`)

Q: *Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?*

A:

Lag2 appears to be the only statistically significant predictor:

```
glm_dir <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Weekly,
               family = "binomial")

summary(glm_dir)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = "binomial", data = Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6949  -1.2565   0.9913   1.0849   1.4579
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

Note that the output from the Z-statistic (z value) is calculated the same as with the T-test in linear regression: $Z = \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)}$, and a large absolute value indicates evidence against the null hypothesis $H_0 : \beta_j = 0$ (again, the same).

(c) Confusion Matrix

Q: *Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.*

A:

The confusion matrix is shown below. I use `caret::confusionMatrix()`, as it has various confusion matrix statistics built in to its output.

```
predicted <- factor(ifelse(predict(glm_dir, type = "response") < 0.5, "Down", "Up"))

confusionMatrix(predicted, Weekly$Direction, positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down  Up
##       Down   54  48
##       Up    430 557
##
##                Accuracy : 0.5611
##                  95% CI : (0.531, 0.5908)
##     No Information Rate : 0.5556
##     P-Value [Acc > NIR] : 0.369
##
##                   Kappa : 0.035
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9207
##             Specificity : 0.1116
##          Pos Pred Value : 0.5643
##          Neg Pred Value : 0.5294
##              Prevalence : 0.5556
##          Detection Rate : 0.5115
##    Detection Prevalence : 0.9063
##       Balanced Accuracy : 0.5161
##
##        'Positive' Class : Up
##
```

This is just over the baseline accuracy (55.56%) achieved by a naive classifier (that predicts `Up` every time).

In fact, this is almost the strategy of the logistic regression model:

```
prop.table(table(predicted))
```

```
## predicted
##       Down         Up
## 0.09366391 0.90633609
```

This is reflected in the very poor specificity (it does not predict the negative class well).

Note also that we are dealing with training accuracy here, so this marginal accuracy improvement over the baseline is not interesting.

(d) `train` and `test` - Logistic Regression

Q: *Now fit the logistic regression model using a training data period from 1990 to 2008, with* `Lag2` *as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).*

A:

I create `train` and `test`, corresponding to the two time periods given in the question.

The confusion matrix is for the `test` predictions this time.

```
train <- Weekly[Weekly$Year <= 2008, ]
test <- Weekly[Weekly$Year > 2008, ]

glm_dir <- glm(Direction ~ Lag2,
               data = train,
               family = "binomial")

predicted <- factor(ifelse(predict(glm_dir, newdata = test, type = "response")
                           < 0.5, "Down", "Up"))

confusionMatrix(predicted, test$Direction, positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down    9  5
##       Up     34 56
##
##                Accuracy : 0.625
##                  95% CI : (0.5247, 0.718)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.2439
##
##                   Kappa : 0.1414
##
##  Mcnemar's Test P-Value : 7.34e-06
##
##             Sensitivity : 0.9180
##             Specificity : 0.2093
##          Pos Pred Value : 0.6222
##          Neg Pred Value : 0.6429
##              Prevalence : 0.5865
```

```
##          Detection Rate : 0.5385
##    Detection Prevalence : 0.8654
##       Balanced Accuracy : 0.5637
##
##        'Positive' Class : Up
##
```

Here we get an Accuracy of **0.625**.

The `confusionMatrix()` function provides a lot of other useful statistics. For example, `No Information Rate : 0.5865` tells us that the largest class (`Up`) is 58.65% of the `test` observations, and hence this is our baseline for a classifier.

Clearly `Accuracy : 0.625` > 0.5865, which is positive. However, our `test` dataset is relatively small so this might not be a meaningful improvement.

We are provided with a p-value for a one-sided test to see whether the accuracy is better than the "no information rate". `P-Value [Acc > NIR] : 0.2439` $> 0.05 \implies$ no significant evidence that our classifier is better than the baseline strategy. Predicting stock movements is hard - who would've thought?

(e) `train` and `test` - LDA

Q: *Repeat (d) using LDA.*

A:

