EFFICIENT SOLUTION OF PORTFOLIO OPTIMIZATION PROBLEMS VIA DIMENSION REDUCTION AND SPARSIFICATION

CASSIDY K. BUHLER AND HANDE Y. BENSON

Abstract. The Markowitz mean-variance portfolio optimization model aims to balance expected return and risk when investing. While this framework is popular among investors, there is a significant limitation when solving large portfolio optimization problems efficiently: the large and dense covariance matrix. Since portfolio performance can be potentially improved by considering a wider range of investments, it is imperative to be able to solve large portfolio optimization problems efficiently, typically in microseconds. We propose dimension reduction and increased sparsity as remedies for the covariance matrix. The size reduction is based on predictions from machine learning techniques and the solution to a linear programming problem. We find that using the efficient frontier from the linear formulation is much better at predicting the assets in Markowitz efficient frontier, compared to the predictions from neural networks, logistic regression, and naive Bayes. Reducing the covariance matrix based on these predictions decreases both runtime and total iterations. We also present a technique to sparsify the covariance matrix such that it preserves positive semi-definiteness, which improves runtime per iteration. The methods we discuss all achieved similar portfolio expected risk and return as we would obtain from a full dense covariance matrix, but with improved optimizer performance.

Department of Decision Sciences and MIS, Bennett S. LeBow College of Business, Drexel University, Philadelphia PA 19104

Date: July 16, 2021.

1. Introduction

The Markowitz mean-variance portfolio optimization model [22] aims to balance expected return and risk when investing. Investors with different risk tolerances can choose to put different levels of relative importance on these objectives and an efficient frontier can be constructed representing the optimal portfolios for all possible risk tolerances.

Let $p_{j,t}$ represent the (known) closing price for stock $j = 1, \ldots, N$ on day $t = 1, \ldots, (T-1)$. The return $x_{t,j}$ is calculated as

$$x_{t,j} = \frac{p_{j,t} - p_{j,t-1}}{p_{j,t-1}}. \tag{1}$$

For portfolio weights $w \in \mathcal{R}^N$, the portfolio return at time $t = 1, \ldots, (T-1)$ is computed by

$$R_t = \sum_{j=1}^{N} w_j x_{t,j}. \tag{2}$$

Denoting the return matrix as $X \in \mathcal{R}^{(T-1)\times N}$, we can also write $R = Xw$.

The portfolio return on day T, $\mathbf{R}_T$, is a random variable with

$$\mathbb{E}[\mathbf{R}_T] = \mu^T w, \qquad \mathbb{V}[\mathbf{R}_T] = w^T \Sigma w \tag{3}$$

where $\Sigma = \mathrm{cov}(X)$ and $\mu_j = \mathbb{E}[x_{T,j}],\ j = 1, ..., N$.

The Markowitz model is formulated as:

$$\begin{aligned} \max_{w} \quad & \mu^T w - \lambda w^T \Sigma w \\ \text{s.t.} \quad & e^T w = 1 \\ & w \geq 0 \end{aligned} \tag{4}$$

where e is a ones vector and $\lambda \geq 0$ is the risk aversion parameter. As we vary λ, we obtain all optimal portfolios and represent them as the *efficient frontier* (Figure (1)).

The expected return, μ, is typically measured as the mean of historical returns in the Markowitz framework, but it can be replaced by any forecast of the returns. Risk is interpreted as volatility, and the covariance matrix, Σ, is used to capture the variance of returns for individual investments and to evaluate opportunities to mitigate (or increase) portfolio risk by simultaneously choosing investments with negatively (or positively) correlated returns. While μ is typically updated daily (or as new forecasts are available), Σ uses extensive historical information to assess correlation and is generally static.

The Markowitz framework is still quite popular among investors [6, 21, 9] and now extends to settings beyond financial instruments, including energy planning and investments [7, 32, 1, 24, 19]. Improved machine learning and other data science techniques to calculate μ and Σ [3, 23, 30, 4] are an area of research. As investment platforms grow, it becomes important to calculate these quantities and develop optimal portfolios in real time. Since portfolio performance can be improved by considering a wider range of investments, it is imperative to solve large instances of (4) efficiently.

The biggest challenge to efficiency is the use of Σ: it is large and dense. Nevertheless, it has several advantages we wish to exploit: it is a good representation of hedging opportunities,

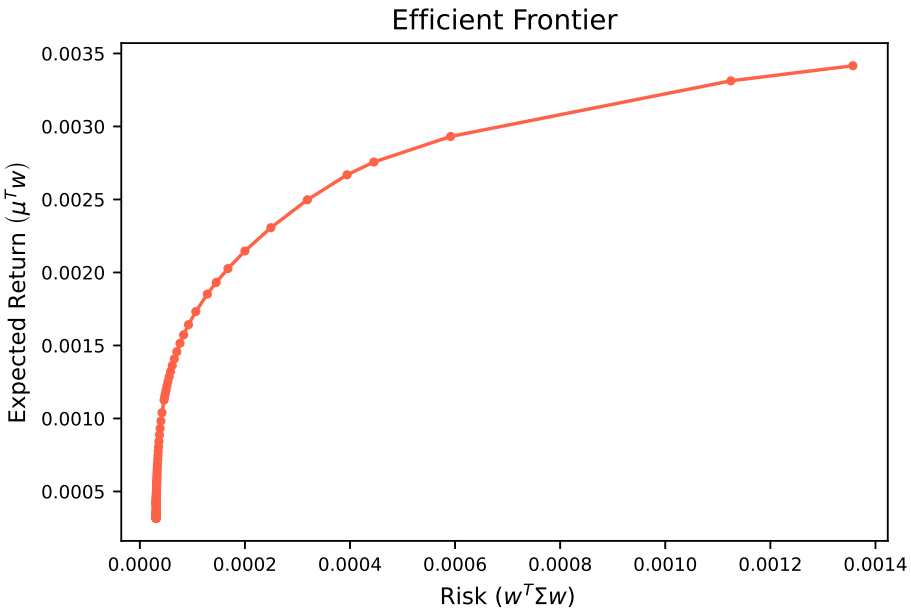

FIGURE 1. The efficient frontier for 374 stocks selected from the S&P500 using the daily returns from January 23rd 2012 to December 31st 2019.

it is positive semidefinite (thus, (4) is a convex quadratic optimization problem), and it does not need to be updated frequently.

For large instances, the number of stocks included in the portfolios on the efficient frontier is often significantly smaller than N. This property gives an opportunity to reduce or sparsify the covariance matrix by omitting entries that do not impact the optimal risk. For example, in Figure (2), only 64 out of the 374 stocks were included in any optimal portfolio. This implies that dimensions or the number of nonzeros in Σ can be significantly reduced.

Therefore, in order to improve our efficiency, we will focus on *reducing the size or increasing the sparsity of* Σ.

- The proposed size reduction techniques are predictive: machine learning algorithms, namely neural networks, or reformulation of (4) as a linear programming problem (LP) is used to predict the assets in the optimal portfolios along the efficient frontier. The risk term in (4) is then limited to these assets. For each approach, our hypothesis is that the predictive models will give a sufficient match to the investments in the optimal portfolios.
- The sparsification techniques are based on the correlation matrix of the stock returns. Correlations close to -1 or 1 not only represent strong and sustained relationships among pairs of stocks but they also represent significant enough contributions to the risk term of (4). As such, correlations close to zero can be replaced by zero, which yields a sparse matrix. Care must be taken to retain positive semidefiniteness. The main hypothesis for this technique is that the variance and large correlations are sufficient to determine the optimal portfolios and represent their risk.

