Building a Trading Robot

Machine Learning Engineer Nanodegree

Calvin Ku
September 27, 2016

Definition

Project Overview

Problem with trading is that you never know when is the best time to buy or sell a stock, as you never know if the stock price will go up or go down in the future. This simple trading bot is an attempt to solve this problem.

Given the historical data of a stock, our chimp will tell you whether you should buy or sell or hold a particular stock today (in our case, the JPM).

Data used in this project

The only data used in this project is the JPM historical data collected from Yahoo Finance. The data ranges from December 30, 1983 to September 27, 2016. We don't use S&P 500 ETF as ETFs are generally arbitrageable which can render the techniques we will use in this project (namely, VPA) useless.

Problem Statement

This project is about building a trading robot. In this proejct we will call it the Chimp. The Chimp is built to give the common user suggestions on whether to buy or sell or hold a particular stock on a particular trading day. The goal of this project is to build a trading robot that can beat a random monkey bot. Inpired by the famous saying of Princeton University professor Burton Malkiel in 1973 that "A blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by experts" and the Forbes article Any Monkey Can Beat the Market, instead of competing on a portfolio basis, we set our battlefield on JPM.

We will use JPM as an example in this project but the same method can be applied to any stock. In the end we will evaluate our method by giving the monkey bot (which chooses the three actions equally on a random basis) and our Chimp 1000 dollars and see how they perform from September 26, 2011 to September 27, 2016 on JPM.

Metrics

In this project we use the cash in hand plus the portfolio value (number of shares in hand times the market price), the total assets as the metric. We also and define the reward function to be the ratio of the difference of the assets divided by the previous assets between the current state and the previous, i.e.:

$$R(s_i) = \frac{Cash(s_{i+1}) + PV(s_{i+1}) - Cash(s_i) - PV(s_i)}{Cash(s_i) + PV(s_i)}$$

This simple metric is in line with what we want the trading bot to achieve in that our ultimate goal is to make as much profit as possible given what we have put into the market, and it doesn't matter whether it's in cash or in shares.

Analysis

Data Exploration

First look

Let's first take a glance at some statistics of our data and then see if there's any missing values

```
Start date: 1983-12-30 00:00:00
End date: 2016-09-27 00:00:00

              Open         High          Low        Close        Volume  \
count  8256.000000  8256.000000  8256.000000  8256.000000  8.256000e+03
mean     46.472522    47.047441    45.882564    46.475389  1.290023e+07
std      20.463781    20.687878    20.250667    20.470789  1.835588e+07
min      10.250010    10.875000     9.624990    10.125000  3.780000e+04
25%      35.124990    35.527499    34.625010    35.060001  1.882275e+06
50%      41.000010    41.500000    40.500000    40.965000  7.037000e+06
75%      52.612501    53.092500    52.000000    52.542501  1.529572e+07
max     147.000000   149.124985   144.000000   147.000000  2.172942e+08

         Adj Close
count  8256.000000
mean     22.914243
std      17.191570
min       1.514014
25%       5.380651
50%      23.981890
75%      33.764691
max      68.076434
```

Since we won't be using data prior to 1993 for training, we can use SPY (S&P 500 ETF) to get trading days and see if we have any data missing for JPM.

```
Number of trading days: 5960
Number of days where JPM are traded: 5960
```

It seems to be good. Let's look at the first few lines:

	Open	High	Low	Close	Volume	Adj Close
1983-12-30	44.000008	44.500006	43.500014	44.000008	211500.0	2.602623
1984-01-03	43.937506	44.249986	43.624979	44.000008	385500.0	2.602623
1984-01-04	44.843758	45.874979	44.249986	45.874979	292500.0	2.713529
1984-01-05	46.812508	47.375008	46.250008	47.375008	344100.0	2.802256
1984-01-06	46.875014	47.375008	46.375021	46.875014	194400.0	2.772681

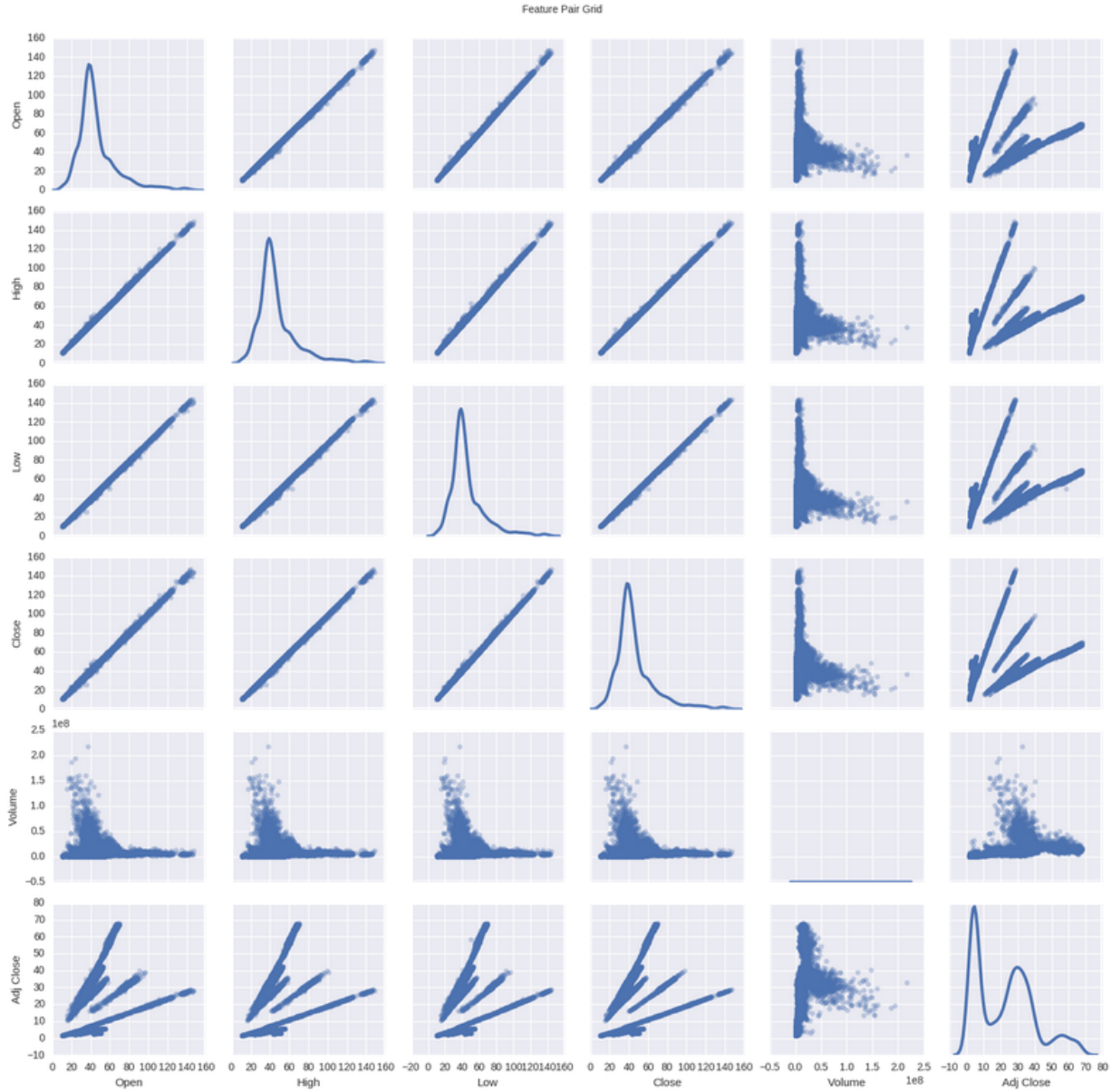
We can see that we have six columns: Open, High, Low, Close, Volume, Adj Close. The Adj Close is the closing price of that day adjusted for “future” dividends payout and splits. For our usage, we will need to adjust the rest of columns as well.

