Project B: Portfolio Manager

Now that you've had the experience of extending an existing database-backed web application (RWB), you're ready to design and implement your own. In this project, you will do so, developing an application from scratch that meets the specifications laid out here. This application combines requirements that are common to most interactive web applications with some elements of datamining and analysis using database engines.

You should do this project in teams of 2-3 people. Please see us if you can't find a team.

There is a fair amount of code and other information available in the project handout directory, ~pdinda/339/HANDOUT/portfolio. Be sure to read the README file there.

Overall Requirements

You are to develop a database-backed web application that lets a user do the following:

- Register for an account and log in.
- Access her portfolios. A user may have one or more portfolios.
- Track a portfolio of the user's stocks, including their current value and other aspects of their performance.
- Analyze and predict stock and portfolio performance based on historical performance. You will have access to about 10 years of historical daily stock data for this purpose. Plus you will add a facility to add new data and integrate this new data into analysis and prediction. This part of the project will give you the opportunity to play with simple data mining, and rather more sophisticated prediction techniques.
- Evaluate automated trading strategies using the historic data.

In the following, we'll explain a bit more about what each of these items mean, and give the concrete specifications for each.

What are Portfolios and Portfolio Management?

A portfolio is a collection of investments that is intended to serve some purpose for the portfolio holder. For the context of this project, investments will consist purely of stocks and cash. We will also ignore stock dividends, taxes, margin, and trading costs. There are *many* other kinds of investments that could be included in a portfolio.

While the stocks actually held by an individual investor certainly constitute a portfolio, portfolios are put together for other reasons too, for example to analyze how a particular collection of stocks has done in the past, to predict how well it may do in the future, or to evaluate how well an automated trading strategy might work for the portfolio.

An important investing problem is how to choose investments and their relative proportions such that the risk (and reward) of the portfolio as a whole is controlled. For

example, a 20-something computer scientist may be willing to have a much riskier portfolio than a retired 70-something teacher.

The intuition behind designing a portfolio with a given "risk profile" is pretty simple. The amount of risk of an investment (a stock here) is basically the variance of the value of the investment, sometimes normalized to the average value of the investment (standard deviation over mean, or "coefficient of variation" (COV)). A high COV means the stock is "volatile" and thus riskier. Now, suppose you are trying to choose two stocks. You can not only compute their individual variances, but also their covariation (and thus correlation). If the two stocks are positively correlated, then this means that if both of them are in your portfolio, the combination will be more volatile (higher risk). If they are negatively correlated, then the combination will be less volatile (lower risk). So, "portfolio optimization" is the process of choosing a collection of stocks such that their covariances/correlations with each other combine to give you the variance (risk) that you want while maximizing the likely return (reward). One simplification is to just consider the correlation of each stock with the market as a whole (this is called the "Beta coefficient") in building a portfolio. The Beta of the whole portfolio can thus be made larger or smaller than the market as a whole by choosing the right stocks.

The devil in the details is that in order to build a portfolio like this, we would need to know the *future* values of volatility, covariance, Beta, etc, or of the stock prices themselves. We only have the past ones. So, a very important discipline is prediction, determining how a stock is likely to move in the future based on how it and all other stocks have moved in the past, as well as predicting what the future values of the other statistical measures will be. Since the statistics are almost certainly nonstationary, we will occasionally fail completely in predicting the future. The best we can do is muddle through, but there is a huge range of possibility, and a big part of any serious trading enterprise is datamining historical data to develop better and better predictors.

Another important consideration is automation. We would like to have a computer program that continuously adapts the portfolio holdings in pursuit of maximizing return while controlling risk. These programs are called "trading strategies", and another important goal of datamining of historical financial data is to find them.

