Intelligent Investing: Applications of Machine Learning to Portfolio Theory

Kai Bernardini (https://github.com/kaidb/Stockalicious-)

***Abstract*— This paper is primarily concerned with the applications of machine learning to active portfolio management. In particular, using a crude framework for executing simple trading strategies, and machine learning models for predicting expected returns, portfolios can be constructed and back-tested. Depending on an investors appetite for risk, portfolios can be constructed that outperform the the S& P 500 SPY index with respect to returns, and Risk This paper compares machine learning model for predicting both price movements and percent change in price. The following models are used to predict expected returns: LASSO, Ridge Regression, AdaBoost Regression, Multi-Layer Perception Regressor, Random Forrest Regressor, Gradient Boosting Regressor, Support Vector Regressor (linear kernel, and Gaussian Kernel). Two classification models are also considered for predicting price movements for stocks. The average yearly return over all models and strategies is 10%**

I. Introduction

A. Overview of Portfolio Theory

The goal of modern portfolio theory is to consistently provide excess returns against a benchmark. In particular, given a universe of stocks, and some principal investment, a portfolio manager must distribute capital to a subset of the Universe in such a way that the overall value of the portfolio will increase. There are many potential Universes that can be considered, and even more financial instruments that can be included, but in the interest of simplicity, this paper will only consider stocks with at least 4000 days of historical data from the S&P500. The primary reason for only considering stocks from the S&P500 stems from the fact these stocks are highly liquid. In particular, there will never be a situation where a stock selected from the S&P500 cannot be bought or sold. Deviating outside of this universe into the Russel 2000 adds new real world issues that are beyond the scope of this paper. In addition to restricting the universe of stocks, a restriction is also placed on the type of trading strategy implemented. Only simple long/short strategies will be considered. Going long on a stock is buying shares in the hopes that the price will appreciate in value. Going short on a stock is when a stock is borrowed from another party and immediately liquidated. The hope is the stock will depreciate in value and after the duration of the contract, can be returned to the lender at a lower price than the initial. An example of this could be shorting one share of a stock that is currently priced at $100. After selling the one share, the value of that portfolio would be $100. Suppose after the duration of the contract, the price of that 1 share has drops to $90. Then the obligation is satisfied by buying one share at $90 and returning it to the lender. The resulting return is $10.

Every strategy for active portfolio management needs 3 thing:

1) Alpha Model: A model that forecasts excess returns of stocks. If the returns distribution is characterized by the expected return and the standard deviation, it is often the expected return that determines whether or not a stock is bought, sold or shorted.
2) Risk Model: A model that quantifies uncertainty and variability in the return of a stock.
3) Trading Strategy: A set of rules used to carryout trades based on the Alpha model and the risk model.

For this project, the risk model and trading strategy will be relatively simplistic. A rolling window standard deviation will be used to quantify risk and a simple long/short strategy is used to execute trades. For every stock in the universe $u_i \in U$, allocate $d \in \mathbb{R}$ dollars. For trading period, the alpha model will predict whether or not a stock will move up or down. If the alpha model predicts the stock will move up sufficiently high, take a long position. Otherwise, sell off all shares and/or short the stock (depending on how much the alpha model predicts the price will drop). Informally, a successful Alpha model will produce predictions for the future price of a stock that are *close* to the actual value. The alpha model is instrumental in maximizing an investors gains.

This only missing component to address is how the alpha model is actually constructed. In practice, an alpha model along with the data used to construct the model are almost always proprietary. However, with the advent of cheap and available computing power and data, it is possible to compare multiple methods for generating machine learning alpha models.

B. Overview of Machine Learning

Broadly speaking, machine learning attempts to get a machine to act without being explicitly told to do so. This paper will focus primarily on the applications of supervised learning to training alpha models. The goal of a supervised learning algorithm is to leverage a labeled data set to train a model capable of predicting the label of novel,previously unseen data points. Within the domain of supervised learning, there are two classes of problems: regression, and classification. Regression models predicts the value of a continuous, real-valued function whereas classification predicts the one of finitely many classes for a given input. An example of a regression problem is predicting the price of a stock based on its volume, as the price of a stock is a continuous value. An example of classification is predicting whether or

a stock will go up or down tomorrow (binary classification). More formally, define the following for a supervised learning algorithm:

1) A list of predictor variables $x_1, \ldots, x_p$
2) The space of predictor variables X consisting of all possible combinations of features
3) The space of all possible labels Y.
4) The ground truth function $f : X \to Y$ where for

$$\mathbf{x} \in \mathbf{X}, \mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x_1}, \ldots, \mathbf{x_p}) = \mathbf{y}$$

.