Exploratory Visualization

Now let’s have a look on the performance of JPM itself:

Starting from the beginning, the stock price generally has a upward trend, with a bad time from 2001 to 2003 and the crush at the end of 2008.

Now we can take a look at the correlations between the variables:



We can see it clearly on the `Adj Close` rows and columns there are several lines. This is due to the fact that the `Adj Close` varaible are adjusted for times where there are splits and dividends payout. From here we know we'll need to adjust other variables to match it.

Algorithms and Techniques

Algorithms overview

The trading bot (from now on we refer to it as the *Chimp*, coined with our wishful expectation that she will be smarter than the Monkey Trader) consists of two parts. In the first part we implement Q-learning and run it through the historical data for some number of iterations to construct the Q-table. The Chimp can then go with the optimal policy by following the action of which the state-action pair has the maximum Q value. However, since the state space is vast and the coverage of the Q-table is very small, in the second part we use supervised learning to train on the Q-table and make predictions for the unseen states.

Reinfocement learning

Q-learning

The core idea of reinforcement learning is simple.

1. The Chimp senses its environment (data of some sort)
2. The Chimp takes an action
3. The Chimp gets a reward for that action she took
4. The Chimp "remembers" the association between the state-action pair and the reward, so next time when she is in the same situation, she'd carry out the action that she thinks best, under the condition that,
5. The Chimp has a really good memory so she doesn't just remember the immediate reward for each state-action pair, she also remembers all the rewards that are prior to and lead to the current state-action pair so that she can maximize the total reward she get

One way to do this is to use a method called **Q-learning**. At the core of Q-learning is the **Bellman equation**.

In each iteration we use the Bellman equation to update the cell of our Q-table:

$$Q(s,a) \longleftarrow (1-\alpha)\, Q(s,a) + \alpha\,(R(s) + \gamma\, max_{a'}\, Q(s^{'}, a^{'}))$$

where (s, a) is the state-action pair, α the learning rate, $R(s)$ the reward function, and γ the discount factor.
And then the Chimp will follow the policy:

$$\pi(s) = argmax_a Q(s,a)$$

Although we don't have the Q value of any state-action pair to begin with, the reward function contains the "real" information and throughout the iterations that information will slowly propagate to each state-action pair. At some point the Q values will converge to a practical level (hopefully), and we end up with a Q table in psudo-equilibrium.

Exploration-exploitation dilemma

However, there's a catch. How does the Chimp know what to do before she has learned anything?

One important concept of reinforcement learning is the **exploration-exploitation dilemma**. Essentially it means when we take an action, we have to choose between whether to explore new possibilities, or just to follow what we've known to be best, to the best of our knowledge. And of course, if we don't know much, then following that limited knowledge of ours wouldn't make much sense. On the other hand, if we've already known pretty much everything, then there's just not much to explore, and wandering around mindlessly wouldn't make sense either.

To implement this concept, we want our Chimp to set out not having a bias (not that she's got much anyways), so we introduce the variable ϵ, which represents the possibility of the Chimp taking random actions. Initially we set $\epsilon = 1$, and gradually decreases its value as the Chimp getting to know more and more about its environment. As time goes, the Chimp will become wiser and wiser and spends most of her time following what's best for her, and less time on being an explorer.

Supervised learning with random forest

For the supervised learning part, we will use the random forest. The random forest doesn't expect linear features or assuming features interacting with each other linearly. It has the advantage of a decision tree which generally can fit into any shape of data, while being ensemble and eliminates the problem of a decision tree being easily overfitting. The random and ensemble nature of the algorithm makes it very unlikely to overfit on the training data. Since we are combining supervised learning with reinforcement learning, the problem gets more complicated and we will have more parameters to tune, and it's good to have an algorithm that kind of works "out of the box" most of the time. On the other hand, random forest handles high dimensionality very well, which makes feature engineering a bit easier where we don't need to worry too much about whether it's the newly engineered features not representative or it's the algorithm not able to handle the enlarged dimensionality. In addition to this, random forest itself is very easy to tune. We can easily grid search through number of choices of features for each splitting and the number of trees. The ensemble nature of the algorithm also makes it scalable when we need to: 1. train on more data, 2. build more trees, 3. take more stocks into account. Overall, random forest generally gives good results and it has been recognized that ensemble algorithms like random forest perform over other traditional algorithms in the Kaggle community over the years and have become quite popular.

