Tackling Temporal Effects in Automatic Document Classification

Thiago Salles[1]⋆, Thiago Cardoso[1], Vitor Oliveira[1], Leonardo Rocha[2], Marcos André Gonçalves[1]

[1] Dep. de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos 6627 - ICEx - 31270-010 Belo Horizonte, Brasil
{tsalles,thiagon,vitorco,mgoncalv}@dcc.ufmg.br
[2] Dep. de Ciência da Computação - Universidade Federal de São João Del Rei (UFSJ)
Pc. Dr. Augusto Chagas Viegas 17 - DCOMP - 36300-088 São João Del Rei, Brasil
lcrocha@ufsj.edu.br

Abstract. Automatic Document Classification (ADC) has become an important research topic due the rapid growth in volume and complexity of data produced nowadays. ADC usually employs a supervised learning strategy, where we first build a classification model using pre-classified documents and then use it to classify unseen documents. One major challenge in building classifiers has to do with the temporal evolution of the characteristics of the dataset (a.k.a temporal effects). However, most of the current techniques for ADC does not consider this evolution while building and using the classification models. Recently we have proposed temporally-aware algorithms for ADC in order to properly handle these temporal effects. Despite of the their effectiveness, the temporally-aware classifiers have a major side effect of being naturally lazy, since they need to know the creation time of the test document to build the model. Such lazy property incurs in a potentially high test phase runtime and brings a critical scalability issue that may make these classifiers infeasible to handle large volumes of data, such as the Web and very large digital libraries. This work aims at addressing the following challenge: can we deal with the temporal effects in some entirely off-line setting, reducing the test phase runtime and without compromising its effectiveness due to varying data distributions? We propose to address this question by tackling the temporal effects from a data engineering perspective. We devise a pre-processing—classifier independent—step able to moving all the overhead of considering the temporal effects to an off-line setting, called Cascaded Temporal Smoothing (CTS). The CTS consists of a controlled data oversampling strategy which aims at smoothing the observed temporal effects in the data distribution. This new training set can be used by any traditional classifier in a way that produces similar effectiveness as the lazy temporally-aware classifiers but not incurring in any overhead at test phase. As our experimental evaluation shows, the use of CTS before learning a traditional Naïve Bayes classifier was able to improve classification effectiveness in two real datasets (gains up to 5.00% in terms of MacroF$_1$) exhibiting scalability properties not present in the lazy temporally-aware classifiers, guaranteeing its practical applicability in large classification problems.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data mining

General Terms: Algorithms, Experimentation

Keywords: automatic document classification, temporal effects, oversampling

1. INTRODUCTION

Automatic Document Classification (ADC) plays an important role in many information retrieval applications. Basically, the ADC goal is to create effective models to automatically associate documents

⋆Corresponding author.

This research is partially funded by the MCT/CNPq/CT-INFO project InfoWeb (grant number 55.0874/2007 – 0), by InWeb – The National Institute of Science and Technology for the Web (MCT/CNPq/FAPEMIG grant number 573871/2008 – 6), and by the authors' individual research grants from FAPEMIG, FINEP and CNPq.

with semantically meaningful categories. With the rapid growth in volume and complexity of data produced nowadays, this task has become essential for a variety of applications such as ad-matching systems [Rajan et al. 2010], news recommendation [Chiang and Chen 2004] and spam filtering [Fdez-Riverola et al. 2007].

Similarly to other machine learning techniques, ADC usually follows a supervised learning approach: a set of already classified documents (training set) is employed for creating a classification model. Once built, the model is used for labeling a new set of unclassified documents (test set). A fundamental assumption of the majority of ADC algorithms is that the data used to learn a model are random samples independently and identically distributed (*i.i.d.*) from a stationary distribution that governs the test data. However, in many (perhaps most) real-world classification problems, the training data may not be randomly drawn from the same distribution as the test data (to which the classifier will be applied).

Recently, in [Mourão et al. 2008] the authors have distinguished three different temporal effects that may have a significant impact on ADC algorithms. More specifically, the *term distribution variation* effect refers to changes in the term's representativeness with respect to the classes as time goes by (i.e., the observed variations over time in the strength of term-class relationships). The second effect, *class distribution variation*, accounts for the impact of the temporal evolution on the relative frequencies of the classes. Finally, the *class similarity variation* considers how the similarity between classes, as a function of the terms that occur in their documents, changes over time. Therefore, as the authors showed, the design of classification models that account for the temporal dynamics of data is a critical challenge to be tackled. Neglecting this issue potentially leads to less effective classifiers as assumptions made when the model is built (i.e., learned) may no longer hold due to the temporal effects.

In [Salles et al. 2010], the authors proposed a strategy to incorporate temporal information directly into document classifiers, aiming at improving their effectiveness by handling data with varying distributions. Such strategy is based on the evolution of the term-class relationship over time, captured by a metric of *dominance*. In that work, the authors developed a statistically well founded way to determine the *temporal weighting function* (TWF) according to the characteristics of the target dataset to which the classifier will be applied. Having defined the TWF, they also proposed two strategies to incorporate this function to ADC algorithms, both following a lazy instance-weighting classification approach. In these strategies, the weights assigned to each training document depend on the notion of a temporal distance δ, defined as the difference between the time of creation p of a training document and a reference time point p_r (i.e., the creation time of test document). One drawback of these lazy algorithms is that they postpone the construction of a classification model until the classifier receives the test document, in order to assess its creation time. Thus, for each test document, a specific classification model is learned. In fact, learning a classification model is a rather costly procedure and, since it is performed for each test instance, the scalability of the temporally-aware classifiers becomes compromised (i.e., they do not handle large datasets efficiently). This scenario poses the following challenge: can the temporal effects be addressed in some entirely off-line setting (instead of a lazy learning setting), ultimately contributing to a scalable classification system without being negatively affected by varying data distributions?

In this work we intend to answer the posed question from a data engineering perspective. We propose a pre-processing step capable of moving all the overhead associated with handling the temporal issues to an off-line setting, through the use of what we call Cascaded Temporal Smoothing (CTS). The CTS consists of a controlled temporally-aware data oversampling strategy which aims at smoothing the observed temporal effects in the underlying data distribution. More specifically, it aims at reducing the influence of potentially misleading unstable information in the training set, when learning a classifier. Basically, for each class c, at each point in time p, CTS checks whether the number of documents is sufficient to ensure a good representativity for c in that point in time. If the number of samples is not enough (that is, smaller than a threshold m previously defined), CTS starts an oversampling cascaded process generating documents and incorporating them to the dataset—at point in time p—in order to improve the class' representativity while minimizing the observed variations in the data distribution. The main motivation behind CTS is that, as reported in [Salles 2011], a large presence of the temporal effects in the training data can significantly degrade the classification effectiveness, motivating us to devise a pre-processing step able to reduce these variations in the training data characteristics (by propagating documents through the time line) as a way to improve classification effectiveness. Our hope is that, with a higher quality training set (i.e., with the temporal effects smoothed out), a regular (traditional) classifier may be able to provide more accurate predictions.

Unlike previously proposed temporally-aware classifiers, which demand a lazy classification approach, the strategy proposed here handles the temporal effects entirely off-line. This means that, using CTS, the test phase runtime becomes independent on the training set size while maintaining the classifier robust to the temporal effects. Moreover, since the data is preprocessed before the actual classification, the time required to perform CTS is spent only once for any classifier used. Both aspects directly contribute to the scalability of the solution. Finally, this preprocessing step is independent of the classifier selected for the ADC, as it does not impose any constraints regarding which classifier should be used to make the predictions and does not involve any modifications to existing classifiers.[1] Although this data engineering approach may incur in some additional time due to this new pre-processing step, as our experimental evaluation will show, this is still worthwhile. The increase in training time is indeed very small when compared with the drastic reduction achieved in the test runtime. That is, CTS increases the classifier throughput in terms of the number of classified documents per time unit, preserving the classification effectiveness in face of the temporal effects. This makes the CTS a very effective and robust solution to ADC in face of large datasets with varying distributions.

