Lag Correlation Discovery and Classification for Time Series

Georgios Dimitropoulos[1], Estela Papagianni[2] and Vasileios Megalooikonomou[2]
[1]*Department of Mathematics, University of Patras, Rio-Patras, Greece*
[2]*Department of Computer Engineering and Informatics, University of Patras, Rio-Patras, Greece*

Keywords: Data Mining, HSLC Algorithm, Data Streams, Time series, Classification, Lag Correlation, Incremental SVM Learning Algorithm, Stock Trend Prediction, Features Extraction.

Abstract: Time series data are ubiquitous and their analysis necessitates the use of effective data mining methods to aid towards decision making. The mining problems that are studied in this paper are lag correlation discovery and classification. For the first problem, a new lag correlation algorithm for time series, the Highly Sparse Lag Correlation (HSLC) is proposed. This algorithm is a combination of Boolean Lag Correlation (BLC) and Hierarchical Boolean Representation (HBR) algorithms and aims to improve the time performance of Pearson Lag Correlation (PLC) algorithm. The classification algorithm that is employed for data streams is an incremental support vector machine (SVM) learning algorithm. To verify the effectiveness and efficiency of the proposed schemes, the lag correlation discovery algorithm is experimentally tested on electroencephalography (EEG) data, whereas the classification algorithm that operates on streams is tested on real financial data. The HSLC algorithm achieves better time performance than previous state-of-the-art methods such as the PLC algorithm and the incremental SVM learning algorithm that we adopt, increases the accuracy achieved by non-incremental models.

1 INTRODUCTION

The high volume of time series data in various domains (e.g. medicine, financial markets) necessitates the use of effective data mining methods to aid towards decision making. The mining problems that are studied in this paper are lag correlation discovery and classification. The purpose of lag correlation discovery is to detect correlations with lag that are present in time series. The most common way for detecting correlation is the Pearson correlation coefficient. In many applications, due to the fact that there is a great number of time series that have a high correlation, it is necessary for a method to accept only the pairs that have correlation greater than a threshold. Such a method is the Pearson Lag Correlation (PLC) algorithm that can be used, not only for time series, but also for data streams. However, for online applications, the time performance of the algorithm is questionable. There is a rich body of literature on lag correlation algorithms for data streams. The Muscles algorithm proposed by Yi et al. (2000) is able to adapt to changing correlations on data streams, handle a great number of long sequences efficiently and predict precisely the missing values. Another approach is the StatStream incremental algorithm proposed by Zhu et al. (2002) which is based on Discrete Fourier Transform (DFT) and a time interval hierarchy of three levels, to calculate efficiently high correlation among all pairs of streams. The Boolean Lag Correlation (BLC) algorithm proposed by Zhang et al. (2011) and the Hierarchical Boolean Representation (HBR) algorithm proposed by Zhang et al. (2009) calculate lag correlation among multiple data streams and are based on boolean representation. HBR is based on the fact that, in the majority of applications, only a small fraction of all possible pairs have a high correlation. Thus, in order to reduce the computational time, a technique that spots only the pairs which are correlated is used.

The purpose of classification for time series on the other hand is to create a matching of a set of unknown features into some known categories. For classification of data streams there is a rich body of research work related to incremental SVM learning algorithms. A framework for exact incremental learning and adaption of SVM classifiers has been studied by Poggio et al. (2001) and Diehl et al. (2003). The algorithm that is used is able to learn

Dimitropoulos, G., Papagianni, E. and Megalooikonomou, V.
Lag Correlation Discovery and Classification for Time Series.
DOI: 10.5220/0006215901810188
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 181-188
ISBN: 978-989-758-245-5

and unlearn manifold examples, adapt the current SVM to changes in regularization and kernel parameters and evaluate generalization performance.

In terms of applications, the medical and macroeconomic environments are fields in which there has been extensive research regarding classification algorithms of time series. The prediction of certain events in EEG or ECG has been studied among others by Mporas et al. (2015). Similarly, indicators in the financial market and recognition of patterns in stocks, known as stock trends, has been studied among others by Edwards et al. (2007). Kim (2003) has used SVMs for the prediction of the trend of the daily Korea Composite Stock Price Index (KOSPI).