```
lda_dir <- lda(Direction ~ Lag2, data = train)


predicted_lda <- predict(lda_dir, newdata = test)

confusionMatrix(data = predicted_lda$class,
                reference = test$Direction,
                positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down    9  5
##       Up     34 56
##
##                Accuracy : 0.625
##                  95% CI : (0.5247, 0.718)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.2439
##
##                   Kappa : 0.1414
##
##  Mcnemar's Test P-Value : 7.34e-06
##
##             Sensitivity : 0.9180
##             Specificity : 0.2093
##          Pos Pred Value : 0.6222
##          Neg Pred Value : 0.6429
##              Prevalence : 0.5865
##          Detection Rate : 0.5385
```

```
##    Detection Prevalence : 0.8654
##       Balanced Accuracy : 0.5637
##
##        'Positive' Class : Up
##
```

Here we get an Accuracy of **0.625**. Note that, as before, we have `P-Value [Acc > NIR] : 0.2439` > 0.05, so whilst the accuracy of the classifier is `0.625` $>$ `0.5865`, the `test` sample size is not large enough for this increase over the baseline to be meaningful.

`predict.lda()$posterior` gives a data frame of probability predictions, with one column per response class. `predict.lda()$class`gives the class prediction for each observation (the class with the greatest probability). I check this below:

```
identical(as.character(predicted_lda$class),
as.character(ifelse(predicted_lda$posterior[ ,2] < 0.5, "Down", "Up")))
```

```
## [1] TRUE
```

(f) `train` and `test` - QDA

Q: *Repeat (d) using QDA.*

A:

```
qda_dir <- qda(Direction ~ Lag2, data = train)


predicted_qda <- predict(qda_dir, newdata = test)

confusionMatrix(data = predicted_qda$class,
                reference = test$Direction,
                positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down    0  0
##       Up     43 61
##
##                Accuracy : 0.5865
##                  95% CI : (0.4858, 0.6823)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.5419
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.504e-10
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.5865
##          Neg Pred Value :    NaN
##              Prevalence : 0.5865
##          Detection Rate : 0.5865
##    Detection Prevalence : 1.0000
```

```
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : Up
##
```

Here we get an Accuracy of **0.5865**.

Note that the QDA classifier just predicts `Up` for every `test` observation - it behaves identically to the naive classifier on this dataset, with a sensitivity of 1 and a specificity of 0.

(g) `train` and `test` - KNN (K = 1)

Q: *Repeat (d) using KNN with K = 1.*

A:

Usually as a pre-processing step for KNN (with multiple predictors over different scales), we would want to standardize the predictors ($x_{new} = \frac{x-\mu}{\sigma}$) so that each x_{new} will have a mean of 0 and standard deviation of 1. In this case, however, there is only 1 predictor (`Lag2`), so the nearest neighbour would not be effected by this.

Note also that there is some element of randomness with the KNN classifier. Take the following `test` observation that requires prediction:

```
test[100, "Lag2"]
```

```
## [1] 0.043
```

```
# test[75, "Lag2"] # another one here
```

Notice that there are two `train` observations of identical distance (w.r.t `Lag2`), but both have different `Direction` values:

```
train[c(10, 808), c("Lag2", "Direction")]
```

```
##      Lag2 Direction
## 10  0.041      Down
## 808 0.041        Up
```

In this case, the KNN probability will be 0.5:

```
set.seed(1)
predicted_knn <- knn(train = data.frame(Lag2 = train$Lag2),
                     test = data.frame(Lag2 = test$Lag2),
                     cl = train$Direction,
                     k = 1,
                     prob = T)

attr(predicted_knn, "prob")[100]
```

```
## [1] 0.5
```

However, the classifier must make a prediction, which will just be chosen at random:

```
predicted_knn[100]
```

```
## [1] Down
## Levels: Down Up
```

See the confusion matrix below:

