On Dynamic Feature Weighting for Feature Drifting Data Streams

Jean Paul Barddal[1], Heitor Murilo Gomes[1], Fabrício Enembreck[1], Bernhard Pfahringer[2], and Albert Bifet[3]

[1]Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná, Brazil
[2]Department of Computer Science, University of Waikato, New Zealand
[3]Computer Science & Networks Dept (INFRES), Institut Mines-Télécom, Télécom ParisTech
Université Paris-Saclay, France
{jean.barddal,hmgomes,fabricio}@ppgia.pucpr.br
bernhard@cs.waikato.ac.nz
abifet@waikato.ac.nz

Abstract. The ubiquity of data streams has been encouraging the development of new incremental and adaptive learning algorithms. Data stream learners must be fast, memory-bounded, but mainly, tailored to adapt to possible changes in the data distribution, a phenomenon named concept drift. Recently, several works have shown the impact of a so far nearly neglected type of drift: feature drifts. Feature drifts occur whenever a subset of features becomes, or ceases to be, relevant to the learning task. In this paper we (i) provide insights into how the relevance of features can be tracked as a stream progresses according to information theoretical Symmetrical Uncertainty; and (ii) how it can be used to boost two learning schemes: Naive Bayesian and k-Nearest Neighbor. Furthermore, we investigate the usage of these two new dynamically weighted learners as prediction models in the leaves of the Hoeffding Adaptive Tree classifier. Results show improvements in accuracy (an average of 10.69% for k-Nearest Neighbor, 6.23% for Naive Bayes and 4.42% for Hoeffding Adaptive Trees) in both synthetic and real-world datasets at the expense of a bounded increase in both memory consumption and processing time.

1 Introduction

Data streams are ubiquitous, potentially unbounded and generated at a very fast pace. Examples of streaming data include, but are not limited, to: ATM transactions, readings in mobile sensor networks, social networks posts and stock trades. Motivated by these real world problems, data stream mining grew in popularity and became a very active research field with new techniques proposed every year aiming at learning from these sequences of data in an incremental, fast and memory-bounded fashion. Many of these new developments in data stream learning focus on the ephemeral characteristics of data streams, i.e. when the

underlying data distribution shifts with time, a phenomenon named *concept drift* [27].

More recently, studies [6, 7] shed light onto a specific kind of drift which has so far practically been neglected, the so-called *feature drifts*, sometimes referred to as contextual concept drifts in seminal works [27]. In practice, a feature drift occurs whenever a subset of features of a data stream becomes, or ceases to be, relevant to the learning task. As surveyed in [7] and empirically analyzed in [6], feature drifts pose challenges that are yet to be tackled by the data stream mining community.

In this paper we propose a low complexity and memory-bounded solution to track the relevance of features in streaming data accordingly to the information theoretical Symmetrical Uncertainty. Additionally, we show how this metric can be used to enhance prediction accuracy in k-Nearest Neighbor, Naive Bayes and Hoeffding Adaptive Tree classifiers when they are applied feature drifting data streams.

2 Learning from Data Streams

As times goes by, data acquisition and storage becomes cheaper and easier. As a consequence, companies and individuals can generate and store data at an increasing rate. Some of these data are generated sequentially and are so massive that it would not be practical nor useful to store them all. For instance, the data generated by a wearable gadget may only be meaningful for a small period of time, such that the burden to store or transmit it may be unjustifiable. These abundant sources of raw data may also be unintelligible to its possessors and in these situations, data mining techniques are often employed to extract meaningful patterns from apparent chaos. Currently, a lot of effort has been directed towards mining data that is generated in a continuous stream, an area that has been commonly known as data stream mining [10, 2].

Generally, data stream mining combines almost all problems of conventional batch learning (e.g. missing values, noisy data, outliers) with problems such as instability of the underlying concept and restrictive resources constraints. Specifically, data stream learners must (i) be able to process instances sequentially according to their arrival, (ii) act within limited memory space and processing time, (iii) deal with data instability (concept drifts); and (iv) be able to generalize well as instances' labels become available [17].

Ideally, algorithms for learning from data streams must include techniques for dealing with all aforementioned problems. However, not all of them must be addressed at once since it depends on the problem being tackled. For instance, a given problem setting may exhibit concept drifts but not suffer from a lack of labeled data or vice-versa.

2.1 Data Stream Classification

The most common (and widely explored) learning task in a data stream setting is undoubtedly classification. Formally, given a set of possible class labels $Y =$

$\{y_1, \ldots, y_c\}$ and a set of labeled training instances $X = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^n, y^n)\}$, a classifier uses the training set to build a model $f : \boldsymbol{x} \rightarrow Y$ capable of predicting the class label of an unlabeled instance $\boldsymbol{x}^i$. Precisely, each instance $\boldsymbol{x}$ is a d-dimensional feature vector belonging to a feature set $\mathcal{D} = \bigcup_{i=1}^{d} \{D_i\}$, that is possibly categorical, ordinal, numeric or most likely mixed.

