Portfolio Optimization

Kannan Singaravelu, CQF

1 Modern Portfolio Theory

Modern portfolio theory also popularly called Mean-Variance Portfolio Theory (MVP) is a major breakthrough in finance. It is based on the premise that returns are normally distributed and by looking at mean and variance, we can essentially describe the distribution of end-of-period wealth.

The basic idea of this theory is to achieve diversification by constructing a portfolio for a minimal portfolio risk or maximal portfolio returns. Accordingly, the **Efficient Frontier** is a set of optimal portfolios in the risk-return spectrum, and portfolios located under the Efficient Frontier curve are considered sub-optimal.

This means that the portfolios on the frontier offered

- `Highest expected return for a given level of risk`
- `Lowest level of risk for a given level of expected returns`

In essence, the investors' goal should be to select a level of risk that he/she is comfortable with and then find a portfolio that maximizes returns based on the selected risk level.

1.1 Import Libraries

We'll import the required libraries that we'll use in this example.

[ ]:
```
# Import warnings
import warnings
warnings.filterwarnings('ignore')

# Import pandas & yfinance
import pandas as pd
import numpy as np
from numpy.linalg import multi_dot
import yfinance as yf

# Set numpy random seed
np.random.seed(42)
```

```
# Import cufflinks
import cufflinks as cf
cf.set_config_file(offline=True, dimensions=((1000,600)))

# Import plotly express for EF plot
import plotly.express as px
px.defaults.width, px.defaults.height = 1000,600

# Set precision
pd.set_option('display.precision', 4)
```

1.2 Retrive Data

We will retrieve price data for selected stocks from our database to build our portfolio

[ ]:
```
# Import & create database
import sqlalchemy
engine = sqlalchemy.create_engine('sqlite:///India')
```

[ ]:
```
# Read data from wikipedia - refer Lab 1 for further details
nifty50 = pd.read_html('https://en.wikipedia.org/wiki/NIFTY_50')[2].Symbol.
 ↪to_list()

# Fetch data from yahoo using list comprehension
data = [yf.download(symbol+'.NS', start="2019-01-01", end="2023-12-31",␣
 ↪progress=False).reset_index() for symbol in nifty50]

# save it to database
for frame, symbol in zip(data, nifty50):
    frame.to_sql(symbol, engine, if_exists='replace', index=False)
```

[ ]:
```
# Specify assets / stocks
# Indian stocks : bank, consumer goods, diversified, it, consumer durables
assets = sorted(['ICICIBANK', 'ITC', 'RELIANCE', 'TCS', 'ASIANPAINT'])
print(assets)

# Number of assets
numofasset = len(assets)

# Number of portfolio for optimization
numofportfolio = 5000
```

[ ]:
```
# Query close price from database
df = pd.DataFrame()
for asset in assets:
    df1 = pd.read_sql_query(f'SELECT Date, Close FROM {asset}', engine,␣
 ↪index_col='Date')
    df1.columns = [asset]
    df = pd.concat([df, df1], axis=1)

# View output
df
```

1.3 Visualize Time Series

[ ]:
```
# Plot price history
df['2023':].normalize().iplot(kind='line')
```

[ ]:
```
# Dataframe of returns and volatility
returns = df.pct_change().dropna()
annual_returns = round(returns.mean()*260*100,2)
annual_stdev = round(returns.std()*np.sqrt(260)*100,2)

# Subsume into dataframe
df2 = pd.DataFrame({
    'Ann Ret': annual_returns,
    'Ann Vol': annual_stdev
})

# Get the output
df2
```

[ ]:
```
# Plot annualized return and volatility
df2.iplot(
    kind='bar',
    shared_xaxes=True,
    orientation='h'
)
```

1.4 Portfolio Composition

[ ]:
```
df2.reset_index().iplot(
    kind='pie',
    labels='index',
    values='Ann Ret',
    textinfo='percent+label',
    hole=0.4
)
```

2 Portfolio Statistics

Consider a portfolio which is fully invested in risky assets. Let w and μ be the vector of weights and mean returns of n assets.

$$w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} ; \ \mu = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{pmatrix}$$

where the $\sum_{i=1}^{n} w_i = 1$

`Expected Portfolio Return` is then the dot product of the expected returns and their weights.