In this paper, we focus on lag correlation algorithms for time series and classification algorithms for data streams and examine their applications on medical and financial data respectively. The main contributions of this work can be summarized as follows:

- *Better time performance on lag correlation discovery of time series*: the new HSLC algorithm achieves better computational time than the PLC algorithm.
- *Better accuracy on classification of data streams*: we improve the classification accuracy accomplished by Kim (2003) on the KOSPI dataset by employing the incremental SVM learning algorithm proposed by Diehl et al. (2003) concluding that the way of training models for streams should be incremental.

In Section 2 the Highly Sparse Lag Correlation (HSLC) is presented, whereas in Section 3 the incremental SVM learning approach is analyzed. Both algorithms are experimentally studied in Section 4 followed by some remarks. Section 5 concludes the paper.

2 HIGHLY SPARSE LAG CORRELATION ALGORITHM

In the case of a large number of time series or data streams the set of all possible correlated pairs is rapidly increased. In order to be able to examine them for possible correlation it is vital to use algorithms which limit calculations to a smaller set and provide faster implementation through accurate approximations. The proposed approach, HSLC, calculates lag correlation on time series and can be used in applications where the number of highly correlated pairs is much smaller than the number of all possible pairs. Given k time series of length *n*, the algorithm calculates lag correlation by first calculating the pairs that have lag correlation which is greater than a threshold and returning these correlated pairs and the value of their lag *l*. It is based on the HBR To cope with the high computational cost, the algorithm uses a correlated pair detection technique based on the BLC algorithm including the Boolean Representation method. After that, the lag correlation is calculated for the set of pairs which were detected using the Pearson correlation coefficient. Before the presentation of the HSLC algorithm, we give some definitions:

Definition 2.1 (**Pearson correlation coefficient)** The Pearson correlation coefficient for two time series X and Y with the same length n for lag l with $0 \le l \le \frac{n}{2}$ is given by the formula:

$$r(lag) = \frac{\sum_{t=l+1}^{n} (X_t - \bar{X}) \cdot (\mathrm{Y}_{t-l} - \bar{Y})}{\sqrt{\sum_{t=l+1}^{n} (X_t - \bar{X})^2} \cdot \sqrt{\sum_{t=1}^{n-l} (\mathrm{Y}_t - \bar{Y})^2}} \tag{1}$$

where:

$$\bar{X} = \frac{1}{n-l} \cdot \sum_{t=l+1}^{n} X_t \tag{2}$$

$$\bar{Y} = \frac{1}{n-l} \cdot \sum_{t=1}^{n-l} Y_t \tag{3}$$

Definition 2.2 (**Boolean series)** The transformation of a time series *X* of length *n* into the corresponding Boolean series *W* of the same length *n* is given by:

$$w_t = \begin{cases} 1, x_t > \bar{x} \\ 0, x_t \le \bar{x} \end{cases} \tag{4}$$

where:

$$\bar{x} = \frac{1}{n} \sum_{t=1}^{n} x_t \tag{5}$$

Definition 2.3 (**Correlation coefficient for Boolean series)** The calculation of the lag correlation coefficient for two Boolean series (W_i, W_j) is given by the formula:

$$\delta(l) = \max\left\{\delta_{pos}(l), \left|\delta_{neg}(l)\right|\right\} for\, l = 0,1,...,\frac{n}{2} \tag{6}$$

$$\delta_{pos}(l) = 1 - corr(l) \tag{7}$$

$$\delta_{neg}(l) = -corr(l) \tag{8}$$

$$corr(l) = \frac{\sum_{t=l+1}^{n} w_{t-l}^{i} XOR w_{t}^{j}}{n-l} \quad (9)$$

The proposed algorithm is described below:

HSLC algorithm

```
Input: $k$ time series $X_1, X_2, ..., X_k$
$detection\_threshold, correlation\_threshold$, $C$.
// detection threshold: used to find
if the boolean series have a
correlation; correlation_threshold:
used to find if the time series,
which were detected through the
previous calculation of their
corresponding boolean series, have a
correlation; set C: contains the
pairs of time series $(X_i, X_j)$ which have
corresponding boolean series with
correlation coefficient bigger than
the detection_threshold //
Output: $(X_i, X_j)$, max, lag
// $(X_i, X_j)$: the correlated pairs (if
any); max: the maximum value of the
Pearson correlation coefficient for
every possible value of the lag $l$ for
the pairs $(X_i, X_j)$; lag: the value of
$l$ where the Pearson correlation
coefficient takes its maximum value//

$C \leftarrow \varnothing$
for each time series $X_i$ do
   Calculate $\bar{X}_i$; //mean value of $X_i$//
   Calculate $W_i$; //the transformed
   Boolean series of time series $X_i$
   calculated by eq. 4 (Def. 2.2.)//
end for
for each pair of series $W_i$ and $W_j$ do
   $max = 0$; //maximum positive or
   negative correlation//
   for $l = 0$ to $n/2$ do
      Calculate the Boolean
      correlation coefficient $\delta(l)$
      using eq. 6(Def. 2.3)
      If $|\delta(l)| > |max|$ then
         $max = \delta(l)$;
      end if
   end for
   if |max|>=detection_threshold then
      Add the pair$(X_i, X_j)$ to the set $C$
   end if
end for
for each pair of time series $X_i$ and
$X_j$ that belong to the set $C$ do
   $max = 0$; //maximum positive or
   negative correlation//
   for $l = 0$ to $n/2$ do
      Calculate the Pearson
      correlation coefficient $R(l)$
      using eq. 1 (Def. 2.1)
      If $|R(l)| > |max|$ then
         $max = R(l)$;
         $lag = l$;
      end if
   end for
   if $|max| \geq correlation\_threshold$ then
      return $(X_i, X_j)$, max, lag
   end if
end for
```

For k time series of length n, the time complexity of the HSLC algorithm for the transformation of the initial time series into the Boolean ones is *O(kn)*. The corresponding time complexity for the calculation of the correlation on the Boolean series is *O(1)* as it only involves calculations on bits. So, the total time of detecting the correlated pairs is *O(kn)*. Finally, the time complexity for the calculation of the correlation on the pairs which belong to the set *C* using the Pearson correlation coefficient is *O(ck)* where *c* is the number of the correlated pairs which is very small. Thus, the aggregate time complexity of the algorithm is *O(kn)* in contrast to the time complexity of $O(k^2n)$ of the naive PLC method.

In the experiments the above algorithm was tested on EEG data from 22 different channels with a variety of sensors types and a frequency of 256 Hz. An example of features of an input series is as follows: {*Sensor id*: EEG1, *Frequency*: 256, *Value*: -1.8, *Sensor Type*: Fp2AV, *Time Stamp*: 15/08/2016 17:30:55}.

3 SVM INCREMENTAL MODEL

3.1 Support Vector Machines and Incremental Learning for Stock Prediction

SVMs are among the best classifiers available today. They consider data items as points in a n-dimensional space (where n is the number of features) and perform classification by finding the hyper-plane that best differentiates the classes that are present. In this paper the Gaussian function is used (due to its speed), expressed in the form of

$$k(x,z) = e^{-\frac{\|x-z\|^2}{2\cdot scale^2}} \tag{10}$$

where $x \in R^{(m\times 12)}$, are the support vectors (margin, error and reserve), $z \in R^{(1\times 12)}$ is the input vector, *scale* is the support vectors' size and *m* is the support vectors' number.

Traditional data mining algorithms calculate their results for a pre-defined set of data. However, in the vast majority of applications, the set of data is not known from the beginning. Thus, it is vital for the algorithms to be adapted to the new upcoming data. This means that algorithms use the knowledge that had developed from their previous implementations and so are able to increase the precision in their results. This process is referred as incremental learning. Such approaches that have looked at the above problem through SVMs, have been employed by Poggio et al. (2001) and Syed et al. (1999). In our work, we adopt the case that is studied Poggio et al. (2001).

The proposed model for stock trend prediction involves as a second step the extraction of features. These features should be able to be separated by the classification algorithm into two classes, the upwards and the downwards trend of the index respectively. These features are technical indicators, such as the *William & R, A/D Oscillator,* which are used for the creation of the SVM model that predicts the direction of the KOSPI and they have also been used by a variety of similar papers (e.g., (Shin, 2005)). In this paper the twelve indicators case was adopted and implemented following the approach described by Kim (2003).

