2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Overcoming Feature Drifts via Dynamic Feature Weighted k-Nearest Neighbor Learning

Jean Paul Barddal, Heitor Murilo Gomes, Jones Granatyr, Alceu de Souza Britto Jr., Fabrício Enembreck
Graduate Program in Informatics (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
{jean.barddal, hmgomes, alceu, fabricio}@ppgia.pucpr.br, jones.granatyr@pucpr.edu.br
Curitiba, Brazil

***Abstract*—Extracting useful knowledge from data streams is problematic, mainly due to changes in their data distribution, a phenomenon named concept drift. Recently, studies have shown that most of existing algorithms for learning from data streams do not encompass techniques for a specific kind of drift: feature drifts. Feature drifts occur when features become, or cease to be, relevant to the learning task. In this paper, we propose an extension to the *k*-nearest neighbor classifier, so its distances' computations are weighted according to their current discriminative power. On our proposal, the discriminative power of features is given by entropy, which is swiftly computed over a sliding window. Empirical evidence shows that our approach is able to overcome several existing algorithms in accuracy and feature drift adaptation, while at the expense of bounded processing time and memory space.**

I. Introduction

Extracting useful knowledge from data streams is problematic, mainly due to possible changes in their data distributions, a phenomenon named concept drift. In the last years, the data stream community has developed several approaches for mining data streams in both supervised [1], [2] and unsupervised [3] fashions, always focusing on overcoming concept drifts.

In this paper, we tackle a specific kind of drift that has not earned much attention in the literature: feature drifts. Feature drifts occur whenever features become - or cease to be - relevant to the learning task. We propose an extension to the *k*-nearest neighbor classifier, so its distances' computations are weighted according to their current discriminative power. We show that the discriminative power of features can be calculated based on the information theory concept of entropy, which can be swiftly computed over sliding windows.

This paper is divided as follows. Firstly, Section II introduces the problem of classification in data streams while Section III states our aim: feature drifts. Following, Section IV briefly reviews the original k-Nearest Neighbor classifier, recalls formulas for computing entropy over sliding windows and introduces our proposal, namely k-Nearest Neighbor with Feature Weighting (*kNN-FW*). Later, Section V provides an empirical evaluation of *kNN-FW* against several data stream classification algorithms, showing its efficiency in both synthetic and real-world problems. Finally, Section VI concludes this paper and states envisioned future works.

II. Data Stream Classification

Let $\mathcal{S} = [i^t]_{t=0}^{t\to\infty}$ define a data stream providing instances $i^t = (\vec{x}^t, y^t)$, each of which arriving at a timestamp t, where $\vec{x}^t$ is a d-dimensional feature vector belonging to a feature set $\mathcal{D}$ and $y^t \in Y$ is the ground-truth label of $\vec{x}^t$. The feature set of a data stream is described as $\mathcal{D} = [D_j]_{j=1}^{d}$, that is possibly continuous, ordinal, categorical or mixed.

By far, the most common approach for extracting useful knowledge from data streams is classification. Assuming a set of possible classes $Y = \{y_1, \dots, y_c\}$, a classifier builds and incrementally updates a model $f : \vec{x} \to Y$ from labeled seen data in order to classify future unlabeled instances.

Classifying data streams embed several restrictions [4]. Firstly, instances arrive over time and there is no control over their order nor how they should be processed. Also, streams are potentially unbounded, so instances must be discarded right after their processing or periodically, given available main memory space. Finally, the data distribution is expected to change over time, implying in changes in the concept to be learned, a phenomenon named concept drift. Therefore, classifiers must possess strategies to detect drifts and adapt their models f accordingly [4].

III. Feature Drift

Most of the existing works for classifying data streams tackle the infinite length and drifting concept properties of these ephemeral environments. Recently, studies [5], [6] have demonstrated that existing data stream learners do not encompass techniques for one specific kind of drifts: feature drifts [7], [8]. As previously described, feature drifts occur whenever a feature of the stream becomes, or ceases to be, relevant to a concept C_t.

Given a set of features $\mathcal{D}$ at a timestamp t_i, we are able to select its ground-truth relevant subset of features $\mathcal{D}^*_{t_i} \subseteq \mathcal{D}$. A feature drift occurs if, at any two timestamps t_i and t_j, $\mathcal{D}^*_{t_i} \neq \mathcal{D}^*_{t_j}$ betides [5].

