Model Approximation for Learning on Streams of Graphs on a Budget

Giovanni Da San Martino GMARTINO@QF.ORG.QA
Qatar Computing Research Institute, HBKU, Doha, Qatar

Nicolò Navarin NNAVARIN@MATH.UNIPD.IT
Department of Mathematics, University of Padova, via trieste 63, Padova, Italy

Alessandro Sperduti SPERDUTI@MATH.UNIPD.IT
Department of Mathematics, University of Padova, via trieste 63, Padova, Italy

Abstract

In many problem settings, for example on graph domains, online learning algorithms on streams of data need to respect strict time constraints dictated by the throughput on which the data arrive. When only a limited amount of memory (budget) is available, a learning algorithm will eventually need to discard some of the information used to represent the current solution, thus negatively affecting its classification performance. More importantly, the overhead due to budget management may significantly increase the computational burden of the learning algorithm. In this paper we present a novel approach inspired by the Passive Aggressive and the Lossy Counting algorithms. Our algorithm uses a fast procedure for deleting the less influential features. Moreover, it is able to estimate the weighted frequency of any feature and use it for prediction.

1. Introduction

Data streams are becoming more and more frequent in many application domains thanks to the advent of new technologies, mainly related to web and ubiquitous services. In a data stream, data elements are generated at a rapid rate and with no predetermined bound on their number. For this reason, processing should be performed very quickly (typically in linear time) and using bounded memory resources. Another characteristic of data streams is that they tend to evolve with time (*concept drift*). In many real world tasks involving streams, representing data as graphs is a key for success, e.g. in fault diagnosis systems for sensor networks (Alippi et al., 2012), malware detection (Eskandari & Hashemi, 2012), image classification or the discovery of new drugs (see Section 4 for some examples). In this paper, we address the problem of learning a classifier from a (possibly infinite) stream of graphs respecting a strict memory constraint. We propose an algorithm for processing graph streams on a fixed budget which performs comparably to the non-budget version, while being much faster. Our proposal is to combine a state-of-the-art online learning algorithm, i.e. Passive Aggressive with budget (Wang & Vucetic, 2010) with an adapted version of a classical result from stream mining, i.e. Lossy Counting (Manku & Motwani, 2002), to efficiently manage the available budget so to keep in memory only relevant features. Specifically, we extend Lossy Counting to manage both weighted features and a budget, while preserving theoretical guarantees on the introduced approximation error.

2. Problem Definition and Background

The traditional online learning problem can be summarized as follows. Suppose a, possibly infinite, data stream in the form of pairs $(x_1, y_1), \ldots, (x_t, y_t), \ldots$, is given. Here $x_t \in \mathbb{X}$ is the input example and $y_t = \{-1, +1\}$ its classification. Notice that, in the traditional online learning scenario, the label y_t is available to the learning algorithm after the class of x_t has been predicted. Moreover, data streams may be affected by *concept drift*, meaning that the concept associated to the labeling function may change over time, as well as the underlying example distribution. The goal is to find a function $h : \mathbb{X} \to \{-1, +1\}$ which minimizes the error, measured with respect to a given loss function, on the stream. There are additional constraints on the learning algorithm about its speed: it must be able to process the data at least at the rate it gets available to the learning algorithm. We consider the case in which the amount of memory available to represent $h()$ is limited by a budget value $\mathcal{B}$.

Most online learning algorithms assume that the input data can be described by a set of features, i.e. there exists a function $\phi : \mathbb{X} \to \mathbb{R}^s$, which maps the input data onto a feature vector of size s where learning is performed[1]. In this paper it is assumed that s is very large (possibly infinite) but only a finite number of $\phi(x)$ elements, for every x, is not null, i.e. $\phi(x)$ can be effectively represented in sparse format. Note that most current state-of-the-art graph kernels (Shervashidze & Borgwardt, 2009; Costa & De Grave, 2010; Da San Martino et al., 2012) have such property. Due to lack of space we will focus on the ODD kernel (Da San Martino et al., 2012) in the experimental section: its features are tree structures extracted from specific visits of the graph. While there exists a large number of online learning algorithms which our proposal can be applied to (Rosenblatt, 1958), here we focus our attention on the Passive Aggressive (PA) (Crammer et al., 2006; Wang & Vucetic, 2010), given its state-of-the-art performances. Our aim is to modify its budget management rule with our proposed technique. There are two version of the PA, *Primal* and *Dual*.

