MACHINE LEARNING BASED EFFICIENT FRONTIERS AND MODERN PORTFOLIO THEORY

MTH392 Project Report

Author:
Sai Praneeth Donthu
Supervisor:
Prof. Amit Mitra

Abstract

In the field of portfolio management the leading method for managing assets in a portfolio comes from the **Modern Portfolio Theory**. Introduced by Harry Markowitz in the 1950s the method is still used to this day for portfolio management. However the theory makes use of historical data of assets to make predictions, however the real world analysis has shown us that the stock market rarely follows historical trends. In the last few decades or so, machine learning methods came to the forefront with ML techniques being used in different fields. In this project we thus try to use ML techniques along with MPT to see how we can improve the classsical methods.

Contents

1 Introduction **4**

2 Modern Portfolio Theory **5**
2.1 The Efficient Frontier and the Sharpe Ratio 6
2.2 Finding the best Portfolio on the Efficient Frontier 6
2.3 Implementation 7
2.4 Criticisms 8

3 A Machine Learning Generated Efficient Frontier **9**
3.1 Methodology 9
3.2 Training Procedure 10

4 Long Short Term Memory Model **11**

5 Future Work **12**

6 Appendix **13**
6.1 Code for Figure 3 13
6.2 Code for Figure 4: Efficient Frontier 13
6.3 Logistic Regression Model 14
6.4 Training Procedure of Model 15
6.5 LSTM model Build 15

1 Introduction

We carry out this report in four parts. In the first part we study various literature available on the widely popular **Modern Portfolio Theory** and analyze it ourself. We then make an attempt at implementing the method on real life data. After understanding some of the shortcomings and criticisms of the theory we then attempt two different methods machine learning based methods to build an machine learning based efficient frontier. For this we again read material available on the topic and then go on to implement these methods to see how they perform

2 Modern Portfolio Theory

Modern portfolio theory or mean variance analysis, as introduced by Harry Markowitz for which he was awarded the Nobel Prize in Economic Sciences, is a method used to assemble a portfolio or set of assets so as to maximize the expected return at a certain level of risk. Variance is used as a measure of risk. A central idea of MPT is the idea of diversification, or the idea that investing in multiple assets rather than only one is less risky than investing in say only one asset.

The framework assumes that investors are risk averse, or that given two portfolios with the same return, investors would choose the portfolio with the lesser risk. It also assumes that the returns are normally distributed.

Some definitions firstly. **Portfolio Return** is the proportion weighted combination of the returns of the assets which is given as the expected return of the portfolio:

$$\mathbb{E}(R_p) = \sum_i w_i \mathbb{E}(R_i) \tag{1}$$

$$= \mathbf{w}'\mathbf{R} \tag{2}$$

Portfolio Volatility is the risk associated with the portfolio, and is the standard deviation of the portfolio given as:-

$$\sigma_p = \sum_i w_i^2 \sigma_i^2 + \sum_i \sum_j w_i w_j \sigma_{ij} \tag{3}$$

$$= \mathbf{w}'\Sigma\mathbf{w} \tag{4}$$

Where $\mathbf{w} \in \mathbb{R}^d$ are the weights that are allocated to d assets in our universe and w_i are the weights associated to the ith asset. σ_i and σ_{ij} are the variance and covariance of the historical returns.
Modern portfolio theory gives us that the optimal weights are those satisfying the optimisation problem:

$$\min_{\mathbf{w}} \mathbf{w}'\Sigma\mathbf{w} \tag{5}$$

$$\text{s.t} \quad \mathbf{w}'\mathbf{R} = m \tag{6}$$

$$\mathbf{w}'\mathbf{1} = 1 \tag{7}$$

Where Σ is the Covariance Matrix, $\mathbf{R}$ is the mean returns vector, m is the target mean and $\mathbf{1}$ is the vector of ones. In practice to calculate Σ and $\mathbf{R}$ we must rely on historical data. Noticing carefully, the above optimization problem minimizes the risk for a certain set target return, the optimization problem may have various other forms depending on the requirements. The above optimization problem for the given parameter m can be solved to reach the following system using Lagrange multipliers.

$$\begin{bmatrix} 2\Sigma & -\mathbf{R} & -\mathbf{1} \\ \mathbf{R}' & 0 & 0 \\ \mathbf{1}' & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ m \\ 1 \end{bmatrix} \tag{8}$$

2.1 The Efficient Frontier and the Sharpe Ratio

On the plot of the expected returns against the risk associated, the efficient frontier is the set of portfolios which have no portfolio of higher expected returns but with the same risk level.

