A Model-Free Reinforcement Learning Trading Agent

Udacity Machine Learning Nano Degree Capstone Project Proposal

Sebastian Janisch

November 29, 2016

Summary

Quantitative trading strategies make up a large part of financial investing. Over decades trading models have been developed that seek to find market inefficiencies by evaluating an ever growing amount of financial data.

Machine learning techniques offer a new suite of tools to tackle the challenge of generating superior investment returns. This project proposal aims to outline the usage of a **model-free** reinforcement learning technique called *Q-Learning* to build a self learning trading agent based on quantiative input data.

1 Domain Background

In quantitative asset management any available information that can be quantified is used to uncover potential market opportunities. These opportunities are typically caused by mispricing, where a stock's price is not reflective of its true (intrinsic) value. The *Efficient Market Hypothesis* (henceforth referred to as *EMH*) suggests that a "market in which prices always 'fully' reflect available information is called 'efficient'"[1] indicating that in such efficient markets opportunities due to mispricing don't exist. Empirical analysis suggests however that this theory can, at least in part, be refuted[2], opening the playing field for a broad array of quantitative approaches to find said market inefficiencies.

Theories such as the *EMH* and the *Random Walk Hypothesis*[3] call into question the right of existence for active portfolio management (the discipline that seeks to outperform against a chosen benchmark by uncovering market inefficiencies). Whilst the jury on the matter is still out there is an understanding that said market inefficiencies are difficult to expose[4]. Finding yet another clever way of generating alpha (i.e. return greater than that of the market benchmark) is the problem that quantitative active portfolio management tries to solve.

2 Problem Statement

The assumption of all active asset management is that markets are not always efficient and can be outperformed (i.e. risk adjusted outperformance for given benchmark). Much of traditonal quantitative asset management uses a model based approach, where investment ideas are cast into a model which generates trading signals[4]. This implies that a concrete statement is made as to whether or not a given piece of quantitative information is positive or negative (e.g. high corporate leverage indicates poor financial health and possibility of financial distress). The interpreation of a given quantitive metric is however not always so clear (e.g. some industries are characterised by higher leverage without posing a risk of financial distress). The system proposed in this project sets forward an alternative model-free reinforcement learning technique approach using Q-Learning, where no statement is made as to the interpretation of a quantitative metric but it is rather left to the system to learn it autonomously.

3 Datasets and Inputs

Fortunately in the finance domain quantiative data is readily available and abundant. It is understood that more complete, timely and extensive data can be obtained from vendors at cost, however, for the purpose of this project data readily available and free data sources shall suffice.

One of the most crucial information about a traded stock is surely its stock price. As pointed out by [2] the *EMH* can be strongly refuted for daily stock prices, evidence for longer frequencies is thinner though. It is hence that for this project daily asset returns will be used as a primary data input.

Based on the daily (adjusted, i.e. corrected for stock splits and dividend payments) stock price alone alongside with daily trading volumes a whole field has emerged engaging in the discipline referred to as Technical Analysis [5]. Using metrics that are essentially a function of stock price together with trading volume a whole battery of different metrics is created that is thought to help find market inefficiencies (e.g. Momentum, Bollinger Bands, etc.).

There is much controversy as to whether or not plain technical analysis is truly a strong predictor of future stock prices. Enriching the data set with information other than the stock price alone can add more insight into the details of a company behind a stock and as such yield a more predictive set of information. Fundamental quantitative data derived from financial statements (e.g. profit margin, dividend payout ratio, etc.) is hence used as a second set of input data into the learning algorithm.

The data source used will be the Quandl API to retrieve stock prices and financial statement data for given stocks. To access the necessary data feeds the below Quandl databases will be used.

- End of Day US Stock Prices (Quandl Code *EOD*) - to retrieve stock prices for the US

equity market - historical data up to 1996.

- Core US Fundamentals Data (Quandl Code *SF1*) - to retrieve point in time fundamental metrics for the US equity market - historical data up to 2005.

Given that the earliest available date of both data inputs is the year 2005 this gives 11 years of data to base the system on.

Data Retrieval Example

Assuming Alphabet Inc. was a US equity stock covered in our investment universe and we wanted to obtain daily stock prices and also the profit margin (expressed as the ratio of net income and gross revenue) of the firm to include in our model, then the Quandl database would be utilised for the below dataset codes.

- *EOD/GOOGL* - prices, dividends, splits for Alphabet Inc. (GOOGL).
- *SF1/GOOGL_NETINC_MRQ* - net income, the portion of profit or loss for the period for Alphabet Inc. (GOOGL).
- *SF1/GOOGL_REVENUE_ARQ* - revenues, amount of revenue recognized from goods sold, services rendered, insurance premiums, or other activities that constitute an earning process.

4 Solution Statement

Given the above data source appropriate quantiative metrics will be selected to form the state space for the Q-Learning algorithm[8]. Generally, Q-Learning is a model-free reinforcement learning algorithm which allows to realise delayed rewards and find optimal policies for selecting an action out of a universe of possible actions (e.g. buy vs. sell). This notion of a 'delayed reward' is much in contrast to traditional quantitative methods of investing where models are based on an ex-ante prediction of future asset prices.

More formally, the Q-Learning algorithm is based on a finite Markov Decision Process and maximises total reward. A reward is achieved by transitioning from a given state S_t to S_{t+1} by taking an action a_t. By specifying an appropriate reward function that rewards desired and penalises undesired behaviour the algorithm will learn a policy of behaviour. The speed of learning can be calibrated using an α parameter.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right) \tag{1}$$

State Space

The state space will be formulated based on the below two premises.

