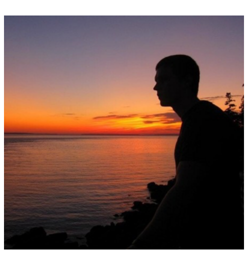

Portfolio Optimization with Python

There are a lot of interesting applications of convex optimization; in this post I'll explore an application of convex optimization in finance. I'll walk through using convex optimization to allocate a stock portfolio so that it maximizes return for a given risk level. We'll use real data for a mock portfolio, and solve the problem using Python. All of the code can be found on GitHub – the code shown here is from **portfolio_opt.py** and uses code in **stocks.py**, which pulls stock data from Yahoo Finance.

Motivation

Let's say you want to invest some money in the stock market. You choose a set of stocks and have a sum of money to invest. How should you distribute the money into the different stocks? There is a general tradeoff between risk and return; with higher potential return we often face higher risk. If we have a goal return in mind, then we should choose the portfolio allocation that minimizes the risk for that return. How can we do this?

Borrowing ideas from modern portfolio theory, we can view the return of each stock as a random variable, and estimate the variable's parameters – namely the mean return and covariance – with past data. Then we solve an optimization problem to find the combination of stocks that maximizes expected return for a given risk level.

We'll choose n different assets, viewing the portfolio as a vector $x \in R^n$. Each x_i will represent the percentage of our budget invested in asset i. As a running example, we'll have $n = 10$ different stocks, identified by their ticker symbols:

```
symbols = ['GOOG', 'AIMC', 'GS', 'BH', 'TM',
           'F', 'HLS', 'DIS', 'LUV', 'MSFT']
```

So for instance, $x_1 = 0.14$ would mean that we invested 14% of our budget in Google. A naive allocation could be investing equal amounts (10%) into each stock, so that $x = [0.10, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10, 0.10]$. Our goal is to do better, and choose the best possible x.

Looking to the Past

First we need an estimate of the expected returns and covariance for the portfolio, which will be used in our optimization. A simple way of estimating a stock's expected return is to look to its past performance; we'll use average yearly return from the past four years. Four is somewhat arbitrary, but the emphasis here is illustrating the optimization approach rather than the estimation of a stock's return. The average historical return will be an n dimensional vector $r_{avg} \in R^n$, where r_{avg_i} is the average return of asset *i*.

Using `avg_return()` from **stocks.py**, we have:

```
start = '1/1/2010'
end = '1/1/2014'

# average yearly return for each stock
r_avg = map(lambda s: stocks.avg_return(s, start, end, 'y'), sym-
bols)
```

Similarly, we can find the portfolio's covariance using past data; the covariance of asset returns is $\Sigma \in R^{n \times n}$. Using `cov_matrix()` from **stocks.py**:

```
# covariance of asset returns
sigma = numpy.array(stocks.cov_matrix(symbols, start, end, 'y'))
```

The last parameter is our goal return threshold, r_{min}:

```
# minimum expected return threshold
r_min = 0.10
```

With these quantities in mind, we can now formulate a convex optimization problem to find the optimal portfolio allocation; that is, the portfolio that achieves the lowest amount of risk while meeting our return goal r_{min}.

The problem

We can use the quantity $x^T \Sigma x$ as a measure of risk for a given portfolio allocation x with covariance Σ. Our objective is to minimize $x^T \Sigma x$. This objective function is a convex function, meaning that we're able to formulate a convex optimization problem, specifically a quadratic program (QP), to find its minimum. To start out, we have the problem:

$$\textbf{minimize } x^T \Sigma x$$

We can build in our goal return as a constraint $r_{avg}^T x \geq r_{min}$. Since we want each x_i to be a percentage (of the budget), we can also add the constraints $\sum_{i=1}^n x_i = 1$ and $x \geq 0$. These are simple linear constraints, maintaining convexity of the new problem:

$$\begin{aligned} \textbf{minimize} \quad & x^T \Sigma x \\ \textbf{subject to} \quad & r_{avg}^T x \geq r_{min} \\ & \textstyle\sum_{i=1}^n x_i = 1 \\ & x \geq 0 \end{aligned}$$

Solving with Python

Now it's time to translate the math into code. In order to setup and solve the problem in Python, we'll use the CVXOPT library. CVXOPT allows us to solve a convex optimization problem as long as we can put it into the proper form. First, we convert the covariance and average return arrays into CVXOPT matrices:

```
r_avg = matrix(r_avg)
sigma = matrix(sigma)
# that was easy
```

