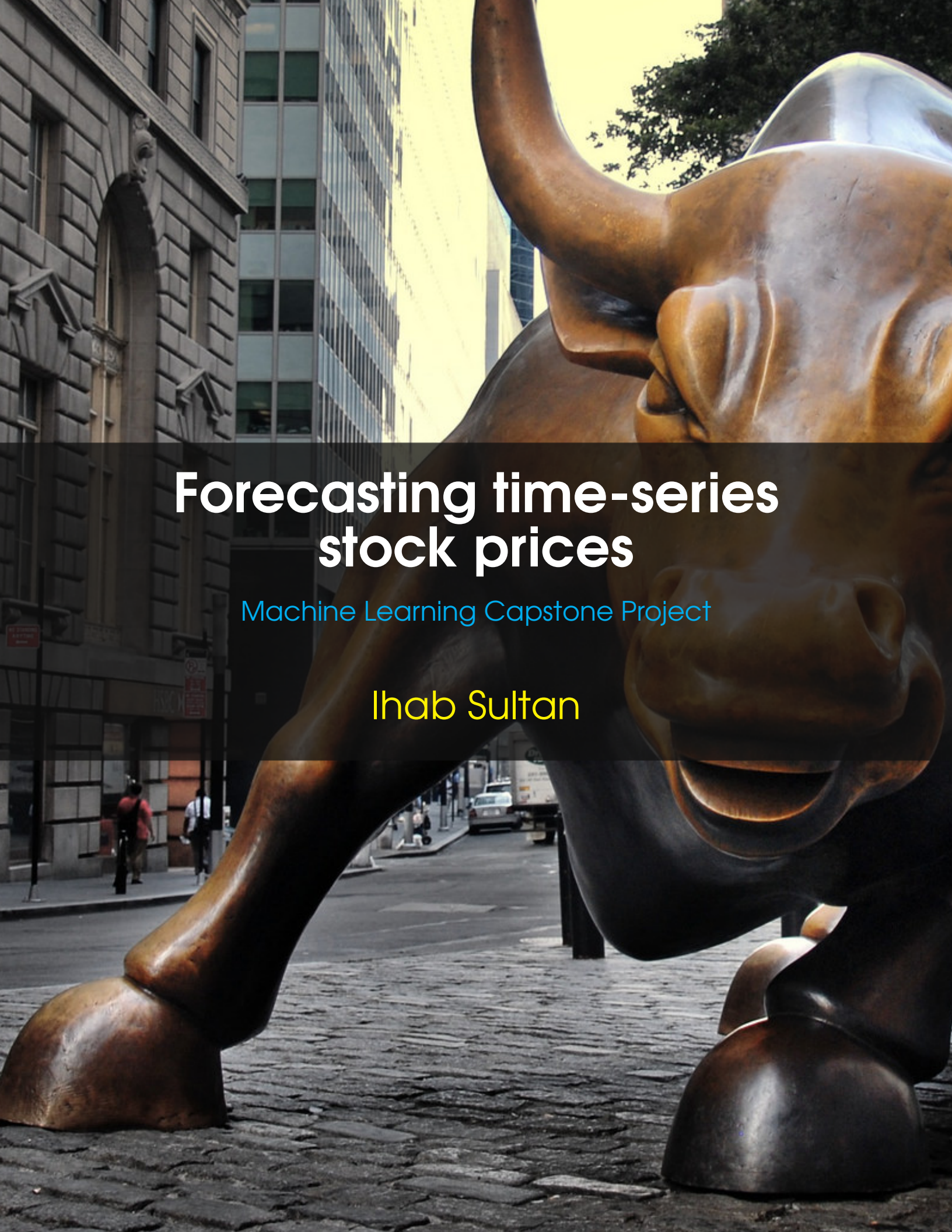

Forecasting time-series stock prices

Machine Learning Capstone Project

Ihab Sultan

Contents

Charging Bull - New York City by Sam valadi used under CC BY 2.0
Bankers by Tim Kwee used under CC BY-NC 2.0

1. Definition

1.1 Project Overview

In this project we will apply few machine learning algorithms aiming to predict, as close as possible, the future price of a given stock in the market based on information that we have till now.

If the claim that 'history repeats itself' was true in the context of stock markets, it would be enough to search through historic trading prices of a given stock to predict what will happen tomorrow, in essence, this is what a basic nearest neighbor implementation would do in this regards.

Truth be said, history does not repeat itself, and especially in stock markets. Otherwise, one basic machine learning algorithm would be the only tool we need to beat the market day in, day out.

Machine learning, in this context, would help us build a strong model that can sufficiently predict the future to some degree, and would enable us to capitalize on this knowledge to generate returns in the stock market.

Besides comparing predictive models, selecting inputs(factors) that drive these models is an important area of research in itself, and one needs to utilize domain knowledge for that purpose. Previous research already investigated the use of factors beyond historic prices, such as international markets, company news sentiment, and fundamental factors of the company. Analysis of these factors can be a natural extension to the work presented here and might aid to improved prediction accuracy.

The input data we utilize in this project is all available online, and can be obtained from different online services such as Yahoo! finance[1], Google finance[2], and Quandl[3].

[1] finance.yahoo.com
[2] www.google.com/finance
[3] www.quandl.com

1.2 Problem Statement

Let's state the problem by first introducing an important term that we'll use frequently throughout this project:

Time series: Time series is a sequence of observations, y_t of a variable y at equal time intervals.

We use time series as a mathematical formulation in this project to help formalize the stock forecasting problem, and based on this framework, we devise machine learning tools to predict the stock price at predefined future time steps.

In particular, our goal is to find the (possibly non-linear) function f, which maps T previous observations of a (vector) variable y, to a prediction of y at future time steps, which lies H time-steps ahead of the latest observation.

Work from Takens[7] concluded that for many dynamic systems, one can accurately construct current state based on previous observations falling in a finite window of the time series, known as time delay embedding. In light of the current problem, this means that a finite number of time-series observations is all that is needed to construct the current state driving future observations of observable variable y.

It is worth mentioning here that the original formulation by Takens[7] assumed that samples within the window are taken with τ steps apart, where τ is a hyper-parameter known as embedding delay. In our system we consider τ to be fixed at 1 and do not optimize over the parameter.

Similar to Non-Linear Auto Regressive (NAR) formulation of [3], we can formalize the above as follows:

$$y_{t+h} = f(y_t, y_{t-1}, \cdots, y_{t-T+1}) + w(t)$$

where T is the embedding dimension of the time series (number of previous observations used in the prediction), and $w(t)$ is a term including noise and modeling error. The forecasting problem then becomes one of estimating:

$$\text{forecast } \equiv E[y_{t+h} | y_t, y_{t-1}, \cdots, y_{t-T+1}].$$

Where h is called the forecast horizon. If $h = 1$, we call the problem a single-step forecasting, and we define multi-step forecasting to be the case where $h > 1$.

Multiple techniques have been devised for multi-step forecasting, which can be broadly categorized in:

- Recursive strategy: modeling each future step as a single-step recursion over moving window of old observations. The downside of this approach is the accumulation of single-step prediction errors specially at large horizons.
- Direct strategy: modeling each horizon value separately, with the downside of losing the functional dependence modeling between predicted future values as in the recursive strategy.

- DirRec strategy: a combination of the above strategies, where a new model is traied for each horizon, albeit with increasing window size at each step, such that the beginning of the window remains fixed, while the end moves with the horizon.

In this project we use the direct strategy, and predict for four horizon levels: 1 day, 1 week, 2 weeks, and 1 month.

The time unit in stock pricing is days, and on average there are 252 trading days per year in US stock exchange [2]. This means there are only 252 (multi-dimensional) samples per year. With such a low number of samples, over-fitting becomes even more dangerous, and one needs to test properly for over-fitting in small sample applications.

1.3 Metrics

Forecasting in the framework above is a regression problem, yet it can also be modified to become a classification problem if we quantize the output into countable classes (such as up or down trends) as we will see below.

For a regression problem, one can report multiple metrics as we do in this project, but we pick only one metric for grid search algorithm for hyper-parameter tuning.

In this project we chose mean squared error as our tuning metric. With the nomenclature outlined in section 1.2, MSE is defined as:

$$\text{MSE} = E[(y_{t+h} - \hat{E}[y_{t+h}|y_t, y_{t-1}, \cdots, y_{t-T+1}])^2]$$

where $\hat{E}$ is expectation estimate that is provided by the different modeling techniques studied in this paper. Moving forward, we will denote $\hat{E}[y_{t+h}|y_t, y_{t-1}, \cdots, y_{t-T+1}]$ as F_{t+h}, where F stands for forecast.

Besides MSE, the following metrics are reported to compare models performance:

- RMSE (Root Mean Squared Error):

$$\text{RMSE} = \sqrt{E[(y_{t+h} - F_{t+h})^2]}$$

- MAE (Mean Absolute Error):

$$\text{MAE} = E[|y_{t+h} - F_{t+h}|]$$

- MAPE (Mean Absolute Percentage Error):

$$\text{MAPE} = E\left[\left|\frac{y_{t+h} - F_{t+h}}{y_{t+h}}\right|\right]$$

As mentioned above, it is also common in forecasting applications to treat the problem as a classification problem, where the two classes are up trend and down trend. One useful metric commonly used in forecasting classification is the hit rate[9], which is defined as the ratio of correct direction predictions to total predictions.

2. Analysis

2.1 Data Exploration

The system built in this project enables the user to run many common explorations types for stock indexes, a list of explorations and brief description of each can be found in the appendix.

In this section we will choose AAPL and GOOG as main symbols and throughly outline stock behavior in training(validation), and testing periods.

2.1.1 Training and Validation Period

Training and validation period covers the time interval between Jan 1^{st} 2010 and Jan 1^{st} 2016. The behavior of AAPL stock during this period is outlined in the statistics summary table below:

	SPY	AAPL	GOOG
count	1510.0	1510.0	1510.0
mean	147.27	71.36	413.18
std	37.38	28.49	139.37
min	90.34	25.25	217.82
25%	115.16	46.94	292.53
50%	135.78	69.69	371.99
75%	185.72	91.29	538.14
max	208.08	129.88	776.6
Sharpe ratio	0.83	0.95	0.7

Figure 2.1: *Statistics for AAPL and GOOG stocks between Jan 1^{st} 2010 and Jan 1^{st} 2016*

Notice that average Sharpe ratio of both AAPL and GOOG is below 1, which is considered low, and means that the average return of the stock is less than the standard deviation of return. It is expected to have better forecasting performance when average Sharpe ratio in

training and testing periods is higher, one can either search for stocks of high Sharpe ratio or utilize online tools for this purpose.

The other point to consider here is that there is a great variation in mean between different stocks, which makes absolute comparison between stocks difficult. For this purpose, we will highlight MAPE error metric measurements in the results section to have a better feel of the error percentage without reference to actual stock price.

