Support Vector Regression vs. Locally Weighted Projection Regression: a Comparison Using Financial Data

Team O: Allan Bellahsene, Lionel Brodard
MICRO-570 Advanced Machine Learning, EPFL, Switzerland

***Abstract*—This paper's goal is to compare the Support Vector Regression (SVR) and the Locally Weighted Projection Regression (LWPR) algorithms. We use the two models to make predictions of the return in time $t+1$ of a portfolio composed of different funds using primary financial data such as gold price or exchange rate available at time t. The out-of-sample performance of both algorithms is compared. Even if the LWPR algorithm performs better than the SVR algorithm with the best nMSE of 0.729, the predictive power of the model is relatively low.**

I. Introduction

Predicting the performance of financial assets is a task of great interest to investors and speculators around the world. If done successfully, it can, of course, be extremely profitable. The advent of Machine Learning tools has thus created a lot of excitement in the financial sphere. However, the theory of efficient markets, which claims that no information available at time *t* can help to predict the price of an asset at time *t+1*, still has many defenders 50 years after its first enunciation by Eugène Fama [1]. In this paper, we compare the performance of two non-linear regression techniques on financial data: Support Vector Regression (SVR) and Locally Weighted Projection Regression (LWPR). The non-linearity exhibited by financial returns has been observed for many years now and is considered as a stylized fact describing the behavior of financial returns [2] [3]. We examine here the advantages and weaknesses of using one algorithm over the other for predicting the performance of a portfolio at time *t+1* using economic variables available at time *t*. After presenting the data in detail and its preprocessing, we discuss the algorithms implemented for this prediction task.

II. Data

The initial dataset consists of 1111 daily observations on the value of financial assets. These financial assets can be broken down as follows:

- Market indices: Close Prices of the main world stock exchange indices, namely S&P500, DOW JONES, CAC 40, DAX, EURO100, NIKKEI225 and VIX.
- Commodities: Close Prices of futures contracts on Oil, Gold and Silver.
- Main Exchange Rates: EUR/USD, EUR/CHF, EUR/GBP, GBP/USD and USD/JPY.
- 10-year treasury bond yields from the following countries
- Daily returns for the following funds: Carmigny Bond Fund (EUR), Credit Suisse Commodity Fund (US), Credit Suisse Commodity Fund (CHF), Fidelity Bond Fund (JPY), Pictet Gold Fund (JPY), and Vanguard Bond Fund (EUR).

We then build a portfolio in which we allocate an equal share to each fund. In other words, for each observation t, the return on our portfolio is given by the following equation :

$$R_t = \frac{1}{N}\sum_{i=1}^{N} r_{i,t} \tag{1}$$

where N, the number of funds, equals 5. Hence, our target variable is this daily portfolio return, which is constructed as a linear combination of five variables which are exposed in many asset classes (equity, bonds), currencies (EUR, USD, JPY, CHF) and commodities. This is why we choose our features to be the market indices, commodities, exchange rates and sovereign bonds yields presented above. But more specifically, since this prediction task is of time series nature, we must account for the fact that our variables can be non-stationary, i.e. time-dependent, which can lead to problems such as multicollinearity. We overcome this problem by simply taking the daily returns of these variables, which renders them stationary. In other words, at first our data, for each feature, is expressed in daily price, hence to obtain their returns, we use the following transformation:

$$x_{j,t} = \log\frac{p_{j,t}}{p_{j,t-1}}, \tag{2}$$

meaning, the transformed feature $x_{j,t}$ is simply the log of its price at time *t* $p_{j,t}$ divided by its price at time $t-1$ $p_{j,t-1}$ (the log-return), where j = {1,...D} is a subscript for feature. There exists a number of strong arguments in favor of log-return use [4]. One can enumerate, for example, that:

1) log-returns can be seen as continuously compounded returns
2) log-returns are time additive
3) log-returns prevent security prices from becoming negative
4) log-returns are normally distributed if the stock price follows a geometric Brownian motion
5) log-returns are approximately equal to simple returns

