Suárez-Cetrulo, A.L., Cervantes, A. (2017). An online classification algorithm for large scale data streams: iGNGSVM. *Neurocomputing*, 262, pp. 67-76.

DOI: https://doi.org/10.1016/j.neucom.2016.12.093

An Online Classification Algorithm for Large Scale Data Streams: iGNGSVM

Andrés L Suárez-Cetrulo[a,1,*], Alejandro Cervantes[a,1], David Quintana[a,1]

[a] *Computer Science Department, University Carlos III, Avda. Universidad, 30, 28911, Madrid, Spain*

Abstract

Stream Processing has recently become one of the current commercial trends to face huge amounts of data. However, normally these techniques need specific infrastructures and high resources in terms of memory and computing nodes. This paper shows how mini-batch techniques and topology extraction methods can help making gigabytes of data to be manageable for just one server using computationally costly Machine Learning techniques as Support Vector Machines. The algorithm iGNGSVM is proposed to improve the performance of Support Vector Machines in datasets where the data is continuously arriving. It is benchmarked against a mini-batch version of LibSVM, achieving good accuracy rates and performing faster than this.

Keywords: Data Classification, Topology Extraction, Online Learning, Large Datasets, Growing Neural Gas, Support Vector Machines.

1. Introduction

The daily growth of data in Internet exceeds storage and processing limits for analytical purposes. For example, in 2012 IBM indicated that there was generated a daily average of 2.5 exabytes [1]. This big amount of information, with a 40% of average annual increase, has become the front door of massive data streams that help to give rise to the widespread trend known as "Big Data". According to Doug Laney [2], a given amount of data is considered this when characterized by the dimensions Volume, Velocity and Variety. The data processing and understanding necessities here rely on the improvement of decision-making processes inside companies and organizations. This large-scale data domain, where the term "analytics" is being used to define data analysis and data mining activities [3], is a landscape for Machine Learning techniques. These last play an important role detecting and identifying data patterns and

*Corresponding author

Email addresses: suarezcetrulo@gmail.com (Andrés L Suárez-Cetrulo), acervant@inf.uc3m.es (Alejandro Cervantes), dquintan@inf.uc3m.es (David Quintana)

relationships when either the data analyst or data scientist are unable to inspect the whole amount of information. In the meantime, the biggest explosion of the Internet of Things is still coming [4].

Nowadays, distributed computing techniques or the programming model MapReduce [5] are one of the current trends. One of the inconveniences of these normally batch techniques is the inability to efficiently deal with data updates as well as its evolution, in either stationary or non-stationary domains where a hidden context [6] may influence the predictive model behaviour over-time in unforeseen ways. It is in this regard where the term Concept Drift is used in this paper [7]. Moreover, a large portion of the data that is generated in real-world applications can be obtained incrementally through continuous streams [8].

Recently, Stream Processing [9][10][11] has appeared inside the Big Data trend as a fashion to face continuous streams such as logs, chat conversations, data scraped from the web, etc. However, all these techniques need high resources in terms of memory and computing nodes. Therefore, less powerful devices are still unable to run complex analysis over large data sets without external clusters.

Regarding to Machine Learning classification techniques, Support Vector Machines (SVMs) [12] [13] have proof their performance in the literature dealing with non-linear and complex datasets. The SVM training algorithm creates an hyperplane (or a set of them) in a high dimensional space [14] at the largest distance to the nearest prototypes of any class. This is built in terms of a relatively small number of input examples (Support Vectors), which are the nearest data-points of each class to this hyperplane defining each class boundaries.

However, the SVMs classifier piggybacks a high computational complexity [15] on Large-Scale classification. Its use for Large-Scale problems in the literature has required to be in conjunction with other techniques as Incremental Learning techniques [16], Prototype Reduction, or Prototype Generation techniques as Learning Vector Quantization [17] [18] [19] [20] [21], in order to reduce the amount of data in training stage. Ajalmar et al. [18] provide a good overview on this last matter.

In terms of incremental approaches that have been previously applyed to SVMs, N.A. Syed et al. [22] propose one algorithm that deals with the SVs of previous iterations similar to this work. Nevertheless, that algorithm needs to be re-trained at every iteration to pick which Support Vectors should be considered at the next iteration. Cauwenberghs et al. [23] propose another alternative for incremental learning and decremental unlearning using SVMs that has not been widely accepted in the literature [24]. Stefan [25] proposed a weighted model where the oldest SVs are more costly, facing then changes of the feature space across time. Laskov et al. [26] proposed another online algorithm based on Support Vectors. However it suffered from a high computational cost.

Another research area of interest for classification using online data streams are Evolutive Intelligent Systems [27] [28], or more recently Evolutive Fuzzy Systems [29] [30] [31] [32]. These online and incremental systems are able to adapt themselves to context changes on-the-fly through adaptive fuzzy-rules.

According to P.Angelov [33], they collect training data continuously, and some of the new data may reinforce the model but other data would bring new information, involving changes in the learning model while saving the price of re-training the algorithm. This requires less computational power since every iteration training is just performed over a small snapshot of training data (or ideally over only the new data).

The algorithm introduced in this work uses Growing Neural Gas (GNG) [34] as Prototype Generation technique to reduce the dataset before classifying with SVMs. GNG has been proven to reduce massively the number of instances of a dataset preserving the original topology [34]. It has already been used in conjunction with a standard SVM solver [19] as the work proposed here. We use SVMs [15] instead of Incremental SVMs due to the above-mentioned inefficiencies in incremental approaches. The proposed algorithm collects data continuously in a buffer until it reaches a mini-batch size. However, batches are not independent tasks each other. The learned model is inherited between mini-batches. As a consequence, the algorithm does not need to be re-trained with previous data.

This paper is focused in the above-mentioned area, proposing an instance-based online mini-batch learning algorithm for data classification, with aim to improve the performance of SVMs in datasets where the data is continuously arriving. Thus, the work is organized as follows. Section 2 contains the proposed algorithm and a brief summary of the conducted work; section 3 describes the experimental study, and section 4 the future work and the conclusions.

2. Proposed algorithm

2.1. Definition of the iGNGSVM algorithm and variations

The algorithm proposed in this work, iGNGSVM, inherits features from two previous algorithms:

- In GNG-SVM [19], the authors propose using Growing Neural Gas (GNG [34]) before classification to replace a potentially large number of input patterns with a set of prototypes that reproduces the patterns' topology. GNG is applied independently to patterns of different classes, and then SVM classification is applied to the combined results. For each class, GNG builds a networked cloud of neurons that grows until covering the pattern distribution. The stopping criterion is the desired size for the resulting topology.

- In OISVM [17], the authors also inserted a prototype generation procedure before performing classification. In this work prototypes are created simultaneously for all classes. Upon initial testing it was noticed that OISVM creates unnecessary Support Vectors during the iterations when the boundaries of previous iterations overlap, in the feature space, with data from the current iteration.

