Black-Litterman model based on individual investors' views

2018-16135 전지훈 (경제학부)

1. Introduction

Since Markowitz (1952) formulated the concept of "efficient frontier", which deals with the mean-variance optimization problem of the assets under the known mean and variance, a myriad of subsequent portfolio analysis models focused on the same task. However, most of them turned out to be practically ineffective, and one the reasons for this was that they are using the "ex-post" mean and variance of returns in the hope that such trend would be persistent in the future periods. To handle this problem, we need to utilize "ex-ante" mean and variance of mean and variance of future returns. There may be various ways to estimate them, but Black and Litterman (1991) constructed a novel model which obtains the posterior distribution of the mean and variance of returns based on the "views" of the market participants. This enabled investors to determine optimal portfolio weights by incorporating analysts' views on the assets into the mean-variance optimization problem, but it still has limited applicability in that the releases of explicit "views" are infrequent and in most cases they are produced by the experts. Now that individual non-expert investors are significantly increasing in the stock market, and their information and opinions are highly abundant, it seems reasonable to extend the original Black-Litterman model to include the individual investors' opinions when constructing the "view" matrix.

2. Original Black-Litterman model

Consider the market of N assets, whose excess returns vector $\boldsymbol{R}$ is normally distributed, i.e.,

$$\boldsymbol{R} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \cdots (1)$$

where $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ are uncertain priors. More specifically, these priors are designed as the following:

$$\boldsymbol{\mu} \sim N(\boldsymbol{\pi}, \tau\boldsymbol{\Sigma}) \quad \cdots (2)$$

τ captures the uncertainty level of $\boldsymbol{\mu}$ given $\boldsymbol{\Sigma}$. There has been many works discussing the optimal way to set the proper τ, among which Idzorek (2007) suggests an innovative algorithm that cancels out the effect of τ. In other words, we can set τ to be any arbitrary value under the Idzorek's method. For the convenience as well as consistency of the result, this paper also follows the method devised by Idzorek.

The final goal of Black-Litterman model is to get the posterior distribution of $\boldsymbol{\mu}$ from the views of

the investors, and the prior distribution of μ, which is defined in above (2). Let K be the total number of views of the investors, $\boldsymbol{P}$ be a $K \times N$ matrix whose i_{th} row consists of the weights for each asset included in the i_{th} views, and $\boldsymbol{Q}$ be a $K \times 1$ vector which contains the returns implied by each view. Then Black-Litterman model further introduces the uncertainty of the views, which we denote as $\boldsymbol{\Omega}$, and formulate the views as the following:

$$\boldsymbol{P\mu} \sim N(\boldsymbol{Q}, \boldsymbol{\Omega}) \quad \cdots (3)$$

Alternatively, we can also write above (2) and (3) as the followings:

$$\boldsymbol{\mu} = \boldsymbol{\pi} + \epsilon^{(\boldsymbol{\pi})} \quad \cdots (4)$$

$$\boldsymbol{P\mu} = \boldsymbol{Q} + \epsilon^{(\boldsymbol{v})} \quad \cdots (5)$$

where $\epsilon^{(\boldsymbol{\pi})}$ and $\epsilon^{(\boldsymbol{v})}$ are independent, and thus $\begin{pmatrix} \epsilon^{(\boldsymbol{\pi})} \\ \epsilon^{(\boldsymbol{v})} \end{pmatrix} \sim N\left(0, \begin{pmatrix} \tau\boldsymbol{\Sigma} & 0 \\ 0 & \boldsymbol{\Omega} \end{pmatrix}\right)$.

From this formulation, we can obtain the posterior distribution of $\boldsymbol{\mu}$ by the following derivation. Let $f(\cdot)$ denote the probability density function of the random vector. From above (2), we get:

$$f(\boldsymbol{\mu}) = (2\pi)^{-\frac{N}{2}} |\tau\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{\pi})^T(\tau\boldsymbol{\Sigma})^{-1}(\boldsymbol{\mu}-\boldsymbol{\pi})} \quad \cdots (6)$$

Also, from above (5), we can write:

$$\boldsymbol{Q} = \boldsymbol{P\mu} + \epsilon^{(\boldsymbol{Q})} \quad \cdots (7)$$

where $\epsilon^{(\boldsymbol{Q})} \sim N(0, \boldsymbol{\Omega})$ since $\epsilon^{(\boldsymbol{Q})} = -\epsilon^{(\boldsymbol{v})}$, and thus $\epsilon^{(\boldsymbol{Q})}$ and $\epsilon^{(\boldsymbol{v})}$ are distributionally identical. Then from above (7), we can further write:

$$\boldsymbol{Q}|\boldsymbol{\mu} \sim N(\boldsymbol{P\mu}, \boldsymbol{\Omega}) \quad \cdots (8)$$

From this, we obtain the conditional distribution of $\boldsymbol{Q}$ given $\boldsymbol{\mu}$ as the following:

$$f(\boldsymbol{Q}|\boldsymbol{\mu}) = (2\pi)^{-\frac{K}{2}} |\boldsymbol{\Omega}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{Q}-\boldsymbol{P\mu})^T\boldsymbol{\Omega}^{-1}(\boldsymbol{Q}-\boldsymbol{P\mu})} \quad \cdots (9)$$

By the Bayes' rule, we can derive the posterior distribution of $\boldsymbol{\mu}$ given $\boldsymbol{Q}$ as the following:

$$f(\boldsymbol{\mu}|\boldsymbol{Q}) \propto f(\boldsymbol{Q}|\boldsymbol{\mu}) f(\boldsymbol{\mu}) \quad \cdots (10)$$

Combining the above (6) and (9), it is straightforward that:

$$\begin{aligned} f(\boldsymbol{Q}|\boldsymbol{\mu}) f(\boldsymbol{\mu}) &= (2\pi)^{-\frac{K}{2}} |\boldsymbol{\Omega}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{Q}-\boldsymbol{P\mu})^T\boldsymbol{\Omega}^{-1}(\boldsymbol{Q}-\boldsymbol{P\mu})} (2\pi)^{-\frac{N}{2}} |\tau\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{\pi})^T(\tau\boldsymbol{\Sigma})^{-1}(\boldsymbol{\mu}-\boldsymbol{\pi})} \\ &= (2\pi)^{-\frac{K+N}{2}} |\boldsymbol{\Omega}|^{-\frac{1}{2}} |\tau\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}[(\boldsymbol{Q}-\boldsymbol{P\mu})^T\boldsymbol{\Omega}^{-1}(\boldsymbol{Q}-\boldsymbol{P\mu}) + (\boldsymbol{\mu}-\boldsymbol{\pi})^T(\tau\boldsymbol{\Sigma})^{-1}(\boldsymbol{\mu}-\boldsymbol{\pi})]} \quad \cdots (11) \end{aligned}$$

After some calculation that simplifies (11), from above (10) and (11) we get:

$$\boldsymbol{\mu}|\boldsymbol{Q},\boldsymbol{\Omega} \sim N(\boldsymbol{\mu_{BL}},\boldsymbol{\Sigma_{BL}}) \quad \cdots (12)$$

where

$$\boldsymbol{\mu_{BL}} = \left[(\tau\boldsymbol{\Sigma})^{-1} + \boldsymbol{P}^T\boldsymbol{\Omega}^{-1}\boldsymbol{P}\right]^{-1}\left[(\tau\boldsymbol{\Sigma})^{-1}\boldsymbol{\pi} + \boldsymbol{P}^T\boldsymbol{\Omega}^{-1}\boldsymbol{Q}\right] \quad \cdots (13)$$

$$\boldsymbol{\Sigma}_{BL} = \left[(\tau\boldsymbol{\Sigma})^{-1} + \boldsymbol{P}^T\boldsymbol{\Omega}^{-1}\boldsymbol{P}\right]^{-1} \quad \cdots (14)$$

As we can observe in the above derivation, we need to calibrate $\boldsymbol{\pi}$, $\boldsymbol{\Sigma}$, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{\Omega}$ in order to get the posterior distribution of $\boldsymbol{\mu}$ under the given investors' views. The simplest one is $\boldsymbol{\Sigma}$. We simply use the variance-covariance matrix of the excess returns of each asset as $\boldsymbol{\Sigma}$, and thus it can be readily and directly computed from the market data.