We evaluate our proposal considering two real and large textual datasets, namely ACM-DL and MEDLINE. We evaluate the traditional Naïve Bayes classifier trained with the new training set produced by CTS contrasting it with the Temporally-Aware Naïve Bayes.[2] Our experimental evaluation shows that the use of CTS before learning the classifier achieved gains up to 5.00% in terms of MacroF_1 for the ACM-DL dataset when compared to a traditional classifier learned from the original training set, while being statistically tied with the temporally-aware classifier. Moreover, in the case of MEDLINE, using the training set augmented by CTS achieved gains to up to 5.13% in MacroF_1 when compared to a Naïve Bayes classifier learned from the original training set, and achieved a marginal (but statistically significant) gain of 2.75% when compared to the temporally-aware classifier. Furthermore, we show that our CTS approach exhibits scalability properties not presented by the lazy temporally-aware classifiers, guaranteeing its practical applicability in large classification problems.

The remainder of this work is organized as follows: Section 2 discusses some related work. Section 3 describes our proposed CTS algorithm, as well as a probabilistic analysis regarding its properties. In Section 4 we evaluate our approach and, finally, in Section 5 we conclude and discuss future work.

2. RELATED WORK

While ADC is a widely studied topic, the analysis of the impact caused by the temporal aspects has only started in the last decade. As stated before, a recent effort was made in order to characterize this problem and provide valuable insights to the development of temporally robust classifiers. In [Mourão et al. 2008], the authors present a qualitative analysis regarding the existence of three main temporal effects, which are, in fact, manifestations of the drifting patterns first analyzed by [Forman 2006]. Further effort was made by [Salles 2011] in order to quantitatively characterize these effects using a factorial experimental design. That work advances the qualitative analysis reported in [Mourão et al. 2008] by *quantifying* to what extent each temporal effect influences three real datasets and their

[1]Although not exploited in this work, our approach also allows us to explore other state-of-the-art algorithms such as SVM, whose any lazy temporally-aware version would be infeasible.
[2]This was the modified classifier that produced the best overall results in [Salles et al. 2010] for both explored datasets.

negative impact to the effectiveness of four widely used ADC algorithms. The main findings are that the presence of temporal effects in the data used to learn a classifier really do negatively impact the classification effectiveness, motivating us to develop techniques to smooth out such effects.

Proposed methods that attempt to *minimize* the impact of temporal effects in ADC can be categorized in two broad areas, namely, adaptive document classification and concept drift. Adaptive document classification [Cohen and Singer 1999] embodies a set of techniques to deal with changes in the underlying data distribution in order to improve the effectiveness of classifiers through incremental and efficient adaptation of the classification models. On the other hand, concept (or topic) drift [Tsymbal 2004] groups techniques in which the classifier is completely retrained according to some instance selection or weighting method. It comprises most of the recent efforts in dealing with temporal aspects and data streams with changing distributions, from a wide domain of applications like textual classification [Wang et al. 2010], recommendation systems based in collaborative filtering [Koren 2010] and others. A number of previous studies fall into this category. In [Klinkenberg and Joachims 2000] a sliding window with examples sufficiently "close" to the current target concept has its size automatically adjusted in order to minimize the estimated generalization error. The method proposed in [Žliobaitė 2009] builds the classification model using training instances which are close to the test in terms of both time and space. In [Klinkenberg 2004] different approaches are used such as adaptive time window on the training data and selection or weighting of representative training examples for model construction. [Widmer and Kubat 1996] describe a set of algorithms that react to concept drift in a flexible way and can take advantage of situations of recurring contexts. The main idea of these algorithms is to keep only a window of currently trusted examples and hypothesis, and store concept descriptions in order to reuse them if a previous context reappears. [Rocha et al. 2008] introduce the concept of *temporal context*, defined as a subset of the dataset that minimizes the impact of temporal effects in the performance of classifiers. These temporal contexts are used to sample the training examples for the classification process, discarding instances that are considered to be outside this context. An algorithm, named *Chronos*, to identify these contexts based on the stability of the terms in the training set is also proposed.

Instance selection approaches may be considered too rigid since they may miss valuable information laying outside of the window. This motivates the use of instance weighting approaches. Following this direction, in [Salles et al. 2010] the temporal information is incorporated to document classifiers based on the evolution of term-class relationship over time, which is captured by a metric of *dominance* [Rocha et al. 2008]. A *temporal weighting function* (TWF) is defined for a given dataset by means of a series of statistical tests aimed at determining the TWF's expression. A curve fitting procedure is then employed to determine its parameters. The discovered TWF is then incorporated to ADC algorithms, by means of two different strategies, namely, "TWF in documents" and "TWF in scores". The first strategy weights each training document by the TWF according to its temporal distance to the test document d'. The second strategy considers the "scores" produced by the traditional classifiers learned from a training set whose documents' class is transformed to a new derived class $\langle c, p\rangle$, where c is the actual document class and p denotes its creation point in time. The test document d' is then classified by such traditional classifier, which generates scores for each $\langle c, p\rangle$. The obtained scores for each derived class are aggregated through a weighted sum, where the weights correspond to the TWF considering the temporal distance between p and the creation time of d'. Both strategies have one common property: they follow a lazy classification approach, since the classification model is only defined after discovering the creation time of the test document.

A Temporal Inductive Transfer (TIX) method is proposed in [Forman 2006]. The TIX method follows a transfer learning paradigm [Pan and Yang 2010] and does not assume that the data distributions remain the same as time goes by. It considers the data created around the same time as the test document as the target domain, whereas the remaining documents compose the source domain. The TIX transfers knowledge from the source domain to the target domain in order to improve classification effectiveness when dealing with documents from the target domain. Such strategy works as follows: given a test instance, one initial model is created at the first point in time (p_0). This model is applied for predicting the test document and the resulting prediction is appended to both the test and training data, in the form of a new binary feature. This augmented test document is then classified by the next model (p_1), learned from the training set with augmented feature space, and its output is also appended to the documents. This cascaded procedure proceeds until the test document is classified with the model associated to the target domain (p_n). In [Zhao and Hoi 2010], the authors also propose a classification strategy based on the transfer learning paradigm, however considering an online learning [Crammer et al. 2006] task. In this case, transfer learning is used to combine one previously built model with an online learning model, in order to take advantage of data whose distribution may not be the same as the distribution currently observed. The proposed Online Transfer Learning (OTL) algorithm uses this combination in order to learn from data that potentially comes from a distinct distribution (for example, due to concept drift) increasing both the initial performance of the online classifier and its asymptotic effectiveness.

All the previously described solutions incur in overhead at the test phase of the classification task. Sliding windows depends on the test data to define its size: fast variations on the test set reduces the windows size, whereas gradual variations increase its size. For each window configuration, there exists an overhead of determining if its size is (near-)optimal, ultimately increasing the test phase runtime. Instance weighting strategies are naturally lazy since the model becomes dependent on the instance weights which, by their own, depend on the temporal distance between test and training samples. The TIX method propagates the classifiers' predictions regarding each point in time prior to the test's creation time (using binary features), thus incurring in overhead at test phase. The OTL algorithm, on the other hand, moves all calculations to the test phase (online learning). Differently from the previously described methods, the strategy proposed in this work eliminates the overhead of dealing with varying data distributions in the test phase of the classification task (i.e., through a lazy setting), by means of a classifier independent pre-processing step in which the underlying temporal effects in the training set are smoothed out. As we shall see, our solution leads to a scalable temporally robust classification system.

3. CASCADED TEMPORAL SMOOTHING

In this section, we describe our approach to handle the temporal effects without incurring in any overhead at the test phase of the classification task, namely the Cascaded Temporal Smoothing (hereafter, CTS). For the following discussion, let $\mathbb{D}_{train} = \{d_i\}$ be the training set composed by documents $d_i = \langle x_i, c_i, p_i \rangle$ such that x_i denotes the vector representation (bag of words) of d_i, c_i represents its assigned class and p_i represents its creation time point. Moreover, let $\mathbb{C}$ and $\mathbb{P}$ be, respectively, the set of classes and points in time observed in $\mathbb{D}_{train}$. Finally, we also assume that the set of time points observed in the test set is $\mathbb{P}' \subseteq \mathbb{P}$ and that during test phase, the test data is given to the classifier without any temporal ordering.[3] The general idea is that, given a class $c \in \mathbb{C}$, CTS tries to smooth out the differences observed in the characteristics of the evolving documents in this class as time goes by. This is accomplished by an iterative process in which, for each target point in time $p \in \mathbb{P}$, the CTS propagates documents from other points in time nearby p in which c is representative enough (i.e., it has enough documents from class c). As we shall see, this data propagation causes the characteristics of c to be less divergent over time, reducing the temporal effects.

