Data Stream Classification using Random Feature Functions and Novel Method Combinations

Jesse Read
Aalto University and HIIT
Finland
Email: jesse.read@aalto.fi

Albert Bifet
HUAWEI Noah's Ark Lab
Hong Kong
Email: bifet.albert@huawei.com

Abstract—**Data streams are being generated in a faster, bigger, and more commonplace. In this scenario, Hoeffding Trees are an established method for classification. Several extensions exist, including high-performing ensemble setups such as online and leveraging bagging. Also, k-nearest neighbours is a popular choice, with most extensions dealing with the inherent performance limitations over a potentially-infinite stream. At the same time, gradient descent methods are becoming increasingly popular, owing to the proliferation of interest and successes in deep learning. Although deep neural networks can learn incrementally, they have so far proved too sensitive to hyperparameter options and initial conditions to be considered an effective 'off-the-shelf' data streams solution. In this work, we look at combinations of Hoeffding trees, nearest neighbour, and gradient descent methods with a streaming preprocessing approach in the form of a random feature functions filter for additional predictive power. Our empirical evaluation yields positive results for the novel approaches that we experiment with, and also highlight important issues, and shed light on promising future directions in approaches to data stream classification.**

I. Introduction

There is a trend towards working with big and dynamic data sources. This tendency is clear both in real world applications and the academic literature. Many modern data sources are not only dynamic but often generated at high speed and must be classified in real time. Such contexts can be found in sensor applications (e.g., tracking and activity monitoring), demand prediction (e.g., of electricity), manufacturing processes, robotics, email, news feeds, and social networks. Real-time analysis of data streams is becoming a key area of data mining research as the number of applications in this area grows.

The requirements for a classifier in a data stream are to

- Be able to make a classification at any time
- Deal with a potentially infinite number of examples
- Access each example in the stream just once

These requirements can in fact be met by variety of learning schemes, including even batch learners (e.g., [1]), where batches are constantly gathered over time, and newer models replace older ones as memory fills up. Nevertheless, incremental methods remain strongly preferred in the data streams literature, and particularly the Hoeffding tree (HT) and its variations [2], [3], k-nearest neighbours (kNN) [4]. Support for these options is given by large-scale empirical comparisons [5], where it is also found that methods such as naive Bayes and stochastic gradient descent-based (SGD) – although naturally incremental – are relatively poor performers.

Massive **O**nline **A**nalysis (MOA) [6] is a software environment for implementing algorithms and running experiments for online learning from data streams in Java. It implements a large number of modern methods for classification in streams.

In this paper we make use of MOA's extensive library of methods to form novel combinations and further employ an extremely rapid preprocessing technique of projecting the input into a new space via random feature functions. This leads to an empirical evaluation with benchmark and state of the art methods, and a discussion of implications of the results.

Classification in data streams is a major area of research, in which Hoeffding trees have long been a favoured method. The main contribution of this paper is to show that random feature function can be leveraged by other algorithms to obtain similar or even improved performance over tree-based methods.

II. Hoeffding Tree and Extensions

Hoeffding trees [2] are state-of-the-art in classification for data streams and they predict by choosing the majority class at each leaf. However, these trees may be conservative at first and in many situations naive Bayes method outperforms the standard Hoeffding tree initially, although it is eventually overtaken [7]. A proposed hybrid adaptive method (by [7]) is a Hoeffding tree with naive Bayes at the leaves, i.e., returning a naive Bayes prediction at the leaves, if it has been so far more accurate overall than the majority class. Given it's widespread acceptance, this is the default in MOA, and we denote this method in the experimental section simply as HT. In fact, the naive Bayes classification comes for free, since it can be made with the same statistics that are collected anyway by the tree.

In this paper we adapt this methodology to deal with other classifiers in a similar way, namely kNN and an SGD-based method (rather than naive Bayes) at the leaves. We denote these cases HT-kNN and HT-SGD, respectively. For example, in HT-SGD, a gradient descent learner is employed in the leaves of each tree. Similarly to HT, predictions by the kNN and an SGD-based method are only used if they are more accurate on average than the majority class.