5) A Collection of Observations drawn from f (potentially with noise) $D = \{(\mathbf{x_i}, \mathbf{y_i})\}_{\mathbf{i=1}}^{\mathbf{m}}$
6) A cost function $J(y_i, \hat{y}_i)$ that is large when $\hat{f}$ is a bad approximation to f and small otherwise.

Using this notation, define a supervised learner $\hat{f}$ as

$$\hat{f} : X \to Y$$

where $\hat{f}$ is *trained* on $D = \{(\mathbf{x_1}, \mathbf{y_1}), \ldots, (\mathbf{x_m}, \mathbf{y_m})\}$ and the observations are feature vectors $\mathbf{x_i} \in \mathbf{X}$ and the labels $y_i \in Y$. For any $x_j \in X$, $\hat{f}(\mathbf{x_j}) = \hat{\mathbf{y}}_\mathbf{j}$. IE, $\hat{f}(x_j)$ yields the predicted label associated to $\mathbf{x_j}$. The goal of a supervised learning algorithm is to construct a function $\hat{f}$ such that f can be effectively approximated by $\hat{f}$. To make this more precise, let J be the cost function associated to $\hat{f}$. Then the goal of a supervised algorithm is to find the function $\hat{f}$ that minimizes $J(y_i, \hat{y}_i)$ over all i. That is,

$$\min_{\hat{f}} \sum_{i=1}^{m} J(y_i, \hat{y}_i)$$

where canidates for f are typically smooth. Notice that this is of course minimized when the model perfectly predicts the class of every data point. Since computing $J(y_i, \hat{y_j})$ requires knowledge of y_j, a common practice for performance evaluation is to randomly divide D into a training set D_{Train} and a testing D_{Test}. The model $\hat{f}$ is trained using D_{Train} and the performance of $\hat{f}$ is evaluated using D_{Test}. If the model has hyper-parameters, a validation dataset is used for hyper-parameter selection.

II. Dataset Construction

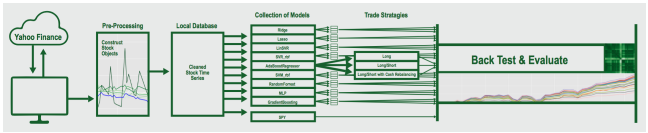
For this project, all data is derived from Yahoo Finance Historical data. Only stocks with as many historical trading days as the SPY index are included in the universe of stocks (for convenience). The format of the data is daily time series with the following original features:

- Date: (index): The date of the trading day (yyy/mm/dd)
- open: The opening market value price of a stock
- high: Highest daily market value
- low: Lowest daily market value
- close: Market value as right before the exchanges close.
- adj_close volume: The adjusted closing price on a specific date reflects all of the dividends and splits since that day. Each time a dividend is paid or a stock split declared, the adjusted closing price changes for every day in the history of the stock.
- volume: Number of shares traded for a particular day.

From there, temporal features were constructed using the time series data.

1) 2 day Adjusted closing percent change (real number)
2) 10 day Adjusted closing percent change (real number)
3) 20 day Adjusted closing percent change (real number)
4) Volatility Change(real number)
5) Daily pct change (real number)
6) percent change from 20 day rolling mean Volume (real number)
7) Percent Monthly change (regression target) (real number)
8) Positive/Negative monthly movement (Classification target) (1/0)

III. Approach



Once the data is pulled from Yahoo Finance, the model construction can begin. Start by selecting a supervised ML model. The pipeline is setup to support any supervised learning algorithm. From there, iterate over each stock and construct a stock object with the specified model. Initializing a new stock object builds an ADT and performs all of the data preprocessing and forms all feature vectors. The data is then split into 2 or 3 contiguous periods depending on whether or not hyper parameters are used. A model is then fit on the training period. For models using regularization, a validation period is used to evaluate a grid search over hyper-parameters. The highest validation score is selected as it is likely to generalize the best. Since the universe of stocks is reasonably large, this becomes computationally expensive, and a random sample of stocks is used to select hyper-parameters resulting in identical hyperparameters across all stocks for a fixed model architecture.