3.2 SVM Incremental Model

The streams in the aforementioned implemented model are the financial indicators previously mentioned. The constructed model is able not only to be adapted to the upcoming data but also maintain its existing knowledge through the adaption of the hyperplane of separation. More specifically, the incremental SVM learning algorithm (no instance memory) is used. Subsequently, our model classifies the test set into two classes, the upward and the downward trend of the indicators. Concurrently, this test set becomes part of the current training set, thus an increase of model's knowledge is achieved. Finally, the model is used to predict the trend (upward or downward) that the stock is going to have in the next days.

This model consists of the following parts and it constantly evolves as there are new input data, therefore the knowledge increases providing updated predictions:

1. INPUT: It receives the data streams that consist of *Date, Open, High, Low, Close*, and *Volume* for each stock.
2. FEATURE EXTRACTION: The system calculates the twelve technical indicators (Kim, 2003) along with an additional feature, the *trend* which has the value (+1, *if Close(t)<Close(t+1)*, upward trend), for time *t*, otherwise it has the value (-1, downward trend).
3. MODEL INCREMENT: It receives the above indicators through overlapping windows of fixed size and is adapted to these in order to be able to increase its knowledge. It uses the incremental SVM learning algorithm (Diehl, 2003). Every time that there is an increase in the knowledge through the algorithm, a new model is derived as an output. Thus, for time *t*, we have the model *T(i)*, where *i* is the set of samples which has been already processed by the system, and for a new input window, we have the model *T(i+w),* where *w* is the predefined length of the input. The number of margin vectors (which are constantly increased as the system increases its knowledge), the number of reserve vectors (which are increased at a slower rate than the margin vectors) and the number of error vectors (which are only a few, during the whole implementation) are obtained as an output.
4. PREDICTION: After the adaptation, the model predicts the trend of the stocks for the next days and finds the class that they belong to.
5. OUTPUT: It receives the above results and presents the trend prediction (upward or downward).

An example of features from an input stream as retrieved from the Google/Finance (2016) database is: {*Date*: 12-Dec-14, *Open*: 1921.6, *High*: 1926.6, *Low*: 1915, *Close*: 1921.7, *Volume*: 36347000}. This file had 2965 rows, where each row represents a different day. To create and handle the streams we used the Microsoft StreamInsight Platform (2016) and to implement our functions we used Matlab. An input adapter in Microsoft StreamInsight was generated, which reads the input file (.csv format) and matches each row to a sample stream, whereas each column is an input feature with the field *Date* to be the key for the creation of the streams. In order to create the model of incremental learning we run the incremental SVM learning algorithm proposed by Diehl et al. (2003) through the library Diehl (2011) which is implemented in Matlab.

4 EXPERIMENTS

The experimental procedure includes two parts. In the first part, the lag correlation discovery on time series is considered, applying the HSLC algorithm on EEG data. In the second part, experimentation with data stream classification is conducted, by implementing the SVM incremental learning algorithm and testing it on the KOSPI data set.

4.1 Time series Lag Correlation Discovery

The algorithms HSLC, PLC and BLC were implemented in Matlab and compared. The experiments are separated into two parts. In the first part we split the time series in windows and in the second part the lag correlation on them was calculated in their whole length. The precision of an algorithm Y in comparison to the theoretical correct results of an algorithm X is given by the formula:

$$precision(i)=\begin{cases} \frac{lag^i X}{lag^i Y}, lag^i X \le lag^i Y, \text{for the TP pairs} \\ \frac{lag^i Y}{lag^i X}, lag^i X > lag^i Y, \text{for the TP pairs} \\ 0, \textit{for the TP or FN pairs} \end{cases} \quad (11)$$

where *TP* (True Positives) are the correlated pairs that were detected correctly by the algorithm *Y* compared to the ones that the algorithm *X* had detected, *FP* (False Positives) are the pairs that were detected as correlated by the algorithm *Y* but not by *X*, *FN* (False Negatives) are the pairs that were correlated according to *X*, but they were not detected by *Y* and lag^i is the output of the algorithms for the pair *i*, *i=1,2,…,N*, where *N* is the number of all pairs that we examined, except for the *TN* (True Negatives). The final value of the variable precision is the mean of the vector *precision(i)*.

