Building a Market Index

Learning from data when we have a training sample with the correct answer is called **supervised learning**: we find structure in our data using a signal that tells us whether we're doing a good job of discovering real patterns.

But often we want to find structure without having any answers available to us about how well we're doing; we call this **unsupervised learning**. For example, we might want to perform dimensionality reduction, which happens when we shrink a table with a huge number of columns into a table with a small number of columns. If you have too many columns to deal with, this **dimensionality reduction** goes a long way toward making your data set comprehensible. Although you clearly lose information when you replace many columns with a single column, the gains in understanding are often valuable, especially when you're exploring a new data set.

One place where this type of **dimensionality reduction** is particularly helpful is when dealing with stock market data.

The simplest approach is called **principal components analysis**, or **PCA**. The main idea of PCA is to create a new set of 25 columns that are ordered based on how much of the raw information in our data set they contain. The first new column, called the first principal component, or just the principal component for short, will often contain most of the structure in the entire data set.

PCA is particularly effective when the columns in our data set are all strongly correlated. In that case, you can replace the correlated columns with a single column that matches an underlying pattern that accounts for the correlation between both columns.

1. Import and preprocessing data

```
Library('ggplot2')
prices <- read.csv(file.path('stock_prices.csv'), stringsAsFactors
                   = FALSE)
prices[1,]
```

Our raw data set isn't in the format we'd like to work with, so we need to do some preprocessing. The first step is to translate all of the raw date stamps in our data set to properly encoded date variables. To do that, we use the **lubridate** package from CRAN. This package provides a nice function called **ymd** that translates strings in year-month-day format into date objects:

```
library('lubridate')
prices <- transform(prices, Date = ymd(Date)) head(prices)
```

Once we've done this, we can use the cast function in the reshape library to create a data matrix like the table we saw earlier in this chapter. In this table, the rows will be days and the columns will be separate stocks. We do this as follows:

```
library('reshape')
date.stock.matrix <- cast(prices, Date ~ Stock, value = 'Close')
```

The **cast** function has you specify which column should be used to define the rows in the output matrix on the left-hand side of the tilde, and the columns of the result are specified after the tilde. The actual entries in the result are specified using value.

```
summary(date.stock.matrix)
```

Remove missing entries:

```
prices <- subset(prices, Date != ymd('2002-02-01'))
prices <- subset(prices, Stock != 'DDR')
date.stock.matrix <- cast(prices, Date ~ Stock, value = 'Close')
```

2. Correlations

```
cor.matrix <- cor(date.stock.matrix[, 2:ncol(date.stock.matrix)])
correlations <- as.numeric(cor.matrix)
 ggplot(data.frame(Correlation =
correlations), aes(x = Correlation, fill =
1)) + geom_density() +
  theme(legend.position = 'none')
```

3. PCA

As we can see, most of the correlations are positive, so PCA will probably work well on this data set. Having convinced ourselves that we can use PCA, how do we do that in R? Again, this is a place where R shines: the entirety of PCA can be done in one line of code. We use the **princomp** function to run PCA

```
pca <- princomp(date.stock.matrix[, 2:ncol(date.stock.matrix)])
pca
```

From the summary, the standard deviations tell us how much of the variance in the dataset is accounted for by the different principal components. The first component, calledComp.1, accounts for 29% of the variance, while the next component accounts for 20%. By the end, the last component, Comp.24, accounts for less than 1% of the variance. This suggests that we can learn a lot about our data by just looking at the first principal component.

We can examine the first principal component in more detail by looking at its loadings, which tell us how much weight it gives to each of the columns. We get those by extracting the **loadings** element of the **princomp** object stored in **pca**. Extracting **loadings** gives us a big matrix that tells us how much each of the 25 columns gets puts into each of the principal components. We're only interested in the first principal component, so we pull out the first column of the **pca** loadings:

```
principal.component  <-  pca$loadings[,  1]

loadings <- as.numeric(principal.component)

ggplot(data.frame(Loading = loadings),
  aes(x  =  Loading,  fill  =  1))  +
  geom_density() +
  theme(legend.position = 'none')
```

4. Prediction

Now that we have our principal component, we might want to generate our one-column summary of our data set. We can do that using the **predict** function:

```
market.index <- predict(pca)[, 1]
```

How can we tell whether these predictions are any good? Thankfully, this is a case where it's easy to decide whether we like our results because there are famous market indices that we can compare our results against. For this chapter, we'll use the Dow Jones Index, which we'll refer to as just the DJI. We load the DJI into R as follows:

```
dji.prices <- read.csv(file.path('DJI.csv'), stringsAsFactors =
                       FALSE)
dji.prices <- transform(dji.prices, Date = ymd(Date))
```

Because the DJI runs for so much longer than we want, we need to do some subsettingto get only the dates we're interested in:

```
dji.prices <- subset(dji.prices, Date > ymd('2001-12-31'))
dji.prices <- subset(dji.prices, Date != ymd('2002-02-01'))
```

After doing that, we extract the parts of the DJI we're interested in, which are the daily closing prices and the dates on which they were recorded. Because they're in the opposite order of our current data set, we use the **rev** function to reverse them:

```
dji <- with(dji.prices, rev(Close))
dates <- with(dji.prices, rev(Date))
```

Let's put them in a data frame, and plot it for comparison:

```
comparison <- data.frame(Date = dates,
                         MarketIndex = market.index,
                         DJI = dji)
 ggplot(comparison, aes(x = MarketIndex, y = DJI))
+
  geom_point() + geom_smooth(method =
  'lm', se = FALSE)
```

As we can see, those negative loadings that seemed suspicious before, now turns out to be a real source of trouble for our data set: our index is negatively correlated with the DJI.

We can fix this by multiply our index by -1 to produce an index that's correlated in the right direction with the DJI:

```
comparison <- transform(comparison, MarketIndex = -1 * MarketIndex)

ggplot(comparison, aes(x = MarketIndex, y = DJI)) + geom_point()
  +
  geom_smooth(method = 'lm', se = FALSE)
```

This graph makes more sense, it seems to track DJI quite well, what we should do next is to get a sense of how well our index tracks the DJI over time. We will use melt function to get a **data.frame** that's easy to work with for visualizing both indices at once. Then we make a line plot in which the x-axis is the date and the y-axis is the price of each index.

```
alt.comparison    <-    melt(comparison,    id.vars    =    'Date')

names(alt.comparison) <- c('Date', 'Index', 'Price')

ggplot(alt.comparison, aes(x = Date, y = Price, group = Index,
       color = Index)) +
  geom_point() + geom_line()
```

Our first attempt doesn't work very well, because DJI takes on very high values, whereas our index takes on very small values. We can fix this with scale, which puts both indices on a common scale:

```
comparison <- transform(comparison, MarketIndex = -1 * MarketIndex)

comparison <- transform(comparison, MarketIndex = scale(MarketIndex))

comparison <- transform(comparison, DJI = scale(DJI)) alt.comparison

<- melt(comparison, id.vars = 'Date') names(alt.comparison) <-

c('Date', 'Index', 'Price')

ggplot(alt.comparison, aes(x = Date, y = Price, group = Index, color = Index))
+ geom_point() +
  geom_line()
```