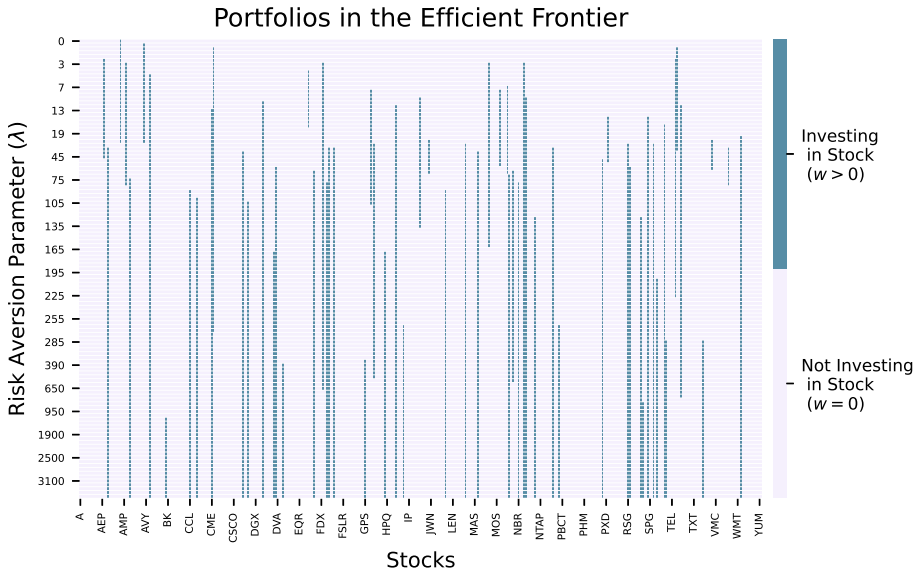

FIGURE 2. 119 portfolios form the efficient frontier for the problem instance from Figure 1. Each row is an optimal portfolio for a given λ. A stock is teal if it is included in the portfolio, and light purple otherwise.

We will refer to the three techniques as reduction by neural networks, reduction by LP, and sparsification by correlation and now present the motivation for choosing these three techniques for our study:

(1) *Reduction by neural networks*: Neural networks–once trained–are much faster and can yield better results than other probabilistic classifiers such as logistic regression or naive Bayes. They have been used in literature for stock price prediction [26, 12] or classifying stocks based on market performance [16, 13]. Essentially, the goal of this existing work is to predict the data for (4), whereas our goal in this paper is to predict which stocks appear in the optimal basis for (4) for any value of λ. Neural networks are ideally suited for such tasks.

(2) *Reduction by LP*: Another way to predict the optimal solution is to solve a similar, simpler optimization problem. By redefining the way that risk is measured, we can reformulate (4) as an LP [29]. Not only can the LP be solved faster than the QP for each value of λ, the use of the parametric simplex method, as proposed by [29], allows us to recover the entire efficient frontier within the solution of a single LP.

(3) *Sparsification by correlation*: (4) is typically solved using an interior-point method, which requires the factorization of a dense matrix in the KKT system at each iteration. Moreover, the nonnegativity requirement means that the dense matrix changes in each iteration and requires $\mathcal{O}(N^3)$ operations. On the other hand, the time complexity of sparse factorization is proportional to the number of nonzero elements in the matrix [14], which can yield significant improvement.

It is important to answer the question of why sparsification may be pursued when we know that a size reduction will typically improve runtimes. To illustrate the motivation behind sparsification, we generated a large sparse matrix and a small dense matrix to compare how

long it takes to factor each matrix. The first matrix is 99% sparse and 10000×10000, and the second matrix is dense and 2000×2000. We used Cholesky factorization and factored the matrices 100 times. The large sparse matrix took an average of 0.0037 seconds to factor while the small dense matrix took 0.0752 seconds. It is, therefore, possible that we can do better than size reduction techniques when a significant level of sparsity can be attained.

The outline of the paper is as follows. In Section 2, we introduce our reduction methods with a long short-term memory neural network and a linear programming formulation, followed by the proposed method of sparsification by correlation. In Section 3, we provide details on our financial data, which consists of daily closing prices for individual stocks in the S&P 500 index. The numerical results are provided in Section 4 for all three proposed methods, and we account for other commonly used predictive methods (logistic regression, naive Bayes, and pattern recognition neural networks) as well. We finish with a discussion of our findings and future work in Section 5.

2. Methodologies

In this section, we present the three proposed methods in detail.

2.1. Reduction by Neural Networks: Prediction with Long Short-Term Memory Network.

We used classification to predict the stocks that will be included in the portfolio and use these predictions to create a reduced covariance matrix. The network architecture we chose to implement is a long short-term memory (LSTM) network [17] due to its high predictive ability for financial data as shown in literature [10, 28]. Note that predicting the solution to every optimal portfolio on the efficient frontier is a much more complex problem than predicting stock prices or returns, but the network architecture is quite similar.

In order to train a network, we split up the matrix of historical daily returns X by rows to get a train and test set. The first t rows of X are denoted as X_{train} and the remaining $T - t$ rows as X_{test}. We then use X_{train} as the input to the Markowitz model and obtain a solution $w^*(\lambda)$ for every λ. We introduce the target vector y^*_{train}, defined as

$$(y_{\text{train}})^*_i = \begin{cases} 0 & w^*_i(\lambda) = 0 \text{ for all } \lambda \geq 0 \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

This information is used to classify each stock into two groups: If $(y_{\text{train}})^*_i = 0$, stock i is said to be in Class 0, and if $(y_{\text{train}})^*_i$, stock i is said to be in Class 1. We can then repeat this same process with X_{test} to obtain y^*_{test}.

We train an LSTM network with X_{train} and y^*_{train} and the network gives a prediction which we call $\tilde{y}$. This prediction is then used to reduce the full covariance matrix Σ to $\tilde{\Sigma}$ where $\tilde{\Sigma}$ consist of only the i-th row(s) and column(s) of Σ where $\tilde{y}_i = 1$.

$$\tilde{\Sigma} = \Sigma_{\tilde{y}_i = 1, \tilde{y}_i = 1} \tag{6}$$

LSTM units learn from five weights, and they have three gates that control how much information gets through: forget gate, input gate, and output gate. They also combine each new input with old hidden output, and they can carry some of this older data. This cell state remembers how much information from the past based on the gates. Our network has 5 layers:

(1) **Sequence Input Layer:** The single dimension input layer is given X_{train}.

(2) **LSTM Layer:** We elected to use 150 hidden nodes. The state activation function is the hyperbolic tangent function. The gate activation function is the sigmoid function. The input and recurrent weights are initialized with Matlab's default Glorot [15] and Orthogonal [25], respectively.
(3) **Fully Connected Layer:** This layer takes in the output from the previous layer and reshapes the data to prepare for the classification.
(4) **Softmax Layer:** This function is defined in the appendix, computed with X_{train} and the target vector y^*_{train}.
(5) **Weighted Cross Entropy Classification Output:** The weights are computed by the priors of training data and the function is also defined in the appendix.

After we train the network based on X_{test}, we input X_{test} and the network gives a prediction for $\tilde{y}$. We use this prediction to compare with y^*_{test} to gain insight on the predictive ability of the network.

