post02-jerry-chiang

Jerry Chiang

11/28/2017

Basic Stock Trading Strategies with R

A look at Google (GOOG)

By Jerry Chiang

Today, I will be building on the data manipulation and visualization techniques learned so far to demonstrate how stocks can begin to be analyzed for **"trading signals"** that inform trading strategies.

The post will feature techniques such as data manipulation and visualization through packages such as `dplyr` and `ggplot2` that we have learned in Stat133, in addition to new stock-related packages from CRAN.

Specifically, this post is a continuation of my first post, in demonstrating how after visualizing stocks, we can begin to assess for basic trading strategies. But don't worry if you haven't seen the first post, I'll start with a quick runthrough of what you missed.

Once again, this post was motivated by my personal interest in following the stock markets, and hopefully can serve as a reference for other aspiring traders, or the very least, demonstrate one practical use of R (in analyzing something that's not NBA-related for once, no shade).

What You Missed: A Quick Runthrough of my previous post

Last time, I introduced the `quantmod` package, as a package that allows us to easily retrieve stock data in a useful xts format from sources such as Yahoo!Finance; you can read all about the xts format here.

So let's first install and load the `quantmod` package, in addition to `dplyr` and `ggplot2`, which you should already have:

```
library(dplyr)
library(ggplot2)

#Gets quantmod if you don't already have it
if (!require("quantmod")) {
    install.packages("quantmod")
    library(quantmod)
}
```

For this post, we'll be analyzing Google (GOOG).

Use the following function `getSymbols`, which is part of the package `quantmod`, to extract data for `GOOG`.

```
start = as.Date("2017-01-03", tz = "PST") #any start date, with time zone (tz)
end = as.Date("2017-11-28", tz = "PST") #any end date
getSymbols("GOOG", src = "yahoo", from = start, to = end) #stock symbol and data source (i.e. Yahoo!Finance)
```

```
## [1] "GOOG"
```

```
class(GOOG) #demonstrates that it is an xts object
```

```
## [1] "xts" "zoo"
```

```
head(GOOG) #preview of the xts object created
```

```
##            GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume
## 2017-01-03    778.81   789.630  775.800     786.14     1657300
## 2017-01-04    788.36   791.340  783.160     786.90     1073000
## 2017-01-05    786.08   794.480  785.020     794.02     1335200
## 2017-01-06    795.26   807.900  792.204     806.15     1640200
## 2017-01-09    806.40   809.966  802.830     806.65     1272400
## 2017-01-10    807.86   809.130  803.510     804.79     1176800
##            GOOG.Adjusted
## 2017-01-03        786.14
## 2017-01-04        786.90
## 2017-01-05        794.02
## 2017-01-06        806.15
## 2017-01-09        806.65
## 2017-01-10        804.79
```

Notice that the function takes as inputs the stock abbreviation ('GOOG'), data source ('yahoo'), and the starting and ending dates (year to date). The output of this function is an xts object `GOOG`, but for our purposes, we will convert it to a data frame for easier manipulation; this package just allows us to easily pull data online.

For an explanation of what the column titles mean, refer here for stock jargon.

So now let's convert the xts object `GOOG` into a data frame and apply `ggplot` to visualize.

```
#Convert xts to dataframe for GOOG
GOOG_df = data.frame(date = index(GOOG), coredata(GOOG))
#Notice you access the time stamp through index() instead of row.names, and other columns with coredata()

#Preview of GOOG as a data frame
head(GOOG_df)
```

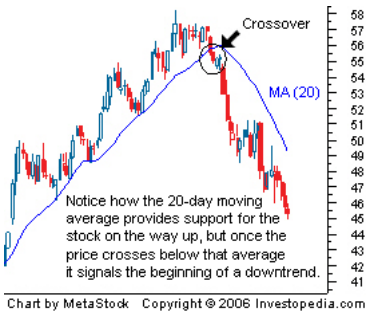
```
##         date GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume
## 1 2017-01-03    778.81   789.630  775.800     786.14     1657300
## 2 2017-01-04    788.36   791.340  783.160     786.90     1073000
## 3 2017-01-05    786.08   794.480  785.020     794.02     1335200
## 4 2017-01-06    795.26   807.900  792.204     806.15     1640200
## 5 2017-01-09    806.40   809.966  802.830     806.65     1272400
## 6 2017-01-10    807.86   809.130  803.510     804.79     1176800
##   GOOG.Adjusted
## 1        786.14
## 2        786.90
## 3        794.02
## 4        806.15
## 5        806.65
## 6        804.79
```

```
#Graph closing GOOG stock prices
#Make sure you have installed the latest ggplot2 or you might get an error message
ggplot(GOOG_df) + geom_line(aes(x = date, y = GOOG.Close)) + labs(title = "GOOG Closing Prices", x = "Date", y = "Price")
```