Our proposal, the iGNGSVM algorithm, combines both approaches. Its objective is large-Scale data classification on continuous data streams (online learning), even on situations that need a classification model to be updated over time (learning from non-stationary data). It applies the Prototype Generation technique previously used in GNG-SVM, that summarizes a dataset before classification, to the online mini-batch structure for Support Vectors Machines defined in OISVM.

Therefore, iGNGSVM has two main components executed iteratively on the incoming data ordered as a sequence of mini-batches of data. The first component, the prototype generation stage, is performed by GNG and tries to reduce the computational cost.

Algorithm 1 Growing Neural Gas algorithm subtracted from [34]

(1) Start with two units a and b at random positions w_a and w_b in $\mathbf{R}^n$.

(2) Generate an input signal ξ according to $P(\xi)$.

(3) Find the nearest unit s_1 and the second-nearest unit s_2.

(4) Increment the age of all edges emanating from s_1.

(5) Add the squared distance between the input signal and the nearest unit in input space to a local counter variable:

$$\Delta errors_{s1} = \|w_{s1} - \xi\|^2$$

(6) Move s_1 and its direct topological neighbors towards ξ by fractions ϵ_b and ϵ_n, respectively, of the total distance:

$$\Delta w_{s1} = \epsilon_b(\xi - w_{s1})$$
$$\Delta w_n = \epsilon_n(\xi - w_{sn}) \text{ for all direct neighbors } n \text{ of } s_1$$

(7) If s_1 and s_2 are connected by an edge, set the age of this edge to zero. If such an edge does not exist, create it.

(8) Remove edges with an age larger than a_{max}. If this results in points having no emanating edges, remove them as well.

(9) If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows:

- Determine the unit q with the maximum accumulated error.
- Insert a new unit r halfway between q and its neighbor f with the largest error variable:

$$w_r = 0.5(w_q + w_f).$$

- Insert edges connecting the new unit r with units q and f, and remove the original edge between q and f.
- Decrease the error variables of q and f by multiplying them with a constant α. Initialize the error variable of r with the new value of the error variable of q.

(10) Decrease all error variables by multiplying them with a constant d.

(11) If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled go to step 1.

The algorithm for Growing Neural Gas, as described in [34] can be seen in Algorithm 1. At step 11, the stopping criterion determines how many prototypes does GNG generate using an specific topology. In our algorithm, this first component receives a data chunk (and also Support Vectors in iGNGSVM$_2$). Thus, the stopping criterion is not just a key factor on the reduction performed over the size of the final training set but also has a role in the time consumed in training stage. A larger stopping criterion imply more time consumed by GNG; therefore, balancing of this parameter to be relatively small without penalizing hugely the classification results had to be performed.
In our experiments every data chunk involve hundreds of thousands of data examples. As a consequence, the GNG stopping criterion is a number lower than 0.1% the size of the received chunk. The size of the data-chunks in iGNGSVM is an static value, being always the same during an experiment. This is also used as starting criterion in the buffer, that collects examples continuously.

The second component, the classification stage, is performed by Support Vector Machines classifiers. The behaviour of Support Vector Machines, which can be widely found in the relevant literature [12] [15], can be seen in Algorithm 2 for a linear model in a simple approach of binary classification. In our work, the SVM classifier trains prototypes since they are created by GNG. Therefore any mention to data points or vectors in Algorithm 2 refers to prototypes in iGNGSVM. A SVM attempts to find the hyperplane that separates the data examples of distinct classes at maximum distance between them.

Algorithm 2 Linear model for Support Vector Machines according to [12] [35]

(1) In a binary classification problem, we have $Y = A, B$ where Y is the set of both possible classes (class A and class B).
(2) A set of vectors $(x_1, y_1), \ldots, (x_n, y_n)$ where $xi \in R_d$ and $y_i \in A, B$ for $i = 1, \ldots, n$ is separable if there is an hyperplane in R_d able to separate the vectors $X = x_1, \ldots, x_n$ with class $y_i = A$ from those with class $y_i = B$.
(3) Given a separable set, there will be at least an hyperplane $\Pi : w \cdot x + b = 0$ able to separate the vectors $X = x_1, \ldots, x_n$.
(4) Once the separator is determined, it is adjusted to more vectors according to Equations Π_1 and Π_2 below.

$$x_i \cdot w + b \geqslant +1 \text{ for } y_i = A \text{ (Equation } \Pi_1)$$
$$x_i \cdot w + b \leq -1 \text{ for } y_i = B \text{ (Equation } \Pi_2)$$

$+1$ and -1 are labels A and B respectively. w is a normal to the hyperplane.

(5) The points for which the equality of equations Π_1 and Π_2 hold are called *support vectors* of its respective class.
(6) Π_1 and Π_2 are parallel as they have the same normal (w) and there is no data examples between them. Thus the pair of hyperplanes that maximizes the margin between classes can be obtained by minimizing $\|w\|$.
(7) The algorithm is formalised as: $minimize\ \|w\|\ s.t.\ y_i(w \cdot xi + b) - 1 \geqslant 0, \forall x_i$

Then, the data points that "support" this hyperplane on both sides are called "support vectors". These are the set of nearest vectors to the separator. In iGNGSVM, as mini-batches are not independent classification tasks, the model trained in the second component is then re-used in the next iteration.

Consequently, the Support Vectors are inherited at the next iteration (once received the next data chunk) helping to keep the previous classification knowledge in the model and incorporating at the same time prototypes of the new data chunk on training stage (see figure 1).

An important issue that prevented the application of the iGNGSVM predecessors to large datasets was the problem of model bloating (unbounded growth of the SV generated on each iteration of the algorithm), which introduces a third component described in Algorithm 3. iGNGSVM deals with excess Support Vectors using a technique of Prototype Reduction, Wilson's Edited Nearest Neighbor (ENN) [36] [37]. The number of neighbors calculated at step 1 is a parameter of the algorithm that in our experiments has normally been set at 3 or 5.

Algorithm 3 ENN algorithm according to [37]

(1) Compute the k Nearest Neighbors for every data example based on the provided input network topology.
(2) For every data example, if most of its neighbors belong to a different class this is labeled as a noise.
(3) Once finished the neighbourhood comparisons for all the data examples, it deletes all the data examples labeled as noise.

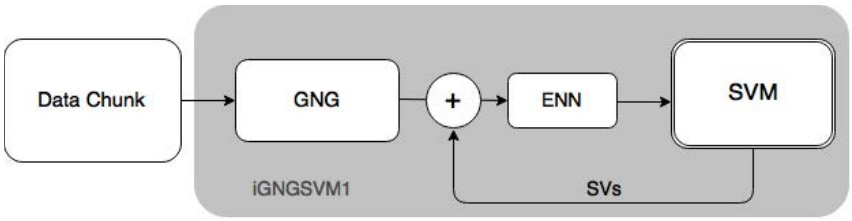
In this work, ENN deletes prototypes or Support Vectors that are surrounded by a majority of examples of another class. This helps to clean noisy prototypes, or SVs that are not longer relevant in the model. This is crucial here as these SVs or prototypes act as noise (see section 3.4) making the amount of SVs grow each iteration and also making the model unscalable for large numbers of iterations or batches.

