Portfolio Composition Comparison by Different Weighting Method

CONTENTS

01

CONTENTS

Introduction of Different Weighting Method

1. Equal Weighting
2. Markowitz Mean-Variance Optimization Weighting
   (Efficient Frontier, Maximum Sharpe Ratio)
3. Market-Cap Weighting
4. Free-Float Weighting
   (Float Adjusted Market Cap Weighting)
5. Weighting by Π
   (Implied Returns)
6. Weighting by E(R)
   (implied Returns with adjusted views)

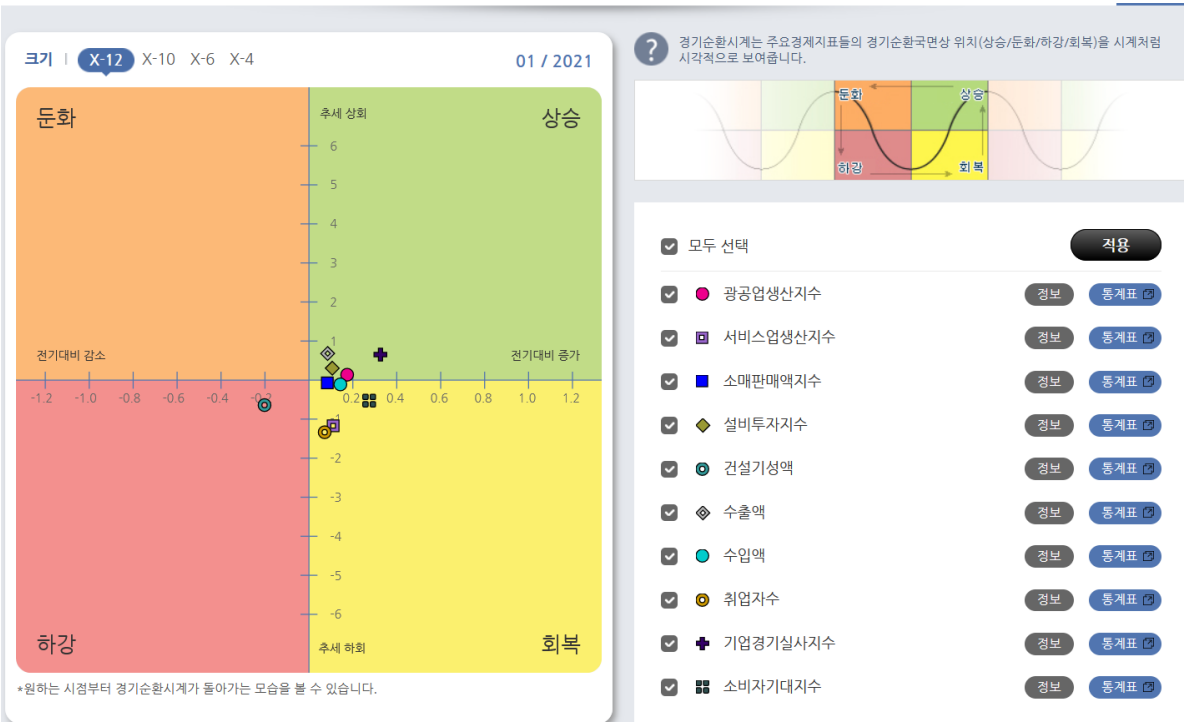
02

CONTENTS

Results

1. Average Return (%)
2. Standard Deviation(%)
3. Sharpe Ratio
4. Minimum Return (%)
5. Maximum Return (%)
6. Alpha (%)
7. Beta

CONTENTS



< 통계청 경기순환시계 그래프, 2021년1월 모습 >

2021년 1Q/ 2Q/ 3Q/ 4Q 국면파악

-> 각 분기별 데이터를 토대로 백테스팅 진행 ->

국면별 유리한 Portfolio Weighting Model 파악

CONTENTS

< 2021년 1Q 분석 >
21년 1월 분석 weights -> 2월 투자 rebalancing -> 성과측정

< 2021년 2Q 분석 >
21년 4월 분석 weights -> 5월 투자 rebalancing -> 성과측정

< 2021년 3Q 분석 >
21년 7월 분석 weights -> 8월 투자 rebalancing -> 성과측정

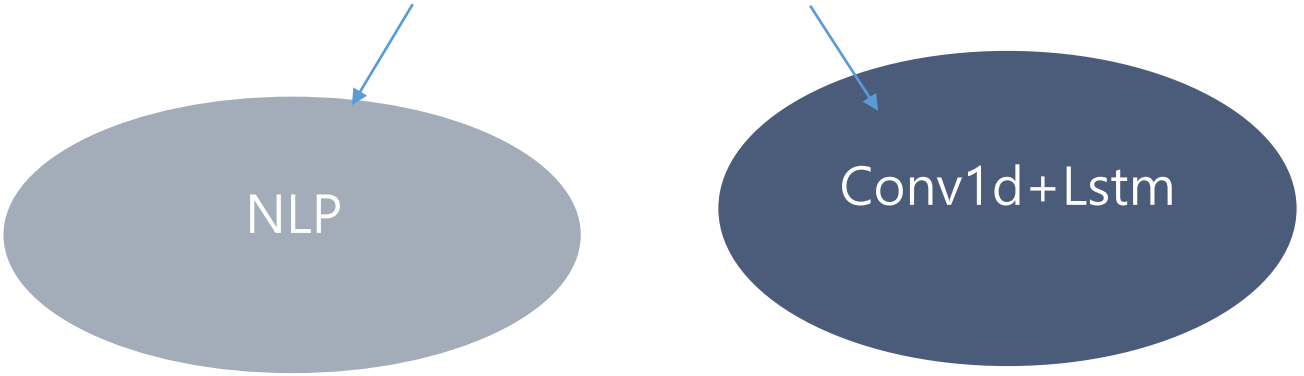
< 2021년 4Q 분석 >
21년 9월 분석 weights -> 10월 투자 rebalancing -> 성과측정

1. Average Return (%)
2. Standard Deviation(%)
3. Sharpe Ratio
4. Minimum Return (%)
5. Maximum Return (%)
6. Alpha (%)
7. Beta

Black Litterman Model

$$E(R) = [(\tau \Sigma)^{-1} + P^T \Omega P]^{-1} [(\tau \Sigma)^{-1} \Pi + P^T \Omega Q]$$

P(시장전망행렬) & Q(시장전망수익률)



조수지, 김흥규, 양철원, "기업 재무분석을 위한 한국어 감성사전 구축"
Korean J Financ Stud. 2021

KIM, Sunwoong. Robo-Advisor algorithm with intelligent view model.
Journal of Intelligence and Information Systems, 2019

S. Jain, R. Gupta and A. A. Moghe, "Stock Price Prediction on Daily Stock Data using Deep Neural Networks," *2018*

L. S. Chong, K. M. Lim and C. P. Lee, "Stock Market Prediction using Ensemble of Deep Neural Networks," *2020*

MIN, Liangyu, et al. A black-litterman portfolio selection model with investor opinions generating from machine learning algorithms. *Engineering Letters, 2021*

Black Litterman Model

Black Litterman Model

Conv1d+Lstm

Feature Selection

Feature Engineering
-Scaling, rolling(window)

Conv1d + Lstm
-pytorch

Comparision
between Close and Prediction

Prediction Sharpe Ratio
-Sort_Value

참고: Stock Market Prediction using Ensemble of Deep Neural Networks

Introduction of Different Weighting Method

Selection of Stock Composition

- **KOSPI200 시가총액 상위 8개 종목 선정**
  -> 삼성전자, SK하이닉스, 삼성바이오로직스, LG화학, 네이버, 현대차, 삼성SDI, 카카오

- **2021. 01. 01 ~ 2021. 12. 31**
  -> 분기별로 나눠서 진행
  -> 향후 블랙리터만 모델의 투자자 전망을 구하기 위함

- Pykrx 라이브러리
->수정주가 종가 및 시가총액 데이터

Introduction of Different Weighting Method

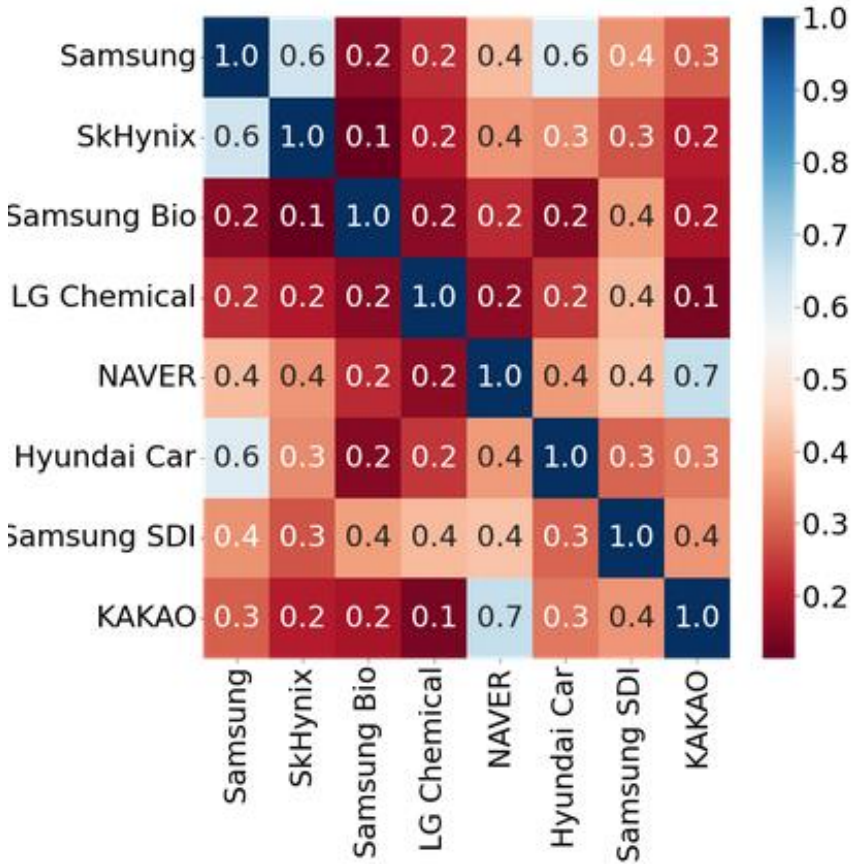
Selection of Stock Composition

<Close>

<Cumsum Of Return>

Introduction of Different Weighting Method

Correlation of Stocks



Introduction of Different Weighting Method

<MVO : Mean Variance Optimization>

Portfolio Return : $R_p = W_A R_A + W_B R_B$

Variance : $S_p^2 = W_A^2 S_A^2 + W_B^2 S_B^2 + 2 W_A W_B Cov(R_A, R_B)$

(참고사항) $Var(X) = (X_i - \bar{X})^2$ 들의 평균 / $Var(Y) = (Y_i - \bar{Y})^2$ 들의 평균

$$Var(aX+bY) = E[(aX+bY)_i - \overline{(aX+bY)}]^2$$

$$= E[(aX+bY)_i - E(aX+bY)]^2$$

$$= E[(aX+bY)_i - E(aX) - E(bY)]^2$$

$$= E[a(X_i - E(X)) + b(Y_i - E(Y))]^2$$

$$= E\left(a^2(X_i-EX)^2 + b^2(Y_i-EY)^2 + 2ab(X_i-EX)(Y_i-EY)\right)$$

$$= E(a^2(X_i-E(X))^2) + E(b^2(Y_i-E(Y))^2) + E(2ab(X_i-E(X))(Y_i-E(Y)))$$

$$= a^2 \underbrace{E((X_i-\bar{X})^2)}_{S_X^2} + b^2 \underbrace{E((Y_i-\bar{Y})^2)}_{S_Y^2} + 2ab \underbrace{E((X_i-\bar{X})(Y_i-\bar{Y}))}_{Cov(X,Y)}$$

$$Var(W_A R_A + W_B R_B) = W_A^2 S_A^2 + W_B^2 S_B^2 + 2 W_A W_B Cov(A, B)$$

포트폴리오 기대수익률 $= W_0 R_0 + W_1 R_1 + \cdots W_n R_n$

$$Var(E(R_p)) = \sum_{i,j}^{n} W_i W_j Cov(R_i, R_j)$$

Introduction of Different Weighting Method

<MVO : Mean Variance Optimization>

$$\sigma_p^2 = \sum_{i,j}^{n} w_i w_j \, Cov(R_i, R_j)$$

$$= w_1 w_1 Cov(R_1, R_1) + w_1 w_2 Cov(R_1, R_2) + \cdots + w_1 w_n Cov(R_1, R_n)$$
$$+ w_2 w_1 Cov(R_2, R_1) + w_2 w_2 Cov(R_2, R_2) + \cdots + w_2 w_n Cov(R_2, w_n)$$
$$\vdots$$
$$+ w_n w_1 Cov(R_n, R_1) + w_n w_2 Cov(R_n, R_1) + \cdots + w_n w_n Cov(R_n, R_n)$$

$$= (w_1, w_2, w_3 \cdots, w_n) \begin{bmatrix} Cov(R_1, R_1), Cov(R_1, R_2) \cdots Cov(R_1, R_n) \\ \vdots \\ Cov(R_n, R_1), Cov(R_n, R_2) \cdots, Cov(R_n, R_n) \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

$$= W^T \cdot \Sigma \cdot W$$

Introduction of Different Weighting Method

<MVO : Mean Variance Optimization>

$$\max_w \left[\frac{\mu_P}{w^T \mu} - \frac{\lambda}{2} \frac{\sigma_P^2}{w^T \Sigma w} \right]$$

subject to $\sum_{k=1}^{N} w_k = 1$

$0 \le w_k \le 1$

Where w : 포트폴리오 종목 비중 $(n \times 1)$

μ : 포트폴리오 종목 과거 수익률 평균 $(n \times 1)$

λ : 위험회피성향 (위험회피계수)

Σ : 공분산 행렬 $(n \times n)$

Differential →

$$\lim_{\lambda \to 0} \left(\mu \cdot w - \frac{\lambda}{2} \Sigma w^2 \right) = 0$$

$$\mu - \lambda \Sigma w = 0$$

$$\mu = \lambda \Sigma w$$

$$w = (\lambda \Sigma)^{-1} \mu$$

Introduction of Different Weighting Method

<MVO : Mean Variance Optimization>

λ (위험회피계수/상수, Risk Aversion)

투자자가 얼마나 위험강도를 회피하는가.

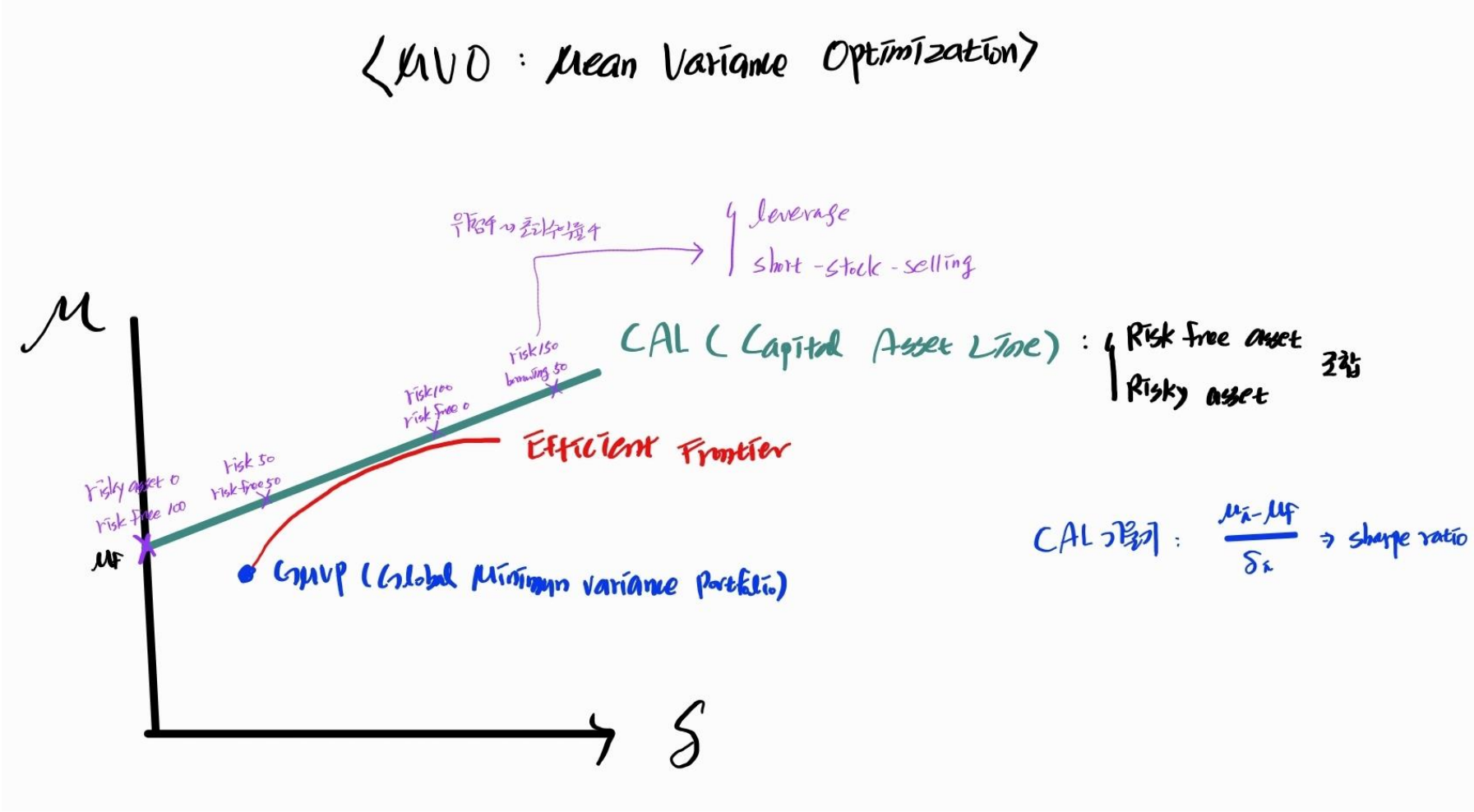
- Risk Aversion ↑ → 무위험자산비중 ↑ → 수익률↓, 변동성↑
- Risk Aversion ↓ → 위험자산비중 ↑ → 수익률↑, 변동성↓

$$\underset{w}{Max}\left[\frac{\mu_p}{w_T\mu} - \frac{\lambda}{2}\frac{\sigma_p^2}{w^T\Sigma w}\right]$$