Let $r(D_i, t_j) \in \{0, 1\}$ denote a function that determines the relevance of a feature $D_i \in \mathcal{D}$ in a timestamp of the stream. A positive relevance, i.e. $r(D_i, t_j) = 1$, states that $D_i \in \mathcal{D}^*_j$ and that it impacts the underlying conditional probabilities $P[\vec{x}|Y]$ of the concept C_t. Therefore, a feature drift occurs if the relevance of a feature D_i changes in a timespan between t_j and t_k, as stated in Eq. 1.

$$\exists t_j \exists t_k, t_j < t_k, r(D_i, t_j) \neq r(D_i, t_k) \tag{1}$$

Changes in $r(\cdot,\cdot)$ affect the decision boundary to be learned by the classifier. As in other types of drifts, several changes in $r(\cdot,\cdot)$ may occur during the stream. This enforces classifiers to discern between relevant and irrelevant features as the stream progresses. Whenever a classifier detects a feature drift, it must either (i) discard and learn an entirely new model; or (ii) adapt the current model to relevance drifts [7], [6].

IV. Proposed Algorithm

In this section we review the original k-Nearest Neighbor algorithm, recall the computation of entropy over sliding windows and introduce our proposal: the k-Nearest Neighbor with Feature Weighting (*kNN-FW*).

A. k-Nearest Neighbor Classification

k-Nearest Neighbor (*kNN*) [9] is one of the most fundamental, simple and widely used classification methods, which is able to learn complex (non-linear) functions [9]. *kNN* is a lazy learner since it does not require building a model before actual use. *kNN*'s version for data streams classifies 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 used one is the Euclidian distance, given by Eq. 2.

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

In order to deal with time and memory space constraints, *kNN* buffers instances as they become available in a FIFO buffer of size W. Defining a value for W is hazy as it must be set according to available memory space and processing time since the computational time for classifying each new instance is $\mathcal{O}(Wd)$.

B. Preliminaries

The success of *kNN* relies on which instances are deemed close, a concept defined by its distance function. The Euclidian distance allows irrelevant features to have as much effect on distances as relevant ones.

The hypothesis behind our proposal is that *kNN* can be extended to overcome irrelevant features in feature drifts through incremental tracking of their discriminative power. Feature weighting is broadly used in batch learning [10], however, to the best of authors' knowledge, only the work of [11] provides feature weighting techniques for streaming scenarios. Our work significantly differs from [11] since weights in [11] are given by temporal functions developed for smooth changes and do not rely on information theory aspects.

Our proposal performs relevancy tracking by adopting Entropy as a metric of discriminative power. Although Entropy is claimed to be biased towards features with bigger domains, its computation is fast, trivial and feasible over a sliding window.

Algorithm 1: Sliding window entropy. Adapted from [12].

```
input  : window size $W$, a data stream $\mathcal{S}$.
output : be ready to provide the entropy $h$ at any time.
Let $\mathcal{W} \leftarrow \emptyset$ be the sliding window;
Let $h \leftarrow 0$ be the entropy;
Let $n \leftarrow 0$ be the amount of instances in $W$;
Let $n_i \leftarrow 0$ be the number of instances with the $y_i$-th label;
foreach $(\vec{x}_i, y_i) \in \mathcal{S}$ do
    if $|\mathcal{W}| = W$ then
        Dequeue oldest element from $\mathcal{W}$ from the $y_j$-th class;
        $h \leftarrow DEC(h, n, n_j)$;
    $W \leftarrow \mathcal{W} \cup \{(\vec{x}_i, y_i)\}$;
    $h \leftarrow INC(h, n, n_i)$;
Function $INC(h, n, n_i)$
    Update $n \leftarrow n + 1$;
    Update $n_i \leftarrow n_i + 1$;
    return
    $\frac{n-1}{n}\left(h - \log_2 \frac{n-1}{n}\right) - \frac{n_i}{n}\log_2\frac{n_i}{n} + \frac{n_i-1}{n}\log_2\frac{n_i-1}{n}$
Function $DEC(h, n, n_i)$
    Update $n \leftarrow n - 1$;
    Update $n_i \leftarrow n_i - 1$;
    return
    $\frac{n+1}{n}\left(h + \frac{n_i+1}{n+1}\log_2\frac{n_i+1}{n+1} - \frac{n_i}{n+1}\log_2\frac{n_i}{n+1}\right) + \log_2\frac{n}{n+1}$
```