2.2. **Reduction by LP: Prediction with the Parametric Simplex Method.** One interpretation of the risk aversion parameter λ in (4) is that it is the penalty parameter in a two-objective model. In such a framework, it is also possible to shift the role of the penalty term to the portfolio return. To do so, we introduce a new penalty parameter $\gamma \geq 0$.

We now review the LP reformulation of (4) as presented in [29]. Recall that the covariance matrix can be written as

$$\Sigma = A^T A, \text{ where } A = X - \bar{X}, \bar{X}_{ij} = \frac{1}{T-1}\sum_{t=1}^{T-1} X_{t,j}.$$

The matrix A represents the deviation of the actual return from its expectation, and the covariance term in the Markowitz model measures the magnitude of this deviation in a quadratic sense. To form the LP, we instead measure it in an absolute sense:

$$\begin{array}{ll} \max_w & \gamma\mu^T w - |A^T w| \\ \text{s.t.} & e^T w = 1 \\ & w \geq 0. \end{array}$$

Then, we introduce an auxiliary variable v to complete the reformulation:

$$\begin{array}{ll} \max\limits_w & \gamma\mu^T w - \dfrac{1}{T}\displaystyle\sum_{i=1}^{T} v_i \\ \text{s.t.} & -v \leq \mathbf{A}^T w \leq v \\ & e^T w = 1 \\ & w, v \geq 0. \end{array} \tag{7}$$

As described in [29], this problem is then solved using the parametric self-dual simplex method ([5]) with a specific pivot sequence that uncovers optimal portfolios for every value of $\gamma \geq 0$ in one pass of the method. Our proposal in this paper is to use these portfolios to predict which investments will the included in the solutions of (4) and reduce its risk term accordingly.

2.3. **Sparsification by Correlation.** In this method, we will replace the covariance matrix, Σ, in (4) with a sparse matrix obtained by replacing entries corresponding to small correlations with zero. There are two reasons why this approach is reasonable: First, the covariance matrix is positive semidefinite, we expect it to be diagonally dominated. Therefore, the typical decision of whether or not to include an investment in the optimal portfolio is made first and foremost with a focus on return and variance. The off-diagonal values impact the optimal solution only if they have a significant impact on the risk. Second, large entries in the covariance matrix can be an indicator of strong correlation between investments' historical returns. We would expect strong correlations to be exhibited for many time periods, making it quite likely that such a relationship will continue into the future. As such, it stands to reason that the small values in the correlation matrix can be replaced with zeros, and that this change would lead to the corresponding covariance matrix to be sparse.

The challenge here is two-fold: (1) we need to determine an appropriate definition of a small correlation, and (2) the resulting sparse covariance matrix needs to be positive semidefinite. To address these challenges, we use a two-step process. We determine a threshold for correlation, below which the correlation and, therefore, the covariance is replaced by 0. After obtaining the resulting sparse matrix, we add back in some of the original values to re-establish positive semidefiniteness. The domain of possible threshold values is taken to be the entries of the correlation matrix, and we are able to implement a brute-force search to find a large enough threshold that increases sparsity while preserving positive semidefiniteness.

Let θ be the dense $N \times N$ correlation matrix, τ be the set of threshold values (with $0 \leq \tau < 1$) obtained from the unique values of θ, and $\hat{\Sigma}(\tau_k)$ be the sparse $N \times N$ covariance matrix defined for threshold τ_k. We propose the following simple scheme to sparsify Σ:

$$\hat{\Sigma}_{ij}(\tau_k) = \begin{cases} 0 & -\tau_k \leq \theta_{ij} \leq \tau_k \\ \Sigma_{ij} & \text{otherwise} \end{cases}, \text{ for all } i, j = 1, \ldots, N \tag{8}$$

However, $\hat{\Sigma}(\tau_k)$ may be indefinite for any value of τ_k with no clear pattern, especially as the matrix size increases. In order to remedy the indefiniteness, we pair our scheme with partial matrix completion. Before formally describing our algorithm, we illustrate it with an example here (provided with further details in the Appendix).

Consider a 4×4 sparse covariance matrix whose nonzero elements are labeled with stars:

$$\begin{bmatrix} * & & & \\ & * & * & \\ & * & * & * \\ & & * & * \end{bmatrix}$$

Since there are off-diagonal values in columns 2, 3 and 4, the matrix completion method would add back in the elements $(2, 4)$ and $(4, 2)$. The resulting matrix would consist of the variance of the returns of Stock 1 and the covariance matrix of the returns for Stocks 2, 3, and 4:

$$\begin{bmatrix} * & & & \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}$$

This method is shown as Algorithm 1.

Algorithm 1: Brute Force Search with Partial Matrix-Completion

$\hat{n} \leftarrow \emptyset$
$\hat{\Sigma} \leftarrow \Sigma$
$\tau \leftarrow$ unique values of θ sorted by magnitude in increasing order
for each $\tau[i]$ in τ **do**
  set $\hat{\Sigma}(\tau[i])$ according to Equation (8).
  $\hat{n} \leftarrow$ column(s) where there exists an off-diagonal non-zero entry in $\hat{\Sigma}$
  $\hat{\Sigma}[\hat{n}, \hat{n}] = \Sigma[\hat{n}, \hat{n}]$
end for

We can show that Algorithm 1 always yields a positive semidefinite matrix, thereby ensuring that replacing Σ with $\hat{\Sigma}$ in (4) always yields a convex quadratic programming problem.

Theorem 1. *Let Σ be a covariance matrix for X. Then, $\hat{\Sigma}$ obtained by applying Algorithm 1 to Σ is positive semidefinite.*

Proof. Since Σ is a covariance matrix, it is symmetric and positive semidefinite. Let S denote the set of column indices of $\hat{\Sigma}$ with only diagonal entries and let S' denote its remaining column indices. Without loss of generality, we can denote the form of $\hat{\Sigma}$ as

$$\hat{\Sigma} = \begin{bmatrix} C & \\ & D \end{bmatrix},$$

where the submatrix C is a diagonal matrix whose entry $C_{j,j}$ is the variance of Stock j's returns for each stock $j \in S$ and submatrix D is a full matrix whose entries match the corresponding terms in Σ. Therefore, D can also be defined as the covariance matrix of the returns of the stocks in set S', and, as such, it is positive semidefinite. We can define $C^{\frac{1}{2}}$ as the diagonal matrix with the standard deviations of the stock returns from S and $D^{\frac{1}{2}}$ as the corresponding entries of A. Then, we can rewrite

$$\hat{\Sigma} = \begin{bmatrix} C^{\frac{1}{2}} & \\ & D^{\frac{1}{2}} \end{bmatrix}^T \begin{bmatrix} C^{\frac{1}{2}} & \\ & D^{\frac{1}{2}} \end{bmatrix}.$$

Therefore, $\hat{\Sigma}$ is positive semidefinite. □

3. Data

The financial data was collected from Yahoo! Finance over the dates January 23rd 2012 to December 31st 2019 for 374 firms selected from the S&P500 index. The returns are computed using the percentage change of the closing price. This gives a total of 1999 days of data. For reduction by neural networks, we partition the data into the first 1499 days for training and the subsequent 500 days for testing. We chose a 75/25 split since it would give a large enough testing set such that the covariance matrix would not be rank deficient.