The *Primal* algorithm represents the solution as a sparse weight vector $w \in \mathbb{R}^s$ (with $|w|$ being the number of non-zero features of w). In machine learning w is often referred to as the *model*. Let us define the score of an example as:

$$S(x_t) = w_t \cdot \phi(x_t). \tag{1}$$

The update rule of the PA finds a tradeoff between two competing goals: preserving w and changing it in such a way that x_t is correctly classified. In (Crammer et al., 2006) it is shown that the optimal update rule is:

$$\tau_t = \min\left(C, \frac{\max(0, 1 - S(x_t))}{\|x_t\|^2}\right), \tag{2}$$

where C is the tradeoff parameter between the two competing goals above.

The *Dual* algorithm represents the solution as a sequence of examples exploiting $w = \sum_{i \in M} y_i \tau_i \phi(x_i)$. It uses the kernel trick to replace the dot products $\phi(x_t) \cdot \phi(x_u)$ in the scoring function with a corresponding kernel function $K(x_t, x_u)$. The memory usage for the dual algorithm is computed in terms of the size of the input examples in M. When there are no budget constraints *Primal* and *Dual* algorithms compute the same solution. Since the problem setting imposes not to use more memory than a predefined budget $\mathcal{B}$, whenever the size of w exceeds such threshold, i.e. $|w| > \mathcal{B}$, elements of w must be removed until all the new features of x_t can be inserted into w. Notice that budget constraints are not taken into account in the original formulation of the PA.

[1]While the codomain of $\phi()$ could be of infinite size, in order to simplify the notation we will use $\mathbb{R}^s$ in the following.

2.1. Lossy Counting

The *Lossy Counting* is an algorithm for computing frequency counts exceeding a user-specified threshold over data streams (Manku & Motwani, 2002). Let $s \in (0, 1)$ be a support threshold, $\epsilon \ll s$ an error parameter, and N the number of items of the stream seen so far. By using only $O\left(\frac{1}{\epsilon}\log(\epsilon N)\right)$ space to keep frequency estimates, *Lossy Counting*, at any time N is able to *i)* list any item whose frequency exceeds ϵN; *ii)* avoid listing any item with frequency less than $(s - \epsilon)N$. Moreover, the error of the estimated frequency of any item is at most ϵN. In the following the algorithm is sketched, for more details refer to (Manku & Motwani, 2002). The stream is divided into buckets B_i. The size of a bucket is $|B_i| = \lceil \frac{1}{\epsilon} \rceil$ items. Note that the size of a bucket is determined a priori because ϵ is a user-defined parameter. The frequency of an item f in a bucket B_i is represented as $F_{f,i}$, the overall frequency of f is $F_f = \sum_i F_{f,i}$. The algorithm makes use of a data structure $\mathcal{D}$ composed by tuples $(f, \Phi_{f,i}, \Delta_i)$, where $\Phi_{f,i}$ is the frequency of f since it has been inserted into $\mathcal{D}$, and Δ_i is an upper bound on the estimate of F_f at time i. The algorithm starts with an empty $\mathcal{D}$. For every new event f arriving at time i, if f is not present in $\mathcal{D}$, then a tuple $(f, 1, i - 1)$ is inserted in $\mathcal{D}$, otherwise $\Phi_{f,i}$ is incremented by 1. Every $|B_i|$ items all those tuples such that $\Phi_{f,i} + \Delta_i \leq i$ are removed. The authors prove that, after observing N items, if $f \notin \mathcal{D}$ then $\Phi_{f,N} \leq \epsilon N$ and $\forall (f, \Phi_{f,N}, \Delta_N). \Phi_{f,N} \leq F_f \leq \Phi_{f,N} + \epsilon N$.