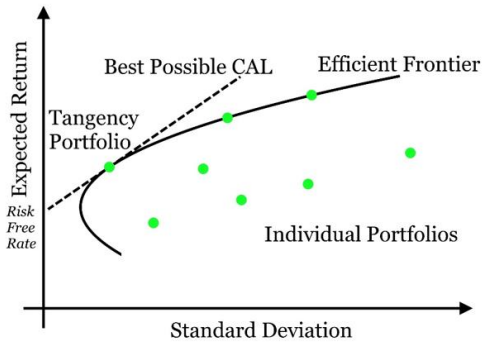


Figure 1: The Efficient Frontier

For a given risk, λ, the efficient frontier can be found by minimizing the expression:-

$$\mathbf{w}'\Sigma\mathbf{w} - \lambda\mathbf{w}'\mu$$

Where $\lambda \in [0, \infty)$, the other symbols are as defined before. The Frontier is therefore defined parametrically on the parameter λ. $\lambda = 0$ results in a portfolio with 0 risk and $\lambda \to \infty$ gives a portfolio with return and risk far out tending to infinity.
"How good" a portfolio is is measured by a metric known as the Sharpe Ratio defined as:

$$S_p = \frac{\mathbb{E}R_p - R_f}{\sigma_p} \tag{9}$$

Where $\mathbb{E}R$ is the expected return of the portfolio, R_f is the risk free rate and σ is the standard deviation of the portfolio. R_f is the risk free rate which is defined as the return on a theoretical investment with zero risk In the above figure, the tangency portfolio is the portfolio with the best sharpe ratio.

2.2 Finding the best Portfolio on the Efficient Frontier

On the plot below, the best portfolio lies on the line ray $R_f - -B$.

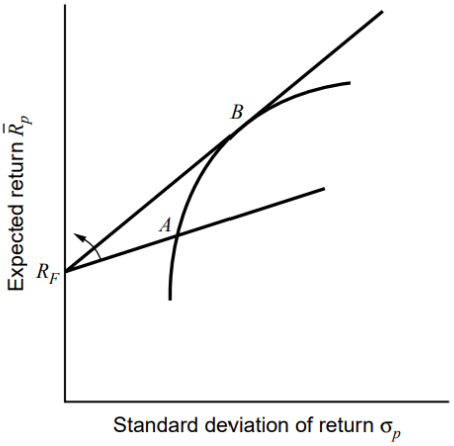


Figure 2: The Efficient Frontier with the best Portfolio

Hence we find the point B such that the line connecting R_f and B has the highest slope, or in other words, the greatest Sharpe's ratio, we thus get the constrained optimisation problem:-

$$\max_{\mathbf{w}} S_p \tag{10}$$

$$\sum_i w_i = 1 \tag{11}$$

Further, R_f can be written as $\sum_i X_i R_f$ since the sum $\sum_i X_i = 1$ hence we would need to maximize the following:

$$\max_{\mathbf{w}} \frac{\sum_i w_i(R_p - R_f)}{[\sum_i w_i^2\sigma_i^2 + \sum_i \sum_j w_i w_j \sigma_{ij}]^{1/2}} \tag{12}$$

We then just solve this by differentiating w.r.t. w_i's i.e $\frac{dS_p}{dw_i} = 0$. Note that we just solve it in an unconstrained fashion. This works since the equation is a homogeneous equation of degree zero. We get:-

$$\frac{dS_p}{dw_i} = -\lambda \sum_{j=1}^{n} w_j \sigma_{ji} + R_i - R_f \tag{13}$$

$$\lambda = \frac{\sum_i w_i(R_p - R_f)}{\sum_i w_i^2\sigma_i^2 + \sum_i \sum_j w_i w_j \sigma_{ij}} \tag{14}$$

Equating to zero we get the following system of equations which can be easily solved for $1 \leq i \leq n$:-

$$R_i - R_f = \lambda \sum_{j=1}^{n} w_j \sigma_{ji} \tag{15}$$

2.3 Implementation

On python, the dedicated library PyPortfolioOpt helps implementation of classical portfolio optimization methods including classical mean variance optimization techniques.