$$\mu_\pi = w^T \cdot \mu$$

which is also equivalent to the $\Sigma_{i=1}^{n} w_i \mu_i$

`Expected Portfolio Variance` is then the multidot product of weights and the covariance matrix.

$$\sigma_\pi^2 = w^T \cdot \Sigma \cdot w$$

where, Σ is the covariance matrix

$$\Sigma = \begin{pmatrix} \Sigma_{1,1} & \dots & \Sigma_{1,n} \\ \vdots & \ddots & \vdots \\ \Sigma_{n,1} & \dots & \Sigma_{n,n} \end{pmatrix}$$

2.1 Portfolio Simulation

Now, we will implement a Monte Carlo simulation to generate random portfolio weights on a larger scale and calculate the expected portfolio return, variance and sharpe ratio for every simulated allocation. We will then identify the portfolio with a highest return for per unit of risk.

[ ]:
```
def portfolio_simulation(returns):

    # initialize the lists
    rets=[]; vols=[]; wts=[]

    # simulate 5000 portfolio
    for i in range(numofportfolio):

        # generate random weights
        weights = np.random.random(numofasset)

        # set weights such that sum of weights equals 1
        weights /= np.sum(weights)
```

```
        # portfolio stats
        rets.append(weights.T@np.array(returns.mean()*260))
        vols.append(np.sqrt(multi_dot([weights.T, returns.cov()*260, weights])))
        wts.append(weights)

    # create a datafrme for analysis
    data = {'port_rets': rets, 'port_vols': vols}
    for counter, symbol in enumerate(returns.columns.tolist()):
        data[symbol+' weight'] = [w[counter] for w in wts]

    portdf = pd.DataFrame(data)
    portdf['sharpe_ratio'] = portdf['port_rets'] / portdf['port_vols']

    return round(portdf,4)
```

2.2 Maximum Sharpe Portfolio

[ ]:
```
# Create a dataframe for analysis
temp = portfolio_simulation(returns)
temp.head()
```

[ ]:
```
# Get the max sharpe portfolio stats
temp.iloc[temp.sharpe_ratio.idxmax()]
```

[ ]:
```
# Verify the above result
temp.describe().T
```

2.3 Visulaize Simulated Portfolio

[ ]:
```
# Plot simulated portfolio
fig = px.scatter(
    temp, x='port_vols', y='port_rets', color='sharpe_ratio',
    labels={'port_vols': 'Expected Volatility', 'port_rets': 'Expected␣
 ↪Return','sharpe_ratio': 'Sharpe Ratio'},
    title="Monte Carlo Simulated Portfolio"
     ).update_traces(mode='markers', marker=dict(symbol='cross'))

# Plot max sharpe
fig.add_scatter(
    mode='markers',
    x=[temp.iloc[temp.sharpe_ratio.idxmax()]['port_vols']],
    y=[temp.iloc[temp.sharpe_ratio.idxmax()]['port_rets']],
    marker=dict(color='RoyalBlue', size=20, symbol='star'),
    name = 'Max Sharpe'
).update(layout_showlegend=False)
```

```
# Show spikes
fig.update_xaxes(showspikes=True)
fig.update_yaxes(showspikes=True)
fig.show()
```

3 Efficient Frontier

The Efficient Frontier is formed by a set of portfolios offering the highest expected portfolio return for a certain volatility or offering the lowest volatility for a certain level of expected returns.

Return objective:

$$\underset{w_1,w_2,\ldots,w_n}{minimize}\sigma_p^2(w_1, w_2, \ldots, w_n)$$

subject to,

$$E[R_p] = m$$

Risk constraint:

$$\underset{w_1,w_2,\ldots,w_n}{maximize}E[R_p(w_1, w_2, \ldots, w_n)]$$

subject to,

$$\sigma_p^2(w_1, w_2, \ldots, w_n) = v^2$$

where, $\sum_{i=1}^{n} w_i = 1$ for the above objectives.

We can use numerical optimization to achieve this objective. The goal of optimization is to find the optimal value of the objective function by adjusting the target variables operating withing some boundary conditions and constraints.

