J. Korean Soc. Math. Educ. Ser. B: Pure Appl. Math.
https://doi.org/10.7468/jksmeb.2017.24.4.211
Volume 24, Number 4 (November 2017), Pages 211–226

ISSN(Print) 1226-0657
ISSN(Online) 2287-6081

COMPARATIVE ANALYSIS ON MACHINE LEARNING MODELS FOR PREDICTING KOSPI200 INDEX RETURNS

Bonsang Gu [a] and Joonhyuk Song [b,*]

Abstract. In this paper, machine learning models employed in various fields are discussed and applied to KOSPI200 stock index return forecasting. The results of hyperparameter analysis of the machine learning models are also reported and practical methods for each model are presented. As a result of the analysis, Support Vector Machine and Artificial Neural Network showed a better performance than k-Nearest Neighbor and Random Forest.

1. Introduction

Various research has been actively conducted to utilize machine learning to predict financial markets. As machine learning models, k-Nearest Neighbor (k-NN), random forest, Support Vector Machine (SVM) and Artificial Neural Network (ANN) were used for financial market prediction. In the study of k-NN, Teixeira & Oliveira [13] showed higher performance than buy-and-hold strategy by using technical analysis index and k-NN. Huang et al. [5] analyzed the index of the Istanbul Stock Exchange and showed that SVM has higher prediction power than ANN. In the study of Kim [7], SVM showed a higher predictive power than ANN in KOSPI index. ANN is one of the most actively studied machine learning techniques for forecasting financial markets, including stock (Yudong & Lenan [14]) and FX (Dunis et al. [3]).

As interest in machine learning has increased, there have been various studies using machine learning models in Korea. Many studies show that the use of machine learning models improves predictability. A genetic algorithm-based artificial intelligence prediction model of Lee [8] showed that the predictive power is meaningful. In addition, there have been a study that used k-NN algorithms (Kim [6]) to

Received by the editors September 18, 2017. Accepted October 08, 2017.
2010 *Mathematics Subject Classification.* 91G70, 91G80.
Key words and phrases. machine learning, support vector machine, artificial neural network, random forest, k-nearest neighbor.
*Corresponding author.

improve prediction performance. Park et al. [10] developed a model to predict using SVM, lasso regression, ANN. In this study, SVM showed higher predictive result than ANN in training data, and ANN has better predictive result in test data.

In recent years, interest in robo-adviser services has increased in the Korean financial market, and there is an increasing tendency to incorporate machine learning techniques into the financial market. Interest in research to predict the stock market using machine learning is increasing. However, there is still a lack of research using machine learning techniques in Korea. Therefore, it is necessary to carry out a comprehensive analysis on various machine learning techniques. In this study, the four major machine learning models are explained, and the results of the stock index prediction according to each machine learning model are analyzed to compare the advantages and disadvantages of each model.

2. Machine Learning Models for Prediction

2.1. k-Nearest Neighbors The k-Nearest Neighbor (k-NN) algorithm is one of the simplest machine learning methods used in pattern recognition. When there are many objects, k nearest neighbors are used to classify objects. The k-NN finds attributes similar to objects and classifies them by majority vote. Generally, the k closest training data in the multidimensional feature space is input, and the objects assigned to the most common items among the k nearest objects are classified by a majority vote. The most widely used distance measure for k-NN is the Euclidean distance scale, and the Euclidean distance is calculated as the square root of each axis. Depending on the application, various distance measures such as Manhattan distance can be used.

Euclidean distance:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{2.1}$$

Manhattan distance:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i| . \tag{2.2}$$

2.2. Random Forest A random forest is an ensemble learning technique based on a multidimensional decision tree. The main concept of random forest was first proposed by Ho [4]. The random forest technique is to create multiple decision trees

on randomly selected subspaces for the feature space. The subspace-based classification method can improve the accuracy while avoiding overfitting as compared with a single-tree classification method. Random forest was established by Breiman [1], famous for its research on bagging (bootstrap aggregating). In Breiman's paper, a random forest is defined as a classifier consisting of a collection of classifiers in a tree structure composed of independent random vectors. The original definition of random forest in Breiman's paper is as follows.