4.1.1 Lag Correlation with the Use of Windows

The input time series, where each of them has 2000 values, are split in windows. In particular, we use a 128-element sliding window in each EEG channel. Thus, the lag correlation on the subsets of the time series is calculated for every slide of the window. The results of the experiments follow below:

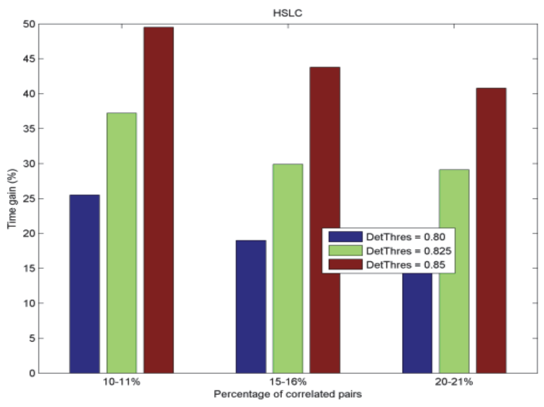

Figure 1: HSLC algorithm. Time gain in detection threshold and in percentage of the correlated pairs of the time series (Number of time series =8).

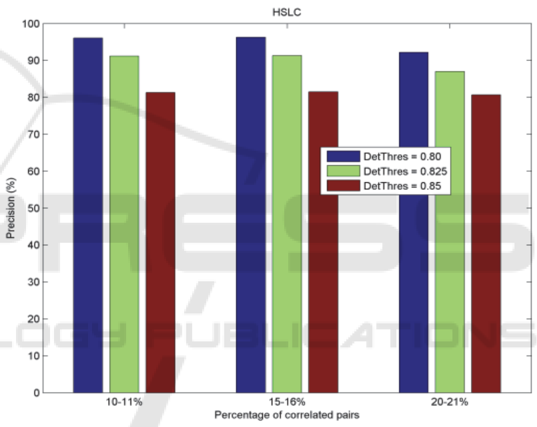

Figure 2: HSLC algorithm. Precision in detection threshold and in percentage of the correlated pairs of the time series. (Number of time series =8).

Figures 1 and 2 show that the time gain ranges from 18.3% to 49.5% and the precision from 83.2% to 95.3%. Moreover, when the detection threshold is increased, the time gain is also increased but the precision is decreased. This happens due to the fact that, when there is a higher detection threshold the algorithm detects fewer correlated pairs and needs less time for the calculation of the correlation coefficients. Thus, there is a trade-off on time gain and precision. When the percentage of the correlated pairs is increased, the time gain is decreased and the precision remains the same. This is because when we have a higher percentage of correlated pairs the detection technique has a smaller time gain.

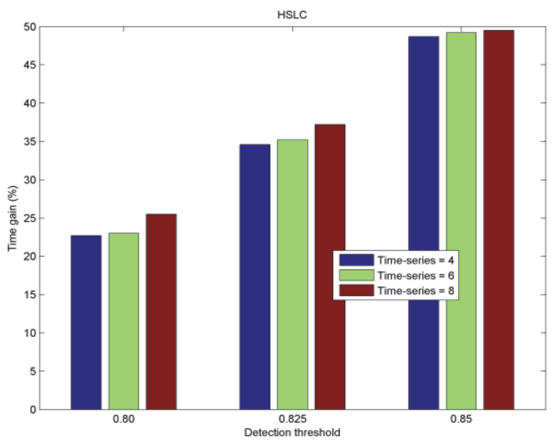

Figure 3: HSLC algorithm. Time gain in number of the time series and in detection threshold. (Percentage of correlated pairs = 10%-11%).