III. Stream Preprocessing: Random Feature Functions

Transforming the feature space prior to learning and classification is an established idea in the statistical and machine learning literature [8], for example with basis (or feature-) functions. Suppose the input instance is $\mathbf{x}$ of length d. This vector is transformed to a new space $\mathbf{z} = \phi(\mathbf{x})$ via function ϕ, creating new vector $\mathbf{z}$ of length h. Any off-the-shelf model now treats $\mathbf{z}$ as if it were the input. The functions can be either chosen suitably by a domain expert, or simply chosen to achieve a more dimensioned representation of the input. Polynomials and splines are a typical choice.

Other established examples include using principal component analysis (reviewed also in [8]) for this transformation, and also restricted Boltzmann machines (RBMs) [9]. RBMs can be seen as a probabilistic binary version of PCA, for finding higher-level feature representations. They have received widespread popularity in recent years due to their use in successful deep learning approaches. In this case, $\mathbf{z} = \phi(\mathbf{x}) = f(\mathbf{W}^\top \mathbf{x})$ for some non-linearity f: a sigmoid function is typical, but more recently rectified linear units (ReLUs, [10]) have fallen into favour. The weight matrix $\mathbf{W}$ is learned with gradient-based methods [11], and the projected output should provide a better feature representation for a neural network or any off-the-shelf method. This approach was applied to data streams already in [12], but concluded that the sensitivity to hyper-parameters and initial conditions prevented good 'out-of-the-box' deployment in data streams.

Approach such as the so-called extreme learning machines (ELMs) [13] avoid tricky parameterizations by simply using random functions (indeed, ELMs are basically linear learners on top of non-linear data transformations). ELMs are typically presented with radial basis functions, but we use ReLUs due to their recent popularity and success, and because there are fewer operations to compute them which makes them more attractive for big data. Here,

$$z_k = f(a_k) = 1_{a_k > 0}$$

where $a_k = \mathbf{W}_k^\top \mathbf{x}$ is the k-th activation function and $\mathbf{W}$ is a random $d \times h$ matrix (d input attributes, h output features), namely in our implementation $W_{j,k} \sim \mathcal{N}(0, 0.1)$. The terrain of a ReLU is exemplified in Figure 1. This process can be initialized essentially instantaneously and we could implement it in the MOA framework as a streaming filter. Thereby, for each instance $\mathbf{x}_t = [x_1, \ldots, x_d]$ at time t in a stream, a new vector $\mathbf{z}_t = [z_1, \ldots, z_h]$ can be instantly created at the cost of a vector-matrix multiplication. A classifier (e.g., HT, kNN, or SGD) learns from the stream $\mathbf{z}_t | t = 1, 2, \ldots$.

Regarding HTs with additional algorithms in the leaves (as described in Section II), this filter can either be placed before the HT, or before the method in the leaves, or both.

IV. Ensembles in Data Streams

Bagging is an ensemble method used to improve the accuracy of classifier methods. Non-streaming bagging [14] builds a set of M base models, training each model with

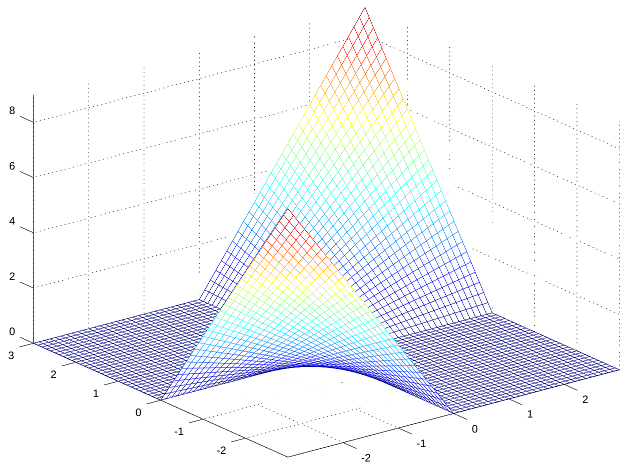


Fig. 1: Terrain of ReLU basis function on two input attributes x_1, x_2 the feature function z is given on the vertical axis.