IV. Machine Learning Models

The generality of the back test code allowed for many candidate alpha models. All implementations are taken from sklearn. The following supervised regression algorithms were trained:

1) LASSO: ℓ_1 regularized linear regression (coefficients are restricted to an ℓ_1 ball)
2) Ridge Regression : ℓ_2 regularized linear regression (coefficients are restricted to an ℓ_2 ball)
3) Linear Support Vector Regressor: An extension to a support vector machine for regression.
4) Support Vector Regressor with Gaussian Kernel
5) AdaBoost Regressor: Boosting Regression Trees using AdaBoost.R2 algorithm
6) Random Forrest Regressor: Bagging (Bootstrap Aggregation) regression trees
7) Gradient Boosting Regressor: Boosting Regression Trees

8) Multi-Layer Perceptron Regressor: Feed Forward Neural network regressor.

The following Classification algorithms were also used:

1) Random Forrest Classifier: Bagging decision trees.
2) Support Vector Machine with RBF Kernel: Maximum margin classifier with Gaussian kernel.

A. Support Vector Machine

A support vector machine is a supervised Discriminative classification algorithm. An SVM is best described by the geometric problem that it attempts to solve. Given binary-labeled data in space, an SVM finds a surface and boundary margin that separates the data with the maximum margin.

The basic idea is to plot the training set in space. Each vector is labeled as either 1 (denoting a vector has a certain attribute) and -1 otherwise. The SVM then finds a linear boundary between two classes such that the separating boundary is as far from each class as possible. One can think of it as choosing the hyperplane, a linear surface embedded in some high dimensional space, that is halfway between the two classes. The data points that are nearest the boundary are support vectors where the collection of support vectors can be used to define the position of the hyperplane. This property makes SVMs especially useful in classifying a small list of vectors that live in high dimensional space.

V. Key Performance Indicators

There is a serious problem that needs to be addressed when it comes to scoring predictive models. In particular,there are some glaring limitations of standard cost functions when it comes to evaluating an alpha mode. This is due to the fact that the pipeline doesn't stop with the prediction; the alpha model's forecast is used to decide which stocks to invest in. In the case of a regression model, it is possible to have a small mean squared error or a high R^2 coefficient of determination despite the alpha model being horrible. If we use MSE, the penalty would be the same for a predicted value of $+2\%$ increase with ground truth +6% increase and a predicted forecast of $+2\%$ and a ground truth of -2% as

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

I.E. for the previous example, $(6-2)^2 = (-2-2)^2$. This is a problem as one prediction got the direction correct and still would have made money, where as the other got the direction wrong and would lose money. One way to get a rough idea of how well the model is performing is to look at both the MSE and the direction. IE, compute

$$sign(y \times \hat{y})$$

and count the number of positive instance. Notice this is positive when the prediction got the direction of increase correct.

In the case of classification, the confusion matrix is not useful in evaluating the performance of a trading model. If we look at the accuracy of a model, the confusion matrix will weight each mistake and reward equally. For example, if we predict a stock will go up (1) for 3 months in a row, and it goes up by 1% for 2 months then down by 10% the third month, the accuracy for this model would be $2/3$ suggesting that the model is correct most of the time. The only problem is we lost money despite being right! Class probability predictions can be used in place of hard classifications to quantify how likely the prediction is correct.

While both of these solutions for regression and classification start to address the problem, a simple solution is to just evaluate stocks its back test. While we could evaluate a stock purely on its return, this approach does not take into account the amount of risk taken on. In particular, an optimal portfolio should maximize the expected return subject to the volatility being small. For this purpose, it is necessary to introduce the Sharpe ratios for risk adjusted returns. Like the name suggests, the Sharpe ratio is a way to quantify the return on a stock when the level of risk is taken into account. In particular, the Sharpe ratio is small when there is not enough reward for the amount of risk taken on, and large when there is a large reward for the given level of risk. More formally, let σ be the standard error of the return (volatility), n be the number of units in the period (e.g. months, μ be the mean return and rf be the Risk-free rate (Eg, a government bond or Cash); then the Sharpe ratio is

