Financial Stock Recommendation System

A Project Report Submitted in Partial Fulfillment of the Requirements for Award of
the Degree of Bachelor of Technology in Information and Communication Technology

Submitted by
Neel Gandhi
Roll No. 18BIT078
Yash Sahitya
Roll No. 18BIT121
Shobhit Sinha
Roll No. 18BIT103
Arihant Kamdar
Roll No. 18BIT004

Under the Supervision and Guidance of
Dr. Jigarkumar Shah
Assistant Professor
Pandit Deendayal Energy University

Submitted to
Department of Information and Communication Technology
School of Technology
Pandit Deendayal Energy University (PDEU)
Gandhinagar, INDIA, 382007

Contents

Chapter 1

Introduction

1.1 Problem statement

Improvising the existing traditional stock recommendation system using portfolio optimization and ad hoc machine learning techniques while also validating the proposed results through backtesting.

1.2 Motivation

Recent advances in the Stock Recommendation Systems, trading technologies by using various Machine Learning and AI techniques to generate a reliable and accurate Stock Portfolio Recommendation system with minimum risk has been the main motivation behind selecting this particular topic. This project has encouraged our team to learn and understand financial terminologies, investment analysis and portfolio management, modern portfolio theory (MPT) while also requiring us to have in-depth domain knowledge and learn about the current AI techniques applied in the domain.

1.3 Objective

To build a 'Stock Portfolio Recommendation' web application and backtest the generated results to find out absolute returns of recommended stocks, Compound Annual Growth Rate (CAGR) and Portfolio Growth.

1.4 Roadmap

- To gain detailed domain knowledge required for better grasp and understanding for the project.
- Reviewing, analysing and inspecting the latest research work in the field to understand how to use and implement AI techniques in the project.
- Coding in Jupyter Notebook with reference to our proposed idea to conduct quantitative analysis and to test the AI algorithms for the same.
- Writing clean and reproducible code for production and deployment of the web application.
- To backtest the recommended portfolio i.e the generated results to verify and validate the proposed results.
- To study and examine if any further improvements can be made regarding the performance of the model by making changes in the required part of the pipeline to produce a more accurate and reliable 'Stock Portfolio Recommendation System'.

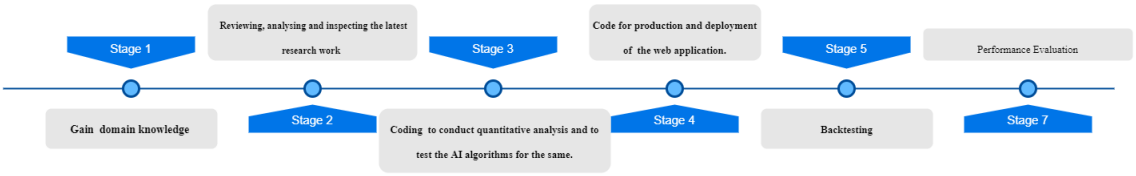


Figure 1.1: Roadmap

Chapter 2

Weekly progress from 20/08/21 to 27/08/21

2.1 Literature review

The field of stock trading investment has been pursued for a long time with many research papers and works for the same by using various kinds of models. This project only presents some relative researches.Huang [4] developed a model for stock selection by using SVR and genetic algorithms (GAs). This model applied SVR to predict future return of each stock whereas GA was employed to optimize model parameters and input features. Back then, the top-ranked stocks were equally weighted to build a portfolio. Empirical results showed that the investment performance of the proposed model performed better than the benchmarks. Krauss et al.[5] in 2017 analyzed the effectiveness of DNN, gradient-boosted trees, Random Forest, and several ensembles of these methods in the context of statistical arbitrage. Each model was trained on lagged returns of all stocks of the SP 500 after elimination of survivor bias. It was concluded from the experiments that the simple equal-weighted ensemble method could generate satisfying results. Lee and Yoo[6] compared three kinds of recurrent neural network for stock return prediction, i.e., recurrent neural network, gated recurrent unit and long short-term memory (LSTM) neural network. The experimental results presented showed that the LSTM neural network performed the best of these three models. Also, they built predictive threshold-based portfolios based on the predictive results of LSTM neural networks. This model was more data-driven than existing models in designing the portfolios.

Experimental results showed that this portfolio earned a promising return. Fischer and Krauss [3] deployed LSTM neural networks for predicting directional movements of the constituent stocks of the SP 500 from 1992 until 2015. They found that LSTM neural networks based portfolios surpassed the memory-free classifica-

tion based portfolio models, i.eRandom Forest, Deep Neural Network and Logistic regression.The common shortcoming of the models is that these models only apply simple methods to build their portfolio, such as equal weighted and threshold-based methods. These portfolio construction methods do not analyze the risk of each stock, which unbalances the portfolio's expected return and risk. Lin, Huang, Gen, and Tzeng[7] proposed a dynamic portfolio optimization model. This model used the Elman neural network to predict future stock return, and then applied a covariance matrix to measure the risk involved. Experimental results presented that this model outperformed vector autoregression model in dynamic portfolio selection models.Alizadeh, Rada, Jolai, and Fotoohi [1] developed a portfolio optimization model by using adaptive neuro-fuzzy inference system for portfolio return prediction and variance index for risk assessment. Experimental results showed that this portfolio optimization model performed better than the mean–variance model, neural network and Sugeno–Yasukawa method. This research informed us that combining artificial intelligence techniques with modern portfolio optimization could generate better performance than each single model for trading investment.Deng and Min [2] applied a linear regression model with ten variables to select stocks from US and global equities, and built a portfolio based on MV model with some practical constraints of risk tolerance, tracking error and turnover. They found that the risk adjusted return of the proposed model of the global equity universe performed better than the domestic equity universe, and the portfolio return increased with the systematic tracking error and risk tolerance. Paiva et al.[8] proposed a unique decision-making model for day trading investments on the stock market, which was developed using a fusion approach of SVM and MV model for portfolio selection. The proposed model was compared with two other models, i.e., SVM+ 1/N and Random+ MV. The experimental evaluation was based on assets from Ibovespa stock market, which showed the proposed model performed the best. Wang et al.[10] developed a portfolio model by using LSTM neural network for stock selection and MV for portfolio optimization. In this model, LSTM neural network first selects k stocks from the total stock set, then the chosen k stocks are used to build the MV portfolio model. They compared LSTM neural network with SVM, RF and ARIMA model in the stock selection process, then used MV for portfolio optimization. Experiments' results showed that their proposed model outperformed the others.Ta et al.[9] built portfolios by using LSTM neural network and three portfolio optimization techniques, i.e., equal weighted method, Monte Carlo simulation and MV model. Also, they applied linear regression and SVM as comparisons in the stock selection process. Experimental results showed that the LSTM neural network had higher predictive accuracy than linear regression and SVM, and its constructed portfolios outperformed the others. These models apply different methods for stock selection, then build portfolio optimization models with selected stocks for trading investment. These methods show us a promising direction to build portfolio models

in practice. However, classic portfolio optimization models are often inappropriate for short term practical investment. Thus, it is important to explore a more efficient approach to combine return predictive results with portfolio optimization models. Hence in our project, we are trying to overcome the shortcomings of traditional portfolio optimization system with help of machine learning techniques including clustering followed by LSTM and Reinforcement Learning methods. Since machine learning and deep learning models have shown overwhelming superiority than time series models.In our project we also analyze technical parameters for generating profit yielding portfolio optimization system.