a bootstrap sample of size N created by drawing random samples with replacement from the original training set. Each base model's training set contains each of the original training example K times where $P(K = k)$ follows a binomial distribution. This binomial distribution for large values of N tends to a Poisson($\lambda = 1$) distribution, where Poisson($\lambda = 1$)$= \exp(-1)/k!$. Using this fact, Oza and Russell [15], [16] proposed *Online Bagging*, an online method that instead of sampling with replacement, gives each example a weight according to Poisson(1). `ADWIN` Bagging [17] is an adaptive version of Online Bagging that uses a change detector to decide when to discard under-performing ensemble models.

Leveraging Bagging (LB, [3]) improves `ADWIN` Bagging, increasing the weights of this resampling using a larger value λ to compute the value of the Poisson distribution. The Poisson distribution is used to model the number of events occurring within a given time interval. It proved very competitive.

Again, we can run a filter on the input instances before entering the ensemble of trees, or at the leaves. It is even possible to run the filter again on the *output* of an ensemble (i.e., the votes), before running an additional stacking procedure. This kind of methodology can give way to rather 'deep' classifiers. Figure 2 illustrates a possible setup. In this sense of multiple levels we could also call our approach *deep* learning. It is debatable whether decision trees can be called a deep method (their levels involve partitioning an existing feature set rather than because they simple partition a space rather than create higher-level feature space). However, several of the methods we investigate have at least multiple levels of feature transformation, which is behind the power of most deep methods. In the following section we investigate the empirical performance of several novel combinations based on the methodology described so far.

V. Experiments

Among the methods we investigate (e.g., HT, kNN, SGD), different levels of filters and ensembles and possibly additional

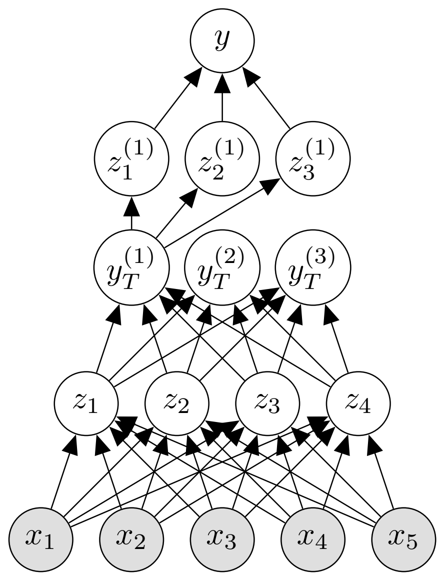


Fig. 2: An example setup: Input $\mathbf{x}$ is filtered (i.e., projected) to random layer $\mathbf{z}$ (first layer of connections), which goes to an ensemble of, for example, HTs (second layer), wherein instances are partitioned to the leaves and are again filtered (third layer) and used as training for, say, SGD, producing (in the firth layer of connections) final vote y. Note, however, that we only draw the final two layers wrt to the first of the HT models.

classification in the leaves (in the case of HT), there are a multitude of possible combinations. We first investigate the viability of random feature functions and their effect on the different classifiers (comparing these common methods with their 'filtered' versions that we denote HT-SGD, kNN-F, and SGD-F. This study led us to novel combinations, which we further compare to the benchmark methods and state-of-the-art Leveraging Bagging (LB-HT).

All experiments were carried out using the MOA framework [6] with prequential evaluation: each individual example is used to test the model before it is used for training, and from this the accuracy can be incrementally updated. We used an 8-core (3.20GHz each) desktop machine allowing up to 1 gigabyte of RAM per run (all methods were able to finish).

Table I lists the data sources used. A thorough description of most of the datasets is given in [5]. Of the others, LOC1 and LOC2 are datasets dealing with classifying the location of an object in a grid of 5×5 and 50×50 pixels respectively, described in [18]. SUSY [19] has features that are kinematic properties measured by particle detectors in an accelerator. The binary class distinguishes between a signal process which produces supersymmetric particles and a background process which does not. It is one of the largest datasets in the UCI repository that we could find.