3. Our Proposal

In this section, we first propose a Lossy Counting (LC) algorithm with budget which is able to deal with events weighted by positive real values. Then, we show how to apply the proposed algorithm to the Passive Aggressive algorithm in the case of a stream of graphs.

3.1. LCB: LC with budget for weighted events

The Lossy Counting algorithm does not guarantee that the available memory will be able to contain the ϵ-deficient synopsis $\mathcal{D}$ of the observed items. Because of that, we propose an alternative definition of Lossy Counting that addresses this issue and, in addition, is able to deal with weighted events. Specifically, we assume that the stream emits events e constituted by couples (f, ϕ), where f is an item id and $\phi \in \mathbb{R}^+$ is a positive weight associated to f. Different events may have the same item id while having different values for the weight, e.g. $e_1 = (f, 1.3)$ and $e_2 = (f, 3.4)$. We are interested in maintaining a synopsis that, for each observed item id f, collects the sum of weights observed in association with f. Moreover, we have to do that on a memory budget. To manage the budget, we propose to decouple the ϵ parameter from the size of the bucket: we use buckets of variable sizes, i.e. the i-th bucket B_i will contain all the occurrences of events which can be accommodated into the available memory, up to the point that $\mathcal{D}$ will use the full budget and no new event can be inserted into it. This strategy implies that the approximation error will vary according to the size of the bucket, i.e. $\epsilon_i = \frac{1}{|B_i|}$. Having buckets of different sizes, the index of the current bucket $b_{current}$ is defined as $b_{current} = 1 + \max_k[\sum_{s=1}^{k} |B_s| < N]$. Deletions occur when there is no more space to insert a new event in $\mathcal{D}$. Trivially, if N events have been observed when the i-th deletion occurs, $\sum_{s=1}^{i} |B_s| = N$ and, by definition, $b_{current} \leq \sum_{s=1}^{i} \epsilon_s |B_s|$.

Let $\phi_{f,i}(j)$ be the weight of the j-th occurrence of f in B_i, and the cumulative weight associated to f in B_i as $w_{f,i} = \sum_{j=1}^{F_{f,i}} \phi_{f,i}(j)$. The total weight associated to bucket B_i can then be defined as $W_i = \sum_{f \in \mathcal{D}} w_{f,i}$. We now want to define a synopsis $\mathcal{D}$ such that, having observed $N = \sum_{s=1}^{i} |B_s|$ events, the estimated cumulative weights are *less* than the true cumulative weights by at most $\sum_{s=1}^{i} \epsilon_i W_i$, where we recall that $\epsilon_i = \frac{1}{|B_i|}$. In order to do that we define the cumulated weight for item f in $\mathcal{D}$, after having observed all the events in B_i, as $\Phi_{f,i} = \sum_{s=i_f}^{i} w_{f,s}$, where $i_f \leq i$ is the largest index of the bucket where f has been inserted in $\mathcal{D}$. The deletion test is then defined as

$$\Phi_{f,i} + \Delta_{i_f} \leq \Delta_i \tag{3}$$

where $\Delta_i = \sum_{s=1}^{i} \frac{W_i}{|B_i|}$. However, we have to cover also the special case in which the deletion test is not able to delete any item from $\mathcal{D}$[2]. We propose to solve this problem as follows: we assume that in this case a new bucket B_{i+1} is created containing just a single *ghost* event $(f_{ghost}, \Phi_i^{min} - \Delta_i)$, where $f_{ghost} \notin \mathcal{D}$ and $\Phi_i^{min} = \min_{f \in \mathcal{D}} \Phi_{f,i}$, that we do not need to store in $\mathcal{D}$. In fact, when the deletion test is run, $\Delta_{i+1} = \Delta_i + \frac{W_{i+1}}{|B_{i+1}|} = \Phi_i^{min}$ since $W_{i+1} = \Phi_i^{min} - \Delta_i$ and $|B_{i+1}| = 1$, which will cause the ghost event to be removed since $\Phi_{ghost,i+1} = \Phi_i^{min} - \Delta_i$ and $\Phi_{ghost,i+1} + \Delta_i = \Phi_i^{min}$. Moreover, since $\forall f \in \mathcal{D}$ we have $\Phi_{f,i} = \Phi_{f,i+1}$, all f such that $\Phi_{f,i+1} = \Phi_i^{min}$ will be removed. By construction, there will always be one such item.