$$SR = \frac{\mu - rf}{\sigma/\sqrt{n}}$$

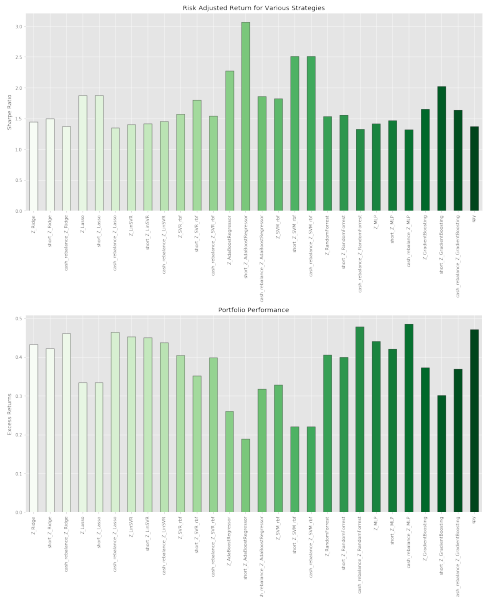
The optimal Portfolio is is chosen based on highest Sharpe Ratio. rf can be increased to account for potentially riskier strategies..

VI. Evaluation

For every combination of model, and trading strategy, a portfolio is constructed over all stocks in the universe. Each stock is given an initial capital allocation of $1,000,000 to invest. The back test is done over 44 months, where for each period, the model sees the feature vector from the current day, and predicts whether or not the stock will go up or down in the coming month (20 trading days).

- If the trading strategy is Long only, the portfolio will either be long a stock, or hold cash.
- If the trading strategy is long/short strategy, it will go long for positive predictions, short for negative predictions, and hold cash for predictions near 0.
- If the trading strategy is cash rebalanced long/short, stocks with high predictions can request idle cash, and stocks with neutral predictions can relinquish their cash.

Below are the returns and risk adjusted returns for all models across all trading strategies.



.

VII. Conclusion

The original goal of this paper was to select a portfolio that maximized the Sharpe ratio. However, due to the nature of the problem, it is possible to give a spread of portfolios. In particular, based on a portfolio manager's appetite for risk, one can offer multiple portfolios that provide variable levels of risk and return. Under our initial goal, the AdaBoost regressor is by far the best option. It consistently makes around 5% in annual returns over the back test, and does so with very little fluctuation. The long short strategy further increases the Sharpe ratio as the strategy is spreading its risk over multiple sources. However, when compared to other returns, the Adaboost regressor appears to perform poorly! The spy index consistently yielded around 12% in annual returns. The obvious benefit to investing in only the spy index is there is no cost of trading, and the bet is safe so long as the market remains stable. Now if a portfolio manager is looking for riskier options with higher returns, they should consider the MLP regressor or the Random Forrest classifier. While the Sharpe is low which means there is not enough reward for the amount of risk taken on, a portfolio manager might be temped to use it anyway. This is where RF outshines the MLP: it is interpretable. When the important features are averaged over all models for each stock in the universe, it becomes clear that the most important features are changes in volatility, and changes in volume. This makes sense intuitively, as when prices are rapidly moving up and down, people are more likely to buy/sell above the usual volume. This strikes me as a poor decision, as high volatility in alpha models typically means (informally) that the returns cannot be attributed solely to the alpha model, there was likely some luck involved. For this reason, I would personally go with the support vector classifier with an RBF kernl. The risk adjusted returns are lower than AdaBoost, but the returns are 1% higher, and therefore, a more attractive portfolio. Further, SVMs are easier to interpret than AdaBoost Regressors (see appendix).

A. Buyer Beware

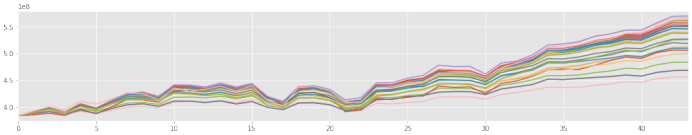
This project is first and foremost a proof of concept. Using a very limited datastream, crude trading rules and massive amounts of starting capital, it is surprising that it performs as well as it does. That said, there are many reasons why this approach is far from optimal. First, this simple POC does not take into account very real issues such as price per trade, price per short, and other such expenses. Second, while the daily purchase is capped at 1% of daily volume, it is unlikely that taking up a large position would have no effect on the price. Another major issue with this strategy is every model is correlated with the spy index. This means the model is effectively betting on the market. While some of these approaches have higher Sharpe ratios, the cost of trades likely makes it worth it to just invest in the SPY index.