Following figure zooms on AAPL stock in 2014-2015 period:

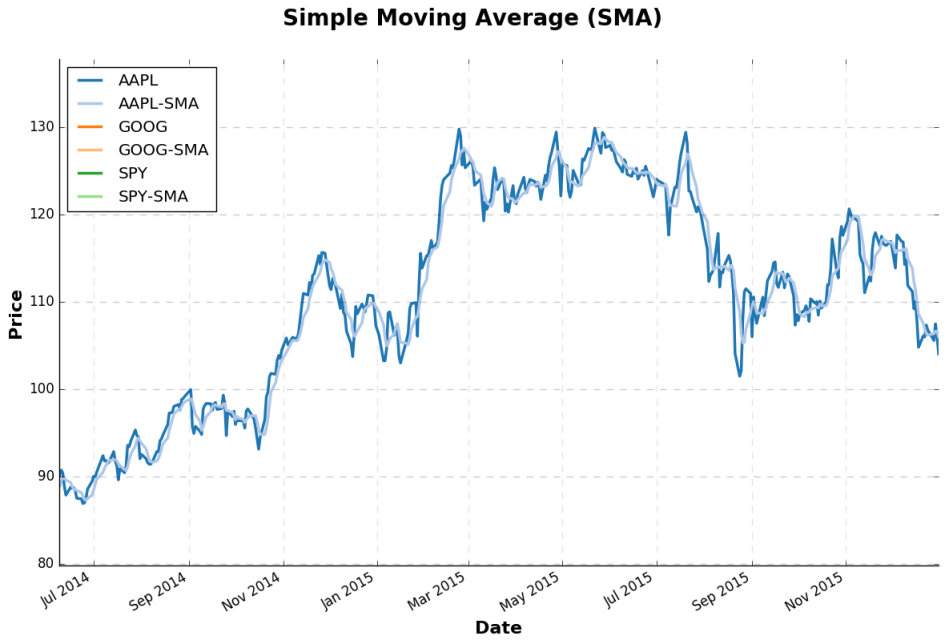

Figure 2.2: *AAPL stock daily closing price and moving average between July 2014 and December 2015*

Notice that statistics of AAPL stock vary considerably with time, and hence the stock price is categorized as a non-stationary sequence. This property might affect performance of learning algorithms, such as k-NN which will not be able to generalize and predict a price that falls outside the closing price range of training samples.

For this reason, the system discussed implements an optional pre-processing step to use daily and cumulative returns instead of absolute closing prices. Making this step optional enables the user to compare performance of stationary and non-stationary sequences with different prediction models.

2.1.2 Testing Period

The testing period in the results reported was chosen between Jan 1^{st} 2016 and July 1^{st} 2016. Figure below gives a visual clue of difficulty of the problem at hand.

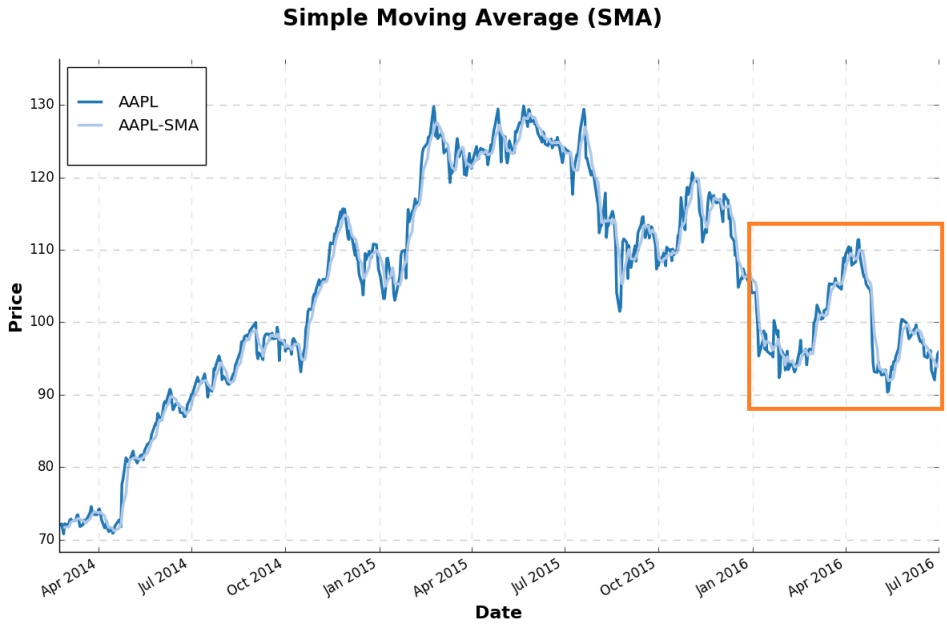

Figure 2.3: *AAPL stock daily closing price between Jan 2014 and July 2016, the testing period is the boxed region, remaining period is part of the validation interval*

By visual inspection, the testing period looks clearly different than any region in training, hence it would be rather difficult to generalize from stock price by itself. For this reason, we add other indicators to aid the prediction algorithm, and we make utilizing the new indicators optional such that one can compare performance difference when using closing prices only against performance when all available indicators are used.

2.2 Algorithms and Techniques

In this project we aim at covering both literature proposed solutions, in addition to investigating novel modifications.

We first investigate a straightforward implementation of basic k-NN without modifications. This serves as the main benchmark result as outlined in the benchmark section below.

Multiple regression methods were compared with hyper-parameter optimization. The following sections highlight these methods with a brief description for each, and lists hyper-parameters each model was optimized over:

2.2.1 k-Nearest Neighbors

k-NN was used in forecasting applications with different proposed modifications, below we list few proposals and justify our choice of whether to include the modification or not based on our specific problem at hand (stock forecasting):

1. Modifying distance metric from Euclidean distance to exponential weighted distance: when comparing two series of points, it is logical to give larger weight to most recent points, therefore this proposal was included in our system as a hyper-parameter.[6]
2. Using nearest trajectories instead of neighbors by forcing the k nearest neighbors to be disperse in time: This technique is specially helpful for oversampled waveforms in order to avoid correlations of the obtained k samples, this might be less of a problem for samples of daily close prices which might not be suffering from oversampling issues, hence we did not include this modification.[5]
3. Efficiency improvement using kdd-tree and balltree: Both methods help improve efficiency specially for lower dimensional embeddings (below 50 dimensions). Given that the number of samples did not warrant further optimization, we chose kd-tree as the method for finding the nearest neighbors.
4. Number of neighbors, k: this parameter can be either optimized globally, or chosen locally using some error minimization criterion (such as Leave-One-Out (LOO)[8]). In this project we empirically search for the optimal k value.

In addition to the above proposals, multiple data presentations were empirically compared, including: absolute stock price, daily percent return, and cumulative return with fixed reference.

Our k-NN implementation ended up with the following hyper-parameters:

- Number of neighbors ($k \in \{1,2,4,8,16\}$).
- Exponential weight ($exp\_wegith \in \{1,0.9,0.8,0.7,0.6,0.5\}$).
- Data presentation ($data\_presentation \in \{'A','C','D'\}$ for Absolute, Cumulative, and Daily).

2.2.2 Least Squares with Regularization

Linear Regression

Linear regression aims to find weights, $\{w_i\}_{i=0}^{T}$, that minimizes the sum of squared errors, defined as:

$$\text{SSE} = \sum_{t \in \text{tset}} \left(y_t - w_0 - \sum_{i=1}^{T} w_i y_{t-i} \right)^2$$

where tset is the training set used to calculate and optimize SSE.

Shrinkage Methods

Shrinkage methods add regularization term to penalize weights size, common methods are:

- *Ridge*: minimizes SSE + L2 norm:

$$\text{SSE} + \alpha \|w\|_2^2$$

- *Lasso*: minimizes SSE + L1 norm:

$$\frac{1}{2|tset|}\text{SSE} + \alpha \|w\|_1$$

- *Elastic net*: minimizes SSE + combination of L1 and L2 norms:

$$\frac{1}{2|tset|}\text{SSE} + \alpha\rho \|w\|_1 + \frac{\alpha(1-\rho)}{2}\|w\|_2^2$$

  An α value of 0 would turn an elastic net objective to ordinary least squares (linear regression). A ρ value of 0 turns elastic net to ridge based optimization, while a ρ of 1 turns it into a Lasso based optimization.

Polynomial Regression

Polynomial regression is accomplished by augmenting input features with n^{th} order multiplications of the features, followed by a linear regression over all the features.

All methods discussed within this section can be combined using the following hyper-parameters:

- Polynomial degree (degree $\in$ 1,2)
- α (alpha $\in$ 1, 0.5, 0)
- ρ (rho $\in$ 1, 0.5, 0)

Additional Indicators

Other than daily adjusted close price, additional indicators were tested out, and performance was compared with and without additional indicators. The following is a list of indicators used in the system:

- **Adjusted closing price (ACP)**: This is a common indicator used in all algorithms listed above.
- **Open-High-Low-Close (OHLC)**: Instead of treating these as four factors, we normalize open, low, and high prices with respect to closing price.

- **Simple Moving Average of ACP**: SMA was calculated with a window size of 2 weeks (10 trading days).
- **Moving Standard Deviation of ACP**: similarly MSD was calculated with a window size of 2 weeks (10 trading days).
- **Exponential Weighted Moving Average of ACP**: EWMA was calculated with a window span of 2 weeks (10 trading days) giving what is commonly referred to as 10-day EWMA. Based on span value, decay parameter (α) can be calculated as[11]: $\alpha = \frac{2}{s+1}$, the output of EWMA is then:

$$z_t = \frac{y_t + (1-\alpha)y_{t-1} + (1-\alpha)^2 y_{t-2} + \cdots + (1-\alpha)^t y_0}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots + (1-\alpha)^t}$$

- **Volume**: Daily trading volume was used as a separate feature in our system.

In most of the cases, last two months worth of samples were used as indicators. All indicators are concatenated as explained in the implementation section of this paper.

2.2.3 Support Vector Regression

Support vector regression is an extension of support vector machines for regression, and aims to minimize the following objective[12]:

$$\begin{aligned} \text{minimize} \quad & C\sum_{i=1}^{T}(\zeta_i + \zeta_i^*) + \frac{1}{2}\,\|w\|_2^2. \\ \text{subject to} \quad & \begin{cases} y_i - \langle w, x_i\rangle - b & \leq \varepsilon + \zeta_i \\ \langle w, x_i\rangle + b - y_i & \leq \varepsilon + \zeta_i^* \\ \zeta_i, \zeta_i^* & \geq 0 \end{cases} \end{aligned}$$

similar to ridge objective, SVR tends to regularize the square of second norm. ε is the default error tolerance, and the C factor controls the trade-off between second norm minimization and error tolerance beyond ε.

The following hyper-parameters are tested during validation process:

- SV Kernel (kernel $\in$ linear, poly, rbf)
- Penalty parameter (C $\in$ 1e-5, 1e-2, 0.1, 1, 10)
- Kernel coefficient (gamma $\in$ 'auto')
- Stopping criterion (tolerance $\in$ 1e-5, 1e-4, 1e-3, 1e-2, 0.1)

2.2.4 Random Forests

Random forests is the first of ensemble regressors we study here. It is an ensemble of decision trees where each decision tree is trained on a randomly selected (with replacement) subset of the training data, and splits are picked from a randomly selected subset of the features[13].