How it works

The random forest is an ensemble method, which "ensembles" a bunch of decision trees. Each decision tree is generated by creating "nodes" with features in our dataset.

Decision tree

In the training stage, a data point comes down and through the nodes of the decision tree. Each node classifies the data point and sends it to the next node. Say, for example we are classifying people to determine whether their annual income is above or below average, and one feature of our data is gender. And we will probably have values like male/female/other. Now say this data point is a female, then it will get sent down the path accordingly to the next node. The node at the bottom of the decision tree is sometimes referred to as a leaf. Our data point will end up in one of the leaves, and the label of the data point is going to mark that leaf. For example, if our data point is above income average, then we mark that leaf as "above count +1". At the end of the training, all leaves will be marked with labels above or below.

In the predicting stage, we run data down the decision tree and see which leaves they end up with. And then we assign the data the labels of the leaves accordingly.

The ensembleness

We now know how each decision tree is constructed and have a forest of decision trees. The last step is to get these decision trees to vote. If 10 trees say this person is above and 5 say below, we predict the person as above.

Randomness of random forest

As said earlier, the decision trees are constructed with the features of our dataset. However, not all of the features are used to construct each decision tree. This is where the random part of the algorithm comes in. Most implementation employs a method called bagging, which generates m sub-datasets from the feature space of the orginal dataset by sampling with replacement, where the size of the sub-datasets is n', relative to the size of the feature space of the original dataset, n. The features of each bag are then used to construct a decision tree model.

Other parts of random forest

We won't cover everything about the random forest here. However it's worth noting some of the more important specifics of random forest that are not covered here:

- Binning of the continuous variables—which are done slightly differently from implementation to implementation
- Splitting methods—when constructing the decision trees we need to decide which feature to be placed on top of the tree and which to be put at the bottom.
- Voting methods—we can decide to give each decision tree with the same voting power, or not.
- Modification of the algorithm for regression problems (recursive partitioning)

Benchmark

We shall set three different benchmarks here. One theoretical, and two practical.

Theoretical benchmark

Since our goal is to make as much money as possible. The best role model we can have, would be a *God Chimp*. A God Chimp is a Chimp that can foresee the future, and trades accordingly. In our case, this is not very hard to do. We can instantiate a Chimp object and get her to iterate through the entire dataset, until the whole Q-table converges. And with that Q-table in hand, we can get the psudo-perfect action series, which can make us a good deal of money. We then compute the accuracy of the action series of our *mortal Chimp* for that of the God Chimp. Theoretically speaking, the higher the accuracy, the closer the behavior of the mortal Chimp to the God Chimp, the more money the mortal Chimp would be making.

Practical benchmarks

That said, the ups and downs of the stock market are not really uniformly distributed. This means our mortal Chimp could have a very decent accuracy for the God Chimp, but say, screwed up most of the important part. And therefore not really doing that well. Conversely, it may appear that our mortal Chimp is doing really terrible mimicing the God Chimp, but still makes a lot of money. So we will need some practical benchmarks that are more down to earth.

We shall test our Chimp against 100,000 random Monkeys and the Patient Trader we have defined earlier. Since these two naive traders don't get influenced by the media or manipulated by the market makers, they are proven to perform better than the average investor. We are happy as long as our Chimp can perform better than the Monkey, which means our Chimp is at least better than chance (and therefore better than any average person), and also it'd be great if she can beat the PT. However beating the PT in general means beating the market, which isn't something really easy to do. So we wouldn't expect that much here.

Methodology

Data Preprocessing

Adjust prices

	Open	High	Low	Close	Volume	Adj Close
1983-12-30	44.000008	44.500006	43.500014	44.000008	211500.0	2.602623
1984-01-03	43.937506	44.249986	43.624979	44.000008	385500.0	2.602623
1984-01-04	44.843758	45.874979	44.249986	45.874979	292500.0	2.713529
1984-01-05	46.812508	47.375008	46.250008	47.375008	344100.0	2.802256
1984-01-06	46.875014	47.375008	46.375021	46.875014	194400.0	2.772681

As said earlier, we need to adjust the prices of Open, High, Low, Close, Volume. This can be done by getting the adjustment fact by dividing Adj Close by Close. We then multiply the prices by this factor, and divide the volume by this factor.

	Open	High	Low	Close	Volume	Adj Close
1983-12-30	2.602623	2.632198	2.573048	44.000008	3.575624e+06	2.602623
1984-01-03	2.598926	2.617409	2.580440	44.000008	6.517272e+06	2.602623
1984-01-04	2.652532	2.713529	2.617410	45.874979	4.945011e+06	2.713529
1984-01-05	2.768984	2.802256	2.735712	47.375008	5.817363e+06	2.802256
1984-01-06	2.772681	2.802256	2.743106	46.875014	3.286531e+06	2.772681

	Open	High	Low	Close	Volume	Adj Close
2016-09-21	66.839996	67.129997	66.309998	66.839996	14116800.0	66.839996
2016-09-22	66.989998	67.419998	66.839996	67.389999	12781700.0	67.389999
2016-09-23	67.389999	67.900002	67.180000	67.250000	13967400.0	67.250000
2016-09-26	66.599998	66.800003	65.540001	65.779999	16408100.0	65.779999
2016-09-27	65.410004	66.410004	65.110001	66.360001	13580600.0	66.360001

Features engineering using volume price analysis

Volume price analysis has been around for over 100 years, and there are many legendary traders who made themselves famous (and wealthy) using it. In addition to this, the basic principle behind it kind of makes sense on its own, that:

1. Price can only be moved by volume; large spread pushed by large volume and small spread by low volume
2. If it's not the case, then there's an anomaly, and you need to be cautious

But then people, especially practioners, tend to think of it as an art rather than science, in that even though you have some clues what's going on on the market, you still don't know what the best timing is. And it takes practice and practice until you "get it".