As summary, iGNGSVM is a mini-batch algorithm for online machine learning classification that reduces the size of the batch received for training every iteration using Growing Neural Gas. A buffer collects incoming data examples continuously. Training starts once achieved the starting criterion (mini-batch size). In the meantime, any data example received for testing is evaluated with the latest trained model. Different mini-batches are not independent tasks each other. The learned model (Support Vectors) is inherited between iterations. iGNGSVM also deletes noisy prototypes through ENN to avoid an unnecessary increase in amount of SVs. Finally, the model is updated using a new SVM classifier every iteration. A first version of iGNGSVM$_1$ is described in pseudo-code at Algorithm 4.

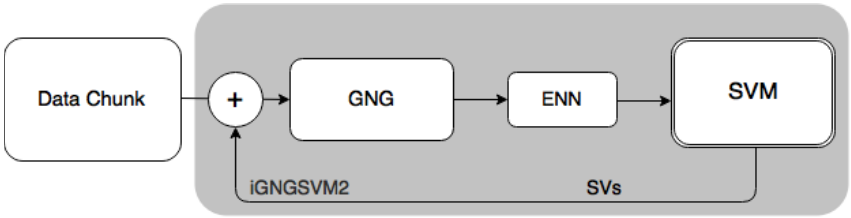
Algorithm 4 iGNGSVM$_1$ training algorithm.

(1) It receives a set of training examples S (data-chunk in figure 1).
(2) It divides the set of examples in one subset per class ($S_1, ..., S_n$ / n classes).
(3) It obtains the topology of every isolated subset using GNG with an stopping criterion that can be a number of prototypes to create or a percentage according to every subset size.
(4) It merges all the subsets, putting the topology of all the classes together ($S*$).
(5) It runs ENN removing prototypes from $S*$ that are surrounded by a majority of neighbors of the opposite class.
(6) If *iteration number* > 1: It adds the SVs of the previous iteration as prototypes to $S*$ to remember the prior model.
(7) It trains a new SVM using the new data chunk topology $S*$.
(8) It saves the SVs calculated for the next iteration. Then it waits for the starting condition to process the next data-chunk (step 1)

As previously mentioned, the purpose of iGNGSVM is to perform classification over large-scale streams (which properties may change over time), avoiding SVMs' high performance bottlenecks through the use of GNG and ENN. Data is continuously received being processed in data-chunks that can either have a fixed or a variable size that is defined by an external parameter (for instance, to process a data-chunk at specific time steps).



(a)



(b)

Figure 1: Comparison between the behavior of the proposed structures for iGNGSVM. iGNGSVM$_1$ above and iGNGSVM$_2$ below. Each data-chunk represents the training examples received every mini-batch.

A first version of iGNGSVM, was developed using MOA[1], improving the execution time against LibSVM[2] and generally performing better than OISVM for large datasets. That study[3] showed how OISVM and iGNGSVM perform better than LibSVM with large datasets that change over-time such as SEA Concepts [38]. However, there was observed how the iGNGSVM performance was not as good in non-stationary datasets with rapid changes where a purely incremental algorithm behaved generally better.

In order to address this problem, a second structure for iGNGSVM is defined in this work. Figure 1 shows both proposed versions for iGNGSVM. iGNGSVM$_1$ (figure 1a) receives the SVs of the prior iteration after it reduces the number of examples to prototypes with GNG. iGNGSVM$_2$ (figure 1b) does it before it uses GNG. Thus, iGNGSVM$_2$ applies Prototype Reduction also to SVs, reducing them also as part of the same topology. As a consequence, previous iterations have less weight in iGNGSVM$_2$ than in iGNGSVM$_1$. ENN is still used in this model as some SVs no longer relevant could generate noisy prototypes.

In this work, the starting condition (see Algorithms 4 and 5) is once the buffer reaches the selected data-chunk (mini-batch) size. Nevertheless, if required, this starting criterion could also be a change of a feature in the data such as a date. e.g. in real world applications where two different hours or days may have concept drift variations between them, a new hour/day could be used as starting criterion (note that this is different than the use of time-stamps as dataset features).

Algorithm 5 iGNGSVM$_2$ training algorithm.

(1) It receives a set of training examples S (data chunk in figure 1).
(2) If *iteration number* > 1: It adds the SVs of the previous iteration as examples to S.
(3) It divides the set of examples in one subset per class ($S_1, ..., S_n$ / n classes).
(4) It obtains the topology of every isolated subset using GNG with an stopping criterion that can be a number of prototypes to create or a percentage according to every subset size.
(5) It merges all the subsets, putting the topology of all the classes together ($S*$).
(6) It runs ENN removing prototypes from $S*$ that are surrounded by a majority of neighbors of the opposite class.
(7) It trains a new SVM using the new data chunk topology $S*$.
(8) It saves the SVs calculated for the next iteration. Then it waits for the starting condition to process the next data-chunk (step 1)

[1]MOA is an open source framework for data stream mining related to the WEKA project: http://moa.cms.waikato.ac.nz

[2]LIBSVM is a Library for Support Vector Machines: http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

[3]First comparative study of iGNGSVM: http://hdl.handle.net/10016/19258

2.2. Computational Complexity

As iGNGSVM is made of different components, its computational complexity relies on the used algorithms for the SVM classifier, GNG and ENN.

- The library for SVMs used in this work is LibSVM. As stated in [15] and [39], its Big-O notation complexity is $O(n_{features} \times n^2_{samples})$ assuming $O(n)$ for each kernel evaluation.
- The complexity of both the basic GNG implementation and ENN is $O(n^2)$ according to [40] and [41]. They require iterations over the neighbourhood of each data example provided.

Table 1 shows an estimate of the complexity for every step in the iGNGSVM$_1$ algorithm. The complexity is explained in Big O notation; a loop that implies iteration over every data example has complexity $O(n)$. Otherwise, when the iteration is only over the number of classes (see step 4 in Algorithm 4), it has complexity $O(k)$.
Nevertheless, the required computing power of every step varies according to the size of the input data-set. Table 1 also shows the input size at every step as reference. The chunk-size |TS| is in all the cases the larger value. It represents the mini-batch size provided as starting criterion.