Finally, we preprocess the data. We thus remain with one target variable, i.e. portfolio daily return at time $t+1$, and D = 17 features expressed at time t:

- Daily Portfolio return at time t
- Daily Log return of market indices
- Daily Log return of commodities
- Daily variation of US 10-year treasury bond yield
- Daily Log return of exchange rates

III. Time Series Cross Validation

Before focusing on the different algorithms set up for this problem, it is appropriate to devote a part of this report to cross-validation. Cross-validation is what makes it possible to best estimate the performance of a model on new data. It ultimately allows us to compare different models, and thus to determine which model is better than another for a given problem. In general, k-fold cross-validation is preferred, in that it allows to split the data set as many times as wished into different trains and test splits. For a time series problem like ours, where the data has a (temporal) order, it is not possible to implement such cross-validation. Therefore, we decide to implement, for each of the two algorithms studied, a "Day-Forward Chaining" cross-validation. After defining an initial train set (for example, 70% of the dataset), it consists in successively considering each following day as the test set, then incorporating it into the train set. Then it considers the following day as the test set until we have incorporated N-1 points into the train set. This concept is illustrated in Fig. 1.

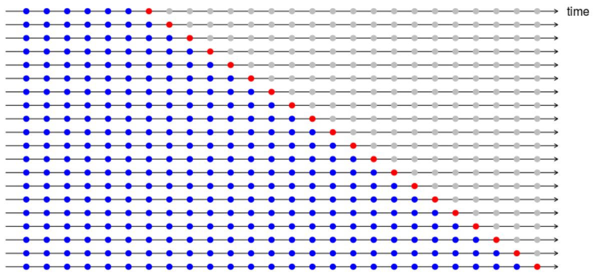

Fig. 1. Schematic representation of the time series cross-validation. Source: https://robjhyndman.com/hyndsight/tscv/.

This type of cross-validation will be systematically used to train and test each of our algorithms.

IV. Methods

A. Support Vector Regression

1) Presentation: One of the main principle of SVR is that it aims at finding a linear function of the transformed features, $f(\tilde{x}) = w^T\tilde{x} + b$, to approximate the target y such that this function at most deviates from an ϵ from y. The optimization problem for SVR can be summarized as follows:

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{3}$$

$$\text{s.t.} \begin{cases} y_i - f(\tilde{x}_i) \leq \epsilon + \xi_i \\ f(\tilde{x}_i) - y_i \leq \epsilon + \xi_i^* \end{cases} \tag{4}$$

Hence, solving the SVR algorithm aims at finding the flattest estimation of the target variable, by minimizing the norm of w while maximizing the ϵ-margin: this tradeoff is captured by the hyperparameter C, such as the higher C, the less penalized the predictions that deviate more than ϵ from the target value. The considered cost function is the so-called ϵ-insensitive cost function::

$$L_i = \begin{cases} 0 & |y_i - f(x_i)| \geq \epsilon \\ |y_i - f(x_i)| - \epsilon & \text{otherwise} \end{cases} \tag{5}$$

The transformation of the features from x to $\tilde{x}$ are implemented using the Kernel Trick, which allows a significant savings of the computational cost that if this transformation was done in preprocessing. The Kernel trick allows different transformations. We will focus on implementing three of them:

- Radial Basis Function (RBF)
- Polynomial
- Sigmoid

Hence, it is clear how such an algorithm can be interesting when predicting the evolution of financial assets prices, by allowing to capture the non-linearity stylized fact shown by financial returns with the kernel transformation, but also where ϵ can be seen as the upper limit for a loss of money.