For we data scientists, everything is science, including art. If a human can stare at the candlesticks telling you when to buy or sell, so can a computer. Thus the following features are extracted from the raw dataset:

For volume:

- -1d Volume
- -2d Volume
- -3d Volume
- -4d Volume
- -5d Volume
- 10d Average Volume
- 21d Average Volume
- 63d Average Volume

For price:

- -1d Spread
- -2d Spread
- -3d Spread
- -4d Spread
- -5d Spread
- 10d Spread
- 21d Spread
- 63d Spread

For wick:

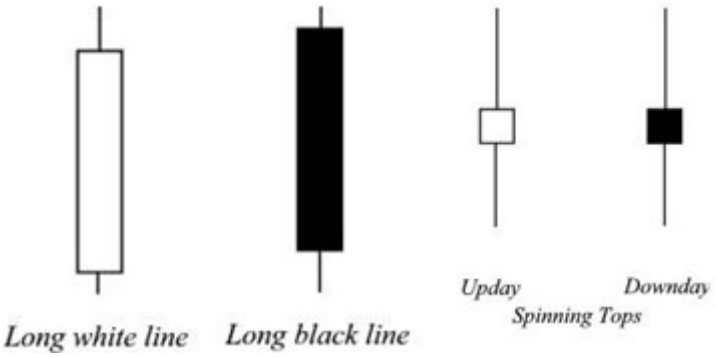
- -1d upperwick/lowerwick
- -2d upperwick/lowerwick
- -3d upperwick/lowerwick
- -4d upperwick/lowerwick
- -5d upperwick/lowerwick
- 10d upperwick/lowerwick
- 21d upperwick/lowerwick
- 63d upperwick/lowerwick

where *-nd* represents *n* day in the past.

More details on feature engineering

The reason why we choose 5, 10, 21, 63 days is because these are the common time ranges used in technical analysis, where 5 days correspond to one trading week, 10 to two, and 21 days correspond to one trading month, 63 to three. We don't want to explode our feature space so to start with we use the most recent 5-day data with longer term average data.

Spread and wicks are terms to describe the status of the candlestick chart (see below).



The spread describes the thick body part of the candlestick which shows the difference of the opening price and the closing price. The time frame (in terms of opening/closing) can range from minutes to months depending on what we want to look at (in our case, the time frame is one day). The wicks are the thin lines that extend at the top and the bottom, which describe whether there are stocks traded at prices beyond opening/closing prices during the day (or the specific time frame of interest). As shown in the picture, we can have white or black bodies on the candlestick chart to indicate the direction of the pricing movement, with white meaning $\text{closing price} > \text{opening price}$ and vice versa. On the other hand, a *candle* can have a upperwick and/or a lowerwick or none at all.

Note that to implement Q-learning we need to make the variables discrete. We use 100 day maximum and 100 day average to divide the above features and get relative levels of those features.

Trading price

We set the trading price of each trading day to be the Adjusted Close:

$$TradePrice = Adj\ Close$$

This information is not available to the Chimp. The properties of the Chimp get updated with this information when she places an order. The portfolio value also gets updated using this price.

Implementation

The problem with time series data in contrast to cross-sectional ones is that we cannot rely on conventional methods such as cross-validation or the usual 4:3:3 train-cv-test testing framework, as all of these methods are based on the assumption that all our data are drawn from the same population and a careful selection of a sample (samples) with proper modelling can say a lot about the entire population (and of course, about the other carefully drawn samples). However, we are not so lucky when it comes to dealing with time series data, mostly because:

1. Most if not all of the time our model really isn't the underlying model, that is, the data doesn't really come from the model we use. So it works only for really limited range of time and as the time series gets longer the difference between our model on the training set and the underlying model starts to show.
2. Essentially the system we are looking at is a time-dependent one so the underlying model itself most likely changes from time to time (unless we're talking about some grand unified model that can model the whole universe), in which case, assuming one model structure will work on the entire period of data is just wishful thinking.

That said, a lot of time we wish that in the process of our research, we can find some "invariants" (or least psudo-invariants) in our data that doesn't change as time goes. Still, we don't know if they are out there or not.

Training-testing framework

As said above, we will thus employ a training-testing framework that rolls as time goes. In our case, we keep 35 trading months of data for training (how this is determined will be shown later), and use the model to predict for 7 days, and since we are probably more interested in the performance of the JPM of the recent years we will proceed as following:

1. Use data from September 25, 2006 to September 25, 2011 as our validation dataset, and
2. Use data from September 26, 2011 to September 27, 2016 as our test dataset
3. Use 35 months of data prior to prediction period as our training set.

Parameters for Q learning

We start off setting our parameters as follows:

- Discount factor: $\gamma = 0.75$
- Learning rate: $\alpha = \frac{1}{t}$, where t is the number of time a state-action pair gets updated
- Eploitation-exploration ratio: $\epsilon = \epsilon - \frac{1}{\text{iter\_number}}$
- Number of iteration: `iter_number = 5000`

Other assumptions

1. **Trading price**—as mentioned earlier, we assume the trading price to be the same as the Adjusted Close.
2. **No transaction cost**—this can simplify the problem so that we can focus on modelling.

3. **Two actions**—we limit ourselves to only two actions, buy and sell. Again since there's no transaction cost, buying when there's no cash is equivalent to hold (and similar for the sell case). By limiting the size of the action space it's easier for our Q value to converge.

Benchmarks for the two phases

As mentioned above, we will use a roll-forward training framework to build our Chimp. We will first give it a few tries and fine-tune our parameters on the validation dataset. We shall call this the **validation phase**. And then we'll move on the to test on the test dataset, which will be referred to as the **test phase**.