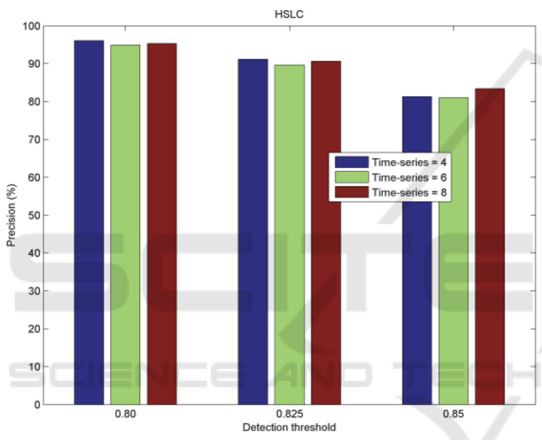

Figure 4: HSLC algorithm. Precision in number of the time series and in detection threshold. (Percentage of correlated pairs =10%-11%).

Figures 3 and 4 show that when the number of time series increases, there is a slight increase in time gain while the precision remains nearly the same. This is because for a greater number of time series there are more possible correlated pairs and the detection technique leads to a larger time gain.

4.1.2 Lag Correlation without Sliding Windows

Here we calculate the lag correlation on time series in their whole length. We also implemented the BLC algorithm which is the detection part of the HSLC as it has a very high time gain in the case that there are no sliding windows.

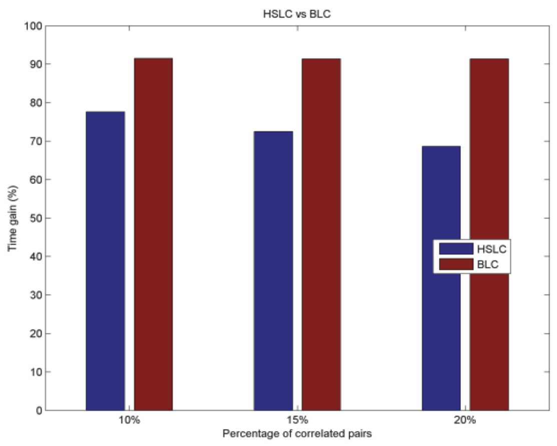

Figure 5: HSLC & BLC algorithms. Time gain in percentage of correlated pairs. (Number of time series = 25).

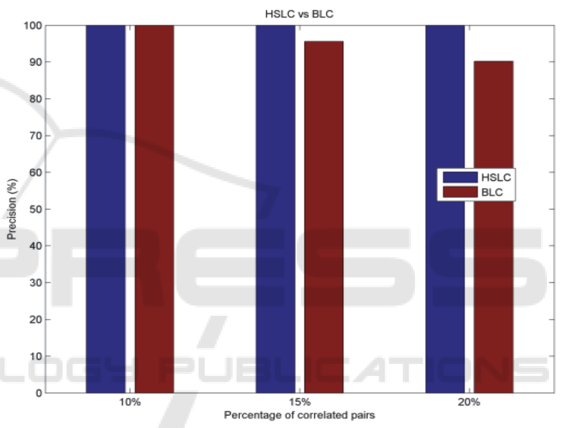

Figure 6: HSLC & BLC algorithms. Precision in percentage of correlated pairs. (Number of time series = 25).

Figures 5 and 6 show that the time gain for the HSLC and BLC is in the range of 68.6%-77.6% and 91.4%-91.5% respectively. The precision for the HSLC is 100% for all cases and for BLC it ranges from 90.2%-100%. Subsequently, the HSLC has a smaller time gain than BLC, but it has an infinitesimal error in precision in contrast to BLC. In HSLC, when the percentage of correlated pairs is increased, the time gain in decreased and the precision remains the same. In contrast, in BLC, when the percentage of correlated pairs is increased, the time gain remains nearly the same and the precision is decreased. Thus, the algorithms have an opposite behavior in time gain and precision.

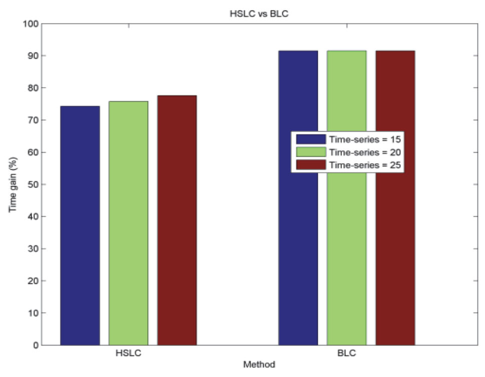

Figure 7: HSLC & BLC algorithms. Time gain in number of time series. (Percentage of correlated pairs = 10%).