Modifying the risk aversion parameter λ can be challenging, since we cannot use sensitivity analysis to guarentee that we find every portfolio along the efficient frontier. We approached this by finding the λ that achieves the minimum risk of a portfolio, where the minimum risk

is the optimal objective to the following problem.

$$\begin{aligned} \min_{w} \quad & w^T \Sigma w \\ \text{s.t.} \quad & e^T w = 1 \\ & w \geq 0 \end{aligned} \tag{9}$$

We then increase the λ in (4) until we obtain a portfolio risk that is within the distance $\epsilon = 1 \times 10^{-8}$ of its minimum risk. This is the maximum λ for the dataset. Then, to find the rest of the values, we modified step sizes based on the number of iterations it took to reach the next portfolio. If for a particular step size it took a considerable number of iteration (15+), we reduced the step size. The step sizes started small then increased as λ increased.

4. Numerical Testing

Numerical testing was conducted to compare the optimizer performance and portfolio performance of each approach. We used the yfinance package [2] v0.1.54 to obtain stock data for data collection. The numerical analysis was performed on a computer with a 2.7 GHz Quad-Core Intel Core i7 processor. We used Gurobi Optimizer version 9.0.2 build v9.0.2rc0 (mac64) in Matlab 2020a to solve (4). Gurobi was also used to solve (7) for an initial value of γ, followed by pivots conducted in a Python 3 implementation of the parametric simplex method. The neural networks were implemented with Matlab Deep Learning toolbox.

For every solution on the efficient frontier, we recorded the CPU time, total number of iterations, and CPU time per iteration. Whenever possible, we attempted to use the same λ values to construct the efficient frontiers. Given that CPU time can vary on each run for reasons external to the numerical testing, we computed the efficient frontier 20 times and recorded the mean and median for the CPU times.

In addition, each solution on the efficient frontier gives an optimal objective, expected risk, expected return and actual return which we also reported. The portfolio performance looks at the mean and median of the objective function $\mu^T w - \lambda w^T \Sigma w$, expected risk $\mathbb{V}[\mathbf{R}] = w^T \Sigma w$, expected return $\mathbb{E}[\mathbf{R}] = \mu^T w$, and actual return. The actual portfolio return ($\tilde{\mathbf{R}}$) is computed using the actual stock returns on day T ($\tilde{\mu}$):

$$\tilde{\mathbf{R}} = \tilde{\mu}^T w.$$

Lastly, we recorded the number of assets which have nonzero weights in any efficient portfolio. The use of the primal simplex solver in Gurobi allowed for a precise count of the number of nonzero weights in each solution.

4.1. Reductions by Neural Networks and by Linear Programming.

As presented in Section 2.1, we used the adaptive moment estimation (Adam) optimizer [20] as the training solver method, with the parameter settings given in Table 1.

For the full covariance matrix, we used 119 values of λ. In Matlab syntax, these values were $\lambda = [0 : 0.5 : 5, 6 : 1 : 20, 25 : 5 : 295, 300 : 30 : 500, 550 : 50 : 1000, 1500 : 100 : 3500]$, to construct the efficient frontier. The covariance matrix given from the predictions of LSTM network and Simplex method had 99 values of λ, $\lambda = [0 : 0.5 : 5, 6 : 1 : 20, 25 : 5 : 295, 300 : 30 : 500, 550 : 50 : 1000, 1500 : 100 : 1500]$.

For each predictive method, we include a confusion matrix [27] to visualize the accuracy.

Hyperparameter	Value
Gradient Decay Factor	0.9000
Squared Gradient Decay Factor	0.9990
Initial Learning Rate	0.00025
L2 Regularization	0.00001
Max Epochs	60
Mini Batch Size	187
Number of Mini Batches	2
Epsilon Denominator Offset	1.0000e-08

TABLE 1. Hyperparameters for Adam optimizer.

Reduction Type	None	LSTM	LP
Matrix Size	374×374	102×102	88×88
TotalIter	146	82	131
MeanIter	1.2269	0.8283	1.3232
AvgRuntime	0.1712	0.1436	0.1477
MedRuntime	0.1689	0.1363	0.1412
AvgRuntimePerIter	0.1396	0.1734	0.1116
MedRuntimePerIter	0.1395	0.1678	0.1070

TABLE 2. Optimizer Performance for Reductive Methods.

Reduction Type	None	LSTM	LP
Matrix Size	374×374	102×102	88×88
Total # Assets	64	41	55
Mean Obj	-0.01818	-0.00818	-0.00594
Median Obj	-0.00552	-0.00530	-0.00395
Mean $\mathbb{V}[\mathbf{R}]$	0.00008	0.00008	0.00009
Median $\mathbb{V}[\mathbf{R}]$	0.00003	0.00004	0.00003
Mean $\mathbb{E}[\mathbf{R}]$	0.00077	0.00095	0.00087
Median $\mathbb{E}[\mathbf{R}]$	0.00046	0.00073	0.00052
Mean $\tilde{\mathbf{R}}$	-0.00016	-0.00468	0.00022
Median $\tilde{\mathbf{R}}$	-0.00338	-0.00702	-0.00407

TABLE 3. Portfolio Performance for Reductive Methods.

4.2. **Sparsification by Correlation.** Our method of using correlations to sparsify the covariance matrix, followed by partial matrix completion to recover positive semidefiniteness, seemed to work well.

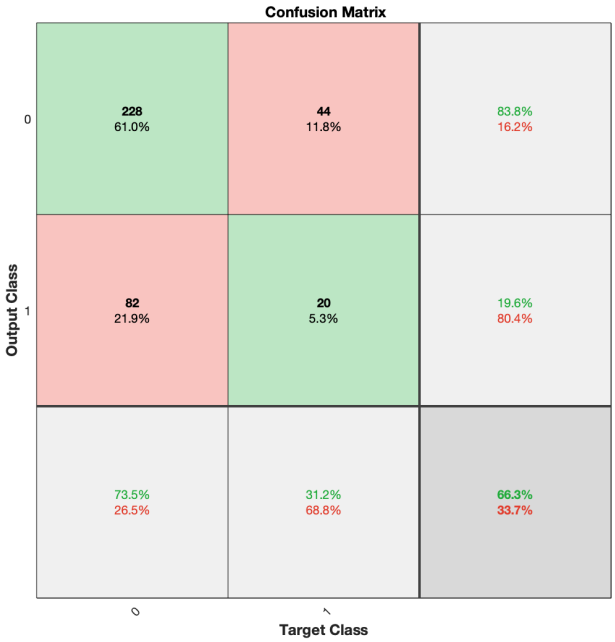

FIGURE 3. The confusion matrix for the LSTM network predictions shows a 66.3% overall accuracy for the 374 stocks.

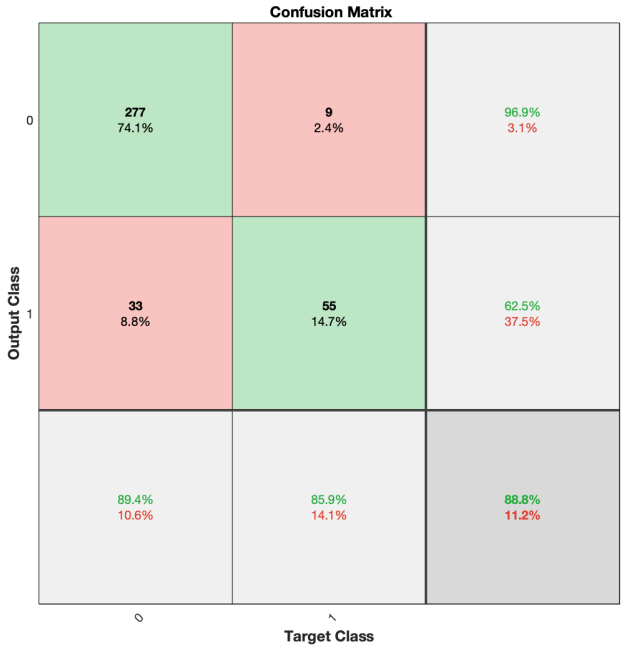

FIGURE 4. The confusion matrix for the LP predictions shows a 88.8% overall accuracy for the 374 stocks.

