Financial Engineering: Project 1

Jungwoo Yang

April 30, 2021

1 Data Collection & Data Processing

- Data는 python의 yfinance library를 이용하여 다운받았다. (`https://pypi.org/project/yfinance/`)
- Stock은 S&P500에 있는 모든 기업이다. 2000년 1월 1일 이후 주가 데이터를 일단위로 받았으며 S&P500 리스트는 다음의 링크를 스크롤링 해서 받은 후에 yfinance에서 데이터를 받았다: `https://en.wikipedia.org/wiki/List_of_S%26P_500_companies`.
- S&P500으로 설정한 이유는 미국에서는 높은 수익률을 올리기 위해 작은 규모의 기업에 투자 하지 않고 큰 규모의 회사들에 투자할 경우에도 충분히 안정적이고 좋은 수익률을 보여주기 때문이다.
- S&P500 기업중에는 GOOG, GOOGL과 같이 class가 다른 주식이 둘다 있는 경우가 있는데 그 경우에는 class A 만 남기고 나머지는 제외하였다. (총 5개의 기업)
- S&P500 기업 전체를 이용하지 않고 market capitalization 기준 상위 기업들만 투자 대상으로 했다. 2021년 4월 26일 기준 market cap을 사용했는데 투자 당시 market cap으로 계산을 해야 look ahead bias가 없겠지만 이전 market cap을 구하는 것이 용이하지 않아서 편의상 현재의 market cap으로 설정했다.
- 무위험자산의 경우 많이 사용하는 3개월 만기 미국 국채를 사용했다(3-Month Treasury Bill: Secondary Market Rate (TB3MS)).
- 위험을 선택하기 위해서 VIX를 이용했다.
- TBILL과 VIX 데이터를 사용하기 위해서 pandas datareader library를 이용해 Federal Reserve Economic Data(FRED)에서 다운 받았다.
- Stock data의 데이터 간격을 일이 아닌 달로 설정했다. 일로 설정후에 달로 계산이 가능하겠지만 좀 더 장기간의 투자를 위해 기간을 달로 설정하였다. 한달의 가격은 한달 중 거래가 일어난 맨 마지막 날의 closing price로 설정했다.
- 데이터를 받기 위해 사용한 코드는 US_data_downloader.ipynb를 확인하면 된다.
- 위 data는 './data' 폴더안에 있으며 S&P500 주식 데이터의 경우 sp500_20000101.csv, market cap list는 mc_list_2021_04_26.csv, VIX 데이터는 VIX_20170101.csv, TBILL 데이터는 TBILL.csv를 확인하면 된다.

- 주가는 그대로 사용하지 않고 아래의 로그 변환을 사용했다.

$$R_{t+1} = \log(P_{t+1}) - \log(P_t)$$
$$R_t := t\text{시점의 log 변환 수익률}$$
$$P_t := t\text{시점의 주가}$$

- 주식 데이터의 기간은 2018년 1월 1일부터 2021월 1월 1일까지 총 3년치의 데이터이다. 달별 데이터이니 각 주식마다 35개의 데이터가 존재한다. (수익률에 대한 log 변환을 거쳤기 때문에 첫번째 데이터는 없다.) 안정적인 covariance matrix 계산을 위해 로그 수익률이 2020월 1월 1일 이후 한번이라도 없는 경우 제외하였다.
- TBILL과 VIX도 마찬가지로 같은 기간의 수익률을 평균을 내서 사용하였다.

2 Dataset

2.1 sp500_20000101.csv

	Close										...	Volume		
	A	AAL	AAP	AAPL	ABBV	ABC	ABMD	ABT	ACN	ADBE	...	XEL	XLNX	XOM
1999-12-31	47.743351	NaN	NaN	0.786428	NaN	2.913709	18.375000	9.846459	NaN	16.693562	...	262900.0	1876700.0	275
2000-01-03	44.462700	NaN	NaN	0.856227	NaN	2.985652	18.250000	9.490563	NaN	16.274673	...	2738600.0	7698700.0	1345
2000-01-04	41.066231	NaN	NaN	0.784038	NaN	2.781813	17.812500	9.219403	NaN	14.909401	...	425200.0	7399600.0	1451
2000-01-05	38.518894	NaN	NaN	0.795511	NaN	2.997643	18.000000	9.202456	NaN	15.204173	...	500200.0	6607300.0	1748
2000-01-06	37.052246	NaN	NaN	0.726669	NaN	3.225464	18.031250	9.524458	NaN	15.328290	...	344100.0	8556600.0	1946
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2021-03-24	120.656403	21.809999	181.729996	120.089996	101.821983	115.370003	294.209991	117.588249	266.724518	451.510010	...	3371400.0	2162000.0	2978
2021-03-25	121.714798	22.770000	185.649994	120.589996	102.632133	117.339996	294.140015	118.614487	267.781250	450.989990	...	3475400.0	1479400.0	3149
2021-03-26	125.449112	22.930000	187.320007	121.209999	104.706917	118.730003	301.399994	121.623436	279.903717	469.089996	...	2644600.0	2360300.0	3422
2021-03-29	125.229446	22.910000	185.059998	121.389999	105.447906	119.050003	305.769989	121.782860	278.677551	469.320007	...	2379000.0	1792200.0	2114
2021-03-30	124.650322	24.120001	186.070007	119.900002	105.507179	119.059998	309.880005	119.311928	277.690582	465.459991	...	2284400.0	1594700.0	2030