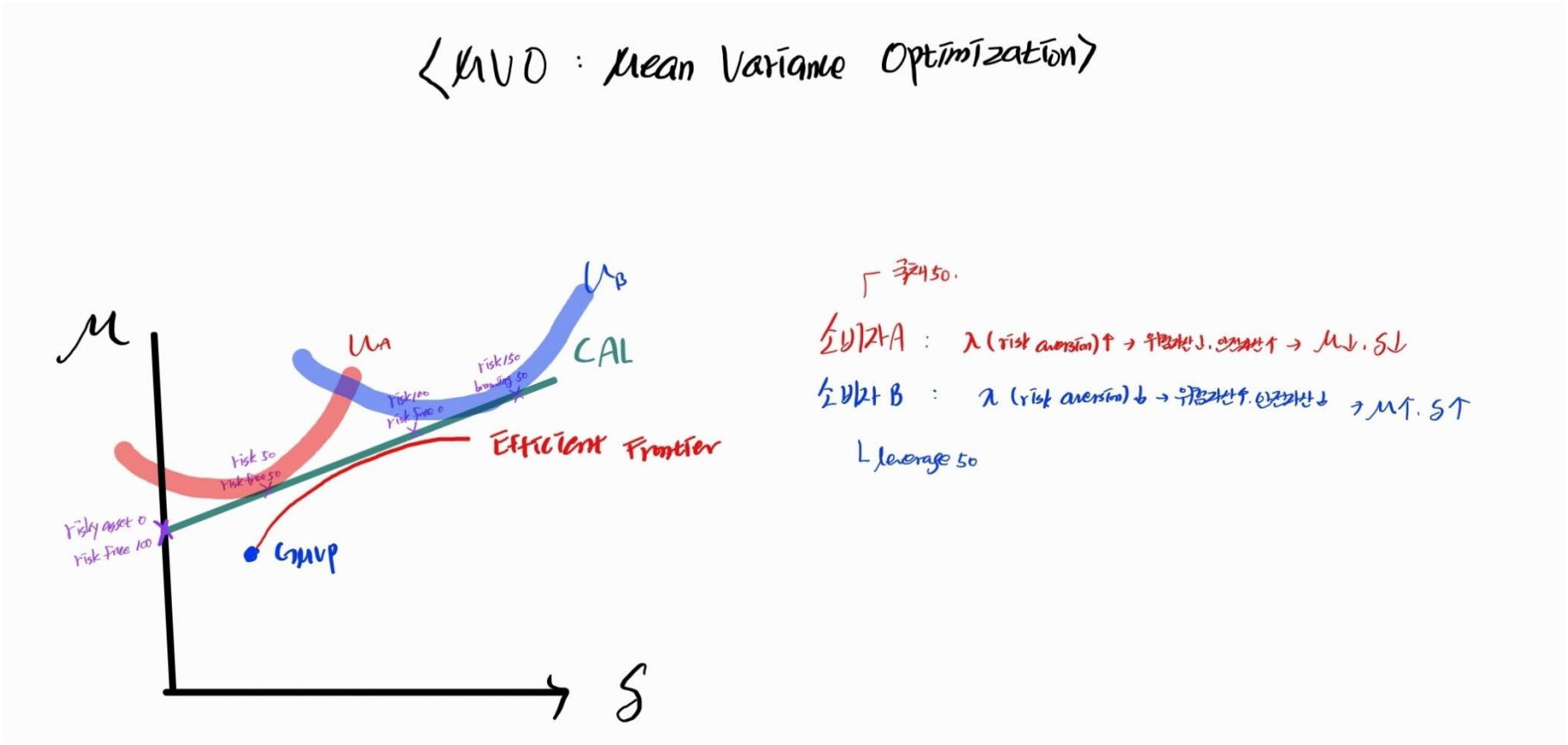
λ↑ → σ_p^2 tolerance↓

λ↓ → σ_p^2 tolearance↑

Introduction of Different Weighting Method



Introduction of Different Weighting Method



Introduction of Different Weighting Method

〈MVO : Mean Variance Optimization〉

* "모든 사람들이 homogeneous 한 선호를 갖는다"는 가정 추가

↓

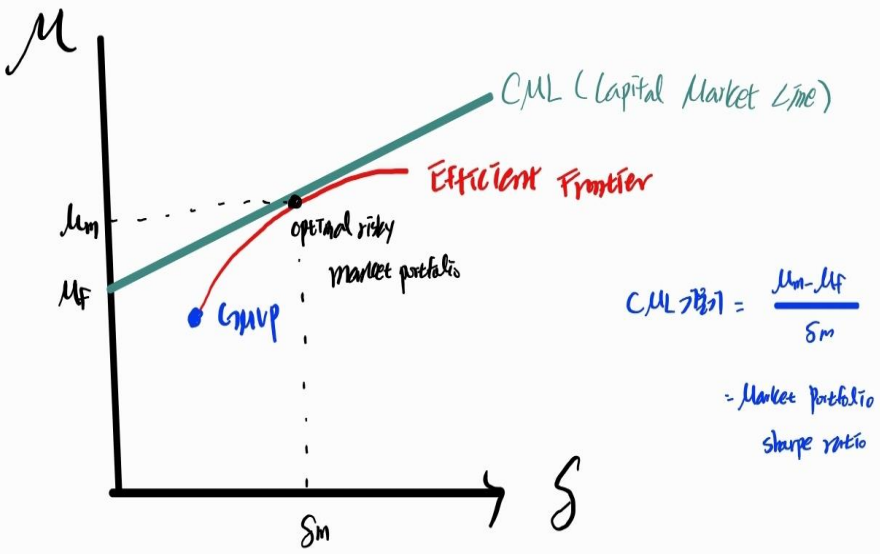
homogeneous expectation (같은 기대수익률)

↓

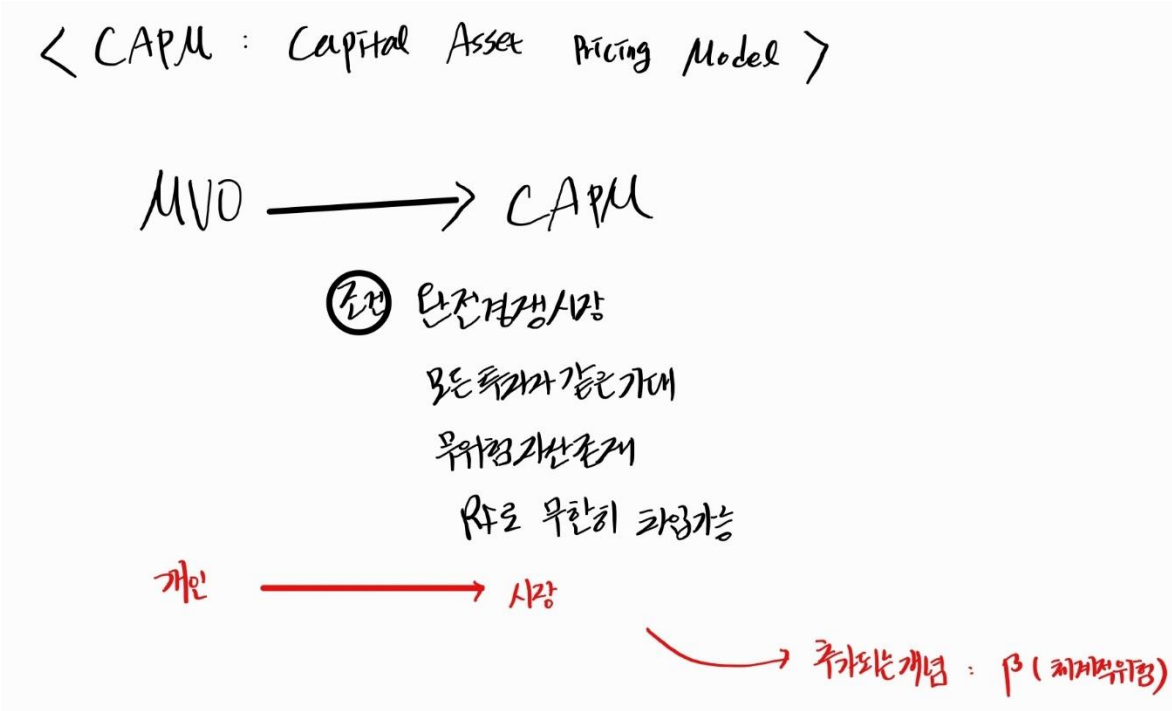
homogeneous Efficient Frontier

↓

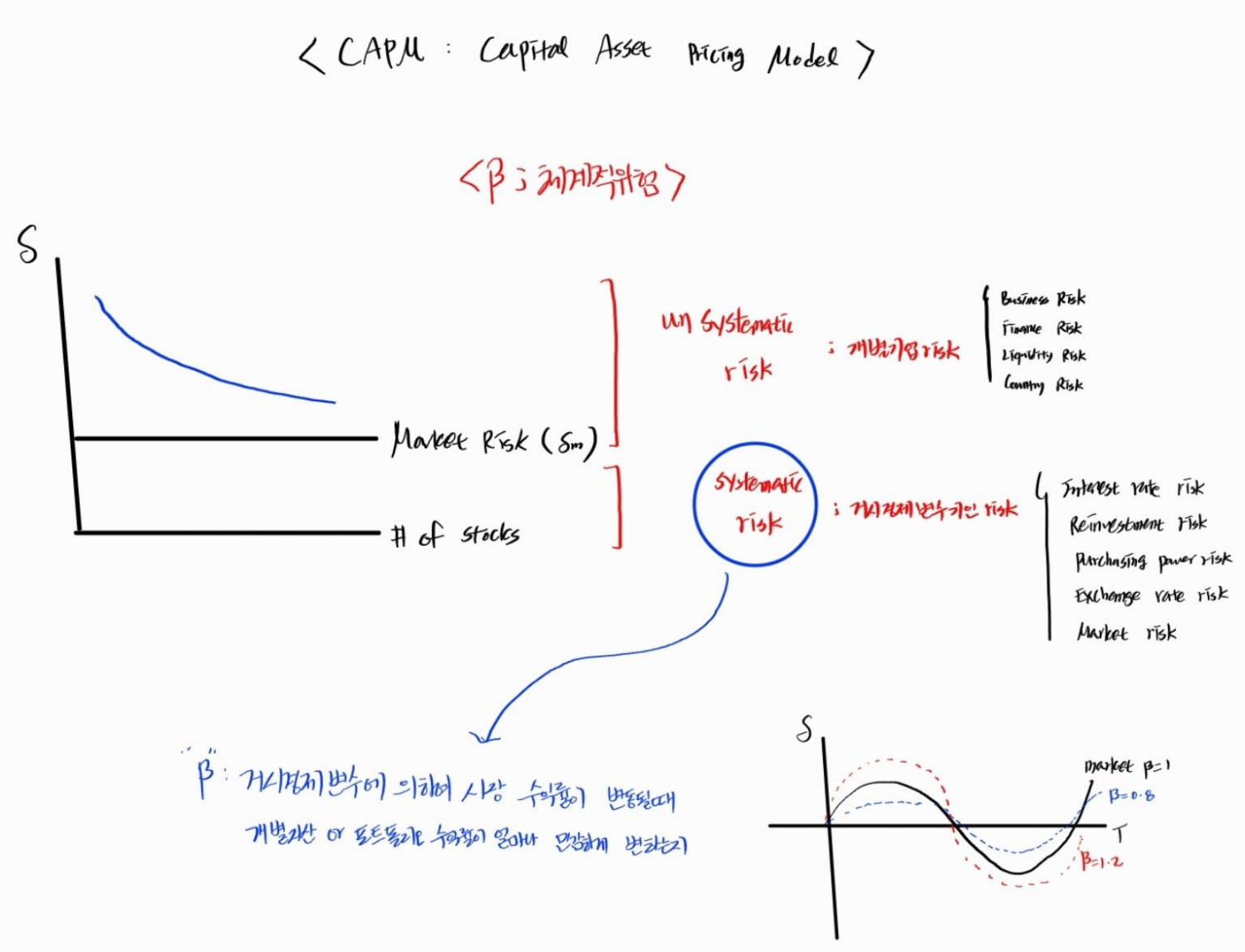
CML (Individual → Market)



Introduction of Different Weighting Method

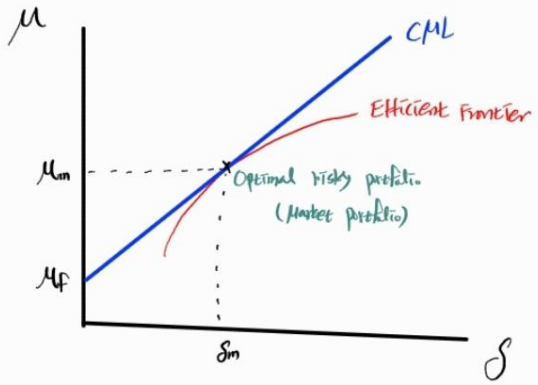


Introduction of Different Weighting Method



Introduction of Different Weighting Method

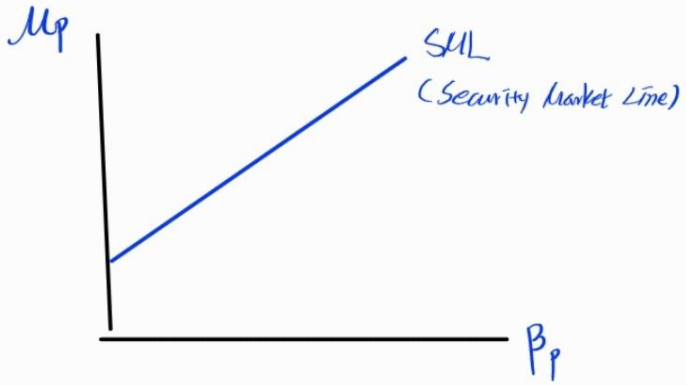
< CAPM : Capital Asset Pricing Model >



CML: $\mu = \frac{\mu_m - \mu_f}{\sigma_m}\sigma + \mu_f$

↓ unsystematic risk 제거

(x market 리스크 상관계수)



포트폴리오 μ_p, σ_p 구하기 :

$$\mu_p = \frac{(\mu_m - \mu_f)}{\sigma_m} \cdot \rho_{p,m}\sigma_p + \mu_f$$

$$\mu_p = \frac{(\mu_m - \mu_f)}{\sigma_m} \cdot \frac{Cov(\mu_p, \mu_m)}{\sigma_p \cdot \sigma_m} \cdot \sigma_p + \mu_f$$

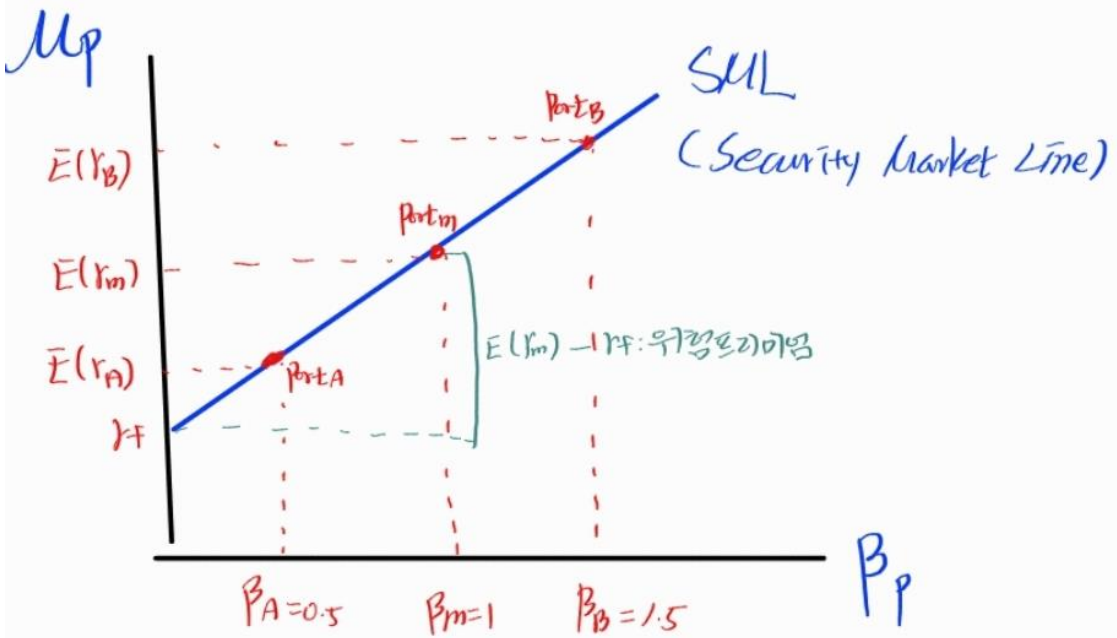
$$\mu_p = (\mu_m - \mu_f) \cdot \frac{Cov(\mu_p, \mu_m)}{\sigma_m^2} + \mu_f$$

$= \beta_p$

$\beta_p = \frac{Cov(\mu_p, \mu_m)}{}$: 시장이 얼마나 민감하?