```r
confusionMatrix(data = predicted_knn,
                reference = test$Direction,
                positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down   21 30
##       Up     22 31
##
##                Accuracy : 0.5
##                  95% CI : (0.4003, 0.5997)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.9700
##
##                   Kappa : -0.0033
##
##  Mcnemar's Test P-Value : 0.3317
##
##             Sensitivity : 0.5082
##             Specificity : 0.4884
##          Pos Pred Value : 0.5849
##          Neg Pred Value : 0.4118
##              Prevalence : 0.5865
##          Detection Rate : 0.2981
##    Detection Prevalence : 0.5096
##       Balanced Accuracy : 0.4983
##
##        'Positive' Class : Up
##
```

Here we get an accuracy of **0.5**, which is again worse than the baseline.

(h) Best Performing Classifier?

Q: *Which of these methods appears to provide the best results on this data?*

A:

Taking the target metric as the accuracy of the classifier: LDA & Logistic Regression get the same `test` accuracy of **0.625**, so these two are tied.

(i) Experimenting (combined predictors, interactions, transformations, etc.)

Q: *Experiment with different combinations of predictors, including possible transformations and interactions, for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the held out data. Note that you should also experiment with values for K in the KNN classifier.*

A:

KNN - selecting best K (using cross-validation):

```r
train$Today <- NULL

ctrl <- trainControl(method = "repeatedcv",
```

```
                     number = 5,
                     repeats = 5)

set.seed(111)

knn_train <- train(y = train$Direction,
                   x = train[ ,-8],
                   method = "knn",
                   metric = "Accuracy",
                   preProcess = c("center", "scale"),
                   tuneGrid = expand.grid(k = seq(1, 50, 2)),
                   trControl = ctrl)

caret::varImp(knn_train)
```

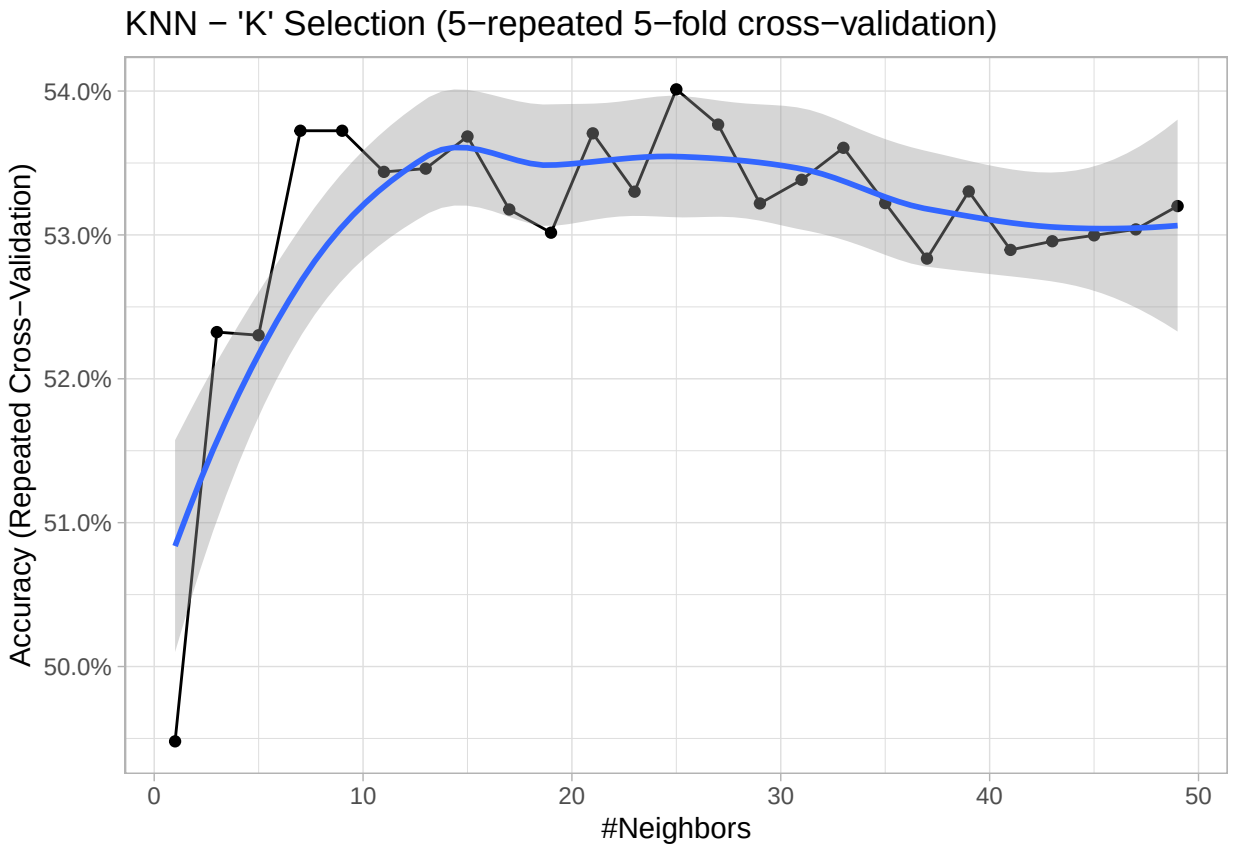
```
## ROC curve variable importance
##
##        Importance
## Lag1      100.000
## Lag2       77.256
## Lag5       64.309
## Year       45.659
## Volume     43.735
## Week       42.513
## Lag4        4.578
## Lag3        0.000
```