Figure 1: sp500_20000101.csv

- 1999-12-31 부터 데이터가 시작되고 2021-03-30까지의 데이터가 있다.
- columns는 multilevel로 되어있고 level 1에는 Close, Dividends, High, Low, Open, Stock Splits, Volume으로 구성되어 있다. Level 2에는 기업들의 list로 구성되어있다.

2.2 mc_list_2021_04_26.csv

- 2021-04-26에 수집한 데이터이고 첫번째 열은 Market Capitalization이고 두번째 열은 기업의 ticker이다.

2.3 VIX_20170101.csv

- 2017-01-02 부터 데이터가 시작되고 날짜에 해당하는 VIX가 있다.

0	5105858560	NOV
1	5598430208	HFC
2	5707326976	PRGO
3	5886745088	UNM
4	6478351872	APA
...	...	...
495	856555257856	FB
496	1540812832768	GOOGL
497	1694851792896	AMZN
498	1965349535744	MSFT
499	2241209827328	AAPL

Figure 2: mc_list_2021_04_26.csv

	VIXCLS
DATE	
2017-01-02	NaN
2017-01-03	12.85
2017-01-04	11.85
2017-01-05	11.67
2017-01-06	11.32
...	...
2021-03-25	19.81
2021-03-26	18.86
2021-03-29	20.74
2021-03-30	19.61
2021-03-31	19.40

Figure 3: VIX_20170101.csv

2.4 TBILL.csv

- 2017-01-01 부터 데이터가 시작하고 월마다 treasury bond rate가 있다.

DATE	TB3MS
2017-01-01	0.51
2017-02-01	0.52
2017-03-01	0.74
2017-04-01	0.80
2017-05-01	0.89
...	...
2020-11-01	0.09
2020-12-01	0.09
2021-01-01	0.08
2021-02-01	0.04
2021-03-01	0.03

Figure 4: TBILL.csv

3 Specifications

- python3를 이용해 프로그램을 구성했다.
- project는 아래와 같은 hierarchy로 구성되어 있다.

```
FE_project_1
├── 양정우hw101.py
├── US_data_downloader.ipynb
├── with_short_no_gurobi.png
├── with_short_with_gurobi.png
├── no_short.png
└── data
    ├── sp500_20000101.csv
    ├── mc_list_2021_04_26.csv
    ├── TBILL.csv
    └── VIX_20170101.csv
```

4 How to run

- `python 양정우hw101.py --help`를 root folder에서 command 창에 칠 경우 다음과 같이 어떠한 argument가 가능하지 알려준다.

```
C:\Users\jungwoo\Desktop\FE_project_1>python 양정우hw101.py --help
usage: 양정우hw101.py [-h] [--no_short] [--no_gurobi] [-n N]
```

```
Calculate Modern Portfolio Theory

optional arguments:
  -h, --help    show this help message and exit
  --no_short    limit short
  --no_gurobi   do not use gurobi
  -n N          Number of companies (default: 30)
```

- `--no_short`: short를 제한하는 argument로 각각의 stock의 비율을 0이상으로 제한한다. (no_gurobi와 같이 사용할 수는 없다)
- `--no_gurobi`: gurobi를 사용하지 않고 short가 가능한 상황에서 stock의 비율을 알려준다.
- `-n`: output하는 기업의 개수를 바꿀 수 있다.