VIII. Appendix

A. Acknowledgements

Special thanks to Wara Tongprasit for all of her help. Without her constant guidance and superb feedback, this project would never have gotten off the ground.

B. Portfolio Time Series



C. Building an SVM

Recall that a supervised learning algorithm takes a collection of labeled vectors (the training set) and produces a classifier that labels novel vectors. The training set D consists of a set of N training vectors $\{\mathbf{x_n}\}_{n=1}^{N}$ and corresponding class labels $\{y_n\}_{n=1}^{N}$, where $\mathbf{x_n} \in \mathbb{R}^d$ and $y_n \in \{-1,1\} \cong \mathscr{L} = \{0,1\}$. For the purposes of simplicity, the initial assumption

is made that the data can be *reasonably* separated by a hyperplane (IE the data is *linearly-separable*). The hyperplane separating the two classes is defined as:

$$\mathbf{w}^T\mathbf{x}_n + b = 0,$$

subject to the constraint

$$\begin{cases} \mathbf{w}^T\mathbf{x}_n + b \geq 1 & \text{for} \quad y_n = +1, \\ \mathbf{w}^T\mathbf{x}_n + b \leq -1 & \text{for} \quad y_n = -1. \end{cases}$$

Define H_1 and H_2 to be the hyperplanes separating the classes. The goal of an SVM is to maximize the margin M between the two classes. The objective function:

$$\begin{aligned} \max_{\mathbf{w}} \quad & (M) \\ \text{s.t.} \quad & y^n(\mathbf{w}^T\mathbf{x_n} + b) \geq M, \\ & \mathbf{w}^T\mathbf{w} = 1. \end{aligned}$$

The margin M is equal to $\frac{2}{\|\mathbf{w}\|}$. Equivalently, the target function can be represented as

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{with} \quad & y^n(\mathbf{w}^T\mathbf{x_n} + b) \geq 1 \end{aligned}$$

In the event that there is no perfect linear separation, the addition of a *slack variable* is necessary. Define the slack variables $\{\xi_n\}_{n=1}^N$ and allow some number of points to be on the wrong side of the hyperplane at some cost C (a tuneable parameter!). The gives:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n \\ \text{s.t.} \quad & y^n(\mathbf{w}^T\mathbf{x_n} + b) + \xi_n \geq 1, \\ & \xi_n \geq 0, \quad \forall n. \end{aligned}$$

Increasing C punishes harder for points on the wrong side of the hyperplane. One way to increase the accuracy is to pick a large C– however this increases the chances of constructing an over fitted model.(Note that the runtime will vary depending on choice of C).

As usual, define the Lagrangian L is given by

$$\begin{aligned} L(\mathbf{w}, b, \xi, \alpha, \mu) = & \frac{1}{2}\mathbf{w}^T \\ & \mathbf{w} + C\sum_{n=1}^{N}\xi_n \\ & -\sum_n \alpha_n \left[y_n(\mathbf{w}^T\mathbf{x}_n + b)\right] \\ & -\sum_n \alpha_n\xi_n + \sum_n \alpha_n - \sum_n \mu_n\xi_n, \end{aligned}$$

where $\{\alpha_n\}$, $\{\mu_n\}$ for $n \in \{1, 2, 3, \ldots, N\}$ are *Lagrange multipliers*.

Partially differentiating the Lagrangian with respect to w, b, ξ_n gives

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \xi, \alpha, \mu) = \mathbf{w} - \sum_n \alpha_n y_n \mathbf{x}_n = 0$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \xi, \alpha, \mu) = -\sum_n \alpha_n y_n = 0$$

$$\frac{\partial}{\partial \xi_n} L(\mathbf{w}, b, \xi, \alpha, \mu) = C - \alpha_n - \mu_n = 0$$

Solving these equations yields:

$$\mathbf{w} = \sum_n \alpha_n y_n \mathbf{x}_n \tag{1}$$

$$\sum_n \alpha_n y_n = 0$$

$$\alpha_n = C - \mu_n \tag{2}$$

Now substitute for $\mathbf{w}$, α_n to get

$$\begin{aligned} g(\alpha, \mu) &= \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} \\ &\quad -\sum_n\sum_n \alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} + \sum_n \alpha_n \\ &= \sum_n \alpha_n - \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} \end{aligned}$$

otice that in our case $\mathbf{x} = (w, b) \in R^{d+1}$ and the constraints are linear in the unknowns x. The characterization of the solution to the convex optimization (CO) problem is given by the so called Karush-Kuhn-Tucker conditions.