Introduction of Different Weighting Method

< CAPM : Capital Asset Pricing Model >



$$= (\mu_m - \mu_f) \cdot \frac{Cov(\mu_p, \mu_m)}{\sigma_m^2} + \mu_f$$

$$= \beta_p$$

$$E(R_p) = R_f + \beta (E(R_m) - R_f)$$

$$E(R_A) = R_f + 0.5 (E(R_m) - R_f)$$

$$E(R_B) = R_f + 1.5 (E(R_m) - R_f)$$

$$\therefore E(R_A) < E(R_m) < E(R_B)$$

where $\beta_A < \beta_{m=1} < \beta_B$

Introduction of Different Weighting Method

< CAPM : Capital Asset Pricing Model >

* 사업에 존재하는 자본자산들의 사업가치를 구하기 위해서는 → 기대수익률 을 알아야한다.
  (→ Capital Asset)

= 할인율

미래에 발생할 현금흐름을 현재가치로 "할인"할 = 현재가치

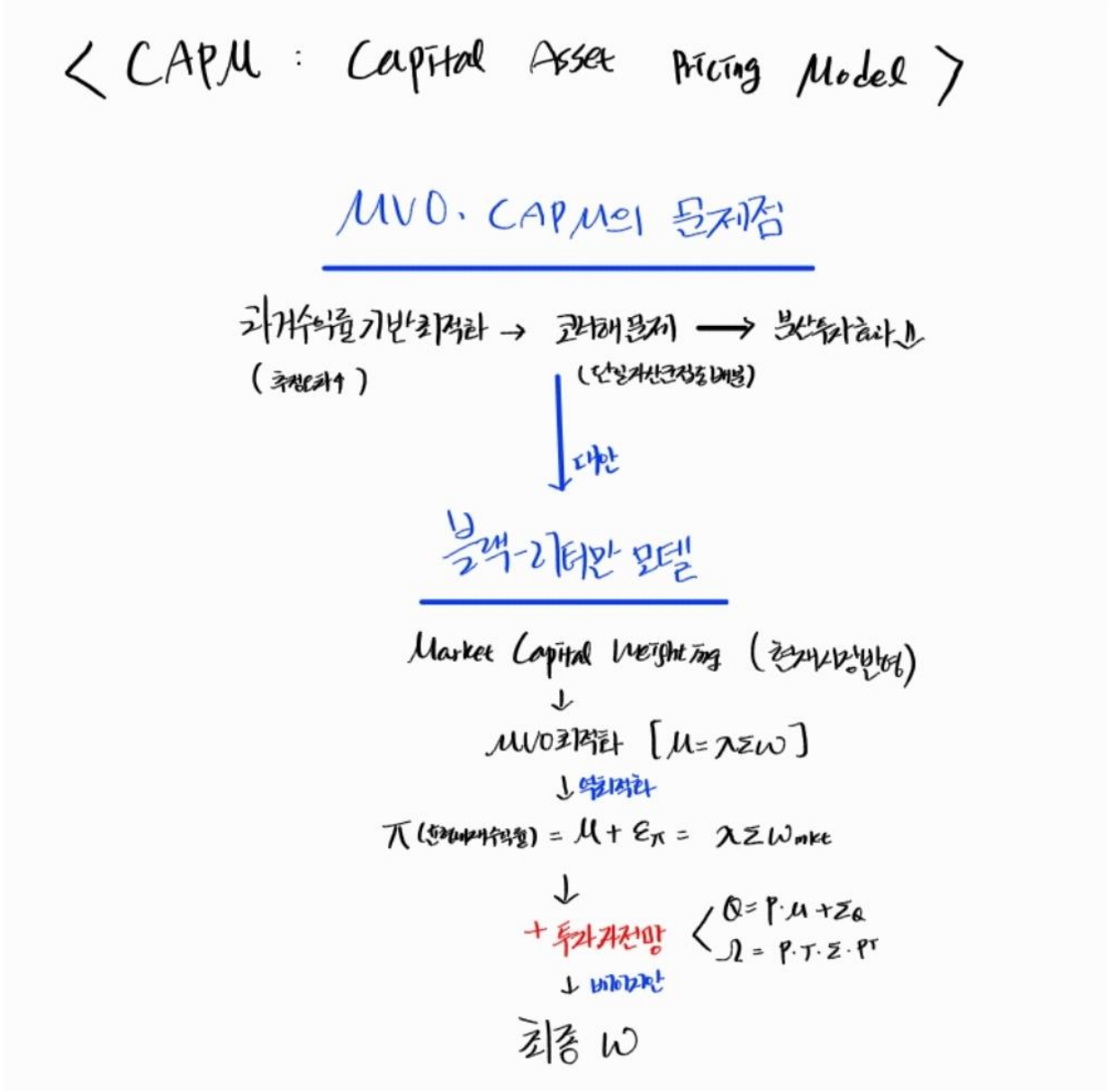
$\beta_p > \beta_m = 1$

$E(R_p) > E(R_m)$ → 할인율↑ → 현재가치 Low (저평가) → 향후가격↑

$\beta_p < \beta_m = 1$

$E(R_p) < E(R_m)$ → 할인율↓ → 현재가치 High (고평가) → 향후가격↓

Introduction of Different Weighting Method



Introduction of Different Weighting Method

< Black Litterman Model >

[사전분포]

$$\max_w \left[w^T \mu - \frac{\lambda}{2} w^T \Sigma w \right]$$

$$\lim_{\lambda \to 0} \left(\mu \cdot w - \frac{\lambda}{2} \Sigma w^2 \right) = 0$$

$$\mu - \lambda \Sigma w = 0$$

$$\mu = \lambda \Sigma w$$

subject to $\sum_{k=1}^{N} w_k = 1$

$0 \le w_k \le 1$

조합기대수익률 ← 역최적화 시장포트폴리오 비중 (w_{mkt})가 내재되어 있는 수익률 역으로 추정

$$\Pi = \lambda_{mkt} \cdot \Sigma \cdot w_{mkt} = \mu + \varepsilon_\Pi, \quad \varepsilon_\Pi \sim N(0, \tau\Sigma)$$

$$\mu \sim N(\Pi, \Sigma)$$

where $\lambda_{mkt} = \frac{R_m - R_f}{\sigma_m^2}$; 위험회피계수 (1×1)

Σ ; 자기 수익률 공분산행렬 (n×n)

w_{mkt} ; 시장포트폴리오 투자비중 (시가총액 비중) (n×1)

μ ; 분포를 알고 싶은 실제 기대수익률 벡터 (모수) (n×1)

ε_Π : 기대수익률 벡터 오차항 (n×1)

τ : 위험조정상수

Introduction of Different Weighting Method

< Black Litterman Model >

[사전분포]

$\Pi = \lambda_{mkt} \Sigma \cdot W_{mkt} = \mu + \varepsilon_\pi, \quad \varepsilon_\pi \sim N(0, \tau\Sigma)$

$\mu \sim N(\pi, \tau\Sigma)$

λ(위험회피계수) $= \dfrac{E(r_m) - r_f}{S_m^2}$; 시장위험 1을 감내해 얼마나 많은 초과수익률을 필요로 하는가

$S_m^2 = W_{mkt}^T \Sigma W_{mkt}$ (시장위험대비 시장투자자들이 원하는 초과수익률 ; 리스크 프리미엄)

τ(위험조정상수) : Sampling을 통하여 Π(내재균형수익률)을 구해주는 과정에서 실제 Market 데이터와 표본데이터와의 괴리정

Sampled N↑ → std↓

Market std < sample std → scaling down

$\tau = \dfrac{1}{\text{No of sampling data}}$

Introduction of Different Weighting Method

< Black Litterman Model >

[시장전망을토]

$$Q = P \cdot E(r) + \varepsilon_Q \quad , \quad \varepsilon_Q \sim N(0, \Omega)$$

$$\Omega = P(\tau\Sigma)P^T$$

where

P : 투자자전망행렬 $(K \times N)$

$E(r)$: 각 자산별 평균수익률 벡터 $(N \times 1)$

Q : 투자자전망기대수익률 $(K \times 1)$

ε : 투자자전망오차 $(K \times 1)$

τ : 위험조정상수

Ω : 각 투자전망의 불확실성을 나타내는 오차항공분산 대각행렬 $(K \times K)$

Introduction of Different Weighting Method

< Black Litterman Model >

[시장전망분포]

$Q = P \cdot E(r) + \varepsilon_Q \quad , \quad \varepsilon_Q \sim N(Q, \Omega)$

$\Omega = P_k (\tau\Sigma) P_k^T$

↳ K개의 투자자전망의 불확실성 → N개 자산사이의 공분산행렬 (Σ)을 통해 P차항 계산

N개 자산사이의 관계 ($\tau\Sigma$)에 전망행렬 (P)를 곱해서 P차항 계산

K=2일때, $\Omega = \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_2 \end{bmatrix} = \begin{bmatrix} (P_1\Sigma P_1')\tau & 0 \\ 0 & (P_2\Sigma P_2')\tau \end{bmatrix}$ = 각 투자관점의 불확실성을 나타내는 P차항의 대각행렬 (KXK)

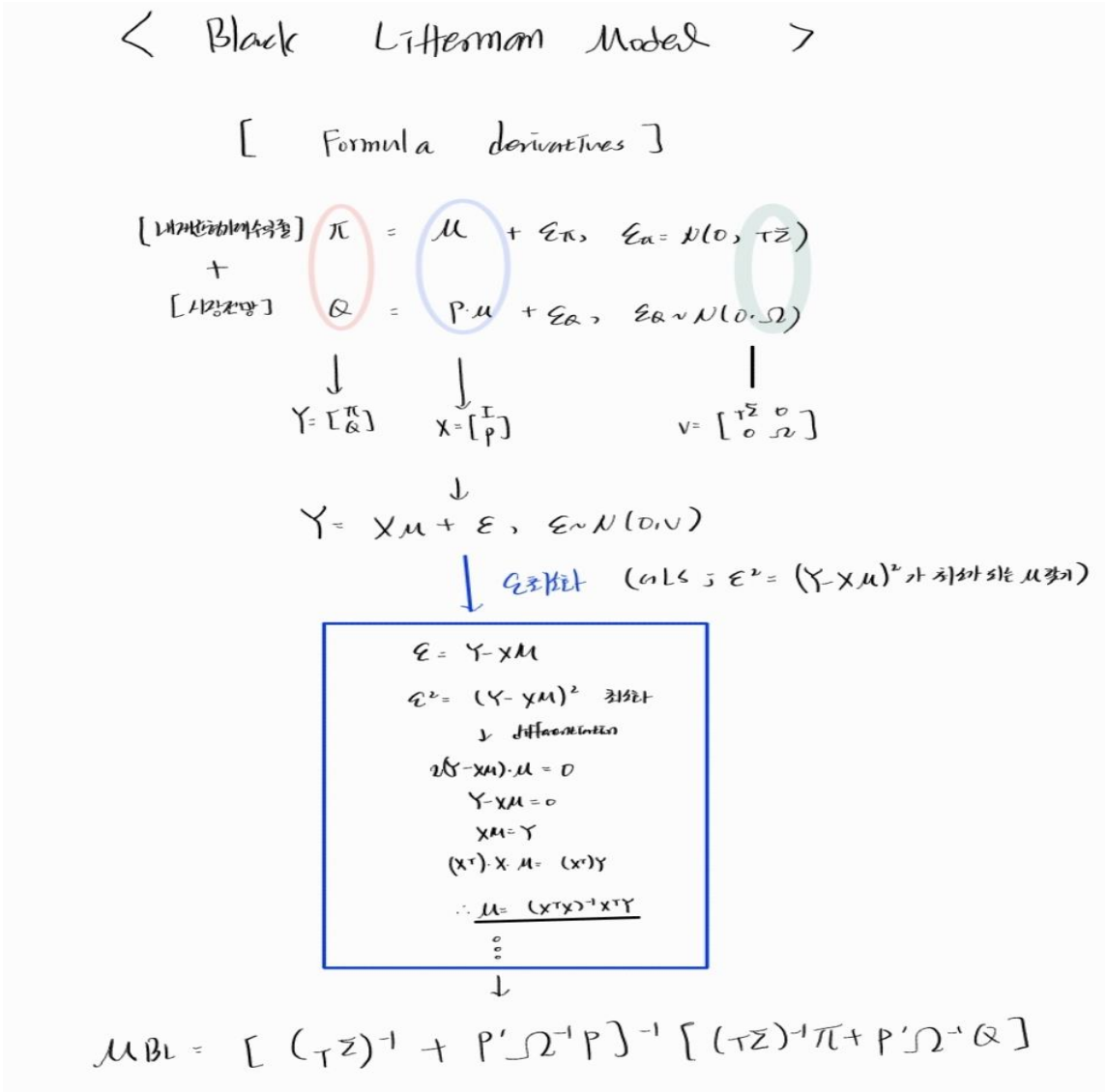
*각 전망에 대한 P라는 독립성 → P차항을 대각성분으로 하는 대칭행렬로 표현

+ Certainty

$\Omega = \frac{C}{1-C} \tau P_k \Sigma P_k'$

C : 전망확실성 (0%~100%) < C=50 → $\Omega = \tau P_k \Sigma P_k'$ / C=30 → $\Omega = \frac{7}{3} \cdot \tau P_k \Sigma P_k'$; 전망의 대가

Introduction of Different Weighting Method



Introduction of Different Weighting Method

1. Equal Weighting

출처: https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/equal-weighted-index/

Introduction of Different Weighting Method

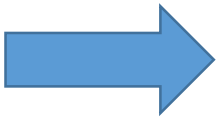
2. Markowitz Mean-Variance Optimization Weighting (Efficient Frontier, Maximum Sharpe)

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

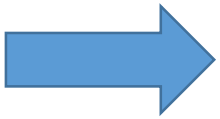
R_p = Return of portfolio

R_f = Risk-Free rate

σ_p = Standard deviation of portfolio's excess return



Random weights portfolios



Maximum Sharpe Ratio

Introduction of Different Weighting Method

2. Markowitz Mean-Variance Optimization Weighting (Efficient Frontier, Maximum Sharpe)

- PyPortfolioOpt로 진행

```
!pip install PyPortfolioOpt
```

```
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
```

```
# Sharpe Ratio 최적화 진행

Ef_Front = EfficientFrontier(return_mean_1Q_annual, Cov_1Q)
weights = Ef_Front.max_sharpe()
print(weights)
cleaned_weights = Ef_Front.clean_weights()
print(cleaned_weights)
Ef_Front.portfolio_performance(verbose=True)
```

```
OrderedDict([(('Samsung',), 0.0), (('SkHynix',), 0.0), (('Samsung Bio',), 0.0), (('LG Chemical',), 0.0), (('NAVER',), 0.42072932
6214383), (('Hyundai Car',), 0.0), (('Samsung SDI',), 0.0), (('KAKAO',), 0.5792706737856168)])
```

Introduction of Different Weighting Method

3. Market-Cap Weighting

출처: https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/equal-weighted-index/

Introduction of Different Weighting Method

3. Market-Cap Weighting

```
start = '20210101'
end = '20211231'
year = 1

stock_names = ['Samsung', 'SkHynix', 'Samsung Bio', 'LG Chemical', 'NAVER', 'Hyundai Car', 'Samsung SDI', 'KAKAO' ]
stock_codes = ['005930', '000660', '207940', '051910', '035420', '005380', '006400', '035720']

list_Cap = []
for code in stock_codes:

    Cap= stock.get_market_cap(start, end, code)
    list_Cap.append(pd.DataFrame(Cap['시가총액']))

Cap_df = pd.concat(list_Cap, axis=1)
Cap_df.columns = [stock_names]
print(Cap_df.head(10))

#-----------------------------------------------

start_1Q = '2021-01-03'
end_1Q = '2021-03-31'

start_2Q = '2021-04-01'
end_2Q = '2021-06-30'

start_3Q = '2021-07-01'
end_3Q = '2021-09-30'

start_4Q = '2021-10-01'
end_4Q = '2021-12-31'


Cap_df_1Q = Cap_df.loc[start_1Q:end_1Q]
Cap_df_2Q = Cap_df.loc[start_2Q:end_2Q]
Cap_df_3Q = Cap_df.loc[start_3Q:end_3Q]
Cap_df_4Q = Cap_df.loc[start_4Q:end_4Q]
```

Introduction of Different Weighting Method

3. Market-Cap Weighting

```
Cap_df.tail(10)
```

날짜	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
2021-12-17	465643038900000	88816288530000	63187575000000	49202863071000	63159275377500	44763485176500	46759880400000	52379723047500
2021-12-20	460270234605000	87724284982500	61864275000000	46308577008000	61270246335000	43801978335000	44972002620000	50373691101000
2021-12-21	466240017155000	90636294442500	61467285000000	45320284206000	61598773125000	43801978335000	44353121850000	51042368416500
2021-12-22	474000734470000	92456300355000	59614665000000	44120214375000	62173695007500	44122480615500	43459182960000	51042368416500
2021-12-23	476985625745000	92820301537500	59283840000000	44755545462000	62010554130000	44656651083000	44215592790000	50373691101000
2021-12-24	480567495275000	93184302720000	58556025000000	43837845003000	62994848640000	46045494298500	44765709030000	51042368416500
2021-12-27	478776560510000	91728297990000	58622190000000	44261399061000	62420676842500	45618157924500	44284357320000	50373691101000
2021-12-28	479373538765000	92820301537500	58953015000000	44614360776000	63076873182500	45724992018000	44765709030000	50596583539500
2021-12-29	470418864940000	92456300355000	58886850000000	44331991404000	62666750470000	45297655644000	45590883390000	50819475978000
2021-12-30	467433973665000	95368309815000	59746995000000	43414290945000	62092578672500	44656651083000	45040767150000	50150798662500

Introduction of Different Weighting Method

3. Market-Cap Weighting

```
def get_cap_weights (Cap_df =Cap_df_1Q):

    cap_weights= np.array([Cap_df.mean()[0]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[1]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[2]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[3]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[4]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[5]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[6]/np.sum(Cap_df.mean()),
                           Cap_df.mean()[7]/np.sum(Cap_df.mean())])

    return cap_weights
```

```
cap_weights_1Q = get_cap_weights (Cap_df =Cap_df_1Q)
cap_weights_1Q
```

```
array([0.54784961, 0.10614301, 0.05610548, 0.0709353 , 0.06441984,
       0.05547059, 0.05376858, 0.04530759])
```

Introduction of Different Weighting Method

4. Free-Float Weighting (Float Adjusted Market Cap Weighting)

Free-Float Weighting : Outstanding Stock 으로 환산한 시가총액으로 Weighting

Outstanding Stock = Issued Stock – Treasury Stock – Major Shareholder Stock

Float Adjusted Market Cap = Outstanding Stock * Price of Stock

Introduction of Different Weighting Method

4. Free-Float Weighting (Float Adjusted Market Cap Weighting)

(기준일 : 2021년 03월 31일) (단위 : 주)

구 분		주식의 종류		비고
		보통주	합계	
Ⅰ. 발행할 주식의 총수		300,000,000	300,000,000	-
Ⅱ. 현재까지 발행한 주식의 총수		240,638,520	240,638,520	-
Ⅲ. 현재까지 감소한 주식의 총수		76,375,125	76,375,125	-
	1. 감자	-	-	-
	2. 이익소각	550,000	550,000	자기주식 소각
	3. 상환주식의 상환	-	-	-
	4. 기타	75,825,125	75,825,125	인적분할
Ⅳ. 발행주식의 총수 (Ⅱ-Ⅲ)		164,263,395	164,263,395	-
Ⅴ. 자기주식수		15,944,705	15,944,705	-
Ⅵ. 유통주식수 (Ⅳ-Ⅴ)		148,318,690	148,318,690	-

Introduction of Different Weighting Method

4. Free-Float Weighting (Float Adjusted Market Cap Weighting)

구분	주주명	소유주식수	지분율(%)	비고
5% 이상 주주	국민연금공단	509,502,939	8.53	-
	삼성생명보험㈜	522,017,952	8.74	특별계정 포함
	BlackRock Fund Advisors	300,391,061	5.03	2019년 1월 28일 기준
	삼성물산㈜	298,818,100	5.01	-
	우리사주조합	-	-	-

※ 5% 이상 주주는 의결권 있는 주식(기타 법률에 의해 의결권 행사가 제한된 주식도 포함) 기준입니다.
자세한 현황은 'VI. 이사회 등 회사의 기관에 관한 사항' 중 '3. 주주총회 등에 관한 사항'의 '라. 의결권 현황'을 참고하시기 바랍니다.