In general, $\boldsymbol{\pi}$ is implicitly derived from the optimization problem of the investors, and the market cap weights w_{mkt}. Note that the mean-variance optimization problem of the investors can be described as the following:

$$max_w\left(w^T\boldsymbol{\mu} - \frac{1}{2}\delta w^T\boldsymbol{\Sigma}w\right) \quad \cdots (15)$$

From the first order condition, we get:

$$w = (\delta\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu} \quad \cdots (16)$$

is the optimal solution for above (15). Furthermore, from (16) we can derive that:

$$\hat{\boldsymbol{\mu}} = \delta\boldsymbol{\Sigma}w_{mkt} \quad \cdots (17)$$

Then we use the derived $\hat{\boldsymbol{\mu}}$ as the value of $\boldsymbol{\pi}$. But as we can see, we need to determine δ before getting the value of $\boldsymbol{\pi}$. There has been a lot of work regarding it, but in this paper we follow the most popular approach of setting δ as the following:

$$\delta = \frac{E(R_m) - R_f}{\sigma_m^2} \quad \cdots (18)$$

where R_m is the (weighted) average return of the entire stock market, R_f is the risk-free rate, and σ_m^2 is the (weighted) volatility of the entire stock market.

Undoubtedly, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are the integral factors of Black-Litterman model. If there are total 5 assets (say, stock A~E) in the market, and the 1_{st} analyst presents a report that suggests the stock A is expected to outperform the stock C by 5% next year, then $(1 \quad 0 \quad -1 \quad 0 \quad 0)$ would be the first row of $\boldsymbol{P}$, and the first element of $\boldsymbol{Q}$ would be 0.05. If the view suggest that a group of assets is expected

to outperform another group of assets, then the weights indicated in the row vector of $\boldsymbol{P}$ would be the proportions of the assets in the corresponding groups.

Finally, $\boldsymbol{\Omega}$ is designed to contain the uncertainty levels of the views, but they are highly subjective and hard to calibrate. Surprisingly, Idzorek (2007)'s method, which was mentioned above as the method that can be used to eliminate the influence of τ parameter, is in fact a method for constructing a robust $\boldsymbol{\Omega}$ matrix through rigorous iteration process. Therefore, in this paper we compute $\boldsymbol{\Omega}$ by Idzorek's algorithm.

3. New Approach

In this Section, we further develop the original Black-Litterman model described in Section 2, by incorporating individual investors' views. Among the required parameters $\boldsymbol{\pi}$, $\boldsymbol{\Sigma}$, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{\Omega}$, and $\boldsymbol{\delta}$ (which is needed when computing $\boldsymbol{\pi}$), all but $\boldsymbol{P}$ and $\boldsymbol{Q}$ can be obtained in the same manner as in the original Black-Litterman model. Therefore, the most important thing we need to consider in this Section would be the way to appropriately set $\boldsymbol{P}$ and $\boldsymbol{Q}$ so that it can properly reflect the sentiments of the individual investors in the stock market.

Let's assume that we have collected the data that illustrates the overall sentiments of the individual investments for each asset, and the sentiment for a particular asset is generally independent of the other assets, i.e., individual investors' views focus on the "absolute" prospect for the asset, not the "relative" prospect compared to the other assets. This assumption may seem unrealistic, but in reality most of the opinions by individual investors are truly naïve and myopic, and rarely compares one asset to the others in a logical manner. Note that under this assumption, each row of $\boldsymbol{P}$ matrix can be simply set to be the one-hot vector, whose non-zero entry indicates the asset related to the corresponding view.

Then it remains to construct the $\boldsymbol{Q}$ matrix that consists of the "returns" implied by the views of individual investors. One challenge regarding this is that unlike the market analysis reports by the experts, opinions of individual investors are vague and do not specify the "returns" they expect. Rather, they express abstract "sentiments" on the asset, such as "it will rise" or "we should sell". Then how can we quantify these abstracts views into the expected returns implied by those views? In this paper, we tackle this complicated problem by deriving the expected value of the returns given the sentiments. More specifically,

$$E[r_{t+1}|s_t] = \int r_{t+1}\, p(r_{t+1}|s_t) dr_{t+1} \quad \cdots (19)$$

where $s_t \in [0,1]$ is the daily ratio of positive sentiments, and $p(r_{t+1}|s_t)$ is obtained empirically.

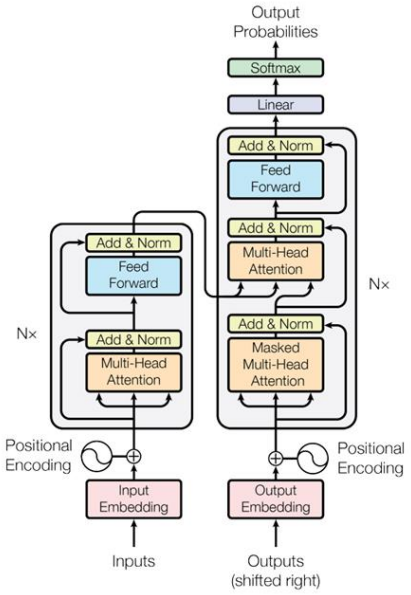
4. Experiments on Korean stock market

4.1. Sentiment Data

In this Section, we apply the proposed method discussed in the Section 3, using the data of 20 KOSPI stocks from 2017.7.10 ~ 2022.12.20, which have the largest market capitalizations as of the end of 2022. In fact, the choice of 20 stocks were not entirely done by the order of market capitalizations, because the second largest stock "LG에너지솔루션" was listed on the market at the beginning of 2022, and "삼성전자", "카카오", "셀트리온" turned out to be highly corrupted by the political opinions which are mostly irrelevant to those assets. Therefore, I chose 20 stocks by excluding these 4 stocks from the top 24 stocks.

Opinion data of individual investors were collected from the "종목토론방" of Naver Finance, and the choice of duration (2017.7.10 ~ 2022.12.20) of the data was mainly due to the availability of this data. Getting these data was conducted by Selenium module of Python, and it took approximately 30 hours (20 opinions per 1 page, and 1.4 seconds for each page on average) for the entire process to get the total 1.4M number of opinions.

In order to convert these massive amount of text data into the sentiment data, I applied the deep learning- based NLP (natural language processing) techniques that embeds the text data into the vectors of real numbers. State-of-the-art models on this domain are GPT (Generative Pretrained Transformer) and BERT (Bidirectional Encoder Representations from Transformers). Both models are based on the Transformer architecture, which has been dominating the NLP domain since 2017.



[Figure 1] Transformer Architecture

Although GPT-based models are being spotlighted these days, in this paper we embedded the text data using the KOBERT model, which is the Korean version of BERT model, and has been more widely researched by the Korean researchers compared to the GPT-based models. After embedding the text data into the real vectors using KOBERT model, we need to model the mapping between these vectors and the underlying sentiments.