For the feature filter we used parameters $h = 5d$ hidden units for kNN-F and $h = 10d$ for SGD-F and HT-F (a decision based on the relative computational sensitivity of kNN to a larger attribute space – for LOC2 this means 25,000 attributes

TABLE I: Data sources used in the experimetnal evaluation. Synthetic datasets are listed first.

Dataset	#Attributes	#Instances
RBF1	10	100,000
HYP1	10	100,000
LED1	24	100,000
LOC1	25	100,000
LOC2	2500	100,000
Poker	10	829,201
Electricity	8	45,312
CoverType	54	581,012
SUSY	8	5,000,000

in the projected space for SGD-F, and half of that for kNN-F) – except where this is varied in Figure 3. For kNN we used a buffer size of 5000. For LB we specify 10 models. In other cases, the default parameters in MOA are used.

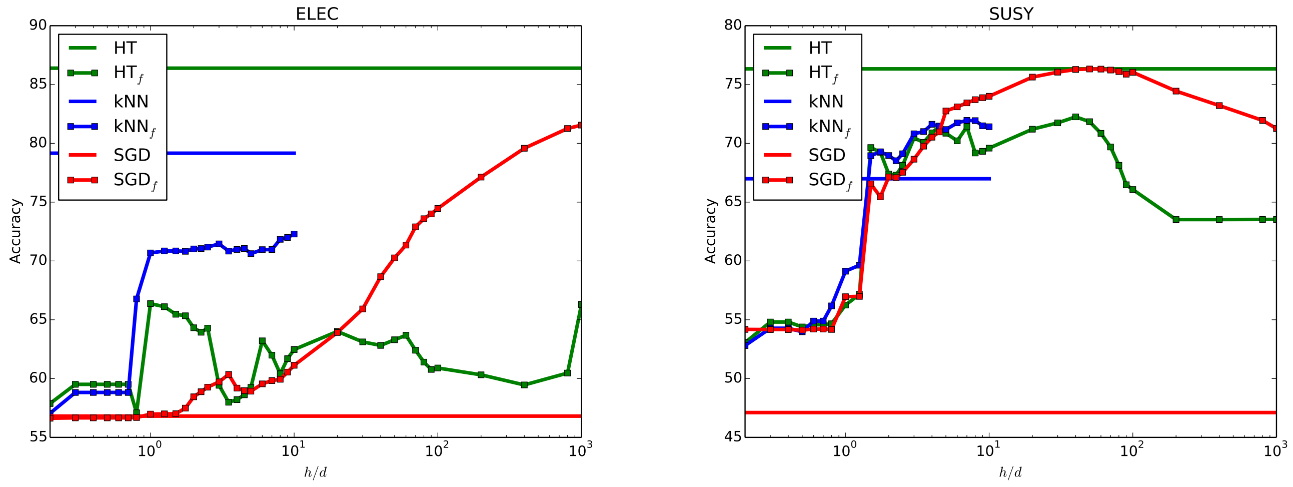
Figure 3 displays the results of varying the relative size of the new feature space (wrt to the original feature space) on two real-world datasets. Note that the feature space is different, so even when this ratio is $1:1$, performance may differ.

With regard to kNN, performance improves with more feature functions. In one of the two cases, this is sufficient to overtake kNN on the original feature space. Unfortunately, kNN is particularly sensitive to the number of attributes, so complexity becomes an issue long before other methods. The new feature space does not help the performance of HT, and in neither case does it reach the performance of HT on the original feature space. In fact, it begins to decrease again. This is because too many features makes it difficult for HT to become confident enough to split on, and may split poorly. Also, by partitioning the feature space, interaction between the features is lost. SGD reacts best to a new feature space. As noticed earlier [5], SGD is a poor performer compared to HTs, however, working in a feature space of random ReLUs, SGD-F actually reaches HT performance (on SUSY, and looks promising under ELEC) with similar time complexity. Even at 1,000 times the original feature space, running time is acceptable (only a several seconds per 10,000 instances). On the other hand, the increased memory use is significant across all methods. SGD requires 1,000 times more memory in this setting.