※ BlackRock Fund Advisors의 소유주식수 및 지분율은 2019년 2월 7일 전자공시시스템(http://dart.fss.or.kr)에 공시된 '주식등의 대량보유상황보고서' 기준입니다.

Introduction of Different Weighting Method

4. Free-Float Weighting (Float Adjusted Market Cap Weighting)

```
import pandas as pd
file_names = ['Float_2020_4Q.csv', 'Float_2021_1Q.csv', 'Float_2021_2Q.csv', 'Float_2021_3Q.csv']


def get_float_df (file_name = file_names[0]):


    df = pd.read_csv(f'../input/float-df/{file_name}', encoding='cp949', index_col = '종목 ')

    try:
        df.drop(columns=['Unnamed: 6'], inplace=True)

    except:
        pass

    df['유동주식수'] = df['발행주식수\n(보통주) '] - df['자기주식수'] - df['우리사주조합'] - df['지분율 5% 이상 최대주주들 주식수']

    df.columns = ['발행주식수', '자기주식수', '우리사주조합', '최대주주(지분5%이상)', '최대주주명',
           '유동주식수']

    df['유동주식비율'] = df.유동주식수 / df.발행주식수

    return df


float_list = []
for file_name in file_names:

    df = get_float_df(file_name)
    float_list.append(df)
```

종목	발행주식수	자기주식수	우리사주조합	최대주주(지분5%이상)	최대주주명	유동주식수	유동주식비율
삼성전자	5969782550	0	0	1762338966	국민연금공단 외 4	4207443584	0.704790
SK하이닉스	728002365	44000570	0	225985445	에스케이텔레콤㈜ 외 1	458016350	0.629141
삼성바이오로직스	66165000	0	33293	49579298	삼성물산 외 1	16552409	0.250169
LG화학	70592343	464842	0	30427781	㈜ LG 외 1	39699720	0.562380
네이버	164263395	16804360	0	27269118	국민연금공단 외 1	120189917	0.731690
현대차	213668187	12766233	0	79411043	현대모비스 외 1	121490911	0.568596
삼성SDI	68764530	3331391	0	23764312	삼성전주㈜ 외 2	41668827	0.605964
KAKAO	88501998	2489934	0	35580364	김범수 외 3	50431700	0.569837

<2020_4Q>

종목	발행주식수	자기주식수	우리사주조합	최대주주(지분5%이상)	최대주주명	유동주식수	유동주식비율
삼성전자	5969782550	0	0	244619819	삼성중공업 외 4	5725162731	0.959024
SK하이닉스	728002365	40354565	0	1415650165	SK스퀘어 외 1	-728002365	-1.000000
삼성바이오로직스	66165000	0	23017	53591050	삼성물산 외 2	12550933	0.189691
LG화학	70592343	367529	0	28331236	㈜LG 외 1	41893578	0.593458
네이버	23534211	14596762	0	22431486	국민연금공단 외 1	-13494037	-0.573380
현대차	213668187	14902914	1929938	73807547	현대모비스 외 2	123027788	0.575789
삼성SDI	68764530	3331391	0	22362343	삼성전자 외 1	43070796	0.626352
KAKAO	446270406	12136741	0	163952172	김범수 외 3	270181493	0.605421

<2021_1Q>

Introduction of Different Weighting Method

4. Free-Float Weighting (Float Adjusted Market Cap Weighting)

```
def get_float_adj_weights (Cap_df =Float_Adj_Cap_1Q):

    float_adj_weights= np.array([Cap_df[0]/np.sum(Cap_df),
                                 Cap_df[1]/np.sum(Cap_df),
                                 Cap_df[2]/np.sum(Cap_df),
                                 Cap_df[3]/np.sum(Cap_df),
                                 Cap_df[4]/np.sum(Cap_df),
                                 Cap_df[5]/np.sum(Cap_df),
                                 Cap_df[6]/np.sum(Cap_df),
                                 Cap_df[7]/np.sum(Cap_df)])

    return float_adj_weights




cap_weights_1Q = get_cap_weights (Cap_df =Cap_df_1Q)
cap_weights_2Q = get_cap_weights (Cap_df =Cap_df_2Q)
cap_weights_3Q = get_cap_weights (Cap_df =Cap_df_3Q)
cap_weights_4Q = get_cap_weights (Cap_df =Cap_df_4Q)

float_adj_weights_1Q = get_float_adj_weights (Cap_df =Float_Adj_Cap_1Q)

float_adj_weights_1Q
```

```
array([0.57052368, 0.13491566, 0.01890393, 0.06164895, 0.06956285,
       0.04703316, 0.05273869, 0.04467308])
```

Introduction of Different Weighting Method

5. Weighting by Π (Implied Returns)

$$\Pi = \delta \Sigma \omega$$

δ = Risk aversion coefficient. It can either be an arbitrary assumption or can be given by $\delta = (E(r) - r_f)/\sigma^2$

- $E(r)$ = Return of the market portfolio (a portfolio that includes all the assets in the market or any other index benchmark that the investor decide to choose)
- r_f = Risk free market rate
- σ^2 = Variance of the market portfolio

Σ = A covariance matrix of the assets (NxN matrix)

ω = Weights of assets according to their market capitalization

Results

Risk Free Rate

	2021 01월	2021 02월	2021 03월	2021 04월	2021 05월	2021 06월	2021 07월	2021 08월	2021 09월	2021 10월	2021 11월	2021 12월	2022 01월
국고채 3년(평균)	0.98	1.00	1.13	1.14	1.13	1.30	1.42	1.41	1.52	1.84	1.95	1.80	2.06
국고채 5년(평균)	1.32	1.35	1.55	1.58	1.65	1.70	1.69	1.65	1.79	2.15	2.17	1.98	2.28
국고채 10년(평균)	1.73	1.85	2.04	2.04	2.13	2.10	1.98	1.91	2.06	2.40	2.36	2.19	2.49
회사채 3년(평균)	2.14	2.06	2.09	1.98	1.89	1.91	1.89	1.84	1.95	2.30	2.49	2.41	2.63
CD 91물(평균)	0.68	0.73	0.75	0.74	0.68	0.66	0.69	0.77	0.98	1.08	1.17	1.27	1.39
콜금리(1일물,평균)	0.49	0.49	0.49	0.48	0.48	0.51	0.53	0.56	0.77	0.74	0.80	1.01	1.19
기준금리	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.75	0.75	0.75	1.00	1.00	1.25

출처 : e나라지표

Introduction of Different Weighting Method

5. Weighting by Π (Implied Returns)

```
import matplotlib.pylab as plt

import numpy as np
from numpy.linalg import inv
import pandas as pd
from pandas_datareader import data as web

from scipy.optimize import minimize


# 무위험수익률, 수익률,공분산으로
# 샤프비율을 최대로 하는 접점포트폴리오 비중 계산
def solveWeights(R, C, rf):

    # 파이썬은 함수안에 함수를 정의할 수 있다
    # 최적비중계산를 위해 다음과 같이 목적함수를 정의한다
    def obj(W, R, C, rf):
        mean = sum(R * W)
        var = np.dot(np.dot(W, C), W)
        #샤프비율를 효용함수로 한다
        util = (mean - rf) / np.sqrt(var)
        # 효용함수 극대화= 효용함수 역함수를 최소화하는 것이다.
        return 1 / util

    n = len(R)  # 투자자산 갯수

    # 동일비중으로 최적화 시작
    W = np.ones([n]) / n
    # 비중범위는 0~100%사이(공매도나 차입조건이 없음)
    bnds = [(0., 1.) for i in range(n)]
    # 제약조건은 비중합=100%
    cons = ({'type': 'eq', 'fun': lambda W: sum(W) - 1.})
    # 최적화
    res = minimize(obj, W, (R, C, rf), method='SLSQP', constraints=cons, bounds=bnds)
    if not res.success:
      # 최적화 실패한 경우
      raise BaseException(res.message)
    # 최적화 결과를 돌려준다
    return res.x
```

Introduction of Different Weighting Method

5. Weighting by Π (Implied Returns)

```
pi_weights_1Q = solveWeights(pi_1Q+Rf, Cov_Annual_1Q, Rf)
pi_weights_1Q
```

```
: array([0.57061686, 0.13495283, 0.01880986, 0.06160627, 0.06932937,
         0.04700903, 0.05279855, 0.04487723])
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

$$E(R) = [(\tau \Sigma)^{-1} + P^T \Omega P]^{-1} [(\tau \Sigma)^{-1} \Pi + P^T \Omega Q]$$

- τ = A scalar number indicating the uncertainty of the CAPM distribution (It is usually within the range of 0.025-0.05)
- P = A matrix with investors views; each row a specific view of the market and each entry of the row represents the weights of each assets (KxN matrix)
- Q = The expected returns of the portfolios from the views described in matrix P (Kx1 vector)
- Ω = A diagonal covariance matrix with entries of the uncertainty within each view (KxK matrix)

Σ and Π as described in previous section.

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률)

[1]
NLP 진행
-> 시장(투자자)전망

[2]
Conv1d+ Lstm 진행
-> 시장(투자자)전망

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률)

1]NLP 진행 -> 시장(투자자)전망

1. 네이버 분기별 뉴스기사제목 Scraping
   - Selenium (Google Webdriver) + BeautifulSoup
   - 2021년 1분기 -> 01/01~01/31 Scraping -> Weighting -> 02/01 ~ 02/28 Rebalancing
   - 2021년 2분기 -> 04/01~04/30 Scraping -> Weighting -> 05/01 ~ 05/31 Rebalancing
   - 2021년 3분기 -> 07/01~07/31 Scraping -> Weighting -> 08/01 ~ 08/31 Rebalancing
   - 2021년 4분기 -> 10/01~10/31 Scraping -> Weighting -> 11/01 ~ 11/30 Rebalancing

2. Konlpy (코엔엘파이) 라이브러리 -> 형태소분석/ 토큰화진행
   - Preprocessing
   - Tokenization

3. 비지도감성분석 진행
   - 감성분석사전

4. Pos_rate & Neg_rate => Total_score
   - Pos_rate = positive matching words/ positive matching words + negative matching words
   - Neg_rate = negative matching words/ positive matching words + negative matching words
   - Total_rate = Pos_rate – Neg_rate

5. Total_rate이 높은순으로 2종목씩 그룹화
   - 상위그룹부터 시장전망수익률 2%, 1.5%, 1%, 0.5% 부여
   - 상대적전망으로만 진행

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

1. 네이버 분기별 뉴스기사제목 Scraping
   - Selenium (Google Webdriver) + BeautifulSoup
   - 2021년 1분기, 2분기, 3분기, 4분기

```
['롯데도 상반기 수시채용 전환… 5대그룹중 삼성만 공채 유지',
 "8만원도 '흔들' 삼성전자, 9층 개미 '구출' 가능할까",
 "'삼성전자가 준비한 봄 맞이 선물'",
 '삼성證 "삼성전자 1Q 영업익 9.2조 전망…기대치 넘기 어려워"',
 '삼성, 친환경 갤럭시S 시리즈로 'ESG' 앞장',
 '삼성전자 국민가전 페스타',
 '대법 "공기청정기 바이러스 99%제거는 과장광고" ..삼성전자 사실상 패소',
 "삼성 Neo QLED 8K, 와이파이 6E 인증 획득 'TV 최초'",
 '신한금투 "삼성전자 주가 충분히 쉬었다…선제적 매수 추천"',
 '삼성이노베이션뮤지엄, 비대면 시대 온라인 전시관으로 변신',
 ''코로나의 역설' 삼성·LG, 판촉비 줄여 수익성 키웠다',
 "8만전자에 갇힌 삼성전자…'12만원' 외치는 증권가, 왜",
 '[김세형 칼럼] 삼성전자 10년후에도 건재할까',
 '삼성 美오스틴 공장 6주만에 가동',
 "삼성전자, '국민가전 페스타' 실시…최대 80만원 상당 풍성 혜택",
 "삼성전자, 미국 환경보호청 'SMM 어워드' 수상",
 "삼성전자 '비스포크 인덕션' 신제품 출시",
 '"최대 80만 원 혜택" 삼성전자, ₩'국민가전 페스타₩' 실시',
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

2. Konlpy (코엔엘파이) 라이브러리 -> 형태소분석/ 토큰화진행
   - Preprocessing ; 정규식 (re)

```
import pandas as pd
Samsung_title_10 = pd.DataFrame(titles_list_df)
Samsung_title_10
```

	0
0	롯데도 상반기 수시채용 전환... 5대그룹중 삼성만 공채 유지
1	8만원도 '흔들' 삼성전자, 9층 개미 '구출' 가능할까
2	'삼성전자가 준비한 봄 맞이 선물'
3	삼성證 "삼성전자 1Q 영업익 9.2조 전망...기대치 넘기 어려워"
4	삼성, 친환경 갤럭시S 시리즈로 'ESG' 앞장
...	...
11035	삼성, 17일 온라인 언팩 초대장 발송..."갤A52·갤A72 공개 예상"
11036	[카드뉴스] 삼성에게 'A'가 중요해진 3가지 이유
11037	삼성전자, '그랑데 AI' 앞세워 동남아 세탁기시장 공략 속도
11038	[부고] 김현석(삼성전자 대표이사 사장)씨 장인상
11039	"리모델링-가전교체 한번에" 삼성전자 손 잡은 한샘...스타일패키지 출시

```
def preprocessing (x):
    x = re.sub(r"\d+", " ", x)
    x = re.sub("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]", "", x)
    x = re.sub("  ", "", x)
    return x
```

+ Code + Markdown

```
import re
from tqdm import tqdm

preprocess_title = []

for title in tqdm(Samsung_title_1Q[0]):
    x = re.sub(r"\d+", " ", title)
    x = re.sub("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]", "", x)
    x = re.sub("  ", "", x)
    preprocess_title.append(x)

Samsung_title_1Q['preprocess'] = preprocess_title
Samsung_title_1Q
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

2. Konlpy (코엔엘파이) 라이브러리 -> 형태소분석/ 토큰화진행
   - Preprocessing

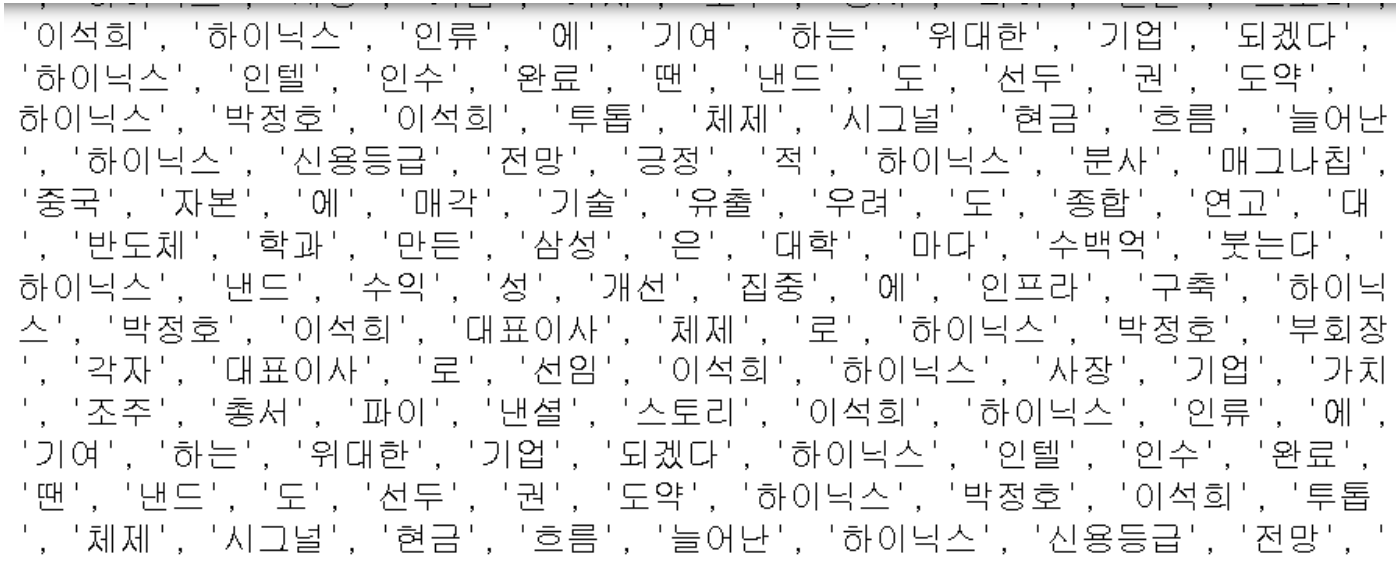
	0	preprocess
0	롯데도 상반기 수시채용 전환… 5대그룹중 삼성만 공채 유지	롯데도 상반기 수시채용 전환대그룹중 삼성만 공채 유지
1	8만원도 '흔들' 삼성전자, 9층 개미 '구출' 가능할까	만원도 흔들 삼성전자층 개미 구출 가능할까
2	'삼성전자가 준비한 봄 맞이 선물'	삼성전자가 준비한 봄 맞이 선물
3	삼성證 "삼성전자 1Q 영업익 9.2조 전망…기대치 넘기 어려워"	삼성 삼성전자 영업익 조 전망기대치 넘기 어려워
4	삼성, 친환경 갤럭시S 시리즈로 'ESG' 앞장	삼성 친환경 갤럭시 시리즈로앞장
...	...	...
11035	삼성, 17일 온라인 언팩 초대장 발송…"갤A52·갤A72 공개 예상"	삼성일 온라인 언팩 초대장 발송갤 갤공개 예상
11036	[카드뉴스] 삼성에게 'A'가 중요해진 3가지 이유	카드뉴스 삼성에게 가 중요해진가지 이유
11037	삼성전자, '그랑데 AI' 앞세워 동남아 세탁기시장 공략 속도	삼성전자 그랑데앞세워 동남아 세탁기시장 공략 속도
11038	[부고] 김현석(삼성전자 대표이사 사장)씨 장인상	부고 김현석삼성전자 대표이사 사장씨 장인상
11039	"리모델링-가전교체 한번에" 삼성전자 손 잡은 한샘…스타일패키지 출시	리모델링가전교체 한번에 삼성전자 손 잡은 한샘스타일패키지 출시

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

2. Konlpy (코엔엘파이) 라이브러리 -> 형태소분석/ 토큰화진행
   - Tokenization



Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

3. 비지도감성분석 진행
   - 감성분석사전

KOSELF 감성사전 사용 (기업 재무분석을 위한 한국어 감성사전 구축)

- 한국어 기반으로 개발한 KOSAC(서울대) & KNU(군산대) 감성사전
  +
  영어감성사전 Harvard IV(HV) & Loughran and McDonald(LM) 구글번역

  = KOSELF (단국대)

출처:
기업 재무분석을 위한 한국어 감성사전 구축
Article information
Korean J Financ Stud. 2021;50(2):135-170
Publication date (electronic) : 2021 April 30
조수지, 김흥규, 양철원,

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

3. 비지도감성분석 진행
   - 감성분석사전