3.1 Description of the Algorithm

Before detailing the CTS algorithm, we will describe some auxiliary functions extensively used in the CTS algorithm. These functions are listed in Algorithm 1. The first two auxiliary functions

[3]While for stream mining it is commonly assumed to expect test data with increasing temporal ordering, for static scenarios as the classification of documents from the Web or digital libraries, this is quite unrealistic. In fact, forgetting mechanisms adopted to handle drifting data streams may not be the best choice, since the notion of "old" instances is not applicable.

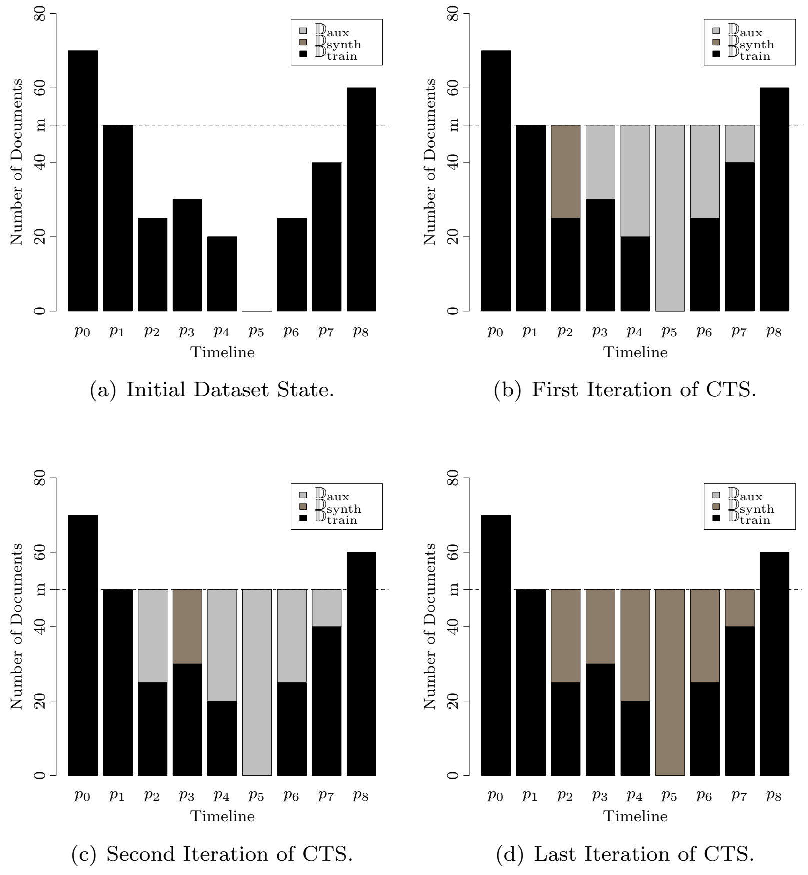
are the FindLowerBound($c \in \mathbb{C}, p \in \mathbb{P}, m$) and the FindUpperBound($c \in \mathbb{C}, p \in \mathbb{P}, m$), where m is some integer. Let $\mathbb{D}_{c,p}$ be the set of documents assigned to class c whose creation point in time is p. FindLowerBound ($c \in \mathbb{C}, p \in \mathbb{P}, m$) returns the maximum point in time $p' < p$ such that $|\mathbb{D}_{c,p'}| \geq m$. If there is not enough documents of class c in all points in time $p' < p$ (i.e., $|\mathbb{D}_{c,p'}| < m$ for all $p' < p$) then the lower bound is undefined and the function returns NIL. Analogously, FindUpperBound($c \in \mathbb{C}, p \in \mathbb{P}, m$) returns the minimum point in time $p' > p$ such that $|\mathbb{D}_{c,p'}| \geq m$. Again, if there is not enough documents of class c in all points in time $p' > p$ ($|\mathbb{D}_{c,p'}| < m$ for all $p' > p$) then the upper bound is undefined and the function returns NIL. The third auxiliary function is GenerateSamples($p_{source} \in \mathbb{P}, p_{target} \in \mathbb{P}, c \in \mathbb{C}, \mathbb{D}, m$). Based on a set of documents $\mathbb{D}$ created in time point p_{source}, this function shall generate representative documents of class c for the target point in time p_{target}. The perhaps most straightforward sample generation strategy is to simply replicate documents (i.e., duplicate). This is the strategy currently used by our CTS implementation. In fact, our solution is modular enough to allow one to easily adopt more sophisticated strategies, such as SMOTE [Chawla et al. 2002] or Probabilistic Generation Model [Bermejo et al. 2011].[4] As listed in Algorithm 1, this function randomly selects documents from $\mathbb{D}$ assigned to class c and created at point in time p_{source}, replicates it and reassigns its creation point in time to p_{target}. This is performed until m new documents are generated.

Algorithm 1 Auxiliary Functions for CTS

```
 1: function FindLowerBound($c \in \mathbb{C}$, $p \in \mathbb{P}$, $m$)
 2:     $\mathbb{P}_{cand} \leftarrow \{p' \in \mathbb{P} \mid |\mathbb{D}_{c,p'}| \geq m \ and \ p' < p\}$        ▷ Candidate points in time
 3:     if $\mathbb{P}_{cand} = \emptyset$ then
 4:         return $NIL$                                                          ▷ Undefined lower bound
 5:     else
 6:         return $\max_p \mathbb{P}_{cand}$
 7:     end if
 8: end function
 9: function FindUpperBound($c \in \mathbb{C}$, $p \in \mathbb{P}$, $m$)
10:     $\mathbb{P}_{cand} \leftarrow \{p' \in \mathbb{P} \mid |\mathbb{D}_{c,p'}| \geq m \ and \ p' > p\}$        ▷ Candidate points in time
11:     if $\mathbb{P}_{cand} = \emptyset$ then
12:         return $NIL$                                                          ▷ Undefined upper bound
13:     else
14:         return $\min_p \mathbb{P}_{cand}$
15:     end if
16: end function
17: function GenerateSamples($p_{source} \in \mathbb{P}$, $p_{target} \in \mathbb{P}$, $c \in \mathbb{C}$, $\mathbb{D}$, $m$)
18:     $\mathbb{D}'_{result} \leftarrow \emptyset$
19:     for $i = 1$ to $m$ do
20:         $d \leftarrow$ RandomSelection($p_{source}, c, \mathbb{D}$)
21:         $d.p \leftarrow p_{target}$
22:         $\mathbb{D}'_{result} \leftarrow \mathbb{D}'_{result} \cup \{d\}$
23:     end for
24:     return $\mathbb{D}'_{result}$
25: end function
```

With the auxiliary functions described above, we are able to detail the CTS algorithm, listed in Algorithm 2. For each class $c \in \mathbb{C}$, the CTS iterates over each point in time $p \in \mathbb{P}$, searching for some point in time such that $|\mathbb{D}_{c,p}| < m$. If so, we say that $\mathbb{D}_{c,p}$ does not have enough documents and the CTS attempts to fill in the remaining ($m - |\mathbb{D}_{c,p}|$) documents, through cascading. CTS will thus propagate documents from points in time nearby p given that these points in time have enough documents from class c (which we try to guarantee through the cascading process, as we shall see). In order to do this, first we determine the lower and upper bounds, l_b and u_b, respectively, that are points in time in which class c has enough documents. These points define an open interval regarding a continuous period of time during which $|\mathbb{D}_{c,p}| < m$. The set of documents created at both points in time will serve as the source of documents to be cascaded through the timeline. Let $p_i \rightarrow p_j$ denote the replication of documents from the source point in time p_i to the target point in time p_j. Also, consider that $p_i \rightarrow p_j \rightarrow p_k$ denotes the replication of documents from p_i to p_j, followed by

[4] We are currently investigating this matter.



(a) Initial Dataset State. (b) First Iteration of CTS. (c) Second Iteration of CTS. (d) Last Iteration of CTS.