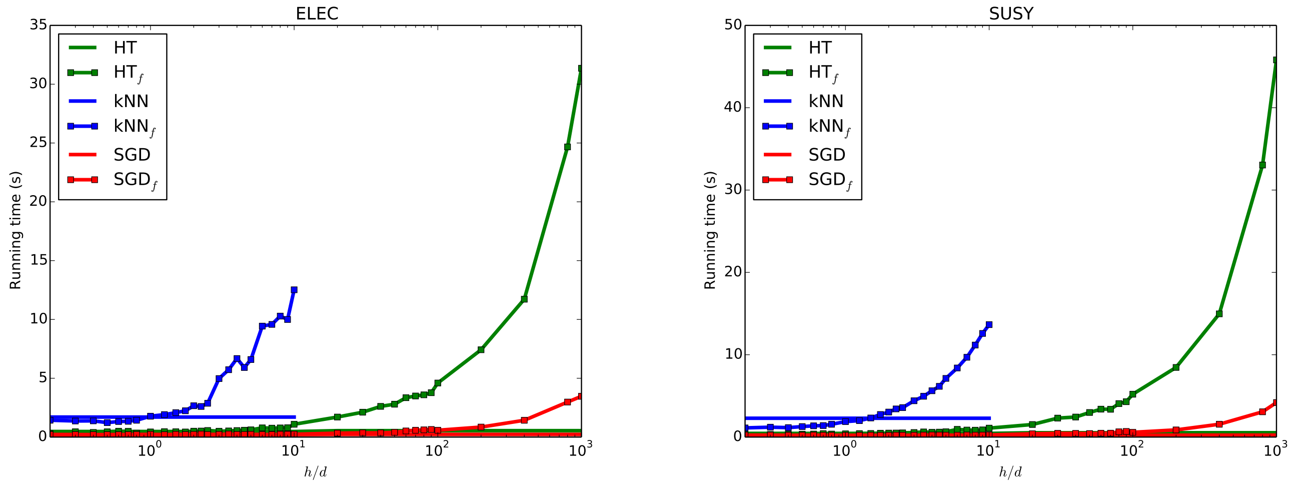
From this initial investigation we formulate several method combinations for a more extensive evaluation. Table II displays the final accuracy over the data stream. The first four columns represent the baselines and state-of-the-art (LB-HT), and remaining columns are a selection of new method combinations. Figure 4 gives a more detailed over-time view of the largest dataset (SUSY), with the average performance plotted over the entire stream over 100 intervals, and also the first 1/10th of the data (again, over 100 intervals). The second plot gives more of an idea about how models respond to fresh concepts. Learning new concepts is a fundamental part in data streams of adapting to concept drift.

Some of the most important observations and conclusions are as follows:

- SGD-F (i.e., SGD with random feature functions) should



(a) Accuracy



(b) Running Time (s)

Fig. 3: Accuracy (Figure 3a) and Running Time (Figure 3b) on a 10,000-instance segment of two real-world datasets (ELEC and SUSY) for varying proportions of h (number of hidden units / basis functions) wrt d. kNN has been cut out after $h = 1000/d$ due to scalability reasons. Note the log scale on the horizontal axis.

be considered as a baseline in data stream evaluations. Even in this first analysis, it out-competes established methods like kNN on several datasets.

- kNN benefits relatively less (than SGD) from the feature functions filter. Accuracy improved only on certain occasions, and the impact on computational performance is significant. However, on a few datasets accuracy is 5-10 percentage points higher with the filter.
- kNN can be used effectively in the leaves of HT instead of the default of naive Bayes. There is an additional computational cost involved, but results showed this to be highly competitive method – best equal overall in predictive performance tied with state-of-the-art LB-HT
- HT is difficult to improve on using feature functions (at least with the ReLUs that we experimented with). Peak accuracy is reached in relatively short space of time.
- SGD takes longer than HT or LB-HT to reach competitive accuracy, but the gap narrows significantly with more examples. On the largest datasets, the final average accuracy is within a percentage point – and this average includes initial poorer performance. Therefore, on particularly big data streams (which are increasingly common), HTs could find themselves increasingly challenged to stay ahead of these methods.
- HT-SGD-F is comparable to the state of the art LB-HT on several datasets, but demonstrates more favourable running times.
- Unlike many deep learning techniques, these random functions do not require sensitive calibration.
- Unsurprisingly, kNN-based methods perform best on the

TABLE II: Final Accuracy and Running Times. The dataset-wise ranking is given in (parentheses) and the average of these ranks is given in the final row.