Chapter 3

Weekly progress from 28/08/21 to 03/09/21

3.1 Dataset Description

Stock market data is widely analyzed for educational, business and personal interests. The National Stock Exchange of India Limited (NSE) is the leading stock exchange of India, located in Mumbai. The NIFTY 50 index is National Stock Exchange of India's benchmark broad based stock market index for the Indian equity market.Apart from NIFTY 50 index, there are also other indices like NIFTY Next 50, Nifty Midcap 150 etc. Exploring these indices may help in taking investment decisions.

3.1.1 Content

The data is the price history and trading volumes of the fifty stocks in the index NIFTY 50 from NSE (National Stock Exchange) India. All datasets are at a day-level with pricing and trading values split across .cvs files for each stock along with a metadata file with some macro-information about the stocks itself. The data spans from 1st January, 2000 to 30th April, 2021. This dataset has day level information on major NIFTY indices starting from 01 January 2000.

Each file represents an index and has the following columns

Since new stock market data is generated and made available every day, in order to have the latest and most useful information, the dataset will be updated once a month.

Date	date of observation
Open	open value of the index on that day
High	highest value of the index on that day
Low	lowest value of the index on that day
Close	closing value of the index on that day
Volume	volume of transaction
Turnover	turn over
P/E	price to earnings ratio
P/B	price to book value
Div Yield	dividend yield

3.2 Acknowledgements

NSE India: https://www.nseindia.com/

Thanks to NSE for providing all the data publicly.

Various machine learning techniques can be applied and explored to stock market data, especially for trading algorithms and learning time series models.

3.3 yfinance : Yahoo Finance Library

yfinance package not just helps to to access the share-price details, it also provides myriad of other financial and non-financial data pertaining to all companies listed in the United States.

yfinance can also be used to obtain data related of bonds, stock indices (SP 500, Nasdaq, Dow 30 etc.), Commodities (Crude Oil , Gold, Silver etc.), Currencies (EUR/USD, GBP/USD, USD/JPY etc.), International Markets (FTSE 100, Nikkei 225 etc.) and even bitcoin and crypto-currency index.

Chapter 4

Weekly progress from 04/09/21 to 11/09/21

4.1 Portfolio Management

4.1.1 Modern Portfolio Theory

Modern portfolio theory (MPT) is a theory on how risk-averse investors can construct portfolios to maximize expected return based on a given level of market risk. Harry Markowitz pioneered this theory in his paper "Portfolio Selection," which was published in the Journal of Finance in 1952.

Modern portfolio theory argues that an investment's risk and return characteristics should not be viewed alone, but should be evaluated by how the investment affects the overall portfolio's risk and return. MPT shows that an investor can construct a portfolio of multiple assets that will maximize returns for a given level of risk. Likewise, given a desired level of expected return, an investor can construct a portfolio with the lowest possible risk. Based on statistical measures such as variance and correlation, an individual investment's performance is less important than how it impacts the entire portfolio. MPT assumes that investors are risk-averse, meaning they prefer a less risky portfolio to a riskier one for a given level of return. As a practical matter, risk aversion implies that most people should invest in multiple asset classes.

The expected return of the portfolio is calculated as a weighted sum of the individual assets' returns. If a portfolio contained four equally weighted assets with expected returns of 4, 6, 10, and 14%, the portfolio's expected return would be: $(4\% \times 25\%) + (6\% \times 25\%) + (10\% \times 25\%) + (14\% \times 25\%) = 8.5\%$

Calculating Portfolio Expected Return

To calculate the expected return of a portfolio, the investor needs to know the expected return of each of the securities in their portfolio as well as the overall weight of each security in the portfolio.

Let's say your portfolio contains three securities. The equation for its expected return is as follows:
Expected Return = $WA \times RA + WB \times RB + WC \times RC$
where:
WA = Weight of security A
RA = Expected return of security A
WB = Weight of security B
RB = Expected return of security B
WC = Weight of security C
RC = Expected return of security C
Python Function for Calculating Portfolio expected Return:

```
def portfolio_return(weights, returns):
    """
    Weights -> Returns
    """
    return weights.T @ returns
```

Calculating Portfolio Volatility

The method to compute the risk of a portfolio is written below and subsequently we explain and give the mathematical formulation for each of the step :

- Calculate the covariance matrix on the returns data
- Annualize the covariance by multiplying by 252
- Compute the portfolio variance by multiplying it with weight vectors
- Compute the square root of the variance calculated above to get the standard deviation. This standard deviation is called as the Volatility of the portfolio

The formula to compute the covariance and annualizing it is :
covariance = returns.covO* 252
since there are 252 trading days, we multiply the covariance by 252 in order to annualize it. The portfolio volatility , to be calculated in step 4 above, depends on the weights of the assets in the portfolio and is defined as :

$\sigma_{portfolio} = \sqrt{(weights)^\top \cdot Cov_{portfolio} \cdot (weights)}$

where, Covportfolio is the covariance matrix of the portfolio.
weights is the vector of weights allocated to each asset in the portfolio.

Python Function for calculating Portfolio Volatility:

```
def portfolio_vol(weights, covmat):
    """
    Weights -> Vol
    """
    return (weights.T @ covmat @ weights)**0.5
```

4.2 Summary

1. ***Expected return***
   $\mathrm{E}(R_p) = \sum_i w_i \mathrm{E}(R_i)$ where R_ρ is the return on the portfolio, is the weighting of component asset $_i$ in the portfolio).R_i is the return on asset i and wi (that is, the proportion of asset "i"

2. ***Portfolio Return Variance***

$$\sigma_p^2 = \sum_i w_i^2 \sigma_i^2 + \sum_i \sum_{j \neq i} w_i w_j \sigma_i \sigma_j \rho_{ij}$$

   where P_{ij}^2 is the correlation coefficient between the returns on assets i and j. Alternatively the expression can be written as:

$$\sigma_p^2 = \sum_i \sum_j w_i w_j \sigma_i \sigma_j \rho_{ij}$$

3. ***Portfolio Return Volatility***
   $\sigma_p = \sqrt{\sigma_p^2}$

Chapter 5

Weekly progress from 12/09/21 to 24/09/21

5.1 Markowitz Portfolio Optimization and Efficient Frontier

The portfolio optimization plays an important role in determining the portfolio strategies for investors. What investors hope to achieve through a portfolio optimization is to maximize portfolio return and minimize portfolio risk. Since the return changes based on risk investors have to balance the contradiction between risk and return for their investment. There is therefore no single opti- mixed portfolio to satisfy all investors. The optimal portfolio is determined by the preferences of the investor's risk and return.