In this paper, we adopted the conditional entropy as a discriminative measure for features. Conditional entropy is an information theoretic measure that quantifies how "impure" a feature D_i is regarding the class Y. This entropy can be computed according to Eq. 3, where q iterates over all possible values of a feature D_i.

$$H(Y|D_i) = \sum_{q \in D_i} P[q] \times H(Y|D_i = q) \tag{3}$$

The smaller the value of entropy $H(Y|D_i)$, bigger is the depiction of Y by D_i. Therefore, features with smaller entropies predict Y better.

Algorithm 1 presents the computation of the $H(Y|D_i = q)$ conditional entropy given a sliding window. For the sake of brevity, proofs for formulas used in Alg. 1 were omitted, thus, the reader is referred to [12] for details.

Clearly, one of the drawbacks of picking entropy as a 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 are unknown a priori, every time they change during the processing of the stream, a new discretization must be performed. In our proposal, we chose to discretize features D_i in 10 bins of equal sizes, each with a $\frac{\max D_i - \min D_i}{10}$ length.

C. kNN-FW

k-Nearest Neighbor with Feature Weighting (*kNN-FW*) is an extension to the original *kNN* algorithm that performs dynamic feature weighting to overcome feature drifts. *kNN-FW* maintains conditional entropies $H(Y|D_i = q)$ updated accordingly a sliding window of size W.

Whenever an instance $(\vec{x}_i, y_i)$ is enqueued or dequeued from buffer, conditional entropies are updated accordingly to Alg. 1. This allows anytime computation of entropy since all $H(Y|D_i = q)$ values are updated and can be accessed in $\mathcal{O}(1)$.

To label instances $\vec{x}_t$, *kNN-FW* compares it to buffered instances. In comparison to the original *kNN*, our proposal assumes an adaptation to the conventional Euclidian distance, which is weighted given the discriminative power provided by conditional entropy. Since smaller values of entropy highlight discriminative features, the differences in Eq. 4 are weighted with the complement of entropy, i.e. $w(D_k) = 1 - H(Y|D_k)$.

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{\sum_{D_k \in \mathcal{D}} w(D_k) \times (\vec{x}_i[D_k] - \vec{x}_j[D_k])^2} \quad (4)$$

Due to the dynamic computation of entropy, *kNN-FW* is expected to dynamically assign weights to features according to their current discriminative power. During non-feature-drifting streams, it is expected that irrelevant features are unaccounted for during voting, while discriminant features are emphasized. In feature-drifting cases, features that become, or cease to be, relevant to the learning task will be promptly detected by changes in their entropies, implying in changes in features' weights.

D. A Note on Computational Complexity

In comparison to the original *kNN*, our proposal performs computations of the entropy of features as new instances arrive. The cost for classifying an instance remains the same ($\mathcal{O}(Wd)$) since the probabilities needed for entropy computation can be promptly obtained in $\mathcal{O}(1)$. Nevertheless, when enqueueing and dequeuing an instance from the buffer, *kNN-FW* must update its conditional entropies, each with an $\mathcal{O}(d)$ cost.

In terms of memory space, besides storing an instance buffer, *kNN-FW* must also maintain conditional entropies' counters with a cost of $\mathcal{O}(cd)$, where $c = |Y|$. These are important overheads that must be accounted for since the usage of *kNN-FW* becomes unfeasible in certain streams where d is big (e.g. SPAM experiment in Sec. V-D).

V. Experimental Results

In this section, we assess accuracy, processing time and memory usage of *kNN-FW* and compare it to other streaming classification algorithms. Firstly, we introduce both synthetic data generators and real-world datasets used in the evaluation. Later, we state the experimental protocol adopted during experiments and comparison. Finally, we discuss the results obtained.

A. Synthetic Data

In this section, we present the synthetic data generators used in experiments. In all cases, streams are generated with 5% noise and the relevant subsets of features in prior and posterior concepts differ.