Theorem 1. *Let $\Phi_{f,i}^{true}$ be the true cumulative weight of f after having observed $N = \sum_{s=1}^{i} |B_s|$ events. Whenever an entry $(f, \Phi_{f,i}, \Delta)$ gets deleted, $\Phi_{f,i}^{true} \leq \Delta_i$.*

Proof. We prove by induction. Base case: $(f, \Phi_{f,1}, 0)$ is deleted only if $\Phi_{f,1} = \Phi_{f,1}^{true} \leq \Delta_1$. Induction step: let $i^* > 1$ be the index for which $(f, \Phi_{f,i^*}, \Delta)$ gets deleted. Let $i_f < i^*$ be the largest value of $b_{current}$ for which the entry was inserted. By induction $\Phi_{f,i_f}^{true} \leq \Delta_{i_f}$ and all the weighted occurrences of events involving f are collected in Φ_{f,i^*}. Since $\Phi_{f,i^*}^{true} = \Phi_{f,i_f}^{true} + \Phi_{f,i^*}$ we conclude that $\Phi_{f,i^*}^{true} \leq \Delta_{i_f} + \Phi_{f,i^*} \leq \Delta_{i^*}$. □

Theorem 2. *If $(f, \Phi_{f,i}, \Delta) \in \mathcal{D}$, $\Phi_{f,i} \leq \Phi_{f,i}^{true} \leq \Phi_{f,i} + \Delta$.*

Proof. If $\Delta = 0$ then $\Phi_{f,i} = \Phi_{f,i}^{true}$. Otherwise, $\Delta = \Delta_{i_f}$, and an entry involving f was possibly deleted sometimes in the first i_f buckets. From the previous theorem, however, we know that $\Phi_{f,i_f}^{true} \leq \Delta_{i_f}$, so $\Phi_{f,i} \leq \Phi_{f,i}^{true} \leq \Phi_{f,i} + \Delta$. □

We notice that, if $\forall e = (f, \phi),\ \phi = 1$, the above theorems apply to the not weighted version of the algorithm.

Let now analyze the proposed algorithm. Let $\mathcal{O}_i$ be the set of items that have survived the last (i.e., $(i-1)$-th) deletion test and that have been *observed* in B_i, $\mathcal{I}_i$ be the set of items that have been *inserted* in $\mathcal{D}$ after the last (i.e., $(i-1)$-th) deletion test; notice that $\mathcal{O}_i \cup \mathcal{I}_i$ is the set of all the items observed in B_i, however it may be properly included into the set of items stored in $\mathcal{D}$. Let $w_i^{\mathcal{O}} = \sum_{f \in \mathcal{O}_i} F_{f,i}$ and $w_i^{\mathcal{I}} = \sum_{f \in \mathcal{I}_i} F_{f,i}$. Notice that $w_i^{\mathcal{I}} > 0$, otherwise the budget would not be fully used; moreover, $|B_i| = w_i^{\mathcal{O}} + w_i^{\mathcal{I}}$. Let $W_i^{\mathcal{O}} = \sum_{f \in \mathcal{O}_i} w_{f,i}$ and $W_i^{\mathcal{I}} = \sum_{f \in \mathcal{I}_i} w_{f,i}$. Notice that $W_i = W_i^{\mathcal{O}} + W_i^{\mathcal{I}}$, and that $\frac{W_i}{|B_i|} = \frac{w_i^{\mathcal{O}}}{|B_i|}[\widehat{W}_i^{\mathcal{O}} - \widehat{W}_i^{\mathcal{I}}] + \widehat{W}_i^{\mathcal{I}} = p_i^{\mathcal{O}}[\widehat{W}_i^{\mathcal{O}} - \widehat{W}_i^{\mathcal{I}}] + \widehat{W}_i^{\mathcal{I}}$, where $p_i^{\mathcal{O}} = \frac{w_i^{\mathcal{O}}}{|B_i|}$ is the fraction of updated items in B_i, $\widehat{W}_i^{\mathcal{O}} = \frac{W_i^{\mathcal{O}}}{w_i^{\mathcal{O}}}$ is the average of the updated items, $\widehat{W}_i^{\mathcal{I}} = \frac{W_i^{\mathcal{I}}}{w_i^{\mathcal{I}}}$ is the average of the inserted items. Thus $\frac{W_i}{|B_i|}$ can be expressed as a convex combination of the average of the updated items and the average of the inserted items, with combination coefficient equal to the fraction of updated items in the current bucket. Thus, the threshold Δ_i used by the i-th deletion test can be written as