Our goal is to choose the weights for each asset in our portfolio such that we maximize the expected return given a level of risk . Mathematically, the objective function can be defined as shown below:
maximize $E[(weights)^T.returns]$
subject to $(weights)^T.cov_{portfolio}.weights = \sigma^2$
Another way to look at the Efficient Frontier and the objective function is that we can minimize the risk given that the Expected return is at least greater than is given value. Mathematically this objective function can be written as:
minimize $(weights)^T.cov_{portfolio}.weight$
subject to $(weights)^T.\mu \geq \mu_{target}$
$(weights)^T.1 = 1$
$weights_i \geq 0$

the first line represents that the objective is to minimize the portfolio variance , i.e in turn the portfolio volatility and therefore minimizing the risk. The subject

to constraints implies that the returns have to be greater than a particular target return, all the weights should sum to 1 and weights should not be negative

5.2 Efficient Frontier

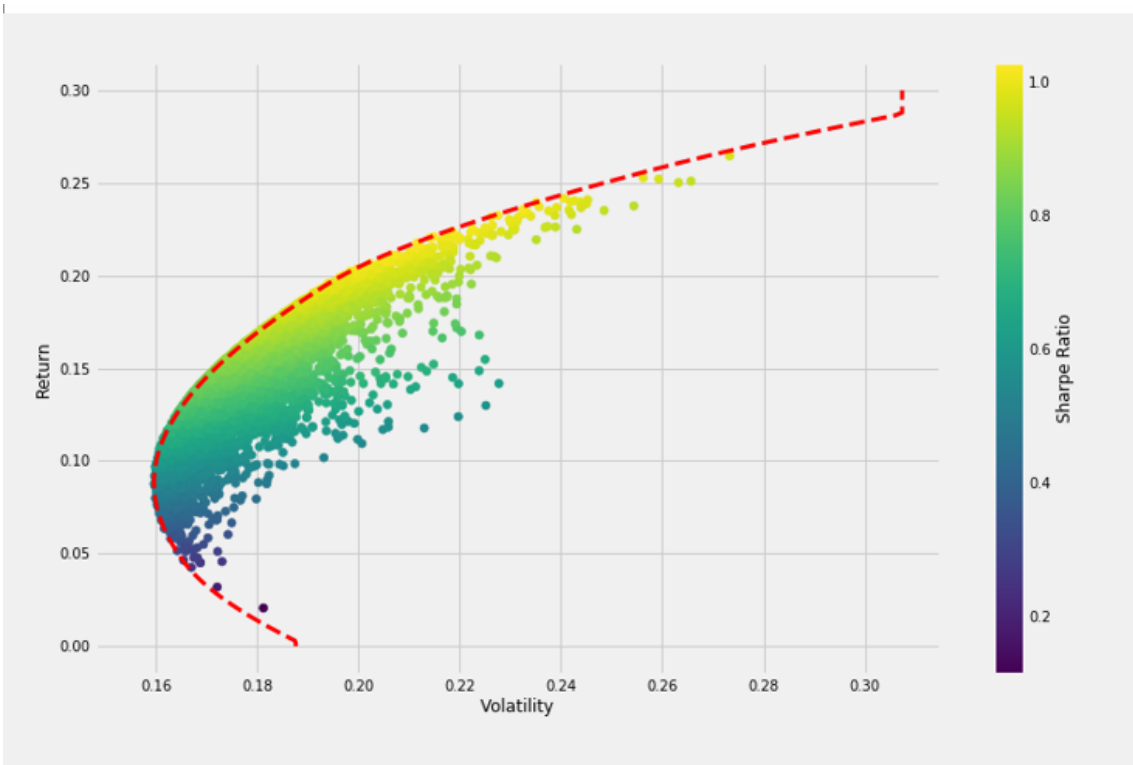


Figure 5.1: Efficient Frontier

The efficient frontier is the set of optimal portfolios that offer the highest expected return for a defined level of risk or the lowest risk for a given level of expected return. Portfolios that lie below the efficient frontier are sub-optimal because they do not provide enough return for the level of risk. Portfolios that cluster to the right of the efficient frontier are sub-optimal because they have a higher level of risk for the defined rate of return.

The efficient frontier rates portfolios (investments) on a scale of return (y-axis) versus risk (x-axis). Compound Annual Growth Rate (CAGR) of an investment is commonly used as the return component while standard deviation (annualized) depicts the risk metric.

The efficient frontier graphically represents portfolios that maximize returns for the risk assumed. Returns are dependent on the investment combinations that

make up the portfolio. The standard deviation of a security is synonymous with risk. The less synchronized the securities (lower covariance), the lower the standard deviation. If this mix of optimizing the return versus risk paradigm is successful then that portfolio should line up along the efficient frontier line.

A key finding of the concept was the benefit of diversification resulting from the curvature of the efficient frontier. The curvature is integral in revealing how diversification improves the portfolio's risk / reward profile. It also reveals that there is a diminishing marginal return to risk. Optimal portfolios that comprise the efficient frontier tend to have a higher degree of diversification than the sub-optimal ones, which are typically less diversified.

Python Function for Efficient Frontier:

```
def plot_ef(n_points, er, cov):
    """
    Plots the multi-asset efficient frontier
    """
    weights = optimal_weights(n_points, er, cov)
    rets = [portfolio_return(w, er) for w in weights]
    vols = [portfolio_vol(w, cov) for w in weights]
    ef = pd.DataFrame({
        "Returns": rets,
        "Volatility": vols
    })
    return ef.plot.line(x="Volatility", y="Returns", style='.-', legend=
```

After obtaining the Efficient Frontier, the next step is to find the optimal weights. We can optimize the using multiple methods as written below:

5.2.1 SHARPE RATIO

The Sharpe-ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk. The formula used to calculate Sharpe-ratio is given below:

$$SharpeRatio = (\mathrm{Rp} - \mathrm{Rf})/\mathrm{SDp}$$

where,
Rp is the return of portfolio
Rf is the risk free rate
SDp is the standard deviation of the portfolio's returns

Python Function for Sharpe-ratio:

```
def sharpe_ratio(r, riskfree_rate, periods_per_year):
    """
    Computes the annualized sharpe ratio of a set of returns
    """
    # convert the annual riskfree rate to per period
    rf_per_period = (1+riskfree_rate)**(1/periods_per_year)-1
    excess_ret = r - rf_per_period
    ann_ex_ret = annualize_rets(excess_ret, periods_per_year)
    ann_vol = annualize_vol(r, periods_per_year)
    return ann_ex_ret/ann_vol
```

5.2.2 Minimum Volatility

One of the most effective smart beta strategies for achieving a reduction of risk are known as 'minimum volatility' strategies, which are specifically designed with this goal in mind, seeking to deliver market-like returns with less volatility by targeting lower volatility stocks

Python Function for Minimum Volatility:

```
def minimize_vol(target_return, er, cov):
    """
    Returns the optimal weights that achieve the target return
    given a set of expected returns and a covariance matrix
    """
    n = er.shape[0]
    init_guess = np.repeat(1/n, n)
    bounds = ((0.0, 1.0),) * n # an N-tuple of 2-tuples!
    # construct the constraints
    weights_sum_to_1 = {'type': 'eq',
                        'fun': lambda weights: np.sum(weights) - 1
    }
    return_is_target = {'type': 'eq',
                        'args': (er,),
                        'fun': lambda weights, er: target_return - portfo
    }
    weights = minimize(portfolio_vol, init_guess,
                       args=(cov,), method='SLSQP',
                       options={'disp': False},
```

```
                        constraints=(weights_sum_to_1, return_is_target),
                        bounds=bounds)
    return weights.x

def optimal_weights(n_points, er, cov):
    """
    Returns a list of weights that represent a grid of n_points on the e
    """
    target_rs = np.linspace(er.min(), er.max(), n_points)
    weights = [minimize_vol(target_return, er, cov) for target_return in
    return weights
```

5.3 Capital Asset Pricing Model (CAPM)

The Capital Asset Pricing Model (CAPM) describes the relationship between systematic risk and expected return for assets, particularly stocks. CAPM is widely used throughout finance for pricing risky securities and generating expected returns for assets given the risk of those assets and cost of capital.

$$\text{ERi} = \text{Rf} + \beta\text{i}(\text{ERm} - \text{Rf})$$

where
ERi=expected return of investment
Rf=risk-free rate
i=beta of the investment
(ERmRf)=market risk premium

The goal of the CAPM formula is to evaluate whether a stock is fairly valued when its risk and the time value of money are compared to its expected return.