1) SEA-FD: To verify the impact of feature drifts in existing streaming learning algorithms, the first synthetic generator used is SEA-FD [5]. SEA-FD is an extension to the SEA generator [13] and it simulates streams with $d > 2$ uniformly distributed features given by the user, where $\forall D_i \in \mathcal{D}, D_i \in [0; 10]$ and only two randomly picked features are relevant to the concept to be learned: D_ω and D_ζ. As in [13], the class value y is given according to Eq. 5, where θ is a user-given threshold. In our experiments, we adopted $\theta = 7$ since this value is widely used in other works [14], [1].

$$y = \begin{cases} 1, \ D_\omega + D_\zeta \leq \theta \\ 0, \ otherwise \end{cases} \quad (5)$$

2) Binary Generator with Feature Drift (BG-FD): The Binary Generator with Feature Drift (BG-FD) generates instances composed by boolean features [15]. From the entire set of features $\mathcal{D}$, only a random subset $\mathcal{D}^* \subset \mathcal{D}$ is relevant to the concept to be learned. Additionally, $|\mathcal{D}^*| = d_r$, where $d_r < d$ is a user-given parameter. Labels of instances are given according to Eq. 6 and labels are evenly likely to occur.

$$y = \begin{cases} 1, \ \bigwedge_{D_i \in \mathcal{D}^*} D_i \\ 0, \ otherwise \end{cases} \quad (6)$$

3) LED Generator: The LED generator (LED), early introduced in [16], generates instances with 24 boolean features, 17 of which are irrelevant. The goal is to predict the digit displayed on a seven-segment LED display, where each feature has a 10% chance of being inverted. Our experiments do not encompass feature drifts using this generator, yet, it is used to assess the performance of classifiers since it does possess several irrelevant features.

B. Real-world Data

Jutting synthetic data, our proposal is also evaluated along real-world datasets. We refrain from providing a detailed description of each dataset for the sake of brevity and due to their broad usage by the community. For more details about the datasets, we refer the reader to the original publications: Electricity (ELEC) [17], Forest Covertype (COV) [18] and Spam Corpus (SPAM) [19].

C. Experimental Protocol

In this section we present the experimental protocol adopted in experiments, focusing on the feature drift framework, parametrization, evaluation and statistical testing.

1) Feature Drift Framework: To promote synthetic feature drifts in streams, we adopted the set-up provided in the Massive Online Analysis (MOA) framework [20]. This set-up treats a drift as a combination of two pure distributions that characterizes concepts before and after the drift. In the case of feature drifts, each of the concepts is defined by different subsets of relevant features. Finally, a window size w determines the length of the drifting region where instances can belong to both prior and posterior concepts.

2) Algorithms and Parametrization: We present results for the original *kNN*, a *kNN* aided with a concept drift detector, namely Adaptive Sliding Windowing (ADWIN) [21] and our proposal: *kNN-FW*. Also, we evaluate classifiers with different biases: an Updatable Naive Bayes (NB) and a Hoeffding Tree (HT) [22]. Both *kNN* and *kNN-FW* assume the same parameters and values: $k = 3$ and $W = 1,000$. We acknowledge that determining an optimal value for both parameters is an open issue that relies on each stream domain and their impact is not assessed in this paper. We forfeit from providing an analysis of the impact of the window size W since it has the same impact as in the conventional *kNN* algorithm, i.e. bigger windows do not allow rapid drift recovery and result in increases in time processing and memory consumption, while smaller windows impact accuracy in stable regions of the stream and provide quicker classification time and lessened memory space usage. Remaining algorithms' parameters are set according to the values presented in original papers.

All synthetic data streams have a length of 100,000 instances and 50 features. SEA-FD and BG-FD possess a feature set size $|\mathcal{D}| = 50$ and 9 equally distributed drifts. Streams with a (A) suffix contain abrupt drifts, i.e. $w = 1$, while streams with a (G) suffix are gradual, where $w = 1000$.

3) Evaluation and Statistical Testing: Accuracy is measured using the Prequential test-then-train method. Although claimed as pessimistic, the Prequential procedure [23] monitors the evolution of the performance of models over time given a sliding window. The window size was set to 1,000 instances for the synthetic experiments and 100 for real-world datasets. Processing time is measured as the time that the algorithms spend in the processor (in seconds) and memory usage is presented in RAM-Hours, where 1 RAM-Hours equals 1 GB of RAM used per hour.