2) Hyperparameters:

- **The margin** ϵ: The choice of the margin ϵ can be particularly problematic, especially from a computational cost point of view. Since our target variable is centered around zero, it would only make sense to search for an optimal ϵ of a very small value. The issue with that is that the lower ϵ, the higher the precision required to approximate the target variable, hence the more SVs needed to encode that, therefore the higher the computational cost. This is illustrated by Fig. 2, where we plotted the time (in seconds) to train the SVR (for a train set of 70% of the whole dataset) as a function of ϵ, and the number of Support Vectors encoded by the algorithm as a function of ϵ.
  The choice of ϵ is therefore also related to the *flateness* of our estimated function $f(x)$: when ϵ is much larger than the average observation (i.e. when no observation deviates from the ϵ-tube), the algorithm never penalizes any wrong prediction, which in turn leads to no Support Vectors at all, which therefore allows the norm w to be set to zero; while at the opposite the predictions become extremely fluctuating as the margin ϵ decreases. This is depicted in Fig. 5.

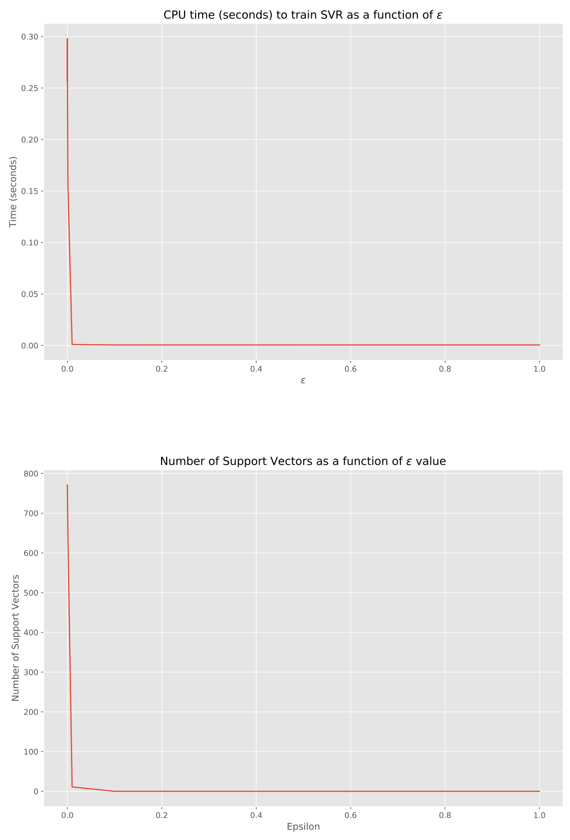

Fig. 2. CPU Train time as a function of ϵ, and number of SVs encoded by the algorithm as a function of ϵ

All these facts give motivation on the necessity of correctly tuning the margin ϵ, but the fact that its optimal value should be relatively low can be expensive if implementing Grid Search. While research has shown that implementing non-Fixed and symmetrical margin (NASM) worked particularly well when implementing SVR on financial data [7], we decide to maintain a classical fixed and symmetrical margin (FASM), but we chose to search for a margin that fits around the standard deviation of our target variable, as in our case standard deviation for financial returns is used to capture volatility. We can accept to have a higher margin when volatility is high, and, at the contrary, low margin when the volatility is low. This limitation allows us to reduce the computational cost when seeking for the optimal margin to train our algorithm.

- **The penalty term** C: C is a penalty term for a deviation of ϵ: the strength of the regularization is inversely proportional to C. This means that in theory, for a given ϵ, the higher C, the less importance is given to predictions deviating from the target more than ϵ, and hence the higher the test error should be. We test this in practice by plotting the value of the ϵ-sensitive mean squared test error as a function of C for a fixed value of ϵ: indeed, there seems to be a straightforward increasing relationship between the two variables (see Fig. 3).