The original definition of random forest :
A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, ...\}$ *where the* $\{\Theta_k\}$ *are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at x.*

2.3. Support Vector Machine The Support Vector Machine (SVM) was first proposed by Boser et al. [2] to find a way to maximize the margins, the distance between categories. SVM is an algorithm that creates a hyperplane that maximizes margins. First, a virtual hyperplane is established and a margin function is derived using a vector representing the distance between the nearest point and the hyperplane. Finally, SVM find the hyperplane that maximizes the margin function.

The function $f(x)$, which represents a hyperplane that divides the D-dimensional space by half, is defined as follows for the D-dimensional vector $\boldsymbol{x}$ and weight $\boldsymbol{w}$

$$f(\boldsymbol{x}) = x_1 w_1 + x_2 w_2 + \cdots + x_D w_D + b = \boldsymbol{x}\boldsymbol{w} + b = 0. \tag{2.3}$$

We can generate $\{\boldsymbol{x_i}, t_i\}$ pairs with the property $t_i \in \{1, -1\}$ for pattern vector $\boldsymbol{x_i}$ with $i = 1, \cdots, N$. If we set the target value t_i according to the value of $f(\boldsymbol{x_i})$,

$$\{\boldsymbol{x_1}, t_1\}, \{\boldsymbol{x_2}, t_2\}, \cdots, \{\boldsymbol{x_N}, t_N\} \text{ where } t_i = \begin{cases} 1 & \text{if } f(\boldsymbol{x_i}) > 0 \\ -1 & \text{if } f(\boldsymbol{x_i}) < 0 \end{cases}. \tag{2.4}$$

These pairs satisfy the following conditions

$$t_i f(\boldsymbol{x_i}) = t_i(\boldsymbol{x_i}\boldsymbol{w} + b) > 0. \tag{2.5}$$

We define h_1 and h_2 in both sides of the hyperplane $f(\boldsymbol{x})$ as

$$h_1(\boldsymbol{x_1}) = \boldsymbol{x_1}\boldsymbol{w} + b = 1 \tag{2.6}$$

$$h_2(\boldsymbol{x_2}) = \boldsymbol{x_2}\boldsymbol{w} + b = -1. \tag{2.7}$$

In order for x_i to exist outside adjacent to h_1 and h_2, the following must be satisfied

$$t_i(\boldsymbol{x_i}\boldsymbol{w} + b) \geq 1. \tag{2.8}$$

If we find the distance between h_1 and h_2 here,

$$\text{dist}(h_1(\boldsymbol{x_1}), h_2(\boldsymbol{x_2}))) = (\boldsymbol{x_1} - \boldsymbol{x_2})\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}(\boldsymbol{x_1}\boldsymbol{w} - \boldsymbol{x_2}\boldsymbol{w}) = \frac{2}{\|\boldsymbol{w}\|}. \tag{2.9}$$

The problem of maximizing the distance $(2/\|\boldsymbol{w}\|)$ between h_1 and h_2 is equivalent to the following optimization problem.

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{subject to } t_i(\boldsymbol{x_i}\boldsymbol{w} + b) \geq 1,\ i = 1, ..., N. \tag{2.10}$$

This optimization problem can be solved using the Lagrange function as follows.

$$L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \lambda_i(t_i f(\boldsymbol{x_i}) - 1). \tag{2.11}$$

where $\lambda_i \geq 0$. $f(\boldsymbol{x})$, which determines hyperplanes, need not be restricted to linear properties. We can replace $\boldsymbol{x_i}$ with the predefined function $\varphi(\boldsymbol{x})$, or use a kernel function $K(\boldsymbol{x}, \boldsymbol{x'})$ in dual space.

$$f(\boldsymbol{x}) = \sum_{j=1}^{D} w_j\varphi_j(\boldsymbol{x}) + b \tag{2.12}$$

$$f(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k K(\boldsymbol{x_k}, \boldsymbol{x}) + b. \tag{2.13}$$

α_k is an adjustable parameter and $\boldsymbol{x_k}$ is a training pattern. K is a predefined function.