Fig. 1. Example: Effect of Cascading Temporal Smoothing in the Distribution of Class c Over Time.

the replication of the resulting augmented set of documents of p_j to p_k. The cascading process works in two stages: *(i)* a forward cascading, where CTS propagates documents from l_b to $p-1$, that is, $l_b \rightarrow l_b+1 \rightarrow l_b+2 \rightarrow \cdots \rightarrow p-1$; and *(ii)* a backward cascading, where CTS propagates documents from u_b to $p+1$, that is, $u_b \rightarrow u_b-1 \rightarrow u_b-2 \rightarrow \cdots \rightarrow p+1$. After that, the neighbors of p have finally been fulfilled with enough documents to be cascaded to p and CTS performs the actual oversampling, where the synthetic samples from these neighbors are replicated to the target point in time p. As our goal is to perform the oversampling to point p, in the two previously described stages of the cascading process—*(i)* and *(ii)*—the CTS uses an auxiliary synthetic dataset $\mathbb{D}_{aux}$ that can be discarded at the end. There are three cases to be considered by the CTS:

(1) If p is the first point in time of the dataset (p_0), then the CTS performs only a backward cascading $p+1 \rightarrow p$ of $(m - |\mathbb{D}_{c,p}|)$ documents;
(2) If p is the last point in time of the dataset (p_f), then the CTS performs only a forward cascading $p-1 \rightarrow p$ of $(m - |\mathbb{D}_{c,p}|)$ documents;
(3) If p is neither the first nor the last point in time of the dataset, then the CTS performs both a backward and a forward cascading, each with $\frac{(m-|\mathbb{D}_{c,p}|)}{2}$ documents.

The generated documents are then stored in a set of synthetic documents $\mathbb{D}_{synth}$, which will be merged to the training set at the end of the process.

In order to illustrate how CTS works, let us consider the example shown in Figure 1, regarding a hypothetical class c. Figure 1(a) shows the initial distribution of class c over time. The CTS algorithm starts with $p = p_0$, and iterates through all points in time till the last one, when $p = p_8$. For each point in time p, CTS evaluates $|\mathbb{D}_{c,p}|$ in order to decide if it shall generate new synthetic documents for class c. This test is done by line 6 and, for our example, it will be set to true when $p = p_2$. Let us consider this iteration. The lower and upper bounds, computed by lines 8 and 9, respectively, will be $l_b = p_1$ and $u_b = p_8$.[5] The algorithm will bypass the lines 10 and 19 and will proceed to the auxiliary synthetic data generation. Since $l_b = p_2 - 1 = p_1$, CTS will not perform the forward cascading (lines 20 to 24), but it will proceed with the backward cascading (lines 25 to 29): $p_8 \rightarrow p_7 \rightarrow p_6 \rightarrow \cdots \rightarrow p_3$. After that, the neighbors of p_2 have finally been fulfilled with enough documents to be cascaded to it. The actual oversampling is performed from line 30 to 39, where the synthetic samples from the neighborhood of p_2 (p_1 and p_3) are replicated to the target point in time p_2. The distribution of documents of class c after this iteration is illustrated in Figure 1(b). The next iteration sets $p = p_3$, which also does not have enough documents. As before, the algorithm determines the lower and upper bounds, which are $l_b = p_1$ and $u_b = p_8$. The documents are thus cascaded both forwards and backwards, considering the auxiliary dataset $\mathbb{D}_{aux}$: $p_1 \rightarrow p_2$ and $p_8 \rightarrow p_7 \rightarrow p_6 \rightarrow \cdots \rightarrow p_4$. Now, the neighbors of p_3 are properly fulfilled, and the actual cascading is performed. After this iteration, the augmented dataset looks like Figure 1(c). The algorithm then proceeds to the remaining points in time p_4 to p_8, until reaching the final class distribution depicted in Figure 1(d).

Notice that CTS does not aim at balancing the dataset. In fact, CTS promotes a reduction in the observed variations of the target class distribution over time. This *may* reduce the overall class imbalance ratio, since the minority classes may not have more than m documents in several points in time. However, this is not true for all cases. For example, consider that c is the majority class. Also, consider that its documents were created at just a few points in time (that is, in several points in time there are less than m documents assigned to class c). In this case, the CTS algorithm would increase the majority class and, consequently, increase the class imbalance ratio. In fact, a perfect balancing is achieved when $m = \max(|\mathbb{D}_{c,p}|)$. However, as we shall see in Section 4, this is may not be the best choice for m.

3.2 CTS Analysis

In this section, we consider some properties of the CTS algorithm. More specifically, we shall provide an analysis regarding the behavior of the cascading process performed by CTS and the effect of its parameter m. As we describe below, when cascading $p_i \rightarrow p_j$, the greater the temporal distance between them, the smaller is the probability of transferring documents from p_i to p_j. This means that CTS will privilege nearby points in time during the cascading process, smoothing out the observed variations in the datasets' characteristics between these points in time while preventing a random behavior (that is, preventing the transference of documents irrespective to their time points). Moreover, as we shall see, smaller m values limit CTS capability of smoothing the variations. On the other hand, when m increases, at the limit, CTS tends to converge into a random oversampling strategy. A good trade-off between CTS smoothing capability and its randomness avoidance can be easily achieved by means of a simple greedy search algorithm, due to this unimodal[6] behavior regarding the m parameter. So, let c be an arbitrary class and m be CTS threshold. We start our analysis by considering the probability of transferring original data from a source point in time p_i to a target point in time p_j, denoted by $P(p_i, p_j)$. Recall that, from the definition of lower and upper bounds, for all points in time $p' \leq l_b$ and $p'' \geq u_b$, $|\mathbb{D}_{c,p'}| \geq m$ and $|\mathbb{D}_{c,p''}| \geq m$. For these cases, both sets will not be affected by the CTS, since they are composed by enough documents. Thus, in the following we shall turn our attention to the points in time $p_i \geq l_b$ and $p_j \leq u_b$. Our two base cases to be considered are $P(l_b, l_{b+1}) = 1$

[5]If l_b is undefined, the CTS generates documents for the first point in time p_0, sets $l_b = p_0$ and proceeds as usual. If u_b is undefined, CTS generates documents for the last point in time p_8, sets $u_b = p_8$ and proceeds as usual. Both situations are handled in lines 10 through 19 of Algorithm 2.

[6]A function $f(x)$ is unimodal if it is monotonically increasing for $x \leq n$ and monotonically decreasing for $x \geq n$. In that case, the maximum value of $f(x)$ is $f(n)$ and this is the global maximum.