The following hyper-parameters are tested during validation process:

- Total number of decision trees in ensemble (n_estimators $\in$ 5, 10, 20)

- Number of features per split (max_features $\in$ sqrt, log2)
- Minimum samples required to split (min_samples_split $\in$ 2, 4, 8)

Notice that number of trees and number of features help us to move along the bias-variance curve. In order to decrease variance, one can increase the number of trees or decrease the number of features per split.

2.2.5 AdaBoost

AdaBoost [14] is a model ensemble of week learners, where each learner considers the weight of input samples in its decision. Sample weights, in turn, get updated based on error rates after each learner is added to the ensemble.

During learning process, AdaBoost picks only features that improve prediction of training samples with large weights, which essentially helps reduce the dimensionality of features space by retaining useful features only.

The following AdaBoost hyper-parameters are tuned during validation process:

- Total number of decision trees in ensemble (n_estimators $\in$ 5, 25)
- Learning rate (learning_rate $\in$ 0.01, 0.1, 1)
- Loss function for updating weights (loss $\in$ 'linear', 'exponential')

2.2.6 Gradient Boosting

Gradient boosting [15] trains week learners on the negative gradient of loss. Notice that negative loss gradient is equivalent to residual error for the case of square loss.[1]. Therefore, gradient boosting, at each iteration, tries to boost the current model in a way that minimizes the loss.

At iteration m, gradient boosting step can be summarized as [15]:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$$

where h is a week learner (commonly a decision tree), with parameters $\mathbf{a}$ that are chosen to model the negative gradient of last iteration's loss, and ρ is a model weight chosen to minimize the total loss post model update.

Stochastic gradient boosting [16] is an extension that uses a bagging method to select a subset of training samples with replacement at each learning iteration. Similar to random forests, a subset of features can also be chosen at each iteration.

The following hyper-parameters are tested during validation process:

- Total number of boosting stages (n_estimators $\in$ 5, 25, 100)
- Learning rate (learning_rate $\in$ 0.01, 0.1)
- Loss function for optimization (loss $\in$ 'lad', 'huber')

2.2.7 Stacked Generalization

Stacked generalization [17, 18] is an ensemble method that combines a number of learners (called level 0 generalizers) using an additional learner (called level 1

[1] For detailed explanation, can refer to: http://www.chengli.io/tutorials/gradient_boosting.pdf

generalizer) which takes level 0 outputs as its input and minimizes loss with respect to correct output.

In this paper we propose a modification to stacked generalization in order to apply it to time series data. The trickiest part of the process is proper identification of how level0 training indexes are related to level1 training and validation samples. The chosen algorithm is illustrated in [Figure 2.4] for both training and testing timelines.

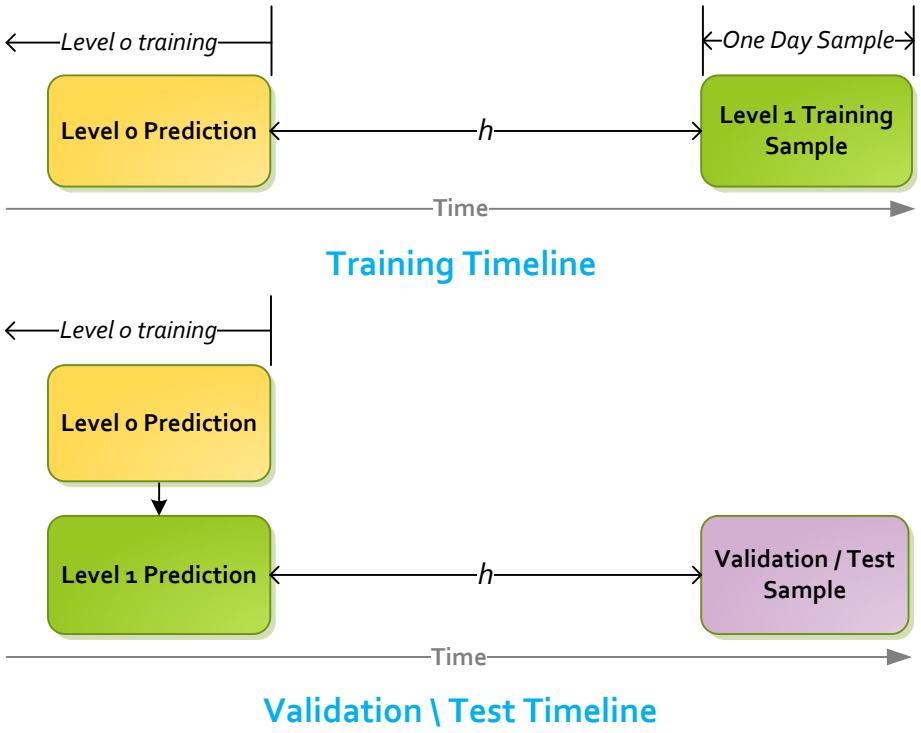

Figure 2.4: *Stacked generalization for time-series*

The training samples for level1 are constructed as follows:

1. Initialize *n_level0*
2. All level0 generalizers in the ensemble give predictions of sample *n_level0* + *h* based on training upto sample *n_level0*, where h is the forecast horizon.
3. Level1 generalizer uses the outcome of level0 at *n_level0* + *h* as input, and proper label at corresponding time index as output to generate a level 1 training sample.
4. Move to next index (increment *n_level0*), then repeat from 2

In our implementation we first build the full training dataset as detailed above, and then perform training + validation at validation points as shown in the figure.

The main problem of this process is that all level0 generalizers in the ensemble need to be trained and to give predictions of all samples in the training set, which is the most time consuming step of the process. There are some ideas to optimize this further mainly

through smart selection of models in the ensemble and smartly reducing the number of training points without affecting accuracy, both considered as plausible venues of study.

2.3 Benchmark

The benchmark used in this project is vanilla k-NN. k-NN has been utilized with many variations for the purpose of data-series forecasting as outlined in the section above, it is also straightforward to implement, and results are easy to interpret.

In later sections we will show how different methods with tuned parameters compare to k-NN in terms of error metrics measured over the test set for different symbols.

3. Methodology

3.1 Data Preprocessing

Input data is obtained through a web interface to an online financial service, initial cleaning is required to remove non-trading days and weekends.

The other data preprocessing step is optional, and consists of changing the closing price presentation. The presentation type can be used as a hyper-parameter which takes one of three options:

- Absolute(A): where stock prices remain unmodified.
- Daily(D): where stock prices are presented in daily percent returns format.
- Cumulative(C): where prices are presented in cumulative percent returns. The choice of reference is arbitrary, the last day of the training sample was chosen in this project.

3.2 Implementation

3.2.1 Training samples

After obtaining and cleaning the financial data our immediate design decision becomes how to layout the data in preparation for training and testing. The system allows a flexible number of indicators to be concatenated, and training data is presented in rows of a numpy array, where each row is a concatenation of corresponding rows in the indicators arrays, as clarified in [Figure 3.1].

The names used in the figure correspond to names used in the python source code, and are explained below:

- x_train: is the training numpy array per indicator, where each row corresponds to one sample from that specific indicator. The most important indicator is the ticker daily closing price, but any number of indicators can be used as mentioned above, in this system we utilize indicators like SMA(simple moving average) and MSTD(moving standard deciation) plus others which are listed and described in section 2.2.2.

- indicator_i: is the number of samples used per indicator. The system allows different indicators to have different number of samples. As an example, one indicator can use 42 samples to cover previous two months of trading days while another indicator might consider last 5 samples only.

- first: is the maximum of all indicator_i values, called first since it is the first sample in y_train samples.
- indicators per sample: is the sum of all indicator_i values
- m: is the total number of input samples, called m since it corresponds to the number of rows in the input dataframe.

It can be noticed that x_train contains less samples than input dataframe, the reason being that we need to accumulate enough samples for all indicators before having a complete training sample.

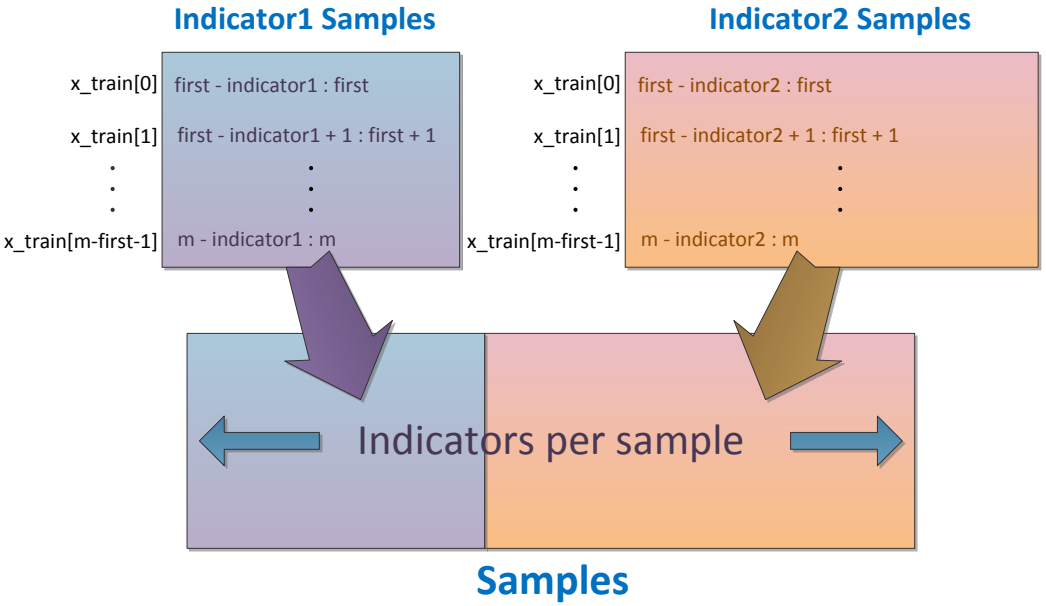

Figure 3.1: *Organization of training samples*

3.2.2 Training labels

The output of the system is a value indicating predicted stock price a number of days ahead (called the forecast horizon), as such, the word label we will use here is a stretch of the terminology used in classification nomenclature.

The system allows a flexible set of forecast horizons to be trained, with a separate model per forecast horizon. This is explained in [Figure 3.2].

Similar to training samples, the names in the figure correspond to variable names in the code.

The default horizon values used in the system are 1 day, 1 week, 2 weeks, and 1 month, corresponding to 1, 5, 10, and 20 average trading days respectively.