In our implementation of this method, we took stock data from the S&P500 dataset from yfinance for the past 25 years. After calculating the mean and the covariance matrix of the data, we randomly choose weights over 50000 iterations and plot each point. The boundary of the scatter plot is our efficient frontier.[1]

Another method we used was to solve the optimization problem and plot the efficient frontier which is parametrically dependent on λ.

Below is the result of our implementation on 28 assets from the S&P500 stock index. The Red Mark corresponds to the portfolio with the highest Sharpe Ratio. The Green Mark corresponds to the portfolio with the least risk associated. Note that both images are the same but with different scalings.

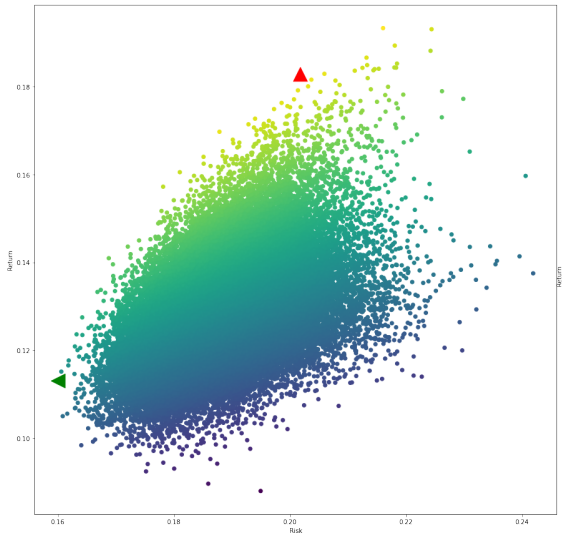


Figure 3: Return vs Risk of 50000 weighted Portfolios

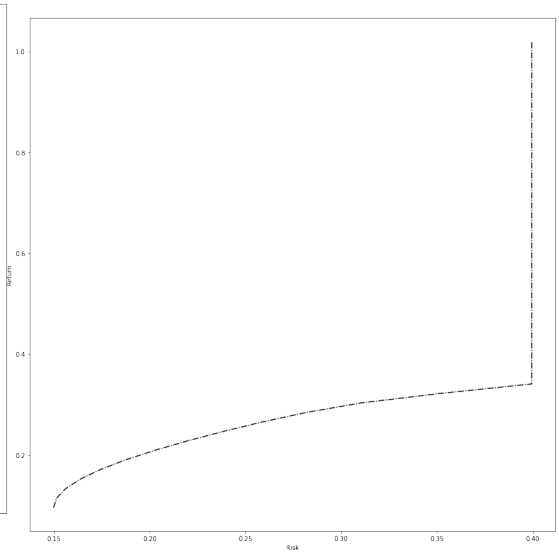


Figure 4: The Efficient Frontier

2.4 Criticisms

While the mean variance analysis method is a great tool for portfolio management, it has had its share of criticisms. It makes many assumptions, the most criticised assumption is that the markets behave rationally. However in the real world, it is well known that the markets behave in no rational manner. We thus look for Machine Learning bases solutions to see if we can come up with better alternatives.

[1] for the codes refer the appendix

3 A Machine Learning Generated Efficient Frontier

In the following section we focus on the method introduced in (1) and implement a similar procedure in python to generate the efficient frontier, we then compare our results with the mean variance generated efficient frontier. The above paper focuses on one high risk asset such as the S&P500 and one low risk bond. We adopt a similar strategy on multiple stocks from the S&P500.

3.1 Methodology

The main ideology of the below ML algorithm is to map the information set given by X_t at a given time t to either +1 or -1 depending on whether the return is above a certain threshold or not.
We define the algorithm more formally.