Algorithm 2 Cascaded Temporal Smoothing.

```
function CTS($\mathbb{D}_{train}$, $m$)
    $\mathbb{D}_{synth} \leftarrow \emptyset$
    for $c \in \mathbb{C}$ do
        for $p \in \mathbb{P}$ do
            $\mathbb{D}_c, p \leftarrow \{d \in \mathbb{D}_{train} | d.c = c \ and \ d.p = p\}$
            if $|\mathbb{D}_{c,p}| < m$ then                                    ▷ Not enough documents in $\mathbb{D}_{c,p}$
                $\mathbb{D}_{aux} \leftarrow \mathbb{D}_{train}$                                   ▷ Auxiliary data
                $l_b \leftarrow$ FindLowerBound($c, p, m$)
                $u_b \leftarrow$ FindUpperBound($c, p, m$)
                if $l_b = NIL$ then                          ▷ Not enough documents between $p_0$ and $p$
                    $l_b \leftarrow$ FindUpperBound($c, p, 1$)              ▷ Find next $p'$ s.t. $\mathbb{D}_{c,p'} \neq \emptyset$
                    $\mathbb{D}_{aux} \leftarrow \mathbb{D}_{aux} \cup$ GenerateSamples($l_b, p_0, c, \mathbb{D}_{aux}, m$)
                    $l_b \leftarrow p_0$                                          ▷ Lower bound now defined
                end if
                if $u_b = NIL$ then                          ▷ Not enough documents between $p$ and $p_f$
                    $u_b \leftarrow$ FindLowerBound($c, p, 1$)          ▷ Find previous $p'$ s.t. $\mathbb{D}_{c,p'} \neq \emptyset$
                    $\mathbb{D}_{aux} \leftarrow \mathbb{D}_{aux} \cup$ GenerateSamples($u_b, p_f, c, \mathbb{D}_{aux}, m$)
                    $u_b \leftarrow p_f$                                          ▷ Upper bound now defined
                end if
                while $l_b + 1 < p$ do                           ▷ Forward cascading: from past to $p$
                    $n \leftarrow m - |\mathbb{D}_{c,l_b+1}|$
                    $\mathbb{D}_{aux} \leftarrow \mathbb{D}_{aux} \cup$ GenerateSamples($l_b, l_b + 1, c, \mathbb{D}_{aux}, n$)
                    $l_b \leftarrow l_b + 1$
                end while
                while $u_b - 1 > p$ do                        ▷ Backward cascading: from future to $p$
                    $n \leftarrow m - |\mathbb{D}_{c,u_b-1}|$
                    $\mathbb{D}_{aux} \leftarrow \mathbb{D}_{aux} \cup$ GenerateSamples($u_b, u_b - 1, c, \mathbb{D}_{aux}, n$)
                    $u_b \leftarrow u_b - 1$
                end while
                ▷*Actual training set oversampling:*
                $n \leftarrow m - |\mathbb{D}_{c,p}|$
                if $p = p_0$ then                                          ▷ Backward Cascading only
                    $\mathbb{D}_{synth} \leftarrow \mathbb{D}_{synth} \cup$ GenerateSamples($p + 1, p, c, \mathbb{D}_{aux}, n$)
                else if $p = p_f$ then                                     ▷ Forward Cascading only
                    $\mathbb{D}_{synth} \leftarrow \mathbb{D}_{synth} \cup$ GenerateSamples($p - 1, p, c, \mathbb{D}_{aux}, n$)
                else                                          ▷ Both backward and forward Cascading
                    $\mathbb{D}_{synth} \leftarrow \mathbb{D}_{synth} \cup$ GenerateSamples($p + 1, p, c, \mathbb{D}_{aux}, \frac{n}{2}$)
                    $\mathbb{D}_{synth} \leftarrow \mathbb{D}_{synth} \cup$ GenerateSamples($p - 1, p, c, \mathbb{D}_{aux}, \frac{n}{2}$)
                end if
            end if
        end for
    end for
    $\mathbb{D}_{train} \leftarrow \mathbb{D}_{train} \cup \mathbb{D}_{synth}$
end function
```

and $P(u_b, u_{b-1}) = 1$, $\forall p_j \in \mathbb{P}$. This follows from the definition of lower and upper bounds, where $|\mathbb{D}_{c,l_b}| \geq m$ and $|\mathbb{D}_{c,u_b}| \geq m$ and, consequently, all the documents transferred to l_{b+1} and u_{b-1} come from l_b and u_b, respectively. Now, consider the probability $P(l_b, l_{b+2})$, that is, the probability of transferring documents from l_b to l_{b+2}. In this case, after the first cascading $l_b \rightarrow l_{b+1}$, the point in time l_{b+1} will be composed by m documents such that $(m - |\mathbb{D}_{c,l_{b+1}}|)$ were cascaded from l_b.[7] Thus, the probability of documents from l_b to be cascaded to l_{b+2} is given by:

$$P(l_b, l_{b+2}) = \frac{m - |\mathbb{D}_{c,l_{b+1}}|}{m} \tag{1}$$

The probability of documents from l_b to be transferred to l_{b+3}—$P(l_b, l_{b+3})$—depends on the $l_b \rightarrow l_{b+1}$, $l_{b+1} \rightarrow l_{b+2}$ and $l_{b+1} \rightarrow l_{b+2}$ cascades, since documents from l_b must first be cascaded to l_{b+1}, and then to l_{b+2} and, finally, to l_{b+3}. Thus, $P(l_b, l_{b+3}) = P(l_b, l_{b+1}) \cdot P(l_{b+1}, l_{b+2}) \cdot P(l_{b+2}, l_{b+3})$. Substituting the first term by 1 and the second term by Equation 1, we get:

$$P(l_b, l_{b+3}) = \frac{m - |\mathbb{D}_{c,l_{b+1}}|}{m} \cdot P(l_{b+2}, l_{b+3}). \tag{2}$$

[7]Notice that, by the lower and upper bounds definition, $(m - |\mathbb{D}_{c,l_{b+1}}|) > 0$.

Recall that after the $l_{b+1} \rightarrow l_{b+2}$ cascading, the point in time l_{b+2} is composed by m documents such that $(m - |\mathbb{D}_{c,l_{b+2}}|)$ were cascaded from l_{b+1}. Thus, we are able to substitute the third term $P(l_{b+2}, l_{b+3})$ of Equation 2, based on the same rationale employed to derive Equation 1:

$$P(l_b, l_{b+3}) = \frac{m - |\mathbb{D}_{c,l_{b+1}}|}{m} \cdot \frac{m - |\mathbb{D}_{c,l_{b+2}}|}{m}. \tag{3}$$

It should be straightforward to verify that

$$P(l_b, l_{b+4}) = \frac{m - |\mathbb{D}_{c,l_{b+1}}|}{m} \cdot \frac{m - |\mathbb{D}_{c,l_{b+2}}|}{m} \cdot \frac{m - |\mathbb{D}_{c,l_{b+3}}|}{m}.$$

The above derivations give us the necessary tools to derive a more general formulation $P(l_b, p), \forall p \in \mathbb{P}$, which gives us the probability of transferring data from the l_b to a point in time p (through the forward cascading):[8]

$$P(l_b, p) = \frac{\prod_{p'=l_{b+1}}^{p} (m - |\mathbb{D}_{c,p'}|)}{m^{p-l_b-1}}. \tag{4}$$

Analogously to the derivation of $P(l_b, p)$ regarding the forward cascading, it is easy to verify that the probability $P(u_b, p)$ regarding the backward cascading is given by:

$$P(u_b, p) = \frac{\prod_{p'=l_{u-1}}^{p} (m - |\mathbb{D}_{c,p'}|)}{m^{u_b-p-1}}. \tag{5}$$

The last two probabilities give us some clues regarding the behavior of CTS. First, considering the cascading $l_b \rightarrow p$ (or $u_b \rightarrow p$), the greater the temporal distance between them, the smaller is the probability of transferring documents from l_b (or u_b) to p. Consequently, the greater is the probability of transferring synthetic data from nearby points in time. This is a key aspect to guarantee the controlled behavior of the CTS oversampling process. If, despite of prioritizing nearby points in time, CTS associates an uniform probability of propagating documents from any point in time, than it would be equivalent to a random oversampling process. Furthermore, privileging nearby time points also avoids the propagation of very abrupt changes in the dataset characteristics, consequently turning the cascading process smoother. Second, greater m implies a higher probability of transferring data from distant point in times to p. As we shall elaborate soon, asymptotically, this converges to a simple random oversampling, irrespective of the documents timeliness.

Equations 4 and 5 enable us to determine the probability of transferring documents from the boundary points in time (l_b and u_b) to the target point p in the actual oversampling step of CTS (lines 30 to 39). As mentioned before, there are three cases to consider: *(i)* if $p = p_0$, just the backward cascading occurs, with probability given by Equation 5; *(ii)* if $p = p_f$, just the forward cascading occurs, with probability given by Equation 4; finally, if $p \neq p_0$ and $p \neq p_f$, then both forward and backward cascading take place (see lines 37 and 38 of Algorithm 2), with probabilities given by $P(l_b, p)$ and $P(u_b, p)$, respectively. If $m \gg |\mathbb{D}_{c,p}|$ then, asymptotically, $P(l_b, p)$ will converge to $\frac{\prod_{l_{b+1}}^{p} m}{m^{p-l_b-1}} = 1$, as will $P(u_b, p)$. It means that, in all cases, in the limit of m, all points in time will share similar probabilities of cascading documents to the target point p, which corresponds to a random oversampling process irrespective to the documents creation time. On the other hand, notice that, if $m \leq \min(|\mathbb{D}_{c,p}|)$ than, by construction, CTS will not cascade any document through the timeline. As we shall see, this unimodal behavior can be exploited to efficiently calibrate the CTS. Since c is arbitrary, then the above discussion holds for all classes of the dataset.

4. EXPERIMENTAL EVALUATION

In this section, we describe our experimental setup, namely, the explored datasets and algorithms, as well as report and analyze our achieved results.

[8]We removed the probability associated to the cascading $l_b \rightarrow l_{b+1}$ since, by definition, it is 1.