Since the portfolio allocation problem is a quadratic program, we need to put our problem into the form:

$$\begin{aligned} \textbf{minimize} \quad & x^T P x + q^T x \\ \textbf{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

In our case, *P* = *sigma*, and *q* = *0*:

```
P = sigma
q = matrix(numpy.zeros((n, 1)))
```

The inequality constraints $x \geq r_{min}$ and $x \geq 0$ are captured using $Gx \leq h$:

```
# inequality constraints Gx <= h
# captures the constraints (avg_ret'x >= r_min) and (x >= 0)
G = matrix(numpy.concatenate((
             -numpy.transpose(numpy.array(avg_ret)),
             -numpy.identity(n)), 0))
h = matrix(numpy.concatenate((
             -numpy.ones((1,1))*r_min,
              numpy.zeros((n,1))), 0))
```

And the equality constraint $\sum_{i=1}^n x_i = 1$ is captured using $Ax = b$:

```
# equality constraint Ax = b; captures the constraint sum(x) == 1
A = matrix(1.0, (1,n))
b = matrix(1.0)
```

We're ready to solve! Throw it into the solver and brace yourself for optimality:

```
sol = solvers.qp(P, q, G, h, A, b)
```

The Results and the Expansions

Here's the result:

```
Optimal solution found.
[0.0, 0.0, 0.0, 0.7, 0.16, 0.0, 0.0, 0.0, 0.0, 0.13]
```

Surprisingly, we should only invest in the 3 of the stocks! Keep in mind that the model that we've used here contains *many* simplifying assumptions; the emphasis here is outlining the approach to casting the problem as a convex optimization problem using real stock data. The great thing is that we can easily change the estimation of expected return, use a different objective function, or introduce new constraints that better reflect our goals, and more generally, the real-world.

Credits

This post was originally inspired by content from Stephen Boyd's great book Convex Optimization. Boyd is also teaching an ongoing online-course called CVX 101 if you are interested in learning more about convex optimization.

Posted on March 23, 2014 by wellecks. This entry was posted in artificial intelligence and tagged Artificial intelligence, convex optimization, finance, optimization, python, stocks. Bookmark the permalink.

← Video: Microsoft Speech Recognition and Machine Translation

These Are Your Tweets on LDA (Part I) →

5 thoughts on "Portfolio Optimization with Python"

Lonewolf
— August 21, 2014 at 9:47 am

Fantastic code, there is a real gap in applying the CVXOPT package into quant finance. This is one of the best posts I've seen, clear and well explained. I really hope that you expand on this in the coming months, LW

Reply

Lonewolf
— August 21, 2014 at 8:12 pm

@ Wellecks, it would be really interesting to see a python example of a portfolio that allowed for long and short positions, with inequality constraints that provided upper and lower bounds of x (say >= -10% & <= 10% per stock), and equality constraints so sum(x) == target net long (.3 or .5 typical) and sum(x.abs()) == target gross leverage (where 1.3 or 2 is typical). Would be interesting to see a min variance vs mean variance target return as you worked up above. There are plenty of examples in R but no one has provided a pythonic/CVXOPT solution.
If its helpful Boyd suggested this should be cast as a SOCP but its unclear to me.

https://groups.google.com/forum/#!searchin/cvxopt/portfolio/cvxopt/7krUdi-IQ5QU/vY3G8GGX0nQJ

Regards LW

Reply

wellecks
— August 22, 2014 at 3:11 am

Thanks for the comment and suggestions! There should be a follow up to this post in the near future

Reply

Lonewolf
— August 22, 2014 at 9:11 am

@ Wellecks , np , let me know if you want to discuss off line, best LW

Reply

Pingi Modern Portfolio Theory – Curiosities in Anansi's Calabash

Leave a Reply

Enter your comment here...

Search …

Recent Posts

Evolving Networks
ECIR 2016 Paper and Presentation
Peering into the Black Box : Visualizing LambdaMART
Learning to Rank Overview
Scala Coursera Highlights

Categories

artificial intelligence
blogging
feedlier
haskell
machine learning
philosophy
programming
projects
technology

Archives

July 2019
April 2016
February 2015
January 2015
November 2014
October 2014
September 2014
March 2014
February 2014
January 2014
December 2013
October 2013

Meta

Register
Log in
Entries feed
Comments feed
WordPress.com