Money Manager

Team Debit-Suisse
Jackson Abascal, Joe Azar-Williams, Anton Relin

May 19 2017

1 The Quantitative Analysis

Out approach for level 1 was fairly straightforward to calculate, as the formulas were all given to use. There were a few optimizations made throughout the implementation of it, the most notable being that all of the relevant information is largely precalculated when the program is loaded.

To approach level 5, we used a version of a method called gradient descent. Typical gradient descent operates on a nice enough (differentiable) function with unbounded real number constraints. In this problem, we represent a portfolio by a tuple of n real numbers $(c_1, \cdots, c_n)$, where $c_i \times 100$ is the percentage that company i should take in our portfolio.

It's required that the sum of portions of everything in our portfolio is equal to one, and also that the return is greater than 12%. To account for these constraints, since general gradient descent does not, we introduced a stochastic element to the general algorithm.

Here is how it works: First, we continually generate random weight samples $(w_1, \cdots, w_{n-1})$ such that $\sum_{i=1}^{n} w_i =\leq 1$. It is implied that the weight of the n-th company is 1 minus the sum of all the others.

As soon as one of these generated samples also meets the constraint that the portfolio it represents has at least 12% return, we continue to the descent stage.

We are trying to minimize the portfolio volatility $\text{vol}(w_1, \cdots, w_{n-1})$. To do this we first compute the gradient, which essentially tells us how we need to shift our portfolio to make it slightly better. It's possible that this slight shift could bring our portfolio out of the range of the constraints, so we do not immediately change it. Instead we generate another sample randomly from within a small radius of this one. If this sample is valid and also better than the current, we take it.

The randomization allows us to not get stuck and to continue optimizing if we get close to the constraint bounds, and also adds some nice random variance that avoids a slight bit of naive over-optimization.

2 The Architecture

All of the quantitative analysis is dynamically loaded into the frontend using a RESTful API. The API is built using Java Spring and is deployed on Heroku. Some notable decisions that allow for the program to be extended include the /getNames route which allows the frontend to accurately query the server regardless of the stocks provided in the data set, extending the /getProportion and /getAllValues routes beyond the specification, allowing for users to return volatility and annual return for any two companies rather than just Ford and Apple, and extending the /getSemiRandomizedPortfolio route to allow for the user to manually input how much they want the minimum annual average return value to be. Heroku is used as a simple and free 'serverless' solution that allows us to access the API from anywhere. If you would like to try out our API, head on over to https://debit-suisse.herokuapp.com/getAllValues?companyOne=F&companyTwo=TSLA and try changing the query params of company one and company two to stocks such as AAPL and SBUX.

3 The Frontend

For the front end, the major design decisions centered around presenting the information in the most concise and useful way possible to the end user.

For level one, we had to consider how best to represent a large and involved dataset in a space efficient and comprehensible way. We decided to solicit input from the user, in the form of a stock symbol, and to display the data for that specific symbol. Furthermore, to streamline the process of locating the desired symbol for the user, we implemented an autocomplete function that pulls symbol data from the server.

For levels two and three, the difficulty was to determine how best to implement a slider that would represent the ratio of one stock to another in an intuitive way. We decided to provide the raw percentage numbers adjacent to the slider itself to reduce the potential for confusion about what was being allocated where, and to place the symbols of the stocks on either side of the slider in an attempt to more firmly establish the idea that the slider was dividing a percentage of the line for each symbol.

Other considerations were in the realm of extensibility. Although we did not have time to implement all of our ideas in this regard, we nonetheless made significant progress. Much of the html in the application is dynamically generated, for example, the correlation data presented in level one. The columns headers in the table are inserted automatically after querying the server for the symbols that will be present in the matrix.