4.1 Testbed

The two reference datasets considered in our study consist of sets of large textual documents, each one assigned to a single class (a single label problem). The considered datasets[9] are:

(1) **ACM-DL:** This dataset [Couto et al. 2006] is a subset of the ACM Digital Library with $24,897$ documents containing articles related to Computer Science created between 1980 and 2002. We considered the first level of the taxonomy in the ACM-DL Computing Classification System (CCS), including 11 classes, which remained the same throughout the period of analysis.
(2) **MEDLINE:** This dataset [Mourão et al. 2008] is a subset of the MedLine dataset, with $861,454$ documents classified into 7 distinct classes related to Medicine, and created between the years of 1970 and 1985.

Recall that each document d_i of these datasets are represented as triples $d_i = \langle x_i, c_i, p_i \rangle$ such that x_i denotes the vector representation of d_i, c_i represents its assigned class and p_i represents its creation time. Since the explored datasets refer to scientific articles, a natural time granularity for p_i is the year, since scientific conferences are usually annual. Thus, p_i denotes the year d_i was created. In order to evaluate the CTS algorithm, we considered four classification tasks, as detailed below:

Traditional. This classification task is characterized by learning a classification model based on the original training set $\mathbb{D}_{train}$ and then using it to classify $\mathbb{D}_{test}$. We adopted the Naïve Bayes, since its temporally-aware version was the top performer in the explored datasets as shown in [Salles et al. 2010]. Such classifier is a probabilistic learning method that aims at inferring a model for each class by assigning to a test document d' the class associated with the most probable model that would have generated it (i.e., the class with maximum a posteriori probability). Here, we adopt the Multinomial Naïve Bayes approach, since it is widely used for probabilistic text classification. The posterior class probabilities $P(d'|c)$ are defined as:

$$P(d'|c) = \eta \times P(c) \times \prod_{t \in d'} P(t|c),$$

where η denotes a normalizing factor, $P(c)$ is the class prior probability and $P(t|c)$ denotes the conditional probability of observing t having already observed c.As said, such classifier assigns to a test example d' the class c with the highest a posteriori probability $P(d'|c)$. Both $P(c)$ and $P(t|c)$ are estimated from the observed frequency of occurrences of documents and terms in the classes, irrespective of the documents timeliness. Furthermore, these probabilities are approximated by a maximum likelihood method, depending solely on the training set. We expect this to be the fastest classification scenario (low runtime for both the training and testing phases), but the one most influenced by the temporal effects, as there are no explicit treatments for them.

Temporally-Aware. This classification task is characterized by applying a Temporally-Aware classifier [Salles et al. 2010], learned from the original training set $\mathbb{D}_{train}$ and used to classify $\mathbb{D}_{test}$. For this purpose, we chose the Temporally-Aware Naïve Bayes classifier, with the temporal weighting function (see Section 2) applied in documents. This was the best performing temporally-aware classifier for the two adopted datasets, as reported in [Salles et al. 2010]. In this strategy, a *temporal weighting function* (TWF) is used to control the influence of training documents used for probability estimation (namely the relative frequencies of occurrences of documents and terms for each class, as time goes by) according to the temporal distance between training and test data. The TWF reflects the observed evolution of the term-class relationships over time, and its expression is determined by means of a series of statistical tests. Then, a curve fitting procedure is employed to determine its parameters. A detailed description about the TWF can be found in [Salles et al. 2010]. The Temporally-Aware Naïve Bayes classifier learns the probability of assigning a test document d' to class c as follows:

$$P(d'|c) = \eta \cdot \frac{\sum_p (N_{cp} \cdot TWF(\delta))}{\sum_p (N_p \cdot TWF(\delta))} \cdot \prod_{t \in d'} \frac{\sum_p (f_{tcp} \cdot TWF(\delta))}{\sum\limits_p \sum\limits_{t' \in \mathbb{V}} (f_{t'cp} \cdot TWF(\delta))},$$

where $\mathbb{D}$ denotes the training set, η denotes a normalizing factor, N_{cp} is the number of training documents of $\mathbb{D}$ assigned to class c and created at the time point p, N_p is the number of training documents created at the time point p, f_{tcp} stands for the frequency of occurrence of term t in training documents of class c that were

[9]These datasets are available upon request. We are currently building a web site to make them publicly avaiable.

created on time point p, TWF denotes the temporal weighting function and, finally, δ denotes the temporal distance between p and the creation time of d' (a.k.a., the reference point in time p_r). Notice that, in this case, both the a priori class probabilities and the term conditionals depend on the temporal distance between training documents and the test. This obligates the classifier to postpone the estimation of these probabilities until it receives the test document, thus characterizing a lazy classifier. We expect this scenario to have the higher test runtime, since it must iterate over $p \in \mathbb{P}$, considering its temporal distance to the reference point p_r, in order to properly estimate both the class a priori and term conditional probabilities. We consider it as our upper bound in terms of handling the temporal effects.

CTS. This classification task involves learning a probabilistic model for the traditional Naïve Bayes algorithm, based on the augmented training set $\mathbb{D}_{train} \cup \mathbb{D}_{synth}$ generated by CTS. That is, we first apply the CTS algorithm to the original training set, considering a threshold m (which ultimately determines $|\mathbb{D}_{synth}|$), and then use the new augmented training set to learn a traditional Naïve Bayes classifier.

Random. This classification task serves the purpose of isolating the improvements made by the CTS algorithm from the random oversampling effect, and provide evidence about the analysis done in Section 3.2 regarding the influence of m in CTS. In this case, the training set $\mathbb{D}_{train}$ is randomly oversampled with exactly the same number os documents generated by the CTS approach. In this case, we do not perform cascading (that is, the random oversampling is performed irrespective to the timeline). The augmented training set have the same number of documents as the training set of "CTS" task.

4.2 Results

Recall that our goal is twofold: *(i)* to provide an effective document classification, robust to the temporal dynamics; *(ii)* which, at the same time, scales with the test set size. Hence, our experimental evaluation will be held considering two dimensions: classification effectiveness and scalability. As we shall see, the CTS was able to significantly improve classification effectiveness, while being scalable enough to handle large volumes of data. Furthermore, for all the described tasks, the test data is presented to the classifier without any temporal ordering, during the test phase. The following experiments were performed in a Intel® Core^{TM}$i7$ CPU ($2.93GHz$) with $8GB$ of RAM. We start by analyzing the classification effectiveness.

4.2.1 *Effectiveness Analysis.*

In order to evaluate the impact that CTS has in the classification effectiveness, we compare the four tasks previously described applied to both adopted datasets (ACM-DL and MEDLINE). For comparison we use a standard information retrieval measure commonly adopted for multi-class classification problems, namely, the macro averaged F_1 (MacroF_1) measure. The MacroF_1 is given averaging the F_1 measure (i.e., the harmonic mean of precision and recall) obtained for each individual class. All experiments were performed using a 10-fold cross-validation [Kohavi 1995] in order to assess the statistical validity of the obtained results and their 99% confidence intervals for the mean. At each iteration of the cross validation process, another 3-fold cross-validation over the *training* set was used to calibrate the m parameter. We searched for values of m in the interval from 10 to $1,000$ (with steps of 10) and from 100 to $10,000$ (with steps of 100), for the ACM-DL and MEDLINE datasets, respectively. These intervals were selected according to the dataset size. We also explored the discussed unimodal behavior regarding the m value (see Section 3.2) by greedily stopping the search when observing a decrease in MacroF_1. Once the best value was determined, it was fixed and used to finally augment the training set. The augmented training set was then used to learn a classification model to classify the test set. The m value chosen was also used by the "Random" task in order to enable us to isolate the cascading effect, as we shall see soon. The results obtained for the ACM-DL and MEDLINE datasets are reported in Table I. In that table, we reported the time spent by the train and test phases, along with the obtained MacroF_1, for each described task. The symbol following the MacroF_1 values refers to the statistical significance test, assessed by a paired two-sided t-test with 99% confidence level. ▲ denotes a statistically significant improvement over the "Traditional" task and ● denotes a statistical tie. The training phase runtime of the "CTS" task covers the time spent both to tune m (through cross-validation) and to learn a classifier. Finally, as the temporally-aware classifier is inherently lazy, we just report the runtime associated with the test phase.