Let $\tau \in \mathbb{R}$. At a point $t+1$, let $Y_{t+1|t}$ be such that

$$Y_{t+1|t} = \left\{ \begin{array}{ll} +1, & \text{if } r_{t+1|t} \geq \tau \\ -1, & \text{if } r_{t+1|t} < \tau \end{array} \right\} \tag{16}$$

Here $r_{t+1|t}$ is the return of the stock at time $t+1$. For our implementation we used $0\%, -0.5\%, -1\%$ for our values of τ.

We then use a basic logistic regression model to predict the conditional probability $\mathbb{P}_t(Y_{t+1} = +1|X_t)$. The above paper uses the following formulation which uses the loading coefficients β_t^+ and β_t^-:-

$$\mathbb{P}_t(Y_{t+1} = +1|X_t) = \frac{exp(X_t'\beta_t^+)}{exp(X_t'\beta_t^+) + exp(X_t'\beta_t^-)} \tag{17}$$

The above can be further trivially simplified to the sigmoid function to get:

$$\mathbb{P}_t(Y_{t+1} = +1|X_t) = \sigma(X_t'(\beta_t^+ - \beta_t^-)) \tag{18}$$

where,

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{19}$$

Thus our weights of the logistic regression model are simply $w_t = \beta_t^+ - \beta_t^-$. Training over the given data over the information set gives us a mapping function

$$\hat{\pi}_{t+1|t} = P_t(Y_{t+1} = +1|X_t) \tag{20}$$

We apply the same procedure for all of our n assets in our universe to obtain mapping functions $\hat{\pi}_{i,t+1|t}$ for the ith asset for $1 \leq i \leq n$.

Generalizing the approach from the paper to n such high risk assets, for some $\kappa \in [0, 1]$

we allocate weights $x_{i,t}(\kappa)$ in the following manner to arrive at a portfolio:

$$x_{i,t}(\kappa) = \frac{\mathbb{I}(\hat{\pi}_{i,t+1|t} \geq \kappa)}{\sum_{j=1}^{n} \mathbb{I}(\hat{\pi}_{j,t+1|t} \geq \kappa)} \tag{21}$$

3.2 Training Procedure

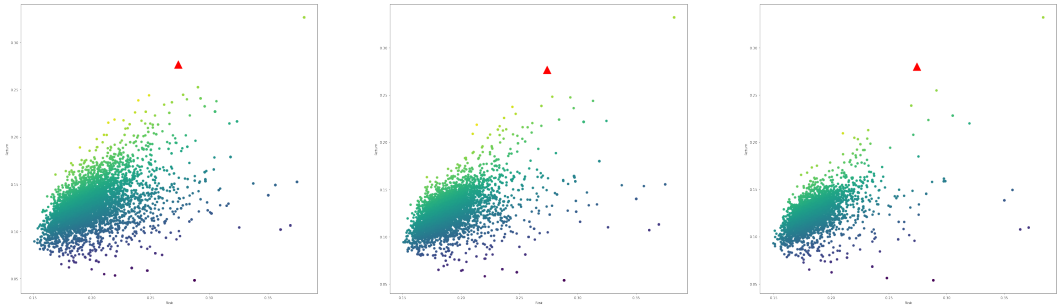
In this section we describe the implementation and the training procedure of the logistic regression model.[2]

For the data we again use 25 years data of 28 stocks from the S&P stock index. For the information set X_t we shall be using the daily volume, the daily return data and the rolling mean data of 25 days. We also generate Y_t based on 16.

Our model will be a logistic regression model with elastic new regularization with l1 ratio equal to 0.5 and C equal to 1. This means that both L1 and L2 regularization are given equal weights, i.e, 0.5.

For the training we first set a window size of 250 days. We then start the training process by training the model on the window, i.e, the first 250 days. We then test the model for the next 7 days and store the signal output. We then shift the window by 7 days and continue the similar process till the end of the data. We repeat this process for different values of $\tau = 0\%, -0.5\%, -1\%$. Training and testing gives us the values of $\hat{\pi}_t + 1|t$ for all the data points after the first 250 days. The weights are then calculated as per (21).

For plotting the machine learning efficient frontier we use the same procedure as plotting the MPT based efficient frontier; we compute the annual mean and the annual volatility and plot each of the portfolios associated to the predicted weights. Below are the plots for the return vs volatilty for the three values of τ.