Table 1: iGNGSVM$_1$ complexity analysis

Repeat the next for every mini-batch	Complexity	Input
(1) Divide the data-chunk in one subset per class.	$O(n)$	|TS|
(2) Obtain the topology of every subset using GNG by separate.	$O(k \times n^2)$	$\sum(\|S_{class_i}\|)$
(3) Merge all the subsets in a prototypes final set.	$O(k)$	$\#classes$
(4) Run ENN to label and remove noise.	$O(n^2)$	$\|S*\|$
(5) Inherit the SVs of the previous iteration.	$O(n)$	SVs
(6) Train a new SVM using the new data chunk topology.	$O(n^3)$	$\|S*\| + SVs$
(7) Save the new SVs for the next iteration.	$O(log(n))$	SVs

Step 2 is repeated for every class of the dataset by separate (note that GNG runs for each sub-topology at step 3 in Algorithm 4), which implies a cost of $O(k \times n^2)$ as there are k classes. Its global cost depends on the the summation of the number of examples for each class in the given mini-batch, its parameter λ (see algorithm 1) and also on the stopping criterion. Both the starting criterion for iGNGSVM and stopping criterion for GNG have been optimized to create representative sub-topologies that help iGNGSVM to run faster than a simple SVM classifier. Step 4 receives the final topology created using GNG for the given data-chunk. The number of prototypes as output of GNG is shown in formula 1.

$$|S*| = \#classes \times stoppingCriterion \tag{1}$$

Step 5 introduces the SVs in the prototypes-set $S*$ in sequential order; thus, $O(n)$ is affected by the number of SVs inherited from the previous mini-batch. Step 6 trains the SVMs using only the prototypes-set plus the Support Vectors just added. The complexity of the iGNGSVM algorithm results in $O(n^3)$ per mini-batch processed, as its counterpart LibSVM. However, iGNGSVM trains its SVM with an amount of prototypes of several orders of magnitude lower than a plain SVM classifier, which impacts straightaway in its computational cost. In comparison, LibSVM would need the full chunk-size for training, of a size of several hundred-thousands of examples in our experiments, against a prototypes-set of only hundreds of prototypes in iGNGSVM. Finally, step 7 saves the SVs for the next iteration in $O(log(n))$.

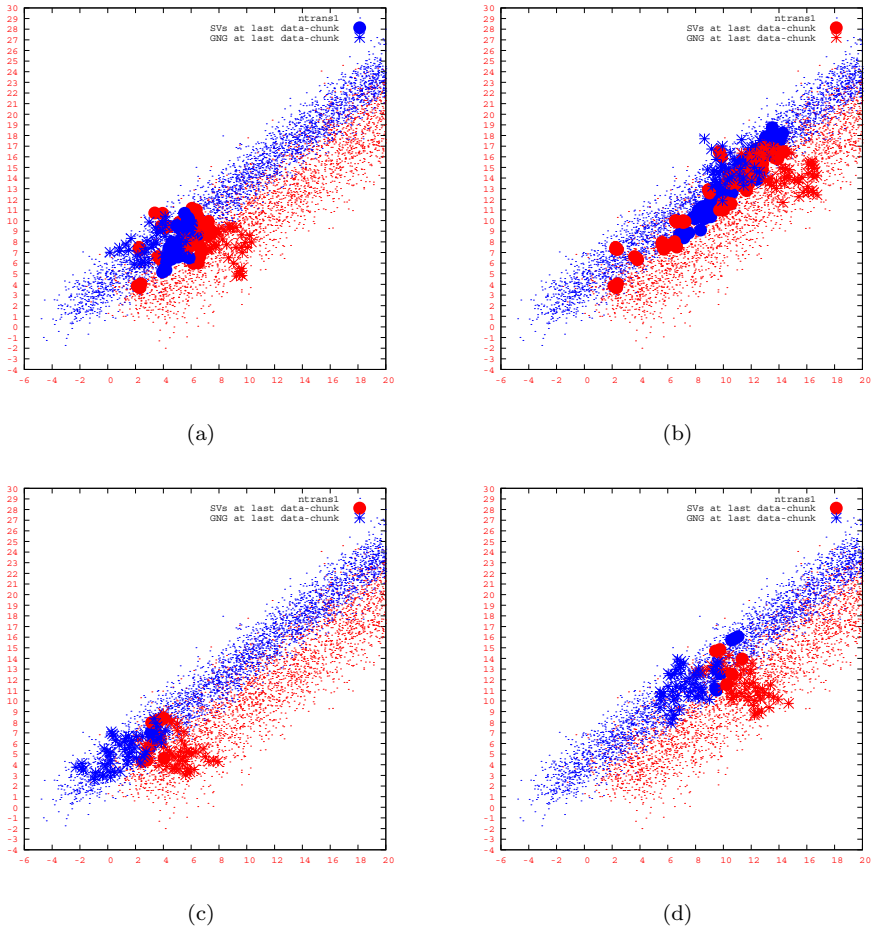
The final Big O notation complexity analysis for iGNGSVM$_2$ is $O(n^3)$ as well as it implies the same amount of steps. However, the order of these is different and ENN runs before GNG. Therefore, training for the SVM classifier will imply only $|S*|$ examples. In section 3.4, we benchmark iGNGSVM$_1$ and iGNGSVM$_2$ against LibSVM sending streams through the MOA framework that are collected as mini-batches by a buffer at training stage.

2.3. Preliminary testing and effect of the structure and other parameters

Both versions of iGNGSVM are compared in Figure 2, where they are used to classify a synthetical non-stationary dataset. The dataset is constructed by sampling two overlapping bi-dimensional distributions of two different classes (blue and red), whose centers are moved at constant speed on the axes X and Y.

In Figure 2, the small dots represent patterns in the dataset, that moves diagonally from the bottom left corner to the upper right corner of the figures. Small dots are used to show the whole dataset; stars are the prototypes generated by the GNG part of the algorithm, in two different time steps for each algorithm. Circles show which are the support vectors retained by the model in that time step after the SVM phase. Comparing Figure 2a with Figure 2c, it is clear that $iGNGSVM_1$ always generates more support vectors than $iGNGSVM_2$, even for the initial data-chunk. Comparing Figure 2b with Figure 2d we show that the final model is quite different in both versions of the algorithm. While $iGNGSVM_1$ exhibits a long-term memory behaviour that retains some SVs from previous iterations, $iGNGSVM_2$ only retains SVs relevant to the current data-chunk.

In Table 2 we show the results in terms of accuracy and number of SV's both for a stationary version of the previous dataset (where distributions do not move) and the former non-stationary dataset. Even though this reduction of SVs had a cost in accuracy rate of about 1% in our tests, both versions of algorithm always performed better using ENN. Also, both the number of SVs and the execution time decreased when changing the number of neighbors. Table 2 provides a good overview of the help provided by ENN which improves the scalability of



(a) (b)

(c) (d)

Figure 2: SVs created by $iGNGSVM_1$(top row, 2a and 2b) and $iGNGSVM_2$ (bottom row, 2c and 2d) while processing a non-stationary dataset with two distributions that move from the bottom left corner (left) to the upper right corner (right).

the algorithm. There was a greater improvement for the non-stationary dataset, it reduced the number of final SVs in up to three thousand, being iGNGSVM$_2$ the one that performed better. Nevertheless, also GNG helped to refresh the model cleaning those SVs that were no longer necessary. It could be noted how ENN by itself is not enough at some cases to handle the accumulation of SVs. iGNGSVM$_1$ was still accumulating some SVs in the non-stationary dataset (see figure 2).