Data stream (or online) classification is a variant of the traditional batch classification, and both are concerned with the problem of predicting class labels for unlabeled instances. The main difference between the batch and the online setting remains on how data are presented to the classifier. In a batch configuration data are entirely accessible in a finite and static dataset, while streaming data are presented sequentially over time [18] while f must be updated accordingly.

2.2 Concept Drift

Due to the inherent temporal aspect of data streams, their underlying data distribution may change over time, directly influencing changes to the concept to be learned, a phenomenon often referred as concept drift.

Let Eq. 1 denote a concept C, a set of prior probabilities of the classes and class-conditional probability density function [22].

$$C = \bigcup_{y_i \in Y} \{(P[y_i], P[\boldsymbol{x}|y_i])\} \tag{1}$$

Given a stream $\mathcal{S}$, retrieved instances i_t will be generated by a concept C_t. If during every instant t_i of $\mathcal{S}$ we have $C_{t_i} = C_{t_{i-1}}$, then the concept is stable. Otherwise, if between any two timestamps t_i and $t_j = t_i + \Delta$ (with $\Delta \geq 1$) it is the case that $C_{t_i} \neq C_{t_j}$, then we have observed a concept drift [17].

3 Problem Statement

Most existing algorithms for data streams tackle the infinite length and drifting concept characteristics. However, not much attention has been given to a specific kind of drift: feature drifts. Conversely to conventional concept drifts, where changes in the data distribution are claimed to occur inside the skewing of classes in ranges of features' values, feature drifts occur whenever a subset of features becomes, or ceases to be, relevant to the concept to be learned.

Until this point, the term "relevance" was used without a proper definition. In this paper we divide features in two types: relevant and irrelevant [7]. Assuming $S_i = \mathcal{D} \setminus \{D_i\}$, a feature D_i is deemed **relevant** *iff* Eq. 2 holds.

$$\exists S'_i \subset S_i, \text{ such that } P[Y|D_i, S'_i] \neq P[Y|S'_i] \tag{2}$$

Otherwise, the feature D_i is said **irrelevant**. In practice, if a feature that is statistically relevant is removed from a feature set, it will reduce overall prediction power since (i) it is strongly correlated with the class; or (ii) it belongs to a subset of features that is strongly correlated with the class [29].

Changes in the relevant subset of features enforce the learning algorithm to adapt its model to ignore the irrelevant attributes and to account for the newly relevant ones [22]. Given a feature space $\mathcal{D}$ at a timestamp t, we are able to select the ground-truth relevant subset $\mathcal{D}_t^* \subseteq \mathcal{D}$ such that $\forall D_i \in \mathcal{D}_t^*$ Eq. 2 holds and $\forall D_j \in \mathcal{D} \setminus \mathcal{D}_t^*$ the same definition does not. A feature drift occurs if, at any two time instants t_i and t_j, $\mathcal{D}_{t_i}^* \neq \mathcal{D}_{t_j}^*$ holds.

Let $r(D_i, t_j) \in \{0, 1\}$ denote a function which determines whether Eq. 2 holds for a feature D_i in a timestamp t_j of the stream. A positive relevance ($r(D_i, t_i) = 1$) states that $D_i \in \mathcal{D}^*$ in a timestamp t_i. A feature drift occurs whenever the relevance of an attribute D_i changes in a timespan between t_j and t_k, as stated in Eq. 3.

$$\exists t_j \exists t_k,\ t_j < t_k,\ r(D_i, t_j) \neq r(D_i, t_k) \tag{3}$$

Changes in $r(\cdot, \cdot)$ directly affect the ground-truth decision boundary to be learned by the learning algorithm. Therefore, feature drifts can be posed as a specific type of concept drift that may occur with or without changes in the data distribution $P[\boldsymbol{x}]$ [6, 7]. We emphasize that feature drifts are indeed targeted by the generic concept drift formalization, however, most existing works on concept drift detection and adaptation assume that the relevant subset of features remains the same and that drifts occur if certain values, or ranges of values, of attributes have their class distribution re-skewed.

As pointed out in [6] and [7], feature drifts are likely to occur in a variety of scenarios, but mainly in text stream scenarios, e.g. social media, SMS chats, online social networks (Facebook, Twitter) and e-mail spam detection systems.

As in conventional concept drifts, changes in $r(\cdot, \cdot)$ may occur during the stream. This enforces learning algorithms to detect changes in $\mathcal{D}^*$, discerning between features that became irrelevant and the ones that are now relevant and vice-versa. In order to overcome feature drifts, a learner must either (i) discard and derive an entirely new classification model that is consistent with the relevant features; or (ii) adapt its current model to relevance drifts [22].

4 Dynamic Feature Weighting

Feature weighting is broadly used in batch learning [11, 1] to assign different weights to features according to their relevance to the concept to be learned and to improve prediction accuracy. As shown earlier, in opposition to static scenarios, the relevance of features may increase or decrease during a data stream, thus, techniques for tracking and quantifying the proportions of such changes are needed.

The main hypothesis behind our proposals is that features can be dynamically weighted in order to augment the importance of relevant features and diminish the importance of those which are deemed irrelevant according to observed feature drifts. In this section we show how Symmetrical Uncertainty can be swiftly computed along a sliding window based on Entropy computation. Later, we introduce how Symmetrical Uncertainty can be applied into two distinct learning

schemes to boost prediction accuracy on feature drifting data streams. Finally, we detail the bounded computational overhead this proposal provides in processing time and memory usage.