(a) $\tau = 0\%$ (b) $\tau = -0.5\%$ (c) $\tau = -1\%$

Figure 5: Plots of the weighted portfolios for different values of τ and $\kappa = 95\%$

[2]for full code see Appendix

4 Long Short Term Memory Model

Another model we use is an Long Short Term Memory Model, since LSTM models are good for sequence data since they have the ability to store an arbitrary amount of previous data, which can be highly useful for time series data, in this case asset data. Thus in this section we build an LSTM model to make predictions of the asset. We then compute the returns of the predicted data and we plot the return vs risk plots in a similar fashion as computed in section 2. To create the features for training we use a time step of a 100 days. This means to predict a particular day's prediction we use the previous 100 days information. The parameters of the LSTM model are as followed. [3]:-

- input_size = 1
- hidden_size = 1
- num_layers = 1
- num_classes = 1
- learning_rate = 0.001
- num_epochs = 10000
- loss_function = MSELoss()
- optimizer = Adam

The plot of return vs risk for 5000 weighted portfolios is shown below:-

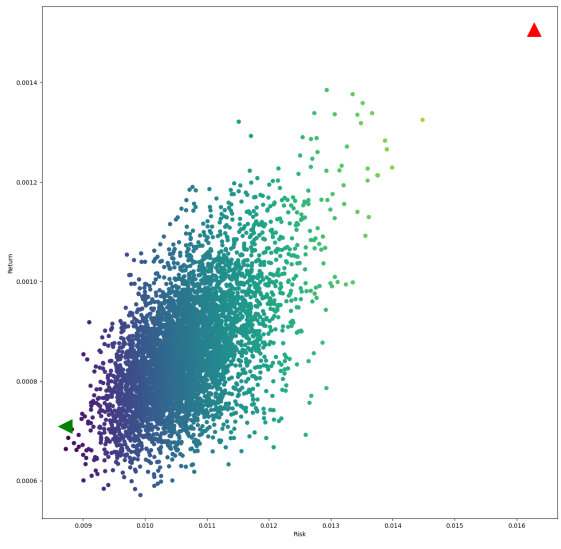


Figure 6: Return vs Risk of 5000 weighted Portfolios for the LSTM predicted returns

[3]The code for the model is in the Appendix

5 Future Work

We have seen a logistic regression model at work to build a machine learning based efficient frontier. Different other machine learning models can be used to train on the data in a similar fashion to predict the weights β_t^+ and β_t^- and a similar approach used in section 3 to construct an Efficient Frontier. From the current work we can conclude that such a ML based efficient frontier that is based on a training size from the previous 50 weeks is more sensitive to sudden changes in stock index data and will result in a safer portfolio than that given by the modern portfolio theory. However this is again dependent on the training size. Further research can be done to determine to what extent such ML methods are better than MPT in extreme cases.

6 Appendix

This Appendix contains the important implementation snippets from different section

6.1 Code for Figure 3

```
return_pred = []
weight_pred = []
std_pred = []

for i in range(50000):
    rand_mat = np.array(np.random.dirichlet(np.ones(len(data_return.columns)), size = 1)[0])
    port_std = np.sqrt(np.dot(rand_mat.T, np.dot(data_return.cov(), rand_mat))) * np.sqrt(252)
    port_return  = np.dot(data_mean[data_return.columns], rand_mat)
    return_pred.append(port_return)
    std_pred.append(port_std)
    weight_pred.append(rand_mat)

final_outputs = pd.DataFrame({'weights:weight_pred,'return':return_pred,'stddev':std_pred})
final_outputs['sharpe']=(final_outputs['return']-risk_free)/final_outputs['stddev']
final_outputs

plt.figure(figsize=(15,15))

plt.scatter(final_outputs.stddev,final_outputs['return'],c=final_outputs.sharpe)
plt.xlabel('Risk')
plt.ylabel('Return')

plt.show
```