New Stuff: Visualizing Trading Strategies with Signals

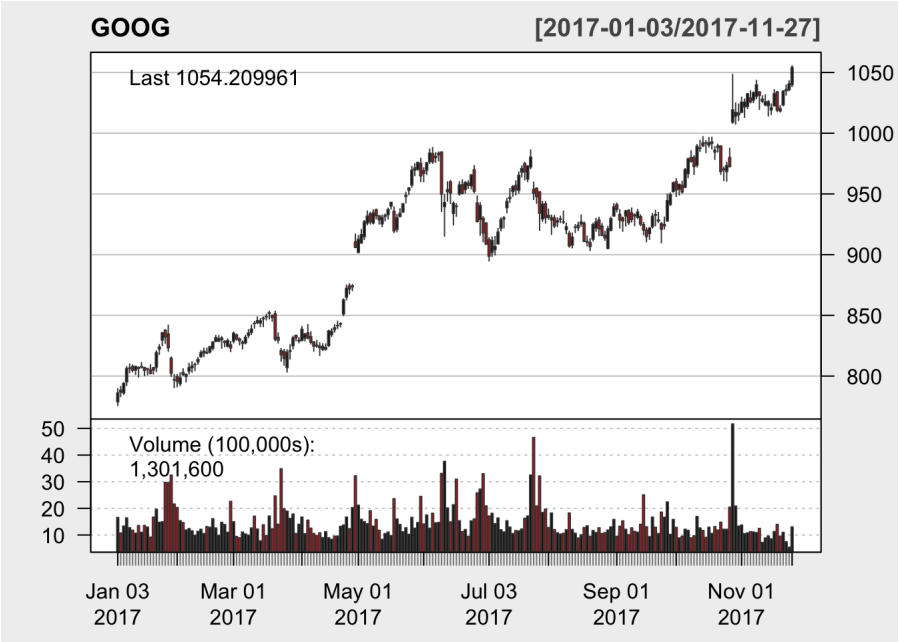
Now that you're caught up to speed, we'll move on to the new stuff: visualizing trade signals from our plots! Specifically, we'll be plotting signals obtained from a graph of moving averages.

An example of a basic signal is the **crossover** (pictured below), where you can easily see a drop in prices for moving averages. This informs a trader to close any long positions.



So first, let's create a candlestick chart of moving averages.

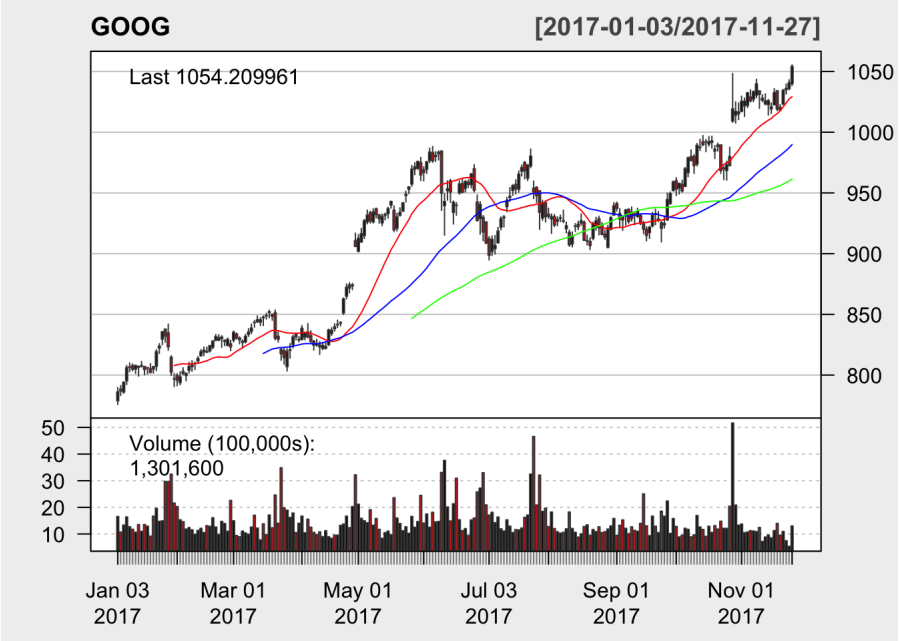
```
#Notice that it takes in our original xts object GOOG as an input and
#essentially combines all column values into a "candle" or boxplot for each date

#Don't worry about the additional inputs; they're purely for a e s t h e t i c s.
candleChart(GOOG, up.col = "black", dn.col = "red", theme = "white")
```



We haven't really made a chart of moving averages yet, only a candlestick chart summarizing the prices and volume. In stock analyses, **k-day moving averages** are essentially the averages of the past *k* days, which is useful for identifying price trends or *signals*.