This is a challenging task which requires a lot of "labeled" text data for the training of the model. But thanks to the previous Korean sentiment analysis projects conducted on the basis of KOBERT model, especially "Naver Sentiment Movie Corpus (NSMC)" project, 200K number of "labeled" text data can be easily obtained. However, training the sentiment analysis model by entirely relying on this data is inappropriate, in that they are review datasets from the movie domain in essence, so they may not reflect the sentiment patterns of the financial opinion data. Therefore, I additionally generated approximately 158K number of labeled data from the collected opinion data of individual investors from Naver Finance. Labeling process was done by manually defining the set of "positive" keywords and "negative" keywords widely used by the investors, such as:

	negative_keywords	positive_keywords
0	매도	매수
1	내리	오르
2	내린	오른
3	내릴	오를
4	하락	상승
5	폭락	폭등
6	판다	산다
7	안산다	안판다
8	안 산다	안 판다
9	안오를	안내릴
10	안 오를	안 내릴
11	안오른	안내린
12	안 오른	안 내린
13	안오르	안내리
14	안 오르	안 내리
15	비싸	벌
16	락	-
17	떨어	-
18	잃	-
19	공매	-

[Figure 2] Negative and Positive keywords for the labeling process

Then I filtered the opinions that contains only the negative keywords, or only the positive keywords. Opinion data that contain both the negative and positive keywords were excluded because they tend to be indefinite on his/her stance. After the filtering process, 158K opinions out of total 1.4M opinions left, and they were labeled according to the defined sentiment of the keywords they contained.

4.2. Sentiment Classification

Having obtained the sufficient labeled data, the next step would be to train the sentiment classification model that classifies the text data into positive/negative sentiments based on the embedding vectors of the text data from BERT model. Total 358K (200K NSMC data + 158K filtered Naver Finance opinion data) number of data was then split into 269K number of train data and 89K number of test data (3:1 ratio), and then the model was trained based on the 269K train data and validated by the 89K test data.

It took 8 hours per each epoch to train on the Ubuntu 20.04 machine with Intel c2-standard-4 CPU, so it found out not to be scalable to train on the CPUs. Thus, the training was conducted on the NVIDIA Tesla T4 GPU, which took approximately 40 minutes per each training epoch. Total training epoch was set to be 5, and the whole training took 12505 seconds (208 minutes 25 seconds). Main hyperparameters for the training process were as the following:

	max_len	batch_size	warmup_ratio	num_epochs	max_grad_norm	log_interval	learning_rate
hyperparameters	48	64	0.1	5	1	200	0.00005

[Figure 3] Training hyperparameters

"max_len" sets the maximum token size of the text data. Input of the BERT model is not the raw text data, but we need to convert the text data into the sequence of "tokens", which are generally semantic components of the text data. Since BERT model is designed to embed only the sequences of "tokens" of the same length, short sequences are zero-padded to fill up the sequence, and the last part of long sequences are discarded to fit into the required length. Then clearly, the benchmark length of the sequence needs to be set manually, and the "max_len" hyperparameter in Figure 3 does this.

"batch_size", "num_epochs", "learning_rate" are broadly used terms in the deep learning domain, so I omit the specific explanations for what they do. Values for these hyperparameters were set to be identical to the training example for the sentiment analysis provided by the developers of KOBERT. Increasing "batch_size" would have increased the accuracy of the model, but due to limited memory of GPU, setting it to 64 seemed to be maximal.

Training/test accuracy, and the loss during the training process were as the following:

```
  0%|          | 1/4204 [00:00<45:40,  1.53it/s]
epoch 1 batch id 1 loss 0.7518500089645386 train acc 0.53125
  5%|▌         | 201/4204 [01:46<35:21,  1.89it/s]
epoch 1 batch id 201 loss 0.7024027109146118 train acc 0.5032649253731343
 10%|▉         | 401/4204 [03:34<34:00,  1.86it/s]
epoch 1 batch id 401 loss 0.3296284079551697 train acc 0.6082839775561097
 14%|█▎        | 601/4204 [05:21<32:26,  1.85it/s]
epoch 1 batch id 601 loss 0.19690009951591492 train acc 0.6962354409317804
 19%|█▋        | 801/4204 [07:08<30:26,  1.86it/s]
epoch 1 batch id 801 loss 0.1867685168981552 train acc 0.746313202247191
 24%|██        | 1001/4204 [08:56<28:38,  1.86it/s]
epoch 1 batch id 1001 loss 0.2463875114917755 train acc 0.7778471528471529
 29%|██▍       | 1201/4204 [10:43<26:51,  1.86it/s]
epoch 1 batch id 1201 loss 0.33933523297309875 train acc 0.8004917776852623
 33%|██▊       | 1401/4204 [12:30<25:02,  1.87it/s]
epoch 1 batch id 1401 loss 0.2703348696231842 train acc 0.8160242683797287
 38%|███▏      | 1601/4204 [14:17<23:17,  1.86it/s]
epoch 1 batch id 1601 loss 0.16353772580623627 train acc 0.8281933166770769
 43%|███▌      | 1801/4204 [16:04<21:26,  1.87it/s]
epoch 1 batch id 1801 loss 0.08808691799640656 train acc 0.8380760688506386
 48%|███▉      | 2001/4204 [17:51<19:41,  1.86it/s]
epoch 1 batch id 2001 loss 0.23847873508930206 train acc 0.8460769615192404
 52%|████▎     | 2201/4204 [19:39<17:54,  1.86it/s]
epoch 1 batch id 2201 loss 0.1727137714624405 train acc 0.8529503634711495
 57%|████▋     | 2401/4204 [21:26<16:03,  1.87it/s]
epoch 1 batch id 2401 loss 0.2230997234582901 train acc 0.8585485214493961
 62%|█████     | 2601/4204 [23:13<14:20,  1.86it/s]
epoch 1 batch id 2601 loss 0.2420775592327118 train acc 0.8634599673202614
 67%|█████▍    | 2801/4204 [25:00<12:33,  1.86it/s]
epoch 1 batch id 2801 loss 0.1946702003479004 train acc 0.8676477151017494
 71%|█████▊    | 3001/4204 [26:47<10:43,  1.87it/s]
epoch 1 batch id 3001 loss 0.10933242738246918 train acc 0.8713605881372876
 76%|██████▏   | 3201/4204 [28:34<08:57,  1.87it/s]
epoch 1 batch id 3201 loss 0.0666903406381607 train acc 0.8746729537644486
 81%|██████▌   | 3401/4204 [30:21<07:10,  1.86it/s]
epoch 1 batch id 3401 loss 0.3035421073436737 train acc 0.8775589900029404
 86%|██████▉   | 3601/4204 [32:08<05:23,  1.86it/s]
epoch 1 batch id 3601 loss 0.1190512403845787 train acc 0.8802936684254373
 90%|███████▎  | 3801/4204 [33:56<03:35,  1.87it/s]
epoch 1 batch id 3801 loss 0.16725970804691315 train acc 0.8828268876611418
 95%|███████▋  | 4001/4204 [35:43<01:48,  1.87it/s]
epoch 1 batch id 4001 loss 0.1133481115102768 train acc 0.8850014058985254
100%|████████▏ | 4201/4204 [37:30<00:01,  1.88it/s]
epoch 1 batch id 4201 loss 0.22325681149959564 train acc 0.8873370923589622
100%|████████▏ | 4204/4204 [37:31<00:00,  1.87it/s]
epoch 1 train acc 0.8873914724072313
100%|████████▏ | 1402/1402 [04:10<00:00,  5.59it/s]
epoch 1 test acc 0.9305456490727532
```