5.3.1 The CAPM and the Efficient Frontier

Using the CAPM to build a portfolio is supposed to help an investor manage their risk. If an investor were able to use the CAPM to perfectly optimize a portfolio's return relative to risk, it would exist on a curve called the efficient frontier, as shown on the following graph

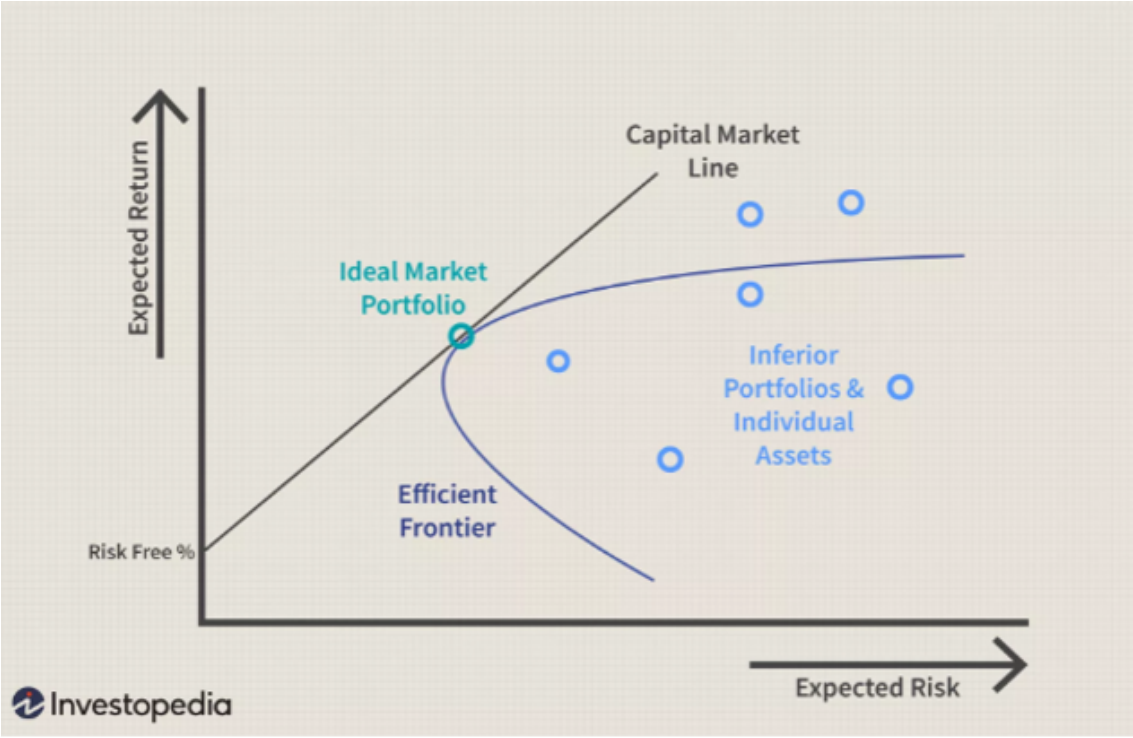


Figure 5.2: Efficient Frontier

The graph shows how greater expected returns (y-axis) require greater expected risk (x-axis). Modern Portfolio Theory suggests that starting with the risk-free rate, the expected return of a portfolio increases as the risk increases.

Chapter 6

Weekly progress from 25/09/21 to 1/10/21

6.0.1 Assumptions of Modern Portfolio Theory

Modern Portfolio theory has a certain assumption that is to be considered while making any decisions in order to arrive at the conclusion that risk, return, and diversification relationships hold true. The different assumptions of the modern portfolio theory are as follows:

- Returns from the assets are distributed normally.
- The investor making the investment is rational and will avoid all the unnecessary risk associated.
- Investors will give their best in order to maximize returns for all the unique situations provided.
- All investors are having access to the same information.
- The cost pertaining to taxes and trading is not considered while making decisions.
- All the investors are having the same views on the rate of return expected.
- The single investors along are not sizeable and capable enough to influence the prices prevailing in the market.
- Unlimited capital at the risk-free rate of return can be borrowed.

6.0.2 Advantages of the Modern Portfolio Theory (MPT)

There are several different advantages of the Modern portfolio theory providing the opportunity for the investors investing their money in the market. Some of the advantages are of the Modern portfolio theory as follows:

1. It helps in evaluating and managing risks and returns associated with the investments. With the help of analysis, the assets which are underperforming assets and the assets having an excessive risk with respect to returns can be scrutinized and then replaced with the new one.

2. The theory is an important tool for avoiding financial ruin because by following these theories, traders don't rely on only one investment for their financial stability; rather, they diversify their portfolio in order to get the maximum return with minimum risk.

6.0.3 Disadvantages of the Modern Portfolio Theory (MPT)

Along with the different advantages, there exist the limitations and drawbacks also of the Modern portfolio theory, which includes the following:

1. In the case of the modern portfolio theory, the past performance of the company under consideration is taken. The performance of the past never provides a guarantee for the result that could arise in the future. Considering only the past performances sometimes leads to overpassing the newer circumstances, which might not be there when historical data were considered but could play an important role in making the decision.

2. This theory assumes that there is a normal distribution of the return on an asset within a class of assets, which is proved to be wrong for individual equities as the correlations of asset class may change over the period of time.

3. In this theory, there is an assumption that securities of any of the sizes can be bought and sold, which doesn't hold true as some of the securities have minimum order sizes, which cannot be dealt with in the fraction.

4. Modern Portfolio Theory even though is accepted widely all over the world and also applied by different investment institution, but at the same time it has also been criticized by different persons particularly by representatives of the behavioral economics who challenges the assumptions of the Modern portfolio theory on the parameters of investor rationality and the expectations for the return.

5. Variance-based – The most serious criticism of modern portfolio theory is that it assesses portfolios based on variance and not the downside risks. As a result, two portfolios with the same level of variance and returns are usually considered equally desirable under the MPT. Frequent small losses could be the reason for the variance for one portfolio whereas the other could be showing similar variance because of two or three extraordinary declines. Most investors are likely to prefer frequent small losses as they are easier to endure. To overcome this shortfall of MPT, investors prefer to use the Post-Modern Portfolio Theory (PMPT), which attempts to improve on modern portfolio theory by minimizing downside risk instead of variance.

6.0.4 Conclusion

The main idea or the purpose of the Modern portfolio theory says that the risk is undertaken and return expected linked directly, which means that in order to achieve the greater rate of expected returns, an investor must have to take a higher level of risk. Also, the theory says that the overall risk of the portfolio having securities can be reduced through the means of diversification. In case two different portfolios are given to the investor having the same level of expected return, then the rational decision would be to choose the portfolio having lower total risk. Modern Portfolio Theory, even though it is accepted widely all over the world and also applied by different investment institutions, but at the same time, it has also been criticized by many people. Due to the above mentioned drawbacks of MPT we are currently learning about machine learning and reinforcement learning techniques to maximize the returns and minimize the risks.

Chapter 7

Weekly progress from 01/10/21 to 09/10/21

7.1 Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

1. **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

2. **Unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

3. **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

4. **Reinforcement machine learning algorithms** is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

7.1.1 Reinforcement learning

Reinforcement Learning deals with designing "Agents" that interacts with an "Environment" and learns by itself how to "solve" the environment by systematic trial and error. An environment could be a game like chess or racing, or it could even be a task like solving a maze or achieving an objective. The agent is the bot that performs the activity. An agent receives "rewards" by interacting with the environment. The agent learns to perform the "actions" required to maximize the reward it receives from the environment. An environment is considered solved if the agent accumulates some predefined reward threshold. This nerd talk is how we teach bots to play superhuman chess or bipedal androids to walk. We would be designing an agent that uses some strategy to interact with a trading environment to maximize the value of the portfolio. Here, the actions would be the decision of the agent on what portfolio to maintain (e.g. 30% stock A,30% stock B,30% stock C,10% Cash split). The agent then receives a positive or negative reward for that action (portfolio allocation). The agent iteratively modifies its strategy till it figures out the best action for a given state of the environment.