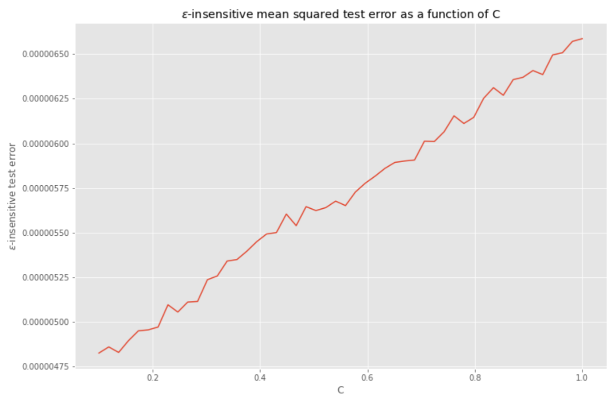

Fig. 3. ϵ-sensitive L2 Error as a function of C

Fig. 4 further illustrates this claim, where we plotted the estimations of the target variable for two different values of C. We can clearly see that the predictions of the model where we fitted a higher value of C (in purple) frequently deviate outside the ϵ-tube (in dashed lines), compared to the other predictions for the lower value of C (in blue) that never deviate outside the ϵ-tube.

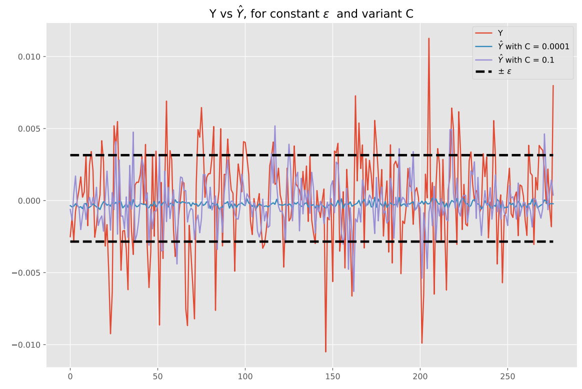

Fig. 4. Out-of-sample predictions of SVR models trained with different values of C vs. actual Y_{test}

- **Optimal hyperparameters**: The values of the optimal hyperparameters for this task are summarized in table I. They were obtained after performing a grid search, for which the range of values was chosen as described in the paragraphs above. Note that the kernel width had no impact whatsoever on our predictions, this is why we did not address it in this work.