For symmetric kernel K,

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \sum_j \varphi_j(\boldsymbol{x})\varphi_j(\boldsymbol{x'}). \tag{2.14}$$

$f(\boldsymbol{x})$ can be expressed in the following form by combining (2.13) and (2.14).

$$f(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k K(\boldsymbol{x_k}, \boldsymbol{x}) + b = \sum_{k=1}^{N} \alpha_k \sum_{j=1}^{D} \varphi_j(\boldsymbol{x_k})\varphi_j(\boldsymbol{x}) + b \tag{2.15}$$

$$= \sum_{j=1}^{D} \left[\sum_{k=1}^{N} \alpha_k\varphi_j(\boldsymbol{x_k})\right] \varphi_j(\boldsymbol{x}) + b. \tag{2.16}$$

Therefore, (2.12) and (2.13) are the same type of function when w_j is as follows.

$$w_j = \sum_{k=1}^{N} \alpha_k \varphi_j(\boldsymbol{x_k}). \tag{2.17}$$

w_j is called the direct parameter and α_k is called the dual parameter.

Various kernel methods are used to create hyperplanes. There is a linear kernel method, which creates a hyperplane at a higher level in order to maximize the margin for classification. To overcome the limitations of the linear kernel, the proposed nonlinear kernel is typically a polynomial kernel and a radial basis kernel. With the kernel approach, you can simply calculate with a dot product, avoiding expensive computation processes and achieving the desired results in a relatively short time.

Polynomial function :

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \left(\boldsymbol{x} \cdot \boldsymbol{x'} + 1\right)^d. \tag{2.18}$$

Radial basis function :

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\gamma \left\|\boldsymbol{x} - \boldsymbol{x'}\right\|^2\right). \tag{2.19}$$

2.4. Artificial Neural Network The Artificial Neural Network (ANN) was initiated by attempting mathematical modeling of biological brains. In the basic mathematical modeling, input the part corresponding to the synapse of the brain $w_i \cdot x_i$ accumulate the electrons and charge them into the cell membrane by summing the variables. The activation function $f(x)$ is defined as the part of the neuron that determines whether to generate an output signal based on the input signal. In the modeling of McCulloch and Pitts [9], a neuron is a binary threshold device that determines whether to generate another output by the sum of the signals of each neuron.

Activation function:

$$f(x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_i \cdot x_i + b > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2.20}$$

The perceptron algorithm for learning neural networks was first introduced by Rosenblatt [11]. In the perceptron, the input weight of the neuron, $w_{i,j}$, has been improved by learning. t_j is the target output of the j^{th} neuron, y_j is the actual output of the j^{th} neuron, and η is the learning rate.

$$w_{i,j} \leftarrow w_{i,j} + \eta\left(t_i - y_j\right) \cdot x_i. \tag{2.21}$$

The backpropagation algorithm was discovered by Rumelhart et al. [12]. Using the backpropagation algorithm and the gradient descent method, we were able to modify the weights while minimizing errors. This has enabled multi-layer perceptron. In the case of multi-layer perceptron, the activation function usually uses the sigmoid function as follows.

$$\phi(x) = \frac{1}{(1+e^{-x})}. \tag{2.22}$$

Multiple perceptrons consist of three or more layers with one or more hidden layers. Each layer is connected to every node in the next layer. For input $\boldsymbol{x^i}$, let n^{th} layer be $\boldsymbol{a}^n$ and the matrix of weights connected to the next layer $\boldsymbol{W}^n$. Each layer $\boldsymbol{a}^n$ can be defined as follows.

$$\boldsymbol{a}^1 = \boldsymbol{x^i} \tag{2.23}$$

$$\boldsymbol{a}^n = \phi(\boldsymbol{W}^n \boldsymbol{a}^{n-1}). \tag{2.24}$$

The final output value can be defined as $h_{\boldsymbol{W}}(\boldsymbol{x^i})$ as follows.

$$h_{\boldsymbol{W}}(\boldsymbol{x^i}) = \boldsymbol{a}^n = \phi(\boldsymbol{W}^n \boldsymbol{a}^{n-1}) = \phi(\boldsymbol{W}^n \phi(\boldsymbol{W}^{n-1} \boldsymbol{a}^{n-2})) = \cdots .$$

Using $h_{\boldsymbol{W}}(\boldsymbol{x^i})$, the cost function is calculated in the following form.

$$J(\boldsymbol{W}, \boldsymbol{x}^i, \boldsymbol{y}^i) = \frac{1}{2} \sum_{m=1}^{M} (h_{\boldsymbol{W}}(\boldsymbol{x}_m^i) - y_m^i)^2. \tag{2.25}$$

This cost function is a simple quadratic cost function(2.25), and the most widely used is the cross entropy cost function(2.26)

$$J(\boldsymbol{W}, \boldsymbol{x}^i, \boldsymbol{y}^i) = \frac{1}{2} \sum_{m=1}^{M} \left[y_m^i \log(h_{\boldsymbol{W}}(\boldsymbol{x}_m^i) + (1 - y_m^i) \log(1 - h_{\boldsymbol{W}}(\boldsymbol{x}_m^i) \right]. \tag{2.26}$$

The neural network uses the back propagation algorithm to advance the learning step to minimize the cost function, and the value of $\boldsymbol{W}$ changes to minimize the cost function.