Figure 2 also shows how the algorithm could evolve its model in presence of Concept Drift. This would be managed by the balance between inheriting SVs and the creation of a new SVM every iteration. Hidden context changes inside the same mini-batch are not handled. However context changes between suc-

cessive mini-batches would be discovered by elimination of the invalid Support Vectors. Thus, the chunk-size is one of the main parameters to be balanced in these sort of problems. Nevertheless, although Concept Drift changes may be handled by iGNGSVM, it is worth to mention that the current work is rather focused in the ability to classify large amounts of data overcoming scalability issues by using mini-batch and Prototype reduction techniques.

Table 2: Test using iGNGSVM$_1$ (accuracy$_1$ and SVs_1) and iGNGSVM$_2$ (accuracy$_2$ and SVs_2) for several data chunk sizes (column 1) over the *stationary* (LHS) and *non-stationary* (RHS) synthetical datasets. ENN$_{ne}$ is the number of neighbors (N/A if ENN is disabled).

		*stbal*1				*ntransSingle*1			
|TS|	# ENN$_{ne}$	% accuracy$_1$	SVs_1	% accuracy$_2$	SVs_2	% accuracy$_1$	SVs_1	% accuracy$_2$	SVs_2
100	N/A	91.96	753	91.88	52	85.36	3714	89.72	54
100	3	91.40	268	91.44	28	84.6	745	90.28	12
100	5	91.00	249	90.72	28	83.28	667	91.44	12
200	N/A	91.76	400	91.56	44	84.84	1910	88.44	54
200	3	91.40	96	90.72	20	85.04	372	89.56	10
200	5	91.24	109	90.56	27	84.32	344	89.76	4
300	N/A	91.48	282	91.56	40	84.8	1263	86.04	50
300	3	91.36	101	91.40	25	85.00	214	86.6	3
300	5	91.24	102	91.32	23	84.80	105	88.4	6
500	N/A	92.00	161	91.44	45	84.64	759	85.76	51
500	3	91.56	40	91.12	21	85.6	38	85.76	7
500	5	91.32	36	91.52	17	84.6	4	83.48	4

3. Benchmarking iGNGSVM

This section benchmarks the accuracy rate and execution times of iGNGSVM against two algorithms that obtain a good balance between these two measures.

A mini-bach version of LibSVM, LibSVM*, has been developed to benchmark the execution time of iGNGSVM against LibSVM. The LibSVM batch library was not executable due to the large amount of data to be processed, that did not fit in the server memory for processing. LibSVM* simply splits the training data into batches of 10,000 instances and treats these batches separately. The final accuracy is the average accuracy for all the batches.

In order to conduct this work we have used two datasets, HIGGS [4] and SUSY [5], due to its large volume. HIGGS contains 11,000,000 instances and 28 attributes. SUSY has 5,000,000 instances and 18 attributes. Both were generated through Monte-Carlo simulations and created for binary classification with deep-learning algorithms in [42]. The server used is a Macbook Pro 13" from early 2011 with an i5 processor of 2,3 GHz and 4GB of RAM. The goal of using this laptop to perform Machine Learning tasks with a dataset of many

[4] https://archive.ics.uci.edu/ml/datasets/HIGGS
[5] https://archive.ics.uci.edu/ml/datasets/SUSY

Gigabytes is to show how mini-batch machine learning techniques are able to deal with large datasets, even when they do not fit in main memory, performing tasks which would be barely scalable avoiding the use of distributed computing techniques. Datasets are always presented to our algorithm by the MOA framework as a finite, ordered stream of data.

3.1. Evaluation Mode

All the experiments have been performed evaluating and training the model at the same time[6]. It is done in a sequence of batches, depending on the size of the dataset and the selected data chunk size. The process always starts with testing, therefore the first pass is stochastic (before training) and it may affect to the final accuracy result. Even though this impact is not hugely significant given the amount of batches processed, the first batch has not been taken into account avoiding to interfere in the results. We have rejected evaluation techniques that start training (and then they test) aiming a parallel execution for both training and testing. Therefore, the algorithm is always testing incoming prototypes that, once evaluated, are also sent as data examples to a buffer for training. As it also becomes crucial not to accumulate more than one massive data chunk on memory, thus parallel threading has not been a choice in this work.

Testing has been performed using a sliding window[7] in which every batch is tested with the latest trained model. The test and training data chunks have the same size, defined as an input parameter (chunk size) in the algorithm. All the used datasets are just made of one file (of many Gigabytes of size and millions of training examples) that do not fit on main memory in the server used.

3.2. Parameter Optimization

Growing Neural Gas, Support Vector Machines and ENN present some resilience to be scaled and reaching accurate topologies or classification models, respectively, when using them on online Machine Learning. For example, the fact of setting a high value for the GNG stopping criterion or sending a massive number of non-linearly separable prototypes as an input to SVM might imply a huge increase on training times. Consequently, the following critical parameters have been the ones optimized in order to conduct this research: the **batch size** or **chunk size**, the GNG **stopping criterion** and **the number of neighbors** in ENN. The rest of the input parameters have been set with their default values in its most relevant papers [37] [19]. LibSVM was the library chosen for

[6] $EvaluateInterleavedTestThenTrain$ is an evaluation mode present on MOA: www.cs.waikato.ac.nz/∼abifet/MOA/API/

[7] $WindowClassificationPerformanceEvaluator$, that is an evaluation task present on MOA `www.cs.waikato.ac.nz/∼abifet/MOA/API/`

the SVM component. It has been used with a linear kernel and the default parameter values on WEKA. Furthermore, no treatment, parsing or data cleaning process has been done to optimize the above-mentioned parameters.

The stopping criterion of GNG has been set to values between 100 and 500 prototypes trying to improve the scalability of SVMs while creating accurate summaries for every iteration. The chunk-size was tested in preliminary experimentation with values between $25,000$ and $2,000,000$ examples. Lower sizes make increase unnecessarily the training times and greater sizes did not fit on main memory on the server used. Regarding ENN we used 3, 5 and 7 as the number of neighbors for prototype reduction, as these are common values found in literature.

Table 3: Final parameters for benchmark of LibSVM* & iGNGSVM.

Algoritmh	Dataset	Chunk-size	GNG stop.crit.	ENN	SVM cost
$iGNGSVM_1$	SUSY	400,000	300 Proto.	3 nbr.	8
$iGNGSVM_2$	SUSY	400,000	300 Proto.	0 nbr.	8
$iGNGSVM_1$	HIGGS	100,000	300 Proto.	3 nbr.	8
$iGNGSVM_2$	HIGGS	100,000	150 Proto.	3 nbr.	8

3.3. Effect of parameters in the overall behaviour of iGNGSVM

Both versions of $iGNGSVM$ on SUSY and HIGGS showed a similar behaviour when changing the number of neighbors in ENN. The resulted accuracy and execution times versus the chunk-size was similar. However, it changed depending on the number of neighbors. $iGNGSVM_2$ using ENN obtained better accuracy than $iGNGSVM_1$. The lack of a kernel transformation function in the version of ENN implemented in this paper may play an important role in the results on the next section. This may cause that the deletion of noisy prototypes is less effective. Moreover, handling the SVs as data examples ($iGNGSVM_2$) might work better for datasets that are non-stationary in nature.