4.1 Preliminaries

The relevance of a feature can be computed in diverse ways. In this section we discuss evaluation techniques for measuring the goodness of features for classification. Generally, a feature is good if it is relevant to predict the class. If one adopts correlation to measure the goodness of a feature, a feature will be deemed as relevant if its value surpasses a given threshold.

Several approaches exist to measure the correlation between two random variables. One such approach use linear correlation and another one is based on measures from information theory.

The most common formula for computing the correlation for a pair of variables (X, Y) is the linear correlation coefficient, which can be computed as follows:

$$c(X,Y) = \frac{\sum_{q \in D_i} \sum_{y_i \in Y} (q - \bar{D}_i)(y_i - \bar{Y})}{\sqrt{\sum_{q \in D_i} (q - \bar{D}_i)^2} \sqrt{\sum_{y_i \in Y} (y_i - \bar{Y})^2}} \tag{4}$$

The linear correlation coefficient is bounded in the $[-1; 1]$ interval. If X and Y are completely correlated, c takes the value of 1 or -1; and if these variables are completely uncorrelated, c is 0. Adopting linear correlation as a feature goodness measure has the benefit of eliminating completely uncorrelated features. Also, if data are linearly separable in its original representation then they will also be separable if all but one a group of linearly dependent features are removed [28]. Nevertheless, assuming linear correlations is not safe for a variety of domains. Linear correlation is likely to be unable to depict correlations which are non-linear in nature.

In our proposal, we adopt information theory approaches to compute the goodness of a feature. The first one is a measure of uncertainty of a random variable, named Entropy. The Entropy of a variable X is given by:

$$H(X) = -\sum_{x_i}^{X} P[X = x_i] \log_2 P[X = x_i] \tag{5}$$

On the other hand, the Entropy of a variable X after observing values of a variable Y (Conditional Entropy) is given by:

$$H(X|Y) = -\sum_{y_j}^{Y} P[Y = y_j] \sum_{x_i}^{X} P[X = x_i | Y = y_j] \tag{6}$$

Clearly, one of the drawbacks of picking Entropy as a goodness measure is that is it unable to work with numeric features, unless they are discretized. Since minimum (min) and maximum (max) values of features in streaming scenarios

Algorithm 1: Sliding window entropy. Adapted from [25].

input : window size w, a data stream $\mathcal{S}$.
output : be ready to provide the entropy h at any time.

1 Let $W \leftarrow \emptyset$ be the sliding window;
2 Let $h \leftarrow 0$ be the entropy;
3 Let $n \leftarrow 0$ be the number of instances in W;
4 Let $n_i \leftarrow 0$ be the number of instances with the y_i-th label;
5 **foreach** $(\boldsymbol{x}_i, y_i) \in \mathcal{S}$ **do**
6 **if** $|W| = w$ **then**
7 Dequeue oldest element from W from the y_j-th class;
8 $h \leftarrow DEC(h, n, n_j)$;
9 $W \leftarrow W \cup \{(\boldsymbol{x}_i, y_i)\}$;
10 $h \leftarrow INC(h, n, n_i)$;
11 **Function** $INC(h, n, n_i)$
12 Update $n \leftarrow n + 1$;
13 Update $n_i \leftarrow n_i + 1$;
14 **return** $\frac{n-1}{n}\left(h - \log_2 \frac{n-1}{n}\right) - \frac{n_i}{n}\log_2 \frac{n_i}{n} + \frac{n_i-1}{n}\log_2 \frac{n_i-1}{n}$
15 **Function** $DEC(h, n, n_i)$
16 Update $n \leftarrow n - 1$;
17 Update $n_i \leftarrow n_i - 1$;
18 **return** $\frac{n+1}{n}\left(h + \frac{n_i+1}{n+1}\log_2 \frac{n_i+1}{n+1} - \frac{n_i}{n+1}\log_2 \frac{n_i}{n+1}\right) + \log_2 \frac{n}{n+1}$

are unknown a priori, we adaptively discretized features using a sliding-window version of the Partition Incremental Discretization algorithm [16] with 10 bins.

One of the advantages of Entropy is that it can be computed along sliding windows. In Algorithm 1 we present the pseudocode for Entropy computation over sliding windows. Proofs for Entropy equations (lines 14 and 18) were omitted from this paper for the sake of brevity, thus, the reader is referred to [25] for details.

Entropy is the base for computing more robust metrics. One example is Information Gain, which is the amount by which the Entropy of a variable X decreases reflecting additional information about X provided by Y, and is given by:

$$IG(X|Y) = H(X) - H(X|Y) \tag{7}$$

An important trait of Information Gain is that it is symmetrical, i.e. $IG(X|Y) = IG(Y|X)$. To prove it, one needs to verify that $H(X) - H(X|Y) = H(Y) - H(Y|X)$ and this can be derived from $H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$.