```
# 감성 사전

neg = ['결국, 결함, 공허한, 과적, 극심한, 둔화, 마이너스, 몰수, 미완성,

neg = neg[0].split(',')

neg_dic = [i.replace(' ','') for i in neg]

pos = ['가치, 가치 있는, 강세, 개선, 개선된, 개선되는, 경신, 경의, 고급,

pos = pos[0].split(',')

pos_dic = [i.replace(' ','') for i in pos]

print('<부정어 감성사전>')
print(neg_dic)
print('₩n')
print('<긍정어 감성사전>')
print(pos_dic)
```

<부정어 감성사전>
['결국', '결함', '공허한', '과적', '극심한', '둔화', '마이너스', '몰수', '미완성', '배상', '부적합한', '부정적', '부정적인', '부주의', '불리한', '소란', '스캔들', '실패', '악화', '약점', '약화', '여파', '연기', '우발적', '의심', '의혹', '잘못', '저품질', '저해', '정체', '조치', '주의', '지나친', '지독한', '지연', '질타', '차질', '초라한', '충격', '침체', '투자회수', '파산', '평가절하', '하락', '해제', '혼란', '훼손']

<긍정어 감성사전>
['가치', '가치있는', '강세', '개선', '개선된', '개선되는', '경신', '경의', '고급', '기꺼이', '더좋은', '도움이되는', '뛰어나', '명성', '바닥', '본격적인', '상승효과', '상회한', '성공', '수혜', '순조롭게', '실현', '안정된', '완전한', '우세', '우월', '우호적', '심의', '여지가없는']

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

4. Pos_rate & Neg_rate => Total_score

$$POS\% = \frac{\text{긍정어수}}{\text{부정어수}+\text{긍정어수}}$$

$$NEG\% = \frac{\text{부정어수}}{\text{부정어수}+\text{긍정어수}}$$

Total_ratio = POS % - NEG%

출처:
기업 재무분석을 위한 한국어 감성사전 구축
Article information
Korean J Financ Stud. 2021;50(2):135-170
Publication date (electronic) : 2021 April 30
조수지, 김흥규, 양철원

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

4. Pos_rate & Neg_rate => Total_score

```python
# 사전 비교

def get_comparision (tokens_list=tokens_list, pos=pos_dic, neg=neg_dic):

    import numpy as np

    N = len(tokens_list)

    res = np.zeros(shape=(N,2), dtype= np.int64)


    for i in range(N):
        t = tokens_list[i]
        cnt = 0

        for j in range(len(pos)):
            p = pos[j]

            if t == p:
                cnt += 1
            else:
                pass

        res[i, 1] = cnt

    for i in range(N):
        t = tokens_list[i]
        cnt = 0

        for j in range(len(neg)):
            n = neg[j]

            if t == n:
                cnt += 1
            else:
                pass

        res[i, 0] = cnt


    return res


Samsung_1Q_res = get_comparision (tokens_list=Samsung_1Q_tokens_list, pos=pos_dic, neg=neg_dic)
```

	Samsung
pos_matching	55.000000
neg_matching	103.000000
pos_rate	0.348101
neg_rate	0.651899
total_rate	-0.303797

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

5. Total_rate이 높은순으로 2종목씩 그룹화
   - 상위그룹부터 시장전망수익률 2%, 1.5%, 1%, 0.5% 부여
   - 상대적전망으로만 진행

	Samsung	SkHynix	Samsung_Bio	LG_Chemical	NAVER	Hyundai_Car	Samsung_SDI	KAKAO
pos_matching	55.000000	121.00000	101.000000	43.000000	99.000000	89.000000	49.000000	38.000000
neg_matching	103.000000	78.00000	91.000000	24.000000	124.000000	29.000000	58.000000	49.000000
pos_rate	0.348101	0.60804	0.526042	0.641791	0.443946	0.754237	0.457944	0.436782
neg_rate	0.651899	0.39196	0.473958	0.358209	0.556054	0.245763	0.542056	0.563218
total_rate	-0.303797	0.21608	0.052083	0.283582	-0.112108	0.508475	-0.084112	-0.126437

1그룹 : 삼성바이오로직스, 현대차
2그룹 : SK하이닉스, LG화학
3그룹 : 삼성SDI, 네이버
4그룹 : 카카오, 삼성

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – NLP

5. Total_rate이 높은순으로 2종목씩 그룹화
   - 상위그룹부터 시장전망수익률 2%, 1.5%, 1%, 0.5% 부여
   - 상대적전망으로만 진행

1그룹 : 삼성바이오로직스, 현대차
2그룹 : SK하이닉스, LG화학
3그룹 : 삼성SDI, 네이버
4그룹 : 카카오, 삼성

```
# 시장전망수익률 Q / 2% 1.5% 1% 0.5%

Q = np.array([0.02, 0.015, 0.01, 0.005])

# 시장전망행렬

P = np.array([[-0.25, -0.25, 1-0.25, -0.25, -0.25, 1-0.25, -0.25, -0.25],
              [-0.25, -0.25, 1-0.25, -0.25, 1-0.25, -0.25, -0.25, -0.25],
              [-0.25, -0.25, -0.25, -0.25, 1-0.25, -0.25, 1-0.25, -0.25],
              [1-0.25, -0.25, -0.25, -0.25, -0.25, -0.25, -0.25, 1-0.25]])
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

$$E(R) = [(\tau \Sigma)^{-1} + P^T \Omega P]^{-1} [(\tau \Sigma)^{-1} \Pi + P^T \Omega Q]$$

```
# 시장전망수익률 Q / 2% 1.5% 1% 0.5%

Q = np.array([0.02, 0.015, 0.01, 0.005])

# 시장전망행렬

P = np.array([[-0.25, -0.25, 1-0.25, -0.25, -0.25, 1-0.25, -0.25, -0.25],
              [-0.25, -0.25, 1-0.25, -0.25, 1-0.25, -0.25, -0.25, -0.25],
              [-0.25, -0.25, -0.25, -0.25, 1-0.25, -0.25, 1-0.25, -0.25],
              [1-0.25, -0.25, -0.25, -0.25, -0.25, -0.25, -0.25, 1-0.25]])

# 위험조정상수 1/3개월 = 3


T = 1/3

# 투자자전망의 불확실성 = 시장전망행렬 * (위험조정상수 * 공분산) * 시장전망행렬.T

uncertainty = np.dot(np.dot(P, T*Cov), P.T)
```

```
# 블랙리터만 모형으로 결합전망기대수익률 구하기 / E(E(r)\균형기대수익률)

# 1. Part1

# 1-1. (T * 공분산).-1

Black_1_1 = np.linalg.inv(np.dot(T, Cov))

# 1-2. (P.T * 전망불확실성.-1 * P)

Black_1_2 = np.dot(np.dot(P.T, np.linalg.inv(uncertainty)), P)

# 1. [Black1 + Black2]-1

Black_1 = np.linalg.inv(Black_1_1 + Black_1_2)

# ------------------------------------------------------------------

# 2. Part 2

# 2-1 (T * 공분산).-1 * 균형기대수익률

Black_2_1 = np.dot(np.linalg.inv(T * Cov), balanced_Expected_R)

# 2-2 P.T * Uncertainty.-1 * Q

Black_2_2 = np.dot(np.dot(P.T, np.linalg.inv(uncertainty)), Q)


# 2. [Black_2_1 + Black_2_2]

Black_2 = Black_2_1 + Black_2_2

#------------------------------------------------------------------

# 3. Part 3

# 3. 최종 결합기대수익률,  E(r) = Black_1 * Black_2

Black_Er = np.dot(Black_1, Black_2) + Rf
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

$$E(R) = [(\tau \Sigma)^{-1} + P^T \Omega P]^{-1} [(\tau \Sigma)^{-1} \Pi + P^T \Omega Q]$$

```
pi_adj_weights_1Q = solveWeights(pi_adj_1Q+Rf, Cov_Annual_1Q, Rf)
pi_adj_weights_1Q
```

```
array([5.74530353e-01, 4.48528812e-02, 1.85966937e-01, 9.92666201e-18,
       1.23633542e-01, 3.27659601e-18, 4.22386628e-17, 7.10162872e-02])
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

1] Conv1d + Lstm 진행 -> 시장(투자자)전망

1. Feature Selection
   - Previous Returns
   - Previous Volume
   - Momentum related technical indicators
     - Moving Average / Bollinger Bands

2. Feature Engineering
   -Train / Test Spilt
   -Scaling (Features ; MinMax / target ; log1p)
   -Rolling Window (Features; 60 days / target ; 20 days)

3. Conv1d+ Lstm
   - Pytorch
   - Conv1D -> LSTM -> Dense -> Dropout -> Dense(10)

4. Comparison between Close and Predicted Close
   - Predicted Close -> μ, σ, Sharpe ratio

5. Sharpe ratio가 높은순으로 2종목씩 그룹화
   - 상위그룹부터 시장전망수익률 2%, 1.5%, 1%, 0.5% 부여
   - 상대적전망으로만 진행

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

1. **Feature Selection**
   - **Previous Returns**
   - **Previous Volume**
   - **Momentum related technical indicators**
     - **Moving Average / Bollinger Bands**

```python
from sklearn.base import BaseEstimator, TransformerMixin

# 종가이동평균선

class MA(BaseEstimator,TransformerMixin):
    def __init__(self):
        pass
    def fit(self,x,y):
        return self
    def transform(self,x):
        x['MA5'] = x['Close'].rolling(window=5).mean() #5일평균
        x['MA10'] = x['Close'].rolling(window=10).mean() #10일평균
        x['MA20'] = x['Close'].rolling(window=20).mean() #20일평균
        x['MA60'] = x['Close'].rolling(window=60).mean() #60일평균
        x['MA120'] = x['Close'].rolling(window=120).mean() #120일 평균
        return x

# 거래이동평균선 (Volume Adjusted Moving Average)

class VMA(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self,x,y=None):
        return self
    def transform(self,x):
        x['VMA5'] = x['Volume'].rolling(window=5).mean()
        x['VMA10'] = x['Volume'].rolling(window=10).mean()
        x['VMA20'] = x['Volume'].rolling(window=20).mean()
        x['VMA60'] = x['Volume'].rolling(window=60).mean()
        x['VMA120'] = x['Volume'].rolling(window=120).mean()
        return x
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

1. **Feature Selection**
   - **Previous Returns**
   - **Previous Volume**
   - **Momentum related technical indicators**
     - **Moving Average / Bollinger Bands**

```python
# 볼린저밴드생성
class bolinger(BaseEstimator,TransformerMixin):
    def __init__(self):
        pass
    def fit(self,x,y):
        return self
    def transform(self,x):
        x['std'] = x['Close'].rolling(window=20).std() # 볼린저 밴드 상한선, 하한선을 구하기 위한 20일 주가 표준편차
        x['20_Upper'] = x['MA20'] + 2 * x['std'] # 볼린저 밴드의 상한선 : 20일 이평선 값 + (20일동안의 주가 표준편차 값)*2
        x['20_Lower'] = x['MA20'] - 2 * x['std'] # 볼린저 밴드의 하한선 : 20일 이평선 값 - (20일동안의 주가 표준편차 값)*2
        x.drop('std',axis=1,inplace=True)
        x.dropna(inplace=True)
        return x
```

```python
from sklearn.pipeline import Pipeline

data_pipeline = Pipeline([
    ('MA',MA()),
    ('VMA',VMA()),
    ('Bolinger',bolinger()),
])
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

1. **Feature Selection**
   - **Previous Returns**
   - **Previous Volume**
   - **Momentum related technical indicators**
     - **Moving Average / Bollinger Bands**

날짜	Close	Volume	Return	MA5	MA10	MA20	MA60	MA120	VMA5	VMA10	VMA20	VMA60	VMA120	20_Upper	20_Lower
2017-06-28	47700	191450	-0.012422	47972.0	47330.0	46364.0	44506.666667	41783.500000	194851.8	224191.9	229793.60	2.420408e+05	2.381835e+05	48891.910725	43836.089275
2017-06-29	47940	166131	0.005031	47968.0	47556.0	46526.0	44606.000000	41882.166667	182254.8	221491.0	219431.05	2.420750e+05	2.387928e+05	49019.953952	44032.046048
2017-06-30	47540	237551	-0.008344	47952.0	47752.0	46669.0	44711.666667	41974.333333	191704.6	210439.2	221555.10	2.425341e+05	2.395461e+05	49042.342302	44295.657698
2017-07-03	47220	136111	-0.006731	47740.0	47818.0	46732.0	44808.000000	42066.500000	184715.6	201442.5	215871.90	2.416095e+05	2.393517e+05	49092.959662	44371.040338
2017-07-04	47000	159220	-0.004659	47480.0	47704.0	46785.0	44890.000000	42161.833333	178092.6	187274.5	216233.50	2.401780e+05	2.388506e+05	49118.531320	44451.468680
–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
2021-12-24	80500	12086380	0.007509	79000.0	78220.0	76890.0	73046.666667	75388.333333	13655888.6	12898439.9	16395581.25	1.523492e+07	1.614813e+07	81395.855839	72384.144161
2021-12-27	80200	10783368	-0.003727	79620.0	78560.0	77285.0	73148.333333	75390.000000	13559687.2	12472901.7	16100621.70	1.508265e+07	1.616504e+07	81470.325240	73099.674760
2021-12-28	80300	18226325	0.001247	80060.0	78890.0	77735.0	73266.666667	75389.166667	14355892.6	13197868.2	15493695.90	1.512303e+07	1.624751e+07	81057.190146	74412.809854
2021-12-29	78800	19794795	-0.018680	79940.0	79010.0	77955.0	73376.666667	75369.166667	14893673.2	14218853.8	15385692.85	1.505272e+07	1.631137e+07	80909.728592	75000.271408
2021-12-30	78300	14236700	-0.006345	79620.0	79060.0	78080.0	73493.333333	75348.333333	15025513.6	14442911.0	14914880.85	1.497404e+07	1.629674e+07	80857.048793	75302.951207

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

1. **Feature Selection**
   - **Previous Returns**
   - **Previous Volume**
   - **Momentum related technical indicators**
     - **Moving Average / Bollinger Bands**

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

2. **Feature Engineering**
   -Train / Test Spilt

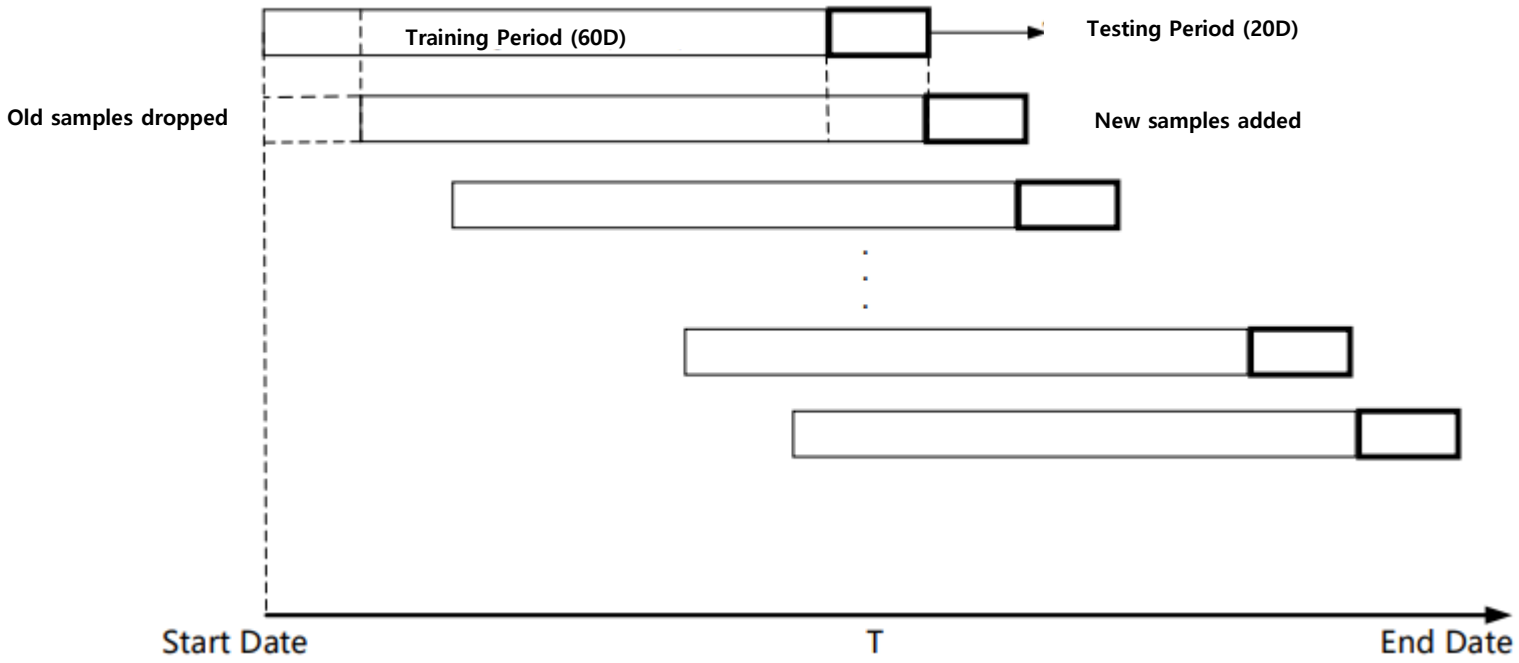
Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