gurobi 없이 short가 가능할 경우:

```
python 양정우hw101.py --no_gurobi
```

gurobi로 short가 가능할 경우:

```
python 양정우hw101.py
```

short가 제한될 경우:

```
python 양정우hw101.py --no_short
```

5 Code Description

5.1 main

```
args = parse_arguments()
log_df, VIX, rf_rate = load_dataframes(args)
cov = log_df.cov().to_numpy()
mu = log_df.mean().to_numpy()
if args.no_short: #no short with gurobi
    A = VIX*0.1
    rf_ratio, theta_P, mu_P, sig_P = \
        no_short(log_df, A, rf_rate, mu, cov)
elif args.no_gurobi: #short exists and no gurobi
    A = VIX*15
    rf_ratio, theta_P, mu_P, sig_P = \
        with_short_no_gurobi(log_df, A, rf_rate, mu, cov)
else: #short exists and with gurobi
    A = VIX*15
    rf_ratio, theta_P, mu_P, sig_P = \
        with_short_with_gurobi(log_df, A, rf_rate, mu, cov)
print("Number of top market cap companies:", args.n)
print("Data start:", start)
print("Data end(excluding):", end)
print("VIX:", VIX)
print("Risk free rate:", rf_rate)
print("Expected portfolio return:", mu_P)
print("Expected portfolio standard deviation:", sig_P)
```

```
print("Percentage of portfolio to invest in risk free asset:",
    rf_ratio)
print("Company list:")
print(log_df.columns.to_numpy())
print("Percentage of rest of the portfolio to invest:")
print(theta_P)
```

Listing 1: Main

- 프로그램이 시작할때 제일 처음 시작되는 부분이다.
- argument를 읽고 load_dataframes function을 통해 위에 있는 data 파일들을 모두 읽어서 log변환된 dataframe과 평균 VIX, risk free rate을 리턴한다
- Numpy 함수를 이용해 log_df에서 평균과 covariance matrix를 구한다
- argument에 따라 실행되는 부분이 다르다. short가 없을 때 `if` 구문, short가 있지만 gurobi를 쓰지 않을때 `elif` 구문, short가 있고 gurobi를 사용할경우 `else` 구문으로 가게 된다.
- 각각의 함수에서는 `rf_ratio`, `theta_P`, `mu_P`, `sig_P`를 리턴한다.
- `rf_ratio`는 전체 포트폴리오에서 무위험자산에 투자해야하는 비율이다. 마이너스가 나올 경우 risk free rate에 빌려서 주식에 투자한다는 뜻이다.
- `theta_P`는 각각의 주식마다 어느 정도 투자해야하는지를 뜻한다.
- `mu_P`는 기대 수익률이다.
- `sig_P`는 기대 standard deviation이다.
- 마지막 print 구문들은 필요한 정보를 standard output에 출력한다.

5.2 with_short_no_gurobi function

```
def with_short_no_gurobi(log_df, A, rf_rate, mu, cov):
    try:
        cov_inv = np.linalg.inv(cov)
    except:
        cov_inv = np.linalg.pinv(cov)

    alpha = mu.dot(cov_inv).dot(mu)
    beta = np.sum(cov_inv.dot(mu))
    gamma = np.sum(cov_inv)
    D = alpha*gamma - beta**2
    E = alpha - 2*beta *rf_rate + gamma*rf_rate**2

    def std(v):
        return math.sqrt((gamma*v**2 - 2*beta*v + alpha) / D)

    def CML(risk):
        return rf_rate + math.sqrt(E)*risk

    sig_M = math.sqrt(E)/ (beta - rf_rate*gamma)
    mu_M = rf_rate + E / (beta - rf_rate*gamma)

    theta_M =  1/(beta - gamma*rf_rate)*cov_inv.dot(mu - rf_rate)

    sig_P = math.sqrt(E)/A
```

```
    mu_P = rf_rate + math.sqrt(E)*sig_P
    UTIL = mu_P - A*sig_P*sig_P/2

    max_x = max(mu_M, mu_P)

    x_CML = np.linspace(0, max_x, 100).tolist()
    y_CML = [CML(x) for x in x_CML]

    y_eff = np.linspace(0, y_CML[-1], 100).tolist()
    x_eff = [std(y) for y in y_eff]

    x_util = np.linspace(0, max_x, 100).tolist()
    y_util = [utility(x, UTIL, A) for x in x_util]

    rf_ratio = (mu_M - mu_P)/(mu_M - rf_rate)
    stock_ratio = 1- rf_ratio
    theta_P = stock_ratio * theta_M

    #top 5
    k = 5
    idx = np.argpartition(theta_P, -k)
    top_k_names = log_df.columns[idx[-k:]].tolist()
    top_y = mu[idx[-k:]].tolist()
    top_x = [math.sqrt(cov[x][x]) for x in idx[-k:]]

    title = 'with_short_no_gurobi'
    plt.plot(x_eff, y_eff, label='Efficient Frontier')
    plt.plot(x_CML, y_CML, label='CML')
    plt.plot(x_util, y_util, label='Utility')
    plt.scatter(sig_M, mu_M)
    plt.scatter(sig_P, mu_P)
    plt.scatter(top_x, top_y, label='top_5')
    plt.title(title)
    plt.xlabel('sigma')
    plt.ylabel('return')
    plt.legend()
    for i, name in enumerate(top_k_names):
        plt.annotate(name, (top_x[i], top_y[i]))
    plt.tight_layout
    plt.savefig(title+'.png')

    return rf_ratio, theta_P, mu_P, sig_P
```