If f_0 and f_i's are convex, C^1-differentiable, and the feasible space has some interior points, then x^* and λ_i^*'s are the optimal solutions of the primal and dual problems $\iff$ they satisfy the following conditions:

$$\begin{aligned} f_i(x^*) &\leq 0 \\ \lambda_i^* &\geq 0, \ \forall i \in 1, \ldots, K \\ \frac{\partial}{\partial x} L(x^*, \lambda_1^*, \ldots, \lambda_K^*) &= 0 \\ \lambda_i^* f_i(x^*) &= 0 \end{aligned}$$

$$\begin{aligned} \max_{\alpha} \quad & \sum_n \alpha_n - \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m \mathbf{x_n}^T\mathbf{x_m} \\ \text{s.t.} \quad & 0 \leq \alpha_n \leq C. \end{aligned}$$

The actual theorem formulation is as follows–

Theorem 8.1: $\bar{x}$ solves the Conved Optimiaztion (CO) problem if and only if there exists a vector of non-negative Lagrange multipliers

$$\bar{\lambda} = (\bar{\lambda}_1, \ldots, \bar{\lambda}_n) \geq 0$$

so that

$(\bar{x}, \bar{\lambda})$ is a saddle point of the Lagrangean,

$$L(x,\lambda) = f(x) - \sum_{j=1}^{n} \lambda_j f_j(x).$$

i.e., for all x and for all $\lambda \geq 0$ we have,

$$L(\bar{x},\lambda) \leq L(\bar{x},\bar{\lambda}) \leq L(x,\bar{\lambda}).$$

The $\min\max = \max\min$ characterization of the saddle point of the Lagrangean L provides an alternative way to find the solution of the (CO) problem. Instead of minimizing $f(x)$ subject to the $f_j(x) \geq 0$ constraints one can equivalently maximize $W(\lambda)$, where

$$W(\lambda) = \min_x L(x,\lambda)$$

subject to the constraint that $\lambda \geq 0$. This provides an alternative route to the same saddle point of L. Therefore, using the equation (2) and KKT,the dual optimization problem is concave and can be solved in polynomial time. The dual variables (α_n) lie within a box with side length C. Now vary α_i and α_j and numerically optimize the dual function. Finally, plug in the values of the α_n^*'s to the equations (1) to obtain the primal solution $\mathbf{w}^*$.

These provide a complete characterization of the optimal hyperplane. In summary, the normal w must be a linear combination of the observed vectors x_j, the coefficients of this linear combination must add up to 0, and finally the complementarity conditions tell us that the only non-zero Lagrange multipliers λ_j are those associated to the vectors x_j right on the margin, i.e., such that,

$$y_j(\langle w,x_j\rangle + b) = 1.$$

Recall that the vectors that lie directly on the margin are called support vectors are sufficient since

$$w = \sum_{j\in J_0} \lambda_j y_j x_j$$

where $J_0 = \{j : x_j \text{ is a s.v.}\}$. The support vectors are the observations x_j at the exact distance $\rho = 1/|w|$ from the separating plane.