2. Feature Engineering

-Rolling Window (Features; 60 days / target ; 20 days)

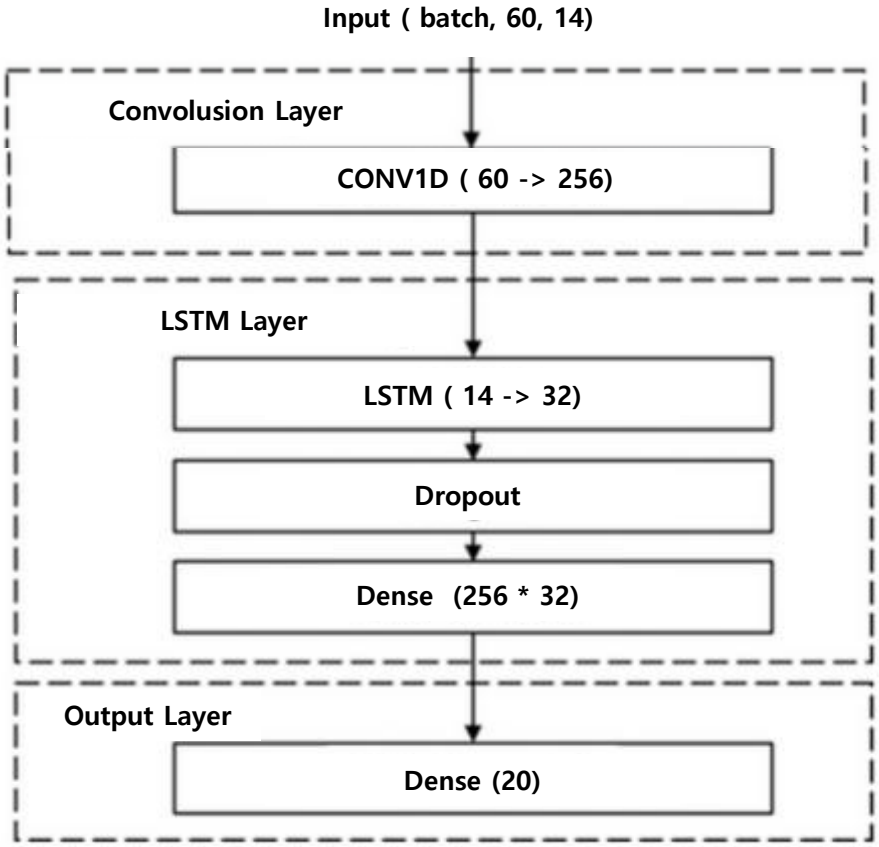


Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm



Introduction of Different Weighting Method

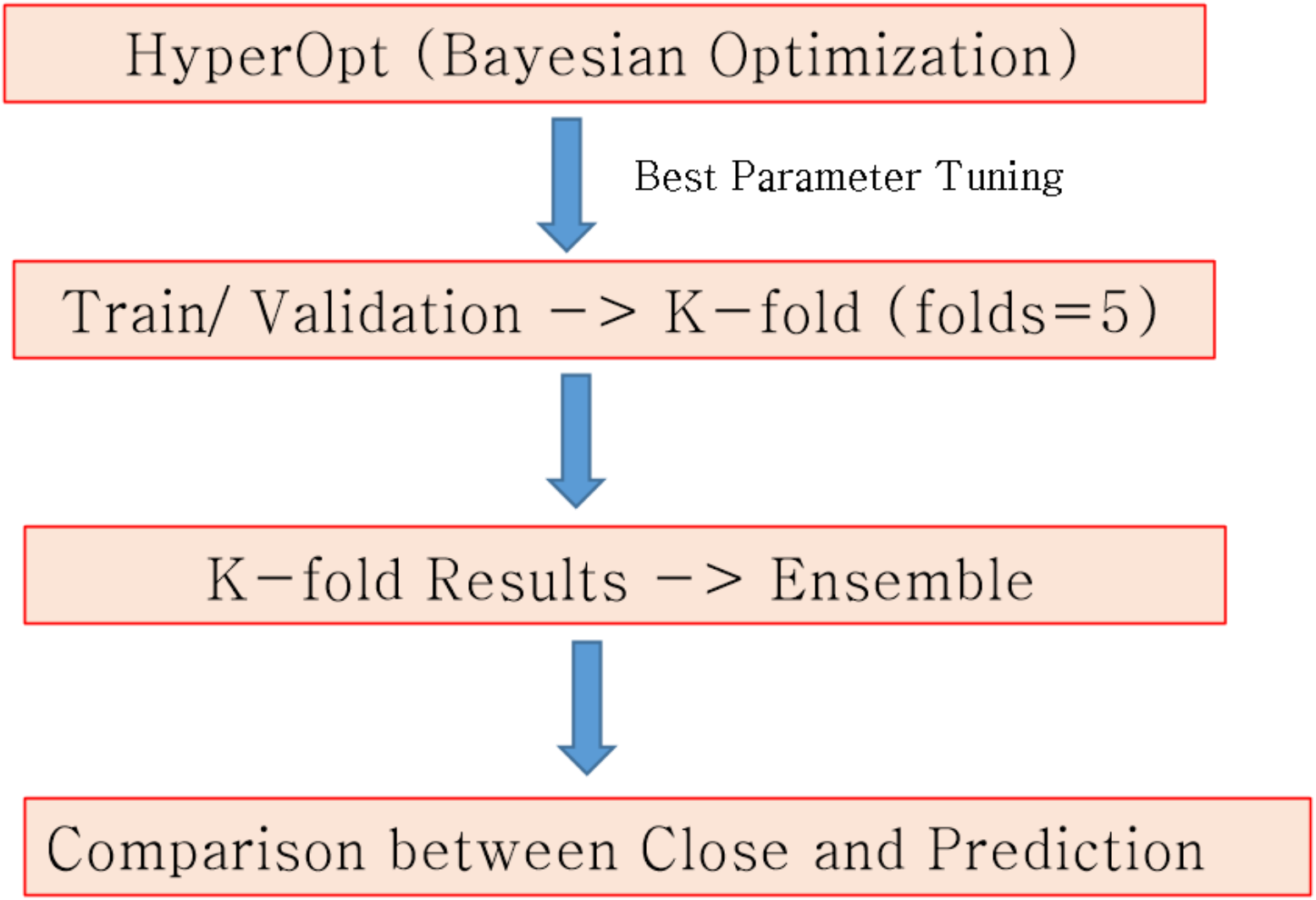
6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

```
Model(
  (con_layer): Sequential(
    (0): Conv1d(60, 128, kernel_size=(1,), stride=(1,))
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv1d(128, 256, kernel_size=(1,), stride=(1,))
    (4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
  )
  (lstm1): LSTM(14, 64, num_layers=4)
  (bnn1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tanh1): Tanh()
  (lstm2): LSTM(64, 128, num_layers=4)
  (bnn2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tanh2): Tanh()
  (lstm3): LSTM(128, 64, num_layers=4)
  (bnn3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tanh3): Tanh()
  (lstm4): LSTM(64, 32, num_layers=4)
  (bnn4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (tanh4): Tanh()
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_layer): Sequential(
    (0): Linear(in_features=8192, out_features=512, bias=True)
    (1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.25, inplace=False)
    (4): Linear(in_features=512, out_features=256, bias=True)
    (5): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.25, inplace=False)
    (8): Linear(in_features=256, out_features=128, bias=True)
    (9): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
    (11): Dropout(p=0.25, inplace=False)
    (12): Linear(in_features=128, out_features=10, bias=True)
    (13): ReLU()
  )
)
```

Conv1d+ Lstm


HyperOpt (Bayesian Optimization)
Best Parameter Tuning
Train/ Validation -> K-fold (folds=5)
K-fold Results -> Ensemble
Comparison between Close and Prediction

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

Train/ Validation -> K-fold (folds=5)

K-folds = 5 :
⇒ Train = 700 * 80%
⇒ Validation = 700 * 20%

Early_Stopping = 50
Epochs = 100
Batch_size = 20

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

Train/ Validation -> K-fold (folds=5)

```
model = Model(num_layers=int(best['num_layers']),
                            p=round(best['p'],5)
                            )

#model.apply(weight_init)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model.to(device)


kfold = KFold(n_splits=4, shuffle=False)

print('-'*50)

model_result = []
fold = 1
```

```
for fold, (train_idx, valid_idx) in enumerate(kfold.split(Dataset_train_list[0])): #dataset은 아직 batchsize 정하기이전!!

    print(f'FOLD {fold}')
    print('-'*50)

    # index 가져오기
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_idx) # 2568 중 4/5를 random으로 가져옴 (shuffle의 개념)
    valid_subsampler = torch.utils.data.SubsetRandomSampler(valid_idx) # 2568 중 1/5

    # trainloader, validloader 선언

    trainloader = torch.utils.data.DataLoader(Dataset_train, batch_size=20, sampler=train_subsampler)
    # 2568 중 4/5인 2054개를 minibatch로 10개씩 가져옴
    validloader = torch.utils.data.DataLoader(Dataset_train, batch_size=20, sampler=valid_subsampler)
    # 2568 중 1/5인 513개를 minibatch로 10개씩 가져옴

    list_lr = [num_layer] # vest lr로 튜닝

    for lr in list_lr:
        optimizer = optim.Adam(model.parameters(), lr=lr)
        mse_loss = F.mse_loss

        print(f'fold: {fold}, lr:{lr}')
        print('-'*50)

        epochs = 200
        train_loss = []
        valid_loss = []
        train_acc =[]
        valid_acc = []

        early_stopping = EarlyStopping(patience = 50, verbose = True, path='checkpoint.pt'+str(fold))


    for epoch in range(epochs):

        total_train_loss = 0
        total_train_acc = 0
        total_valid_loss = 0
        total_valid_acc = 0

  # model training
```

```
# model training

    model.train()
    total_train = 0

    tqdm_dataloader_train = tqdm(trainloader)
    for idx, (X_train, y_train) in enumerate(tqdm_dataloader_train, 0): #batchsize=10개씩 2568의 4/5 (trainset의 크기) 만큼반복: 207번 (1epoch당)

        X_train, y_train = X_train.to(device), y_train.to(device)
        optimizer.zero_grad()
        pred_y = model(X_train)

        loss = torch.sqrt(mse_loss(pred_y, y_train)) #RMSE로 변환
        total_train_loss += loss.item() # batch가 끝났을때 나오는 loss를 계속 더해줌 -> 총루 207번 (1epoch)이 다 끝나면 1epoch의 평균 loss를 구하기 위함

        loss.backward()
        optimizer.step()

        total_train = total_train +1

        #target_binary, pred_binary = indices_binarizer(y_train,pred_y)
        #acc = accuracy_score(target_binary,pred_binary)

        #total_train_acc += acc.item()


    #total_train_acc = total_train_acc / (idx +1)
    #train_acc.append(total_train_acc)  # 207번의 batch에서 나온 1epoch의 평균을 list안에 더해줌

    total_train_loss = total_train_loss / (idx +1)
    train_loss.append(total_train_loss)
```

```
# validation
    model.eval()
    total_valid = 0

    tqdm_dataloader_valid = tqdm(validloader)
    for idx, (X_valid, y_valid) in enumerate(tqdm_dataloader_valid, 0):

        X_valid, y_valid = X_valid.to(device), y_valid.to(device)
        pred = model(X_valid)


        loss_valid = torch.sqrt(mse_loss(pred, y_valid)) #RMSE로 변환
        total_valid_loss += loss_valid.item()


        total_valid = total_valid +1

        #target_binary, pred_binary = indices_binarizer(y_valid,pred)
        #acc = accuracy_score(target_binary,pred_binary)

        #total_valid_acc += acc.item()


    #total_valid_acc = total_valid_acc / (idx +1)
    #valid_acc.append(total_valid_acc)

    total_valid_loss = total_valid_loss / (idx +1)
    valid_loss.append(total_valid_loss)


    early_stopping(total_valid_loss, model)

    if early_stopping.early_stop:
        print("Early stopping")
        break


    #print(f'pred: {pred[-1]}')
    #print(f'y_valid: {y_valid[-1]}')
    if epoch % 5 == 0:
        #print('Epoch: {}/{}, Train Loss: {:.4f}, Val Loss: {:.4f}, Val Acc : {:.4f}'.format(epoch, epochs, total_train_loss, total_valid_loss, total_valid_acc))
        print('Epoch: {}/{}, Train Loss: {:.4f}, Val Loss: {:.4f}'.format(epoch, epochs, total_train_loss, total_valid_loss))
model.load_state_dict(torch.load('checkpoint.pt' + str(fold)))
model_result.append(model)
fold += 1
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

K-fold Results -> Ensemble

```
# test / ensemble

ensemble_pred = []
ensemble_y_test = []
for i in range(4):
    model.load_state_dict(torch.load(f'checkpoint.pt{i}'))
    model.eval()
    device = ('cuda' if torch.cuda.is_available() else 'cpu')
    with torch.no_grad():
        for i, (X_test, y_test) in enumerate(Dataloader_test):
            X_test, y_test = X_test.to(device).float(), y_test.to(device).float()
            pred = model(X_test).detach().cpu().numpy()
            y_test = y_test.detach().cpu().numpy()
            if len(pred) == 20:
                ensemble_pred.append(pred)
            if len(y_test) == 20:
                ensemble_y_test.append(y_test)


pred =np.mean(ensemble_pred, axis=0)
pred = torch.Tensor(pred)
y_test = np.mean(ensemble_y_test, axis=0)
y_test = torch.Tensor(y_test)
loss_test = torch.sqrt(mse_loss(pred, y_test)) #RMSE로 변환
total_valid = total_valid +1
```

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

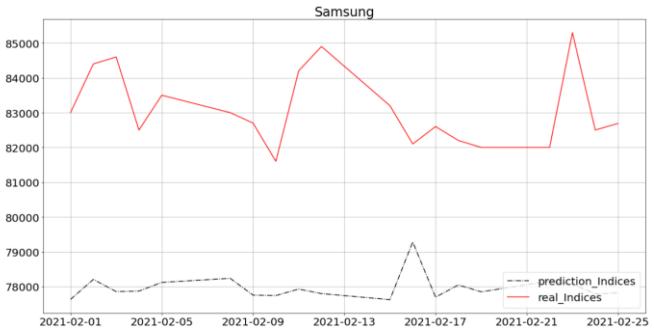
P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

Comparison between Close and Prediction

	Close	Predict	pred_return
2021-02-05	83500	78121.5402	0.003215
2021-02-08	83000	78237.9741	0.001490
2021-02-09	82700	77755.1544	-0.006171
2021-02-10	81600	77743.2876	-0.000153
2021-02-11	84200	77930.0610	0.002402
2021-02-12	84900	77799.3414	-0.001677
2021-02-15	83200	77625.6228	-0.002233
2021-02-16	82100	79282.9554	0.021350
2021-02-17	82600	77696.8434	-0.020006
2021-02-18	82200	78048.9534	0.004532
2021-02-19	82000	77849.8314	-0.002551
2021-02-22	82000	78158.8764	0.003970
2021-02-23	85300	78075.2412	-0.001070
2021-02-24	82500	77789.5932	-0.003659
2021-02-25	82690	77819.5110	0.000385

<1Q_1 삼성전자>



<1Q_1 삼성전자>

	pred_average_return	pred_standard_deviation	pred_sharpe_ratio
Samsung	-0.000234	0.167142	-0.060035
SkHynix	-0.000783	0.083871	-0.126179
Samsung Bio	0.017992	0.071901	0.113938
LG Chemical	0.051587	3.144590	0.013288
NAVER	0.017807	0.174786	0.045812
Hyundai Car	-0.028741	0.197122	-0.195520
Samsung SDI	0.005027	0.109611	-0.043546
KAKAO	0.029614	0.052652	0.376313

<1Q_1 종목별 predict of μ, σ, Sharpe ratio >

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

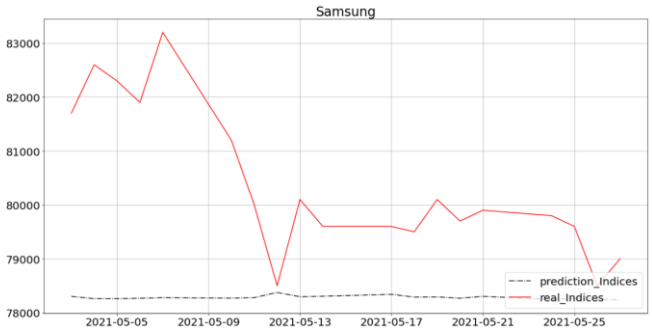
P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

Comparison between Close and Prediction

	Close	Predict	pred_return
2021-05-04	82600	78260.3118	-0.000539
2021-05-05	82300	78260.0940	-0.000003
2021-05-06	81900	78267.1956	0.000091
2021-05-07	83200	78278.1252	0.000140
2021-05-10	81200	78270.0930	-0.000103
2021-05-11	80000	78278.8446	0.000112
2021-05-12	78500	78375.3498	0.001233
2021-05-13	80100	78296.2884	-0.001009
2021-05-14	79600	78305.0466	0.000112
2021-05-17	79600	78340.0926	0.000448
2021-05-18	79500	78289.5168	-0.000646
2021-05-19	80100	78292.7442	0.000041
2021-05-20	79700	78268.3902	-0.000311
2021-05-21	79900	78303.6672	0.000451
2021-05-24	79800	78244.8018	-0.000752
2021-05-25	79600	78265.1430	0.000260
2021-05-26	78500	78243.8184	-0.000272
2021-05-27	79000	78239.0466	-0.000061

<2Q_1 삼성전자>



<2Q_1 삼성전자>

	pred_average_return	pred_standard_deviation	pred_sharpe_ratio
Samsung	-0.000899	0.010272	-1.197281
SkHynix	0.001619	0.368113	-0.026572
Samsung Bio	0.020187	0.206000	0.042653
LG Chemical	0.026909	0.052887	0.293242
NAVER	-0.004132	0.179210	-0.086667
Hyundai Car	-0.011325	0.205964	-0.110335
Samsung SDI	0.029100	0.090076	0.196505
KAKAO	0.027750	0.084491	0.193507

<2Q_1 종목별 predict of μ, σ, Sharpe ratio >

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

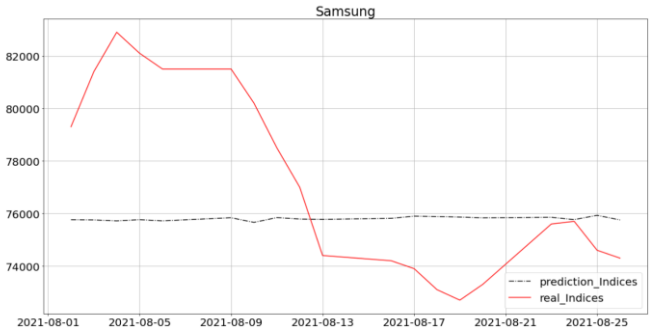
P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

3. Conv1d+ Lstm

Comparison between Close and Prediction

	Close	Predict	pred_return
2021-08-03	81400	75754.0608	-0.000111
2021-08-04	82900	75718.5920	-0.000468
2021-08-05	82100	75763.6864	0.000596
2021-08-06	81500	75719.4752	-0.000584
2021-08-09	81500	75838.0032	0.001565
2021-08-10	80200	75660.5760	-0.002340
2021-08-11	78500	75843.7632	0.002421
2021-08-12	77000	75787.8720	-0.000737
2021-08-13	74400	75776.0064	-0.000157
2021-08-16	74200	75814.6560	0.000510
2021-08-17	73900	75898.6432	0.001108
2021-08-18	73100	75883.9168	-0.000194
2021-08-19	72700	75864.2304	-0.000259
2021-08-20	73300	75833.0944	-0.000410
2021-08-23	75600	75854.9056	0.000288
2021-08-24	75700	75765.8688	-0.001174
2021-08-25	74600	75930.4128	0.002172
2021-08-26	74300	75753.5168	-0.002330

<3Q_1 삼성전자>



<3Q_1 삼성전자>

	pred_average_return	pred_standard_deviation	pred_sharpe_ratio
Samsung	-0.000115	0.025961	-0.551403
SkHynix	0.013273	0.036651	-0.025290
Samsung Bio	0.006094	0.048342	-0.167683
LG Chemical	0.027706	0.044759	0.301755
NAVER	0.014602	0.052584	0.007654
Hyundai Car	0.002737	0.146493	-0.078252
Samsung SDI	0.046000	0.427403	0.074402
KAKAO	0.042220	0.215750	0.129873

<3Q_1 종목별 predict of μ, σ, Sharpe ratio >

Introduction of Different Weighting Method

6. Weighting by E(R)(expected returns)

P(시장전망행렬) & Q(시장전망수익률) – Conv1d + Lstm

4. Predict Sharpe Ratio가 높은 순으로 2종목씩 그룹화

- 상위그룹부터 시장전망수익률 2%, 1.5%, 1%, 0.5% 부여
- 상대적전망으로만 진행