[Figure 4-A] Training loss and accuracy: 1st epoch

```
  0%|          | 1/4204 [00:00<36:28,  1.92it/s]
epoch 2 batch id 1 loss 0.35756170749664307 train acc 0.859375
  5%|▌         | 201/4204 [01:47<35:50,  1.86it/s]
epoch 2 batch id 201 loss 0.34262320399284363 train acc 0.9251399253731343
 10%|▉         | 401/4204 [03:34<33:59,  1.86it/s]
epoch 2 batch id 401 loss 0.06666381657123566 train acc 0.9269404613466334
 14%|█▎        | 601/4204 [05:22<32:10,  1.87it/s]
epoch 2 batch id 601 loss 0.10525047034025192 train acc 0.9287645590682196
 19%|█▋        | 801/4204 [07:09<30:21,  1.87it/s]
epoch 2 batch id 801 loss 0.18274588882923126 train acc 0.9302239388264669
 24%|██        | 1001/4204 [08:56<28:40,  1.86it/s]
epoch 2 batch id 1001 loss 0.15293416380882263 train acc 0.9317401348651349
 29%|██▍       | 1201/4204 [10:44<26:50,  1.86it/s]
epoch 2 batch id 1201 loss 0.20816847681999207 train acc 0.9331416527893422
 33%|██▊       | 1401/4204 [12:31<25:01,  1.87it/s]
epoch 2 batch id 1401 loss 0.23310773074626923 train acc 0.933953426124197
 38%|███▏      | 1601/4204 [14:17<23:12,  1.87it/s]
epoch 2 batch id 1601 loss 0.05080941319465637 train acc 0.9347478138663335
 43%|███▌      | 1801/4204 [16:04<21:23,  1.87it/s]
epoch 2 batch id 1801 loss 0.03408567234873772 train acc 0.9357041227096058
 48%|███▉      | 2001/4204 [17:51<19:40,  1.87it/s]
epoch 2 batch id 2001 loss 0.1424965113401413 train acc 0.93655515992004
 52%|████▎     | 2201/4204 [19:38<17:50,  1.87it/s]
epoch 2 batch id 2201 loss 0.16334424912929535 train acc 0.9373580190822354
 57%|████▋     | 2401/4204 [21:25<16:07,  1.86it/s]
epoch 2 batch id 2401 loss 0.13382838666439056 train acc 0.9382353706788839
 62%|█████     | 2601/4204 [23:12<14:21,  1.86it/s]
epoch 2 batch id 2601 loss 0.25854936242103577 train acc 0.9391580161476355
 67%|█████▍    | 2801/4204 [24:59<12:32,  1.87it/s]
epoch 2 batch id 2801 loss 0.15013687312602997 train acc 0.9398763834344876
 71%|█████▊    | 3001/4204 [26:46<10:42,  1.87it/s]
epoch 2 batch id 3001 loss 0.10279393196105957 train acc 0.9404104881706098
 76%|██████▏   | 3201/4204 [28:33<08:56,  1.87it/s]
epoch 2 batch id 3201 loss 0.06728874891996384 train acc 0.9409461886910341
 81%|██████▌   | 3401/4204 [30:20<07:09,  1.87it/s]
epoch 2 batch id 3401 loss 0.21516314148902893 train acc 0.9414694207586004
 86%|██████▉   | 3601/4204 [32:07<05:22,  1.87it/s]
epoch 2 batch id 3601 loss 0.0707777664065361 train acc 0.942025652596501
 90%|███████▎  | 3801/4204 [33:54<03:35,  1.87it/s]
epoch 2 batch id 3801 loss 0.11862091720104218 train acc 0.9425808997632202
 95%|███████▋  | 4001/4204 [35:42<01:48,  1.87it/s]
epoch 2 batch id 4001 loss 0.04799315705895424 train acc 0.9431001624593851
100%|████████| 4201/4204 [37:29<00:01,  1.88it/s]
epoch 2 batch id 4201 loss 0.18729785084724426 train acc 0.943819179957153
100%|████████| 4204/4204 [37:30<00:00,  1.87it/s]
epoch 2 train acc 0.9438406874405328
100%|████████| 1402/1402 [04:10<00:00,  5.61it/s]
epoch 2 test acc 0.9365749821683309
```

[Figure 4-B] Training loss and accuracy: 2nd epoch

```
  0%|          | 1/4204 [00:00<36:27,  1.92it/s]
epoch 3 batch id 1 loss 0.21786531805992126 train acc 0.90625
  5%|▌         | 201/4204 [01:47<35:40,  1.87it/s]
epoch 3 batch id 201 loss 0.20705074071884155 train acc 0.9497823383084577
 10%|▉         | 401/4204 [03:34<33:55,  1.87it/s]
epoch 3 batch id 401 loss 0.039436206221580505 train acc 0.9528522443890274
 14%|█▍        | 601/4204 [05:21<32:11,  1.87it/s]
epoch 3 batch id 601 loss 0.05108603462576866 train acc 0.9545549084858569
 19%|█▊        | 801/4204 [07:08<30:26,  1.86it/s]
epoch 3 batch id 801 loss 0.07231220602989197 train acc 0.9561095505617978
 24%|██▎       | 1001/4204 [08:55<28:33,  1.87it/s]
epoch 3 batch id 1001 loss 0.07741715759038925 train acc 0.9569493006993007
 29%|██▊       | 1201/4204 [10:43<26:45,  1.87it/s]
epoch 3 batch id 1201 loss 0.20922128856182098 train acc 0.9577175270607827
 33%|███▏      | 1401/4204 [12:30<25:08,  1.86it/s]
epoch 3 batch id 1401 loss 0.186459481716156 train acc 0.9582552640970735
 38%|███▋      | 1601/4204 [14:17<23:04,  1.88it/s]
epoch 3 batch id 1601 loss 0.040096353739500046 train acc 0.9589221580262336
 43%|████▏     | 1801/4204 [16:04<21:28,  1.86it/s]
epoch 3 batch id 1801 loss 0.0372525118291378 train acc 0.9597359106052193
 48%|████▋     | 2001/4204 [17:51<19:35,  1.87it/s]
epoch 3 batch id 2001 loss 0.14670135080814362 train acc 0.9603245252373813
 52%|█████▏    | 2201/4204 [19:38<17:49,  1.87it/s]
epoch 3 batch id 2201 loss 0.11553248018026352 train acc 0.9608984552476147
 57%|█████▋    | 2401/4204 [21:25<16:05,  1.87it/s]
epoch 3 batch id 2401 loss 0.14017708599567413 train acc 0.9614743856726364
 62%|██████▏   | 2601/4204 [23:12<14:19,  1.87it/s]
epoch 3 batch id 2601 loss 0.14788664877414703 train acc 0.9620218185313341
 67%|██████▋   | 2801/4204 [24:59<12:31,  1.87it/s]
epoch 3 batch id 2801 loss 0.19709154963493347 train acc 0.9625635933595145
 71%|███████   | 3001/4204 [26:46<10:44,  1.87it/s]
epoch 3 batch id 3001 loss 0.03631721809506416 train acc 0.9629862962345884
 76%|███████▌  | 3201/4204 [28:34<08:57,  1.87it/s]
epoch 3 batch id 3201 loss 0.005295498296618462 train acc 0.9634342783505154
 81%|████████  | 3401/4204 [30:21<07:10,  1.87it/s]
epoch 3 batch id 3401 loss 0.2150292992591858 train acc 0.963769847103793
 86%|████████▌ | 3601/4204 [32:08<05:23,  1.86it/s]
epoch 3 batch id 3601 loss 0.10755099356174469 train acc 0.9643371632879756
 90%|████████▉ | 3801/4204 [33:55<03:36,  1.86it/s]
epoch 3 batch id 3801 loss 0.09058941155672073 train acc 0.9647748947645356
 95%|█████████▍| 4001/4204 [35:42<01:48,  1.87it/s]
epoch 3 batch id 4001 loss 0.024729078635573387 train acc 0.9651688640339915
100%|█████████▉| 4201/4204 [37:29<00:01,  1.87it/s]
epoch 3 batch id 4201 loss 0.08839485049247742 train acc 0.9656480599857177
100%|█████████▉| 4204/4204 [37:31<00:00,  1.87it/s]
epoch 3 train acc 0.9656614236441484
100%|█████████▉| 1402/1402 [04:11<00:00,  5.58it/s]
epoch 3 test acc 0.9392385877318117
```