Stocks

A share of stock represents a tiny bit of ownership of a company. Companies pay dividends (cash money) on their outstanding shares. The value of a stock is essentially the (discounted) sum of all of the future dividends it will pay. Since no one knows what that is, markets try to estimate it by buying and selling. The price at which a stock is sold (the "strike price") is an estimate of its value---the seller thinks the stock's value is less than the strike price, while the buyer thinks it's more. Notice that a "price event" happens on every sale, and there may be 1000s of sales per day of a stock. If you look at the "stock price" in some typical free web stock quoting service, you're seeing an average of these sales over some interval of time, or, in some cases, the most recent sale.

For the purpose of this project, we will consider only information about the "day range" of a stock. In particular, for each day and stock, we shall have:

- Timestamp
- Symbol (the alphabetic string representing the company – for example, AAPL represents Apple Computer)
- Open (the strike price of the first trade of the day)
- High (the highest strike price during the day)
- Low (the lowest strike price during the day)
- Close (the strike price of the last trade of the day)
- Volume (the total number of shares traded during the day)

You can access 10 years of such historic data from the Oracle database on murphy, in the table cs339.StocksDaily. The information on how to access this data is given in the README file in the handout directory, which you should certainly read. BTW, the "timestamp" in StocksDaily is the Unix time, the number of seconds since January 1, 1970. The same data is also available in MySQL and Cassandra for you to play with those systems, but we want you to use Oracle for this project.

Portfolios

A portfolio consists of the following. You will need to figure out what parts of this need to live in the database and what parts can be generated on the fly.

- Cash account – money that the owner can withdraw or use to buy stocks. If he sells a stock the money it brings in goes here. The owner can also deposit more money from other accounts.
- A list of stock holdings. A stock holding consists of a stock symbol and the number of shares of the stock that the owner has. (We ignore stock lots here since we're ignoring taxes)

The owner of a portfolio should be able to do the following:

- Deposit and withdraw cash from the cash account.
- Record stocks bought and sold (changing the cash and stock holdings in the portfolio)
- Record new daily stock information (like the above), beyond that in the historic data we give you. If you want to be fancy, you can pull this information from your favorite Internet stock quoting service (see the quoting script, quote.pl, which get the current value the service, or quotehist.pl, which gets historical data from the service).
- Integrate stock information from the historical data (which is read-only) and from the new daily stock information to provide a unified view of stock information. That is, the view of information about a stock should be the union of the views in the historic data and in the new stock data that user or script enters.
- Examine the portfolio. The portfolio display should show the total amount of cash, and the holdings of each of the stocks. It should probably also show the information in the following item:
- Get the estimated present market value of the portfolio, the individual stock holdings, and the cash account. We will consider the present market value of a stock to be based on the last close price we have available.
- Get statistics of the portfolio. Based on historic data, we should be presented with information about the volatility and correlation of the stocks in the portfolio. Note that these can take some time to compute, so they should be cached.

- Get the historic prices of the individual stock holdings. It should be possible to click on any holding and get a table and plot of its past performance.
- Get the future price predictions for individual stock holdings. It should be possible to click on any holding and get a plot of its likely future performance, based on past performance.
- Run at least one automated trading strategy on a stock, using historical information, to see how well it performs.

Note that for the last four items, we provide some example code that can be used in the handout directory. We would like to encourage you to go beyond just using it blindly, though. The example code we provide operates over the historical data, while your code should operate over the union of the historical data and your new stock data. If you design the tables you use to represent the new stock data carefully, it will be straightforward to do this union.

Most of the code we give you is designed to run directly from the command-line, not from a CGI script. However, it can be integrated into a CGI program. Also, plot_symbol.pl is an example of using the core functionality (accessing historical stock market data) in a CGI program. Note that it depends on the stock_data_access.pm file – both files need to be copied to your web directory for it to work.

We now describe what we expect as far last four items in a bit more detail.

Portfolio Statistics

It should be possible to display the following information about the portfolio, all of which should be computed from the data in the database.

- The coefficient of variation and the Beta of each stock.
- The covariance/correlation matrix of the stocks in the portfolio. Note that this sounds harder than it is – we give you example code for how to compute this information from the historical data. However, computing this information can be slow, especially for large intervals of time, so you may want to consider some way of caching the correlation statistics.