All experiments' results presented in this paper were obtained on an Intel Xeon CPU E5649 @ 2.53GHz×8 computer with 16GB of memory and under the Massive Online Analysis (MOA) framework [20].

Statistical differences are verified according to Friedman's and Nemenyi's non-parametric tests [24] with a confidence level of 95% ($\alpha = 0.05$) and results are reported with Critical Differences (CD) graphics.

D. Discussion

In Tabs. I, II and III we present the results obtained during experiments in terms of accuracy, processing time and memory usage, respectively. Tab. I presents the average prequential accuracy obtained, where one can see that *kNN-FW* presents higher accuracy rates in most cases, especially when compared to the original *kNN*, where gains range from 3.33% to 11.84% and are in average 6.59% higher. In order to clarify the results obtained, we present in Figs. 1 and 2 the prequential accuracy obtained by algorithms during experiments. We omit plots for SEA-FD(A) and BG-FD(A) experiments since they are similar to the ones provided in SEA-FD(G) and BG-FD(G). Based on the results obtained, we highlight the high adaptability to feature drifts in SEA-FD and BG-FD experiments and the boost in the accuracy of *kNN-FW* to other algorithms in real-world data. Also, the usage of ADWIN has not allowed *kNN* to overcome feature drifts (Figs. 1a through 1e), showing that ADWIN is incapable of promptly detecting this kind of drift. One example of ADWIN's inefficiency is given in the LED experiment. In this experiment, no drifts occur, however, ADWIN still detects several false positives, thus, performs several *kNN* resets. These resets jeopardize prediction accuracy since it frees up the buffer of instances during stable areas.

TABLE I: Accuracy obtained during experiments.

Experiment	*kNN*	*kNN-FW*	*kNN-ADWIN*	NB	HT
SEA-FD(A)	75.28	**85.26**	75.23	79.83	80.82
SEA-FD	75.28	**84.25**	75.20	79.83	80.80
BG-FD(A)	86.00	**90.12**	86.11	69.99	78.21
BG-FD	85.70	**89.51**	84.68	69.99	78.25
LED	78.54	82.81	59.08	86.64	**86.47**
COV	91.20	**94.53**	86.76	60.51	80.29
ELEC	78.32	**90.16**	77.84	73.40	79.23
SPAM	78.14	**84.52**	84.15	75.22	79.32

TABLE II: Processing time (s)

Experiment	*kNN*	*kNN-FW*	*kNN-ADWIN*	NB	HT
SEA-FD(A)	112.08	241.44	203.69	**1.98**	4.37
SEA-FD	113.67	206.58	199.35	**1.93**	4.27
BG-FD(A)	102.41	212.43	176.51	**1.42**	3.84
BG-FD	104.04	171.10	161.46	**1.49**	3.64
LED	49.50	52.15	25.91	**0.79**	1.41
COV	304.71	757.32	358.06	**27.16**	268.94
ELEC	8.35	12.96	13.62	**0.53**	1.45
SPAM	6150.98	8379.30	5621.52	283.28	**262.03**

We highlight the results obtained by HT (Figs. 2a through 2f). Although its learning strategy is based on feature selection for building the decision tree, it does not embed strategies to overcome feature drifts, therefore, presents noticeable accuracy drops in these scenarios.

With empirical evidence, we verify that *kNN-FW* is able to overcome both feature drifts and irrelevant features better than a conventional *kNN* classifier, highlighting the need for future works in feature selection for data streams and feature drift detection and adaptation. The only experiment where our proposal was unable to achieve the highest prediction rates was the LED experiment. We claim that this occurred since LED has several irrelevant features that, when combined with noise rates, damage the computation of entropy.

Finally, with the aid of Friedman's and Nemenyi's tests, we were able to determine that $\{$*kNN-FW*, HT$\} \succ \{$*kNN*, *kNN-ADWIN*, NB$\}$ with a 95% confidence level in terms of accuracy (Fig. 3).