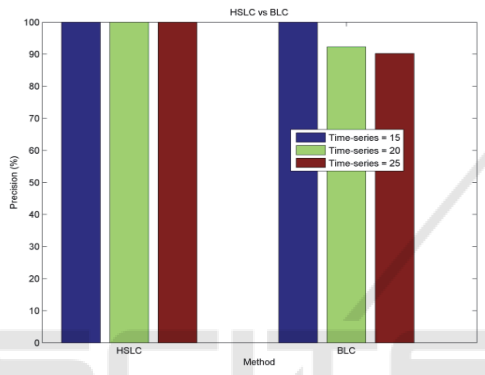

Figure 8: HSLC & BLC algorithms. Precision in number of time series. (Percentage of correlated pairs =10%).

Figures 7 and 8 show that in HSLC when the number of time series is increased there is a slight increase in time gain while the precision remains the same. In contrast, in BLC, when the number of time series increases the time gain remains nearly the same and the precision decreases. Figures 9 and 10 show the general performance of the algorithms. In case that we use windows, the HSLC has a considerable reduction in computational time in comparison to PLC. The degree of the reduction depends on the value that the user provides for the detection threshold by taking into account the tradeoff on time and precision. When no windows are used HSLC has a dramatic reduction in computational time compared to PLC regardless of the value of the detection threshold (0.825 in this experiment). Thus, in this case we do not have to take into account a tradeoff as we achieve the maximum precision (100%) and a very high time gain with the above value of the detection threshold.

In the implementation without windows, BLC could also be used as it achieves a smaller computational time than HSLC. However, it does

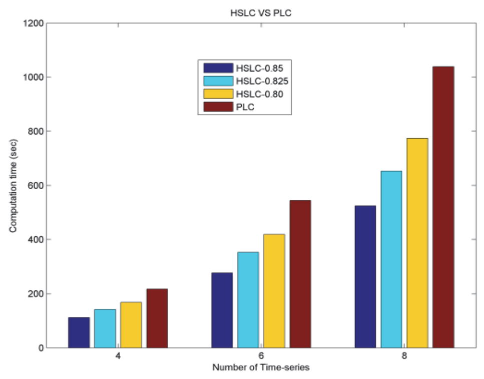

Figure 9: With sliding windows. Comparison of the computational time of HSLC & PLC algorithms in number of time series. (Percentage of correlated pairs =10%-11%).

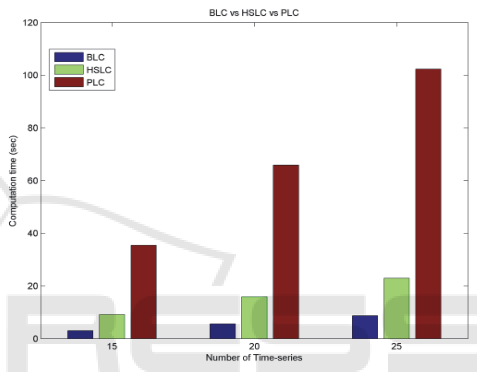

Figure 10: Without sliding windows. Computational time of BLC, HSLC & PLC for different number of time series. (Percentage of correlated pairs = 10%).

not ensure an infinitesimal error as we noticed above.

4.2 Data Stream Classification

In the classification experiments we compare our model with three other versions of the SVM algorithm. In our approach, we train the incremental SVM learning algorithm with the first 3000 examples then increase its knowledge with feedback from the next 3000 examples. Thus, the test set becomes part of the training set. We use a window that has a size of 10 examples. The second approach is the classic SVM which is trained with the same first set of 3000 examples and then tested with the next 3000 examples. The third version is the static incremental SVM algorithm which uses the same training and test sets of 3000 samples each. The difference in the latter case is the use of the SVM incremental learning algorithm instead of the classic SVM. However, it is only trained once, in contrast to our approach. The fourth version is the online SVM algorithm which is trained with the same first 3000 examples and then increases the initial training set with the following 3000 samples. The difference between this and our method is that this model is re-trained every time a new instance arrives (full instance memory). Table 1 shows the average accuracy the methods achieved in predicting the price of stock for 3000 examples. The incremental SVM algorithm achieves the highest accuracy which is increased with the increase of the training set. It achieves 71% accuracy for the first half of the training set and 77% for the second half.