It should be possible to compute these values over any interval of time. We expect you to do the computation within the database, not just by pulling the data down to the front-end. You may want to look at the scripts get_info.pl and get_covars.pl, which we provide in the handout directory. Recall that you need to integrate both historic data and current data.

Historic Prices of Stocks

From the display of all the stocks in the portfolio, it should be possible to click on any stock to see a new page that shows a graph of the stock's past value (the close price) for some interval of time. It should be possible to set the interval (typical intervals for daily information include a week, a month, a quarter, a year, and five years). Recall again that you need to integrate both historic and current stock price data.

Future Prices of Stocks

This is the most open-ended part of the project. You should create (or use) a tool that predicts the future value (close price) of a stock from its past. In the portfolio view, the user can then click on a stock and see a graph showing your predictions, for some user-selectable interval of time into the future.

It's important to note that this is a databases course, not a course on machine learning or statistics. ***You're not being graded on your predictor.*** We want you to have the experience of integrating this kind of tool with a database. If you think a bit about prediction in addition, that's nice, but not essential. We've included a few fun kinds of predictors in the handout directory, of varying difficulty to use. These include:

- Markov models (markov_symbol.pl to get you started)
- Genetic programming (genetic_symbol.pl – this is probably the toughest to use in the context of this project since it's set up to evolve a predictor[1], but it doesn't actually use the predictor it evolves to make predictions. To use this you'd need to write an interpreter for the mini-language the predictor is implemented in.)
- Time series models (time_series_symbol.pl and time_series_symbol_project.pl – the latter is probably the easiest to integrate.)

The current scripts access only the historic data. You will need to adjust them to use both the historic data and the new data you collect.

Automated Trading Strategy

An automated trading strategy is simply a program that decides on each day whether to buy, sell, or hold a stock, or collection of stocks, based on all the stock information available up to that point. Your system should permit the user to select one stock from his portfolio, one or more trading strategies, and some window of time, and see the results.

We provide code for the "Shannon Ratchet" trading strategy. The Shannon Ratchet balances the amount of one stock and the amount of cash. It's a very simple strategy – it just keeps exactly the same market value in the stock and the cash (50/50), buying and selling stock to do so. Claude Shannon proposed this strategy (to an MIT student group!) in response to the "efficient market hypothesis", which argues that since markets already incorporate all information into prices, the prices must be efficient and thus the price of a stock is really an (unpredictable geometric) random walk[2] The Ratchet converts volatility in the random walk into gains, confusingly making *un*predicabilty into a virtue.

[1] A predictor here means the source code of a function, in a Lisp/Scheme-like language, that computes the next value of a stock based on the values seen over window of previous days.

[2] It's considerably more subtle than this. Please read *A Random Walk Down Wall Street* and *Fortune's Formula*, if you'd like to learn a bit more.

Trading strategies are fun to play around with, especially when you have lots of data to evaluate them on. We encourage you to be adventurous here.

Project Steps

The preceding part of this document is essentially an ***informal specification*** given by me, your client. In the course of the project, you will **formalize it** and **implement it**. You will document this work on a project web site, and ultimately link to your project.

Here are the steps to the project, and what is expected to be made available on the web site for each one.

1. Application design. In this phase, you will "storyboard" and "flowchart" the application, drawing out all the pages and showing how one page switches to the next based on user input. It should be possible to understand the user interface you aim to achieve from the storyboard/flowchart. You should also note the transactions and queries needed for each page or page switch. The document itself can be simple. It's OK to develop the document by hand and then scan it in.
   HANDIN: storyboard and flowchart available on project web site.
2. Entity-relationship design. In this phase, you will create an ER diagram for the data model that is needed to support your application design. Note that you already have some of the ER diagram in the format of the historical data we provide.
   HANDIN: ER diagram available from project web site.
3. Relational design. In this phase, you will convert your ER diagram to the relational model.
   HANDIN: Relational design document on web page.
4. SQL DDL. Here you will write the SQL "create" table/sequence/view/etc code that implements your relational design, including its constraints and relationships as inherited from the ER design.
   HANDIN: Code available from web page.
5. SQL DML and DQL (transactions and queries). In this phase you will write the SQL for the transactions and queries needed to support your application design. Naturally, some of these will templates because you expect them to be generated from your application logic.
   HANDIN: Code available from web page.
6. Application: You will now write the actual application. We want you to use Perl and Oracle. Note that the historical data in the cs339 account is "read-only". Only the historical data is available there, and everyone can access the same.