3.1 Constrained Optimization

Construction of optimal portfolios is a constrained optimization problem where we specify some boundary conditions and constraints. The objective function here is a function returning maximum sharpe ratio, minimum variance (volatility) and the target variables are portfolio weights. We will use the ***minimize*** function from `scipy` optimization module to achieve our objective.

[ ]:
```
# Import optimization module from scipy
# sco.minimize?
import scipy.optimize as sco
```

3.2 Portfolio Statistics

Let's subsume key statistics into a function which can be used for optimization exercise.

[ ]:
```
def portfolio_stats(weights):

    weights = np.array(weights)
    port_rets = weights.T @ np.array(returns.mean() * 260)
    port_vols = np.sqrt(multi_dot([weights.T, returns.cov() * 260, weights]))

    return np.array([port_rets, port_vols, port_rets/port_vols])

# Minimize the volatility
def min_volatility(weights):
    return portfolio_stats(weights)[1]

# Minimize the variance
def min_variance(weights):
    return portfolio_stats(weights)[1]**2

# Maximizing sharpe ratio
def max_sharpe_ratio(weights):
    return -portfolio_stats(weights)[2]
```

3.3 Efficient Frontier Portfolio

For efficient frontier portfolios, we fix a target return and derive for objective function.

[ ]:
```
# Specify constraints, bounds and initial weights
cons = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
bnds = tuple((0,1) for x in range(numofasset))
initial_wts = numofasset*[1./numofasset]
```

[ ]:
```
# Optimizing for maximum sharpe ratio
opt_sharpe = sco.minimize(max_sharpe_ratio, initial_wts, method='SLSQP',␣
 ↪bounds=bnds, constraints=cons)

# Optimizing for minimum variance
opt_var = sco.minimize(min_variance, initial_wts, method='SLSQP', bounds=bnds,␣
 ↪constraints=cons)
```

[ ]:
```
opt_sharpe
```

[ ]:
```
opt_var
```

[ ]:
```
# Efficient Frontier
targetrets = np.linspace(0.15,0.24,100)
tvols = []
```

```
for tr in targetrets:

    ef_cons = ({'type': 'eq', 'fun': lambda x: portfolio_stats(x)[0] - tr},
               {'type': 'eq', 'fun': lambda x: np.sum(x) - 1})

    opt_ef = sco.minimize(min_volatility, initial_wts, method='SLSQP',␣
 ↪bounds=bnds, constraints=ef_cons)

    tvols.append(opt_ef['fun'])

targetvols = np.array(tvols)
```

[ ]:
```
# Create EF Dataframe for plotting
efport = pd.DataFrame({
    'targetrets' : np.around(100*targetrets,2),
    'targetvols': np.around(100*targetvols,2),
    'targetsharpe': np.around(targetrets/targetvols,2)
})

efport.head()
```

[ ]:
```
# Plot efficient frontier portfolio
fig = px.scatter(
    efport, x='targetvols', y='targetrets',  color='targetsharpe',
    labels={'targetrets': 'Expected Return', 'targetvols': 'Expected␣
 ↪Volatility','targetsharpe': 'Sharpe Ratio'},
    title="Efficient Frontier Portfolio"
     ).update_traces(mode='markers', marker=dict(symbol='cross'))


# Plot maximum sharpe portfolio
fig.add_scatter(
    mode='markers',
    x=[100*portfolio_stats(opt_sharpe['x'])[1]],
    y=[100*portfolio_stats(opt_sharpe['x'])[0]],
    marker=dict(color='red', size=20, symbol='star'),
    name = 'Max Sharpe'
).update(layout_showlegend=False)

# Plot minimum variance portfolio
fig.add_scatter(
    mode='markers',
    x=[100*portfolio_stats(opt_var['x'])[1]],
    y=[100*portfolio_stats(opt_var['x'])[0]],
    marker=dict(color='green', size=20, symbol='star'),
    name = 'Min Variance'
```

```
).update(layout_showlegend=False)

# Show spikes
fig.update_xaxes(showspikes=True)
fig.update_yaxes(showspikes=True)
fig.show()
```

4 References

- Numpy Linear Algebra
- Python Resources
- Scipy Optimization
- Styling Plotly Express Figures
- YFinance Documentation

Python Labs by Kannan Singaravelu.