Considering the results obtained for the ACM-DL dataset (see Table I), we can observe that the temporally-aware classifier outperforms its traditional counterpart (with a gain of 6.55%). CTS also outperformed the traditional classifier (with a gain of 5.00% over the traditional classifier), and achieved results comparable to those obtained by the temporally-aware classifier. One question remains to be answered: *was this improvement due to the temporally-aware data cascading or was a result of a purely random oversampling effect when*

Table I. Effectiveness Analisys: Obtained Results.

Dataset	**ACM-DL**			**MEDLINE**		
Strategy	Runtime (seconds)		Macro-F_1	Runtime (seconds)		Macro-F_1
	Train	Test		Train	Test	
Traditional	3.93 ± 0.01	0.30 ± 0.01	57.27 ± 0.50	195.60 ± 0.04	18.52 ± 0.06	64.72 ± 0.03
Random	4.11 ± 0.09	0.50 ± 0.02	57.13 ± 0.47 ●	197.60 ± 0.06	19.71 ± 0.07	63.75 ± 0.04 ●
CTS	19.06 ± 0.17	0.30 ± 0.01	60.13 ± 0.90 ▲	431.90 ± 1.14	18.57 ± 0.06	68.04 ± 0.04 ▲
Temporally-Aware	–	9.53 ± 0.14	61.02 ± 0.98 ▲	–	294.33 ± 0.10	66.22 ± 0.04 ▲

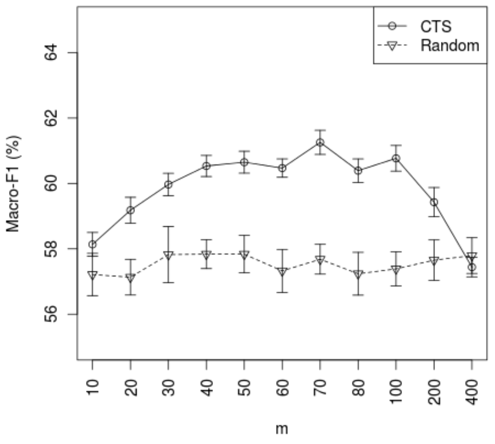
cascading ? In other words, we must isolate the obtained improvements and determine if it was a result of considering the documents' timeliness when propagating them, or if the same improvements should be achieved by randomly oversampling the dataset. This question is answered by considering the "Random" classification task. As it can be observed, CTS performs better than a simple random oversampling strategy (with a gain of 5.25% over the "Random" task), while the "Random" task was no better than the traditional classifier. Thus, the improvements achieved by CTS can in fact be explained by the smoothing of the temporal effects. However, this improvement comes with an extra cost: one needs to properly calibrate the CTS parameter and to execute the CTS, which may impose an additional runtime to the training phase. In fact, as it can be observed from Table I, the training phase runtime of "CTS" is greater than the "Traditional" training time. Such runtime is also greater than the time spent by the temporally-aware classifier. However, we think the benefits obtained by the CTS are worth, for two reasons. First, such pre-process step is executed only once, before learning and deploying the classification system. Second, unlike the CTS approach, the lazy temporally-aware classifier has a critical scalability issue, as we shall see in Section 4.2.2.

Turning our attention to the MEDLINE dataset, in Table I we can observe that the temporally-aware classifier outperformed the traditional classifier in terms of MacroF_1. Such result is also in conformity with those reported in [Salles et al. 2010]. However, the key aspect here is the behavior of CTS. In fact, CTS improved the classification effectiveness in terms of MacroF_1, with a gain of 5.13% over the "Traditional" task and a gain of 6.73% over the "Random" task. Furthermore, it was statistically superior to the temporally-aware classifier, with a marginal gain of 2.75% in MacroF_1. Considering that the CTS performed better than the "Random" task and was competitive to the temporally-aware classifier, it means that CTS was indeed able to tackle the temporal effects. Again, this improvement comes with a cost of an additional step at training phase. But, as we have already stated, such additional step is performed only once, before the classifier deployment. Furthermore, as we will discuss next, unlike CTS, the temporally-aware classifier does not scale well with the test set.

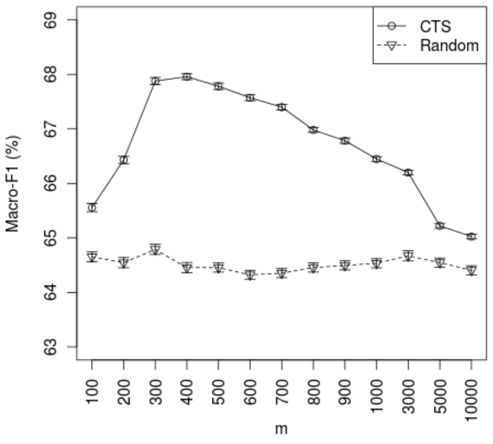
Finally, let us consider the behavior of CTS with regards to the m parameter. In order to have a better understanding about how the m values influence the classification effectiveness (in terms of the discussed unimodal behavior of CTS w.r.t. m), we explicitly varied such parameter and evaluate the obtained MacroF_1, for both the "CTS" and the "Random" task. Such evaluation is reported in Figure 2 and serves the unique purpose of corroborating the probabilistic analysis done in Section 3.2. As we can see for both datasets, for smaller values of m the capability of CTS to smooth the temporal variations is smaller, as reflected by the lower MacroF_1. On the other hand, as m increases the MacroF_1 also increases. But such improvement occurs up to a certain level at which the CTS performance converges to the performance of the "Random" task. This is in consonance with the probabilistic analysis presented in Section 3.2. The proper calibration of m aims at producing a good compromise between the smoothing capability of CTS while preventing the CTS to degenerate into a case in which it performs purely randomly (irrespective to the timeliness). This observed unimodal behavior of the CTS with regard to the values of m makes the choice of this parameter easily tunable by some greedy searching algorithm. Such property is useful for practical application.

4.2.2 *Scalability Analysis.* Recall that, besides maintaining the classification task robust to the temporal effects, our main motivation is to reduce the test phase runtime. More specifically, we aim at providing a strategy to tackle the temporal effects in a scalable way, which is able to classify large sets of data in a reasonable amount of time. As we have seen, CTS indeed maintains such temporal robustness, achieving statistically equivalent (or even superior) results to those obtained by the temporally-aware algorithm. Now, we turn our attention to runtime and scalability issues.

In order to analyze the scalability properties of the CTS and the temporally-aware approaches, we considered our largest dataset (i.e., MEDLINE) and conducted the following experiment: for each $i = 2 \cdots 10$, we



(a) ACM-DL Dataset.



(b) MEDLINE Dataset.

Fig. 2. An Analysis Regarding the Effect of m in the CTS Algorithm.

generated a new test set $\mathbb{D}'_{test}$ such that $|\mathbb{D}'_{test}| = i \times |\mathbb{D}_{test}|$. This was accomplished by concatenating $\mathbb{D}_{test}$ with itself i times. Then, for each $|\mathbb{D}'_{test}|$ we measured the time spent to classify this test set by both CTS and temporally-aware settings. More specifically, for the CTS we measured the time spent to calibrate m through cross-validation over the training set, together with the time spent to train and test. For the temporally-aware approach, we measured the time spent to classify $\mathbb{D}'_{test}$ (recall that this is a lazy approach). This experiment was replicated 10 times by means of a 10-fold cross-validation, and the averaged measurements are reported in Figure 3.