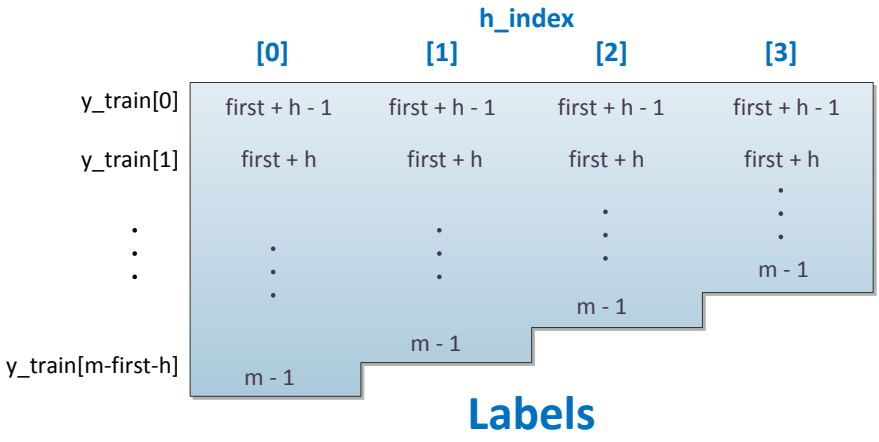

Figure 3.2: *Organization of training labels*

3.2.3 Validation set

The system outlined here calculates error metrics on validation samples at different time steps. The validation point moves with time in what is known as "rolling forecasting origin" validation[4]. At each validation point, training happens using previous time steps that are at least h ahead of the validation point, where h is the forecast horizon.

Samples are taken sparsely from the beginning of the time series (every iter_step samples) and more densely (all samples) from the last 100 samples of the time-series [Figure 3.3]. Figure 3.3 outlines the selection of samples for validation and testing. Notice that validation and testing here refer to the selection of samples only.

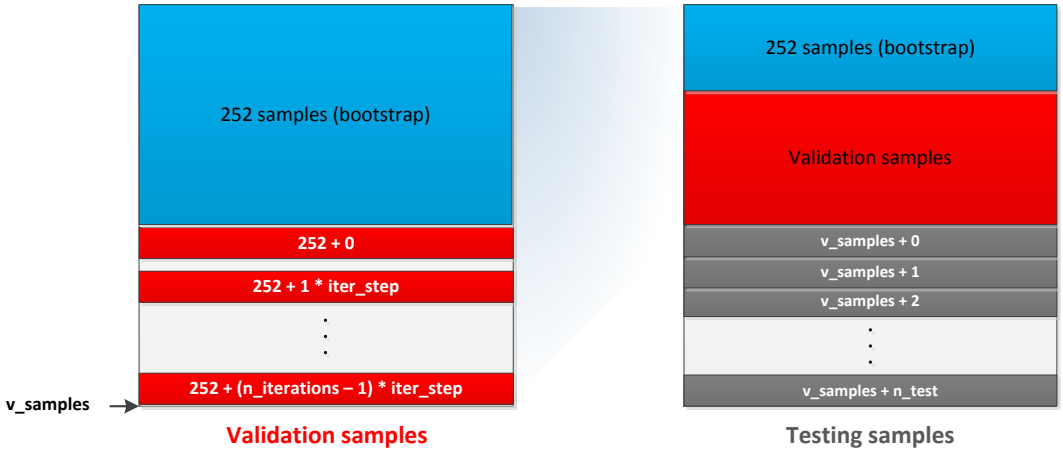

Figure 3.3: *Validation and testing sets*

The reason samples are taken more densely at the end is that most recent samples should give a better indication of the performance of the model when applied to unforeseen data that occurs beyond the last sample.

In our implementation, we take a full year worth of data samples as initialization samples. If a user requests training starting from Jul 1st 2016 for example, there is no way to make a prediction on that specific date without utilizing prior sample. One year (or 252 trading days) worth of samples is automatically attached prior to the selected training starting date, this is referenced in the blue region in the figure.

3.2.4 Training Framework

The training framework is the core of the system and is outlined in [Figure 3.4]. The first part is the viewer, which merely holds the GUI part of the training framework and exchanges user-initiated requests with the controller.

The controller talks to training master which interacts with the set of predictive models in the system through the train_validate() method.

At the completion of training, all training results are summarized in a training summery, which will be described in a following section. What matters here is that the controller receives the training summary prepared by the predictive model and caches it for future use by the testing framework.

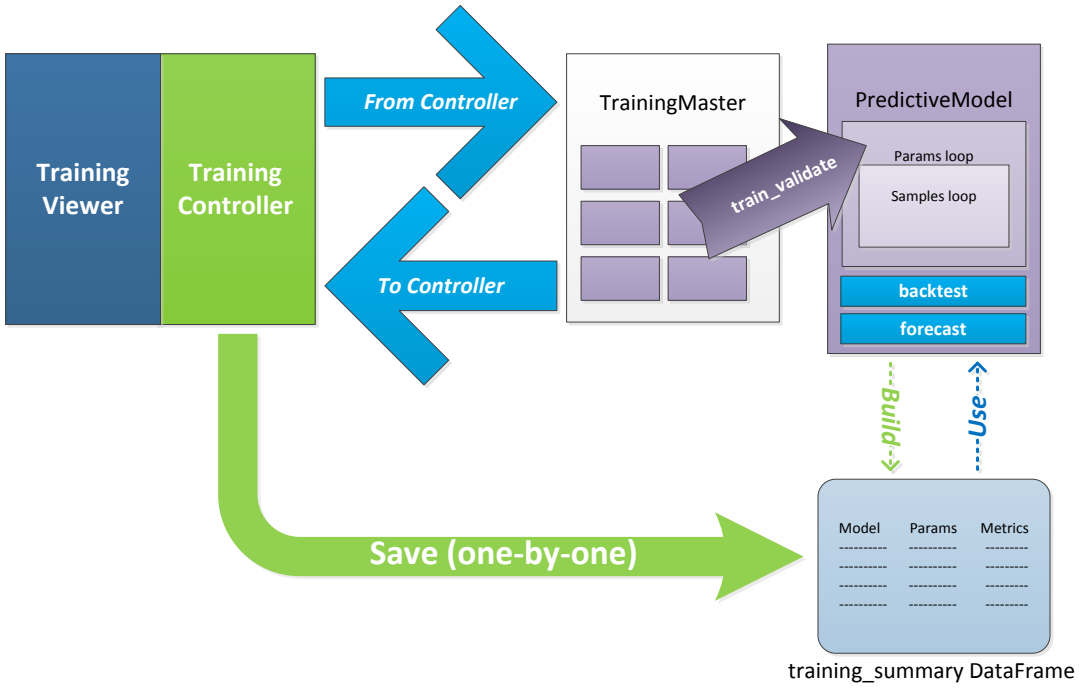

Figure 3.4: *Training framework*

3.2.5 Testing Framework

Testing framework utilizes the training summary produced by the training framework and selects the 'n' best models according to any of the saved metrics, where n is a controllable parameter.

[Figure 3.5] illustrates the testing framework, notice there is no direct communication between training and testing frameworks, the only way the two can communicate is through the training summary dataframe.

There are two viewers in the testing framework, one is called 'backtest', which applies the n-best models to a range of dates (ideally out-of-sample dates), and the other is 'forecast', which applies the n-best models to a single point. The main difference between the two is that the backtest module rebuilds the model at each test point, whereas the predict utilizes the final model built by the training framework.

The reason behind this difference is due to the difference in use case between the two: the backtest can be used to test a model outside its training range, whereas forecast is a tool for the user to make a very fast prediction using a pre-trained model.

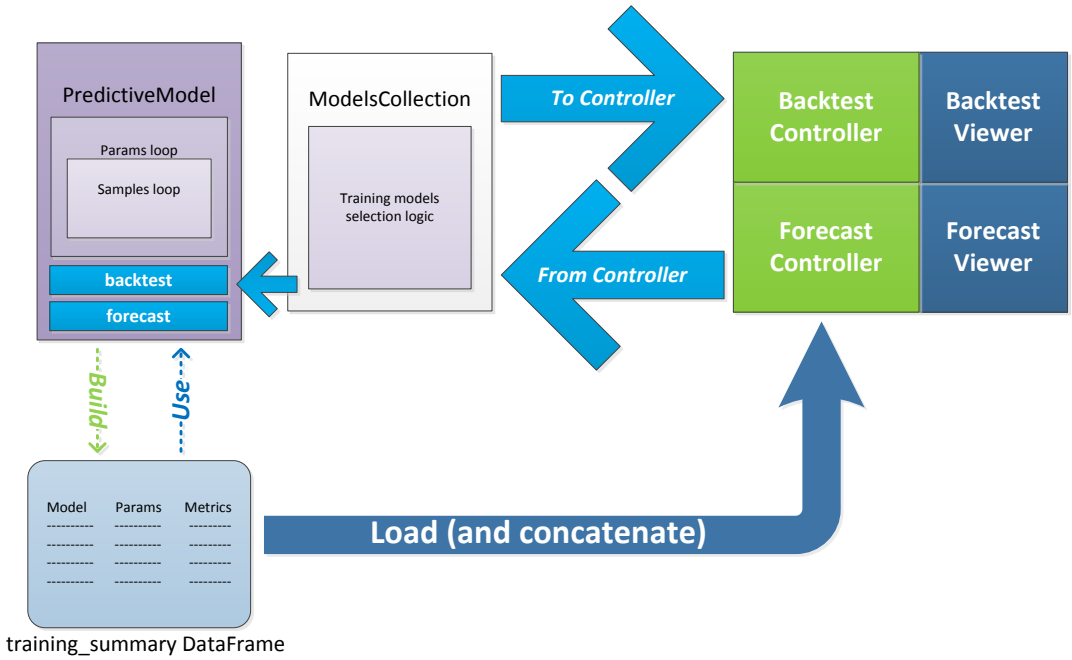

Figure 3.5: *Testing framework*

3.2.6 Training Summary

The last implementation detail we'll describe in this section is the training summary. As explained above, the training summary is a dataframe that contains the outcome of the training framework.

The training summary contains the trained model, its parameters, and all calculated metrics to allow the testing framework to chose the n-best models to create an ensemble of n models.

The cache is structured as displayed in [Figure 3.6]. This structure allows the controllers to quickly scan the trained symbols and models without opening any files, if a user selects a specific model, it then retrieves the training summary and uses it as explained in the previous section.

Within this system, the training_summary file is saved with 'trm' extension (standing for trained model), and a viewable 'csv' file is saved at the same location to facilitate viewing the results.

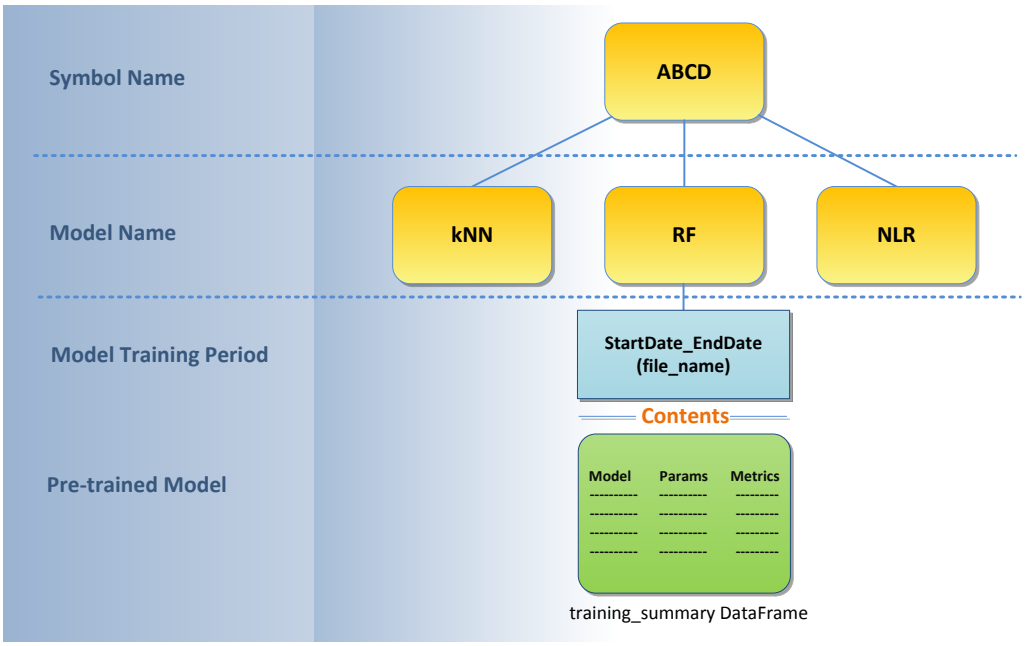

Figure 3.6: *Training summary folder structure*

4. Results and Analysis

This section begins by specifying different outputs generated by our system with an explanation of each, the following part discusses key observations from our results, and section is concluded by discussing the best performing algorithms in the list of stocks we examined.