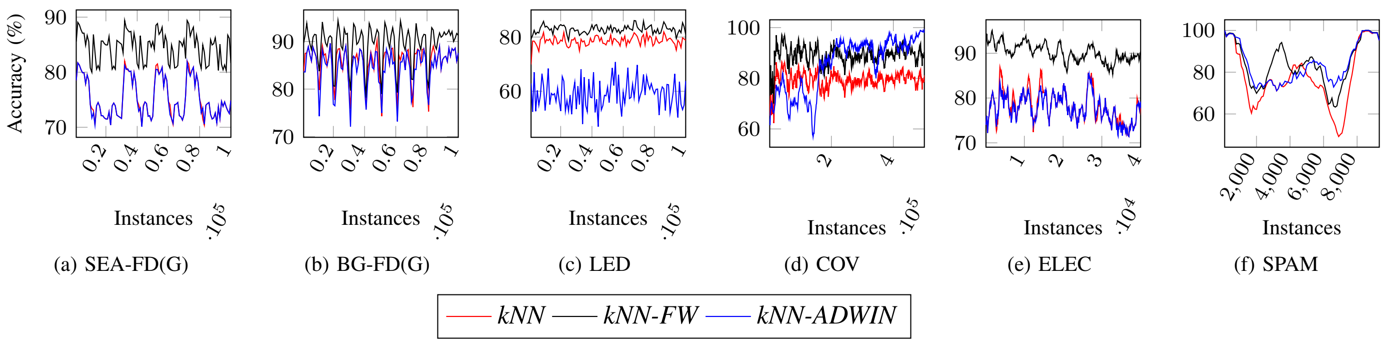

Fig. 1: Prequential accuracies (%) for *kNN*-based algorithms.

TABLE III: RAM-Hours (GB-Hour)

Experiment	*kNN*	*kNN-FW*	*kNN-ADWIN*	NB	HT
SEA-FD(A)	6.03×10^{-6}	1.51×10^{-5}	2.56×10^{-5}	$\mathbf{1.50 \times 10^{-8}}$	6.17×10^{-7}
SEA-FD	2.77×10^{-7}	8.21×10^{-7}	2.47×10^{-5}	$\mathbf{1.33 \times 10^{-8}}$	6.07×10^{-7}
BG-FD(A)	4.12×10^{-5}	5.21×10^{-5}	2.27×10^{-5}	$\mathbf{1.28 \times 10^{-8}}$	6.07×10^{-7}
BG-FD	1.40×10^{-5}	5.25×10^{-5}	1.87×10^{-5}	$\mathbf{1.35 \times 10^{-8}}$	5.69×10^{-7}
LED	8.68×10^{-5}	5.20×10^{-4}	6.01×10^{-7}	$\mathbf{5.56 \times 10^{-9}}$	1.48×10^{-8}
COV	3.44×10^{-6}	9.43×10^{-6}	3.52×10^{-5}	$\mathbf{3.67 \times 10^{-7}}$	1.32×10^{-4}
ELEC	1.30×10^{-5}	2.67×10^{-5}	4.50×10^{-7}	$\mathbf{7.59 \times 10^{-10}}$	3.40×10^{-8}
SPAM	3.32×10^{-1}	5.33×10^{-1}	3.62×10^{-1}	2.08×10^{-3}	$\mathbf{1.38 \times 10^{-3}}$

Naturally, keeping track of entropies incrementally as the stream progresses has its cons. In Tabs. II and III, we present processing time and memory usage of both classifiers during experiments, where one can see that *kNN-FW* is outperformed by other algorithms in all cases. This result is expected since *kNN-FW* has an extra computational complexity of $\mathcal{O}(2d)$ (due to enqueueing and dequeueing) in processing time and $\mathcal{O}(cd)$ in memory space when compared to the original *kNN*. With the aid of Friedman's and Nemenyi's tests, we are able to determine that {NB, HT} $\succ$ {*kNN*, *kNN-ADWIN*, *kNN-FW*} for both processing time and memory space usage (Figs. 4 and 5).

VI. Conclusion

This paper presented *kNN-FW*, a k-nearest neighbor algorithm with dynamic feature weighting. With the track of features' discriminative power at all moments of the stream by using a sliding window, *kNN-FW* is able to dynamically assign weights for features as the stream passes using entropy. Entropy is computed along a sliding window, which enables the computation of features' discriminative powers as new instances arrive. Empirical evidence shows that *kNN-FW* presents important gains in accuracy in both synthetic feature drifting streams and in real-world datasets. This shows the need for future works in feature drift detection and adaptation.