As we can see in Figure 3, as the test set size increases, the runtime of the temporally-aware approach increases much faster than the CTS approach. This is explained by the lazy nature of the former: for each test document the training set must be revisited in order to learn the classification rules according to the observed temporal distance between training and test documents. Thus, as $|\mathbb{D}'_{test}|$ increases the cost of this approach

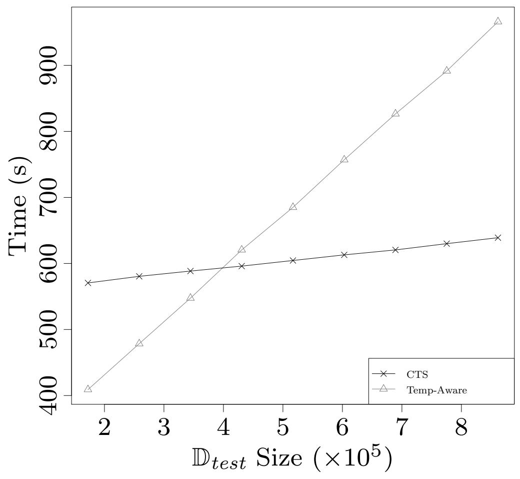


Fig. 3. Scalability Analysis: CTS *versus* Temporally-Aware Approaches.

may become infeasible for very large collections such as the Web or very large digital libraries.[10] On the other hand, we observe that the time spent by the CTS approach increases in a much smaller rate, meaning that the throughput of the classifier employing the CTS (in terms of classified documents per time unit) is superior to the throughput of the temporally-aware classifier. This highlights the practical applicability of CTS when classifying large amounts of data, without being compromised by the temporal effects observed in training data. Finally, the CTS and its possible derived approaches (such as CTS with SMOTE replication) open opportunities for future developments using new classifiers whose lazy versions would be infeasible to deal with the temporal effects in large datasets (e.g., SVM).

5. CONCLUSIONS AND FUTURE WORK

This work proposed a data engineering strategy aimed at preventing a classifier to be negatively impacted by the observed temporal effects in textual datasets. Such strategy, called Cascaded Temporal Smoothing, is characterized by handling the temporal effects in an entirely off-line setting by means of a controlled cascading of documents through the timeline in order to smooth out these effects in the training data. Unlike the previously proposed temporally-aware classifiers, our strategy does not incur in any overhead at the test phase, offering good scalability when tackling such issue, while not compromising the classification effectiveness. Thus, we can summarize the CTS advantages over the temporally-aware classifiers previously proposed, as being:

(1) the use of CTS does not involve any modifications on existing classification algorithms, ultimately being independent of classifier and ensuring its applicability in already consolidated automatic classification techniques (some of which were previously very difficult to extend by the fact that our temporally-aware solutions were inherently lazy);
(2) since CTS moves all the overhead of tackling the temporal effects to an off-line setting, its use maintains the classification throughput of the traditional classifiers while increasing classification effectiveness.

Clearly, there is room for further improvements in, at least, three directions. First, the synthetic data generator can be made much more sophisticated. Besides simple replication, as adopted by CTS, we can use other strategies, as SMOTE [Chawla et al. 2002] or the probabilistic generation strategy discussed in [Chen et al. 2011; Bermejo et al. 2011].The use of SMOTE when generating new samples for CTS may be advantageous since it defines less specific class boundaries with the introduction of new informative instances (which can not be achieved by simple replication). A probabilistic based sample generation can be even more advantageous since, besides generating informative instances, it is also able to reduce inter-class overlapping, which is a well known challenge in automatic classification. Second, we may significantly reduce the computational cost of CTS, eliminating the needs of maintaining the auxiliary data $\mathbb{D}_{aux}$, by directly determining the probability of sampling documents according to their creation time and the dataset temporal dynamics (e.g., considering the stability level of documents [Salles 2011] to determine the documents to be propagated through the timeline). Third, based on the probabilistic analysis of CTS, we plan to extend CTS to automatically adjust the number of documents to be transferred from the lower and upper bounds l_b and u_b to the target point in time p according to the temporal distance between them, in a way that the user will not need to tune the parameter m anymore. Moreover, we will apply the CTS in scenarios where the previously proposed temporally-aware solutions are infeasible due to the lazy requirement (e.g., SVM). Finally, we plan to provide a quantitative analysis regarding the temporal effects, as done in [Salles 2011], before and after the application of CTS in the dataset, in order to provide a better understanding regarding the behavior of CTS with relation to these effects.

REFERENCES

Bermejo, P., Gámez, J. A., and Puerta, J. M. Improving the performance of Naive Bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Systems with Applications* 38 (3): 2072–2080, 2011.

Chawla, N., Bowyer, K., Hall, L., and Kegelmeyer, W. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* vol. 16, pp. 321–357, 2002.

[10]Notice also that the number of classes in the experimented tasks is small (i.e., 11 classes in ACM-DL and 7 classes in MEDLINE). In tasks with hundreds or thousands of classes, the need for scalable classifiers may be even more critical (i.e., hierarchical classifiers).

Chen, E., Lin, Y., Xiong, H., Luo, Q., and Ma, H. Exploiting Probabilistic Topic Models to Improve Text Categorization Under Class Imbalance. *Information Processing & Management* 47 (2): 202–214, 2011.

Chiang, J.-H. and Chen, Y.-C. An Intelligent News Recommender Agent for Filtering and Categorizing Large Volumes of Text Corpus. *International Journal of Intelligent Systems* 19 (3): 201–216, 2004.

Cohen, W. W. and Singer, Y. Context-Sensitive Learning Methods for Text Categorization. *ACM Transactions on Information Systems* 17 (2): 141–173, 1999.

Couto, T., Cristo, M., Gonçalves, M. A., Calado, P., Ziviani, N., Moura, E., and Ribeiro-Neto, B. A Comparative Study of Citations and Links in Document Classification. In *Proceedings of the ACM IEEE Joint Conference on Digital Libraries*. Chapel Hill, USA, pp. 75–84, 2006.

Crammer, K., Dekel, O., Keshet, J., Shalev-shwartz, S., and Singer, Y. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research* vol. 7, pp. 551–585, 2006.

Fdez-Riverola, F., Iglesias, E., Díaz, F., Méndez, J., and Corchado, J. Applying Lazy Learning Algorithms to Tackle Concept Drift in Spam Filtering. *Expert Systems with Applications* 33 (1): 36–48, 2007.

Forman, G. Tackling Concept Drift by Temporal Inductive Transfer. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval*. Washington, USA, pp. 252–259, 2006.

Klinkenberg, R. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis* 8 (3): 281–300, 2004.

Klinkenberg, R. and Joachims, T. Detecting Concept Drift with Support Vector Machines. In *Proceedings of the International Conference on Machine Learning*. Stanford, USA, pp. 487–494, 2000.

Kohavi, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Québec, Canada, pp. 1137–1143, 1995.

Koren, Y. Collaborative Filtering with Temporal Dynamics. *Communications of the ACM* vol. 53, pp. 89–97, 2010.

Mourão, F., Rocha, L., Araújo, R., Couto, T., Gonçalves, M., and Meira Jr., W. Understanding Temporal Aspects in Document Classification. In *Proceedings of the International Conference on Web Search and Web Data Mining*. Palo Alto, USA, pp. 159–170, 2008.

Pan, S. J. and Yang, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22 (10): 1345–1359, 2010.

Rajan, S., Yankov, D., Gaffney, S. J., and Ratnaparkhi, A. A Large-Scale Active Learning System for Topical Categorization on the Web. In *Proceedings of the International Conference on World Wide Web*. Raleigh, USA, pp. 791–800, 2010.

Rocha, L., Mourão, F., Pereira, A., Gonçalves, M. A., and Meira Jr., W. Exploiting Temporal Contexts in Text Classification. In *Proceedings of the International Conference on Information and Knowledge Engineering*. Napa Valley, USA, pp. 243–252, 2008.

Salles, T. Automatic Document Classification Temporally Robust. M.Sc. Thesis, Federal University of Minas Gerais, Belo Horizonte, Brazil, 2011.

Salles, T., Rocha, L., Pappa, G. L., Mourão, F., Gonçalves, M. A., and Jr., W. M. Temporally-Aware Algorithms for Document Classification. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval*. ACM Press, Genebra, Switzerland, pp. 307–314, 2010.

Tsymbal, A. The Problem of Concept Drift: Definitions and Related Work. Technical report, Department of Computer Science, Trinity College, Dublin, Ireland, 2004.

Žliobaitė, I. Combining time and space similarity for small size learning under concept drift. In *Proceedings of the International Symposium on Foundations of Intelligent Systems*. Prague, Czech Republic, pp. 412–421, 2009.

Wang, H., Yu, P. S., and Han, J. Mining Concept-Drifting Data Streams. In *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach (Eds.). Springer, pp. 789–802, 2010.

Widmer, G. and Kubat, M. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning* 23 (1): 69–101, 1996.

Zhao, P. and Hoi, S. C. H. OTL: A Framework of Online Transfer Learning. In *Proceedings of the International Conference on Machine Learning*. Haifa, Israel, pp. 1231–1238, 2010.