In terms of execution time, we have observed that it tended to decrease when increasing the chunk size (but not the GNG stopping criterion). For greater reductions of the data-chunk, the obtained accuracy rate did not decrease too much in comparison, helping to significantly reduce the number of iterations. The trade-off of these was a key factor, as bigger data-chunks do require bigger reductions which could have a cost in accuracy. The impact of the variation of the GNG stopping criterion was greater for small chunks-sizes ($25,000$ examples), given the greater number of iterations that it implies to process the full dataset.

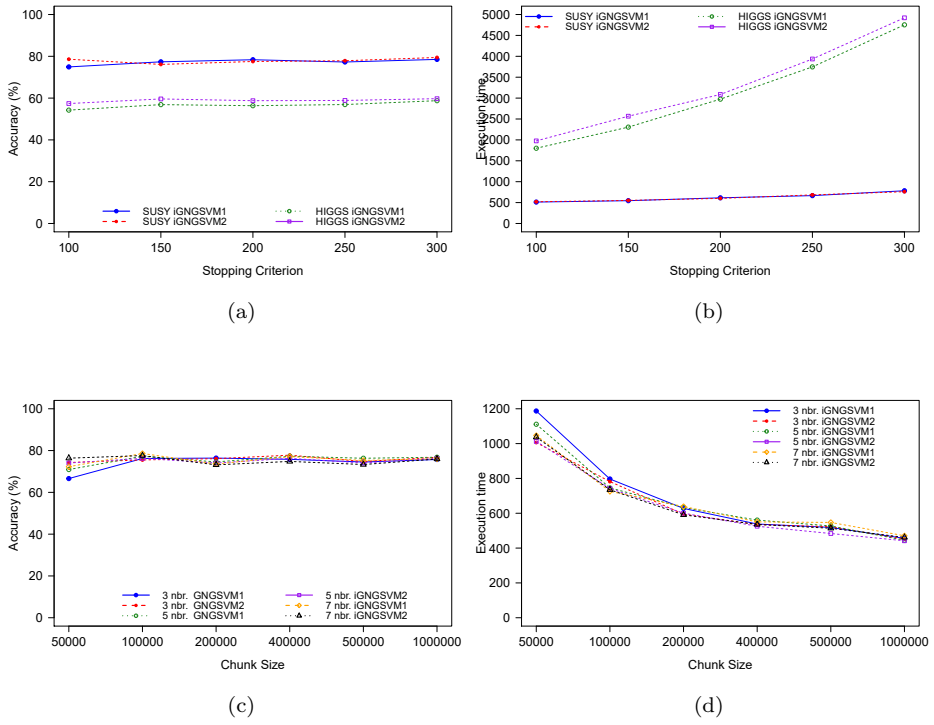


Figure 3: Accuracy rate (3a) and execution time (3b) for different stopping criterion using both versions of iGNGSVM on SUSY and HIGGS. Also, accuracy rate (3c) and execution time (3d) for different chunk-sizes and number of neighbors at ENN using both versions of iGNGSVM on SUSY.

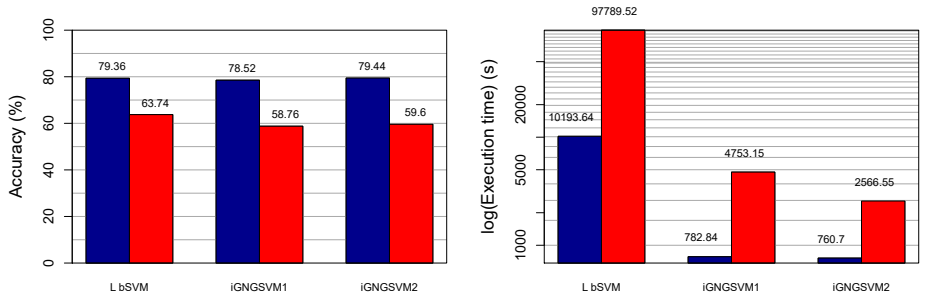
Measuring accuracy, the absolute maximum on SUSY and HIGGS was obtained for chunks of $100,000$ data examples. The best balance accuracy vs execution time was obtained using a chunk-size of 1 million examples, being therefore the chosen result for the benchmark on the next section. Setting the stopping criterion, the greatest values obtained the best precision in terms of topology created by GNG. However, this implied more execution time given both the greater number of iterations performed by GNG creating extra prototypes and, afterwards, a greater number of prototypes as input to ENN and the SVM classifier.

- SUSY performs well for stopping criterion equal or lower than 300 prototypes per class.
- HIGGS performs quite differently locally, but it tends to improve its accuracy and times when the stopping criterion increases.

Some of the tests for $iGNGSVM_2$ did not use ENN for SUSY as it does not make an exceptional impact to the model. The reduction of SVs that GNG already does in $iGNGSVM_2$ seems to be enough to delete noising prototypes in the dataset SUSY. The variation of the stopping criterion impacted differently on each dataset, which can be justified according to the number of attributes of HIGGS (28) greater than for SUSY (18). The linear SVM kernel used may handle in a different way the topology for the chunks of each dataset when expanding GNG until the stopping criterion. In this sense, any difference in the variance between data-chunks, distribution or features between both datasets might cause a huge impact.

3.4. Benchmark

Figure 4 and Table 4 shows the average execution time and accuracy on five iterations for datasets SUSY and HIGGS, respectively. For SUSY, LibSVM* obtained an accuracy rate of 79.36% on $10,200$ seconds. Nevertheless, iGNGSVM obtained similar results in just 700-800 seconds. GNG Prototype Reduction, SVs inheritance and ENN are the key difference between both algorithms. Therefore it can be seen how they help iGNGSVM to reduce the execution time without a significant cost on accuracy rate.



(a) (b)

Figure 4: Accuracy rate (4a) and execution time (4b) for every algorithm benchmarked. SUSY in blue bars, HIGGS in red bars.

Table 4 shows that in the larger dataset, HIGGS, iGNGSVM has a decrease of 5% compared to LibSVM. This suggests that, for this dataset, larger chunks are desirable in order to achieve a better results. However, with the current chunk size, reduction in time is very important. For instance, $iGNGSVM_2$ performs 37 times faster than LibSVM*. The reason is that the SVM process is performed on small number of prototypes. The accuracy is of the same order

Table 4: Accuracy rates and execution times obtained on iGNGSVM against LibSVM* (with data-chunks of 10,000 examples).

	LibSVM*		$iGNGSVM_1$		$iGNGSVM_2$	
dataset	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
SUSY	79.36	10,193.64	78.52	782.84	79.44	760.70
HIGGS	63.74	97,789.52	58.76	4,753.15	59.60	2,566.55

of literature results [42], improving often its result for both datasets. Moreover, the execution time is very low given the poor scalability of SVMs. Comparison with other algorithms may be biased by our use of a linear kernel.