[Figure 4-C] Training loss and accuracy: 3rd epoch

```
  0%|          | 1/4204 [00:00<36:14,  1.93it/s]
epoch 4 batch id 1 loss 0.06578122824430466 train acc 0.96875
  5%|▌         | 201/4204 [01:47<35:49,  1.86it/s]
epoch 4 batch id 201 loss 0.1422354280948639 train acc 0.9684390547263682
 10%|▉         | 401/4204 [03:34<33:50,  1.87it/s]
epoch 4 batch id 401 loss 0.0192982479929924 train acc 0.9712827306733167
 14%|█▎        | 601/4204 [05:22<32:27,  1.85it/s]
epoch 4 batch id 601 loss 0.048108797520399094 train acc 0.9732217138103162
 19%|█▋        | 801/4204 [07:09<30:29,  1.86it/s]
epoch 4 batch id 801 loss 0.13177327811717987 train acc 0.974211922596754
 24%|██        | 1001/4204 [08:56<28:43,  1.86it/s]
epoch 4 batch id 1001 loss 0.1322590857744217 train acc 0.9747908341658341
 29%|██▍       | 1201/4204 [10:43<26:45,  1.87it/s]
epoch 4 batch id 1201 loss 0.2027164250612259 train acc 0.975176935886761
 33%|██▊       | 1401/4204 [12:31<25:13,  1.85it/s]
epoch 4 batch id 1401 loss 0.17166078090667725 train acc 0.9757873840114204
 38%|███▏      | 1601/4204 [14:18<23:12,  1.87it/s]
epoch 4 batch id 1601 loss 0.007427994627505541 train acc 0.976235555902561
 43%|███▌      | 1801/4204 [16:05<21:25,  1.87it/s]
epoch 4 batch id 1801 loss 0.01476898230612278 train acc 0.9768184342032205
 48%|███▉      | 2001/4204 [17:52<19:35,  1.87it/s]
epoch 4 batch id 2001 loss 0.07528883963823318 train acc 0.9772379435282359
 52%|████▎     | 2201/4204 [19:39<17:52,  1.87it/s]
epoch 4 batch id 2201 loss 0.0529000423848629 train acc 0.9777302930486143
 57%|████▋     | 2401/4204 [21:26<16:04,  1.87it/s]
epoch 4 batch id 2401 loss 0.062436170876026154 train acc 0.9780755414410662
 62%|█████     | 2601/4204 [23:13<14:23,  1.86it/s]
epoch 4 batch id 2601 loss 0.10317745059728622 train acc 0.9784578046905037
 67%|█████▍    | 2801/4204 [25:01<12:30,  1.87it/s]
epoch 4 batch id 2801 loss 0.059405311942100525 train acc 0.9788747322384862
 71%|█████▊    | 3001/4204 [26:48<10:43,  1.87it/s]
epoch 4 batch id 3001 loss 0.01971041038632393 train acc 0.9790850966344552
 76%|██████▏   | 3201/4204 [28:35<08:57,  1.87it/s]
epoch 4 batch id 3201 loss 0.00317357387393713 train acc 0.9794449000312402
 81%|██████▌   | 3401/4204 [30:22<07:09,  1.87it/s]
epoch 4 batch id 3401 loss 0.1611330360174179 train acc 0.9796750955601293
 86%|██████▉   | 3601/4204 [32:09<05:23,  1.87it/s]
epoch 4 batch id 3601 loss 0.029594168066978455 train acc 0.9799534851430158
 90%|███████▎  | 3801/4204 [33:56<03:35,  1.87it/s]
epoch 4 batch id 3801 loss 0.08613819628953934 train acc 0.9802107997895291
 95%|███████▋  | 4001/4204 [35:43<01:48,  1.87it/s]
epoch 4 batch id 4001 loss 0.0122053362429142 train acc 0.9804423894026494
100%|████████▏| 4201/4204 [37:31<00:01,  1.87it/s]
epoch 4 batch id 4201 loss 0.03261991962790489 train acc 0.9806928409902405
100%|████████▏| 4204/4204 [37:32<00:00,  1.87it/s]
epoch 4 train acc 0.9807029019980971
100%|████████▏| 1402/1402 [04:10<00:00,  5.59it/s]
epoch 4 test acc 0.9398849857346647
```

[Figure 4-D] Training loss and accuracy: 4th epoch

```
  0%|          | 1/4204 [00:00<36:39,  1.91it/s]
epoch 5 batch id 1 loss 0.06526624411344528 train acc 0.96875
  5%|▌         | 201/4204 [01:47<35:49,  1.86it/s]
epoch 5 batch id 201 loss 0.04494037479162216 train acc 0.9849191542288557
 10%|▉         | 401/4204 [03:34<33:56,  1.87it/s]
epoch 5 batch id 401 loss 0.004987768363207579 train acc 0.9848425810473815
 14%|█▎        | 601/4204 [05:21<32:12,  1.86it/s]
epoch 5 batch id 601 loss 0.11397985368967056 train acc 0.985648918469218
 19%|█▋        | 801/4204 [07:08<30:18,  1.87it/s]
epoch 5 batch id 801 loss 0.015250646509230137 train acc 0.9858770287141073
 24%|██        | 1001/4204 [08:55<28:41,  1.86it/s]
epoch 5 batch id 1001 loss 0.0535762794315815 train acc 0.9862481268731269
 29%|██▍       | 1201/4204 [10:42<26:44,  1.87it/s]
epoch 5 batch id 1201 loss 0.17787984013557434 train acc 0.9865346586178185
 33%|██▊       | 1401/4204 [12:29<25:06,  1.86it/s]
epoch 5 batch id 1401 loss 0.15884895622730255 train acc 0.9868286045681656
 38%|███▏      | 1601/4204 [14:16<23:08,  1.88it/s]
epoch 5 batch id 1601 loss 0.006339280866086483 train acc 0.9869710337289195
 43%|███▌      | 1801/4204 [16:03<21:23,  1.87it/s]
epoch 5 batch id 1801 loss 0.004466739017516375 train acc 0.9873160744031094
 48%|███▉      | 2001/4204 [17:50<19:35,  1.87it/s]
epoch 5 batch id 2001 loss 0.010880510322749615 train acc 0.9873891179410295
 52%|████▎     | 2201/4204 [19:37<17:50,  1.87it/s]
epoch 5 batch id 2201 loss 0.005512502044439316 train acc 0.9876831553839164
 57%|████▋     | 2401/4204 [21:24<16:05,  1.87it/s]
epoch 5 batch id 2401 loss 0.020474033430218697 train acc 0.9877915451895044
 62%|█████     | 2601/4204 [23:11<14:17,  1.87it/s]
epoch 5 batch id 2601 loss 0.10626467317342758 train acc 0.987943339100346
 67%|█████▍    | 2801/4204 [24:58<12:34,  1.86it/s]
epoch 5 batch id 2801 loss 0.05251137167215347 train acc 0.98814597465191
 71%|█████▊    | 3001/4204 [26:46<10:43,  1.87it/s]
epoch 5 batch id 3001 loss 0.004179703537374735 train acc 0.9882122625791403
 76%|██████▏   | 3201/4204 [28:33<08:57,  1.86it/s]
epoch 5 batch id 3201 loss 0.001932527287863195 train acc 0.9883288425492034
 81%|██████▌   | 3401/4204 [30:20<07:07,  1.88it/s]
epoch 5 batch id 3401 loss 0.16232435405254364 train acc 0.9883168553366657
 86%|██████▉   | 3601/4204 [32:07<05:24,  1.86it/s]
epoch 5 batch id 3601 loss 0.023717235773801804 train acc 0.9884146764787559
 90%|███████▎  | 3801/4204 [33:53<03:36,  1.87it/s]
epoch 5 batch id 3801 loss 0.008132265880703926 train acc 0.9885556432517758
 95%|███████▋  | 4001/4204 [35:41<01:48,  1.87it/s]
epoch 5 batch id 4001 loss 0.016083626076579094 train acc 0.9886356535866033
100%|████████▏ | 4201/4204 [37:28<00:01,  1.87it/s]
epoch 5 batch id 4201 loss 0.04420106112957001 train acc 0.9886857295881933
100%|████████▏ | 4204/4204 [37:29<00:00,  1.87it/s]
epoch 5 train acc 0.9886900868220743
100%|████████▏ | 1402/1402 [04:11<00:00,  5.58it/s]
epoch 5 test acc 0.9408991619115549
```