```
knn_train
```

```
## k-Nearest Neighbors
##
## 985 samples
##   8 predictor
##   2 classes: 'Down', 'Up'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 788, 788, 788, 788, 788, 787, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.4947996  -0.020756148
##    3  0.5232477   0.031373990
##    5  0.5230292   0.028769372
##    7  0.5372435   0.053353378
##    9  0.5372415   0.049612758
##   11  0.5343834   0.040819116
##   13  0.5346081   0.037598994
##   15  0.5368437   0.040363712
##   17  0.5317675   0.026161027
##   19  0.5301555   0.021608137
##   21  0.5370622   0.033488872
##   23  0.5330064   0.024861594
##   25  0.5401182   0.036317004
##   27  0.5376693   0.029998090
```

```
##    29  0.5321890   0.015868622
##    31  0.5338310   0.018687256
##    33  0.5360563   0.021431639
##    35  0.5322138   0.012669022
##    37  0.5283538   0.002535516
##    39  0.5330239   0.011432961
##    41  0.5289589   0.002400751
##    43  0.5295618   0.003081904
##    45  0.5299700   0.003587628
##    47  0.5303854   0.003745293
##    49  0.5320056   0.005552184
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 25.
```

```
ggplot(knn_train) +
  geom_smooth() +
  theme_light() +
  scale_y_continuous(labels = scales::percent_format()) +
  ggtitle("KNN - 'K' Selection (5-repeated 5-fold cross-validation)")
```



`caret` automatically chooses the best value for `k`. Evaluating the performance of this new model on `test`:

```
knn_pred <- predict(knn_train, newdata = test)

confusionMatrix(data = knn_pred,
                reference = test$Direction,
                positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down   19 20
##       Up     24 41
##
##                Accuracy : 0.5769
##                  95% CI : (0.4761, 0.6732)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.6193
##
##                   Kappa : 0.1156
##
##  Mcnemar's Test P-Value : 0.6511
##
##             Sensitivity : 0.6721
##             Specificity : 0.4419
##          Pos Pred Value : 0.6308
##          Neg Pred Value : 0.4872
##              Prevalence : 0.5865
##          Detection Rate : 0.3942
##    Detection Prevalence : 0.6250
##       Balanced Accuracy : 0.5570
##
##        'Positive' Class : Up
##
```

It appears that performance increased compared to the K = 1 version. However, we are still below the performance of the baseline approach. Similar (worse) results can be seen by adding other predictors, so I leave KNN for now in favour of other methods.

New Features:

Since logistic regression produced one of the best classifiers so far, it could be appropriate to call the `best_predictor()` function I wrote in previous chapters (which applies logistic regression for a categorical response variable), in the hopes that a potentially useful transformation might be identified.