```
KAKAO          0.376313
Samsung Bio    0.113938
NAVER          0.045812
LG Chemical    0.013288
Samsung SDI   -0.043546
Samsung       -0.060035
SkHynix       -0.126179
Hyundai Car   -0.195520
Name: pred_sharpe_ratio, dtype: float64
```

<1Q_1 Sharpe Ratio 내림차순정렬 값>

1그룹 : 카카오, 삼성바이오로직스
2그룹 : 네이버, LG화학
3그룹 : 삼성SDI, 삼성전자
4그룹 : SK하이닉스, 현대차

```
LG Chemical    0.293242
Samsung SDI    0.196505
KAKAO          0.193507
Samsung Bio    0.042653
SkHynix       -0.026572
NAVER         -0.086667
Hyundai Car   -0.110335
Samsung       -1.197281
Name: pred_sharpe_ratio, dtype: float64
```

<2Q_1 Sharpe Ratio 내림차순정렬 값>

1그룹 : LG화학, 삼성SDI,
2그룹 : 카카오, 삼성바이오로직스
3그룹 : SK하이닉스, 네이버
4그룹 : 현대차, 삼성전자

Results

Final Composition of Stocks

Equal Weighting

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%

MVO Maximum Sharpe Ratio Weighting

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	0	0	0	0	0	0	0	1

Market Cap Weighting

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	55.75%	10.22%	5.81%	7.32%	5.62%	5.60%	5.48%	4.20%

Float Adjusted Weighting

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	15.25%	13.61%	5.41%	12.17%	15.83%	12.30%	13.11%	12.33%

Implied Weighting

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	18.74%	12.47%	6.80%	11.79%	15.43%	10.57%	12.11%	12.09%

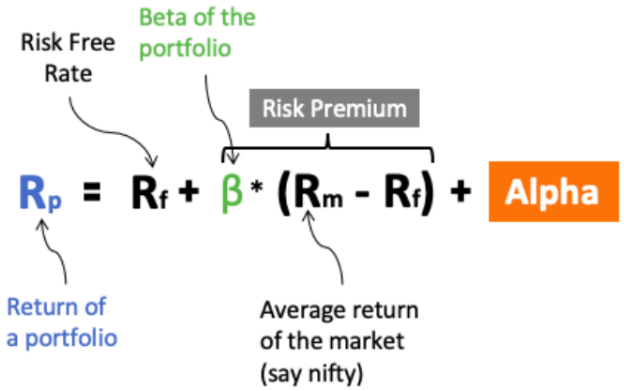
Implied Weighting with adjusted views

	Samsung	SkHynix	Samsung Bio	LG Chemical	NAVER	Hyundai Car	Samsung SDI	KAKAO
Weight	26.30%	3.04%	16.83%	7.15%	11.28%	0.33%	14.00%	21.07%

Results

Beta, Alpha

$$\beta_p = \frac{Cov(r_p, r_b)}{Var(r_b)}$$



출처: https://getmoneyrich.com/alpha-and-beta/

Results

Beta, Alpha

Benchmark: KOSPI200

```python
import numpy as np
from statsmodels import regression
import statsmodels.api as sm
import matplotlib.pyplot as plt
import math
import pandas as pd
import pandas_datareader as pdr

# 포트폴리오 일일 수익률

def get_daily_returns_of_port (returns = returns_1Q, weights= initial_weights):


    daily_return= returns*weights


    daily_return= daily_return.iloc[:, 0]+daily_return.iloc[:, 1]+ \
    daily_return.iloc[:, 2]+daily_return.iloc[:, 3]+ \
    daily_return.iloc[:, 4]+daily_return.iloc[:, 5]+ \
    daily_return.iloc[:, 6]+daily_return.iloc[:, 7]

    return daily_return

daily_return_same_1Q =get_daily_returns_of_port(returns = returns_1Q, weights= initial_weights)
daily_return_ef_1Q = get_daily_returns_of_port(returns = returns_1Q, weights= Ef_weights_1Q)
daily_return_cap_1Q = get_daily_returns_of_port(returns = returns_1Q, weights= cap_weights_1Q)
daily_return_float_adj_1Q = get_daily_returns_of_port(returns = returns_1Q, weights= float_adj_weights_1Q)
daily_return_pi_1Q = get_daily_returns_of_port(returns = returns_1Q, weights= pi_weights_1Q)
daily_return_pi_adj_1Q = get_daily_returns_of_port(returns = returns_1Q, weights= pi_adj_weights_1Q)
```

```python
def get_alpha_beta (KOSPI_Close= KOSPI_Close_1Q_2,
                    daily_return = daily_return_same_1Q_2,
                    Port_Annual_return = Same_Ann_Return_1Q_2,
                   Rf = Rf_1Q):


    #KOSPI200 일일 지수변화율
    KOSPI_return = KOSPI_Close.pct_change()

    #KOSPI200 지수 1년평균 지수변화율 (일일 지수변화율 평균 * 250(영업일))
    KOSPI_Annual_return = KOSPI_return.mean() *250


    #KOSPI200 지수변화율 분산
    KOSPI_Var = np.var(KOSPI_return)

    #위험회피계수 (Rp - Rf)/ VARp
    Risk_Aversion = (KOSPI_Annual_return -Rf)/ KOSPI_Var

    #KOSPI200과 Portfolio 공분산 구하기 위해 dataframe 합치기

    KOSPI_and_Port = pd.concat([KOSPI_return, daily_return], axis=1)
    KOSPI_and_Port.columns=['KOSPI per_change', 'Portfolio per_change']

    #KOSPI200과 Portfolio 공분산
    Cov_KOSPI_and_Port = KOSPI_and_Port.cov().iloc[0,1]

    #Beta = 공분산/market return의 분산
    Beta = (Cov_KOSPI_and_Port/KOSPI_Var)

    #Alpha = Rp - [Rf + Beta[Rm - Rf]] -수수료 (=0 가정)
    # 무위험이자율(Rf) - 3년국채 수익률 : 1.37%
    Alpha =  Port_Annual_return-(Rf + Beta*(KOSPI_Annual_return - Rf))

    return Alpha, Beta , #Risk_Aversion
```

Results

Mean, Standard deviation, Sharpe Ratio

```
def Mu_Std_Sharpe (i = 1, returns = returns_1Q, W= initial_weights):

    print('<' + str(i) + 'Q>')

    Cov = returns.cov()
    Cov_annual = Cov * 250
    #print('Annual Coveriance Matrix of' + str(i) + 'Q :', Cov_annual)

    # 포트폴리오 분산의 기댓값 = WT * (공분산 행렬) * W

    Var = np.dot(W.T, np.dot(Cov_annual, W))
    print('Portfolio Variance of ' + str(i) + 'Q :', Var)
    Std = np.sqrt(Var)
    print('Portfolio Standard Deviation of ' +str(i) + 'Q :', Std)

    Ann_Return = np.sum(returns.mean()*W) * 250
    print('Portfolio Annual Return of ' +str(i) + 'Q :', Ann_Return)

    # 무위험수익률 - 3년국채 수익률 : 1.37%

    print('2021 Risk Free Rate : 1.37%')

    Sharpe = (Ann_Return - 0.0137) / Std

    print('Sharpe Ratio of ' + str(i) + 'Q :', Sharpe)

    return  Ann_Return, Std, Sharpe
```

Results

Min_Return, Max_Return

```
# 최저수익률

min_same_1Q = daily_return_same_1Q.min()
min_ef_1Q = daily_return_ef_1Q.min()
min_cap_1Q = daily_return_cap_1Q.min()
min_float_adj_1Q = daily_return_float_adj_1Q.min()
min_pi_1Q = daily_return_pi_1Q.min()
min_pi_adj_1Q = daily_return_pi_adj_1Q.min()