As Entropy, Information Gain is biased towards features with more values. Therefore, different metrics that compensate for this bias are preferred. In this paper we picked Symmetrical Uncertainty (SU) as a goodness measure since it atones this bias. Symmetrical Uncertainty can be computed as follows:

$$SU(X,Y) = 2\left[\frac{IG(X|Y)}{H(X)+H(Y)}\right] = 2\left[\frac{H(Y)-H(Y|X)}{H(X)+H(Y)}\right] \tag{8}$$

The range of possible values for SU is the $[0;1]$ interval, where 1 indicates that the value of a variable completely predicts the other, while 0 indicates that X and Y are completely independent.

In order to compute SU along a sliding window, one must keep track of $H(D_i)$, $H(Y)$ and $H(Y|D_i)$ entropies. Both $H(D_i)$ and $H(Y)$ can be incremented and decremented in $\mathcal{O}(1)$ accordingly to Alg. 1, while the Conditional Entropy $H(Y|D_i)$ can be computed with separate $H(Y|D_i = q)$ entropies (see Eq. 6), also given by Alg. 1. If we assume that $q \in D_i$ and $|D_i| = m$, then SU can be computed with low computational complexity in the $\mathcal{O}(m)$ order for a single feature and $\mathcal{O}(dm)$ for all features in a d-dimensional data stream.

Memory-wise, the cost of tracking $H(Y)$ is $\mathcal{O}(|Y|)$, while the cost for $H(D_i)$ is $\mathcal{O}(m)$, thus, the total complexity is $\mathcal{O}(md)$ for a d-dimensional stream. Finally, $H(Y|D_i = q)$ incurs a cost of $\mathcal{O}(|Y|)$, therefore the total cost is $\mathcal{O}(md \times |Y|)$, when considering all features $D_i \in \mathcal{D}$.

4.2 Applying Feature Weighting to k-Nearest Neighbor Learning

k-Nearest Neighbor (*kNN*) [5] is one of the most fundamental, simple and widely used classification methods, which is able to learn complex (non-linear) functions [5]. *kNN* is a lazy learner since it does not require building a model before actual use. It classifies unlabeled instances according to the k "closest" buffered instances. The definition of "close" means that a distance measure is used to determine how similar/dissimilar two instances are. There are several approaches to compute distances between instances, nevertheless, the most common one is the Euclidian distance, given by Eq. 9, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are two arbitrary instances, and the summation occurs over all features $D_k \in \mathcal{D}$.

$$d_E(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{D_k \in \mathcal{D}} (\boldsymbol{x}_i[D_k] - \boldsymbol{x}_j[D_k])^2} \tag{9}$$

As discussed in a variety of works [3], Euclidian distances fail on representing in an effective fashion the distance between points (instances) in a high-dimensional space, since both irrelevant and redundant features have the same weight as relevant ones.

k-Nearest Neighbor with Feature Weighting (*kNN-FW*) is an extension to the original *kNN* algorithm that performs dynamic feature weighting to overcome both irrelevant features and feature drifts. *kNN-FW* comprises the following internal structures: an instance buffer queue and variables to track $H(D_i)$, $H(Y)$ and $H(Y|D_i = q)$. Finally, *kNN-FW* has two distinct steps: a training and a classification phase.

During the training step, instances i_t are retrieved from a stream $\mathcal{S}$ and enqueued in a buffer of size W. For every instance being enqueued or dequeued,

the values of $H(D_i)$, $H(Y)$ and $H(Y|D_i = q)$ are updated according to Alg. 1, thus, enabling prompt SU computation.

During the classification step, unlabeled instances $\boldsymbol{x}_t$ are classified according to the k-nearest neighbors available in buffer. In opposition to the conventional *kNN* algorithm, we modify the Euclidian distance to perform feature weighting accordingly to the discriminative power provided by Symmetrical Uncertainty, i.e. $w(D_i) = SU(D_i, Y)$.

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{D_k \in \mathcal{D}} w(D_k) \times (\boldsymbol{x}_i[D_k] - \boldsymbol{x}_j[D_k])^2} \tag{10}$$

Due to the dynamic computation of Symmetrical Uncertainty, *kNN-FW* is expected to assign weights dynamically accordingly to their discriminative power. In feature-drifting cases, features that become, or cease to be, relevant to the learning task will be promptly detected by changes in their Symmetrical Uncertainty values, generating appropriate changes for each feature's weight.

4.3 Applying Feature Weighting to Naive Bayes

Naïve Bayes (NB) is a probabilistic classifier based on Bayes theorem that works under the naïve independence assumption between features. These predictors are easy to build, can easily be incremented, and have no complicated parameter estimation, making it useful for large datasets and data streams. Classification (labeling) of instances in this learning scheme is given by Equation 11, that is, the class is chosen accordingly to the label y_i that maximizes the $P[y_i] \prod_{j=1}^{d} P[\boldsymbol{x}[D_j] \mid y_i]$ probability.

$$y = \underset{y_i \in Y}{\operatorname{argmax}} P[y_i] \prod_{j=1}^{d} P[\boldsymbol{x}[D_j] \mid y_i] \tag{11}$$