Hyperparameter	Optimal value
ϵ	0.001185
C	0.0004295
Kernel	RBF

TABLE I
HYPERPARAMETER TUNING

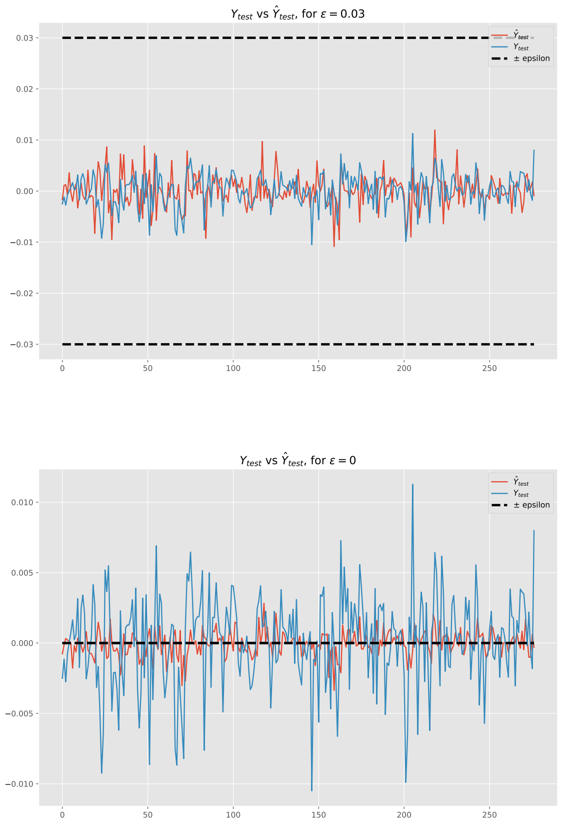

Fig. 5. *Flateness* of the predicted function f(x) as a function of ϵ

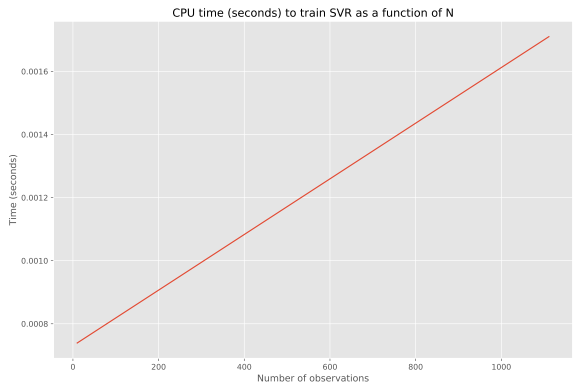

Fig. 6. Computational Train time as a function of number of observations

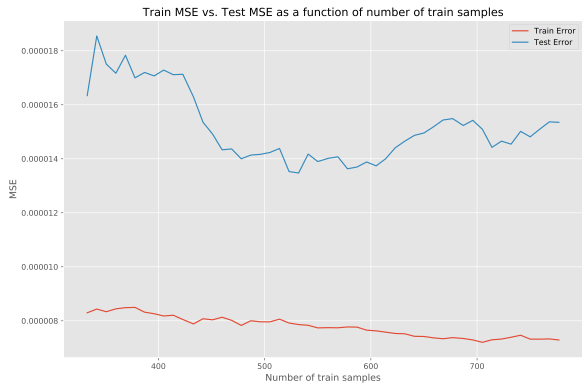

Fig. 7. Train and Test Error as a function of number of samples N in train set

3) Algorithm computational complexity and sensitivity:

- **Computational complexity**: The SVR's fitting time is known to be independent with the number of features [9]. This is clearly a significant advantage of this algorithm. However, there is a clear increasing relationship with the number of observations, as illustrated in Fig. 6.
- **Sensitivity to number of observations**: Although the SVR has proven to have a good generalization power, it clearly performs better as the number of observations in the train set increases. This is illustrated in Fig. 7, where we can see that both train and test errors decrease as the number of train samples increases.

B. Locally Weighted Projection Regression

1) Presentation: Locally Weighted Projection Regression is an algorithm that can approximate nonlinear function in high dimensional spaces using locally linear models. Vijayakumar and Schaal proposed this algorithm [10]. The idea of the algorithm is incremental learning. For each new data point, the LWPR checks if the current model can explain it well. If yes, the model remain the same, if not, it creates a new model (i.e: if $\min_k(w_k) < w_{\text{gen}}$, it creates a new model). The output of the model is the weighted average of each local linear output:

$$\hat{y} = \frac{\sum_{k=1}^{K} w_k \alpha_k}{\sum_{k=1}^{K} w_k} \tag{6}$$

The local linear output is given by:

$$\alpha_k = (XW^kX^T)^{-1}XW^kY \tag{7}$$

where the weight (using a Gaussian kernel) of each data point is given by:

$$w_{ii}^k = \exp(-\frac{1}{2}(x_i - c_k)^T D_k (x_i - c_k)) \tag{8}$$

Parameter	Tested Values	
diagOnly	{0,1}	1 to update only the diagonal of D
meta	{0,1}	1 to allow meta learning
metaRate	{0.001, 0.01, 0.1}	meta learning rate
penalty	{1e-4, 1e-3}	smoothness bias
initAlpha	{1e-5 1e-4 1e-3}	initial α / learning rate
initD	{ 1e-4 1e-3 1e-2 }	initial D / distance metrics
wGen	{1e-4 1.5e-4 2e-4}	threshold
initLambda	{1e-5 1e-4 1e-3 }	initial λ (forgetting factor)
finalLambda	{0.99 0.9999}	final λ
tauLambda	{0.01 0.05}	τ λ

TABLE II
HYPERPARAMETERS

Parameter	Optimal Value
diagOnly	1
meta	1
metaRate	0.01
penalty	1e-4
initAlpha	1e-3
initD	1e-2
wGen	1e-4
initLambda	1e-4
finalLambda	0.99
tauLambda	0.05