```
best_predictor <- function(dataframe, response) {

  if (sum(sapply(dataframe, function(x) {is.numeric(x) | is.factor(x)}))
      < ncol(dataframe)) {
    stop("Make sure that all variables are of class numeric/factor!")
  }

  # pre-allocate vectors
  varname <- c()
  vartype <- c()
  R2 <- c()
  R2_log <- c()
  R2_quad <- c()
  AIC <- c()
  AIC_log <- c()
  AIC_quad <- c()
  y <- dataframe[ ,response]
```

```r
# # # # # NUMERIC RESPONSE # # # # #
if (is.numeric(y)) {

  for (i in 1:ncol(dataframe)) {

    x <- dataframe[ ,i]
    varname[i] <- names(dataframe)[i]

    if (class(x) %in% c("numeric", "integer")) {
      vartype[i] <- "numeric"
    } else {
      vartype[i] <- "categorical"
    }

    if (!identical(y, x)) {

      # linear: y ~ x
      R2[i] <- summary(lm(y ~ x))$r.squared

      # log-transform: y ~ log(x)
      if (is.numeric(x)) {
        if (min(x) <= 0) {
          # if y ~ log(x) for min(x) <= 0, do y ~ log(x + abs(min(x)) + 1)
          R2_log[i] <- summary(lm(y ~ log(x + abs(min(x)) + 1)))$r.squared
        } else {
          R2_log[i] <- summary(lm(y ~ log(x)))$r.squared
        }
      } else {
        R2_log[i] <- NA
      }

      # quadratic: y ~ x + x^2
      if (is.numeric(x)) {
        R2_quad[i] <- summary(lm(y ~ x + I(x^2)))$r.squared
      } else {
        R2_quad[i] <- NA
      }

    } else {
      R2[i] <- NA
      R2_log[i] <- NA
      R2_quad[i] <- NA
    }
  }

  print(paste("Response variable:", response))

  data.frame(varname,
             vartype,
             R2 = round(R2, 3),
             R2_log = round(R2_log, 3),
             R2_quad = round(R2_quad, 3)) %>%
    mutate(max_R2 = pmax(R2, R2_log, R2_quad, na.rm = T)) %>%
```

```
    arrange(desc(max_R2))


  # # # # # CATEGORICAL RESPONSE # # # # #
} else {

  for (i in 1:ncol(dataframe)) {

    x <- dataframe[ ,i]
    varname[i] <- names(dataframe)[i]

    if (class(x) %in% c("numeric", "integer")) {
      vartype[i] <- "numeric"
    } else {
      vartype[i] <- "categorical"
    }

    if (!identical(y, x)) {

      # linear: y ~ x
      AIC[i] <- summary(glm(y ~ x, family = "binomial"))$aic

      # log-transform: y ~ log(x)
      if (is.numeric(x)) {
        if (min(x) <= 0) {
          # if y ~ log(x) for min(x) <= 0, do y ~ log(x + abs(min(x)) + 1)
          AIC_log[i] <- summary(glm(y ~ log(x + abs(min(x)) + 1),
                                    family = "binomial"))$aic
        } else {
          AIC_log[i] <- summary(glm(y ~ log(x), family = "binomial"))$aic
        }
      } else {
        AIC_log[i] <- NA
      }

      # quadratic: y ~ x + x^2
      if (is.numeric(x)) {
        AIC_quad[i] <- summary(glm(y ~ x + I(x^2), family = "binomial"))$aic
      } else {
        AIC_quad[i] <- NA
      }