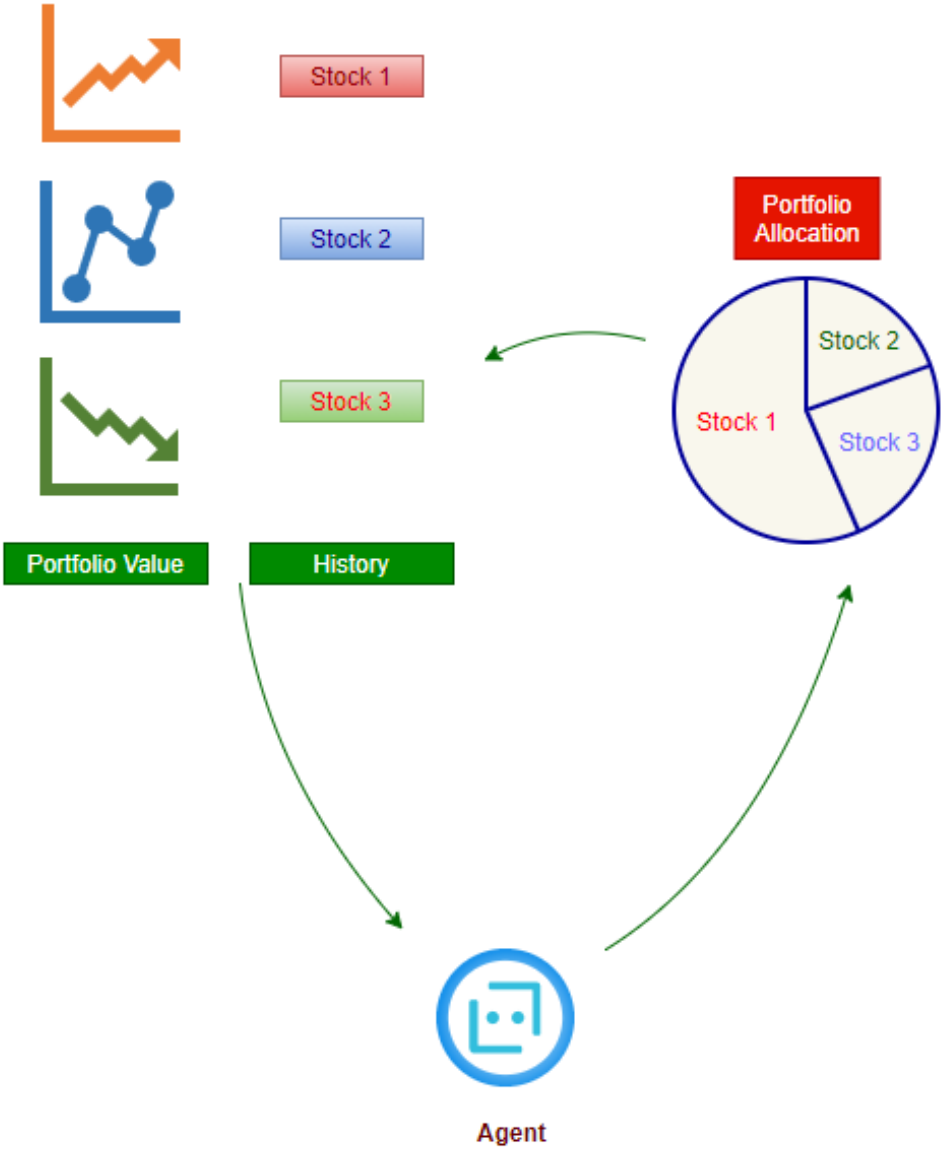


Figure 7.1: Portfolio Optimization using Reinforcement Learning

Chapter 8

Weekly progress from 10/10/21 to 15/10/21

8.1 Reinforcement Learning

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states (sS) of the environment, and choose to take one of many actions (aA) to switch from one state to another. Which state the agent will arrive in is decided by transition probabilities between states (P). Once an action is taken, the environment delivers a reward (rR) as feedback.

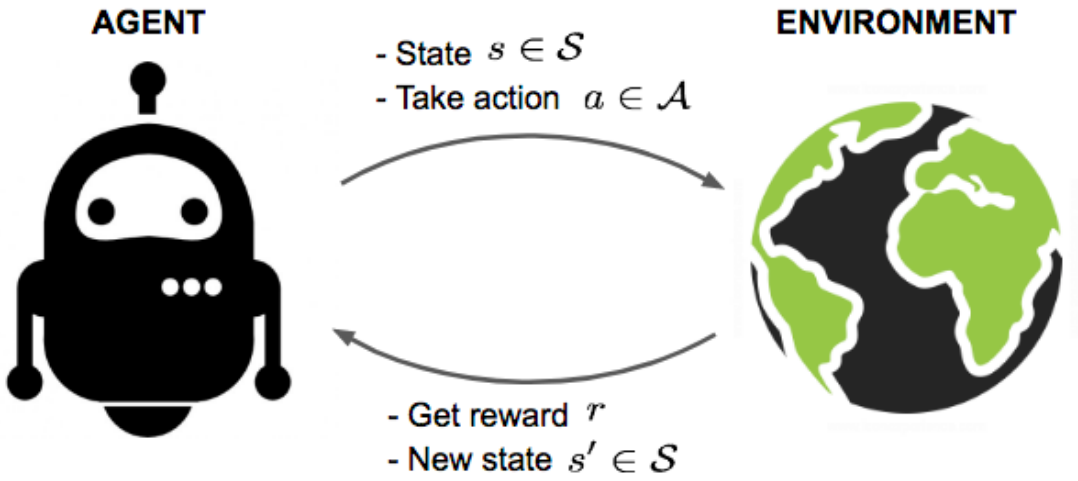


Figure 8.1: Reinforcement Learning

The model defines the reward function and transition probabilities. The agent's policy (s) provides the guideline on what is the optimal action to take in a certain state with the goal to maximize the total rewards. Each state is associated with a value function V(s) predicting the expected amount of future rewards we are able to receive in this state by acting the corresponding policy. In other words, the value function quantifies how good a state is. Both policy and value functions are what we try to learn in reinforcement learning.

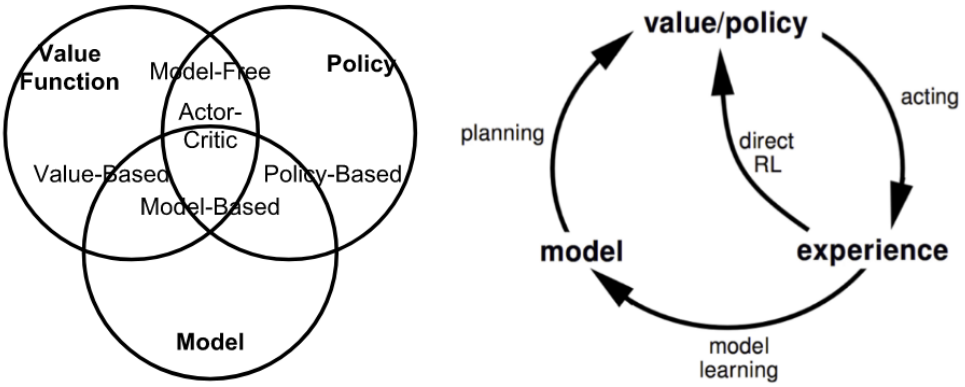


Figure 8.2: Reinforcement Learning

The interaction between the agent and the environment involves a sequence of actions and observed rewards in time, t=1,2,...,T. During the process, the agent accumulates the knowledge about the environment, learns the optimal policy, and makes decisions on which action to take next so as to efficiently learn the best policy. Let's label the state, action, and reward at time step t as St, At, and Rt, respectively. Thus the interaction sequence is fully described by one episode (also known as "trial" or "trajectory") and the sequence ends at the terminal state ST:

$S_1, A_1, R_2, S_2, A_2, \ldots, S_T$

Terms you will encounter a lot when diving into different categories of RL algorithms:

Model-based: Rely on the model of the environment; either the model is known or the algorithm learns it explicitly. Model-free: No dependency on the model during learning. On-policy: Use the deterministic outcomes or samples from the target policy to train the algorithm. Off-policy: Training on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy.