Listing 2: with_short_no_gurobi function

- short가 존재할때 gurobi를 쓰지 않고 lagrange multiplier을 사용하여 optimize하기 위한 코드이다.
- numpy에 있는 function을 통해 covariance matrix의 역행렬을 구했다. inverse가 구해지지 않을 경우 pseudo-inverse를 구하도록했다.
- Lagrange multiplier의 결과 대로 α, β, γ, D, E를 모두 구하였다.
- 구한 값들로 `std(v)`함수와 `CML(risk)`, `utility`함수를 정의하였다.
- 접할때의 값들을 계산하여 `theta_P`에 저장하였다.
- 함수들을 통해 x, y 좌표들을 각각 100개식 구하여 plt를 통해 이미지로 저장하였다.
- 포트폴리오 비중 상위 5개를 구하여 이미지에 포함하도록 하였다.

5.3 with_short_with_gurobi function

```
def with_short_with_gurobi(log_df, A, rf_rate, mu, cov):
    max_x = np.max(mu)*2
    y_eff = np.linspace(0, max_x, 100).tolist()
    x_eff = [math.sqrt(minimize_var(y, mu, cov, shortExists=True).
    ObjVal) for y in y_eff]

    y_CML = np.linspace(0, max_x, 100).tolist()
    x_CML = [math.sqrt(minimize_var_with_rf(y, rf_rate, mu, cov,
    shortExists=True).ObjVal) for y in y_CML]

    slope = (y_CML[-1]-rf_rate)/x_CML[-1]

    sig_P = slope / A
    mu_P = slope*sig_P + rf_rate
    UTIL = mu_P - A *sig_P*sig_P /2

    x_util = np.linspace(0, x_eff[-1], 100).tolist()
    y_util = [utility(x, UTIL, A) for x in x_util]

    m_P = minimize_var_with_rf(mu_P, rf_rate, mu, cov, shortExists=
    True)
    theta_P = np.array([v.x for v in m_P.getVars()])
    rf_ratio = 1- theta_P.sum()

    #top 5
    k = 5
    idx = np.argpartition(theta_P, -k)
    top_k_names = log_df.columns[idx[-k:]].tolist()
    top_y = mu[idx[-k:]].tolist()
    top_x = [math.sqrt(cov[x][x]) for x in idx[-k:]]

    title = 'with_short_with_gurobi'
    plt.plot(x_eff, y_eff, label='Efficient Frontier')
    plt.plot(x_CML, y_CML, label='CML')
    plt.plot(x_util, y_util, label='Utility')
    plt.scatter(sig_P, mu_P)
    plt.scatter(top_x, top_y, label='top_5')
    plt.title(title)
    plt.xlabel('sigma')
    plt.ylabel('return')
    plt.legend()
    for i, name in enumerate(top_k_names):
        plt.annotate(name, (top_x[i], top_y[i]))
    plt.tight_layout
    plt.savefig(title+'.png')

    return rf_ratio, theta_P, mu_P, sig_P
```

Listing 3: with_short_with_gurobi function

- minimize_var 함수(수익률이 주어졌을 때의 최소 variance)(아래 참조)를 사용하여 efficient frontier 함수를 구하였다.
- minimize_var_with_rf 함수(risk free asset이 존재하고 수익률이 주어졌을 때의 최소 variance)(아래 참조)를 사용하여 CML라인을 구하였다.
- CML직선과 접할 때의 utility를 구하여 `utility` 함수를 통해 utility curve를 구했다.

- CML 직선과의 접점에서의 포트폴리오를 찾기 위해 minimize_var_with_rf 함수를 사용해 최소 variance일대의 `theta_P`를 구하였다.
- 위와 마찬가지로 3개의 curve와 5개의 기업의 좌표들을 이미지로 저장하였다.