We will set up our two benchmarks for the two phases for comparison. To recap, which include:

1. Performances of the Patient Trader
2. Performances of the Monkey
3. Action series of the God Chimp

Performances of the Patient Trader

Validation phase (2006-9-25 ~ 2011-9-25)

Start

$$Cash_{init} = 1000.00$$

$$Share_{init} = 0$$

$$PV_{init} = 0$$

$$Trading\ Price_{init} = 36.61$$

$$Share_{start} = floor(\frac{1000.00}{36.61}) = 27$$

$$PV_{start} = 36.61 \cdot 27 = 988.47$$

$$Cash_{start} = Cash_{init} - PV_{start} = 1000.00 - 988.47 = 11.53$$

$$Total\ Assets_{start} = Cash_{start} + PV_{start} = 1000.00$$

End

$$Cash_{end} = 11.53$$
$$Share_{end} = 27$$
$$Trading\ Price_{end} = 27.42$$

$$PV_{end} = 27.42 \cdot 27 = 740.34$$

$$Total\ Assets_{end} = Cash_{end} + PV_{end} = 11.53 + 740.34 = 751.87$$

We can calculate the annual ROI by solving the following equation for r_{val}:

$$(1+r)^{1260/252} = 0.7519$$

$$\Longrightarrow \boxed{r_{val} = -0.05543464 \approx -5.54\%}$$

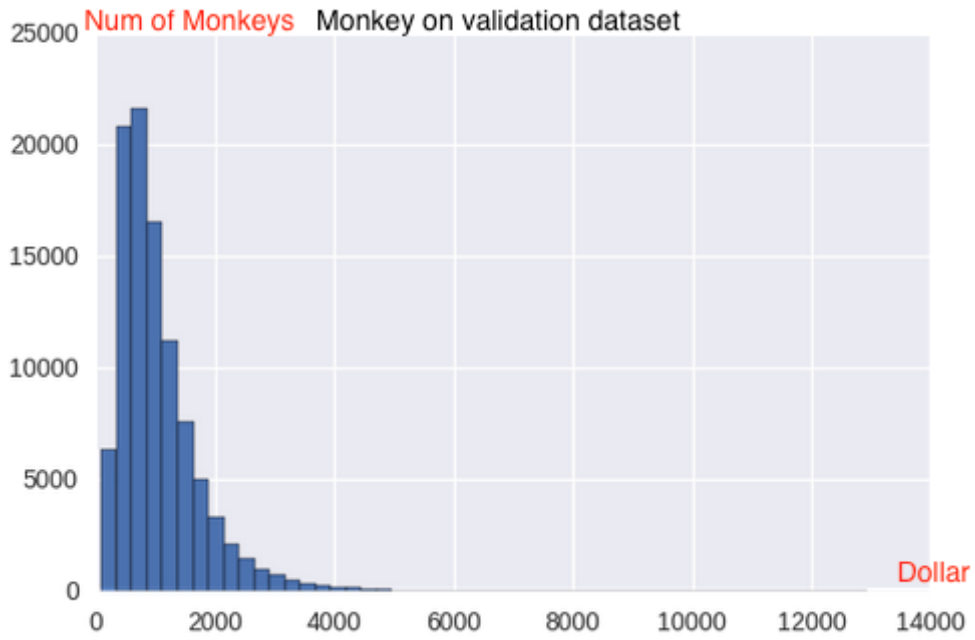
Test phase (2006-9-25 ~ 2011-9-25)

Similarly, we will have

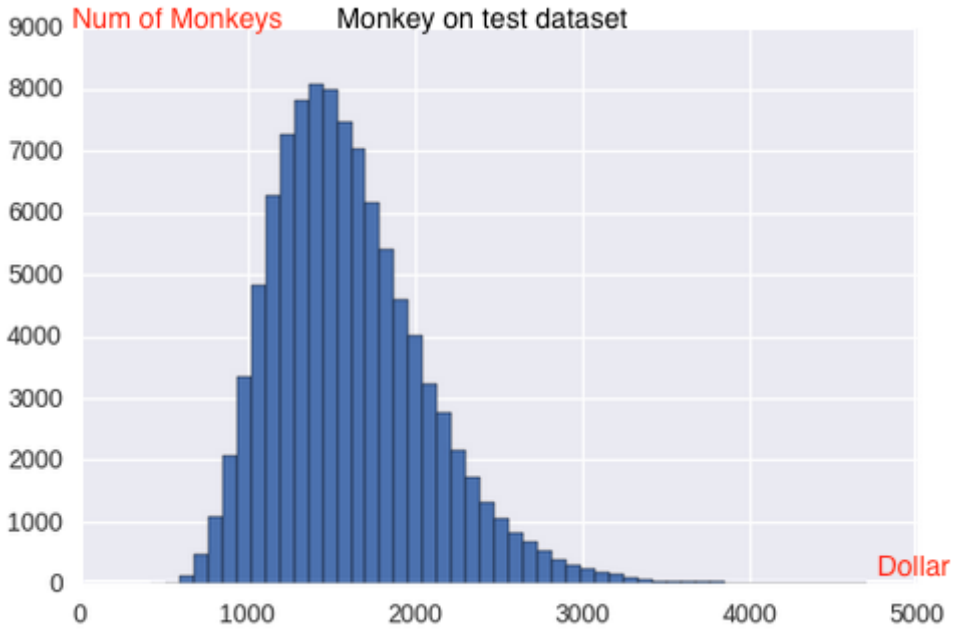
$$\boxed{r_{test} = 0.1912884 \approx 19.13\%}$$

Performances of the Monkey

We use a `MonkeyBot` class which will place one and only one order randomly everyday. We iterate it through the time frame we choose 100,000 times and we get the following distributions:




```
count    100000.000000
mean       1060.238834
std         728.393854
min          85.447874
25%         575.907935
50%         872.079011
75%        1327.014578
max       12935.629597
dtype: float64
```




```
count    100000.000000
mean       1606.960713
std         464.488921
min         422.648034
25%        1272.686842
50%        1542.634717
75%        1869.730139
max        4704.443561
dtype: float64
```

Validation phase (2006-9-25 ~ 2011-9-25)