We measure sparsity by the number of zero-valued elements divided by the total number of elements:

$$\text{Sparsity Level} = \frac{\text{number of zero-valued elements}}{\text{total number of elements}} \tag{10}$$

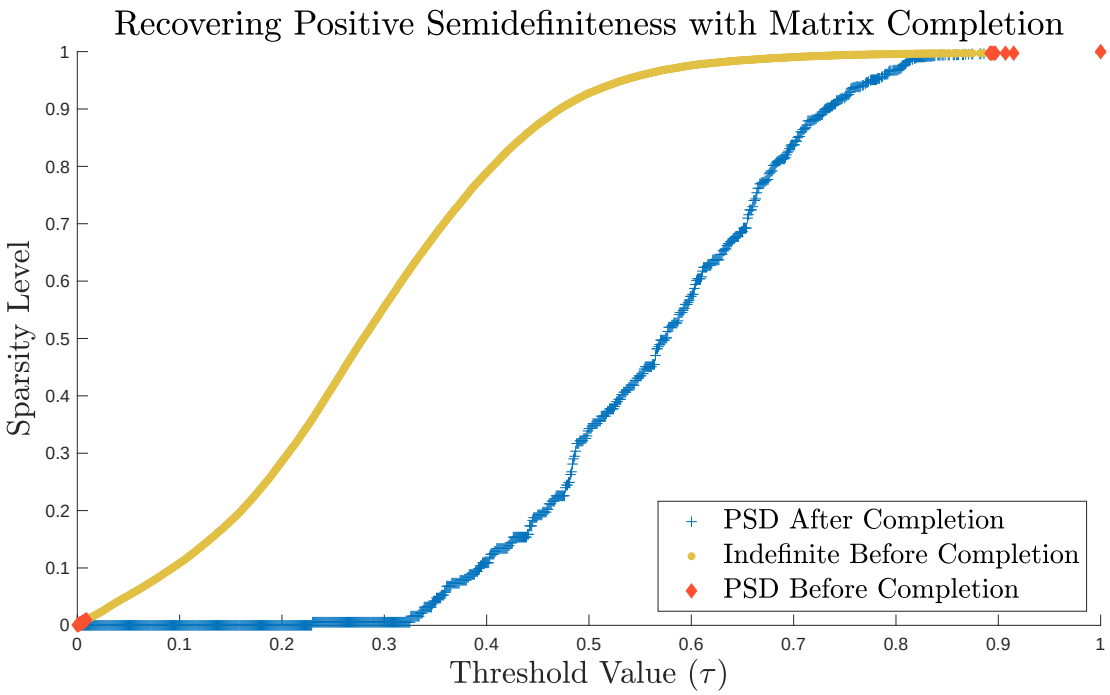

FIGURE 5. This graph compares the positive definiteness and sparsity using the brute force search with and without matrix completion.

Sparsity	0%	50%	60%	70%	80%	90%	95%	99%
TotalIter	146	248	282	327	347	398	422	469
AvgIter	1.2269	2.0840	2.3697	2.7479	2.9160	3.3445	3.5462	3.9412
AvgRuntime	0.1712	0.1739	0.1704	0.1724	0.1728	0.1751	0.1763	0.1800
MedRuntime	0.1689	0.1756	0.1719	0.1742	0.1745	0.1767	0.1778	0.1748
AvgRuntimePerIter	0.1396	0.0835	0.0719	0.0627	0.0593	0.0523	0.0497	0.0457
MedRuntimePerIter	0.1395	0.0836	0.0715	0.0628	0.0591	0.0522	0.0498	0.0437

TABLE 4. Optimizer Performance for Sparsification by Correlation.

We see that for $0.4 \leq \tau \leq 0.7$, partial matrix completion as a means to re-establish positive semidefiniteness also requires a significant compromise on sparsity. However, a correlation coefficient is considered to have a weak relationship for $0 \leq \tau \leq 0.5$ and moderate relationship for $0.5 \leq \tau \leq 0.8$ [8]. Therefore, using Algorithm 1 with $\tau > 0.8$ can indeed result in the retention of only strong relationships in our risk measure while simultaneously promoting both sparsity and positive semidefiniteness.

To measure the method's optimizer and portfolio performance, we chose eight positive semidefinite matrices with varying sparsity levels where the 0% sparse matrix is the full covariance matrix. The input to the Markowitz model are the covariance matrices and the full μ vector, which we did not sparsify since it is a linear coefficient thus has little strain on the runtime. We used the same λ values as in the dense covariance matrix from Section 4.1.

Sparsity	0%	50%	60%	70%	80%	90%	95%	99%
Total Assets	64	150	183	215	244	287	322	373
AvgObj	-0.01818	-0.00074	-0.00041	-0.00013	0.00007	0.00024	0.00035	0.00047
MedObj	-0.00552	0.00007	0.00016	0.00030	0.00037	0.00046	0.00052	0.00056
Avg $\mathbb{V}[\mathbf{R}]$	0.00008	0.00011	0.00011	0.00011	0.00011	0.00012	0.00012	0.00012
Med $\mathbb{V}[\mathbf{R}]$	0.00003	0.00006	0.00006	0.00006	0.00006	0.00007	0.00007	0.00007
Avg $\mathbb{E}[\mathbf{R}]$	0.00077	0.00096	0.00096	0.00100	0.0010	0.00104	0.00106	0.00107
Med $\mathbb{E}[\mathbf{R}]$	0.00046	0.00070	0.00070	0.00079	0.00080	0.00084	0.00088	0.00089
Avg $\tilde{\mathbf{R}}$	-0.00016	0.00386	0.00365	0.00445	0.00489	0.00554	0.00642	0.00675
Med $\tilde{\mathbf{R}}$	-0.00338	-0.00030	-0.00053	0.00034	0.00068	0.00151	0.00255	0.00264

TABLE 5. Portfolio Performance for Sparsification by Correlation.

Sparsity	0%	50%	60%	70%	80%	90%	95%	99%
TotalIter	168	305	347	391	451	512	546	613
AvgIter	1.3023	2.3643	2.6899	3.0310	3.4961	3.9690	4.2326	4.7519
AvgRuntime	0.1389	0.1457	0.1456	0.1435	0.1421	0.1420	0.1444	0.1564
MedRuntime	0.1383	0.1474	0.1472	0.1443	0.1428	0.1426	0.1450	0.1495
AvgRuntimePerIter	0.1067	0.0616	0.0541	0.0474	0.0406	0.0358	0.0341	0.0329
MedRuntimePerIter	0.1067	0.0617	0.0543	0.0472	0.0407	0.0357	0.0341	0.0313

TABLE 6. Optimizer Performance for Sparsification by Correlation when Negative Correlations are Present.