Add moving average lines onto our candlestick chart with the `addSMA()` function as part of our `quantmod` package. We will be looking at 20-, 50- and 100-day moving averages.

```
#Red for 20 days, blue for 50 days, green for 100 days
addSMA(n = c(20, 50, 100), col = c("red", "blue", "green"))
```



It's important to note that the prices shown are the closing prices for GOOG on those days.

Also note, just like how ggplot works, you can just "layer" lines to our original candleChart without replotting!

The Moving Average Signal

In formulating trading strategies, we can often relate the moving average lines with one another to invoke "signals" that inform our strategies. One concept used by traders is called **"regime"**, which refers to the sign (+/-) of the difference between a fast moving average (higher k days) and a slow moving average (lower k days).

This is often used in the context of **"bullish"** or **"bearish"** stocks.

For example: Whenever we have the fast moving average (e.g. the 20-day line) above the slow moving average (e.g. the 50-day line), we have a *bullish regime*. The opposite would be called a *bearish regime*.

Fun Fact: The terms "bullish"" and "bearish" used to describe stock movement was supposedly a reference to the direction each of these animals would attack from.

Now let's dig deeper into the data points displayed for each of the three moving average lines.

Use the `SMA()` function to create xts objects containing the data points, then convert to a data frame. Recall that this is pretty much what we did with `getSymbols` and the xts `GOOG` object! Do this for all 3 data sets.

```
#20-day moving average value
GOOG20 = SMA(Cl(GOOG), n = 20) #'Cl' represents closing price and 'n' represents k number of days
GOOG20_df = data.frame(date = index(GOOG20), coredata(GOOG20))

head(GOOG20_df)
```

```
##         date SMA
## 1 2017-01-03  NA
## 2 2017-01-04  NA
## 3 2017-01-05  NA
## 4 2017-01-06  NA
## 5 2017-01-09  NA
## 6 2017-01-10  NA
```

```
#50-day moving average value
GOOG50 = SMA(Cl(GOOG), n = 50)
GOOG50_df = data.frame(date = index(GOOG50), coredata(GOOG50))

head(GOOG50_df)
```

```
##         date SMA
## 1 2017-01-03  NA
## 2 2017-01-04  NA
## 3 2017-01-05  NA
## 4 2017-01-06  NA
## 5 2017-01-09  NA
## 6 2017-01-10  NA
```

```
#100-day moving average value
GOOG100 = SMA(Cl(GOOG), n = 100)
GOOG100_df = data.frame(date = index(GOOG100), coredata(GOOG100))

head(GOOG100_df)
```

```
##         date SMA
## 1 2017-01-03  NA
## 2 2017-01-04  NA
## 3 2017-01-05  NA
## 4 2017-01-06  NA
## 5 2017-01-09  NA
## 6 2017-01-10  NA
```

And while we won't actually use the data frames we just created, it's a good conceptual exercise. In fact, `quantmod` has a function to automate the tedious manipulation process that we would have otherwise done with these data frames (lucky us):

Moving Average Convergence Divergence (a.k.a. a fancy name for signals from our moving averages)

As we previously discussed, signals such as the crossover signal applied on moving average lines, allow traders to inform trading strategies (such as going long or short).

Let's explore this concept by visualizing `GOOG`'s **MACD (Moving Average Convergence Divergence) line.**

> *In a moving average crossovers strategy two averages are computed, a slow moving average and a fast moving average. The difference between the fast moving average and slow moving average is this line.*

Apply the MACD line to our candlestick chart

```
addMACD()
```

This is the same plot as before but with an added section below containing our plotted signal (MACD).

For the sake of simplicity, all you need to know is that:

> If the MACD signal (gray) crosses above the signal line (red) traders go long.

> And the MACD signal (gray) crosses below the signal line (red) traders go short.

You'll also notice a bar plot centered at 0 with a -1 to 1 y-axis range. This is another representation of this signal, where:

> If it's negative, traders sell.

> If it's 0, traders stay.

> If it's positive, traders buy.

This matches up exactly with the long and short interpretation.