   We advise you to start with the basic portfolio functionality, which should be straightforward. Then add functionality to introduce new quotes, either from the user or automatically from a stock quote server. Next, add the capabilities for historical graphs. Finally, continue on to the predictions and the automated trading strategy.

   Notice that while we encourage you to have fun with the predictions and automated trading strategy parts of the project, we also provide "canned"

examples of both which you can use – you just have to figure out how to interface with them.

HANDIN: The code itself and a running instance on the web site.

Where to go for help

The goal of this project is to learn how design and build a database-based web application starting from a rather informal specification. We don't want you to get stuck on the details of Perl, CGI, or HTML, so please use the following resources:

- Piazza. Don't forget to help others with problems that you've already overcome.
- Office hours. Make sure to use the office hours made available by the instructor and the TA.
- Handouts: the high-level introduction to the Perl programming language and the JavaScript model will come in handy.
- Videos. There are videos available on the course web site that will bring you up to speed on Linux, if needed.
- Web content. You will find many examples of Perl CGI and DBI programming on the web.
- Other portfolio systems. These things are easier to explain by example. We strongly suggest that you look at Google Finance, Yahoo Finance, E*Trade, and other sites to get a sense of a portfolio manager looks like. Note that very few sites provide the prediction or automated trading strategy features to end-users, though. These are usually tools targeted at institutional investors, large scale traders, or developed internally at trading firms.

Hand-in

To hand in the project, you will do the following:

- Update your project web page with all of the hand-in materials listed above
- Provide a link to your *running* application
- Tar up your whole project page, code, etc. Everything.
- On or before the project deadline, email the instructor and the TAs with a copy of the tar file, a URL for your project page, and the account/password to access your running application.

Extra Credit (30% Maximum)

You can gain extra credit by trying the following extensions. If you are interested in doing extra credit, talk to us first, so that we can determine the precise credit that might be available for what you propose to do.

Additional Portfolio and Stock Statistics: There is a wide range of statistics available to summarize a portfolio or a stock, in addition to the ones required. Read about some and then implement them within your application. You should use the database to compute the summary statistics whenever possible.

Additional Predictors and Trading Strategies: You only need to implement one of each, but if you implement more, we will give you extra credit.

Automated Stock Updates: Learn how to interface with a stock quoting service (we give example code for this) and develop code that continuously records new historic data for all stocks in all portfolios. Notice that this tool would be a daemon that runs independently of web accesses and inserts new data into your database.

Best Possible Return With Hindsight: The idea here is to determine for a given stock, for some interval of time in the past, what the maximum amount of money that any trading strategy could have made, if allowed to trade once per day.

Holdings-based Operations: In the main project, you only need to consider the price of an individual stock within the historical plot, prediction plot, and trading strategy components of the project. Even in the portfolio statistics part of the project, only covariance is computed across the stocks the user holds. In a more powerful portfolio system, it would be possible to do historical plots, prediction plots, and trading strategies over groups of stocks, for example the group in the portfolio. Furthermore, instead of considering the price of a stock, you would consider the market value of the portfolio. For example, if you own, say, 100 shares of AAPL and 200 shares of GOOG, you would plot the market value of the portfolio as 100*close_price_of_APPL(t) + 200*close_price_of_GOOG(t) as a function of time (t). Note that both the price of a stock and the amount of shares you own vary over time. You can gain extra credit by making it possible to plot portfolio market value, predict portfolio market value, and/or do automated trading on portfolio market value.