2023 1학기 FR 프로젝트

Portfolio Optimization

35기: 류제현 박민규 황인준
36기: 이대원 장은경 전호연

PPT: 류제현

Contents

I. Project Description : Blueprint

· 포트폴리오 최적화 (Portfolio Optimization) → 자산 배분(Asset Allocation) / Long-Only Portfolio

1. Define Object Function

2. Estimate Expected Return and Covariance Matrix

3. Find Weight Vector that Maximize the Object Function

Target: Maximize Sharpe Ratio

$$\text{maximize} \quad \frac{\mu^T x - r_f}{\sqrt{x^T Q x}}$$

$$\text{s.t.} \quad \sum_j x_j = 1, \quad Ax \geq b, \quad 0 \leq x.$$

· 최적화 방법 (Optimization Method) : How to Find Optimal Solution?

1. SLSQP

2. Deep Learning

I. Project Description : Algorithm

- SLSQP
  - Optimization Under Constraint and Bounds / Approximate Using a Second Order Equation
  - Assets Universe : S&P500 / ETF
  - Sharpe Ratio Maximization / Variance Minimization

- Deep Learning
  - LSTM: Long Short Term Memory
  - Assets Universe : ETF
  - Sharpe Ratio Maximization

I. Project Description : Novel Approach

· Shrinkage Estimated Covariance Matrix

· The Length of Time Series < The Number of Asset → Estimation Issues Arise

· J. Bun, J. P. Bouchaud, and M. Potters (2017)
"The errors in the covariance matrix estimates further propagate to portfolio optimization problems, leading to poor out-of-sample performance of portfolios optimized using noisy estimates"

· Prevent Overfitting

· Regularization of Neural Networks

· Robustness Test

II. Data

Vender

- Daily Security Price: The Center for Research in Security Prices
- Daily S&P500 Constitute: The Center for Research in Security Prices

Preprocessing

- Prevention of Survivorship Bias: Using Daily S&P500 Constitute Data
- Using Adjusted Close Price: To Deal with Merger and Acquisition, Stock Split

III. CODE : Optimizer.py

Optimizer.py → Shrinkage_Method.py → Back_Testing.py

· Estimates Expected Return and Covariance Matrix

· Returns Weight Vector that Maximize the Object Function

```python
def run_optimizer(obj_function, rtn_df:pd.DataFrame, spx_mask, start_year:str, end_year:str, rebalancing:str,
                  look_back_size:int, max_ratio:float, shrinkage_method="None", arg="None"):
    '''
    obj_function: 목적함수 [obj_sharpe, obj_variance]
    rtn_df: 수익률 데이터프레임
    spx_mask: S&P500 mask 데이터프레임 (ETF 최적화 하는 경우에는 "None" 주면 된다)
    look_back_size : int (과거 얼마 동안의 주가를 보고 포트폴리오를 구성할 지, "days")
    max_ratio : float (개별 주식당 포트폴리오 최대 비율 제한)
    shrinkage_method : str -> ["None", "linear", "constant", "clipping"]
    arg: float -> shrinkage method에 맞는 arg 를 주면 된다. (linear, clipping만 해당!)
    '''

    constraints = ({'type': 'eq', 'fun': lambda x: np.sum(x) - 1})
    shrink = {"None": no_shrinkage,
              "linear":linear_shrinkage,
              "constant":constant_correlation_model,
              "clipping":eigenvalue_clipping
              }

    weight_df = pd.DataFrame(columns=rtn_df.columns) # weight를 담을 dataframe

    end_idx = pd.date_range(start_year,end_year, freq=f"{rebalancing}")

    for rebalancing_date in tqdm(end_idx):
        # start~rebalancing_date의 리턴을 보고 포폴 구성 (Look Back Window는 look_back_size로 조절)
        start = (rebalancing_date - pd.Timedelta(days=look_back_size))#.strftime("%Y-%m")

        if type(spx_mask) == pd.DataFrame: # S&P500을 최적화 하는 경우
            mask_sample = spx_mask.loc[:rebalancing_date].iloc[-1]
            universe = mask_sample.loc[~mask_sample.isna()].index # S&P500 구성종목을 가져옵니다
            rtn_lookback = rtn_df.loc[start:rebalancing_date, universe].dropna(axis=1) # v2개선 (이전 Lookb
        else: # ETF 최적화 하는 경우
            rtn_lookback = rtn_df.loc[start:rebalancing_date]

        universe = rtn_lookback.columns
        rtn_vol = np.diag(rtn_lookback.std())
        mean_return = rtn_lookback.mean()
        corr_matrix = rtn_lookback.corr() # corr_matrix를 추정하고 축소한 후에, optimizer에 넣기 전에 cov_mat

        shrinked_corr_matrix = shrink[shrinkage_method](corr_matrix = corr_matrix, arg=arg) # Corr_matrix를
        cov_matrix = rtn_vol.dot(shrinked_corr_matrix).dot(rtn_vol) # corr matrix를 cov matrix로 변경

        bounds = tuple((0, max_ratio) for _ in range(len(rtn_lookback.columns)))
        initial_weights = np.ones(len(rtn_lookback.columns)) / len(rtn_lookback.columns)

        # 최적화 수행
        result = minimize(obj_function,
                          initial_weights,
                          args=(cov_matrix, mean_return,),
                          method='SLSQP',
                          constraints=constraints,
                          bounds=bounds
                          )
        min_variance_weights = result.x
        weight_df.loc[rebalancing_date, universe] = min_variance_weights

    weight_df = weight_df.astype("float64") # v2추가: result의 리턴이 object였음

    return weight_df
```