In the dataset for our numerical testing discussed so far (details provided in Section 3), we favored having a large dataset in order to observe and establish strong correlations. However, given the strong performance of the S&P500 index during this time period and the fact that we only collected data on stocks that were represented in the index for the entire time horizon, we did not observe any significant negative correlations in the dataset. Nevertheless, negative correlations, and the resulting negative covariances, are a critical element of reducing risk via hedging opportunities. As such, we repeated the above experiment with a shorter dataset that has more stocks: 497 assets for 502 days starting November 21 2017 to November 19th 2019. The covariance matrix for this dataset included strong negative correlations. For this problem, we had 129 λ values, $[0 : 0.5 : 5, 6 : 1 : 20, 25 : 5 : 295, 300 : 10 : 500, 550 : 50 : 1500, 1600 : 400 : 4000]$.

4.3. **Reduction by Causal Predictive Models.** In Section 2.1, we proposed to reduce the size of the covariance matrix by first predicting a subset of the stocks that will appear in any of the portfolios on the efficient frontier. One predictive approach we used was Long Short-Term Memory Networks, which is a neural network trained on sequential data. Although causal predictive models were not included in Section 2, we now briefly present

Sparsity	0%	50%	60%	70%	80%	90%	95%	99%
Total Assets	73	195	227	269	319	373	420	495
AvgObj	-0.01183	0.00006	0.00027	0.00044	0.00053	0.00064	0.00070	0.00075
MedObj	-0.00564	0.00027	0.00040	0.00050	0.00056	0.00062	0.00067	0.00069
Avg $\mathbb{V}[\mathbf{R}]$	0.00007	0.00010	0.00011	0.00011	0.00011	0.00011	0.00011	0.00011
Med $\mathbb{V}[\mathbf{R}]$	0.00003	0.00006	0.00007	0.00007	0.00007	0.00007	0.00008	0.00008
Avg $\mathbb{E}[\mathbf{R}]$	0.00072	0.00106	0.00110	0.00113	0.00113	0.00115	0.00116	0.00116
Med $\mathbb{E}[\mathbf{R}]$	0.00042	0.00085	0.00091	0.00094	0.00095	0.00097	0.00099	0.00100
Avg $\tilde{\mathbf{R}}$	0.00456	0.00279	0.00335	0.00381	0.00303	0.00395	0.00431	0.00450
Med $\tilde{\mathbf{R}}$	0.00320	0.00115	0.00181	0.00249	0.00134	0.00255	0.00314	0.00339

TABLE 7. Portfolio Performance for Sparsification by Correlation when Negative Correlations are Present

some numerical results which show their performance and motivate why we favored the three approaches proposed in Section 2.

We have conducted additional numerical testing using Naive Bayes, Logistic Regression and three Artificial Neural Networks (ANN). All three of these approaches use stock features, as shown in Table 8, to predict stocks' inclusion in portfolios on the efficient frontier. We collected features from the Wharton Research Data Services (WRDS) *Compustat* library [31] and *yfinance*. One of the biggest limitations we faced was the limited data from the WRDS features. Starting with nearly 500 stocks, the dataset was reduced to 208 stocks as these features were not reported for all the stocks in the WRDS library. This data was recorded for the dates June 30th 2018 to January 31st 2020.

4.3.1. *Naive Bayes and Logistic Regression:* To avoid over-fitting, we used a ExtraTreesClassifier from sklearn to find the feature importance. Building a minimum of 150 trees and splitting based on entropy, Extra Trees outputted feature importance. Only features in the top 50% percentile were highlighted in Table 8 and included in our model.

By looking at the table, it appears that both methods display high accuracy, with the naive Bayes' portfolio being closer to optimal portfolio. However, the recall and f1-score are low, indicating that this frequently labels good stocks as bad stocks, thus missing out on potential earnings. For Naive Bayes, the stocks that it does identify as good are correctly labeled, and Logistic Regression is likely to misidentify both good and bad stocks.

Another drawback of using Naive Bayes is that the features chosen by ExtraTreesClassifier may not all be independent, which violates the assumptions of the method.

4.4. **Artificial Neural Networks:** This data is between September 30th 2019 to January 1st 2020 for 196 stocks chosen from the S&P 500 index. The features for this are the same as in Naive Bayes and Logistic Regression, however, we used all features not just the bolded ones.

We used a pattern recognition neural network *patternnet* in MATLAB to train. This artificial neural network (ANN) is a feedforward network that uses patterns in the features to make predictions. One of the biggest obstacles with training this data is to split the data

Stock Features	Data Source	Variable Name
Common/Ordinary Shareholders	WRDS	cshr
Market Value - Total - Fiscal	WRDS	mkvalt
Net Income (Loss)	WRDS	ni
Assets - Total	WRDS	at
Book Value Per Share	WRDS	bkvlps
Capital Expenditures	WRDS	capx
Cash	WRDS	ch
Common/Ordinary Stock (Capital))	WRDS	cstk
Dividends - Total	WRDS	dvt
Nonoperating Income (Expense)	WRDS	nopi
Revenue - Total	WRDS	revt
Parent Stockholders' Equity - Total	WRDS	seq
Stockholders Equity - Total	WRDS	teq
Average Return	Yahoo! Finance	–
Return Variance	Yahoo! Finance	–
Return Covariance Median	Yahoo! Finance	–
Return Covariance Max	Yahoo! Finance	–
Return Covariance Min	Yahoo! Finance	–
Trading Volume Median	Yahoo! Finance	–
Trading Volume Max	Yahoo! Finance	–
Trading Volume Minimum	Yahoo! Finance	–
Closing Price Median	Yahoo! Finance	–
Closing Price Max	Yahoo! Finance	–
Closing Price Min	Yahoo! Finance	–

TABLE 8. The list of features. The features that were selected by ExtraTreesClassifier are bolded. Note that statistics on the covariance matrix excluded the diagonal to avoid redundancy with the variance feature. The features' variable names from WRDS are included for the reader to reference on the WRDS website.

	Naive Bayes	Logistic Regression
Accuracy	82.61%	79.71%
Precision	75%	50%
Recall	21.43%	21.43%
F1 Score	33.33%	30%
Average Precision	49%	43%

TABLE 9. Prediction metrics of each model

so the class priors are somewhat similar. There are significantly more stocks in Class 0 than in Class 1, so to prevent over-fitting (predicting that every stock is in Class 0) we included a validation set, increased the L2 regularization parameter, and tried to build a simple model.

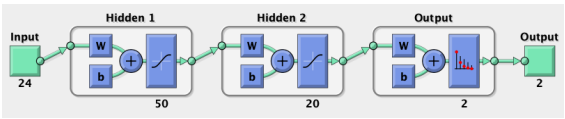

FIGURE 6. ANN selected for the analysis.

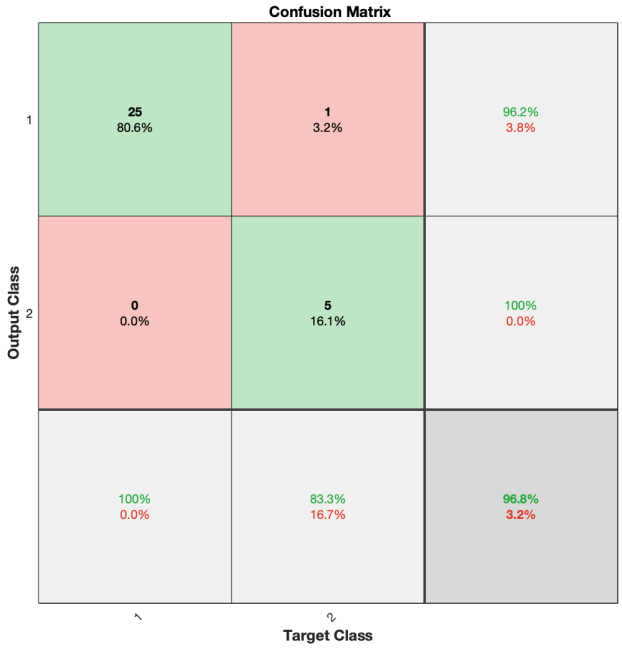

FIGURE 7. Confusion matrix for ANN. 96.8% overall accuracy for 31 stocks

The performance was evaluated by splitting it into sets: 68% training, 16% validation, and 16% testing. If the training set was chosen to be larger than 65%, the model would start to overfit. However, this had the drawback of using only 31 stocks in the testing set.