(a) Accuracy

Dataset	HT	SGD	kNN	LB-HT	LB-SGD-F	kNN-F	SGD-F	HT-kNN	HT-SGD-F
RBF1	75.0 (5)	54.5 (9)	92.0 (2)	88.7 (4)	72.0 (8)	90.4 (3)	72.0 (7)	92.6 (1)	73.7 (6)
RBFD	65.7 (5)	51.3 (9)	88.6 (1)	79.5 (4)	59.8 (8)	86.3 (2)	59.9 (7)	84.9 (3)	59.9 (6)
HYP1	87.7 (1)	50.3 (9)	82.9 (4)	85.7 (2)	67.2 (7)	77.0 (5)	67.2 (7)	83.3 (3)	67.9 (6)
LED1	73.1 (1)	10.3 (9)	62.8 (3)	72.0 (2)	15.6 (6)	49.0 (5)	15.5 (7)	62.8 (3)	15.5 (7)
POKR	76.1 (6)	68.9 (9)	69.3 (8)	87.6 (1)	82.3 (2)	81.5 (4)	81.9 (3)	74.8 (7)	80.1 (5)
LOC1	85.5 (8)	80.4 (9)	91.0 (2)	90.5 (6)	90.7 (4)	88.8 (7)	90.7 (3)	91.3 (1)	90.7 (5)
LOC2	56.3 (5)	51.5 (9)	75.7 (2)	52.6 (8)	56.8 (4)	74.5 (3)	55.9 (7)	75.9 (1)	56.1 (6)
ELEC	79.2 (4)	57.6 (9)	78.4 (5)	89.8 (1)	74.8 (7)	74.2 (8)	74.8 (6)	82.5 (2)	81.8 (3)
COVT	80.3 (5)	60.7 (9)	92.2 (1)	91.7 (2)	78.7 (6)	91.6 (3)	78.7 (7)	91.2 (4)	78.3 (8)
SUSY	78.2 (3)	76.5 (7)	67.5 (9)	78.7 (1)	77.7 (4)	71.2 (8)	77.7 (5)	77.2 (6)	78.4 (2)
avg rank	4.30	8.80	3.70	3.10	5.60	4.80	5.90	3.10	5.40

(b) Running Time (s)

Dataset	HT	SGD	kNN	LB-HT	LB-SGD-F	kNN-F	SGD-F	HT-kNN	HT-SGD-F
RBF1	0 (3)	0 (1)	3 (6)	3 (5)	4 (8)	14 (9)	0 (2)	4 (7)	1 (4)
RBFD	1 (3)	0 (1)	3 (6)	2 (5)	4 (8)	15 (9)	0 (2)	4 (7)	1 (4)
HYP1	0 (2)	0 (1)	3 (6)	2 (5)	4 (7)	13 (9)	0 (3)	4 (8)	1 (4)
LED1	0 (2)	0 (1)	7 (6)	2 (5)	17 (8)	40 (9)	1 (3)	8 (7)	1 (4)
POKR	9 (2)	3 (1)	455 (8)	91 (5)	279 (6)	1539 (9)	21 (3)	422 (7)	26 (4)
LOC1	1 (2)	0 (1)	8 (7)	2 (5)	21 (8)	48 (9)	1 (3)	8 (6)	2 (4)
LOC2	9 (2)	4 (1)	1276 (7)	93 (3)	1917 (8)	2270 (9)	367 (5)	1230 (6)	350 (4)
ELEC	1 (3)	0 (1)	14 (7)	10 (6)	9 (5)	49 (9)	1 (2)	19 (8)	2 (4)
COVT	19 (2)	11 (1)	605 (6)	220 (3)	4119 (9)	3998 (8)	233 (4)	727 (7)	250 (5)
SUSY	45 (2)	25 (1)	1464 (8)	530 (5)	1040 (6)	4714 (9)	118 (3)	1428 (7)	159 (4)
avg rank	2.30	1.00	6.70	4.70	7.30	8.90	3.00	7.00	4.10

dataset RBFD which has a drifting concept, since they automatically phase out older concepts. We did not look into detail about dealing with concept drift in this paper, but this can be dealt with by 'meta methods', e.g., [20].