[Figure 4-E] Training loss and accuracy: 5th epoch

Trained sentiment classification model was then used to classify the 1.4M opinions from individual investors. It took approximately 65 minutes to evaluate the sentiments of these opinions, and the resulting data looked like the following:

datetime	title	up	down	code	sentiment	negative_score	positive_score
2022.12.13 09:24	오늘 많이떨어지것다	2	1	3670	0	4.408072	-4.763484
2022.12.13 09:20	올1천/낙1만-->탈출이 답인가???	1	0	3670	0	4.401919	-4.755765
2022.12.13 09:13	포스코케미칼 반등이 약하게 된다면 [1]	2	1	3670	0	4.195201	-4.457035
2022.12.13 09:10	개인투자자분들께...포케 매매 Tip	2	0	3670	1	-2.990105	3.370367
2022.12.13 09:06	포케는 오늘도,,늬들이 짖어도 간다. [2]	3	0	3670	0	0.277773	-0.260140
2022.12.13 08:59	오늘	5	0	3670	0	1.415466	-1.433906
2022.12.13 08:56	미 IRA법 시행 대표수혜주 30% 조정...	4	0	3670	0	3.563857	-3.760287
2022.12.13 08:40	외국인은 한달내내~~~매수중 [3]	5	1	3670	1	-4.434346	4.749610
2022.12.13 08:11	한달사이~무려 30%빠졌다 [1]	12	2	3670	0	4.313931	-4.623551
2022.12.13 07:54	단기 급락에 대한 반발매수세~들어온다	5	2	3670	1	-4.164011	4.645588
2022.12.13 07:44	돈내려온다	5	1	3670	0	4.259833	-4.633797
2022.12.13 01:07	영원한 사랑도 없듯이	3	2	3670	1	-1.640952	1.851110
2022.12.12 23:47	공매 어지간히 우려먹네...외인은 즙즙하... [3]	12	2	3670	0	4.399431	-4.753947
2022.12.12 22:09	ㅈ [1]	3	1	3670	0	0.091282	-0.085800
2022.12.12 21:58	이제 널부터 매수시작^^	9	4	3670	1	-4.448109	4.781313
2022.12.12 21:36	주봉하단이 20주선에 접하며 반등한 밑... [1]	17	3	3670	0	3.649208	-3.876546
2022.12.12 19:33	공매알바	18	2	3670	0	4.413364	-4.761756
2022.12.12 18:48	공매도가?	4	0	3670	0	4.409524	-4.754293
2022.12.12 18:33	추매합쉬다.	8	0	3670	1	-2.568037	2.926799
2022.12.12 18:38	[삭제된 게시물의 답글]포켐	3	0	3670	0	0.401274	-0.220131
2022.12.12 18:05	실리콘음극재(포스코실리콘솔루션)	5	3	3670	1	-2.932342	3.341092
2022.12.12 17:44	금수저 지원내역.. [4]	28	1	3670	0	2.955530	-3.199262
2022.12.12 17:16	돌문서답..때문에.. [3]	9	1	3670	0	3.720638	-4.048096
2022.12.12 17:12	스틸란티스 최소 12조이상 주문해야	7	3	3670	1	-2.425738	2.793896
2022.12.12 17:00	공매도 상황 지속.. [6]	21	2	3670	0	4.406501	-4.751821
2022.12.12 16:55	연초에 살거없다. 결국2차전지 다시 담는... [1]	15	0	3670	1	-4.374680	4.726540
2022.12.12 16:50	[이데일리] "유럽선 전기차 탈만하네".....	5	0	3670	0	0.401274	-0.220131
2022.12.12 16:40	18지지하고 개같이 반등합니다	1	0	3670	0	2.162091	-2.246669
2022.12.12 16:17	오늘 개미들만 털렸네. 외인은 오늘도 순... [1]	3	0	3670	1	-3.770332	4.220664
2022.12.12 16:10	5만으로 떨어져라	3	1	3670	0	4.412640	-4.763538

[Figure 5] Sentiment Analysis Result

4.3. Market Data

Now that the sentiment data for the individual investors' views are obtained, the next data we need to get is the market excess return data of the chosen 20 assets. Thanks to "pykrx" module of Python, daily return data for 20 assets based on the closing price of each day were obtained. Then to compute the "excess" return, I used 3-year Korean government bond yield data from Investing.com as the risk-free rate. In the world market, especially in the United States, 10-year government bond yield is considered as the standard measure for the risk-free rate, but in Korea there is a convention of using 3-year government bond yield as the measure of the risk-free rate, so I followed this convention.

Focusing on "daily" returns when it comes to the portfolio optimization problem is not a common approach, because most of the models use the yearly return data, and at least the monthly return data. But in this paper, considering the highly frequent nature of opinions provided by individual investors, I decided to apply the Black-Litterman model for the estimation of daily returns.

Out of 5 parameters for the Black-Litterman model, $\boldsymbol{\pi}$, $\boldsymbol{\Sigma}$ can be solely calibrated by the market data. First of all, $\boldsymbol{\Sigma}$ can be directly derived from the excess return data of the 20 assets. The following is the part of $\boldsymbol{\Sigma}$ matrix:

code	000270	000660	003670	005380	005490	005935	006400	012330	015760	028260	032830	033780	034730	035420	051910
code															
000270	0.000480	0.000119	0.000119	0.000336	0.000141	0.000095	0.000133	0.000321	0.000079	0.000150	0.000143	0.000065	0.000160	0.000100	0.000144
000660	0.000119	0.000532	0.000192	0.000130	0.000138	0.000236	0.000228	0.000121	0.000052	0.000168	0.000116	0.000030	0.000197	0.000142	0.000186
003670	0.000119	0.000192	0.001007	0.000136	0.000146	0.000139	0.000437	0.000126	0.000061	0.000153	0.000103	0.000026	0.000218	0.000131	0.000378
005380	0.000336	0.000130	0.000136	0.000454	0.000148	0.000112	0.000137	0.000342	0.000088	0.000162	0.000172	0.000061	0.000149	0.000116	0.000165
005490	0.000141	0.000138	0.000146	0.000148	0.000425	0.000120	0.000121	0.000154	0.000097	0.000149	0.000159	0.000061	0.000155	0.000073	0.000177
005935	0.000095	0.000236	0.000139	0.000112	0.000120	0.000266	0.000178	0.000105	0.000052	0.000148	0.000108	0.000034	0.000135	0.000106	0.000152
006400	0.000133	0.000228	0.000437	0.000137	0.000121	0.000178	0.000662	0.000155	0.000070	0.000178	0.000124	0.000035	0.000227	0.000149	0.000417
012330	0.000321	0.000121	0.000126	0.000342	0.000154	0.000105	0.000155	0.000503	0.000088	0.000173	0.000170	0.000076	0.000159	0.000117	0.000175
015760	0.000079	0.000052	0.000061	0.000088	0.000097	0.000052	0.000070	0.000088	0.000353	0.000110	0.000108	0.000056	0.000096	0.000077	0.000095
028260	0.000150	0.000168	0.000153	0.000162	0.000149	0.000148	0.000178	0.000173	0.000110	0.000367	0.000193	0.000062	0.000211	0.000119	0.000172
032830	0.000143	0.000116	0.000103	0.000172	0.000159	0.000108	0.000124	0.000170	0.000108	0.000193	0.000400	0.000078	0.000148	0.000071	0.000134
033780	0.000065	0.000030	0.000026	0.000061	0.000061	0.000034	0.000035	0.000076	0.000056	0.000062	0.000078	0.000152	0.000070	0.000034	0.000044
034730	0.000160	0.000197	0.000218	0.000149	0.000155	0.000135	0.000227	0.000159	0.000096	0.000211	0.000148	0.000070	0.000512	0.000150	0.000215
035420	0.000100	0.000142	0.000131	0.000116	0.000073	0.000106	0.000149	0.000117	0.000077	0.000119	0.000071	0.000034	0.000150	0.000472	0.000151
051910	0.000144	0.000186	0.000378	0.000165	0.000177	0.000152	0.000417	0.000175	0.000095	0.000172	0.000134	0.000044	0.000215	0.000151	0.000695
055550	0.000142	0.000105	0.000093	0.000140	0.000180	0.000097	0.000086	0.000151	0.000101	0.000138	0.000185	0.000082	0.000155	0.000062	0.000133
066570	0.000169	0.000179	0.000181	0.000155	0.000137	0.000126	0.000180	0.000166	0.000081	0.000160	0.000127	0.000042	0.000184	0.000126	0.000177
096770	0.000185	0.000197	0.000320	0.000225	0.000209	0.000147	0.000336	0.000259	0.000108	0.000201	0.000178	0.000074	0.000309	0.000130	0.000332
105560	0.000153	0.000125	0.000112	0.000143	0.000194	0.000114	0.000116	0.000148	0.000106	0.000152	0.000193	0.000078	0.000175	0.000075	0.000166
207940	0.000076	0.000102	0.000214	0.000091	0.000070	0.000093	0.000204	0.000073	0.000064	0.000197	0.000076	0.000017	0.000139	0.000116	0.000166

[Figure 6] Part of the Var-Cov matrix $\boldsymbol{\Sigma}$ of excess returns for 20 assets

To compute $\boldsymbol{\pi}$, estimation of $\boldsymbol{\delta}$, the risk-aversion parameter of the investors in the Korean market, must be preceded. From above (18) and the market data of the 20 assets, it turned out to be:

$$\boldsymbol{\delta} = 2.77141 \quad \cdots (20)$$

According to the various previous works, $\boldsymbol{\delta}$ is known to have the value of 2.15 ~ 2.65, and thus this result seems to be reasonable. When computing $E(R_m)$ required in above (18), I first obtained R_m by computing the weighted average of the returns of the 20 assets, and $E(R_m)$ was simply computed by the arithmetic average over the observed period, then σ_m^2 was computed from R_m. Slightly high value of $\boldsymbol{\delta}$ obtained in above (20) might be due to the choice of the 20 assets, which had the largest market capitalizations, and thus their investors might tend to be conservative compared to the other assets in the stock market.

From the $\boldsymbol{\delta}$ obtained in above (20), $\boldsymbol{\pi}$, which is the implied prior for the excess returns, was:

code	implied_return
000270	0.000444
000660	0.000553
003670	0.000533
005380	0.000477
005490	0.000412
005935	0.000399
006400	0.000588
012330	0.000481
015760	0.000247
028260	0.000469
032830	0.000390
033780	0.000144
034730	0.000508
035420	0.000384
051910	0.000604
055550	0.000359
066570	0.000442
096770	0.000621
105560	0.000410
207940	0.000421

[Figure 7] Implied prior ($\boldsymbol{\pi}$) for the excess returns

4.4. Defining the view vector $\boldsymbol{Q}$

Now, it remains to define $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{\Omega}$. As discussed in the Section 3, defining $\boldsymbol{P}$ is straightforward. Also, defining $\boldsymbol{\Omega}$ is mostly covered by Idzorek's method, so indeed the most crucial remaining hyperparameter now is the view vector $\boldsymbol{Q}$. The best way to do this would be appealing to above (19), but due to the lack of empirical data, we cannot fully implement (19) directly. Although there are 1.4M number of individual opinions, and 26800 daily ratio of positive sentiments (s_t), it is impossible to cover the whole $[0,1]$ space. And even though that might be possible, low number of certain values of s_t would lead to the inaccuracy of $p(r_{t+1}|s_t)$. Some advanced kernel estimation methodologies might be useful to complement this issue, but considering the highly irrational and irregular nature of public opinions, such estimations are likely to lack robustness as well.

In this paper, we detour this complication by discretizing continuous variable s_t. Many different discretization methods can be applied to it, but I chose to discretize s_t by:

$$s'_t = \lfloor 10 \times s_t \rfloor \quad \cdots (21)$$

where $\lfloor x \rfloor$ denotes the integer part of x, after discarding the decimal part of x. Then above (19) is modified to:

$$E[r_{t+1}|s'_t] = \int r_{t+1}\, p(r_{t+1}|s'_t) dr_{t+1} \quad \cdots (22)$$

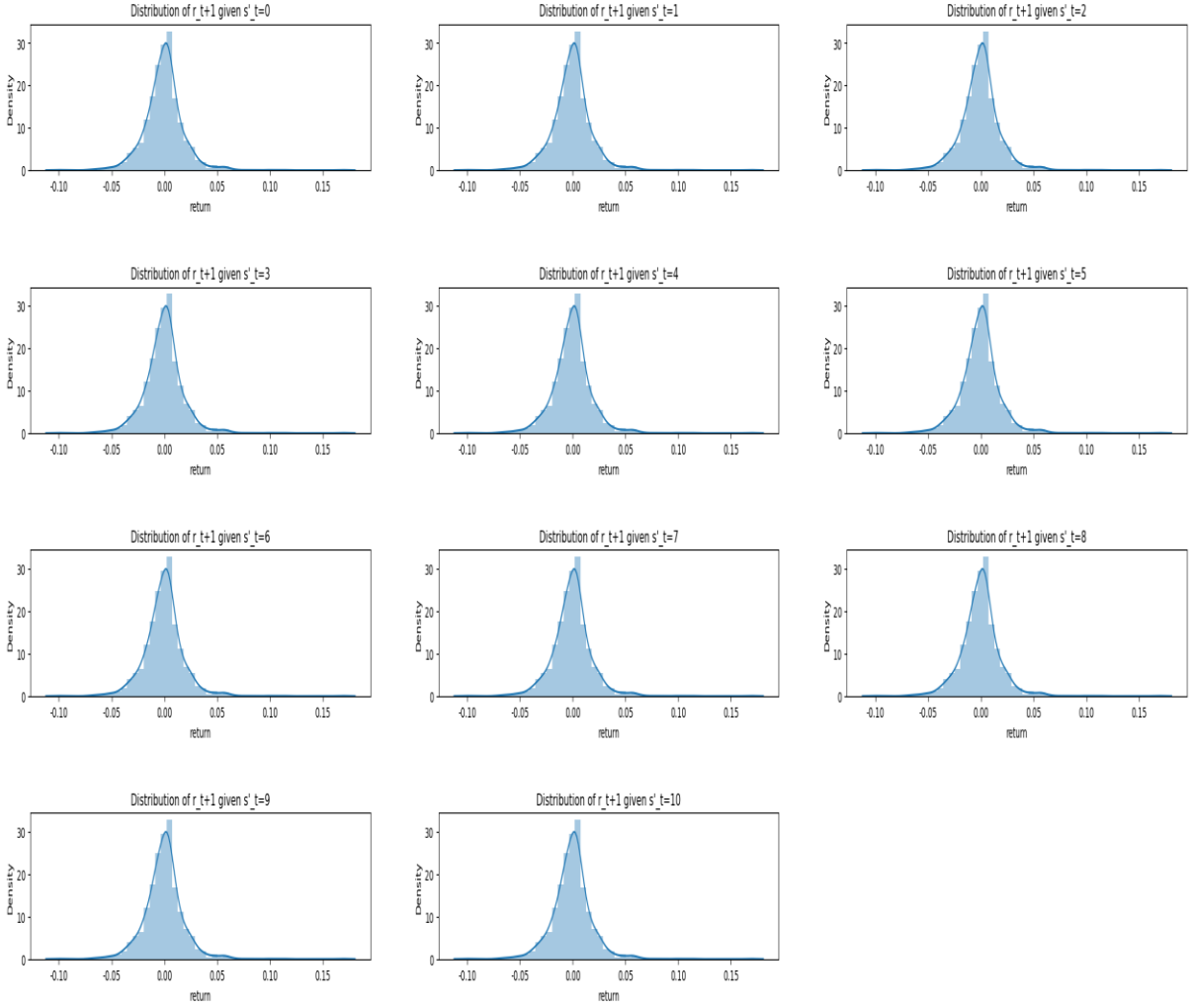
where $s'_t \in \{0,1,\cdots,10\}$.

Also, since we are evaluating above (22) empirically, trying to estimate the intractable $p(r_{t+1}|s'_t)$ is redundant. Instead, we obtain the empirical expectations of r_{t+1} given s'_t, and the following is the result of this:

sentiment_segment	mean	std
0	-0.000546	0.018356
1	-0.001277	0.020348
2	-0.000367	0.023538
3	-0.000432	0.023590
4	0.000306	0.023074
5	0.001206	0.022952
6	0.001376	0.021879
7	0.002316	0.021850
8	-0.001505	0.021266
9	-0.002071	0.010012
10	-0.000005	0.019287

[Figure 8] $E[r_{t+1}|s'_t]$ and standard deviations for each s'_t

In fact, although we ignored the issue of directly estimating the distribution $p(r_{t+1}|s'_t)$, empirical distribution of r_{t+1} given each s'_t turned out to be approximately normal as the following:



[Figure 9] Distribution $p(r_{t+1}|s'_t)$ for each s'_t

Then from above (22) and its result illustrated in above [Figure 8], we can specify the implied return corresponding to each daily sentiment ratio s_t, by discretizing it into s'_t and then obtaining its corresponding $E[r_{t+1}|s'_t]$. By implementing this for each view, we can obtain each "expected" return implied by the sentiment of the current period, and thus view vector $\boldsymbol{Q}$ can be constructed according to it.

Note that $\boldsymbol{P}$ matrix in this paper will always be an identity matrix, because we are using one "daily ratio" for each asset, and thus there would always be a single "view" for each asset implied by its ratio.

4.5. Constructing the uncertainty matrix $\boldsymbol{\Omega}$

Before we construct $\boldsymbol{\Omega}$ through the Idzorek's method, we need to specify the diagonal confidence matrix $\boldsymbol{C}$, which represents the confidence levels of each view, ranging from 0 to 1. Then for each diagonal element c of $\boldsymbol{C}$, corresponding diagonal element ω of $\boldsymbol{\Omega}$ is computed by:

$$\omega = \tau \frac{1-c}{c} \boldsymbol{p}\boldsymbol{\Sigma}\boldsymbol{p}^T \quad \cdots (23)$$

where $\boldsymbol{p}$ is a row vector corresponding to diagonal element c.

Then how should we define the confidence levels for each view? Since we have defined the view as the daily ratio of positive sentiment from the individual investors, expected return implied by this ratio is uncertain in essence. Indeed, standard deviations shown in above [Figure 8] implies that expected returns are highly volatile even when s_t' is fixed. While it makes sense to use those standard deviations in the formulation of the confidence levels, converting standard deviations into confidence levels, which range from 0 to 1, needs more clarification.

In this paper, we define the confidence level by finding the probability of expected return having a different sign from the mean of expected return given s_t'. In other words, confidence level c_i for the i_{th} view, whose sentiment is given as s_t', is computed by:

$$c_i = P([r_{t+1}|s_t'] E[r_{t+1}|s_t'] < 0) \quad \cdots (24)$$

Intuition behind this way of formulation is that "confidence" of the view should be closely related to whether the view correctly expected the sign of the future return. It may seem absurd to someone, but it is $\boldsymbol{\Omega}$ that contains the "subjective" confidence level for each view, so there is no perfectly correct way to do this. On average, the value of c_i calibrated by above (24) turned out to be around 0.578.

4.6. Experiment results

Views are set to be refreshed on a daily basis, i.e., there are 20 views, corresponding to each asset, for each day, which are defined by the ratio of positive sentiments of individual investors. Therefore, estimation under Black-Litterman model was also conducted on a daily basis, by updating $\boldsymbol{\delta}, \boldsymbol{\pi}, \boldsymbol{\Sigma}, \boldsymbol{Q}$ daily. For the computation of $\boldsymbol{\delta}, \boldsymbol{\pi}, \boldsymbol{\Sigma}$, past 1-year data were used for each day, and thus experiment data were ranging from the data of 2017.7.11 ~ 2018.8.11 to the data of 2021.12.20~2022.12.20 for each experiment. Also, for each experiment I recorded the correlation between the actual return of the subsequent trading day and the expected rate implied by the Black-Litterman model, and also the RMSE between them. From these 844 number of experiments, I got the following result:

	corrleation	rmse
count	844.000000	844.000000
mean	0.034515	0.020569
std	0.274225	0.010362
min	-0.766922	0.007377
25%	-0.168107	0.014235
50%	0.037193	0.018510
75%	0.237389	0.023795
max	0.796859	0.109490

[Figure 10] Daily experiment result

It turned out that expected return implied by the Black-Litterman model that incorporates the individual investors' views are almost uncorrelated with the actual future return. There may be several reasons for this, but the biggest reason would be the high volatility of the daily return data. Also, the values of $\boldsymbol{\delta}$ were not stable as well, and it had negative values sometimes as well which is described in the following:

	delta
count	844.000000
mean	1.558907
std	5.967058
min	-8.900276
25%	-3.915978
50%	1.899652
75%	5.667136
max	14.522519

[Figure 11] $\boldsymbol{\delta}$ value from the daily data

Since $\boldsymbol{\delta} = \frac{E(R_m)-R_f}{\sigma_m^2}$ from above (18), result in [Figure 11] contradicts the main assumption of the investing domain: high risk, high return. This contradictory underlying parameters may have led to this result.