$$\Delta_i = \sum_{s=1}^{i} p_s^{\mathcal{O}}[\widehat{W}_s^{\mathcal{O}} - \widehat{W}_s^{\mathcal{I}}] + \widehat{W}_s^{\mathcal{I}}. \tag{4}$$

Combining eq. (3) with eq. (4) we obtain $\Phi_{f,i} \leq \sum_{s=i_f}^{i} p_s^{\mathcal{O}}[\widehat{W}_s^{\mathcal{O}} - \widehat{W}_s^{\mathcal{I}}] + \widehat{W}_s^{\mathcal{I}}$. If $f \in \mathcal{I}_i$, then $i_f = i$ and the test reduces to $w_{f,i} \leq p_i^{\mathcal{O}}[\widehat{W}_i^{\mathcal{O}} - \widehat{W}_i^{\mathcal{I}}] + \widehat{W}_i^{\mathcal{I}}$. If $f \notin \mathcal{I}_i$ (notice that this condition means that $i_f < i$, i.e. $f \in \mathcal{I}_{i_f}$), then the test can be rewritten as $w_{f,i} \leq p_i^{\mathcal{O}}[\widehat{W}_i^{\mathcal{O}} - \widehat{W}_i^{\mathcal{I}}] + \widehat{W}_i^{\mathcal{I}} - \sum_{s=i_f}^{i-1} \gamma_{f,s}$, where $\gamma_{f,s} = w_{f,s} - p_s^{\mathcal{O}}[\widehat{W}_s^{\mathcal{O}} - \widehat{W}_s^{\mathcal{I}}] + \widehat{W}_s^{\mathcal{I}}$ is the *credit/debit*

[2] E.g, consider the stream $(f_1, 10)$, $(f_2, 1)$, $(f_3, 10)$, $(f_4, 15)$, $(f_1, 10)$, $(f_3, 10)$, $(f_5, 1)$, .. and budget equal to 3 items: the second application of the deletion test will fail to remove items from $\mathcal{D}$.

gained by f for bucket B_s. Notice that, by definition, $\forall k \in \{i_f, \ldots, i\}$ the following holds $\sum_{s=i_f}^{k} \gamma_{f,s} > 0$.