#최대수익률

max_same_1Q = daily_return_same_1Q.max()
max_ef_1Q = daily_return_ef_1Q.max()
max_cap_1Q = daily_return_cap_1Q.max()
max_float_adj_1Q = daily_return_float_adj_1Q.max()
max_pi_1Q = daily_return_pi_1Q.max()
max_pi_adj_1Q = daily_return_pi_adj_1Q.max()
```

Results

Overall

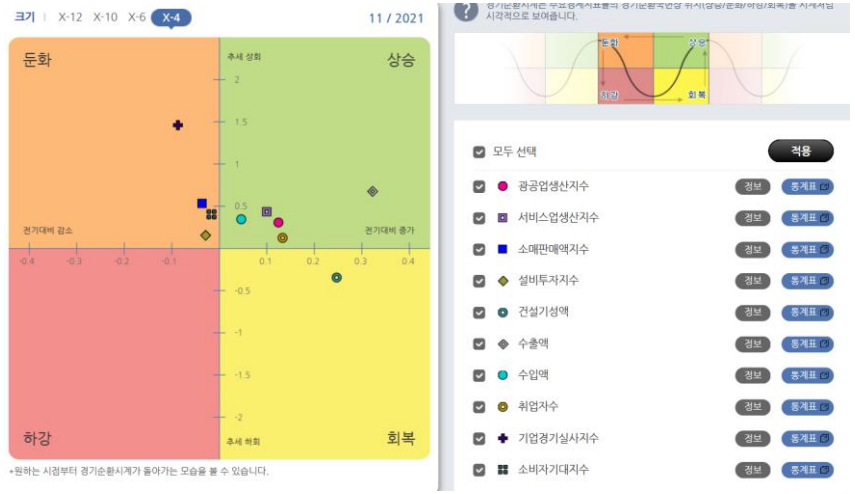
통계청 경기순환시계

출처 : 통계청

<1Q_2 경기국면> : '상승'

<2Q_2 경기국면> : '경기후퇴'

<3Q_2 경기국면> : '경기후퇴 & 둔화'



<4Q_2 경기국면> : '상승 & 둔화'

Results

Overall

<1Q_2 경기국면> : '상승'

	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	31.8429	147.0250	11.5899	32.2902	40.2400	18.0397
표준편차(%)	32.5655	35.8616	25.6714	28.3438	32.6564	28.8402
샤프지수	0.9477	4.0725	0.4133	1.1047	1.2022	0.5915
최저수익률(%)	-3.3431	-2.7724	-3.4880	-3.3250	-3.3004	-3.1125
최대수익률(%)	3.9890	6.1818	4.2626	4.0069	3.9682	3.9430
알파(%)	53.2808	160.7449	31.9140	54.3179	61.8926	36.8666
베타	1.2846	0.8423	1.2208	1.3184	1.2969	1.1350

<NLP방식 >

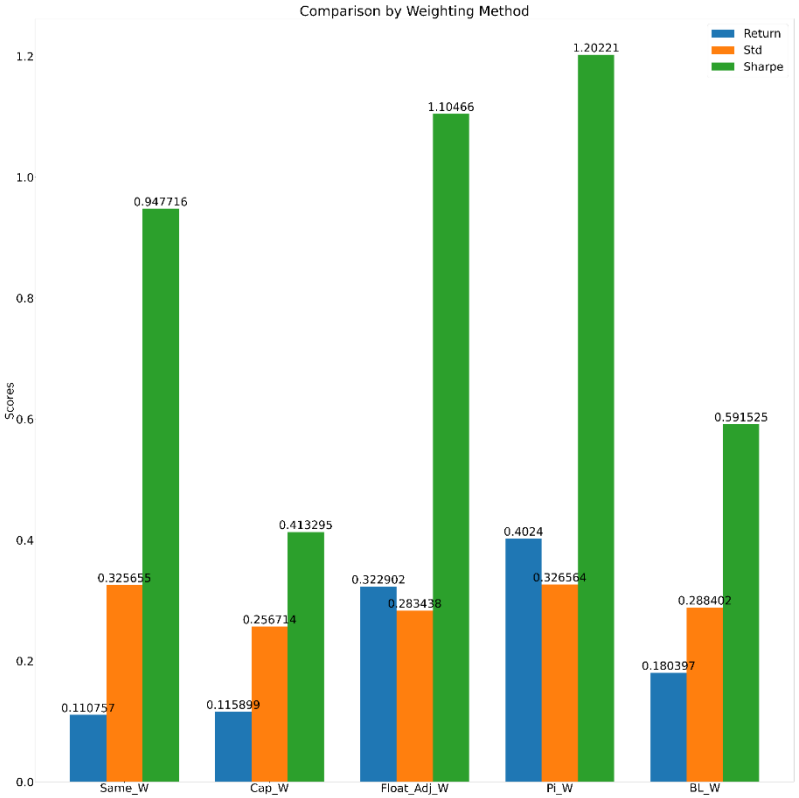
	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	31.8429	147.0250	11.5899	32.2902	40.2400	5.9773
표준편차(%)	32.5655	35.8616	25.6714	28.3438	32.6564	29.2743
샤프지수	0.9477	4.0725	0.4133	1.1047	1.2022	0.1707
최저수익률(%)	-3.3431	-2.7724	-3.4880	-3.3250	-3.3004	-3.3272
최대수익률(%)	3.9890	6.1818	4.2626	4.0069	3.9682	3.9230
알파(%)	53.2808	160.7449	31.9140	54.3179	61.8926	25.1678
베타	1.2846	0.8423	1.2208	1.3184	1.2969	1.1558

<CONV1D + LSTM 방식 >

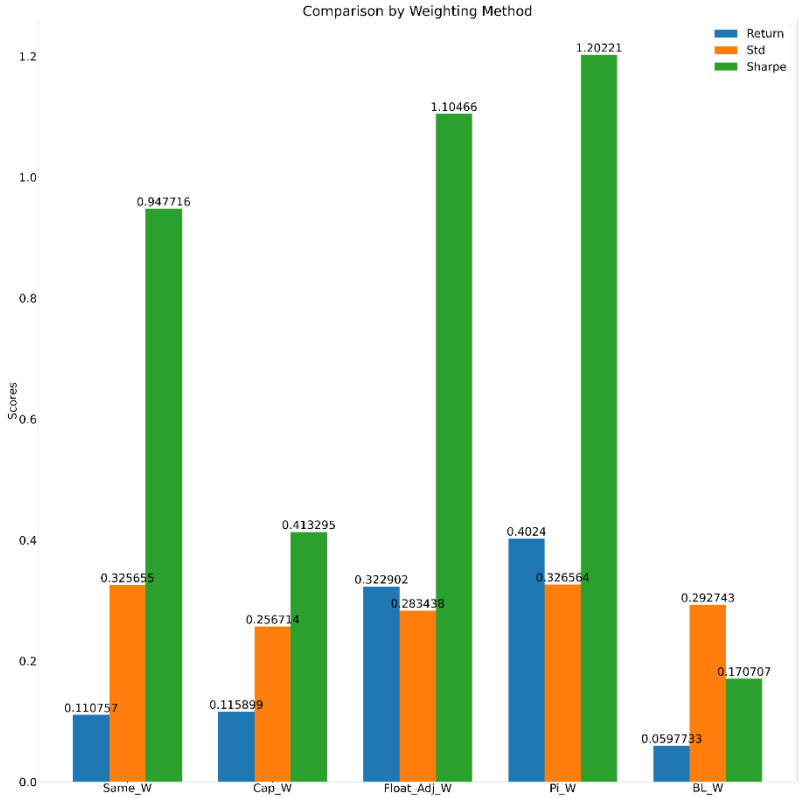
Results

Overall

<1Q_2 경기국면> : '상승'



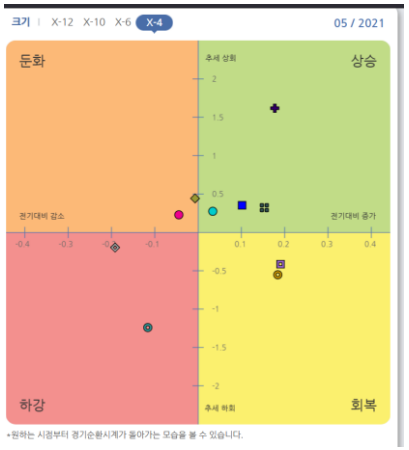
<NLP방식 >



<CONV1D + LSTM 방식 >

Results

Overall



<2Q_2 경기국면> : '경기후퇴'

	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	11.0757	-1.8573	17.1010	130.5964	28.5163	55.6049
표준편차(%)	17.6640	24.7531	15.0795	24.0692	20.2260	19.2108
샤프지수	0.5625	-0.1211	1.0691	5.3852	1.3614	2.8435
최저수익률(%)	-1.9568	-2.8325	-2.3573	-3.0737	-2.2173	-2.3777
최대수익률(%)	2.2028	3.5459	2.0338	2.6919	2.4521	2.3886
알파(%)	-17.5592	-26.2962	-11.7338	54.3179	2.2346	33.4462
베타	1.1155	0.9453	1.1236	0.7953	1.0200	0.8528

<NLP방식 >

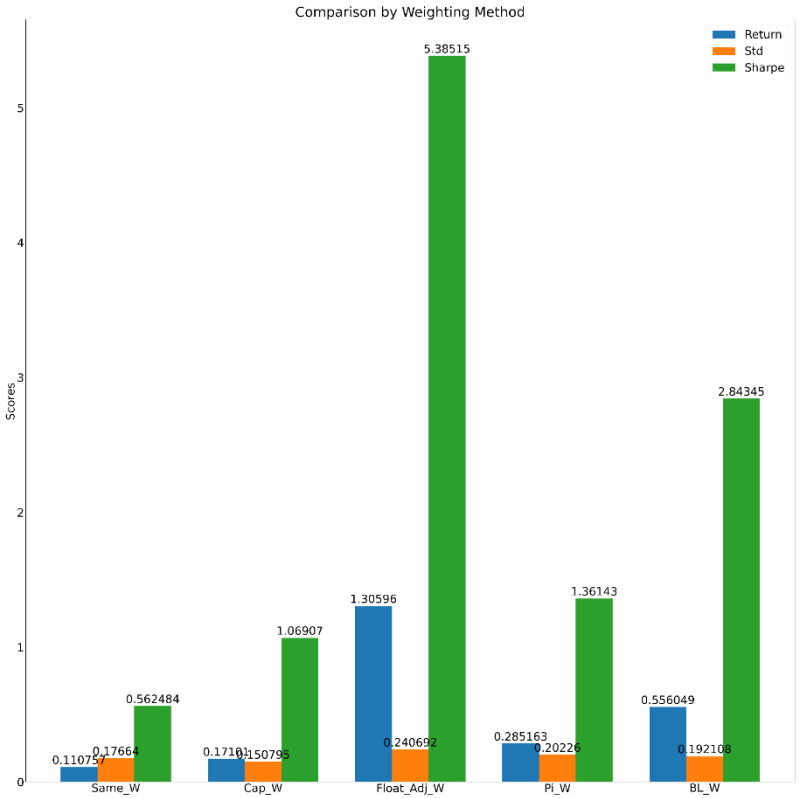
	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	11.0757	-1.8573	17.1010	130.5964	59.1698	44.4654
표준편차(%)	17.6640	24.7531	15.0795	24.0692	29.3970	35.3523
샤프지수	0.5625	-0.1211	1.0691	5.3852	1.9794	1.2301
최저수익률(%)	-1.9568	-2.8325	-2.3573	-3.0737	-3.7618	-4.5672
최대수익률(%)	2.2028	3.5459	2.0338	2.6919	5.1717	5.9613
알파(%)	-17.5592	-26.2962	-11.7338	54.3179	35.2052	22.1594
베타	1.1155	0.9453	1.1236	0.7953	0.9260	0.8587

<CONV1D + LSTM 방식 >

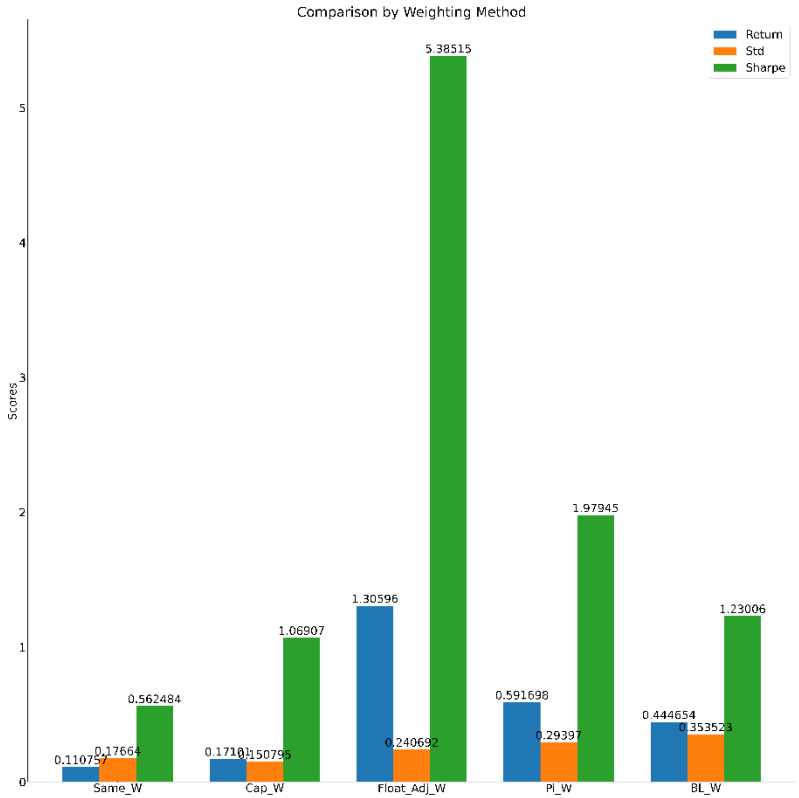
Results

Overall

<2Q_2 경기국면> : '상승' -> '둔화 '



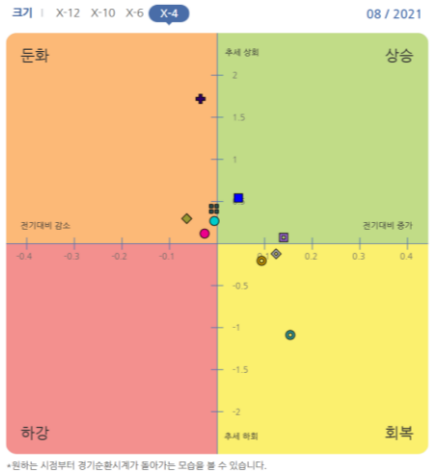
<NLP방식 >



<CONV1D + LSTM 방식 >

Results

Overall



<3Q_2 경기국면> : '경기후퇴' &'둔화'

	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	-18.9523	55.0477	-31.9503	-32.7755	68.4066	83.1915
표준편차(%)	10.2656	24.2650	10.6906	13.0833	29.2787	33.8765
샤프지수	-1.9845	2.2101	-3.0803	-2.5800	2.3029	2.4268
최저수익률(%)	-1.2359	-3.6173	-1.6575	-1.3480	-4.5463	-5.2528
최대수익률(%)	1.2968	3.0838	1.5017	1.1276	3.6416	4.1804
알파(%)	0.7365	35.9053	9.7743	-4.5303	30.0321	36.3776
베타	0.4383	-0.3680	0.8959	0.6160	-0.7673	-0.9426

<NLP방식 >

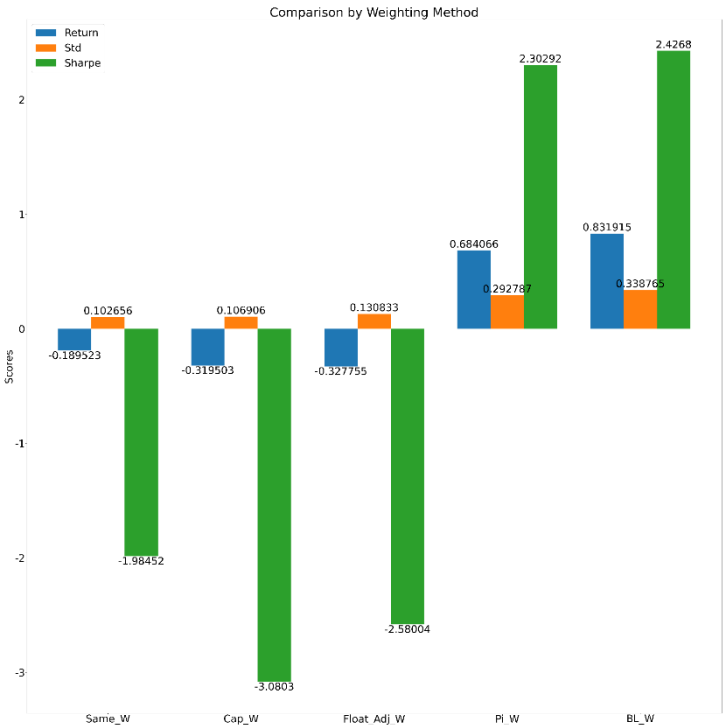
	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	-18.9523	55.0477	-31.9503	-32.7755	68.4066	79.1832
표준편차(%)	10.2656	24.2650	10.6906	13.0833	29.2787	32.6844
샤프지수	-1.9845	2.2101	-3.0803	-2.5800	2.3029	2.3927
최저수익률(%)	-1.2359	-3.6173	-1.6575	-1.3480	-4.5463	-5.0303
최대수익률(%)	1.2968	3.0838	1.5017	1.1276	3.6416	4.0332
알파(%)	0.7365	35.9053	9.7743	-4.5303	30.0321	35.0607
베타	0.4383	-0.3680	0.8959	0.6160	-0.7673	-0.8867

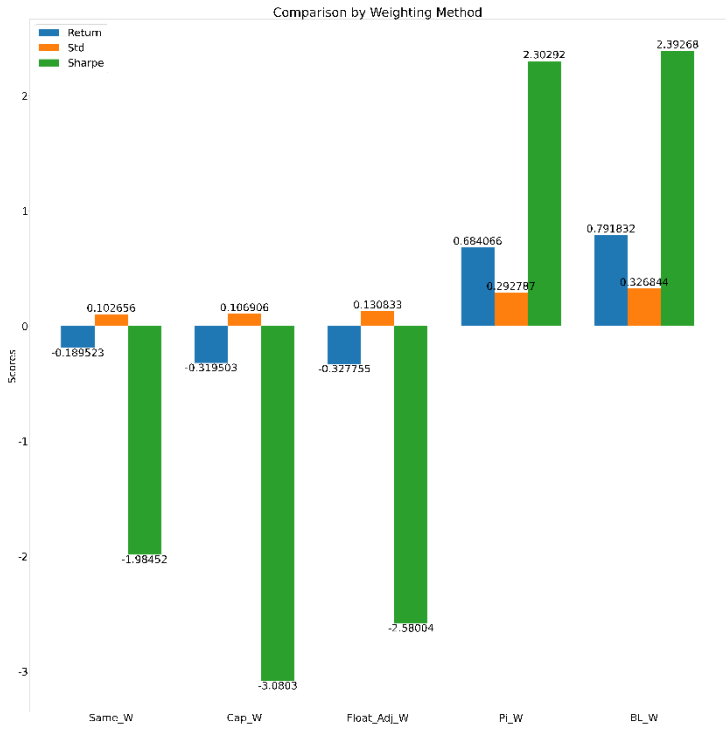
<CONV1D + LSTM 방식 >

Results

Overall

<3Q_2 경기국면> : '둔화'



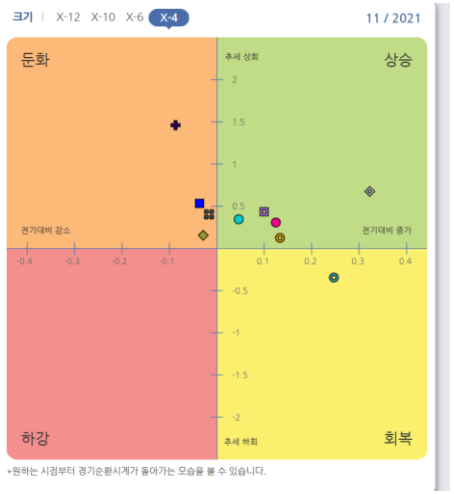


<NLP방식 >

<CONV1D + LSTM 방식 >

Results

Overall



<4Q_2 경기국면> : '둔화'

	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	-33.3572	-119.4542	-15.0752	35.5851	-17.4362	11.8640
표준편차(%)	17.8783	24.3815	19.7952	22.9360	20.8508	23.0345
샤프지수	-1.9687	-4.9749	-0.8111	1.5088	-0.8832	0.4725
최저수익률(%)	-1.7745	-3.3372	-1.7049	-1.8708	-2.0360	-3.1643
최대수익률(%)	2.1263	2.1997	3.5811	2.5565	3.5363	3.6036
알파(%)	16.5350	-57.9786	42.9998	88.8122	44.8155	42.2233
베타	0.9931	1.2155	1.1502	1.0571	1.2304	0.6181

<NLP방식 >

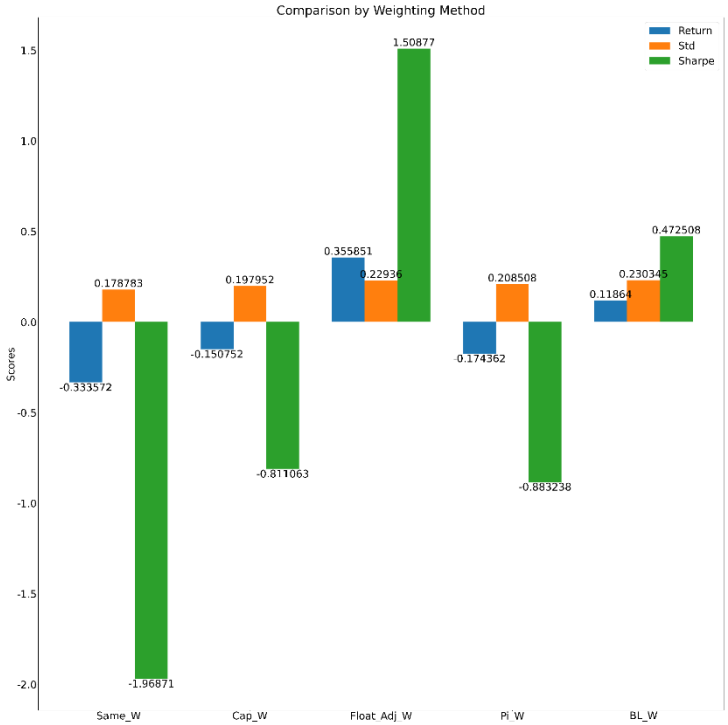
	Same Weighted	MVO weighted	Cap weighted	Float Adj weighted	Pi weighted	Pi_adj weighted
평균수익률(%)	-33.3572	-119.4542	-15.0752	35.5851	-17.4362	-3.6304
표준편차(%)	17.8783	24.3815	19.7952	22.9360	20.8508	24.7602
샤프지수	-1.9687	-4.9749	-0.8111	1.5088	-0.8832	-0.1862
최저수익률(%)	-1.7745	-3.3372	-1.7049	-1.8708	-2.0360	-3.4941
최대수익률(%)	2.1263	2.1997	3.5811	2.5565	3.5363	3.8023
알파(%)	16.5350	-57.9786	42.9998	88.8122	44.8155	24.0704
베타	0.9931	1.2155	1.1502	1.0571	1.2304	0.5671

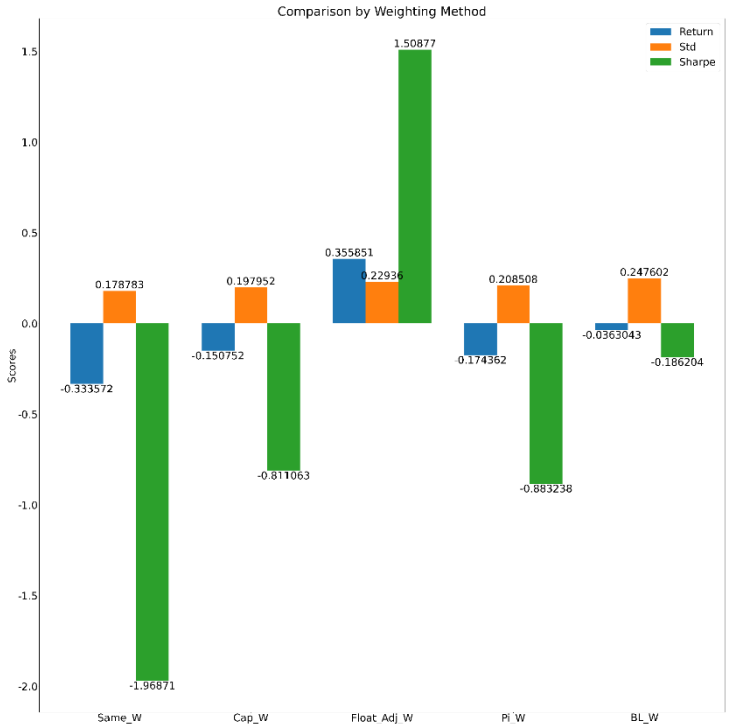
<CONV1D + LSTM 방식 >

Results

Overall

<4Q_2 경기국면> : '상승 & 둔화'





<NLP방식 >

<CONV1D + LSTM 방식 >

Results

Overall

<결론>

1. 경기상승국면에서는 MVO weighted, same_weighted 방식이 유리할 수 있다.
(경기의 전반적 상승으로 인해, 포트폴리오 분산효과를 누리기보다
과거수익률이 좋았던 종목에 집중투자하여 모멘텀 상승효과를 얻을 수 있음)

2. 경기둔화 & 하방국면에서는 float_adjusted_weighted, black_litterman model이 유리할 수 있다.
(단, 투자자전망의 확실성 및 정확성 필요)

3. 경기둔화 & 하방 국면에서 MVO 모델의 경우 코너해 문제로 분산투자로 인한 위험 방어에 취약할 수 있다.

<한계>

1. 투자자 전망 구하는 방식의 정확성 -> 향후 다양한 방식으로 투자자 전망 구해보기
2. 분기별로 1달만 분석한 점 -> DL의 경우 시간 소요가 많이 됨
3. 경기국면 파악의 정확성 -> 더 정확한 경기국면 파악방법모색