4.1 System Output

The system discussed in this paper produces different outputs based on user request. A user request can be classified as training, testing, and forecasting, following subsections highlight the corresponding outputs:

4.1.1 Validation Metrics Table

This first output of our system is the validation results table generated in the training step. Validation results are displayed in a table as in [Figure 4.1], where each row contains a different permutations of hyper-parameters values, along with different metrics that were measured as outlined in section 3.2.3.

trained_model	params	horizon	MAE	MAPE	MSE	RMSE	hit_rate	testing_time	training_time
k-NN	('D', 1, 1)	1	1.866	0.018	6.261	2.502	0.570	0.041	0.196
k-NN	('D', 1, 1)	5	3.948	0.039	24.852	4.985	0.530	0.047	0.202
k-NN	('D', 1, 1)	10	5.888	0.058	57.167	7.561	0.482	0.042	0.177
k-NN	('D', 1, 1)	21	9.393	0.091	139.728	11.821	0.535	0.037	0.189
k-NN	('D', 1, 2)	1	1.784	0.017	5.269	2.295	0.504	0.040	0.201
k-NN	('D', 1, 2)	5	3.600	0.035	21.985	4.689	0.530	0.034	0.192

Figure 4.1: *Validation metrics table: contains the tuple of hyper-parameters, metrics measured on validation set, and training/testing times*

4.1.2 Testing Results Plot and Table

After training a model with different hyper-parameters, the model which led to the best MSE in the training-and-validation step is used in testing. The outcome of testing is displayed both as a plot [Figure 4.2, Figure 4.3], and a metrics table like the one shown in [Figure 4.1]. Difference being that no hyper-parameter tuning takes place in this step. Ideally the interval over which testing is performed is separate from the validation time interval to avoid biased outcome.

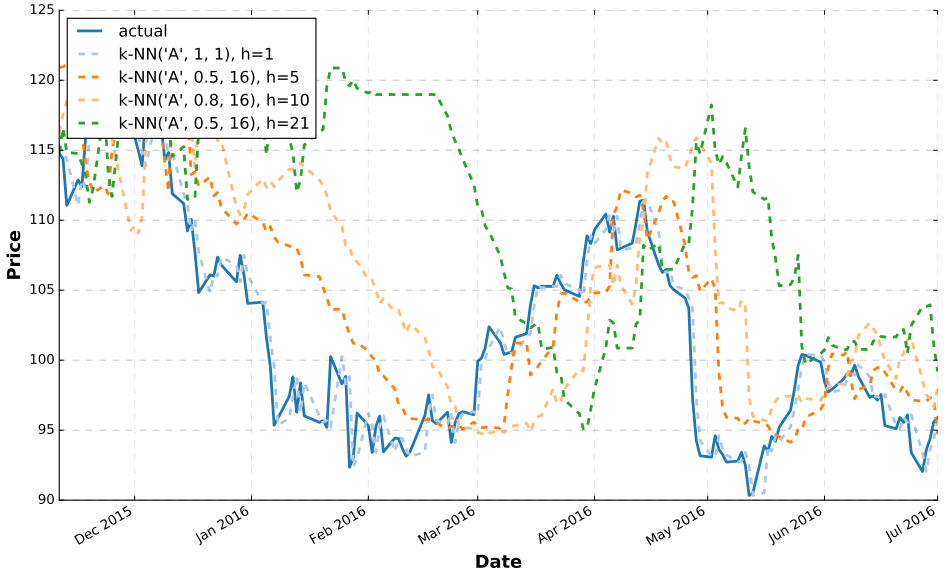

Figure 4.2: *Testing Results Plot: the legend contains the tested model, tuple of hyper-parameters that led to best MSE in validation, and forecasting horizon.*

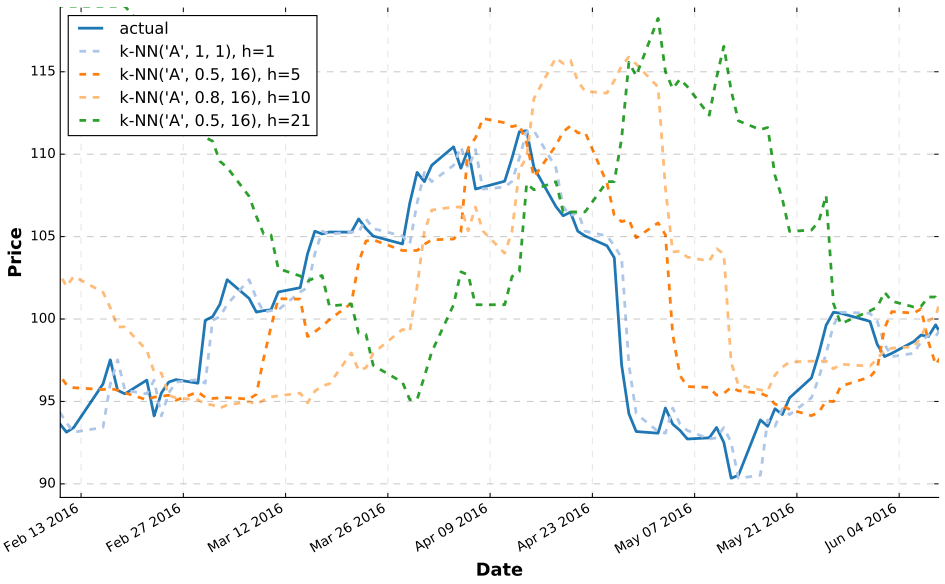

Figure 4.3: *Zooming on testing Results Plot: Further shows the delta between daily adjusted closing price, and one day forecasts generated by the model.*

4.1.3 Forecast Plot and Table

Forecast is similar to testing in that it uses the model that led to the best MST in validation, however it is optimized for efficiency since eventually this is the only step needed by external users. In addition, no metrics are measured in this step since it is assumed that points are future forecasts and do not have corresponding observations.

The results of forecasting are displayed either as a plot [Figure 4.4], or as a table with prediction value per horizon [Figure 4.5].

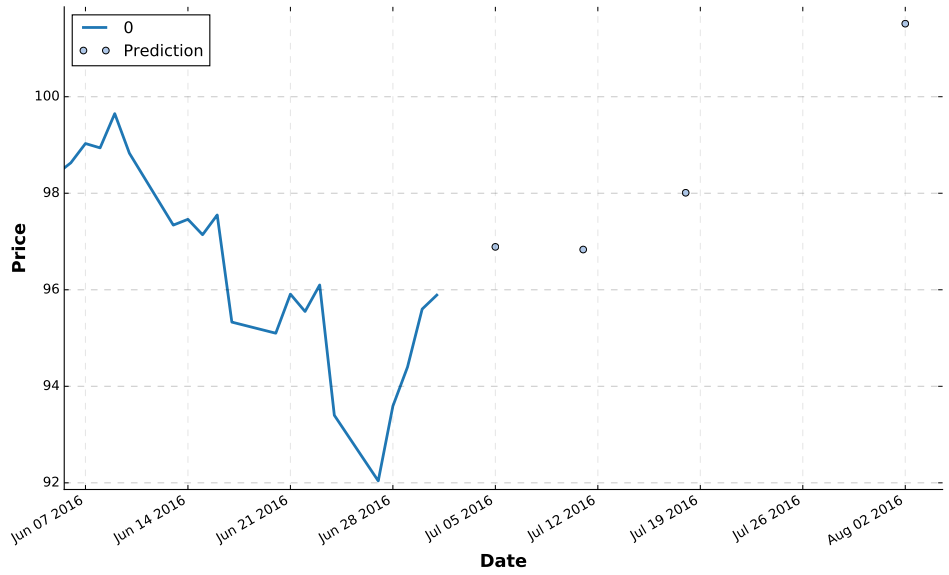

Figure 4.4: *Forecasting plot: results are appended to the last month results and indexed with proper business dates(hence the gap from July 2^{nd} to July 4^{th} in one day forecast)*

	Prediction
h = 1	96.89
h = 5	96.83
h = 10	98.01
h = 21	101.51

Figure 4.5: *Forecasting table: results are displayed in table format showing predicted stock price per horizon*

4.2 Results Analysis

In this section we'll go over results obtained from the system for 4 different symbols: AAPL, GOOG, QCOM, and SPY.

The training and validation period for each symbol is between Jan 1^{st} 2010 and Jan 1^{st} 2016. Testing period is between Jan 1^{st} 2016 and July 1^{st} 2016. All results are directly reproducible with the system.

Results are displayed first, followed by a discussion of highlighted observations. Given the multitude of obtained results, only main highlights will be discussed in this section.

In each of the boxplots in the following sections, each point represents a single permutation of the hyper-parameters of the given model.