A starting point would be the evaluation of other feature discriminative power metrics (e.g. Correlation, Gain Ratio, Information Gain, χ^2, Gini Index), nevertheless, their incremental and adaptive versions are non-trivial. We thus intend to further investigate the combination of our proposed metric to compute more sophisticated metrics (e.g. Information Gain, Gain Ratio and Symmetrical Uncertainty) and their combination with drift detectors (e.g. ADWIN [21]), so feature drifts can be detected with statistical confidence.

Finally, the gain obtained in accuracy comes at the expense of both processing time and memory space, yet, its difference to the conventional kNN is not statistically significant, therefore, feasible and recommended when nearest neighbor learning is a choice.

Since the classification time of an instance by *kNN-FW* requires $\mathcal{O}(Wd)$, in future works we intend to investigate the adoption of dynamic feature selection techniques to diminish both training and classification feature spaces to d', where $d' \ll d$, impacting in a $\mathcal{O}(Wd')$ processing time complexity. This is important in high-dimensional streams, e.g. Spam Corpus, where d is enormous and very few of them (d') are relevant to the concept to be learned.

Acknowledgment

This project was financially supported by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) through the *Programa de Suporte à Pós-Graduação de Instituições de Ensino Particulares* (PROSUP) program.

References

[1] J. P. Barddal, H. M. Gomes, and F. Enembreck, "SFNClassifier: A scale-free social network method to handle concept drift," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*, ser. SAC 2014. ACM, March 2014.

[2] ——, "Advances on concept drift detection detection in regression tasks using social networks theory," *International Journal on Natural Computing Research*, pp. 1–14, 2015.

[3] ——, "SNCStream: A social network-based data stream clustering algorithm," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC)*, ser. SAC 2015. ACM, April 2015.

[4] J. a. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2523813

[5] J. P. Barddal, H. M. Gomes, and F. Enembreck, "Analyzing the impact of feature drifts in streaming learning," in *Proceedings of the 22th International Conference on Neural Information Processing*, ser. ICONIP 2015. Springer, November 2015.

[6] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, and L. Wan, "Heterogeneous ensemble for feature drifts in data streams," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7302, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30220-6_1

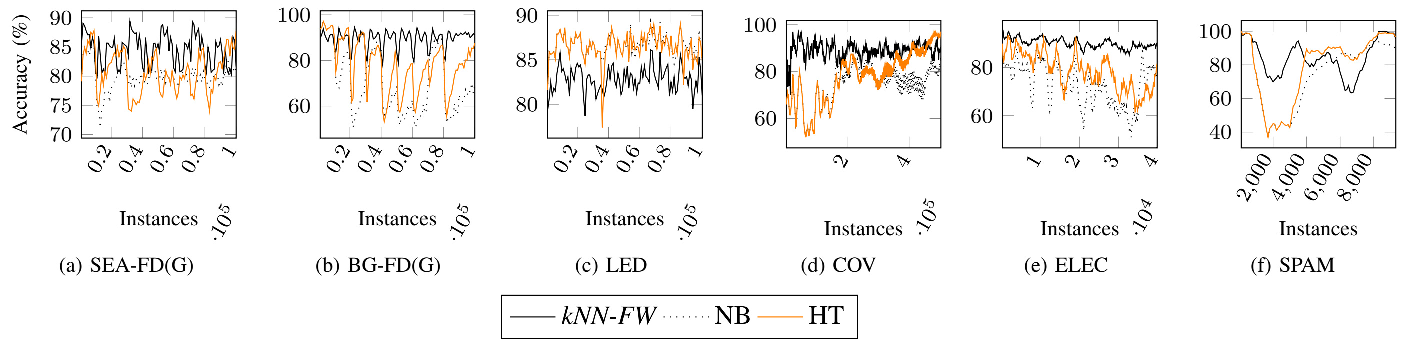

Fig. 2: Comparison of prequential accuracies (%) with different biased classifiers.

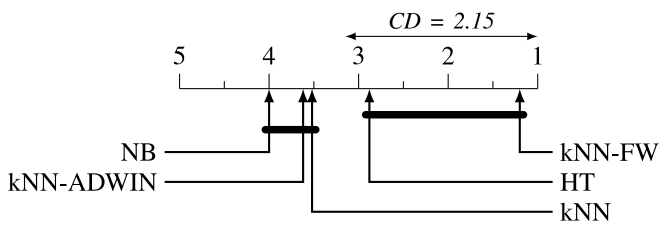

Fig. 3: Critical differences chart for accuracy comparison.