5.4 no_short function

```
def no_short(log_df, A, rf_rate, mu, cov):
    max_y = np.max(mu)

    y_eff = np.linspace(0, max_y, 100).tolist()
    x_eff = [math.sqrt(minimize_var(y, mu, cov, shortExists=False).
    ObjVal) for y in y_eff]

    y_CML = np.linspace(0, max_y, 100).tolist()
    x_CML = [math.sqrt(minimize_var_with_rf(y, rf_rate, mu, cov,
    shortExists=False).ObjVal) for y in y_CML]

    slope = (y_CML[-1]-rf_rate)/x_CML[-1]

    sig_P = slope / A
    mu_P = slope*sig_P + rf_rate
    UTIL = mu_P - A *sig_P*sig_P /2

    x_util = np.linspace(0, x_eff[-1], 100).tolist()
    y_util = [utility(x, UTIL, A) for x in x_util]

    m_P = minimize_var_with_rf(mu_P, rf_rate, mu, cov, shortExists=
    False)
    theta_P = np.array([v.x for v in m_P.getVars()])
    rf_ratio = 1- theta_P.sum()

    #top 5
    k = 5
    idx = np.argpartition(theta_P, -k)
    top_k_names = log_df.columns[idx[-k:]].tolist()
    top_y = mu[idx[-k:]].tolist()
    top_x = [math.sqrt(cov[x][x]) for x in idx[-k:]]

    title = 'no_short'
    plt.plot(x_eff, y_eff, label='Efficient Frontier')
    plt.plot(x_CML, y_CML, label='CML')
    plt.plot(x_util, y_util, label='Utility')
    plt.scatter(sig_P, mu_P)
    plt.scatter(top_x, top_y, label='top_5')
    plt.title(title)
    plt.xlabel('sigma')
    plt.ylabel('return')
    plt.legend()
    for i, name in enumerate(top_k_names):
        plt.annotate(name, (top_x[i], top_y[i]))
    plt.tight_layout
    plt.savefig(title+'.png')

    return rf_ratio, theta_P, mu_P, sig_P
```

Listing 4: no_short function

- with_short_with_gurobi 함수와 거의 유사하다. 다른점은 minimize_var과 minimize_var_with_rf에 `shortExists`가 `False`로 바뀌어 short를 금지한것이 다르다.

5.5 minimize_var function

```
def minimize_var(v, mu, cov, shortExists=True):
    #print(v)
    #create a model
    m = gp.Model("quadratic")
    #create variables
    if shortExists:
        theta = m.addMVar(mu.shape[0], lb=float('-inf'), ub=float('inf'), name='theta')
    else:
        theta = m.addMVar(mu.shape[0], lb=0.0, ub=float('inf'), name='theta')

    #set objective
    obj = theta @ cov @ theta
    m.setObjective(obj)

    #add Constraint
    m.addConstr((theta @ mu)-v==0, 'c0')
    m.addConstr(theta.sum()-1==0, 'c1')

    m.optimize()
    return m
```

Listing 5: minimize_var function

- gurobi를 사용해 수익률 v가 주어졌을 때의 variance를 최소화 한다.
- risk free asset은 없다고 가정한다.
- short가 존재할때는 `theta`의 upper bound와 lower bound가 존재하고 short가 존재하지 않을 때는 lower bound가 0이다.
- 아래 식을 optimize 하고자 한다.

$$\begin{aligned} \min_{\theta} \quad & \theta^t \Sigma \theta \\ \text{s.t.} \quad & \theta^t \mu = v \\ & \theta^t \mathbf{1} = 1 \\ & \theta \geq 0 \quad \text{(if short does not exist)} \end{aligned}$$

5.6 minimize_var function

```
def minimize_var_with_rf(v, rf_rate, mu, cov, shortExists=True):
    #create a model
    m = gp.Model("quadratic")
    #create variables
    if shortExists:
        theta = m.addMVar(mu.shape[0], lb=float('-inf'), ub=float('inf'), name='theta')
    else:
```

```
        theta = m.addMVar(mu.shape[0], lb=0.0, ub=float('inf'), name='theta')

    #set objective
    obj = theta @ cov @ theta
    m.setObjective(obj)

    #add Constraint
    m.addConstr( ((mu-rf_rate) @ theta) == v-rf_rate, 'c0' )

    m.optimize()
    return m
```

Listing 6: minimize_var function

- risk free asset이 존재할 때 gurobi를 사용해 수익률 v가 주어졌을 때의 variance를 최소화 한다.
- 아래 식을 optimize 하고자 한다.

$$\begin{aligned} \min_{\theta} \quad & \theta^t \Sigma \theta \\ \text{s.t.} \quad & (\mu - r_{risk\ free})^t \theta = (v - r_{risk\ free}) \\ & \theta \geq 0 \quad \text{(if short does not exist)} \end{aligned}$$