Although Naïve Bayes is commonly referred as an appropriate solution for high dimensionality problems [11], it has been shown to be prone to feature drifts [6]. Analogously to *kNN-FW*, we now propose the adoption of a dynamic weighting factor during Naive Bayes prediction. Naïve bayes with feature weight (NB-FW) also adopts Symmetrical Uncertainty as a weighting factor during classification, thus, probabilities are also weighted accordingly with $w(D_i) = SU(D_i, Y)$, thus, labeling is performed as follows:

$$y = \underset{y_i \in Y}{\operatorname{argmax}} P[y_i] \prod_{j=1}^{d} (w(D_i) + \xi) \times P[\boldsymbol{x}[D_j] \mid y_i] \tag{12}$$

where ξ is a small padding factor, set to 0.0001, used to avoid zero weights which would nullify the probabilities of some class values.

5 Analysis

In order to assess our proposal's performance, we built an experimentation environment encompassing both synthetic and real-world data. This analysis centers on prediction accuracy, processing time and memory usage.

5.1 Synthetic Data Stream Generators

Drifts are synthesized as the combination of two pure distributions. The probability that an instance is drawn from the prior or posterior concept inside a drift window is given by a sigmoid function. This drift framework is the default provided in the MOA framework [9] and all drift windows in our experiments have a length of 1,000 instances. All synthesized data streams contain 100,000 instances and contain 9 feature drifts. In the following, we introduce three synthetic data generators used to induce feature drifts on our experiments: AGRAWAL [4], Assets Negotiation (ASSETS) [14] and SEA-FD [6]. In the following experiments, we guarantee that feature drifts occur by changing the relevant subset of features between prior and posterior concepts.

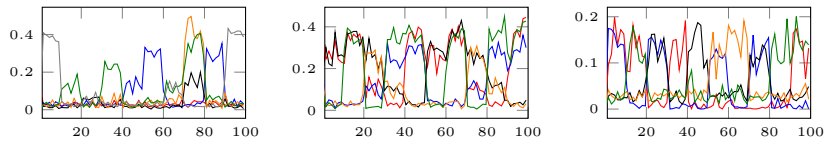
AGRAWAL. The AGRAWAL generator [4] produces data streams with the aim of determining whether a loan should or should not be given to a bank customer. This generator is composed by the following features: salary, commission, age, education level, car make, zip-code, house value, years house is owned and loan value. There are 10 functions for mapping instances to 2 possible classes, each of which relying on different subsets of these features.

Asset Negotiation (AN). This generator was originally presented in [14], where the aim was to simulate drifting bilateral multi-agent system negotiation of assets. Assets are described by the following features: color, price, payment, amount and delivery delay. The task is to predict whether an opposing agent would be interested, or not, in an asset, making this a binary classification problem. Feature drifts are synthesized with changes on the interest of an agent by modifying the concept through time given five functions, each of which is relying on a different subset of features.

SEA-FD. Described in [6], SEA-FD extends the SEA generator [26] and synthesizes streams with $d > 2$ uniformly distributed features, where $\forall D_i \in \mathcal{D}, D_i \in [0; 10]$ and $\mathcal{D}^* = \{D_\alpha, D_\beta\}$ is randomly chosen with the guarantee that it differs from the relevant subset of features from the earlier concept. As in [26], instances are labeled using $y = 1$ if $D_\alpha + D_\beta \leq \theta$ and $y = 0$ otherwise; where θ is a user-supplied threshold. In the following experiments we chose $\theta = 7$ since it is a widely used value in many papers of the area [8, 6].

5.2 Symmetrical Uncertainty Tracking in Synthetic Experiments

In order to exemplify how the dynamic weights are computed during experiments, we devote this section to present and discuss the Symmetrical Uncertainty tracking during synthetic experiments. Figure 1 presents the Symmetrical Uncertainty of features during AGRAWAL, ASSETS and SEA-FD experiments,



(a) AGRAWAL (b) ASSETS (c) SEA-FD

Fig. 1: Symmetrical Uncertainty of several features during synthetic experiments.

where each feature is represented by a curve with a different color. We highlight the fact that different features show higher SU values along the streams, thus confirming that our tracking strategy is able to depict feature drifts correctly.

5.3 Real-World Data

To complement the synthetic data, some real-world datasets were also used for the evaluation of the new algorithms. The adoption of real-world data is beneficial since they present differentiated behavior, e.g. the class distribution is often imbalanced and data are often noisy. On the other hand, it is nearly impossible to affirm whether drifts occur, making evaluation of drift detection unfeasible. We refrain from providing a detailed description of each used dataset for brevity. The used datasets are: Electricity (ELEC) [23], Kaggle's Give me Some Credit[1] (GMSC) and Spam Corpus (SPAM) [21].

5.4 Evaluated Algorithms

Besides kNN and Naive Bayes, we also report results for a Very Fast Decision Tree (VFDT) and a Hoeffding Adaptive Tree (HAT) since both perform embedded feature selection during training.