3. Research Data

In this study, daily closing prices of 1986 days (KOSPI200 indexes from January 2, 2009 to December 29, 2016) were used as the model estimation and data for forecasting stock index. During the entire 8 year period, 75% of the 6 years (from January 2, 2009 to December 30, 2014, 1492 days) were used as training data for

learning, and 25% (From January 2, 2015 to December 29, 2016, 494 days) were used as test data.

11 technical indicators were used as input data for stock index prediction. The calculation method for each technical indicator is summarized below.

Momentum:

$$\text{Momentum} = C_t - C_{t-n} \tag{3.1}$$

Momentum represents the price change over the past n days. It is calculated as the price difference between the close price of day t (C_t) and the close price before n days (C_{t-n}). If it is greater than 0, it indicates that the market is in an upward trend. If it is less than 0, it means that the market is in a downward trend.

Rate-Of-Change (ROC):

$$\text{ROC} = \frac{C_t - C_{t-n}}{C_{t-n}} \times 100 \tag{3.2}$$

ROC is a measure of relative momentum. The ROC is calculated by dividing the Momentum value ($C_t - C_{t-n}$) by the close price before n days (C_{t-n}). If it is greater than 0, it indicates that the market is in an upward trend. If it is less than 0, it means that the market is in a downward trend.

Disparity index (Disparity 5, Disparity 10):

$$\text{Disparity n} = \frac{C_t}{MA_n} \times 100, \text{ where } MA_n = \sum_{i=0}^{n-1} C_{t-i}/n \tag{3.3}$$

Disparity is a value representing the ratio of current close price (C_t) to moving average during n days (MA_n). If it is greater than 1, it means an upward trend. If it is less than 1, it means a downward trend.

Williams %R:

$$\text{Williams \%R} = \frac{HH_n - C_t}{HH_n - LL_n} \times 100 \tag{3.4}$$

Williams % R is a measure of how much it has fallen from recent highest high price. The numerator is the highest high price for n days (HH_n) minus the current close price (C_t), the denominator is calculated by subtracting the lowest low price for n days (LL_n) from the highest high price for n days (HH_n). For Williams %R, 0 to -20 means overbought, and -80 to -100 means oversold.

Commodity Channel Index (CCI):

$$\text{CCI} = \frac{M_t - SM_t}{0.015 D_t} \tag{3.5}$$

where $M_t = (H_t + L_t + C_t)/3$, $SM_t = \dfrac{\sum_{i=1}^{n} M_{t-i+1}}{n}$, $D_t = \dfrac{\sum_{i=1}^{n} |M_{t-i+1} - SM_t|}{n}$

The CCI is a measure of whether average prices (M_t) are trending up or down compared to the smoothed average price (SM_t). Unlike Momentum, which uses only close price (C_t), it uses average prices of high (H_t), low (L_t) and close price (C_t) at time t. Mean deviation (D_t) and constant 0.015 are used for scaling. If CCI is greater than 0, the market is in an upward trend and if it is less than 0, it is in a downward trend

Percentage Price Oscillator (PPO):

$$\text{PPO} = \frac{EMA_n - EMA_m}{EMA_m} \times 100 \tag{3.6}$$

$$\text{where } EMA_n^0 = \sum_{i=0}^{n-1} C_{t-i}/n \text{ , } EMA_n^t = EMA_n^{t-1} + (C_t - EMA_n^{t-1}) \times \lambda$$

PPO is an oscillator that measures momentum by calculating the difference between two exponential moving averages(EMA_n, EMA_m, $n < m$). EMA_n is calculated as weighted sum of previous exponential moving average (EMA_n^{t-1}) and price difference between close price (C_t) and previous EMA (EMA_n^{t-1}) using multiplier ($\lambda = 2/(n+1)$). Generally, if it is greater than 0, it shows an upward trend. If it is less than 0, it shows a decreasing trend.