Using the mean we can calculate r_{val}:

$$(1 + r_{val})^{1260/252} = 0.8721$$

$$\Longrightarrow \boxed{r_{val} = -0.02699907 \approx -2.70\%}$$

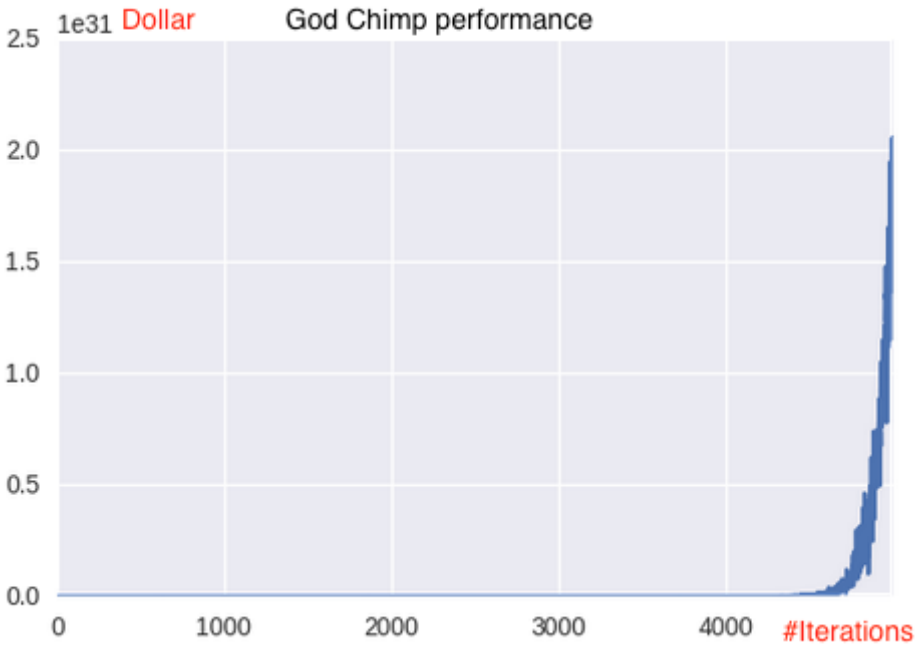
Test phase (2006-9-25 ~ 2011-9-25)

Similarly,

$$\Longrightarrow \boxed{r_{test} = 0.09056276 \approx 9.06\%}$$

Action series of the God Chimp

We let the God Chimp run through all the data to get the converged Q-table.

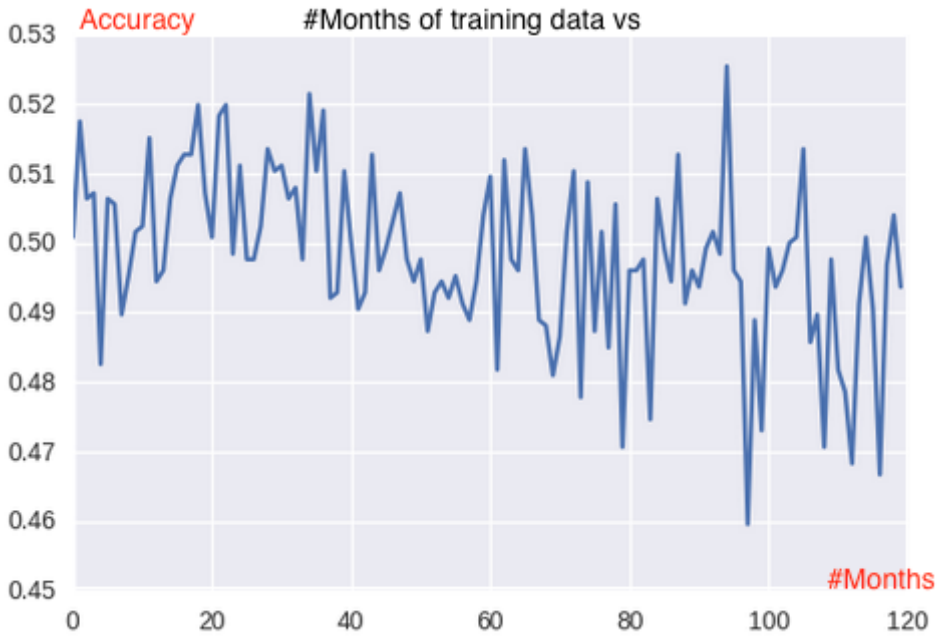



```
count      8156
unique        2
top         Buy
freq       4123
dtype: object
```