6 Results

6.1 with_short_no_gurobi

```
C:\Users\jungwoo\Desktop\FE_project_1>python 양정우hw101.py --no_gurobi
Number of top market cap companies: 30
Data start: 2018-01-01
Data end(excluding): 2021-01-01
VIX: 20.8175
Risk free rate: 0.014040819257822799
Expected portfolio return: 0.04139498871285552
Expected portfolio standard deviation: 0.009359482795215347
Percentage of portfolio to invest in risk free asset:
    0.3028371690645946
Company list:
['AAPL' 'MSFT' 'AMZN' 'GOOGL' 'FB' 'TSLA' 'BRK-B' 'V' 'JPM' 'JNJ' 'WMT'
 'NVDA' 'MA' 'UNH' 'HD' 'PG' 'BAC' 'DIS' 'PYPL' 'INTC' 'CMCSA' 'ADBE' 'VZ'
 'XOM' 'KO' 'NFLX' 'ORCL' 'PFE' 'CSCO' 'ABT']
Percentage of rest of the portfolio to invest:
[-0.51452723  1.03862066 -0.26753326 -0.16667826 -0.15298733  0.10465165
 -0.65737508 -1.17396534  1.18101128 -0.55227163  0.22166718  0.35037503
  1.37502173  0.13258762 -0.56915481  0.34481391 -0.49792726 -0.10582229
  0.33370834 -0.19165117 -0.86521789 -0.37005842  0.77566386 -0.11172911
  0.33313448  0.0784613   0.25316249 -0.09909653  0.20162504  0.26865386]
```

Listing 7: with_short_no_gurobi output

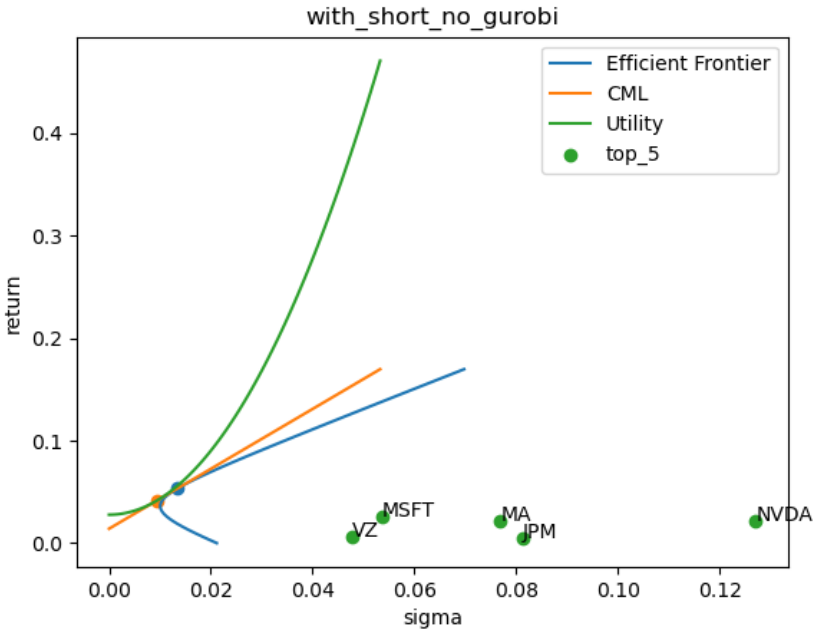

Figure 5: with_short_no_gurobi.png

- Listing 7은 short가 존재하고 gurobi를 사용하지 않았을 때의 결과를 보여준다.
- 30개의 기업을 투자대상으로 보고 있고 2018-01-01부터 2021-01-01까지의 데이터를 사용한다
- 해당 기간동안의 평균 VIX는 20.82이고 risk free rate(TBILL)은 1.4%이다
- 예상 수익률은 4.14%이고 이때 예상 표준편차는 0.0094이다.
- 결과에 따라 포트폴리를 구성하려면 총 포트폴리오 중 30.28%를 risk free asset에 투자하여야한다.
- `Company list`는 각각 기업들의 ticker을 나열하고 다음에오는 `Percentage of rest of the portfolio to invest`에서 기업들마다 투자해야하는 비중을 보여준다.
- Figure 5에는 프로그램이 output하는 그림을 보여주고 있다.
- 파란색선이 Efficient Frontier, 노란색선이 CML, 초록색 선이 utility function을 보여준다. 파란색 점이 CML과 Efficient Frontier의 접점이고 노랜색 점이 CML과 utility function의 접점이다.
- 5개의 초록색점이 비중 상위 5개의 기업이다.
- MSFT 나 JPM 같은 기업에는 가지고 있는 돈의 100% 이상 투자하고 V와 같은 기업에는 가지고 있는 돈의 100%이상 short하라는 결론이다. short를

할경우 현금이 생기므로 공매도를 통해 분산이 작고 수익률이 더 높은 기업을 살수 있는 것이다. CML과 Efficient frontier의 접점보다 CML과 utility funcition의 접점이 왼쪽 아래 있기 때문에 risk free asset의 비중이 양의 값을 보인다.