Relative Strength Index (RSI):

$$\text{RSI} = \frac{RS}{1+RS}, \qquad \text{where } RS = \frac{\sum_{i=0}^{n-1} Up_{t-i}/n}{\sum_{i=0}^{n-1} Dw_{t-i}/n} \tag{3.7}$$

RSI is a momentum oscillator that measures the rate of price fluctuations. RSI is calculated using the ratio of average gain ($\sum_{i=0}^{n-1} Up_{t-i}/n$) to average loss ($\sum_{i=0}^{n-1} Dw_{t-i}/n$). The average gain is the average of the upside change (Up_{t-i}) over the last n days. The average loss is the average of the downside change (Dw_{t-i}) over the last n days. When RSI is above 70, it means overbought and when it is below 30, it means oversold.

Stochastic Oscillator (Stochastic % K, Stochastic % D, Stochastic Slow % D):

$$\text{Stochastic \%K} = StoK_t = \frac{C_t - LL_n}{HH_n - LL_n} \times 100 \tag{3.8}$$

$$\text{Stochastic \%D} = StoD_t = \frac{\sum_{i=0}^{n-1} StoK_{t-i}}{n} \tag{3.9}$$

$$\text{Stochastic slow \%D} = StoSlowD_t = \frac{\sum_{i=0}^{n-1} StoD_{t-i}}{n} \tag{3.10}$$

The Stochastic Oscillator is a measure of how much it has risen from its recent low price. The nominator is the current price (C_t) minus the lowest low price for n days (LL_n), The denominator is calculated by subtracting the lowest low price for n days (LL_n) from the highest high price for n days (HH_n). Stochastic %K stands for an upward trend if it is above 50, and a downward trend if it is below 50. The Stochastic %D value is calculated as the n day moving average of the Stochastic %K value. The Stochastic slow %D value is calculated as the n day moving average of the Stochastic %D value. Stochastic %D is slower indicator than Stochastic %K. Stochastic slow %D is slower indicator than Stochastic %D

Each technical indicator was converted to a normalized value between [0, 1] for use as input data. The output of the forecasting model is classified into 1(rising) and 0(falling) according to the fluctuation of the stock price index. The model was trained so that it would be 1 for a rise and 0 for a fall. As a result, (11×1,986) values composed of values between [0, 1] were used as input data corresponding to 11 technical indicators for 1,986 days as a whole, and as a target value for the output value, (1×1,986) time series data were used

In this study, binary classification is used, so the result is positive(1) or negative(0). True Positive(TP) means that the target value is positive when the predicted value is positive, and False Negative(FN) means that the target value is negative when the predicted value is positive. True Negative(TN) means true when the target value is negative when the predicted value is negative, and false positive(FP) means that the target value is positive when the predicted value is negative. TP, TN, FP, and FN was used to calculate statistical values for measuring performance. Precision is an indicator of how positive the data actually is. Recall is an indicator of the ratio of positive and correct predictions for actual positive data. Accuracy is a value that actually accounts for the exact predicted value. F1-score is a comprehensive index of precision and recall. The calculation method is as follows.

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}, \; F1-\text{Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4. Result and Comparative Analysis for Models

To compare the performance of the four machine learning models, various values

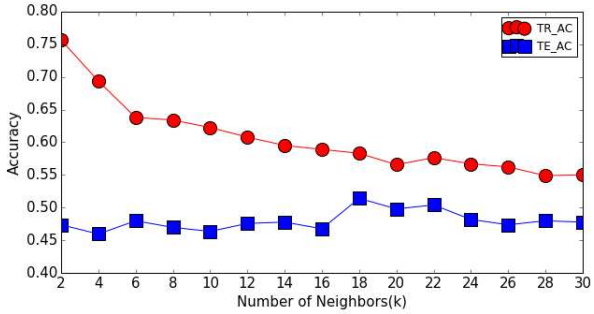
were used as hyperparameter for each machine learning model in Table 2. Hyperparameter is the setting used to control the behavior of the learning algorithm.