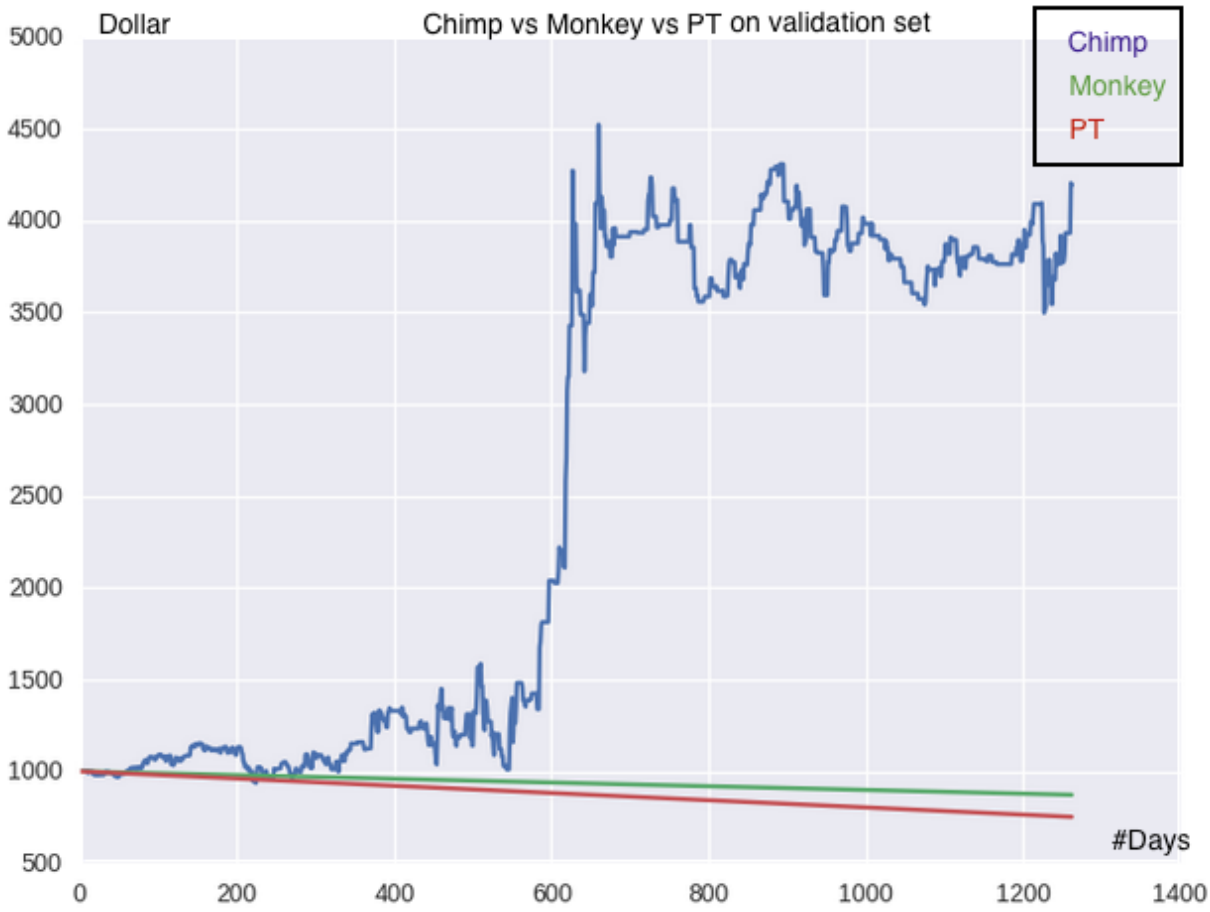
Finding the right size for training set

As said earlier, one problem with time series data is to find the training window size wihtin which the data can be seen as being drawn from the same population as the data we want to predict. Then of course we can generalize what we have learned/modelled from the training to the cross-validation/test dataset.

To do this we can make use of the God Chimp's Q-table we just got and get:



We can see a trend of accuracies going up and then down again. Here we choose 35 months of data to build our model.



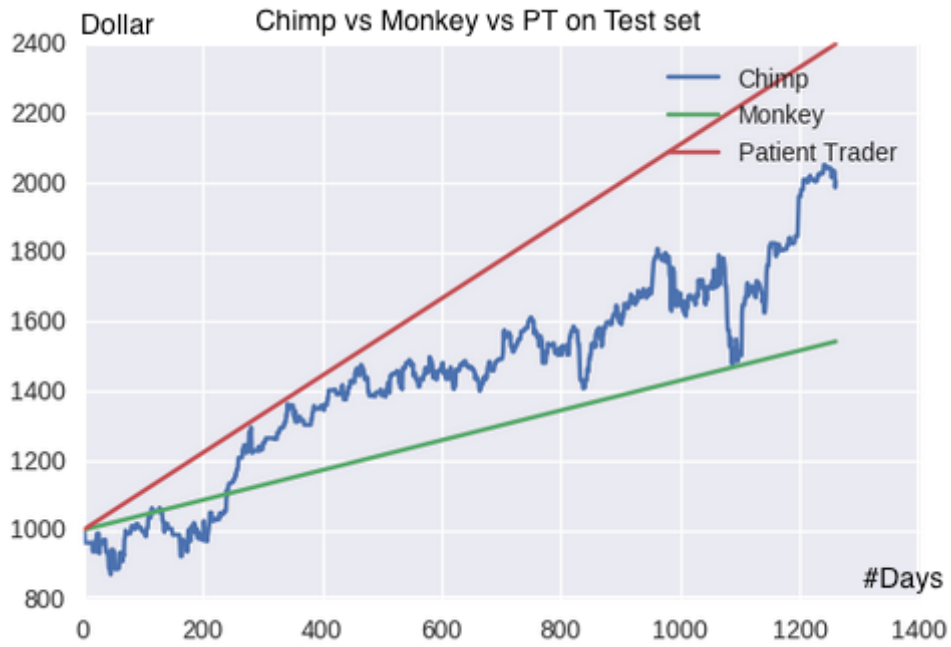
The result turns out to be really good!

And if we look at the equivalent annual ROI:

$$ROI = (1 + r)^{1262/252} = 4.1928$$

$$\implies r = 0.3313846 \approx 33.14\%$$

The result is quite impressive compare to the Patient Trader (-5.54%) and the Monkey (-2.70%). Let's give the test set a shot!



The result turns out to be above the median by quite a lot (above 82% of the Monkeys).

And when we look at the equivalent annual ROI:

$$ROI = (1 + r)^{1260/252} = 2.0025$$

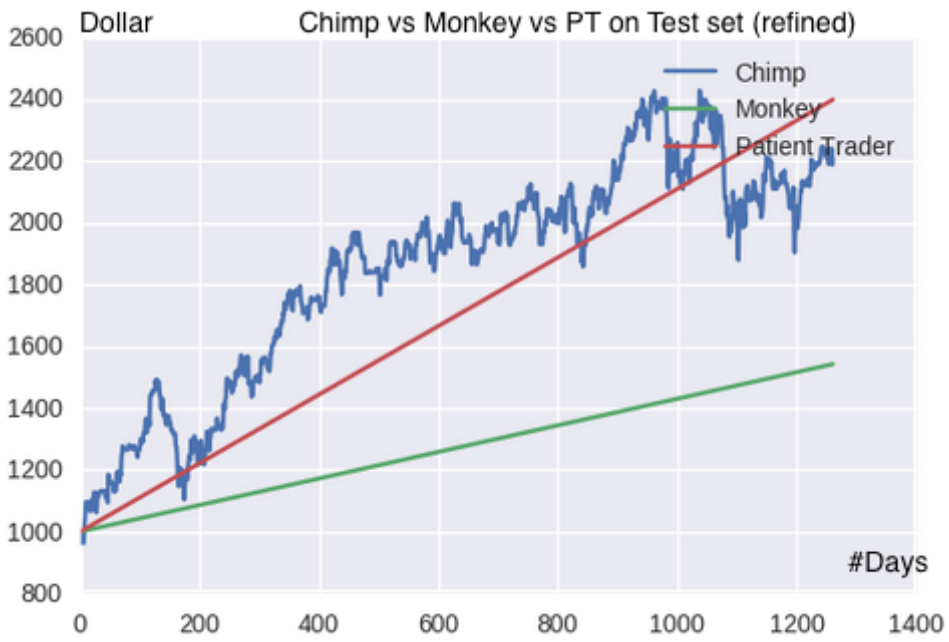
$$\implies r = 0.1489854 \approx 14.90\%$$