Model: Transition and Reward

The model is a descriptor of the environment. With the model, we can learn or infer how the environment would interact with and provide feedback to the agent. The model has two major parts, transition probability function P and reward function R.

Let's say when we are in state s, we decide to take action a to arrive in the next state s' and obtain reward r. This is known as one transition step, represented by a tuple (s, a, s', r).

The transition function P records the probability of transitioning from state s to s' after taking action a while obtaining reward r. We use P as a symbol of "probability".

P(s', r $|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$

Thus the state-transition function can be defined as a function of P(s,r—s,a):

$P^a_{ss'} = P(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} P(s', r|s, a)$

The reward function R predicts the next reward triggered by one action:

R(s, a) = E $[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} P(s', r|s, a)$

Policy

Policy, as the agent's behavior function , tells us which action to take in state s. It is a mapping from state s to action a and can be either deterministic or stochastic:

Deterministic: $\pi(s) = a. Stochastic : \pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$

Value Function

Value function measures the goodness of a state or how rewarding a state or an action is by a prediction of future reward. The future reward, also known as return, is a total sum of discounted rewards going forward. Let's compute the return Gt starting from time t:

$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

The discounting factor [0,1] penalize the rewards in the future, because:

The future rewards may have higher uncertainty; i.e. stock market. The future rewards do not provide immediate benefits; i.e. As human beings, we might prefer to have fun today rather than 5 years later ;). Discounting provides mathematical convenience; i.e., we don't need to track future steps forever to compute return. We don't need to worry about the infinite loops in the state transition graph. The state-value of a state s is the expected return if we are in this state at time t, St=s:

$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$

Similarly, we define the action-value ("Q-value"; Q as "Quality" I believe?) of a state-action pair as:

$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

Additionally, since we follow the target policy , we can make use of the probility distribution over possible actions and the Q-values to recover the state-value:

$\mathrm{V}_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s,a)\pi(a|s)$

The difference between action-value and state-value is the action advantage function ("A-value"):

$\mathrm{A}_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$

Optimal Value and Policy

The optimal value function produces the maximum return:

$\mathrm{V}_*(s) = \max_\pi V_\pi(s), Q_*(s,a) = \max_\pi Q_\pi(s,a)$

The optimal policy achieves optimal value functions:

$\pi_* = \arg\max_\pi V_\pi(s), \pi_* = \arg\max_\pi Q_\pi(s,a)$

And of course, we have $\mathrm{V}_{\pi_*}(s) = V_*(s) and Q_{\pi_*}(s,a) = Q_*(s,a)$.

8.2 Markov Decision Processes

In more formal terms, almost all the RL problems can be framed as Markov Decision Processes (MDPs). All states in MDP has "Markov" property, referring to the fact that the future only depends on the current state, not the history:

$\mathrm{P}[\ \mathrm{S}_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$

Or in other words, the future and the past are conditionally independent given the present, as the current state encapsulates all the statistics we need to decide the future.

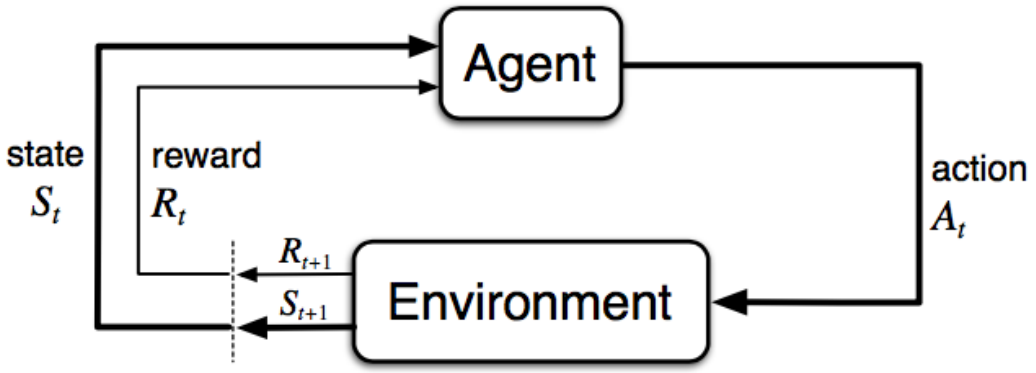


Figure 8.3: Markov Decision Processes

A Markov deicison process consists of five elements M=⟨S,A,P,R,⟩, where the

symbols carry the same meanings as key concepts in the previous section, well aligned with RL problem settings:

S - a set of states; A - a set of actions; P - transition probability function; R - reward function; - discounting factor for future rewards. In an unknown environment, we do not have perfect knowledge about P and R.

8.2.1 Bellman Equations

Bellman equations refer to a set of equations that decompose the value function into the immediate reward plus the discounted future values.

V(s) = E[$G_t|S_t = s$]
= $\mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$
= $\mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots)|S_t = s]$
= $\mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s]$
= $\mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$

Similarly for Q-value, Q(s, a) = E [$R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a$]
= $\mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a]$

Bellman Expectation Equations

The recursive update process can be further decomposed to be equations built on both state-value and action-value functions. As we go further in future action steps, we extend V and Q alternatively by following the policy .

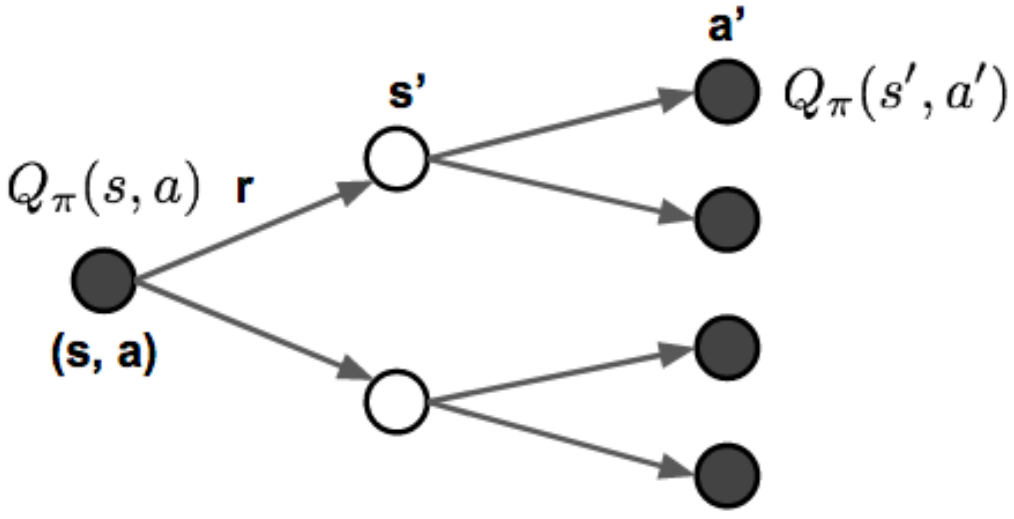


Figure 8.4: Bellman Expectation Equations

$\mathrm{V}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a)$
$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_\pi(s')$
$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_\pi(s'))$
$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a')$

Bellman Optimality Equations

If we are only interested in the optimal values, rather than computing the expectation following a policy, we could jump right into the maximum returns during the alternative updates without using a policy. RECAP: the optimal values V and Q are the best returns we can obtain, $\mathrm{V}_*(s) = \max_{a \in \mathcal{A}} Q_*(s,a)$
$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s')$
$V_*(s) = \max_{a \in \mathcal{A}} (R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s'))$
$Q_*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \max_{a' \in \mathcal{A}} Q_*(s',a')$

Unsurprisingly they look very similar to Bellman expectation equations.