Table 1. Hyperparameters of each model

	Hyperparameter
k-NN	k : $2, 4, 6, ..., 28, 30$
Random Forest	Depth : $2, 3, 4, 5$ / Feature : $2, 3, 4, ..., 10, 11$ Trees : $50, 100, 150, 200, 250$
SVM	γ : $0.5, 1.0, 1.5, ...3.5, 4.0$ / C : $5, 10, 20, 50, 100$
ANN	Node : $10, 20, 30, 40, 50$ Epoch : $30000, 60000, ...270000, 300000$

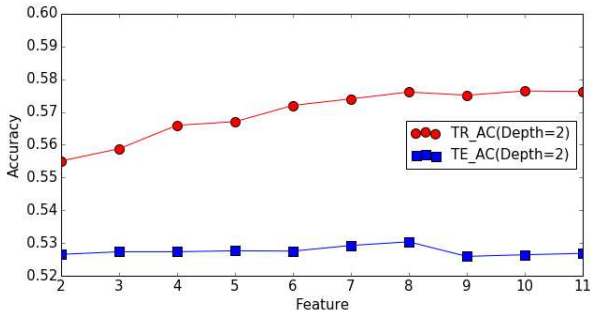
Since there are different hyperparameters according to each machine learning model, the range of values is specified according to the characteristics. In the case of k-NN, the maximum number of k values for the number of neighbors is 30. In k-NN, it is known that the smaller the k value, the more tendency to overfit. In random forest, we changed it according to depth, feature, and tree value. The tree represents the number of each decision tree, and the depth and feature determine the structure of the forest. Tree from 50 to 250, depth from 2 to 5, and the number of features was changed up to 11 which is the same number of input indicators. We used the radial basis kernel for the SVM kernel and analyzed the results by changing the hyperparameters γ and C. Generally, a smaller C value tends to under-fit, while a larger C value tends to over-fit. γ values were compared sequentially from 0.5 to 4.0 and C values were analyzed from 5 to 100. In the ANN, the number of hidden layers is fixed to 3, and the number of hidden layer node for each layer is changed from 10 to 50. The learning rate was fixed to 0.00005, which was small enough, and the epoch corresponding to the learning step was changed up to 300,000.

Figure 1. TR_AC and TE_AC of k-NN with various k



In the results of estimating the direction of the KOSPI 200 index using k-NN, the training accuracy (TR_AC), test accuracy (TE_AC), precision (PR), recall (RC) and f1-score (F1-S) were used to analyze the predictive results. Accuracy is calculated for training data and test data, while precision, recall, and f1-score are calculated only for test data. First, the tendency to appear as a distinctive feature is that the smaller the value of k, the larger the value of TR_AC. The reason why TR_AC is high for a small k value is that the model is finely optimized in a narrower range. On the other hand, TE_AC does not become large unlike TR_AC when the value of k decreases. Because it is overfitted in a region where k is small. Therefore, the hyperparameter should be set to reduce the generalization error of the machine learning model. When the value of k decreases below 18, the value of TE_AC decreases (Figure 1). The TE_AC value was the highest at 0.5142 when the k value was 18.

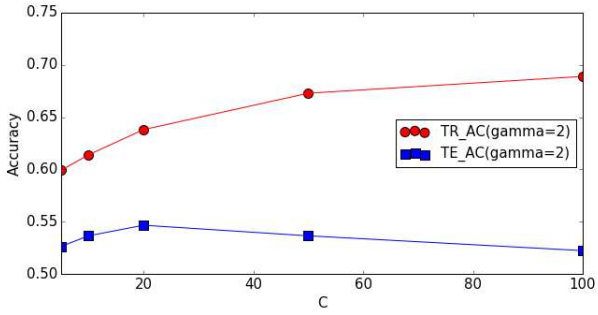
Figure 2. TR_AC and TE_AC of random forest with various feature



In the results for random forest. The effect of tree number was the smallest among the three hyperparameter values. Therefore the number of trees was fixed to 150. The tendency that can be easily observed is that the larger the depth, the larger the value of TR_AC. In this case, as in the case of small k value in k-NN, TR_AC increases. Generally, the larger the depth in the random forest, the more detailed the forest structure can be. In other words, it is possible to classify even the detailed part, but it can be overfitted to training data. In the result of this study, TR_AC increases sequentially as depth increases, but TE_AC does not increase and generalization error increases. TE_AC is largest when depth is 2, and generalization error is the smallest when depth is 2 because the difference between TR_AC and TE_AC is the smallest. When the depth is 2, the TE_AC increases gradually as the feature size increases, and the TE_AC value decreases again when the feature is

larger than 8 (Figure 2). Although we have 11 types of technical indicators used as input data, we can see that the best result is obtained with 0.5304 for TE_AC when feature of random forest is 8. Another characteristic result of the random forest is that the RC value is close to 0.9 when the depth is 2. This is because, when analyzed together with the result that the PR value is near 0.5, the proportion of positive (1) predicted values of in the random forest is larger than that of negative (0) predicted values.