6.2 with_short_with_gurobi

```
[,,,]
Number of top market cap companies: 30
Data start: 2018-01-01
Data end(excluding): 2021-01-01
VIX: 20.8175
Risk free rate: 0.014040819257822799
Expected portfolio return: 0.04139498871285376
Expected portfolio standard deviation: 0.009359482795215047
Percentage of portfolio to invest in risk free asset:
    0.3028544722092881
Company list:
['AAPL' 'MSFT' 'AMZN' 'GOOGL' 'FB' 'TSLA' 'BRK-B' 'V' 'JPM' 'JNJ' '
    WMT'
 'NVDA' 'MA' 'UNH' 'HD' 'PG' 'BAC' 'DIS' 'PYPL' 'INTC' 'CMCSA' '
    ADBE' 'VZ'
 'XOM' 'KO' 'NFLX' 'ORCL' 'PFE' 'CSCO' 'ABT']
Percentage of rest of the portfolio to invest:
[-0.5145007   1.03860569 -0.26753244 -0.16667702 -0.15298856
    0.10464918
 -0.65737774 -1.17392563  1.18096456 -0.55226744  0.22165478
    0.35035825
  1.37496834  0.13258614 -0.56911895  0.34480191 -0.49790157
    -0.10580843
  0.33368717 -0.191646   -0.86517475 -0.37003421  0.77564128
    -0.11173926
  0.33312089  0.07846399  0.25315055 -0.09908082  0.2016186
    0.26864774]
```

Listing 8: with_short_with_gurobi output

- Listing 8은 short가 존재하고 gurobi를 사용했 때의 결과를 보여준다.
- output의 구성은 Listing 7와 같다. 하지만 이 프로그램을 돌릴 경우 gurobi의 output까지 보인다.
- 예상 수익률은 4.14%이고 이때 예상 표준편차는 0.0094이다.
- 결과에 따라 포트폴리를 구성하려면 총 포트폴리오 중 30.28%를 risk free asset에 투자하여야한다.
- Listing 7과 Listing 8을 비교하면 오차범위 내에서 결과가 같은 것을 알 수 있다.
- Figure 6에는 프로그램이 output하는 그림을 보여주고 있다.
- 파란색 점이 CML과 Efficient Frontier의 접점이고 5개의 노란색점이 비중 상위 5개의 기업이다.
- Figure 5와 Figure 6을 비교하면 둘 다 같은 결과를 보여주는 것을 또한 확인할 수 있다.