- Employing random feature functions as a 'filter' in the MOA framework is a convenient and flexible way to apply it in a range of different data-stream methods.

VI. Summary, Conclusions, and Future Work

Hoeffding trees and in particular their ensemble derivatives (such as leveraging bagging) can still be regarded as a state-of-the-art classifier. However, we experimented with some promising techniques, namely using k-nearest neighbour and stochastic gradient descent methods in the leaves. Performance in an empirical evaluation was encouraging. Furthermore, looked at the use of random feature functions to automatically and instantly project the input into a new space. We found that this could turn a simple gradient descent learner into a competitive method, and also provided particular benefits for kNN.

We used an ad-hoc choice of activation function (rectified linear units) and random weights. In the future we intent to look at different activation functions, and adding and pruning units incrementally in the stream over time to respond to make more efficient use of memory and adapt to drifting concepts.

Acknowledgment

This work was supported in part by the Aalto University AEF research programme http://energyefficiency.aalto.fi/en/

References

[1] W. Qu, Y. Zhang, J. Zhu, and Q. Qiu, "Mining multi-label concept-drifting data streams using dynamic classifier ensemble," in *Asian Conference on Machine Learning*, ser. Lecture Notes in Computer Science, vol. 5828. Springer, 2009, pp. 308–321.

[2] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 71–80.

[3] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *ECML PKDD'10*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 135–150.

[4] A. Shaker and E. Hüllermeier, "Instance-based classification and regression on data streams," in *Learning in Non-Stationary Environments*. Springer New York, 2012, pp. 185–201.

[5] J. Read, A. Bifet, B. Pfahringer, and G. Holmes, "Batch-incremental versus instance-incremental learning in dynamic and evolving data," in *11th Int. Symposium on Intelligent Data Analysis*, 2012.

[6] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive Online Analysis http://moa.cs.waikato.ac.nz/," *Journal of Machine Learning Research (JMLR)*, 2010.

[7] G. Holmes, R. Kirkby, and B. Pfahringer, "Stress-testing Hoeffding trees," in *PKDD*, 2005, pp. 495–502.

[8] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

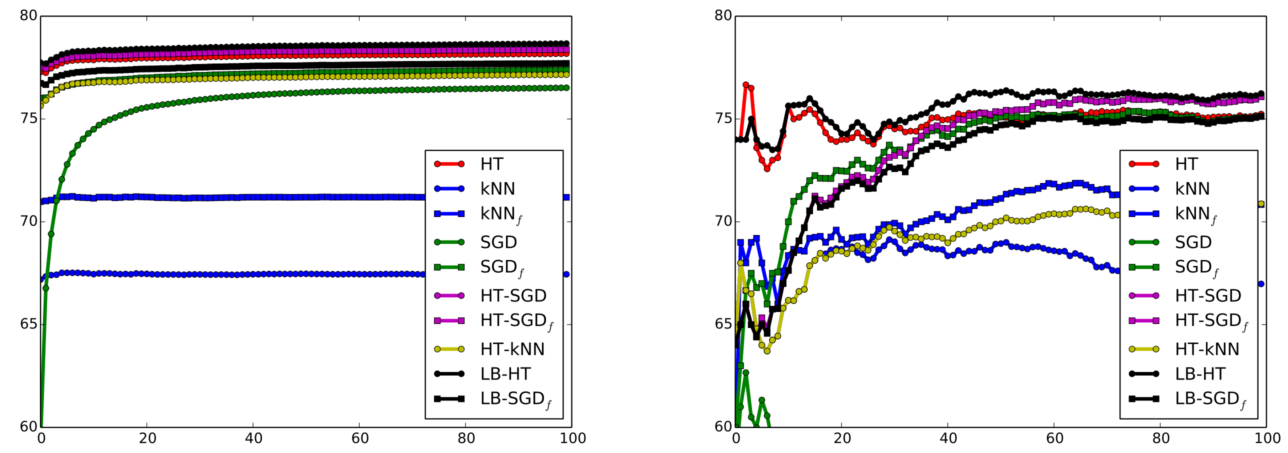
[9] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504 – 507, 2006.