4.2.1 Results for AAPL

Training Time Results

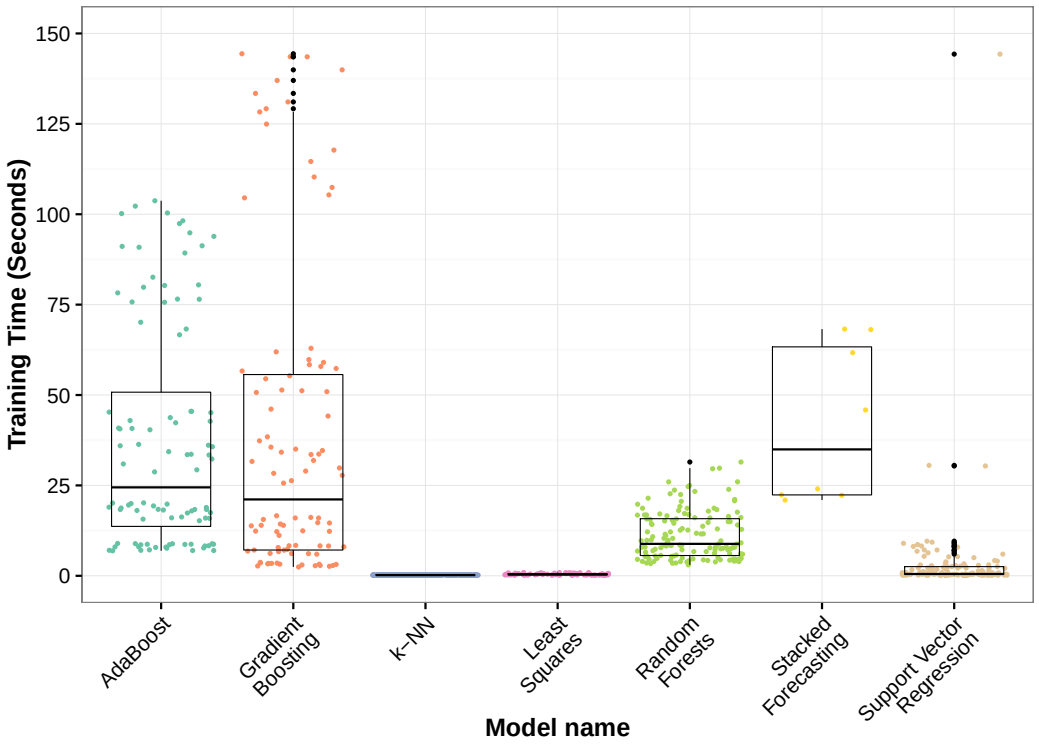

Figure 4.6: *Training time plot: displays the total training time per hyper-parameter permutation of each of the models for AAPL stock.*

As one would expect, k-NN (classified as a lazy learner) takes the least time for learning followed by least squares. Gradient Boosting and AdaBoost both show high variance in results. Specifically, permutations where a large number of indicators or large number of estimators were used negatively affected training time for both algorithms.

Validation Time Results

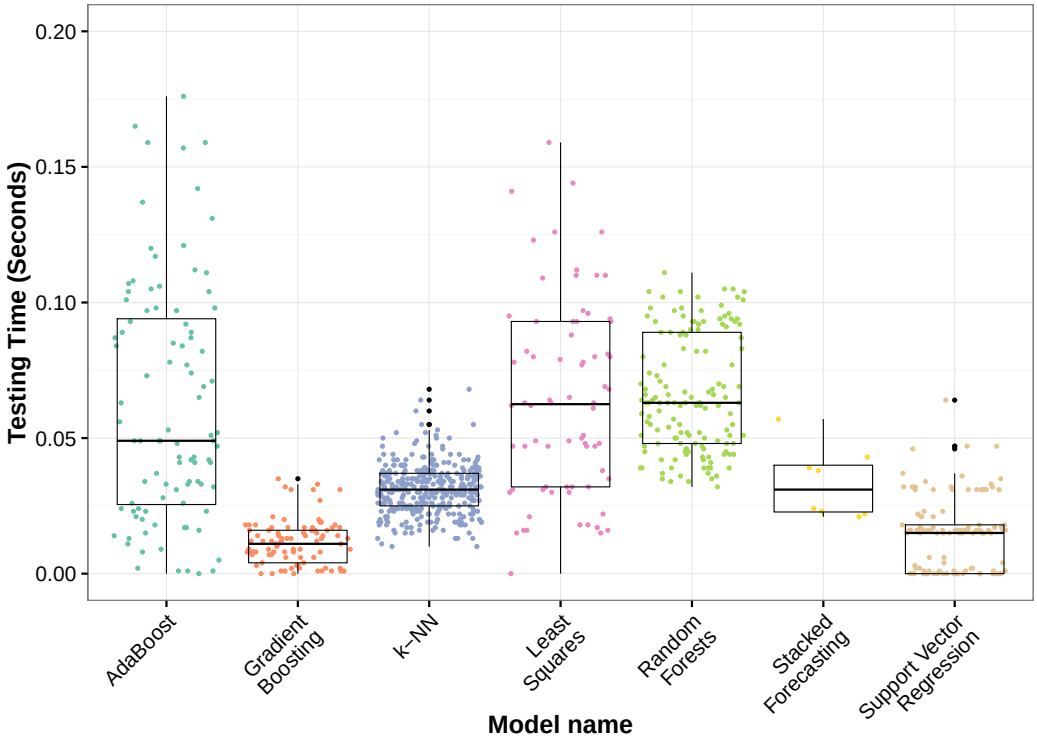

Figure 4.7: *Validation time plot: displays the total validation time per hyper-parameter permutation of each of the models for AAPL stock.*

Testing time has less variance among different predictive models. Training time for k-NN is competitive in this case, we believe the reason being that kd-tree optimization helped performance here, and that the small number of dimensions and test samples played a role as well.

It is possible that different optimizations within sklearn library helped some models more than others in testing time, so it will be difficult to read further in these results.

Validation MAPE Results

In this section we are plotting MAPE instead of MSE results since it is scale invariant. MAPE value of different stocks can be readily compared, while MSE is meaningless unless we know the stock price range. Results are split per forecast horizon since horizon value greatly affects MAPE results as expected.

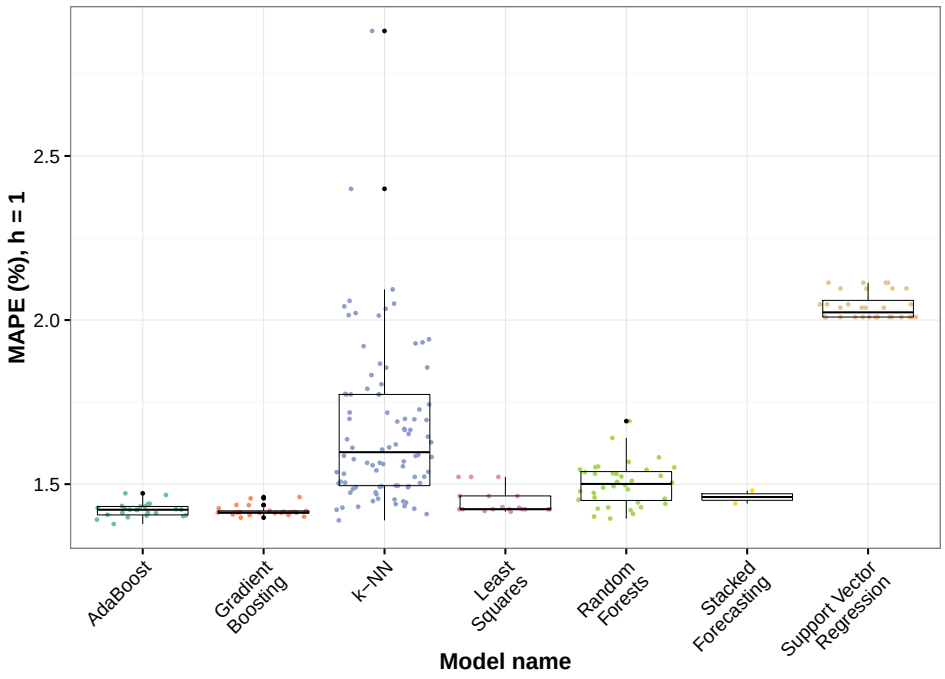

Figure 4.8: *MAPE plot: displays MAPE result for 1 day horizon per predictive model for AAPL stock. Support Vector Regression shows a slightly lacking performance compared to other regression models in this project.*

All models with the exception of SVR were more or less equivalent in terms of MAPE for 1 day horizon, k-NN had a large variation in results of 1.5%, no clear pattern could be found in k-NN results that could cause such variance in this case.

The somehow discouraging point here is that no model could do much better than k=1 with absolute values presentation, which dominantly selects the last day results for its prediction.

Our conclusion is that, for 1 day forecasts, and with any ticker symbol that has a small mean of daily percent change that is close to zero, it is hard to do much better than predicting next day price as today's price even when other indicators are added to the mixture.

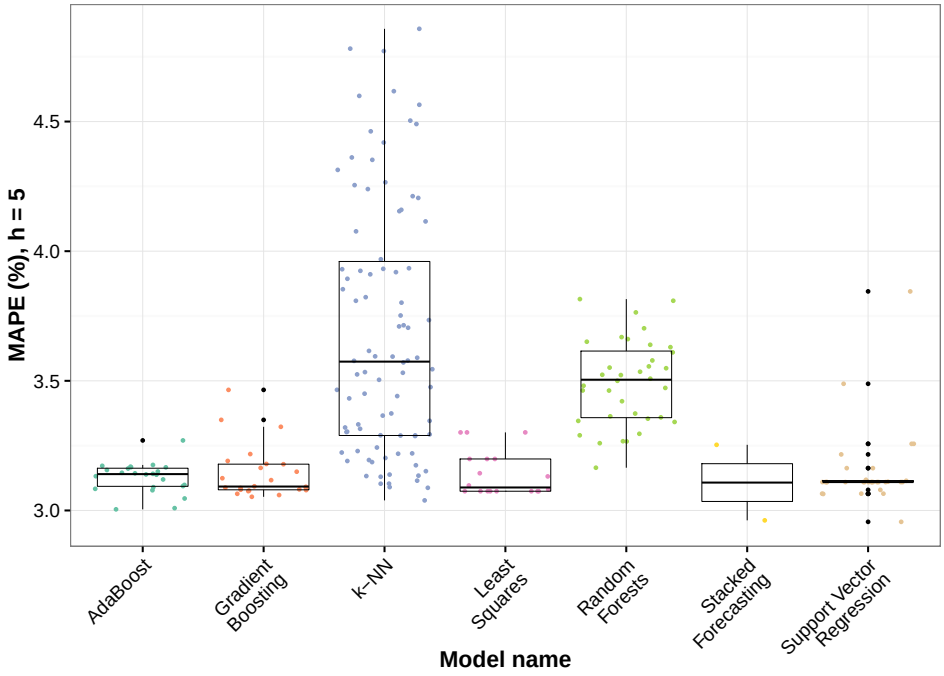

Figure 4.9: *MAPE plot: displays MAPE result for 1 week horizon per predictive model.*

At 1 week, best k-NN permutations performed better than other models, though performance difference is within 1%.