TABLE III
HYPERPARAMETERS

The distance metrics D_k is learned for each local model. It uses a stochastic descent in a penalized leave-one-out cross-validation cost function. The update rule can be written as:

$$D = M^T M \tag{9}$$

$$M^{n+1} = M^n - \alpha \frac{\partial J}{\partial M} \tag{10}$$

where the cost function J is given by:

$$J = \frac{1}{\sum_{i=1}^{M} w_{ii}^k} \sum_{i=1}^{M} w_{ii}^k ||y^i - \hat{y}_k^{i,-i}||^2 + \gamma \sum_{i=1,j=1}^{M} D_{k,ij}^2 \tag{11}$$

2) Hyperparameters: The LWPR algorithm is based on several hyperparameters that are listed on table II. First, we implemented a Grid Search to find the optimal set of hyperparameters without Day-Forward Chaining Validation, saving a lot of computational time.

- **Optimal hyperparameters**: We tested the 7776 combination. The results in term nMSE for the test are shown in Fig. 8. For each parameter, we are interested in the value that gives the lowest nMSE (and not the lowest median of nMSE). The optimal set of hyperparameters is shown on table III. It results in a nMSE of 0.729.

3) Algorithm computational complexity and sensitivity:

- **Computational complexity**: The computational cost of the LWPR increases linearly in the number of inputs. This can be shown in Fig. 9. This is a real advantage when it comes to high dimensional databases.
- **Sensitivity to number of observations**: LWPR is sensitive to the number of observations. Even if at first, an increase in data size increases the test nMSE, the normed test error rapidly decreases as a function of train data size. This can be shown in Fig. 11.

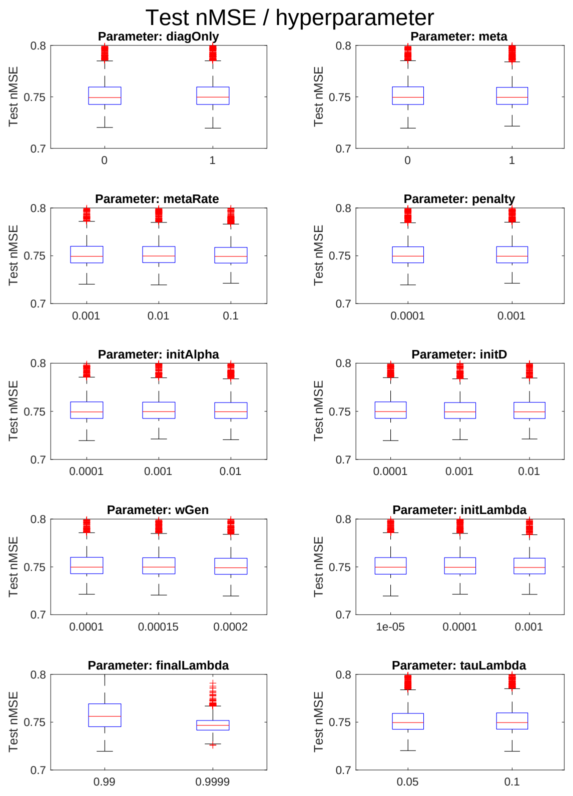

Fig. 8. Test nMSE based on the hyperparameters

Algorithm	Best test nMSE	Computational Cost
SVR	0.822	Independent of D, quadratic in N
LWPR	0.729	Linear in D and N

TABLE IV
RESULTS

V. RESULTS

We now compare the performances of our best algorithm, after tuning for the optimal hyperparameters. The results are on table IV. The predicted returns from our two algorithms are shown in Fig. 11.