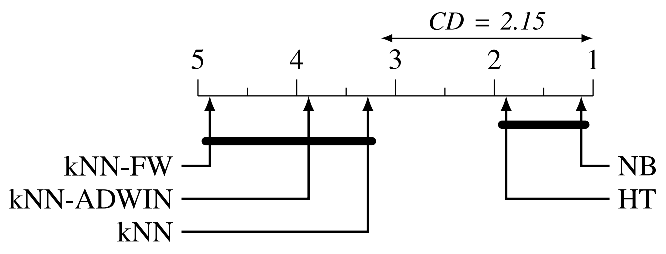

Fig. 4: Critical differences chart for CPU time comparison.

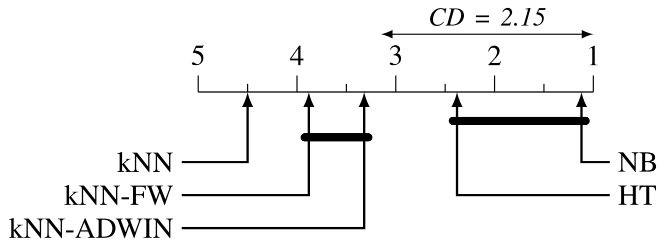

Fig. 5: Critical differences chart for RAM-Hours comparison.

[7] J. P. Barddal, H. M. Gomes, and F. Enembreck, “A survey on feature drift adaptation,” in *Proceedings of the International Conference on Tools with Artificial Intelligence*. IEEE, November 2015.

[8] G. Widmer and M. Kubat, “Learning in the presence of concept drift and hidden contexts,” *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, Apr. 1996.

[9] D. Aha and D. Kibler, “Instance-based learning algorithms,” *Machine Learning*, vol. 6, pp. 37–66, 1991.

[10] C. C. Aggarwal, Ed., *Data Classification: Algorithms and Applications*. CRC Press, 2014. [Online]. Available: http://www.crcnetbase.com/doi/book/10.1201/b17320

[11] C. Alippi, G. Boracchi, and M. Roveri, “Just in time classifiers: Managing the slow drift case,” in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, June 2009, pp. 114–120.

[12] B. Sovdat, “Updating formulas and algorithms for computing entropy and gini index on time-changing data streams,” *CoRR*, vol. abs/1403.6348, 2014. [Online]. Available: http://arxiv.org/abs/1403.6348

[13] W. N. 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*. ACM SIGKDD, Aug. 2001, pp. 377–382.

[14] A. Bifet, G. Holmes, and B. Pfahringer, “Leveraging bagging for evolving data streams,” in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, J. L. BalcÃ¡zar, F. Bonchi, A. Gionis, and M. Sebag, Eds. Springer Berlin Heidelberg, 2010, vol. 6321, pp. 135–150.

[15] M. A. Hall, “Correlation-based feature selection for discrete and numeric class machine learning,” in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML ’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 359–366. [Online]. Available: http://dl.acm.org/citation.cfm?id=645529.657793

[16] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[17] M. Harries and N. S. Wales, “Splice-2 comparative evaluation: Electricity pricing,” 1999.

[18] P. Kosina and J. Gama, “Very fast decision rules for multi-class problems,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC ’12. New York, NY, USA: ACM, 2012, pp. 795–800. [Online]. Available: http://doi.acm.org/10.1145/2245276.2245431

[19] I. Katakis, G. Tsoumakas, and I. 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*. Springer Verlag, 2006, p. 107.

[20] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive online analysis,” *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[21] A. Bifet and R. Gavaldà, “Learning from time-changing data with adaptive windowing,” in *In SIAM International Conference on Data Mining*, 2007.

[22] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’00. New York, NY, USA: ACM, 2000, pp. 71–80. [Online]. Available: http://doi.acm.org/10.1145/347090.347107

[23] 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*. ACM SIGKDD, Jun. 2009, pp. 329–338.

[24] G. W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. New Jersey: Wiley, 2009.