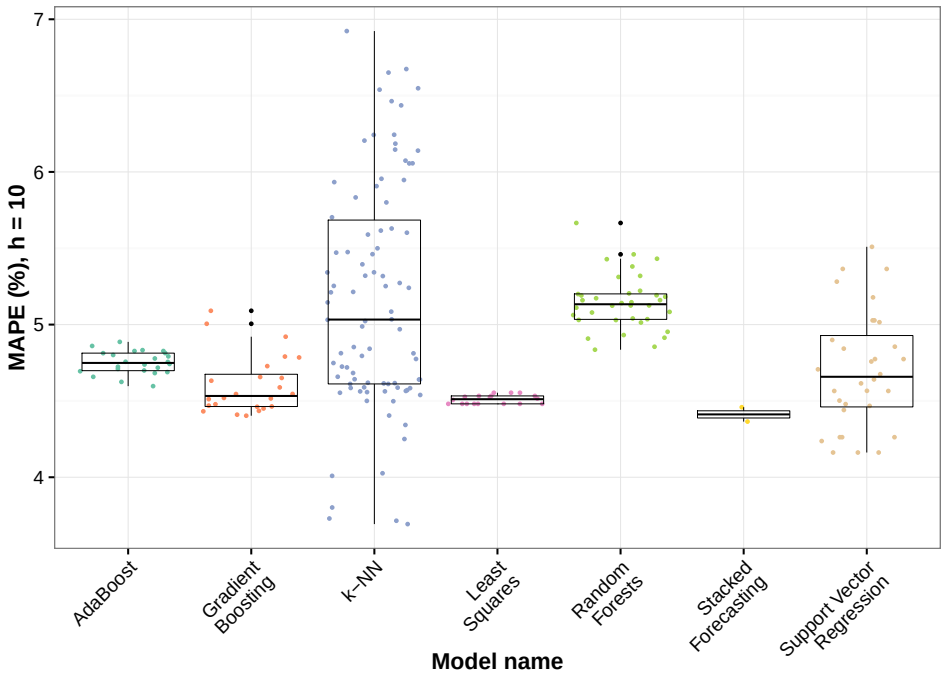

Figure 4.10: *MAPE plot: displays MAPE result for 2 weeks horizon per model.*

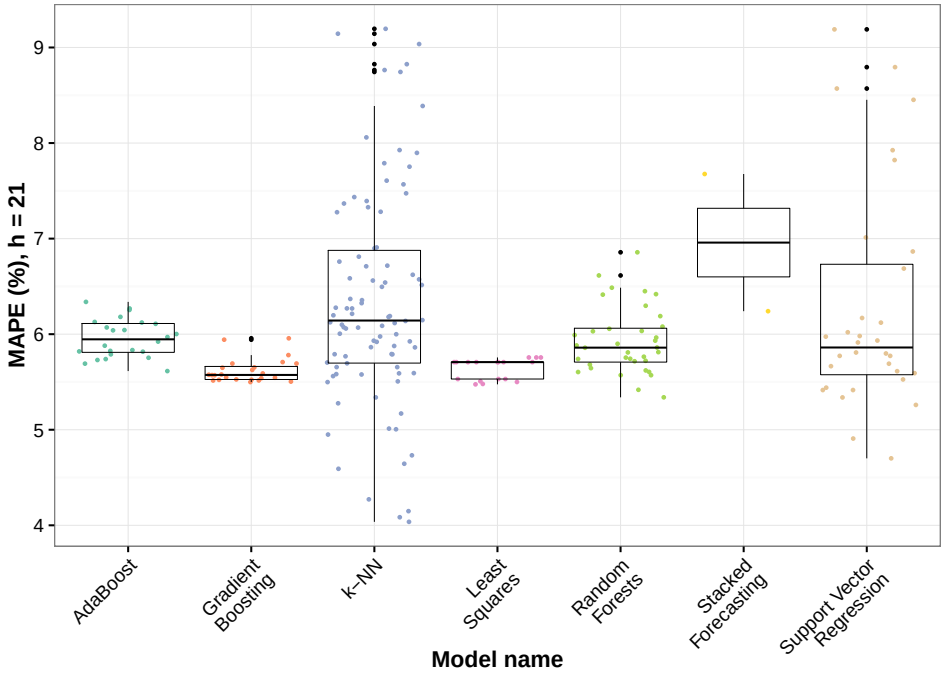

Figure 4.11: *MAPE plot: displays MAPE result for 1 month horizon per model.*

Trend continues for 2 weeks and 1 month forecasts. Bast data point of k-NN beats competition, but the variation in k-NN could mean that any specific choice of k-NN based on validation performance might lead to a surprisingly bad result on validation.

Validation Hit Rate

Validation hit rate is another important validation metric. Similar to MAPE, we split results according to forecast horizon and discuss each case separately.

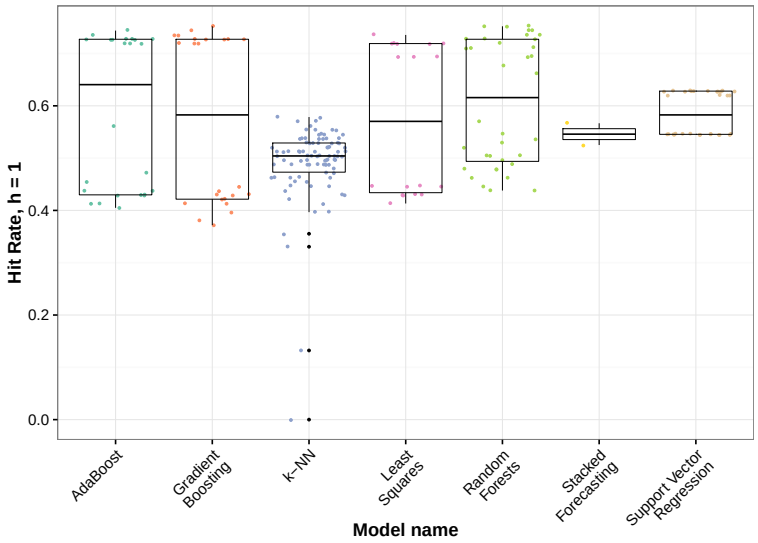

Figure 4.12: *Hit rate plot: displays hit rate result for 1 day horizon per model.*

AT h=1, most points fall above the 0.5 (or 50%)line, many points could cross the 0.6 hit rate which is well above chance probability. One interesting point is the 0.0 hit rate of one k-NN sample. A k-NN model with k=1 would mostly pick the very last sample as nearest neighbor, and hence will use that sample as its prediction. This means hit rate will suffer since any prediction that is the same as last sample (ie, no up or down trend was decided) will be calculated as a miss.

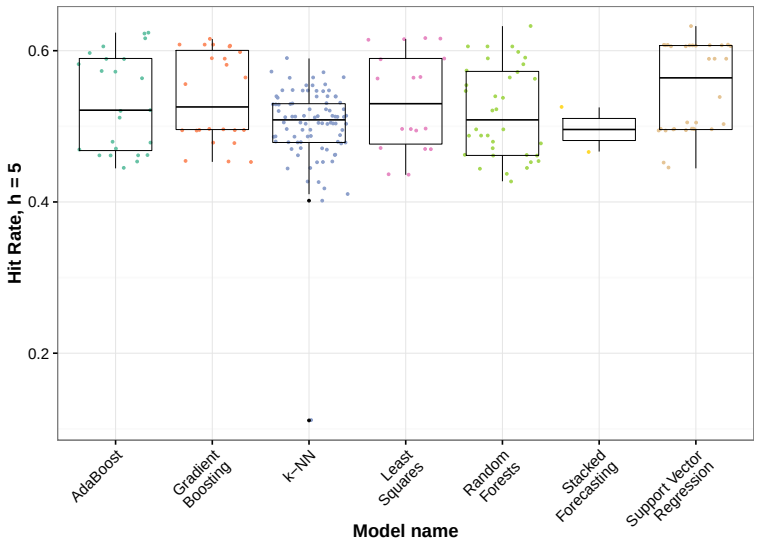

Figure 4.13: *Hit rate plot: displays hit rate result for 1 week horizon per model.*

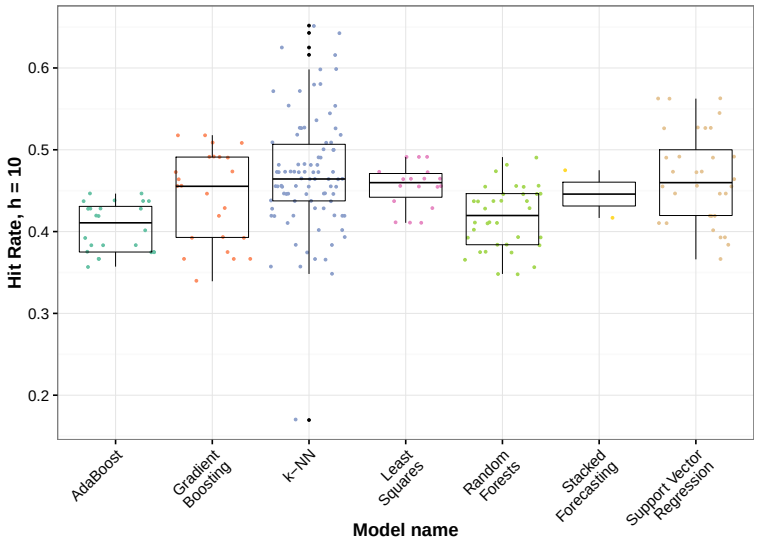

Figure 4.14: *Hit rate plot: displays hit result for 2 weeks horizon per model.*

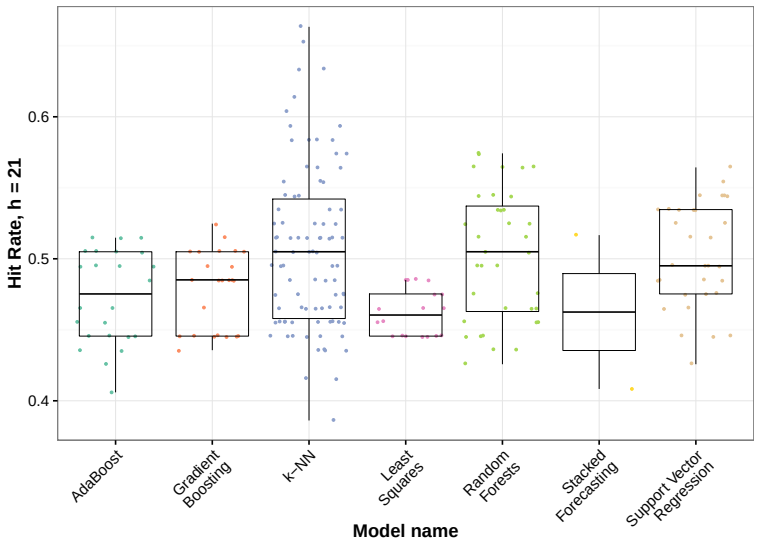

Figure 4.15: *Hit rate plot: displays hit result for 1 month horizon per model.*

Similar to h=1, many points could cross the 0.5 lines but fewer points could cross 0.6. We believe this to be due to increased volatility of the stock in 2015 for AAPL stock (recall that more validation samples are taken from the end of validation period), where volatility was deduced from the standard deviation of the daily stock price in 2015.

Validation Conclusion

After cross-validating all permutations and obtaining the results above, the system automatically selects the model with the lowest MSE, and it becomes the model used in testing phase. Notice that the best model is expected to vary based on chosen symbol and training period.