If we have complete information of the environment, this turns into a planning problem, solvable by DP. Unfortunately, in most scenarios, we do not know Pass or R(s,a), so we cannot solve MDPs by directly applying Bellmen equations, but it lays the theoretical foundation for many RL algorithms.

Chapter 9

Weekly progress from 16/10/21 to 22/10/21

9.1 Kalman filter

In statistics and control theory, Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, including statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

The algorithm works by a two-phase process. For the prediction phase, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with greater certainty. The algorithm is recursive. It can operate in real time, using only the present input measurements and the state calculated previously and its uncertainty matrix; no additional past information is required. The Kalman filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate. The estimate is updated using a state transition model and measurements. denotes the estimate of the system's state at time step k before the k-th measurement yk has been taken into account; is the corresponding uncertainty.

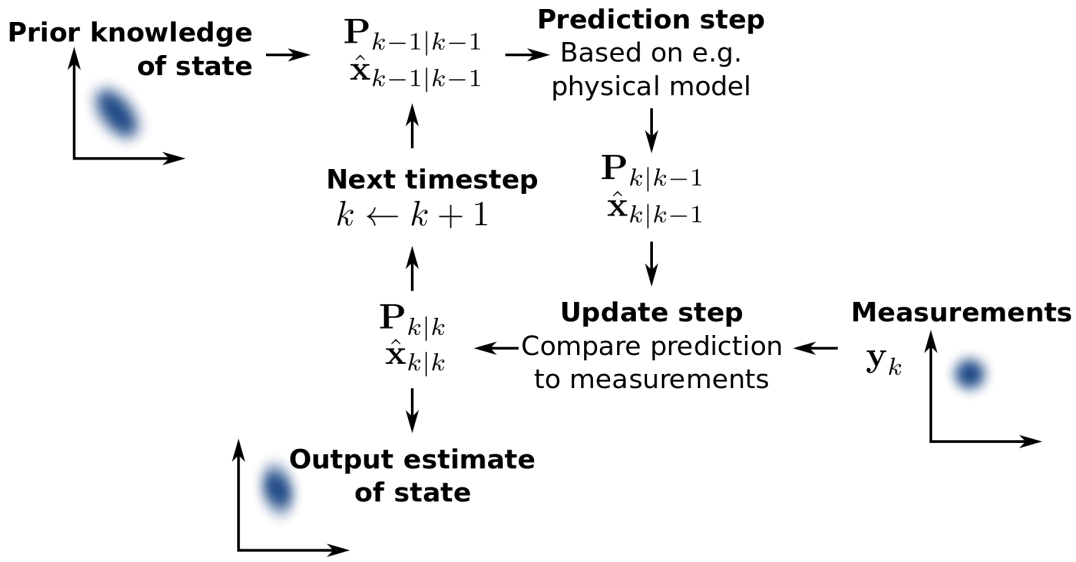
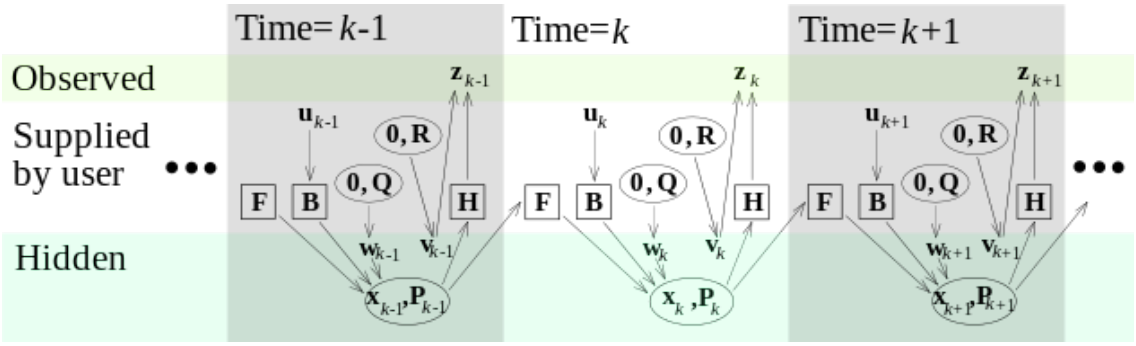


Figure 9.1: Kalman filtering

Kalman filtering is based on linear dynamical systems discretized in the time domain. They are modeled on a Markov chain built on linear operators perturbed by errors that may include Gaussian noise. The state of the target system refers to the ground truth (yet hidden) system configuration of interest, which is represented as a vector of real numbers. At each discrete time increment, a linear operator is applied to the state to generate the new state, with some noise mixed in, and optionally some information from the controls on the system if they are known. Then, another linear operator mixed with more noise generates the measurable outputs (i.e., observation) from the true ("hidden") state. The Kalman filter may be regarded as analogous to the hidden Markov model, with the difference that the hidden state variables have values in a continuous space as opposed to a discrete state space as for the hidden Markov model. There is a strong analogy between the equations of a Kalman Filter and those of the hidden Markov model. A review of this and other models is given in Roweis and Ghahramani (1999),[19] and Hamilton (1994), Chapter 13.[20] In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the following framework. This means specifying the matrices following:

- Fk, the state-transition model;
- Hk, the observation model;
- Qk, the covariance of the process noise;

- Rk, the covariance of the observation noise;
- Bk, the control-input model, for each time-step, k, as described below.



Model underlying the Kalman filter. Squares represent matrices. Ellipses represent multivariate normal distributions (with the mean and covariance matrix enclosed). Unenclosed values are vectors. For the simple case, the various matrices are constant with time, and thus the subscripts are not used, but Kalman filtering allows any of them to change each time step. The Kalman filter model assumes the true state at time k is evolved from the state at (k 1) according to

The Kalman filter model assumes the true state at time k is evolved from the state at (k 1) according to

- Fk is the state transition model which is applied to the previous state xk1;
- Bk is the control-input model which is applied to the control vector uk;
- wk is the process noise, which is assumed to be drawn from a zero mean multivariate normal distribution, , with covariance, Qk:
- At time k an observation (or measurement) zk of the true state xk is made according to
- Hk is the observation model, which maps the true state space into the observed space and
- vk is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance Rk: . The initial state, and the noise vectors at each step x0, w1, ..., wk, v1, ... ,vk are all assumed to be mutually independent.