A. Your analysis

- **Performance**: First, we can see that the two algorithms perform quite poorly. The predicted returns of both models are far from the actual return. Even if they often indicate the right direction (positive or negative returns), they tend to be too smooth. The LWPR seems to perform a bit better, with the best nMSE of 0.729. The SVR can only achieve a nMSE of 0.822. This is still not enough to predict the return of the portfolio in $t+1$ accurately. It seems that the LWPR is better suited for such models. Indeed, the LWPR can incrementally learn, which is an excellent feature to predict financial markets with dynamic changes.

- **Computational Time**: It's impossible to compare both computational times as the LWPR was implemented in MATLAB and the SVR in python. However, we know that the SVR computational cost is independent of D and quadratic in N (where D is the number of dimensions, and N the number of observation), while the LWPR is linear in D and N (see Fig. 9). It seems, therefore, that LWPR would be a better algorithm in our case, especially if we would have taken a more extensive database of asset prices. Both algorithm have long computational steps (especially LWPR).
- **Hyperparameters**: For both algorithms, and especially for the LWPR, the hyperparameters play a considerable role. A more extended search, the use of Bayesian optimisation to find the optimal set of parameters may have improved the performance of both algorithms.

VI. Conclusion

The prediction of financial returns and asset prices using machine learning is trendy. All asset managers want to code the best algorithm to predict the future. We showed that the SVR and LWPR, that are advanced machine learning algorithms, performed quite poorly to predict the actual return of asset prices. The LWPR seems more suitable for such exercise, as it performs empirically better, and is theoretically better due to its incremental leaning feature. To improve our predictions, we could have tested other data. We used common data in financial markets, but its correlation with the price of the funds was relatively low. It might explain why the algorithms performed poorly. Besides, a more in-depth search of the optimal set of parameters might also have improved the performance of the two algorithms. Finally, predicting an asset price in $t + 1$ using information available at t is against the financial theory. It is also a reason why the predictive power is low. The prediction of financial return remains a very delicate exercise even with advanced machine learning techniques.

References

[1] Fama, Eugene F., *Efficient Capital Markets: A Review of Theory and Empirical Work*, The Journal of Finance, 1970

[2] Kantz, H. and T. Schreiber, *Nonlinear Time Series Analysis*, Cambridge, 2003

[3] Hsieh, David A., *Chaos and Nonlinear Dynamics: Application to Financial Markets*, The Journal of Finance, 1991

[4] Hudson, Robert and Gregoriou, Andros, *Calculating and Comparing Security Returns is Harder than you Think: A Comparison between Logarithmic and Simple Returns*, International Review of Financial Analysis, 2010

[5] Hommes, C.H., *Financial markets as nonlinear adaptive evolutionary systems*, Quantitative Finance, 2001

[6] Hiemstra, C. and J.D. Jones, *Testing for Linear and Nonlinear Granger Causality in the Stock Price-Volume Relation*, The Journal of Finance, 1994

[7] H.Yang, L. Chan, I. King, *Support Vector Machine Regression for Volatile Stock Market Prediction*, The Chinese University of Hong Kong, 2002

[8] Q. Chang, Q. Chen, X. Wang, *Scaling Gaussian RBF kernel width to improve SVM classification*, ITNLP Lab, School of Computer Science and Technology

[9] A. J. Smola and, B. Scholkopf, *A Tutorial on Support Vector Regression*, 2003

[10] Vijayakumar, Sethu and Schaal, Stefan, *Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space*,Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), 2000