3.2. LCB-PA on streams of Graphs

This section describes an application of the results in Section 3.1 to the primal PA algorithm. We will refer to the algorithm described in this section as LCB-PA. The goal is to use the synopsis $\mathcal{D}$ created and maintained by LCB to best approximate w, according to the available budget. This is obtained by LCB since only the most influential w_i values will be stored into $\mathcal{D}$. A difficulty in using the LCB synopsis is due to the fact that LCB can only manage positive weights. We overcome this limitation by storing for each feature f in $\mathcal{D}$ a version associated to positive weight values and a version associated to (the modulus) of negative values. Let's detail the proposed approach in the following. First of all, recall that we consider the features generated by the $\phi()$ mapping of the ODD kernel, where the Reproducing Kernel Hilbert Space is very large but, for each example, only a small fraction of the features are nonzero. When a new graph arrives from the stream, it is first decomposed into a bag of features. Then the score for the graph is computed according to eq. (1). If the graph is misclassified then the synopsis $\mathcal{D}$ (i.e., w) has to be updated. In this scenario, an event corresponds to a feature f of the current input graph G. The weight $\phi_{f,i}(j)$ of a feature f appearing for the j-th time in the i-th bucket B_i, is computed multiplying its frequency in the graph G with the corresponding τ value computed for G according to eq. (2), which may result in a negative weight. $\Phi_{f,i}$ is the weighted sum of all $\phi_{f,i}(j)$ values of the feature f since f has last been inserted into $\mathcal{D}$. In this way, the LCB algorithm allows to maintain an approximate version of the full w vector by managing the feature selection and model update steps. In order to cope with negative weights, the structure $\mathcal{D}$ is composed by tuples $(f, |f|, \Phi^+_{f,i}, \Phi^-_{f,i})$, where $\Phi^+_{f,i}$ corresponds to $\Phi_{f,i}$ computed only on features whose graph G has positive classification ($\Phi^-_{f,i}$ is the analogous for the negative class). Whenever the size of $\mathcal{D}$ exceeds the budget $\mathcal{B}$, all the tuples satisfying eq. (3) are removed from $\mathcal{D}$. Here Δ_i can be interpreted as the empirical mean of the τ values observed in the current bucket. Note that the memory occupancy of $\mathcal{D}$ is now $4|w|$, where $|w|$ is the number of features in $\mathcal{D}$. The update rule of eq. (2) is kept. However, when a new tuple is inserted into $\mathcal{D}$ at time N, the Δ_N value is added to the τ value computed for G. The idea is to provide an upper bound to the $\Phi^+_{f,N}$, $\Phi^-_{f,N}$ values that might have been deleted in the past from $\mathcal{D}$. Theorem 1 shows that indeed Δ_N is such upper bound.

4. Experiments

In this section, we report an experimental evaluation of the algorithm proposed in Section 3.2 (*LCB-PA*).

4.1. Datasets and Experimental Setup

We generated different graph streams starting from two graph datasets available from the PubChem website: *AID:123* and *AID:109*. They comprise $40,876$ and $41,403$ compounds. Each compound is represented as a graph where the nodes are the atoms and the edges their bonds. Every compound has a corresponding activity score. The class of a compound is determined by setting a threshold β on the activity score. By varying the threshold a concept drift is obtained. Three streams have been generated by concatenating the datasets with varying β values: "$Chemical1$" as *AID:123* β=40, *AID:123* β=47, *AID:109* β=41, *AID:109* β=50; "$Chemical2$" as *AID:123* β=40, *AID:109* β=41, *AID:123* β=47, *AID:109* β=50; "$Chemical3$" as the concatenation of $Chemical1$ and $Chemical2$. We further generated a stream of graphs from the *LabelMe* dataset[3], which is composed of images whose objects are manually annotated (Russell & Torralba, 2008). The set of objects of any image were connected by the Delaunay triangulation (Su & Drysdale, 1996) and turned into a graph. Each of the resulting $5,342$ graphs belong to 1 out of 6 classes. A binary stream is obtained by concatenating $i = 1..6$ times the set of graphs: the i-th time only the graphs belonging to class i are labeled as positive, the others together forming the negative class. The size of the stream is $32,052$. We refer to this stream as $Image$. We considered as baseline the budget PA algorithm in its $Primal$ and $Dual$ versions, both using the $\phi()$ mapping of the ODD kernel. We compared them against the proposed *LCB-PA* algorithm. After a preliminary set of experiments, we determined the best removal policies: the feature with lowest value in w for $Primal$ PA and the example with lowest τ value for *Dual*. The C parameter has been set to 0.01, while the kernel parameters has been set to λ=1.6, h=3 for chemical datasets, and to λ=1, h=3 for the $Image$ dataset.