1. provide sufficiently large value range covered by the state space to discriminate well over different states of the system.

2. do not overly bloat up the state space to prevent causing high dimensionality issues (refferred to as the *Curse of Dimensionality* [6]).

As a universe of stocks there will be a sample of US equities chosen across different industries.

Intuitively, financial statement ratios for different classes can be chosen. Popular classes are performance ratios (e.g. profit margin), activity ratios (e.g. asset turnover), financing ratios (e.g. leverage) and liquidity (e.g. interest coverage). Given that these metrics are all percent based they are easily discretized to form a state space.

On the side of pure stock price related metrics signals such as momentum, market beta, bollinger band values etc. will be chosen and discretized.

The generated state space will then be explored by randomly iterating over historical examples of stocks for which the above metrics are computed (approach of the Dyna-Q-Learning algorithm).

Risk Adjusted Returns and the Q Reward Function

The reward function will be based on the **risk adjusted** return generated for a given decision the agent took. In active portfolio construction there are many techniques being employed. The premise is mostly to maximise the expected return of a set of assets under a set of constraints. These constraints are typically (but not restricted to) risk constraints and transaction cost constraints, which can be expressed with below figure. [4].

$$\alpha_P - \lambda_A \cdot \Psi_P^2 - TC \tag{2}$$

Here, α represents the return of a portfolio of stocks in excess of its benchmark, Ψ represents some representation of active risk (i.e. risk in excess of the benchmark), λ represents the penalty scalar for taking on active risk and TC represents the transaction costs incurred by the strategy.

Taking into consideration the transaction costs is understood to be a fundamental corner stone of any trading strategy (especially important when investing in low liquidity assets). Since this project however does not make any claim on presenting a realistic real world implementation of a trading strategy but merely aims to outline a more novel way of utilising machine learning techniques transaction costs will be disregarded.

The point to make is that it is easy to achieve high returns by taking excessive active risk. Devising a reward function that is based on pure return would incentivise the Q-Learning algorithm to take risky bets, a behaviour we want to control. By correcting for risk in the reward function we can achieve a similar effect compared to the one outlined in above maximisation problem.

A suitable candidate for a reward function is the *Sharpe Ratio*, the industry standard for calculating risk adjusted return. More formally, the *Sharpe Ratio* is outlined below.

$$S_i = \frac{r_i - r_f}{\sigma_i} \tag{3}$$

Here r_i is the return of asset i, r_f is the risk free rate and σ_i is the standard deviation of the asset's return.

5 Benchmark Model

To evaluate the performance the actions carried out by the agent will be benchmarked against the return of a representative stock index. Given that the learner for the purpose of this project will be based on US stocks the S&P 500 index will be used as a benchmark.

6 Evaluation Metrics

To ensure the outcome of the learner is compared apples to apples a risk adjusted measure will be used to quantify the risk adjusted return of both benchmark index and our stock portfolio. The *Sharpe Ratio* (see above) will be used as the first metric. Additionally, a metric commonly used in active investment is the *Information Ratio* which measures how much **active** return is generated per unit of **active** risk (both compared to the benchmark), compared to the *Sharpe Ratio* which compares against the risk free rate and pure standard deviation.

$$IR = \frac{R_P - R_B}{\sigma_{R_P - R_B}} \tag{4}$$

Using both evaluation metrics will give a good handle on overall risk adjusted return and also excess return with respect to excess risk taken on.

As mentioned above, it is understood that a more realistic system would incorporate transaction costs into the evaluation. For simplicity however this project disregards transaction costs.

7 Project Design

For the implementation of this project the below Python libraries will be used:

- Numpy (for numerical operations)
- Pandas (for tabulating and manipulating data)
- Quandl (to access the Quandl financial database)
- Seaborn and Matplotlib (for charting)

An abstract Q-learning implementation will be implemented agnostic of the financial domain we operate in (i.e. implement state space, learning function, exploration of states given input data, etc.). This learning implementation will then be used to initially train the agent on historical data without executing actions or simulating any portfolio. Following this learning phase the agent will be executed against historical data, progressing through time. Based on input data the agent will at a given point in time take one of the below actions for a given stock:

- BUY
- SELL
- DO NOTHING (i.e. hold or don't buy)

Based on these actions a portfolio will be formed and its performance recorded through time. For the time frame of the execution performance measures will be formulated (see sharpe ratio above) and visualised. The important outcomes to determine are:

- the **risk adjusted** performance of the portfolio against the benchmark (both *Sharpe Ratio* and *Information Ratio*)
- the performance through time (does it improve as the agent learns)

For simplicity the agent itself will be run as a basic command line python program which will generate csv based outputs (actions take at given time, etc). To then measure performance and visualise results an iPython Notebook seems to be the most suitable tool. It allows to read the produced csv outputs using Python, parse them and produce performance outputs.

References

[1] Eugene F. Fama, Efficient Capital Markets: A Review of Theory and Empirical Work, (1970).

[2] Martin Sewell, The Efficient Market Hypothesis: Empirical Evidence, International Journal of Statistics and Probability, Vol. 1, No. 2, (2012).

[3] Paul H. Cootner, The random character of stock market prices, M.I.T. Press, (1964).

[4] Richard C. Grinold and Ronald N. Kahn, Active Portfolio Management : A quantative approach for producing superior returns and selecting superior money managers, (1999).

[5] John J. Murphy, Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications, New York Institute of Finance, (1998).

[6] Raul Rojas, The Curse of Dimensionality, (2015).

[7] Keith C. Brown and Frank K. Reilly, Analysis of Investments and Management of Portfolios

[8] Richard Sutton, Reinforcement Learning: An Introduction, (1998).