VFDT Very Fast Decision Tree (VFDT) is an incremental decision tree learner for non-drifting data streams [13]. The tree is recursively built as instances arrive and new split nodes are generated if the information gain of the two most discriminative features differ at least by ϵ, given by the Hoeffding bound [19]. The prediction at the leaves may occur following three different strategies: majority class, Naive Bayes and Adaptive Naive Bayes. The Adaptive Naive Bayes monitors the error rate of the majority class and Naive Bayes, always employing the one that currently best fits data, as judged by their recent estimated accuracy.

[1] Available at: `https://www.kaggle.com/c/GiveMeSomeCredit`. Last access in Feb. 25th, 2016.

HAT Hoeffding Adaptive Tree (HAT) algorithm is an extension to the VFDT to deal with drifts [20]. HAT updates its tree model over a sliding window and creates or updates decision nodes if the data distribution changes at an arbitrary split node. HAT detects data distribution changes according to the ADWIN change detector [8] provided in MOA [9]. Whenever ADWIN detects a change in a split node, the entire subtree is replaced by a new split node with the most discriminant feature if the Hoeffding bound is still met. As in VFDTs, the decision at leaf nodes may occur according to the majority class, Naive Bayes and Adaptive Naive Bayes methods.

5.5 Experimental Protocol

Accuracy is computed accordingly to the Prequential test-then-train procedure [15]. Prequential was chosen due to its way of monitoring a model's performance over time. Processing time is measured in seconds, while memory usage is given in RAM-Hours, where 1 RAM-Hour equals 1 GB of RAM dispended per hour of processing (GB-Hour). We adopted a window size $W = 1,000$ to keep track of Symmetrical Uncertainty in all experiments with the exception of Spam Corpus, where $W = 100$, due to the smaller number of instances in this dataset. An analysis of the impact of the window size W is later discussed in Section 5.7. All remaining parameters were set accordingly to the defaults provided in the Massive Online Analysis (MOA) framework [9].

5.6 Discussion

Table 1 presents the prequential accuracy results obtained during experiments. In all cases, the usage of our proposed feature weighting scheme was beneficial, providing an average boost of 10.69% and 6.23% for kNN and NB, respectively. To provide statistical significance to our claims, we performed Wilcoxon's, Friedman's and Nemenyi's tests [12]. Pairwise comparisons conducted with Wilcoxon's procedure between the original kNN and NB to their dynamically feature weighted versions with a 95% confidence level corroborated that there is statistical difference between their accuracy rates.

Finally, with the aid of Friedman's and Nemenyi's tests, we compared all algorithms in Table 1. Results showed that, {HAT, *kNN-FW*, VFDT, NB-FW} $\succ$ {*kNN*, NB}, also with a 95% confidence level. These results highlight that the weighting scheme is beneficial since it allows both *kNN* and NB to achieve comparable results with more sophisticated techniques that embed feature selection during stream learning, i.e. VFDT and HAT.

Tables 2 and 3 present processing time and memory usage obtained during the execution of experiments. With the exception of the ASSETS experiment, the adopted weighting scheme provides an computation overhead in both aspects. We claim, however, that this computational overhead is not damaging enough to prevent the usage of our weighting scheme, even in high dimensional problems, e.g. the SPAM experiment.

Table 1: Prequential accuracy (%).

Experiment	kNN	kNN-FW	NB	NB-FW	VFDT	HAT
AGRAWAL	57.74	65.64	59.18	62.68	69.98	**81.13**
ASSETS	85.02	87.87	70.51	77.11	91.57	**93.15**
SEA-FD	64.02	**84.14**	76.05	78.35	82.63	83.24
ELEC	54.31	**84.08**	57.62	73.39	79.23	83.46
GMSC	92.48	92.67	93.09	93.32	93.25	**93.37**
SPAM	80.56	83.87	66.26	75.22	79.32	**84.48**

Table 2: Processing time (s).

Experiment	kNN	kNN-FW	NB	NB-FW	VFDT	HAT
AGRAWAL	20.91	21.54	**0.45**	0.47	1.08	1.57
ASSETS	13.52	13.92	**0.28**	**0.28**	1.04	0.97
SEA-FD	104.68	107.82	**2.04**	2.08	4.85	6.15
ELEC	7.36	7.51	**0.36**	0.37	1.43	1.08
GMSC	32.35	33.64	**1.30**	1.31	7.90	15.43
SPAM	5911.54	6088.89	288.38	291.27	**253.76**	421.45

5.7 On the Impact of the Window Size W

Windowing is a common approach for both data management and dealing with drifting data. Our proposal relies on a window size parameter W that determines how much data should be considered to keep track of SU. Finding an optimal value for W is a trade-off without solution, a problem commonly referred as the stability-plasticity dilemma. While short windows reflect the current data distribution and ensures fast adaptation to drifts (plasticity), shorter ones worsen the performance of the system in stable areas. Conversely, larger windows give better performance in stable periods (stability), however, these imply a slower response to drifts.

In this section we evaluate the impact of the window size W in our proposal. We evaluated the original *kNN*, *kNN*-FW and NB-FW with different W values across the $[5; 2000]$ domain. Results for the Spam Corpus experiment were omitted since there was not enough time to run *kNN*-based algorithms in such high-dimensional scenarios in this amount of window sizes. In Figure 3 we report the average accuracy obtained during experiments.