4. Conclusions

This paper shows how incremental techniques as mini-batch learning and topology extraction methods as Prototype Generation can help making gigabytes of data to be manageable for just one server using very costly Machine Learning techniques as SVMs. This helps in many cases to avoid the current commercial trend of distributing processing, that also needs an specific infrastructures and changes on the algorithms to parallelize them.

This work proposes two versions for iGNGSVM. The difference between them is that the inheritance of SVs happens as prototypes in $iGNGSVM_1$ and as examples in $iGNGSVM_2$. This helps iGNGSVM to suit to different data qualities. Inheriting SVs as examples, GNG does not expand the topology until SVs that are at the boundaries of the network. This helps to reduce the unwanted overlapping between classes, albeit it compromises the knowledge about previous iterations. Nevertheless, $iGNGSVM_2$ performs better in a dataset which examples have a high variation between different iterations. Its structure helps then to forget useless SVs. The initial version of iGNGSVM, studied in section 2, was improved using ENN to handle the issue of the incremental accumulation of SVs that can generate overlap between different classes.

iGNGSVM was compared with a mini-batch version of SVM that did not inherit SVs of previous batches. This was due to the impossibility of loading the full dataset in memory. Both versions of iGNGSVM spend less execution time and achieve a better accuracy against the averaged result of all the batches in SVM in the SUSY dataset. Also, iGNGSVM performs 37 times faster than SVM in the biggest dataset used, reducing the execution time from more than a day to just minutes. iGNGSVM demonstrates to be scalable when changing the dataset size from a thousand to thousand of millions of examples.

The tests have also showed how $iGNGSVM_1$ needs ENN in order to reduce the accumulation of SVs. $iGNGSVM_2$ does not need ENN with SUSY and HIGGS because GNG in this version of the algorithm does not grow until the SVs in areas of overlap. The number of neighbors on ENN becomes as important as more data-chunks are processed. It also depends on the amount and

distribution of examples in the data-chunks. Some bias on specific data-chunks could generate a very different topology and therefore a SVs-set for subsequent iterations. iGNGSVM$_1$ obtains its best performance doing ENN with five neighbors.

These results with a lineal kernel on SVM and default parameters (except the stopping criterion in GNG) open good prospects on future tests with optimized parameters and nonlinear kernels. A distributed computing version of iGNGSVM could also help in upcoming proposals to process and classify continuous data flows in near real-time. As distributed and parallel Machine Learning are still recent trends, an exhaustive study focused on a distributed version of iGNGSVM should be proposed firstly.

About the future work of this paper, the soonest work should imply experimentation with non-linear kernels for SVMs and ENN. The usage of auto-adaptive filtering in the algorithm to perform sampling when handling unbalanced datasets might be another improvement. This may also be useful to optimize ENN depending on the data set properties. For example, establishing a threshold for the amount of inherited SVs incrementally.

As the present work is focused on artificial datasets, another application of the present work is an study of the implication of it on real world applications where the data is non-stationary. An improvement of both iGNGSVM versions using an adaptative sliding window (ADWIN) [43] could help to detect changes, modifying then their input parameters, components or even parts of the algorithm. Moreover, real-world applications normally have data issues as bias or unbalanced classes. An oversampling method as SMOTE [44] could help in the mitigation of this issue. Moreover, the use of dynamic percentages of the data-chunk size as stopping criterion, changing them depending on the last results may help.

Finally, another line of work would be applying distributed and parallel computing trends to iGNGSVM in order to make it relevant to process large streams of data at real-time. Regarding to big data streams, a future line is the implementation of iGNGSVM using Apache SAMOA[8] [45] to run it on the top of an Stream Processing Engine as Storm [9] or S4 [10]. GNG would be able to run in parallel for different classes. The possibility of using Map Reduce to process data-chunks iteratively (e.g. using Spark [46]) or in parallel (e.g. using Storm) is tempting as well. One of the options here is to perform parallel processing using SVMs in an ensemble, where different SVMs could be trained and tested at the same time. In this sense, many approaches in the literature have already introduced parallel mixtures of SVMs for large-scale problems [47] [48]. Another option would be to make a parallel version of the SVM algorithm. This last is a current trend in Machine Learning with big data sets as it is still in development stages (e.g. libraries like MLlib [49] and platforms as GraphLab[50]).

[8] http://samoa-project.net

References

[1] S. Sagiroglu, D. Sinanc, Big data: A review, in: Collaboration Technologies and Systems (CTS), 2013 International Conference on, 2013, pp. 42–47. doi:10.1109/CTS.2013.6567202.

[2] D. Laney, 3d data management: Controlling data volume, velocity and variety, META Group Research Note 6 (2001) 70.

[3] P. Zikopoulos, C. Eaton, et al., Understanding big data: Analytics for enterprise class hadoop and streaming data, McGraw-Hill Osborne Media, 2011.

[4] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks 54 (15) (2010) 2787 – 2805. doi:http://dx.doi.org/10.1016/j.comnet.2010.05.010.
URL http://www.sciencedirect.com/science/article/pii/S1389128610001568

[5] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters 51 (1) 107113. doi:10.1145/1327452.1327492.

[6] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine Learning 23 (1) (1996) 69–101. doi:10.1007/BF00116900.
URL http://dx.doi.org/10.1007/BF00116900

[7] A. Tsymbal, The problem of concept drift: Definitions and related work, Tech. rep. (2004).

[8] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, ACM, New York, NY, USA, 2001, pp. 97–106. doi:10.1145/502512.502529.
URL http://doi.acm.org/10.1145/502512.502529

[9] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, D. Ryaboy, Storm@twitter, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, ACM, New York, NY, USA, 2014, pp. 147–156. doi:10.1145/2588555.2595641.
URL http://doi.acm.org/10.1145/2588555.2595641

[10] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: Distributed stream computing platform, in: 2010 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 170–177. doi:10.1109/ICDMW.2010.172.

[11] M. Zaharia, T. Das, H. Li, S. Shenker, I. Stoica, Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters, in: Presented as part of the, 2012.

[12] M. Hearst, S. Dumais, E. Osman, J. Platt, B. Scholkopf, Support vector machines, Intelligent Systems and their Applications, IEEE 13 (4) (1998) 18–28. doi:10.1109/5254.708428.

[13] V. Vapnik, A. Y. Lerner, Recognition of patterns with help of generalized portraits, Avtomat. i Telemekh 24 (6) (1963) 774–780.

[14] B. E. Boser, I. M. Guyon, V. N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the fifth annual workshop on Computational learning theory, ACM, 1992, pp. 144–152.