The table below summarizes the models with the winning hyper-parameters for AAPL stock:

Model	MAE	MAPE	MSE	RMSE	Hit Rate	horizon	Winning Hyper-Params	Testing T	Training T
AdaBoost	1.44	1.38%	3.74	1.93	0.56	1	('C', False, 25, 1, 'linear')	0.10	30.94
k-NN	1.46	1.39%	3.90	1.98	0.00	1	('A', 1, 1)	0.03	0.30
Gradient Boosting	1.46	1.41%	3.90	1.98	0.73	1	('C', True, 100, 0.01, 'huber')	0.03	143.54
Least Squares	1.48	1.42%	3.99	2.00	0.74	1	('C', True, 1, 1, 0)	0.09	0.58
Stacked Generalization	1.51	1.43%	4.00	2.00	0.52	1	('GB', 'C')	0.00	11.50
Random Forests	1.49	1.43%	4.03	2.01	0.44	1	('C', False, 20, 'sqrt', 2)	0.10	22.49
Support Vector	2.10	2.01%	6.78	2.60	0.63	1	('C', True, 'poly', 0.01, 0.1)	0.00	0.47
Support Vector	3.00	2.96%	16.69	4.09	0.50	5	('C', False, 'poly', 0.01, 0.001)	0.00	0.77
Stacked Generalization	3.06	2.96%	17.01	4.12	0.49	5	('GB', 'C')	0.02	11.11
AdaBoost	3.08	3.01%	17.04	4.13	0.62	5	('C', True, 25, 0.1, 'linear')	0.08	94.86
Least Squares	3.16	3.08%	17.58	4.19	0.59	5	('C', True, 1, 1, 0)	0.11	0.44
Gradient Boosting	3.14	3.06%	17.60	4.20	0.50	5	('C', False, 100, 0.01, 'lad')	0.02	57.92
k-NN	3.21	3.09%	18.26	4.27	0.50	5	('A', 0.5, 16)	0.02	0.31
Random Forests	3.30	3.16%	19.24	4.39	0.58	5	('C', True, 10, 'sqrt', 8)	0.06	12.16
k-NN	3.95	3.80%	26.10	5.11	0.62	10	('A', 0.8, 16)	0.04	0.27
Stacked Generalization	3.93	3.72%	26.69	5.17	0.58	10	('GB', 'A')	0.00	11.39
Support Vector	4.29	4.26%	28.11	5.30	0.53	10	('C', True, 'rbf', 0.1, 0.1)	0.04	0.35
Gradient Boosting	4.45	4.41%	30.99	5.57	0.51	10	('C', True, 25, 0.01, 'huber')	0.00	31.64
Least Squares	4.55	4.53%	31.97	5.65	0.43	10	('C', False, 1, 0.5, 0)	0.03	0.20
AdaBoost	4.65	4.60%	33.23	5.76	0.44	10	('C', True, 25, 0.01, 'exponential')	0.09	91.28
Random Forests	4.92	4.84%	37.47	6.12	0.45	10	('C', True, 5, 'sqrt', 8)	0.03	5.97
k-NN	4.14	4.04%	28.48	5.34	0.66	21	('A', 0.5, 16)	0.03	0.27
Stacked Generalization	4.27	4.13%	30.72	5.54	0.68	21	('GB', 'A')	0.00	10.49
Support Vector	4.70	4.70%	36.64	6.05	0.53	21	('C', True, 'poly', 0.01, 0.1)	0.00	2.55
Gradient Boosting	5.48	5.52%	45.12	6.72	0.45	21	('C', False, 5, 0.01, 'huber')	0.01	2.72
Random Forests	5.34	5.34%	45.63	6.76	0.47	21	('C', False, 20, 'sqrt', 2)	0.09	16.07
Least Squares	5.46	5.53%	45.97	6.78	0.45	21	('C', False, 1, 0, 1)	0.03	0.28
AdaBoost	5.60	5.61%	49.62	7.04	0.50	21	('C', True, 25, 1, 'exponential')	0.11	66.66

Figure 4.16: *Best models in terms of MSE from validation phase: Hyper-parameters listed here are explained in section 2.2. The models listed in this table are the ones picked automatically for testing in the following section.*

Testing Results

Testing period falls in first half of 2016 as explained earlier. Below we are listing all metrics for each model at a given horizon and highlight the most interesting observation as we did in the validation section.

Predictive Model	Horizon	MAE	MAPE	MSE	RMSE	hit_rate
AdaBoost	1	1.26	1.28%	3.02	1.74	0.45
Gradient Boosting	1	1.20	1.22%	2.81	1.68	0.50
k-NN	1	1.18	1.20%	2.74	1.66	0.00
Least Squares	1	1.20	1.22%	2.80	1.67	0.43
Random Forests	1	1.31	1.34%	3.25	1.80	0.50
Stacked Forecasting	1	1.25	1.27%	3.04	1.74	0.50
Support Vector	1	2.00	2.02%	5.52	2.35	0.48

Figure 4.17: *Testing Metrics Table: displays all metrics results for testing with 1 day forecast horizon.*

For 1 day horizon, k-NN with last sample prediction had the best MSE of all models, which is explained in earlier discussion.

Predictive Model	Horizon	MAE	MAPE	MSE	RMSE	hit_rate
AdaBoost	5	3.18	3.24%	16.21	4.03	0.40
Gradient Boosting	5	3.11	3.18%	16.31	4.04	0.43
k-NN	5	3.15	3.21%	18.34	4.28	0.48
Least Squares	5	3.11	3.17%	16.19	4.02	0.37
Random Forests	5	3.56	3.65%	19.06	4.37	0.43
Stacked Forecasting	5	3.84	3.89%	25.41	5.04	0.52
Support Vector	5	3.18	3.23%	17.34	4.16	0.52

Figure 4.18: *Testing Metrics Table: displays all metrics results for testing with 1 week forecast horizon.*

At 1 week horizon, RMSE of different models was comparable, with stacked forecasting giving the highest hit rate, and least squares is marginally giving the best MSE.

Predictive Model	Horizon	MAE	MAPE	MSE	RMSE	hit_rate
AdaBoost	10	5.18	5.32%	39.76	6.31	0.37
Gradient Boosting	10	4.63	4.74%	35.04	5.92	0.52
k-NN	10	5.26	5.41%	50.01	7.07	0.60
Least Squares	10	4.87	5.00%	38.92	6.24	0.46
Random Forests	10	5.35	5.52%	45.25	6.73	0.45
Stacked Forecasting	10	5.69	5.66%	47.15	6.87	0.51
Support Vector	10	4.96	5.04%	35.02	5.92	0.39

Figure 4.19: *Testing Metrics Table: displays all metrics results for testing with 2 weeks forecast horizon.*

As forecast horizon increases, k-NN emerges as the winning model in terms of hit rate, and gradient boosting in terms of MSE.

Predictive Model	Horizon	MAE	MAPE	MSE	RMSE	hit_rate
AdaBoost	21	8.76	8.97%	111.71	10.57	0.46
Gradient Boosting	21	8.23	8.44%	99.46	9.97	0.48
k-NN	21	8.68	9.01%	123.40	11.11	0.56
Least Squares	21	8.70	8.94%	110.57	10.52	0.40
Random Forests	21	8.66	8.87%	112.49	10.61	0.44
Stacked Forecasting	21	7.90	7.90%	85.95	9.27	0.50
Support Vector	21	9.05	9.20%	108.97	10.44	0.56

Figure 4.20: *Testing Metrics Table: displays all metrics results for testing with 1 month forecast horizon.*

Finally, at 1 month forecast horizon, k-NN is again the winner in terms of hit rate and stacked generalization in terms of MSE.

Notice that at 1 month, stacked forecasting performed equally well in validation and testing, unlike other models which performed nicely on validation but failed to perform equally well on testing. We believe this to be due to the ensemble gain, explained by the fact that stacked forecasting ensembles a number of models, allowing it to put more weight on the model that performs well on test data, unlike single models which have no such advantage, and need to use the same parameters that performed well on validation even if they are sub-optimal on testing.

We conclude here by saying that k-NN was best performer for AAPL hit rate, but apparently AAPL stock itself is very hard to predict within the chose period as the stock dealt with large volatility compared to market.

4.2.2 Results for GOOG

Results for GOOG are included in the appendix. and we arrived at a similar conclusion as to AAPL stock: that winning model is different based on the following factors:

- Whether hit rate or MSE is more concerning
- Symbol to be predicted
- Forecast horizon

Based on the items above, we chose the best model in testing. Ideally the model needs to be retrained to extend its training interval to include validation period as well before using it to make future predictions.

5. Conclusion

5.1 Reflection

In this study, a number of machine learning predictive models were investigated for the purpose of time-series forecasting. Four forecasting horizon levels were studied, and performance metrics were compared in validation and testing. The study concluded that the winning model depends on a number of factors, including: the symbol, interval, desired metric, and forecasting horizon.

Stacked forecasting emerged as a good default model that is simple to build, performs equally well on validation and testing, and is efficient to train and test.

5.2 Improvement

Future studies are planned to complete the work presented in this report:

- First, we are planning to investigate neural network based models, such as LSTM. LSTM is a recurrent model that has the potential to capture relations between multiple time steps while training a shared set of weight. Such an iterative model will be able to forecast for any desired horizon, though accumulation of errors might become an issue at larger horizons. LSTM model can be compared to other models in the same systematic manner presented here to evaluate the model's performance.
- Second, we plan to have an in-depth study of stacked generalization and how to optimize it further. Stacked generalization is commonly used to improve classification performance, and its application to time-series prediction in this project has shown decent performance. One idea which we believe might improve reported results is to apply weighting to L1 samples such that most recent samples affect L1 model more than older ones, which will add a new hyper-parameter to the model in terms of exponential weighting factor.
- Third enhancement is to integrate additional technical indicators. Traders performing technical analysis commonly rely on more complex indicators than the ones presented in this work. An example of such indicators is the moving average convergence divergence (MACD) indicator, which combines two exponential moving averages to measure momentum. Other indicators include relative strength index (RSI), stochastic oscillator, accumulation/distribution line, and others.
- In terms of the code of this project, we plan to refactor it to make the core a generic time-series framework with logging capabilities, eventually making the framework usable in other time-series applications instead of the single application it is intended for in this project.

Bibliography

[1] S&P Dow Jones Indices. (n.d.). Retrieved June 27, 2016, from http://us.spindices.com/indices/equity/sp-500-information-technology-sector

[2] Trading day. (2016, March 29). In Wikipedia, The Free Encyclopedia. Retrieved 03:23, May 30, 2016, from https://en.wikipedia.org/w/index.php?title=Trading_day&oldid=712452952

[3] Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne. "Machine learning strategies for time series forecasting." Business Intelligence. Springer Berlin Heidelberg, 2013. 62-77.

[4] Hyndman, Rob J., and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2014.

[5] McNames, James. "A nearest trajectory strategy for time series prediction." Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling. KU Leuven Belgium, 1998.

[6] Murray, Daniel B. "Forecasting a chaotic time series using an improved metric for embedding space." Physica D: Nonlinear Phenomena 68.3 (1993): 318-325.

[7] Takens, Floris. Detecting strange attractors in turbulence. Springer Berlin Heidelberg, 1981.

[8] Birattari, Mauro, Gianluca Bontempi, and Hugues Bersini. "Lazy learning meets the recursive least squares algorithm." Advances in neural information processing systems (1999): 375-381.

[9] Atsalakis, George S., and Kimon P. Valavanis. "Surveying stock market forecasting techniques–Part II: Soft computing methods." Expert Systems with Applications 36.3 (2009): 5932-5941.

[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[12] Smola, Alex J., and Bernhard Schölkopf. "A tutorial on support vector regression." Statistics and computing 14.3 (2004): 199-222.

[13] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[14] Y. Freund, and R. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, 1997

[15] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[16] Friedman, “Stochastic Gradient Boosting”, 1999

[17] Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

[18] Ting, Kai Ming, and Ian H. Witten. "Stacked Generalization: when does it work?." (1997).