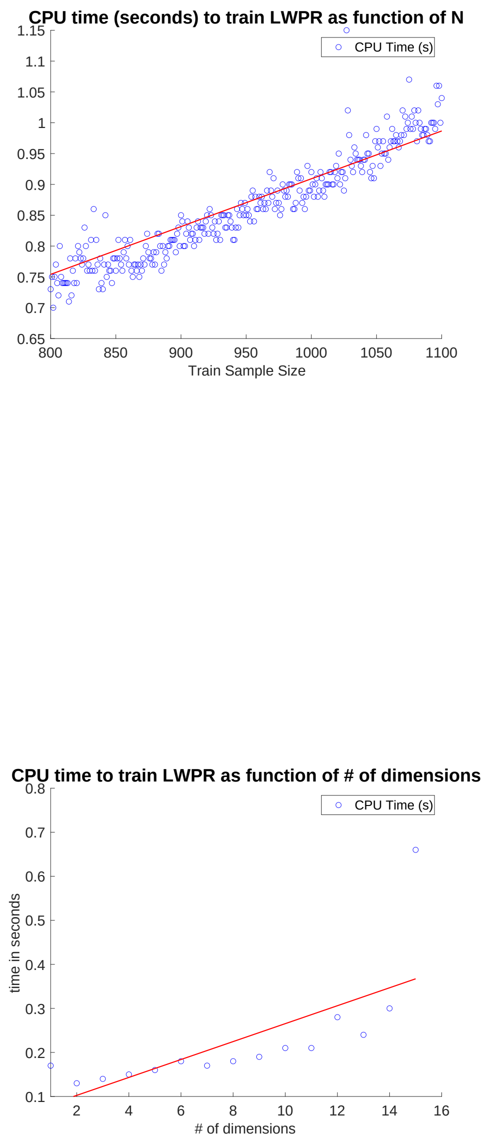

Fig. 9. CPU train time for LWPR as function of N and D

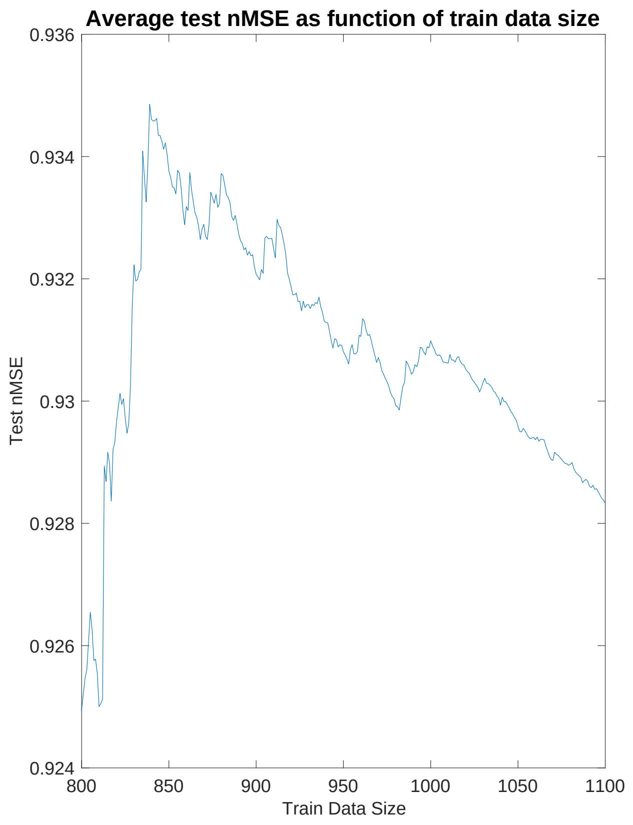

Fig. 10. Test nMSE as a function of number of samples N in train set

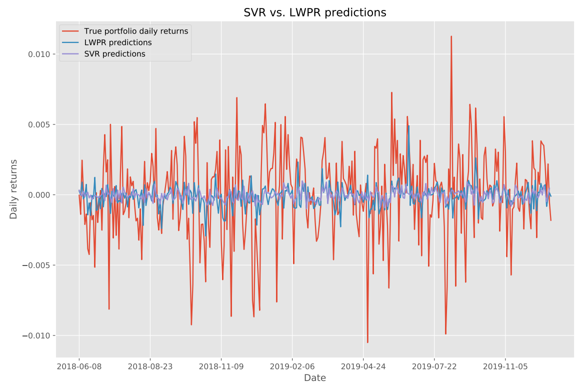

Fig. 11. Test nMSE as a function of number of samples N in train set