[15] C.-C. Chang, C.-J. Lin, Libsvm: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (3) (2011) 27:1–27:27. doi:10.1145/1961189.1961199.
URL http://doi.acm.org/10.1145/1961189.1961199

[16] S. Tong, D. Koller, Support vector machine active learning with applications to text classification, J. Mach. Learn. Res. 2 (2002) 45–66. doi:10.1162/153244302760185243.
URL http://dx.doi.org/10.1162/153244302760185243

[17] J. Zheng, F. Shen, H. Fan, J. Zhao, An online incremental learning support vector machine for large-scale data, Neural Computing and Applications 22 (5) (2013) 1023–1035. doi:10.1007/s00521-011-0793-1.
URL http://dx.doi.org/10.1007/s00521-011-0793-1

[18] A. Neto, G. Barreto, Opposite maps: Vector quantization algorithms for building reduced-set svm and lssvm classifiers, Neural Processing Letters 37 (1) (2013) 3–19. doi:10.1007/s11063-012-9265-6.
URL http://dx.doi.org/10.1007/s11063-012-9265-6

[19] O. Linda, M. Manic, Gng-svm framework - classifying large datasets with support vector machines using growing neural gas, in: Neural Networks, 2009. IJCNN 2009. International Joint Conference on, 2009, pp. 1820–1826. doi:10.1109/IJCNN.2009.5178713.

[20] I. Triguero, J. Derrac, S. Garcıa, F. Herrera, Prototype generation for nearest neighbor classification: Survey of methods.

[21] J. Derrac, I. Triguero, S. Garcıa, F. Herrera, Survey of new approaches on prototype selection and generation.

[22] N. A. Syed, H. Liu, S. Huan, L. Kah, K. Sung, Handling concept drifts in incremental learning with support vector machines, in: In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99, ACM Press, 1999, pp. 317–321.

[23] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning (2000).

[24] N. A. Syed, S. Huan, L. Kah, K. Sung, Incremental learning with support vector machines (1999).

[25] S. Rüping, Incremental learning with support vector machines, in: icdm, IEEE, 2001, p. 641.

[26] P. Laskov, C. Gehl, S. Krüger, K.-R. Müller, Incremental support vector learning: Analysis, implementation and applications, J. Mach. Learn. Res. 7 (2006) 1909–1936.
URL http://dl.acm.org/citation.cfm?id=1248547.1248616

[27] P. P. Angelov, X. Zhou, Evolving fuzzy-rule-based classifiers from data streams, IEEE Transactions on Fuzzy Systems 16 (6) (2008) 1462–1475. doi:10.1109/TFUZZ.2008.925904.

[28] N. Kasabov, D. Filev, Evolving intelligent systems: Methods, learning, applications, in: 2006 International Symposium on Evolving Fuzzy Systems, 2006, pp. 8–18. doi:10.1109/ISEFS.2006.251185.

[29] E. Lughofer, P. Angelov, Handling drifts and shifts in on-line data streams with evolving fuzzy systems, Appl. Soft Comput. 11 (2) (2011) 2057–2068. doi:10.1016/j.asoc.2010.07.003.
URL http://dx.doi.org/10.1016/j.asoc.2010.07.003

[30] E. Lughofer, Evolving fuzzy systems-methodologies, advanced concepts and applications, Vol. 53, Springer, 2011.

[31] E. Lughofer, C. Cernuda, S. Kindermann, M. Pratama, Generalized smart evolving fuzzy systems, Evolving Systems 6 (4) (2015) 269–292. doi:10.1007/s12530-015-9132-6.
URL http://dx.doi.org/10.1007/s12530-015-9132-6

[32] M. Pratama, S. G. Anavatti, M. Joo, E. D. Lughofer, pclass: An effective classifier for streaming examples, IEEE Transactions on Fuzzy Systems 23 (2) (2015) 369–386. doi:10.1109/TFUZZ.2014.2312983.

[33] P. P. Angelov, D. P. Filev, An approach to online identification of takagi-sugeno fuzzy models, Trans. Sys. Man Cyber. Part B 34 (1) (2004) 484–498. doi:10.1109/TSMCB.2003.817053.
URL http://dx.doi.org/10.1109/TSMCB.2003.817053

[34] B. Fritzke, A growing neural gas network learns topologies, in: Advances in Neural Information Processing Systems 7, MIT Press, 1995, pp. 625–632.

[35] G. Dror, D. Raijman, I. Ulitsky, Analysis of gene expression data spring semester, Tel Aviv University, lecture 6 (3 2005).

[36] An experiment with the edited nearest-neighbor rule, Systems, Man and Cybernetics, IEEE Transactions on SMC-6 (6) (1976) 448–452. doi:10.1109/TSMC.1976.4309523.

[37] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, Data Min. Knowl. Discov. 6 (2) (2002) 153–172. doi:10.1023/A:1014043630878.
URL http://dx.doi.org/10.1023/A:1014043630878

[38] W. N. Street, Y. Kim, A streaming ensemble algorithm (sea) for large-scale classification, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 377–382.

[39] A. Abdiansah, R. Wardoyo, Article: Time complexity analysis of support vector machines (svm) in libsvm, International Journal of Computer Applications 128 (3) (2015) 28–34, published by Foundation of Computer Science (FCS), NY, USA.

[40] C. A. T. Mendes, M. Gattass, H. Lopes, Fgng: A fast multi-dimensional growing neural gas implementation, Neurocomputing 128 (2014) 328 – 340. doi:http://dx.doi.org/10.1016/j.neucom.2013.08.033.
URL http://www.sciencedirect.com/science/article/pii/S0925231213009259

[41] N. Jankowski, M. Grochowski, Comparison of Instances Seletion Algorithms I. Algorithms Survey, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 598–603. doi:10.1007/978-3-540-24844-6_90.
URL http://dx.doi.org/10.1007/978-3-540-24844-6_90

[42] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat Commun 5.
URL http://dx.doi.org/10.1038/ncomms5308

[43] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing., in: SDM, Vol. 7, SIAM, 2007, p. 2007.

[44] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: Synthetic minority over-sampling technique, J. Artif. Int. Res. 16 (1) (2002) 321–357.
URL http://dl.acm.org/citation.cfm?id=1622407.1622416

[45] G. De Francisci Morales, A. Bifet, Samoa: Scalable advanced massive online analysis, J. Mach. Learn. Res. 16 (1) (2015) 149–153.
URL http://dl.acm.org/citation.cfm?id=2789272.2789277

[46] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets., HotCloud 10 (2010) 10–10.

[47] R. Collobert, S. Bengio, Y. Bengio, A parallel mixture of svms for very large scale problems, Neural computation 14 (5) (2002) 1105–1114.

[48] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, V. Vapnik, Parallel support vector machines: The cascade svm, in: Advances in neural information processing systems, 2004, pp. 521–528.

[49] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, Mllib: Machine learning in apache spark, CoRR abs/1505.06807.
URL http://arxiv.org/abs/1505.06807

[50] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, J. M. Hellerstein, Distributed graphlab: A framework for machine learning and data mining in the cloud, Proc. VLDB Endow. 5 (8) (2012) 716–727. doi:10.14778/2212351.2212354.
URL http://dx.doi.org/10.14778/2212351.2212354