In Figs. 3a and 3b we present the results obtained by *kNN* and *kNN*-FW, where it is clear that finding an optimal value that achieves the best results on all datasets is not trivial. However, by comparing the results in both graphics, we highlight that regardless of the chosen W value, the adoption of the proposed weighting scheme is beneficial.

On the other hand, the results presented in Fig. 3c show that for NB-FW results are robust across different window sizes, although the accuracy drops very slightly for windows with $W > 1,000$. Finally, we highlight the ELEC and

Table 3: RAM-Hours (GB-Hour).

Experiment	kNN	kNN-FW	NB	NB-FW	VFDT	HAT
AGRAWAL	7.57×10^{-7}	7.88×10^{-7}	$\mathbf{1.15 \times 10^{-9}}$	1.18×10^{-9}	5.05×10^{-8}	3.99×10^{-8}
ASSETS	3.77×10^{-7}	3.99×10^{-7}	$\mathbf{4.34 \times 10^{-10}}$	4.51×10^{-10}	7.84×10^{-8}	3.22×10^{-8}
SEA-FD	1.29×10^{-5}	1.34×10^{-5}	$\mathbf{1.40 \times 10^{-8}}$	1.45×10^{-8}	8.16×10^{-7}	2.65×10^{-7}
ELEC	2.43×10^{-7}	2.55×10^{-7}	$\mathbf{6.14 \times 10^{-10}}$	6.51×10^{-10}	3.45×10^{-8}	9.27×10^{-9}
GMSC	5.12×10^{-1}	5.43×10^{-1}	$\mathbf{2.12 \times 10^{-3}}$	2.19×10^{-3}	1.34×10^{-3}	4.21×10^{-3}
SPAM	1.20×10^{-6}	1.26×10^{-6}	$\mathbf{2.38 \times 10^{-9}}$	2.45×10^{-9}	3.88×10^{-7}	1.70×10^{-6}

Table 4: Prequential accuracy (%) for different leaf prediction strategies in HAT.

Experiment	HAT	HAT-kNN-FW	HAT-NB-FW
AGRAWAL	81.13	88.45	**91.03**
ASSETS	93.15	**95.63**	93.37
SEA-FD	83.24	81.12	**84.80**
ELEC	83.46	83.24	**83.56**
GMSC	93.37	**93.43**	93.39
SPAM	84.48	**92.62**	85.25

GMSC experiments, where the differences between the maximum and minimum accuracies were just 0.35% and 0.97%, respectively. This shows that the concept is relatively stable during the whole experiment, thus, the weights obtained across different window sizes are consistent.

5.8 Using Dynamically Weighted Classifiers as Leaves in Hoeffding Adaptive Trees

Although our weighting scheme favored *kNN* and NB classifiers, the Hoeffding Adaptive Tree (HAT) still outperforms both. In this section we investigate the adoption of our weighting scheme at the leaves of the HAT classifier in replacement of the adaptive Naive Bayes.

In Table 4 we compare the results for HAT with feature weighted *KNN* (HAT-kNN-FW) and NB (HAT-NB-FW) leaves against the original HAT. Results show that, with the exception of the SEA-FD experiment, the weighted approaches provide accuracy gains, regardless if it is under *kNN* or NB learning schemes. On average, results obtained showed a prediction rate gain of 4.42%.

6 Conclusion

In this paper we presented a time and memory-bounded solution for tracking the relevance of features based on the information theoretic concepts of Entropy and Symmetrical Uncertainty. We showed how these metrics can be successfully used to enhance k-Nearest Neighbor, Naive Bayesian and Hoeffding Adaptive Tree algorithms during both stable and feature drifting regions of data streams.

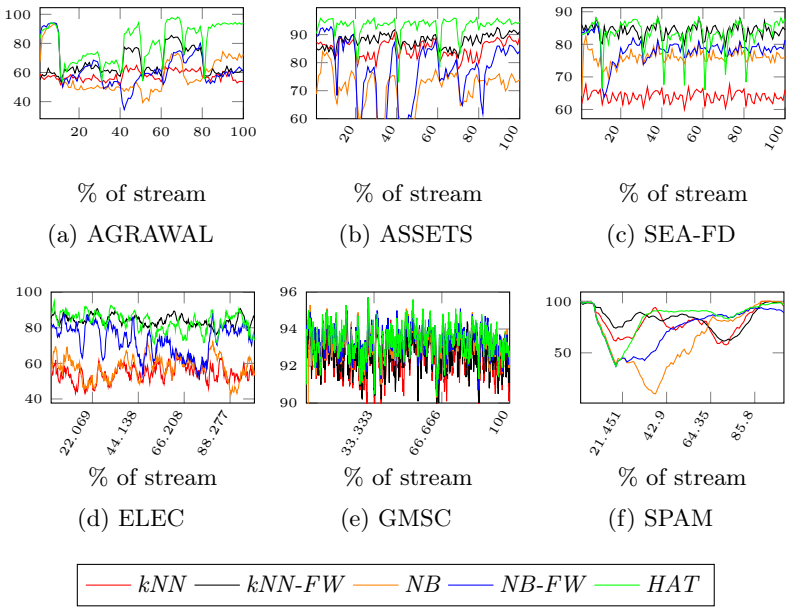


Fig. 2: Prequential accuracy (%) obtained during experiments.