III. CODE : Shrinkage_Method.py

Optimizer.py → **Shrinkage_Method.py** → Back_Testing.py

· Basic Linear Shrinkage

$$\mathbf{R}^{(lin.)} = \alpha \mathbf{I}_N + (1-\alpha)\mathbf{R}.$$

· Constant Correlation Model

$$r_{ij} = r = \frac{1}{N(N-1)} \sum_{i \neq j} R_{ij}, \quad \forall i \neq j.$$

· Eigen Value Clipping

$$\mathbf{R}^{(clip.)} = \sum_{k=1}^{N} \xi_k^{(clip.)} \mathbf{u}_k \mathbf{u}_k^{\intercal}, \quad \xi_k^{(clip.)} = \begin{cases} \lambda_k, & \text{if } k \leq K \\ \gamma, & \text{otherwise} \end{cases}$$

```python
def no_shrinkage(corr_matrix, arg):
    return corr_matrix


def linear_shrinkage(corr_matrix, arg):
    alpha = arg
    '''
    alpha: float
    corr matrix를 리턴합니다
    '''
    return  alpha * np.identity(corr_matrix.shape[0]) + (1-alpha) * corr_matrix


def constant_correlation_model(corr_matrix, arg):
    n = len(corr_matrix)
    sum_r = np.sum(corr_matrix).sum() - np.sum(np.diag(corr_matrix)).sum()
    r = sum_r / (n*(n-1))
    return np.full(corr_matrix.shape, fill_value=r) - ((r-1) * np.identity(n))


def eigenvalue_clipping(corr_matrix, arg):
    '''
    k:int
    '''
    k = arg
    eigen_value, eigen_vector = np.linalg.eigh(corr_matrix)
    eigen_value_bigger = np.where(eigen_value >= k, eigen_value, 0)
    eigen_value_smaller = eigen_value[eigen_value_bigger == 0]
    eigen_value_otherwise = np.nanmean(eigen_value_smaller)
    eigen_value_clipped = np.where(eigen_value >= k, eigen_value_bigger, eigen_value_otherwise)
    return eigen_vector @ np.diag(eigen_value_clipped) @ eigen_vector.T
```

III. CODE : Back_Testing.py

Optimizer.py → Shrinkage_Method.py → **Back_Testing.py**

· Most Important: Avoid Forward Looking Bias

· Back-Testing Should Operate Like a Market Simulator

· Consider Transaction Fees

```python
def simulate_strategy(group_weight_df:pd.DataFrame, daily_rtn_df:pd.DataFrame, fee_rate:float):
    '''
    전략의 수익을 평가합니다(Long-Only Portfolio)
    '''
    pf_value = 1
    pf_dict = {}

    weight = group_weight_df.iloc[0] # 시작 weight를 지정해준다 (첫 weight에서 투자 시작, 장마감 직전에 포트
    dollar_value = weight * pf_value # Start Dollar Value를 지정

    rebalancing_idx = group_weight_df.index # 리밸런싱 할 날들
    start_idx = rebalancing_idx[0]          # 투자 시작일

    idx = daily_rtn_df.loc[start_idx:].index
    weight_df = pd.DataFrame(index=idx, columns=daily_rtn_df.columns) # Weight 변화를 기록할 빈 데이터프레
    weight_df.loc[start_idx] = weight # 시작 weight를 기록

    for idx, row in daily_rtn_df.loc[start_idx:].iloc[1:].iterrows(): # Daily로 반복 / 시작 weight 구성
        # 수익률 평가가 리밸런싱보다 선행해야함
        dollar_value = dollar_value * (1+np.nan_to_num(row)) # update the dollar value
        pf_value = np.nansum(dollar_value) # update the pf value

        weight = dollar_value / pf_value # update the weight

        if idx in rebalancing_idx: # Rebalancing Date (장마감 직전에 리밸런싱 실시)
            weight = group_weight_df.loc[idx] # Weight Rebalancing
            target_dollar_value = np.nan_to_num(pf_value * weight) * (1 - fee_rate)
            dollar_fee = np.nansum(np.abs(target_dollar_value - np.nan_to_num(dollar_value)) * fee_rate)
            pf_value = pf_value - dollar_fee # fee 차감

            dollar_value = weight * pf_value  # dollar value를 Rebalancing 이후로 update

        weight_df.loc[idx] = weight # weight 변화를 기록
        pf_dict[idx] = pf_value

    # 결과를 pct로 정렬
    pf_result = pd.Series(pf_dict)
    idx = pf_result.index[0] - pd.Timedelta(days=1)
    pf_result[idx] = 1
    pf_result.sort_index(inplace=True)
    pf_result = pf_result.pct_change().fillna(0)

    return pf_result, weight_df
```

IV. Personal Achievement

· 100% Object Oriented Programming (OOP)

· Ray : Python Multi Processing Library

· More Accurate Back-Testing

Thank you