    } else {
      AIC[i] <- NA
      AIC_log[i] <- NA
      AIC_quad[i] <- NA
    }
  }

  print(paste("Response variable:", response))

  data.frame(varname,
             vartype,
```

```
                AIC = round(AIC, 3),
                AIC_log = round(AIC_log, 3),
                AIC_quad = round(AIC_quad, 3)) %>%
      mutate(min_AIC = pmin(AIC, AIC_log, AIC_quad, na.rm = T)) %>%
      arrange(min_AIC)
  }
}
```

However, due to the nature of what we are trying to predict, along with the relatively small sample size, it seems highly likely that the the appeared usefulness of any transformations will be due to overfitting to the training data.

To illustrate this, I call `best_predictor(train, "Direction")`, but I also add 10 noise variables (`junk_1`, ..., `junk_10`), and we can see that it's certainly not clear that any of the variables are useful at all, as the two best-performing variables are random $\mathcal{N}(0, 1)$ variables:

```
train$junk_1 <- rnorm(nrow(train))
train$junk_2 <- runif(nrow(train))
train$junk_3 <- factor(as.numeric(rnorm(nrow(train)) > 0))
train$junk_4 <- rnorm(nrow(train))
train$junk_5 <- runif(nrow(train))
train$junk_6 <- factor(as.numeric(rnorm(nrow(train)) > 0))
train$junk_7 <- rnorm(nrow(train))
train$junk_8 <- runif(nrow(train))
train$junk_9 <- factor(as.numeric(rnorm(nrow(train)) > 0))
train$junk_10 <- rnorm(nrow(train))

best_predictor(train, "Direction")
```

```
## [1] "Response variable: Direction"
```

```
##      varname     vartype      AIC  AIC_log AIC_quad  min_AIC
## 1       Lag1     numeric 1354.446 1354.199 1356.442 1354.199
## 2       Lag2     numeric 1354.543 1355.148 1355.435 1354.543
## 3     junk_6 categorical 1355.795       NA       NA 1355.795
## 4     junk_9 categorical 1356.462       NA       NA 1356.462
## 5     Volume     numeric 1356.838 1356.751 1358.833 1356.751
## 6     junk_4     numeric 1357.707 1358.426 1356.864 1356.864
## 7       Year     numeric 1357.111 1357.112 1358.772 1357.111
## 8       Week     numeric 1357.260 1358.273 1359.076 1357.260
## 9    junk_10     numeric 1357.319 1357.314 1359.128 1357.314
## 10    junk_2     numeric 1357.358 1357.721 1359.071 1357.358
## 11      Lag5     numeric 1357.365 1358.527 1358.188 1357.365
## 12    junk_7     numeric 1357.460 1357.617 1359.344 1357.460
## 13    junk_5     numeric 1358.566 1357.927 1359.136 1357.927
## 14    junk_1     numeric 1358.008 1357.938 1359.327 1357.938
## 15    junk_8     numeric 1357.945 1358.487 1359.603 1357.945
## 16      Lag3     numeric 1358.354 1358.038 1360.286 1358.038
## 17      Lag4     numeric 1358.497 1358.685 1359.007 1358.497
## 18    junk_3 categorical 1358.700       NA       NA 1358.700
## 19 Direction categorical       NA       NA       NA       NA
```

While I don't think standard transformations of the current features will be fruitful, I have a few ideas for features/modifications that *could* be useful:

- Interaction between `Lag1` & `Lag2` (the two predictors most likely to be useful)

- `Lag_avg_abs` - The average absolute value of `Lag1` to `Lag5` (a measure of the absolute size of recent market fluctuations - high values mean high-variance periods)
- `Lag_pos_cnt` - A count of how many of `Lag1` - `Lag5` were positive days (has the market trended upwards recently?)
- `Quarter` - A factor variable, created by binning the `Week` variable into the four quarters of the year
- Removing all current predictors aside from `Lag1` & `Lag2` (these are likely more useful than the older Lag variables)

I make these changes below to both `train` and `test`. Here is the resulting `train` dataset:

```
train <- train %>%
  mutate(Lag_avg_abs = abs(Lag1) + abs(Lag2) + abs(Lag3) + abs(Lag4) + abs(Lag5),
         Lag_pos_cnt = (Lag1 > 0) + (Lag2 > 0) + (Lag3 > 0) + (Lag4 > 0) + (Lag5 > 0)) %>%
  group_by(Year) %>%
  mutate(Week_of_year = row_number()) %>%
  ungroup() %>%
  mutate(Week_of_year = case_when(Year == 1990 ~ as.numeric(Week_of_year + 5),
                                  TRUE ~ as.numeric(Week_of_year))) %>%
  # data appears to start 5wks into 1990
  mutate(Quarter = factor(case_when(Week_of_year <= 13 ~ "Q1",
                                    Week_of_year <= 26 ~ "Q2",
                                    Week_of_year <= 39 ~ "Q3",
                                    TRUE ~ "Q4"))) %>%
  select(Direction, Lag1, Lag2, Lag_avg_abs, Lag_pos_cnt, Quarter)


test <- test %>%
  mutate(Lag_avg_abs = abs(Lag1) + abs(Lag2) + abs(Lag3) + abs(Lag4) + abs(Lag5),
         Lag_pos_cnt = (Lag1 > 0) + (Lag2 > 0) + (Lag3 > 0) + (Lag4 > 0) + (Lag5 > 0)) %>%
  group_by(Year) %>%
  mutate(Week_of_year = row_number()) %>%
  ungroup() %>%
  mutate(Quarter = factor(case_when(Week_of_year <= 13 ~ "Q1",
                                    Week_of_year <= 26 ~ "Q2",
                                    Week_of_year <= 39 ~ "Q3",
                                    TRUE ~ "Q4"))) %>%
  select(Direction, Lag1, Lag2, Lag_avg_abs, Lag_pos_cnt, Quarter)


glimpse(train)
```

```
## Rows: 985
## Columns: 6
## $ Direction   <fct> Down, Down, Up, Up, Up, Down, Up, Up, Up, Down, Down, U...
## $ Lag1        <dbl> 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.372, 0.8...
## $ Lag2        <dbl> 1.572, 0.816, -0.270, -2.576, 3.514, 0.712, 1.178, -1.3...
## $ Lag_avg_abs <dbl> 10.037, 6.823, 9.170, 8.748, 7.888, 8.250, 9.352, 7.583...
## $ Lag_pos_cnt <int> 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 3, 3...
## $ Quarter     <fct> Q1, Q1, Q1, Q1, Q1, Q1, Q1, Q1, Q2, Q2, Q2, Q2, Q2, Q2,...
```

Here I test the model containing all of these predictors, along with an interaction term between `Lag1` and `Lag2`:

```
glm_dir_2 <- glm(Direction ~ . + Lag1:Lag2,
                 data = train,
                 family = "binomial")
```

```r
predicted <- factor(ifelse(predict(glm_dir_2, newdata = test, type = "response")
                           < 0.5, "Down", "Up"))

confusionMatrix(predicted, test$Direction, positive = "Up")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down Up
##       Down   18 16
##       Up     25 45
##
##                Accuracy : 0.6058
##                  95% CI : (0.5051, 0.7002)
##     No Information Rate : 0.5865
##     P-Value [Acc > NIR] : 0.3847
##
##                   Kappa : 0.1613
##
##  Mcnemar's Test P-Value : 0.2115
##
##             Sensitivity : 0.7377
##             Specificity : 0.4186
##          Pos Pred Value : 0.6429
##          Neg Pred Value : 0.5294
##              Prevalence : 0.5865
##          Detection Rate : 0.4327
##    Detection Prevalence : 0.6731
##       Balanced Accuracy : 0.5782
##
##        'Positive' Class : Up
##
```

The classifier performed better on the `test` data than the baseline approach with an accuracy of **60.58%**, but this is slightly worse than the simple LDA & Logistic classifiers which scored 62.5%. Due to the small sample size, I am skeptical as to whether any of these classifiers would continue to perform above the baseline for data beyond 2010.