Empirical evidence shows that the gains in prediction accuracy are significant and occur in both synthetic and real-world datasets. Results point out the need for future research into feature drift detection and adaptation.

Both Entropy and Symmetrical Uncertainty are computed along a sliding window, thus allowing adaptation to feature drifts. Finding an optimal window size is a trade-off without solution, thus, future works include the adopting of change detectors (e.g. ADWIN [8] and EWMA [24]) to eliminate the need of a predefined window size, which is a drawback of the proposed method.

Finally, there is the need to investigate the usage of these adaptive metrics (Entropy and Symmetrical Uncertainty) for the task of dynamic feature selection for data streams. This would allow a generic filter method that does not depend on any specific base classifier and that would select features dynamically according to the occurrence of feature drifts.

References

1. Charu C. Aggarwal. An introduction to data classification. In *Data Classification: Algorithms and Applications*, pages 1–36. 2014.
2. Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 81–92. VLDB Endowment, 2003.

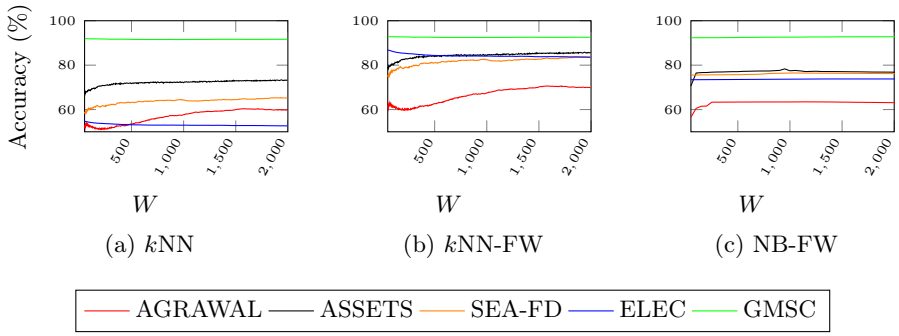


Fig. 3: Impact of W in prediction accuracy.

3. Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer Berlin Heidelberg, 2001.
4. R. Agrawal, T. Imielinski, and Arun Swami. Database mining: a performance perspective. *Knowledge and Data Engineering, IEEE Transactions on*, 5(6):914–925, Dec 1993.
5. D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
6. Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. Analyzing the impact of feature drifts in streaming learning. In *Proceedings of the 22th International Conference on Neural Information Processing*, ICONIP 2015. Springer, November 2015.
7. Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. A survey on feature drift adaptation. In *Proceedings of the International Conference on Tools with Artificial Intelligence*. IEEE, November 2015.
8. Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.
9. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.
10. Albert Bifet, Jesse Read, Indre Zliobaite, Bernhard Pfahringer, and Geoff Holmes. Pitfalls in benchmarking data stream classification and how to avoid them. In *ECML/PKDD (1)*, pages 465–479, 2013.
11. Lifei Chen and Shengrui Wang. Automated feature weighting in naive bayes for high-dimensional data classification. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1243–1252, New York, NY, USA, 2012. ACM.
12. G.W. Corder and D.I. Foreman. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Wiley, 2011.
13. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

14. Fabrício Enembreck, Bráulio Coelho Ávila, Edson E. Scalabrin, and Jean-Paul A. Barthès. Learning drifting negotiations. *Applied Artificial Intelligence*, 21(9):861–881, 2007.
15. J. Gama and P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proc. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM SIGKDD, Jun. 2009.
16. João Gama and Carlos Pinto. Discretization from data streams: Applications to histograms and data mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 662–667, New York, NY, USA, 2006. ACM.
17. João Gama, Indre Zliobaite, Albert Bifet, Mykole Pechenizkiy, and Abderlhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014.
18. Joao Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010.
19. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.
20. Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.
21. Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual data streams. In *in ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. 2006*, page 107. Springer Verlag, 2006.
22. Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. Heterogeneous ensemble for feature drifts in data streams. In *Advances in Knowledge Discovery and Data Mining*, volume 7302 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2012.
23. P.P. Rodrigues, J. Gama, and J.P. Pedroso. Hierarchical clustering of time-series data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 20(5):615–627, May 2008.
24. G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand. Exponentially weighted moving average charts for detecting concept drift. *ArXiv e-prints*, 2012.
25. Blaz Sovdat. Updating formulas and algorithms for computing entropy and gini index on time-changing data streams. *CoRR*, abs/1403.6348, 2014.
26. W. Nick Street and Y. Kim. A streaming ensemble algorithm (sea) for large-classification. In *Proc. of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM SIGKDD, Aug. 2001.
27. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, April 1996.
28. Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 856–863. AAAI Press, 2003.
29. Zheng Zhao, Fred Morstatter, Shashvata Sharma, Salem Alelyani, Aneeth Anand, and Huan Liu. Advancing feature selection research. *ASU feature selection repository*, pages 1–28, 2010.