It is certainly not as impressive as what she did on the cross-validation dataset which means we have probably overfitted the cv dataset with the grid search. Although it is above about 82% of the Monkeys (with median equal to 9.06%), it can still be pure chance and plus, with all the effort, it still can't beat the PT (19.13%). Let's try to refine it a bit.

Refinement

Surely we wouldn't expect the Chimp to give us a 33% of ROI on the test dataset when there is 2008 financial crisis, which though not impossible, is quite unlikely. However there surely is room for improvement. One way to refine it is to do it on the reinforcement learning part. It could be that the God Chimp is doing her job way too well. We can think of the God Chimp as someone who can see the future. And if you think about it: 1. we have someone who can see the future, 2. there's no trading cost. This means this "someone" will make use of every possible ups and downs to make all the money he or she wishes to. And this is bad, because he or she can make money out of the noise too, and we can't model the noise. If most of the time our model is trying to fit the noise, then it's not very good.

So to fix this, we can introduce trading cost on the training stage so if it were just small ups and downs, the Chimp would refrain herself from being too greedy or she would be penalized by the trading cost.



Seems like our refinement has beaten 89.7 percent of the Monkeys. And what is more is that it's in the lead from the beginning and really put a good distance between itself and Monkey and PT.

Let's look at the equivalent annual ROI:

$$ROI = (1 + r)^{1260/252} = 2.2068$$

$$\implies r = 0.1715278 \approx 17.15\%$$

Results

Model Evaluation and Validation

The number of iterations required for the Q-table to converge is tightly related to the number of valid actions and the discount factor. We set our discount factor $\gamma = 0.75$, and with two possible valid actions, any rewards that happen 10 days from the day of interest will be discounted to less than 5%. This means exploring all the possible actions would require roughly 2^{10} of iterations (given that we have only two valid actions). Therefore, 5000 iterations as what we have done earlier should guarantee a good convergence.

Justification

The result shows that our Chimp performs better than 89.7% of the Monkeys, which has not yet achieved statistical significance but seems to be a good start. For our trading bot to be useful, we need to test it against more stocks and make it outperform the PT in general. Due to that lack of resources, we won't cover that is this report.

Conclusion

Reflection

This is a really interesting and challenging project for me for a few reasons. First is that I wasn't really familiar with time series data, and second is that I've never bought any stocks and had zero knowledge of the stock market whatsoever.

I started out using only supervised learning, and my first submit (which was like three months ago) wasn't very successful. I was then advised by my reviewer to try out some time series specific techiques such as ARIMA, which I did and failed, and reinforcement learning, which is what I'm using now. And though it hasn't achieved the level for practical use, I feel that this is the right way to go. About one month I after I started going with this approach, I was actually contacted on LinkedIn by the CEO of a Fintech sponsored hedge fund. And I realized they were actually doing something really similar, only that they were doing it with deep learning. This makes me feel more confident about this approach. I really want to thank my first reviewer for leading me to this path!

Following are some of the words of my reflection on my first submit:

There were quite a few places that I found puzzled and difficult.

First is the benchmark. I had no idea what a reasonable benchmark should be. How precise the prediction need to be? How far into the future do I want to predict? And can I predict that with the data?

The second is working with time series data. It's my first time working with time series data. The "physics" behind the scene is dynamic and time-dependent. How do I convert this data into something I can work with traditional machine learning methods, without losing information and constrains?

Third comes to model building. Should we build one model to predict for all the stocks? Or should we build one for each? If I build each of them independently, how do I incorporate sector information?

There are a lot more than what I've listed here, but these are the ones that came to mind first and were definitely among the hardest. However, though the word difficulty is used here, never for a second did I think any of them was difficult. Now that I think back, I think it's because I didn't know what would a reasonable goal be so there was not much pressure. The only thing that kept driving me forward was how to improve it, how to make it better. All of these questions are not so easy to answer, but my ignorance definitely lent me a hand this time.

It's intersting to look back on what I felt three months ago. A lot of things was really fresh an new to me back then but now it all feels quite natural to me. I really like this project. It got me into trading, and has made me more interested in reinforcement learning.

Improvement

1. **Try out a different feature set—**we basically hand-picked our features in the hope that they could capture the patterns of the stock market. Our selection was based on volume price analysis but it may very well not be a good representation of what good VPA pracitioners are really doing. So one way to do this is to include more raw features (say we use tick data to the minute, and trace back more days) and then use PCA or an autoencoder to compress all that information. We can then use RFE or other feature selection methods to pick the better ones.
2. **Feature discretization my not ideal—**since the feature space is vast, one way or the other we are only going to cover a fraction of the space. As such, discretizing it really won't give us any benefits. In doing so we're actually losing a lot of information.
3. **Use other data—**we may also include economic data and social data so the trading bot can capture more than just what VPA method can capture.
4. **Try out different parameters on reinforcement learning—**the best trading strategy is that we make every order placement counts. To achieve this we can increase the discount factor and the trading cost.
5. **Try out different algorithms on supervised learning—**yes, I'm talking about deep learning!