6.2 Code for Figure 4: Efficient Frontier

```
def efficient_portfolio_target(target):

    constraints = ({'type':'eq','fun': lambda x: portfolio_return(x)- target},
                   {'type' :'eq','fun': lambda x: np.sum(x)-1})
    w0          = np.random.dirichlet(np.full(n_assets,0.05)).tolist()
    bounds      = ((0,1),)*n_assets

    return minimize(portfolio_volatility,w0, method = 'SLSQP',
                    bounds      = bounds,
                    constraints = constraints)

def efficient_frontier(return_range):
    return [efficient_portfolio_target(ret) for ret in return_range]

def portfolio_volatility(weight):
    return np.sqrt(np.dot(weight.T,
                          np.dot(sigma,weight)))*np.sqrt(252)

def portfolio_return(weight):

    return np.sum(mean_returns*weight)*252
```

```
def portfolio_performance(weight):
    return_p = portfolio_return(weight)
    vol_p    = portfolio_volatility(weight)
    return return_p, vol_p
def min_vol():

    n_assets           = data_return.shape[1]
    weight_constraints = ({'type':'eq','fun': lambda x: np.sum(x)-1})
    w0                 = np.random.dirichlet(np.full(n_assets,0.05)).tolist()
    bounds             = ((0,1),)*n_assets

    return minimize(portfolio_volatility,w0,method='SLSQP',
                    bounds      = bounds,
                    constraints = weight_constraints)

sharpe_maximum      = max_sharpe_ratio()
return_p,vol_p      = portfolio_performance(sharpe_maximum['x'])
min_volatility      = min_vol()
return_min,vol_min  = portfolio_performance(min_volatility['x'])


plt.figure(figsize =(15,15))
target               = np.linspace(return_min,1.02,50)
efficient_portfolios = efficient_frontier(target)
plt.plot([i.fun for i in efficient_portfolios], target, linestyle ='dashdot', color ='black',
         label='Efficient Frontier')

plt.xlabel('Risk')
plt.ylabel('Return')
```

6.3 Logistic Regression Model

```
def model_train(df_train,df_test,target_train,target_test):

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(df_train)
    X_test_scaled = scaler.transform(df_test)

    logreg =LogisticRegression(penalty='elasticnet',solver='saga',l1_ratio=0.5,C=1,max_iter=1000)

    logreg.fit(X_train_scaled, target_train)

    target_test_pred = logreg.predict(X_test_scaled)
    accuracy = accuracy_score(target_test, target_test_pred)
    precision = precision_score(target_test, target_test_pred)
    recall = recall_score(target_test, target_test_pred)
    f1 = f1_score(target_test, target_test_pred)
    conf_matrix = confusion_matrix(target_test, target_test_pred)

    probas_test = logreg.predict_proba(X_test_scaled)
    probas_test_pos = probas_test[:,1]
    probas_test_neg = probas_test[:,0]

    return probas_test, probas_test_pos, target_test_pred
```

6.4 Training Procedure of Model

```
for column in columns:
    df = pd.DataFrame()
    df['Return'] = data_return[column]
    df['Volume'] = data_volume[column]
    df['MA25'] = data_MA25[column]

    target = data_signal[column]

    print(column)

    index = 250
    while index + 7<= len(df):

        sub_df = df[index-250:index]
        sub_target = target[index-250:index]
        sub_df_test = df[index:index+7]
        sub_target_test = target[index:index+7]

        probas_test,probas_test_pos,target_test_pred =model_train(sub_df,sub_df_test,
        sub_target,sub_target_test)

        i = index
        ind = 0
        while i < index + 7:
            test_proba[column][i] = probas_test_pos[ind]
            i = i + 1
            ind = ind + 1
        index = index + 7
```

6.5 LSTM model Build

```
class LSTM_model(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(LSTM_model, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.num_classes = num_classes

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h0 = Variable(torch.zeros(self.num_layers, x.size(0),
                                  self.hidden_size)).to(device)
        c0 = Variable(torch.zeros(self.num_layers, x.size(0),
                                  self.hidden_size)).to(device)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

def build_model(input_size=1, hidden_size=1, num_layers=1, num_classes=1, learning_rate=0.001):
```

```
    model = LSTM_model(input_size, hidden_size, num_layers, num_classes).to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    loss_function = torch.nn.MSELoss()
    return model, optimizer, loss_function
```

References

[1] B. Clark, Z. Feinstein, and M. Simaan, "A machine learning efficient frontier," *Operations Research Letters*, vol. 48, no. 5, pp. 630–634, Sep. 2020, publisher Copyright: © 2020 Elsevier B.V.