4.2. Results and discussion

The measure we adopted for performance assessment is the Balanced Error Rate (*BER*), $\text{BER} = \frac{1}{2}\left(\frac{\text{fp}}{\text{tn + fp}} + \frac{\text{fn}}{\text{fn + tp}}\right)$, where tp, tn, fp and fn are, respectively, true positive, true negative, false positive and false negative examples. In order to increase the readability of the results, in the following we report the 1-BER values. This implies that higher values mean

[3] http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php

	Budget	Dual	Primal	LCB-PA	PA ($\mathcal{B} = \infty$)
$Chemical1$	10,000	0.534	**0.629**	0.608	
	25,000	0.540	**0.638**	0.637	
	50,000	0.547	0.642	**0.644**	**0.644**
	75,000	-	0.643	**0.644**	($\mathcal{B}$=182, 913)
	100,000	-	**0.644**	**0.644**	
$Chemical2$	10,000	0.535	**0.630**	0.610	
	25,000	0.541	**0.638**	**0.638**	
	50,000	0.546	0.642	**0.644**	0.644
	75,000	-	0.644	**0.645**	($\mathcal{B}$=182, 934)
	100,000	-	0.644	**0.645**	
$Chemical3$	10,000	0.532	**0.640**	0.601	
	25,000	0.542	**0.652**	0.643	
	50,000	0.549	**0.658**	**0.658**	**0.661**
	75,000	-	**0.660**	**0.660**	($\mathcal{B}$=183, 093)
	100,000	-	**0.661**	**0.661**	
$Image$	10,000	0.768	0.845	**0.855**	
	25,000	0.816	0.846	**0.853**	
	50,000	0.822	0.846	**0.852**	0.852
	75,000	-	0.846	**0.852**	($\mathcal{B}$=534, 903)
	100,000	-	0.845	**0.852**	

Table 1. 1-BER values for $Primal$, $Dual$ and *LCB-PA* algorithms, with different budget sizes, on $Chemical1$, $Chemical2$, $Chemical3$ and $Image$ datasets. Best results for each row are reported in bold. A missing data item indicates that the execution did not finish in 3 days.

	$Chemical1$		$Chemical2$	
Budget	$10,000$	$50,000$	$10,000$	$50,000$
Dual	5h44m	31h02m	5h42m	31h17m
Primal	44m19s	1h24m	43m54s	1h22m
LCB-PA	4m5s	3m55s	3m52s	3m57s
	$Chemical3$		$Image$	
Budget	$10,000$	$50,000$	$10,000$	$50,000$
Dual	10h50m	60h31m	49m34s	6h18m
Primal	1h32m	2h44m	7m21s	33m18s
LCB-PA	7m41s	7m45s	0m49s	0m50s

Table 2. Execution times of $Dual$, $Primal$ and *LCB-PA* algorithms for budget values $\mathcal{B} = 10,000$ and $\mathcal{B} = 50,000$.