The regularization was chosen as 0.01, scaled conjugate gradient propagation and least cross entropy objective function were used. The network, as shown in Figure 7, had a hidden layer with 50 nodes. The confusion matrix is given in Figure 8.

The ANN has a 96.2% true positive rate, so it was good predicting the stocks that are in portfolios on the efficient frontier. However, it also has a 83.3% false negative rate which is to be expected given the imbalance between the two class sizes. Moreover, the small testing size (31 assets) also did not allow for a good match to practice.

5. CONCLUSION

Reducing the size of the covariance matrix decreased the number of iterations and CPU runtime, with mixed outcomes for runtime per iteration. The dimension reduction also improved the portfolio's $\mathbb{E}[\mathbf{R}]$ while achieving a similar $\mathbb{V}[\mathbf{R}]$. Both reduction and no reduction showed that $\tilde{\mathbf{R}}$ differs from $\mathbb{E}[\mathbf{R}]$, which is not uncommon since the Markowitz model has been documented to emphasize estimation error, especially in larger portfolios [11, 18].

Sparsification most notably decreased the runtime per iteration. As the covariance matrix grows more sparse, the optimal solution invests in more assets. This requires more basis

changes to compute, which explains the increase in total number of iterations and average runtime. With respect to the portfolio performance, the $\mathbb{V}[\mathbf{R}]$ and $\mathbb{E}[\mathbf{R}]$ both increased as the sparsity increased. This behavior could suggest that removing the off-diagonal values from the covariance matrix may increase diversification in the case of predominantly positively correlated stock returns.

A common issue with the machine learning methods, is that the imbalanced dataset made it challenging for models to distinguish when an asset should be invested in, because Class 1 is much smaller than Class 0. This results in the model labeling nearly every asset as Class 0, thus being impractical as the model does not invest in any stocks. We saw this primarily in Section 4.3, so we opted to take extra precautions, such as using the weighted cross entropy, in Section 2.2 since the weight would impose a penalty, thus forcing Class 1 to be larger. However, Section 4.1 showed that that of the 102 stocks predicted to in Class 1, only 19.6% of these are actually in Class 1 and from the 64 stocks that should be in Class 1, only 31.2% of them were correctly labeled. This suggests that the LSTM network still struggled to distinguish between the two classes. For this reason, we favor LP over LSTM since the predictions from LP were more accurate and implementing it did not require cautious tinkering with hyperparameters as LSTM did. Furthermore, the LSTM network also took around 10 minutes to train, whereas the LP model took one minute.

Overall, the reduction and sparsification yield better optimizer results while achieving similar portfolio $\mathbb{V}[\mathbf{R}]$ and $\mathbb{E}[\mathbf{R}]$. In our testing, a highly efficient solver such as Gurobi exhibited improvement, thus we would expect the improvement to be even more impactful for larger problem sizes. As future work, we will investigate other sparsification procedures that preserve positive semidefiniteness. In addition, we plan to develop a parametric self-dual simplex method for quadratic programming, with specific application to the Markowitz model for portfolio optimization.

6. Appendix

6.1. **Activation and Loss Functions.** The activation and loss functions for the LSTM are provided in Table 10.

6.2. **Example of Sparse Partial Matrix Completion.** Let P be the price matrix where the rows are time steps and the columns are assets.

$$\mathbf{P} = \begin{bmatrix} 2 & 3 & 5 & 2 \\ 6 & 7 & 9 & 3 \\ 4 & 8 & 6 & 5 \\ 5 & 2 & 1 & 2 \\ 2 & 5 & 3 & 6 \end{bmatrix} \tag{11}$$

We compute the return matrix X as in Equation 2.

$$X = \begin{bmatrix} 2 & 1.3 & 0.8 & 0.5 \\ -0.3 & 0.14 & -0.3 & 0.67 \\ 0.25 & -0.75 & -0.83 & -0.6 \\ -0.6 & 1.5 & 2 & 2 \end{bmatrix} \tag{12}$$

Function	Equation
Softmax	$f(x_i) = \frac{\exp(x_i)}{\sum_{j \in K} \exp(x_j)}$
Sigmoid	$f(x) = \frac{1}{1 + \exp(-x)}$
Hyperbolic Tangent	$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$
Cross Entropy Loss	$L(y, \hat{y}) = -\frac{1}{\eta} \sum_{i=1}^{K} y_i \ln(\hat{y}_i)$
Weighted Cross Entropy Loss	$L(\beta, y, \hat{y}) = -\frac{1}{\eta} \sum_{i=1}^{K} \beta_i y_i \ln(\hat{y}_i)$

TABLE 10. Activation and loss functions for training networks. η is the number of samples, K is total number of classes, β_i is the weight for class i, y_i is the actual value for class i, and $\hat{y}_i$ is the prediction value for class i.

This gives us the covariance $cov(X) = \Sigma$ and correlation $corr(X) = \theta$.

$$\Sigma = \begin{bmatrix} 1.37 & 0.27 & -0.08 & -0.47 \\ 0.27 & 1.12 & 1.25 & 0.93 \\ -0.08 & 1.25 & 1.59 & 1.21 \\ -0.47 & 0.93 & 1.21 & 1.14 \end{bmatrix}, \quad \theta = \begin{bmatrix} 1 & 0.21 & -0.05 & -0.38 \\ 0.21 & 1 & 0.93 & 0.82 \\ -0.05 & 0.93 & 1 & 0.9 \\ -0.38 & 0.82 & 0.9 & 1 \end{bmatrix} \tag{13}$$

Sorting the elements of θ in order of magnitude, we get

$$\tau = \{0.05, 0.21, 0.38, 0.82, 0.9, 0.93, 1\} \tag{14}$$

We start with $\tau_{(1)} = 0.05$ and obtain the following matrix.

$$\hat{\Sigma}(\tau_{(1)}) = \begin{bmatrix} 1.37 & 0.27 & 0 & -0.47 \\ 0.27 & 1.12 & 1.25 & 0.93 \\ 0 & 1.25 & 1.59 & 1.21 \\ -0.47 & 0.93 & 1.21 & 1.14 \end{bmatrix} \not\succeq 0 \tag{15}$$

And following the same process for $\tau_{(2:n)}$

$$(16)\quad \hat{\Sigma}(\tau_{(2)}) = \begin{bmatrix} 1.37 & 0 & 0 & -0.47 \\ 0 & 1.12 & 1.25 & 0.93 \\ 0 & 1.25 & 1.59 & 1.21 \\ -0.47 & 0.93 & 1.21 & 1.14 \end{bmatrix} \succeq 0 \quad \hat{\Sigma}(\tau_{(3)}) = \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 1.25 & 0.93 \\ 0 & 1.25 & 1.59 & 1.21 \\ 0 & 0.93 & 1.21 & 1.14 \end{bmatrix} \succeq 0$$

$$(17)\quad \hat{\Sigma}(\tau_{(4)}) = \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 1.25 & 0 \\ 0 & 1.25 & 1.59 & 1.21 \\ 0 & 0 & 1.21 & 1.14 \end{bmatrix} \not\succeq 0 \qquad \hat{\Sigma}(\tau_{(5)}) = \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 1.25 & 0 \\ 0 & 1.25 & 1.59 & 0 \\ 0 & 0 & 0 & 1.14 \end{bmatrix} \succeq 0$$

$$(18)\quad \hat{\Sigma}(\tau_{(6)}) = \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 0 & 0 \\ 0 & 0 & 1.59 & 0 \\ 0 & 0 & 0 & 1.14 \end{bmatrix}$$

This method gives $\{\tau_2, \tau_3, \tau_5, \tau_6, \tau_7\} \in \hat{\tau}$. See that the matrix became indefinite in the process but the final matrix is positive semidefinite. As mentioned, there is no pattern other than $\tau_{(n)} \in \hat{\tau}$.