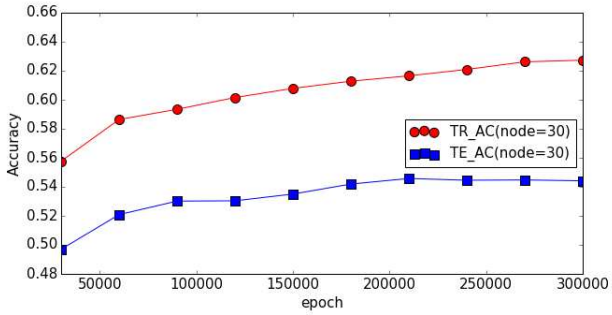
Figure 3. TR_AC and TE_AC of SVM with various C



In the result for the SVM, TR_AC showed strong tendency to both γ and C values. The larger the value of γ and C, the larger the TR_AC, and when γ is 4, the results are close to 0.7. As in the case of k-NN and random forests, TR_AC increases but TE_AC does not increase significantly. This indicates that the greater the value of γ and C, the more overfitting it shows. TE_AC showed the best performance when γ value was 2, and the highest value was 0.5466 when C value was 20 (Figure 3). In the SVM, unlike the random forest, the RC value did not increase excessively. As the PR and RC values are maintained at values between 0.5 and 0.6, it can be seen that the predicted value of SVM is relatively evenly distributed without being biased to either positive (1) or negative (0).

In ANN, the number of hidden layers is fixed to 3, and the learning rate is fixed to a sufficiently small value of 0.00005. Then, the number of nodes for each layer and the epoch corresponding to the learning step are changed. As the epoch increases, both TR_AC and TE_AC tend to increase until some point. However, in the case of TE_AC, there is a level where the TE_AC value does not increase even if the epoch increases. Especially, when the node is 40, the TE_AC decreases even though the epoch increases in the epoch more than 200,000. This result can be regarded as the overfitting result where the generalization error increases because

Figure 4. TR_AC and TE_AC of ANN with various epoch



TE_AC decreases when TR_AC increases. Unlike the results of the three machine learning models analyzed earlier, the TR_AC value of the ANN does not increase at around 0.6, indicating that ANN has the smallest generalization error. As a result of analyzing the characteristics according to the number of nodes, TE_AC increases when the number of nodes increases from 20 to 30, and the result shows that TE_AC decreases as the node increases to 40 or more. TE_AC showed the best performance when node value was 30, and the highest value was 0.5457 when epoch value was 210,000 (Figure 4). It shows that the number of nodes of ANN does not need to be excessively larger than the number of input data.

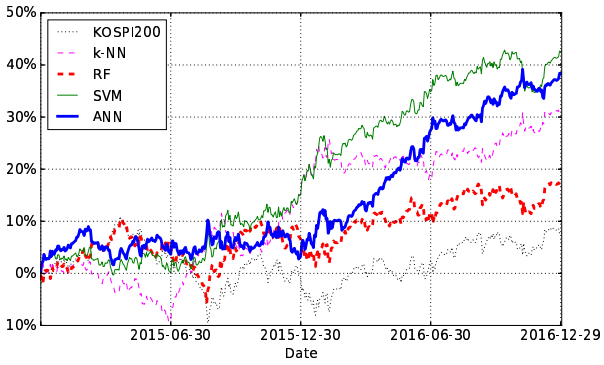
Table 2. Comparative analysis for various models

	TR_AC	TE_AC	PR	RC	F1-S
k-NN	0.5831	0.5142	0.5446	0.4692	0.5041
RF	0.5761	0.5304	0.5312	0.9173	0.6728
SVM	0.6381	0.5466	0.5714	0.5538	0.5625
ANN	0.6165	0.5457	0.5193	0.5385	0.5287

TR_AC: training accuracy, TE_AC: test accuracy,
PR: precision, RC: recall, F1-S: f1-score