[10] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

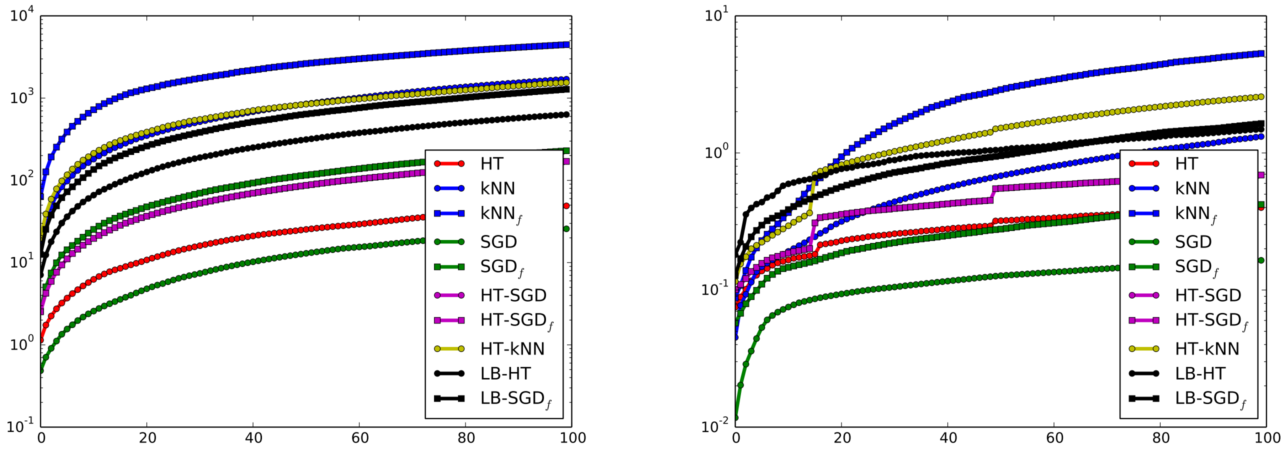
[11] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1711–1800, 2000.

[12] J. Read, F. Perez-Cruz, and A. Bifet, "Deep learning in multi-label data-streams," in *Symposium on Applied Computing*. ACM, April 2015.

[13] G.-B. Huang, D. Wang, and Y. Lan, "Extreme learning machines: a survey," *International Journal of Machine Learning and Cybernetics*,



(a) Accuracy



(b) Running Time (s)

Fig. 4: Performance over all 5,000,000 instances (left) and first 50,000 examples (right) of the SUSY data, in each divided into 100 windows.

vol. 2, no. 2, pp. 107–122, 2011. [Online]. Available: http://dx.doi.org/10.1007/s13042-011-0019-y

[14] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[15] N. C. Oza and S. J. Russell, “Experimental comparisons of online and batch versions of bagging and boosting,” in *KDD*, 2001, pp. 359–364.

[16] N. Oza and S. Russell, “Online bagging and boosting,” in *Artificial Intelligence and Statistics 2001*. Morgan Kaufmann, 2001, pp. 105–112.

[17] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, “New ensemble methods for evolving data streams,” in *KDD*, 2009, pp. 139–148.

[18] J. Read and J. Hollmén, “A deep interpretation of classifier chains,” in *Advances in Intelligent Data Analysis XIII - 13th International Symposium, IDA 2014*, October 2014, pp. 251–262.

[19] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for exotic particles in high-energy physics with deep learning,” *Nature Communications*, vol. 5, no. 4308, 2014.

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