better performances. In our experiments, we sampled 1-BER every 50 examples. In Table 1 are reported, for each algorithm and budget, the average of the 1-BER values over the whole stream. It is easy to see that the performances of the $Dual$ algorithm are poor. Indeed, there is no single algorithm/budget combination in which the performance of this algorithm is competitive with the other two. This is probably due to the fact that support graphs may contain many features that are not discriminative for the tasks. Let us consider the $Primal$ and the *LCB-PA* algorithms. Their performances are almost comparable. Concerning the chemical datasets, it's clear that on the three datasets the algorithm $Primal$ performs better than *LCB-PA* for budget sizes up to $25,000$. For higher budget values, the performances of the two algorithms are very close, with *LCB-PA* performing better on $Chemical1$ and $Chemical2$ datasets, while the performances are exactly the same on $Chemical3$. Let's analyze in more detail this behavior of the algorithms. In $Primal$ every feature uses 3 budget units, while in *LCB-PA* 4 units are consumed. In the considered chemical streams, on average every example is made of 54.56 features. This means that, with budget $10,000$, $Primal$ can store approximatively the equivalent of 60 examples, while *LCB-PA* only 45, i.e. a 25% difference. When the budget increases, such difference reduces. The *LCB-PA* performs better then the $Primal$ PA with budget size of $50,000$ or more. Notice that *LCB-PA*, with budget over a certain threshold, reaches or improves over the performances of the PA with $\mathcal{B} = \infty$. On the $Image$ dataset we have a different scenario. *LCB-PA* with budget $10,000$ already outperforms the other algorithms. Table 2 reports the time needed to process the streams. It is clear from the table that *LCB-PA* is by far the fastest algorithm. $Dual$ algorithm is slow because it works in the dual space, so it has to calculate the kernel function several times. Notice that when the *LCB-PA* performs the cleaning procedure, it removes far more features from the budget then $Primal$. As a consequence, $Primal$ calls the cleaning procedure much more often than *LCB-PA*, thus inevitably increasing its execution time. For example, on $Chemical1$ with budget $50,000$, $Primal$ executes the cleaning procedure $132,143$ times, while *LCB-PA* only 142 times. Notice that a more aggressive pruning for the baselines could be performed by letting a user define how many features have to be removed. Such a parameter could be chosen a priori, but it would be unable to adapt to a change in the data distribution and there would be no guarantees on the quality of the resulting model. The parameter can also be tuned on a subset of the stream, if the problem setting allows it. In general, developing a strategy to tune the parameter is not straightforward and the tuning procedure might have to be repeated multiple times, thus increasing significantly the overall computational burden of the learning algorithm. On the contrary, our approach employs a principled, automatic way, described in eq. (4), to determine how many and which features to prune.

5. Conclusions

This paper presented a fast technique for estimating the weighted frequency of a stream of features based on an extended version of *Lossy Counting* algorithm. It uses it for: *i)* pruning the set of features of the current solution such that it is ensured that it never exceeds a predefined budget; *ii)* prediction, when a feature not present in the current solution is first encountered. The results on streams of graphs show that the proposed technique for managing the budget is much faster than competing approaches. Its classification performance, provided the budget exceeds a practically very low value, is superior to the competing approaches, even without budget constraints.

References

Alippi, Cesare, Roveri, Manuel, and Trovò, Francesco. A ”learning from models” cognitive fault diagnosis system. In *ICANN*, pp. 305–313, 2012.

Costa, Fabrizio and De Grave, Kurt. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, 2010.

Crammer, Koby, Dekel, Ofer, Keshet, Joseph, Shalev-Shwartz, Shai, and Singer, Yoram. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

Da San Martino, Giovanni, Navarin, Nicolò, and Sperduti, Alessandro. A tree-based kernel for graphs. In *Proceedings of the 12th SIAM International Conference on Data Mining*, pp. 975–986, 2012.

Eskandari, Mojtaba and Hashemi, Sattar. A graph mining approach for detecting unknown malwares. *Journal of Visual Languages & Computing*, mar 2012. ISSN 1045926X. doi: 10.1016/j.jvlc.2012.02.002. URL http://dx.doi.org/10.1016/j.jvlc.2012.02.002.

Manku, Gurmeet Singh and Motwani, Rajeev. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB ’02, pp. 346–357. VLDB Endowment, 2002. URL http://dl.acm.org/citation.cfm?id=1287369.1287400.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

Russell, BC and Torralba, Antonio. LabelMe: a database and web-based tool for image annotation. *International journal of Computer Vision*, 77(1-3):157–173, 2008. URL http://www.springerlink.com/index/76X9J562653K0378.pdf.

Shervashidze, Nino and Borgwardt, Karsten. Fast subtree kernels on graphs. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 22*, pp. 1660–1668. 2009.

Su, Peter and Drysdale, Robert L Scot. A Comparison of Sequential Delaunay Triangulation Algorithms. pp. 1–24, 1996.

Wang, Zhuang and Vucetic, Slobodan. Online passive-aggressive algorithms on a budget. *Journal of Machine Learning Research - Proceedings Track*, 9:908–915, 2010.