Overall, analyzing the results of various machine learning models shows the advantages and disadvantages of each machine learning model. The best performance results for TE_AC and the returns generated from each model are compared in Table 4 and presented in Figure 4, respectively. First, the results of TE_AC in SVM and ANN are superior to k-NN and random forest. For the SVM and ANN, the TE_AC values were similar (0.5466 and 0.5457). Comparing the RC values according to the model, the RC value in the random forest was around 0.9, which is a large value indicating the result was biased toward a positive value. However, in SVM and ANN,

Figure 5. Realized returns generated from each model



the RC value was less than 0.6 and this suggest the bias problems in these models are not conspicuous relative to the random forest. In summary, SVM and ANN are superior to k-NN or random forests, which is partly in agreement with previous research results mentioned in previous studies. Comparing the SVM and ANN, the deviation of the TE_AC value of the SVM was larger than that of the ANN by the hpyerparameter. In ANN, if the epoch is large, the deviation of the result according to the node is relatively small. From this point of view, ANN can be considered to be more stable than SVM. In terms of speed, SVM has the advantage that it can produce faster results than ANN.

5. Conclusion

In order to utilize various machine learning models in financial markets, we applied KOSPI200 stock index return forecasting and compared the results. We compared the results of the hyperparameters for each machine learning model and analyzed the tendency of the results. In order to prevent overfitting, which is an important issue in machine learning, we selected a suitable hyperparameter to reduce the generalization error. Comparing the predictive power according to the machine learning model, SVM and ANN are superior to k-NN and random forest in terms of TE_AC and generalization error. In addition, SVM and ANN did not show bias toward positive results that can be deduced from the high RC values observed in random forests. Based on the comparison results of the machine learning models analyzed in this study, further studies should explore better models for individual financial markets.

Acknowledgment

We appreciate the anonymous referees for their comments and suggestions. The corresponding author acknowledges financial support from the Hankuk University of Foreign Studies 2017 research fund.

References

1. L. Breiman: Random Forests. Machine Learning **45** (2001), 5-32
2. B.E. Boser, I.M. Guyon & V.N. Vapnik: A training algorithm for optimal margin classifiers. Proceeding COLT '92 Proceedings of the fifth annual workshop on Computational learning theory (1992), 144-152
3. C.L. Dunis, J. Laws & G. Sermpinis: Higher order and recurrent neural architectures for trading the EUR/USD exchange rate. Quantitative Finance **11** (2011), 615-629
4. T.K. Ho: The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998), 832-844
5. W. Huang, Y. Nakamoria & S. Wangb: Forecasting stock market movement direction with support vector machine. Computers & Operations Research **32** (2005), 2513-2522
6. M. Kim: K-Nearest Neighbors (K-NN) Algorithm for KOSPI200 Futures Index. Journal of Korean Management Association **28** (2015), 2613-2633
7. K. Kim: Financial time series forecasting using support vector machines. Neurocomputing **55** (2003), 307-319
8. H.Y. Lee: A Combination Model of Multiple Artificial Intelligence Techniques Based on Genetic Algorithms for the Prediction of Korean Stock Price Index(KOSPI). The Review of Financial Studies **1** (1988), 41-66
9. W.S. McCulloch & W. Pitts: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5** (1943), 115-133
10. J.Y. Park, J.P. Ryu & H.J. Shin: Predicting KOSPI Stock Index using Machine Learning Algorithms with Technical Indicators. Omega **29** (2001), 309-317
11. F. Rosenblatt: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review **65** (1958), 386-408
12. D.E. Rumelhart, G.E. Hinton & R.J. Williams: Learning Internal Representations by Error Propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition **1** (1985), 318-362
13. L.A. Teixeira & A.L. Oliveira: A method for automatic stock trading combining technical analysis and nearest neighbor classification. Journal of Korean Future Research **22** (2014), 45-70

14. Z. Yudong & W. Lenan: Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network. Indian Institute of Capital Markets 9th Capital Markets Conference Paper **9** (2005), 1-16

[a]DEPARTMENT OF ECONOMICS, HANKUK UNIVERSITY OF FOREIGN STUDIES, SEOUL 02450, REPUBLIC OF KOREA
Email address: bonsang9@gmail.com

[b]DEPARTMENT OF ECONOMICS, HANKUK UNIVERSITY OF FOREIGN STUDIES, SEOUL 02450, REPUBLIC OF KOREA
Email address: jhsong@hufs.ac.kr