Using Signals to Estimate Cumulative Returns

To determine the usefulness of signals, traders often take these signals and apply them to historic results and calculate hypothetical returns if these strategies were implemented according to signal interpretations. This is known as **backtesting**, an important step in determining the

accuracy and usefulness of your own tests and strategies by supplying historic data.

To do so, first create an object `macd`, which is what was exactly plotted above through `addMACD()`.

```
data = GOOG[,4] #Get closing prices from xts object GOOG
macd = MACD(data) #MACD function from quantmod outputs an xts object for MACD data
```

Now, a common mistake in **backtesting** is not accounting for **look ahead bias**, which is the idea that in the real-world (present-future), you don't know what happens tomorrow and so in treating historic data, you should apply a 1-day lag to simulate this "unknown".

We apply this lag through the `Lag` function (capital L!) as part of the `quantmod` package and create a new object `sig` to represent a corrected `macd`.

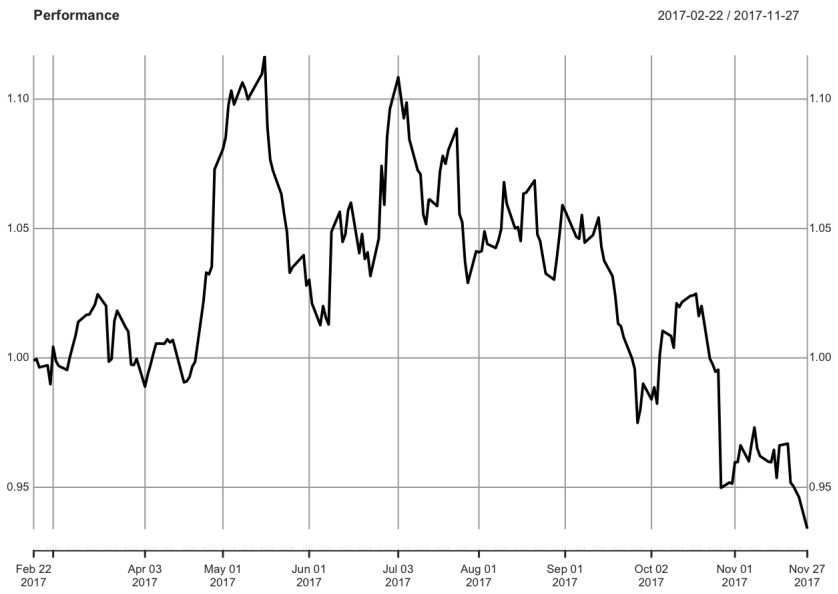
```
sig = Lag(ifelse(macd$macd < macd$signal, -1, 1))
#You'll notice we had a ifelse statement to essentially convert long/short signals into buy/sell/stay signals (whi
ch we explained above); having values as -1 and 1 makes it easier, especially when we go on to calculate returns
```

Now we can calculate returns and create a `returns` object to plot.

```
returns = ROC(data) * sig #ROC function from quantmod, which outputs the % difference between the two closing pric
es
```

Lastly, we can just simply plot our cumulative returns to finish up our backtesting.

```
Performance = exp(cumsum(returns[!is.na(returns)])) #Creates object with cumulative returns
#We remove any NAs, get cumulative sum through cumsum function, then transform into an exponential function throug
h exp to remove negatives

plot(Performance)
```



To interpret the plot, these values represent the returns implementing just our strategy (buy/sell/stay based on MACD line) compared to historical returns (no particular strategy). So for example, 1.10 represents a 10% higher return than the historical return.

Conclusion

In implementing our strategy based on MACD signals, our backtest indicated that our performance compared to the historical performance would have yielded us higher returns most of the time (observe that our plot is only below 1.0 towards the end of October, representing lower returns). Obviously, more thorough and advanced signals and metrics need to be explored, along with more backtests of different strategies and stocks, before we can arrive at a stronger conclusion on if our strategy is accurate and useful and should be used in the real-world.

Takeaway: Conceptually, signals are a useful way to backtest and perfect trading strategies. But practically-speaking, more advanced and thorough tests are required before any meaningful conclusion can be made. And that's why quants at trading firms make $.

Hopefully you've found my post easy to follow along, and interesting, if not practical.

I've tried to combine what we were taught in class (`dplyr`, `ggplot2`) with some new packages and functions (such as `quantmod`) to address and expand on a topic I've previously explored.

All my references are hyperlinked throughout the post but can also be found below for ease of access:

Thanks for reading!

-Jerry

References

- Yahoo!Finance
- More on Xts

- Stock jargon
- Moving averages
- Long position
- Candlestick charts
- Bullish and bearish
- Short position
- Common mistakes in backtesting