Now let's sparsify with the partial matrix completion method for the $\tau_{(i)} \notin \hat{\tau}$.

$$\hat{\Sigma}(\tau_{(1)}) = \begin{bmatrix} 1.37 & 0.27 & 0 & -0.47 \\ 0.27 & 1.12 & 1.25 & 0.93 \\ 0 & 1.25 & 1.59 & 1.21 \\ -0.47 & 0.93 & 1.21 & 1.14 \end{bmatrix} \not\succeq 0 \implies \begin{bmatrix} 1.37 & 0.27 & -0.08 & -0.47 \\ 0.27 & 1.12 & 1.25 & 0.93 \\ -0.08 & 1.25 & 1.59 & 1.21 \\ -0.47 & 0.93 & 1.21 & 1.14 \end{bmatrix} \succeq 0$$

$$\hat{\Sigma}(\tau_{(4)}) = \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 1.25 & 0 \\ 0 & 1.25 & 1.59 & 1.21 \\ 0 & 0 & 1.21 & 1.14 \end{bmatrix} \not\succeq 0 \implies \begin{bmatrix} 1.37 & 0 & 0 & 0 \\ 0 & 1.12 & 1.25 & 0.93 \\ 0 & 1.25 & 1.59 & 1.21 \\ 0 & 0.93 & 1.21 & 1.14 \end{bmatrix} \succeq 0$$

For a less sparse matrix, such as $\hat{\Sigma}(\tau_{(1)})$, the matrix completion method loses the sparsity. However, we see that for $\hat{\Sigma}(\tau_{(4)})$, we are able to gain back positive semidefiniteness by adding in only 2 elements. The trade-off of sparsity and positive semidefiniteness is shown in Figure 5.

References

[1] M Arnesano, AP Carlucci, and D Laforgia. Extension of portfolio theory application to energy planning problem–the Italian case. *Energy*, 39(1):112–124, 2012.

[2] Ran Aroussi. yfinance. `https://pypi.org/project/yfinance/`. Accessed on July 21 2020.

[3] Gah-Yi Ban, Noureddine El Karoui, and Andrew EB Lim. Machine learning and portfolio optimization. *Management Science*, 64(3):1136–1154, 2018.

[4] Mark Bennett. Data mining with Markowitz portfolio optimization in higher dimensions. *Available at SSRN 2439051*, 2014.

[5] George Bernard Dantzig. *Linear programming and extensions*, volume 48. Princeton university press, 1998.

[6] Sanjiv Das, Harry Markowitz, Jonathan Scheid, and Meir Statman. Portfolios for investors who want to reach their goals while staying on the mean–variance efficient frontier. *The Journal of Wealth Management*, 14(2):25–31, 2011.

[7] Fernando deLlano Paz, Paulino Martínez Fernandez, and Isabel Soares. Addressing 2030 eu policy framework for energy and climate: Cost, risk and energy security issues. *Energy*, 115:1347–1360, 2016.
[8] Jay L Devore and Kenneth N Berk. *Modern mathematical statistics with applications*. Springer, 2012.
[9] Martín Egozcue, Luis Fuentes García, Wing-Keung Wong, and Ričardas Zitikis. Do investors like to diversify? a study of Markowitz preferences. *European Journal of Operational Research*, 215(1):188–193, 2011.
[10] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.
[11] George M Frankfurter, Herbert E Phillips, and John P Seagle. Portfolio selection: the effects of uncertain means, variances, and covariances. *Journal of Financial and Quantitative Analysis*, pages 1251–1262, 1971.
[12] Fabio D Freitas, Alberto F De Souza, and Ailson R de Almeida. Prediction-based portfolio optimization model using neural networks. *Neurocomputing*, 72(10-12):2155–2170, 2009.
[13] XingYu Fu, JinHong Du, YiFeng Guo, MingWen Liu, Tao Dong, and XiuWen Duan. A machine learning framework for stock selection. *arXiv preprint arXiv:1806.01743*, 2018.
[14] John R Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, 1992.
[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
[16] Carol Anne Hargreaves, Prateek Dixit, and Ankit Solanki. Stock portfolio selection using data mining approach. *IOSR Journal of Engineering*, 3(11):42–48, 2013.
[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[18] J David Jobson and Bob Korkie. Estimation for Markowitz efficient portfolios. *Journal of the American Statistical Association*, 75(371):544–554, 1980.
[19] Florian Kellner and Sebastian Utz. Sustainability in supplier selection and order allocation: Combining integer variables with Markowitz portfolio theory. *Journal of Cleaner Production*, 214:462–474, 2019.
[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[21] Hui-Shan Lee, Fan-Fah Cheng, and Shyue-Chuan Chong. Markowitz portfolio theory and capital asset pricing model for kuala lumpur stock exchange: A case revisited. *International Journal of Economics and Financial Issues*, 6(3S), 2016.
[22] Henry Markowitz. Portfolio selection. *The Journal of Finance*, 7:77–91, 1952.
[23] John M Mulvey. Machine learning and financial planning. *IEEE Potentials*, 36(6):8–13, 2017.
[24] Bakhtiar Ostadi, Omid Motamedi Sedeh, and Ali Husseinzadeh Kashan. Risk-based optimal bidding patterns in the deregulated power market using extended Markowitz model. *Energy*, 191:116516, 2020.
[25] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
[26] Mohammad Maholi Solin, Andry Alamsyah, Brady Rikumahu, and Muhammad Apriandito Arya Saputra. Forecasting portfolio optimization using artificial neural network and genetic algorithm. In *2019 7th International Conference on Information and Communication Technology (ICoICT)*, pages 1–7. IEEE, 2019.
[27] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.
[28] Van-Dai Ta, Chuan-Ming Liu, and Direselign Addis Tadesse. Portfolio optimization-based stock prediction using long-short term memory network in quantitative trading. *Applied Sciences*, 10(2):437, 2020.
[29] Robert J Vanderbei. *Linear programming: foundations and extensions*, volume 285. Springer Nature, 2020.
[30] Jian Wang and Junseok Kim. Applying least squares support vector machines to mean-variance portfolio analysis. *Mathematical Problems in Engineering*, 2019, 2019.

[31] Wharton Research Data Services. Compustat. "https://wrds-www.wharton.upenn.edu/demo/compustat". Accessed on June 11 2020 via Python.

[32] Shuang Zhang, Tao Zhao, and Bai-Chen Xie. What is the optimal power generation mix of China? an empirical analysis using portfolio theory. *Applied Energy*, 229:522–536, 2018.