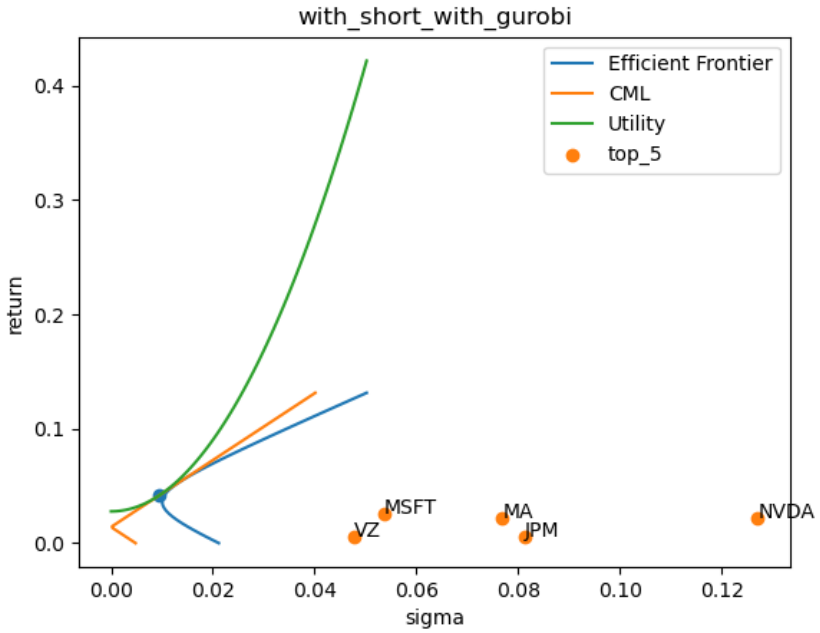

Figure 6: with_short_with_gurobi.png

6.3 no_short

```
[...]
Number of top market cap companies: 30
Data start: 2018-01-01
Data end(excluding): 2021-01-01
VIX: 20.8175
Risk free rate: 0.014040819257822799
Expected portfolio return: 0.05006469022678291
Expected portfolio standard deviation: 0.13154698616225616
Percentage of portfolio to invest in risk free asset:
    -0.5537208459916463
Company list:
['AAPL' 'MSFT' 'AMZN' 'GOOGL' 'FB' 'TSLA' 'BRK-B' 'V' 'JPM' 'JNJ' '
    WMT'
 'NVDA' 'MA' 'UNH' 'HD' 'PG' 'BAC' 'DIS' 'PYPL' 'INTC' 'CMCSA' '
    ADBE' 'VZ'
 'XOM' 'KO' 'NFLX' 'ORCL' 'PFE' 'CSCO' 'ABT']
Percentage of rest of the portfolio to invest:
[9.21067457e-02 9.82624231e-01 1.97382550e-10 5.25205976e-11
 5.21800045e-11 4.29098831e-01 3.12229578e-11 1.00336520e-10
 3.33808709e-11 3.87653422e-11 9.74145006e-11 8.90404123e-11
 3.61231384e-10 1.17824018e-10 6.54572010e-11 2.44685369e-10
 2.43501239e-11 9.76557120e-11 4.98910010e-02 3.17735416e-11
 5.21216185e-11 3.44244611e-08 5.20014003e-11 1.61648959e-11
 5.30758971e-11 7.23208461e-10 5.18646477e-11 5.60784090e-11
 5.62933781e-11 3.28636575e-10]
```

Listing 9: no_short output

- Listing 9은 short가 존재하지 않을 때의 결과를 보여준다.

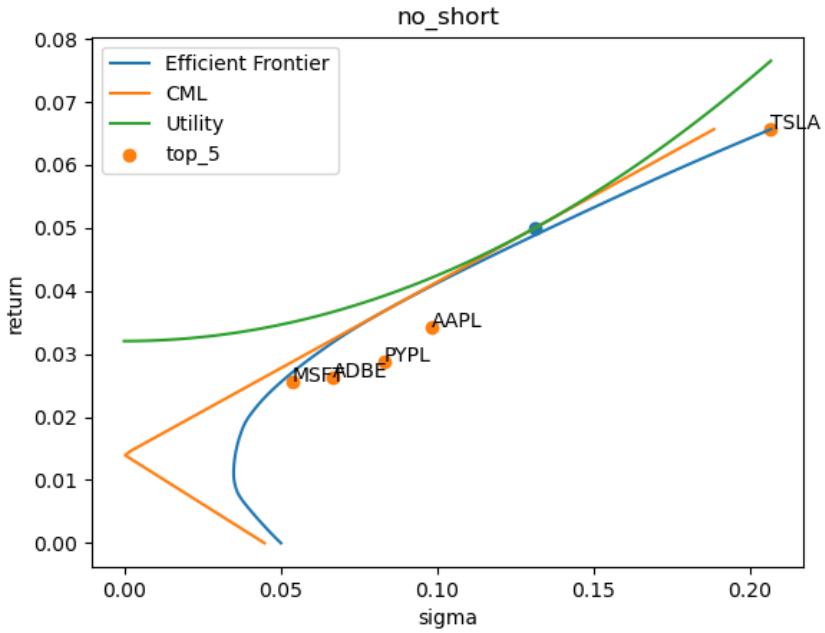

Figure 7: no_short.png

- output의 구성은 Listing 7와 같다. 하지만 이 프로그램을 돌릴 경우 gurobi의 output까지 보인다.
- 예상 수익률은 5.00%이고 이때 예상 표준편차는 0.1315이다.
- short가 없을 때 short가 있을 때보다 예상 수익률이 더 큰 이유는 utility function에서 coefficient A를 더 작은 값을 사용하여 위험에 더 둔감하게 했기 때문이다.
- 결과에 따라 포트폴리를 구성하려면 총 포트폴리오 중 -55.37%를 risk free asset에 투자하여야한다. 이 뜻은 risk free asset을 사지 않고 risk free rate에 포트폴리오의 55.37%만큼 빌려서 주식을 사야한다는 뜻이다.
- Figure 7에는 프로그램이 output하는 그림을 보여주고 있다.
- 파란색 점이 CML과 Efficient Frontier의 접점이고 5개의 노란색점이 비중 상위 5개의 기업이다.
- Short가 존재할 때와 다르게 대부분의 값들이 매우 작고 몇개의 기업들에만 유의미하게 비중이 큰 것을 알 수 있다. 이는 평균 수익률이 크고 분산이 작은 몇몇의 "좋은" 기업들에 투자하는 것이 좋다는 결과를 보여준다. Short가 있었을 때는 상대적으로 평균 수익률이 작고 분산이 큰 기업들로 분산을 줄일 수 있었지만 더이상 그렇게 할 수 없기 때문이다. 실제로 MSFT 에 포트폴리오의 98.26%를, 포트폴리오의 42.91%를 TSLA에 비중을 두라고 보여주고 있다. 그림에서도 알수 있듯이 MSFT는 다른 기업들에 비해 수익률이 높고 분산이 작은 것을 볼수 있다. TSLA의 경우 분산이 크지만 그만큼 수익률도 매우 크다.