Once an SVM is trained, a new vector x is assigned $\hat{y}$ by computing

$$\hat{y} = w^T x$$
$$w = \left(X^T X\right)^{-1} X^T \vec{y}$$

. Support vector classification uses a least square regression which computes

$$\min_w \sum_n \frac{1}{2}\|\hat{y}^n - y^n\|^2$$

Note that we only compute the regression *onto the support vector*

$$\begin{aligned}
\min_w \ & \frac{1}{2}\|w\|^2 + C\sum_n \left(\xi_n + \hat{\xi}_n\right) \\
\text{s.t. } & y^n - w^T x - \xi_n \leq \varepsilon \\
& -(y^n - w^T x) - \hat{\xi}_n \leq \varepsilon \\
& \xi_n \geq 0 \\
& \hat{\xi}_n \geq 0
\end{aligned}$$

Although the heart of the SVM is a linear separator, it is not restricted to making linear decisions. If the data is not linearly separable, a non-linear separation can be achieved by using a non-linear mapping of the data into a higher dimensional space. The key observation is a linear decision boundary in the new space corresponds to a non-linear boundary in original space. In particular, a Mercer kernel function transform data points into a higher dimensional feature space with a *kernel function*, where the dataset can be separated again without the need to explicitly represent the points in higher dimensional space.

In particular, while the mathematically the map transforms the data to a higher dimensional space,these transformations are applied efficiently without the need to compute the data explicitly in the new space. The transformation is applied implicitly by using a *kernel function*. For the purposes of this paper, the Radial Basis Function (RBF)is the sole kernel used. The RBF is defined as

$$K\left(x,x'\right) = e^{-c\left\|x-x'\right\|^2}$$

Generally, a Kernel function is a measure of how similar two vectors x and x' are. That is, in the case that $x = x$, $K(x,x')$ is maximal. The key mathematical property of a kernel function is the existence of a function $\phi(\cdot)$ such that $K(x,x') = \phi(x)\phi(x')$. In general, once can prove that $\forall K(\cdot,\cdot)$, $\exists!\phi(\cdot)$ such that $K(x,x') = \phi(x)\phi(x')$ and vice versa.

REFERENCES

[1] Drucker, Improving Regressors using Boosting Techniques, 1997

[2] Markowitz, Harry. PORTFOLIO SELECTION. The Journal of Finance, Blackwell Publishing Ltd, 30 Apr. 2012, onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.1952.tb01525.x/abstract.

[3] BISHOP, CHRISTOPHER M. PATTERN RECOGNITION AND MACHINE LEARNING. SPRINGER-VERLAG NEW YORK

D. Performance Table

	Sharpe_Ratio	Final Return	Yearly_Return
short_Z_AdaBoostRegressor	3.063609	0.189048	0.051559
cash_rebalance_Z_SVM_rbf	2.510802	0.220823	0.060225
short_Z_SVM_rbf	2.510802	0.220823	0.060225
Z_AdaBoostRegressor	2.273211	0.260335	0.071000
short_Z_GradientBoosting	2.023472	0.301726	0.082289
Z_Lasso	1.876098	0.333595	0.090981
short_Z_Lasso	1.876098	0.333595	0.090981
cash_rebalance_Z_AdaBoostRegressor	1.863190	0.318219	0.086787
Z_SVM_rbf	1.826205	0.327771	0.089392
short_Z_SVR_rbf	1.803171	0.351136	0.095764
Z_GradientBoosting	1.652350	0.372674	0.101638
cash_rebalance_Z_GradientBoosting	1.634583	0.369170	0.100683
Z_SVR_rbf	1.570690	0.404667	0.110364
short_Z_RandomForrest	1.558308	0.400146	0.109131
cash_rebalance_Z_SVR_rbf	1.538753	0.398857	0.108779
Z_RandomForrest	1.534099	0.405973	0.110720
short_Z_Ridge	1.493654	0.421850	0.115050
short_Z_MLP	1.471305	0.420643	0.114721
cash_rebalance_Z_LinSVR	1.450390	0.436578	0.119067
Z_Ridge	1.448034	0.432530	0.117963
Z_MLP	1.419430	0.440101	0.120028
short_Z_LinSVR	1.413623	0.450206	0.122783
Z_LinSVR	1.398608	0.452846	0.123503
spy	1.374987	0.471533	0.128600
cash_rebalance_Z_Ridge	1.373571	0.460618	0.125623
cash_rebalance_Z_Lasso	1.350951	0.464234	0.126609
cash_rebalance_Z_RandomForrest	1.329300	0.477947	0.130349
cash_rebalance_Z_MLP	1.319526	0.484761	0.132208
short_Z_RandomForrestCLF	0.341374	0.416251	0.113523
cash_rebalance_Z_RandomForrestCLF	0.341374	0.416251	0.113523
Z_RandomForrestCLF	0.320232	0.445811	0.121585