9.2 Hidden Markov model

The HMM is based on augmenting the Markov chain. A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. The states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

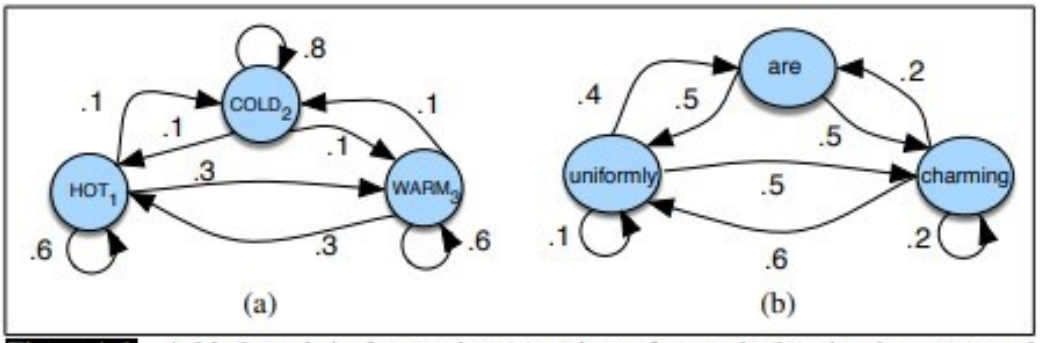


Figure A.1 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Figure 9.2: Markov chain

Markov Assumption: P(qi = a—q1...qi1) = P(qi = a—qi1) (A.1) Figure A.1a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given state must sum to 1. Figure A.1b shows a Markov chain for assigning a probability to a sequence of words w1...wn. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability p(wi —wj)! Given the two models in Fig. A.1, we can assign a probability to any sequence from our vocabulary.

A hidden Markov model (HMM) allows us to talk about both observed events Hidden Markov model (like words that we see in the input) and hidden events (like part-of-speech tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions.

Figure 9.3: Hidden Markov model

First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state: Markov Assumption: P(qi —q1...qi1) = P(qi —qi1) Second, the probability of an output observation oi depends only on the state that produced the observation qi and not on any other states or any other observations: Output Independence: P(oi —q1 ...qi ,...,qT ,o1,...,oi ,...,oT) = P(oi —qi)

Example: Alice believes that the weather operates as a discrete Markov chain. There are two states, "Rainy" and "Sunny", but she cannot observe them directly, that is, they are hidden from her. On each day, there is a certain chance that Bob will perform one of the following activities, depending on the weather: "walk", "shop", or "clean". Since Bob tells Alice about his activities, those are the observations. The entire system is that of a hidden Markov model (HMM). Alice knows the general weather trends in the area, and what Bob likes to do on average. In other words, the parameters of the HMM are known. They can be represented as follows in Python:

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
   'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
   }
```

```
emission_probability = {
    'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
    'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
    }
```

In this piece of code, startprobability represents Alice's belief about which state the HMM is in when Bob first calls her (all she knows is that it tends to be rainy on average). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately 'Rainy': 0.57, 'Sunny': 0.43. The transitionprobability represents the change of the weather in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow will be sunny if today is rainy. The emissionprobability represents how likely Bob is to perform a certain activity on each day. If it is rainy, there is a 50% chance that he is cleaning his apartment; if it is sunny, there is a 60% chance that he is outside for a walk.

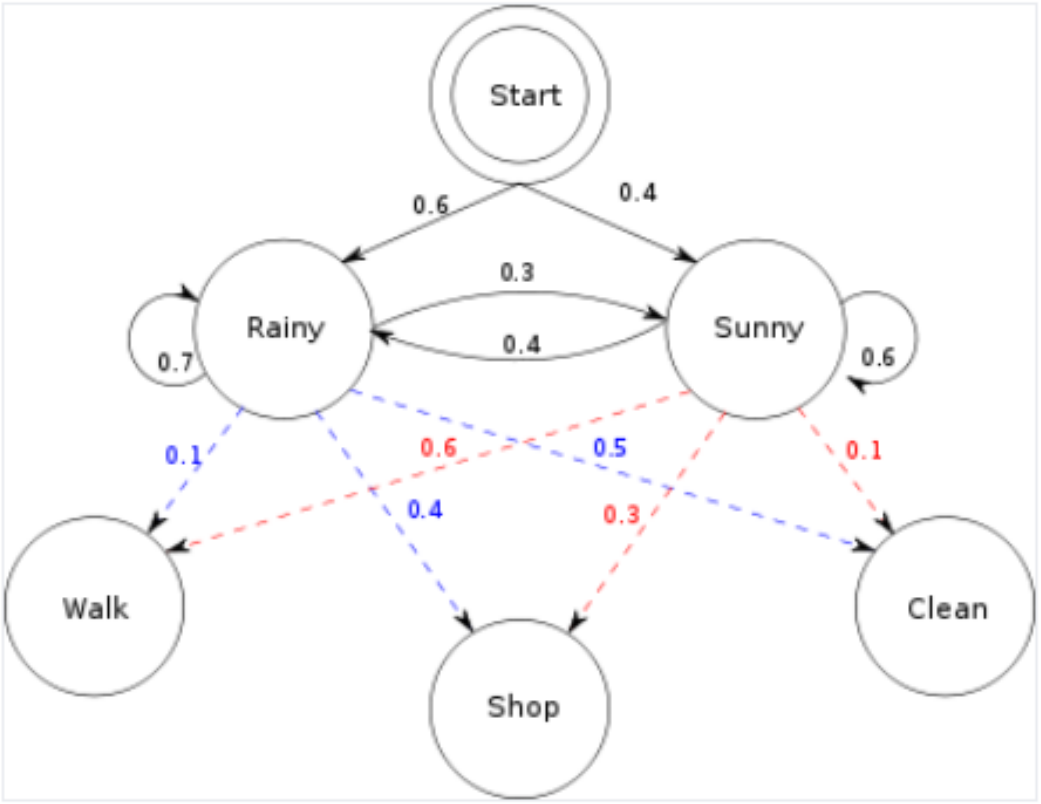


Figure 9.4: Hidden Markov model Example

Bibliography

[1] M. Alizadeh, R. Rada, F. Jolai, and E. Fotoohi. An adaptive neuro-fuzzy system for stock portfolio analysis. *International Journal of Intelligent Systems*, 26(2):99–114, 2011.

[2] S. Deng and X. Min. Applied optimization in global efficient portfolio construction using earning forecasts. *The Journal of Investing*, 22(4):104–114, 2013.

[3] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[4] C.-F. Huang. A hybrid stock selection model using genetic algorithms and support vector regression. *Applied Soft Computing*, 12(2):807–818, 2012.

[5] C. Krauss, X. A. Do, and N. Huck. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the SP 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

[6] S. I. Lee and S. J. Yoo. Threshold-based portfolio: the role of the threshold and its applications. *The Journal of Supercomputing*, 76(10):8040–8057, 2020.

[7] C.-M. Lin, J.-J. Huang, M. Gen, and G.-H. Tzeng. Recurrent neural network for dynamic portfolio selection. *Applied Mathematics and Computation*, 175(2):1139–1146, 2006.

[8] F. D. Paiva, R. T. N. Cardoso, G. P. Hanaoka, and W. M. Duarte. Decision-making for financial trading: A fusion approach of machine learning and portfolio selection. *Expert Systems with Applications*, 115:635–655, 2019.

[9] V.-D. Ta, C.-M. Liu, and D. A. Tadesse. Portfolio optimization-based stock prediction using long-short term memory network in quantitative trading. *Applied Sciences*, 10(2):437, 2020.

[10] W. Wang, W. Li, N. Zhang, and K. Liu. Portfolio formation with preselection using deep learning from long-term financial data. *Expert Systems with Applications*, 143:113042, 2020.