Table 1: Accuracy of various SVM algorithms.

ALGORITHM	ACCURACY
Classic SVM	52%
Static Incremental SVM	60%
Online SVM	62%
Incremental SVM Learning	**74%**

5 CONCLUSION

In this paper we proposed two different data miming techniques for time series and data streams. The first is associated with the problem of lag correlation discovery of time series. The proposed HSLC algorithm achieved a reduction in time complexity in contrast to the state-of-the-art method PLC while preserving high accuracy in the results. Furthermore, it has an infinitesimal error for both the value of the lag and the value of the correlation coefficient for every detected correlated pair of series. In the second part of this paper, we examined the problem of classification of data streams and evaluated several approaches on a stock prediction case. Specifically, an incremental SVM learning algorithm used for data mining on streams was employed on the KOSPI dataset. The algorithm achieved higher accuracy compared to three other versions of the SVM algorithm concluding that the training models for the stock prediction problem should follow an incremental iteration methodology.

ACKNOWLEDGEMENTS

This work was supported in part by the European Union’s Horizon 2020 research and innovation programme under grant agreement No 690140.

REFERENCES

Diehl, C. P. and Cauwenberghs, C. (2003) SVM Incremental Learning, Adaption and Optimization. In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 2685-2690.

Edwards, R.D., Magee, J. and Bassetti, W.H.C. (2007) *Technical Analysis of Stock Trends*, 9th ed.

Kim, Kyoung-jae (2003) Financial time series forecasting using support vector machines. *Neurocomputing Journal*, 55(1-2), pp. 307-319.

Mporas I., Tsirka, V., Zacharaki, E. I., Koutroumanidis, M., Richardson, M., Megalooikonomou, V. (2015) Seizure detection using EEG and ECG signals for computer-based monitoring, analysis and management of epileptic patients. *Expert Systems with Applications*, 42(6), pp. 3227-3233.

Poggio, T. and Cauwenberghs, C. (2001) Incremental and Decremental Support Vector Machine Learning. In: *Proceedings of the 2000 Conference on Advances in Neural Information Processing Systems*, Vol. 13, MIT Press.

Shin, K.S., Lee, T.S. and Kim, H. (2005) An application of support vector machines in bankruptcy model. *Expert Systems with Applications*, 28(1), pp. 127-135.

Syed, N. A., Liu, H. and Sung, K. (1999) Incremental Learning with Support Vector Machines. In: *Proceedings of the International Joint Conference on Artificial Intelligence*.

Yi, B.-K., Sidiropoulos, N.D., Johnson, T., Jagadish, H.V., Faloutsos, C., Biliris, A. (2000) Online Data Mining for Co-Evolving Time Sequences. In *Proceedings of the 16th International Conference on Data Engineering*, pp. 13-22.

Zhang, T., Yue, D., Gu, Y., Wang, Y. and Yu, G. (2009) Adaptive correlation analysis in stream time series with sliding windows. *Computers & Mathematics with Applications*, 57(6), pp. 937-948.

Zhang, T., Yue, D., Wang, Y. and Yu, G. (2011) A Novel Approach for Mining Multiple Data Streams based on Lag Correlation. In: *Proceedings of the Control and Decision Conference*, pp. 2377-2382.

Zhu, Y. and Shasha, D. (2002) StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time: In: *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 358-369.

Google/Finance. (2014). [Online] Available from: *http://www.google.com/finance.* [Accessed 14/3/17].

Microsoft StreamInsight. (2016). [Software] [Online]Available from: https://technet.microsoft.com/enus/library/ee362541.

Diehl, C. (2011). Github, inc. [Online] Available from: *https://github.com/diehl/Incremental-SVM-Learning-in-MATLAB.*