ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

Enhash: A Fast Streaming Algorithm For Concept Drift Detection

Aashi Jindal[1], Prashant Gupta[1], Debarka Sengupta[2,3] and Jayadeva[1]

1- Indian Institute of Technology Delhi - Department of Electrical Engineering
Hauz Khas, Delhi 110016, India
2- Indraprastha Institute of Information Technology -
Department of Computer Science and Engineering, Delhi 110020, India
3- Queensland University of Technology - Institute of Health and Biomedical Innovation
Brisbane, QLD 4000, Australia

Abstract. We propose Enhash, a fast ensemble learner that detects *concept drift* in a data stream. A stream may consist of abrupt, gradual, virtual, or recurring events, or a mixture of various types of drift. Enhash employs projection hash to insert an incoming sample. Benchmark tests on 6 artificial and 4 real data sets consisting of various types of drift show that Enhash is competitive with state-of-the-art ensemble learners while being significantly faster. It also has moderate resource requirements.

1 Introduction

A data stream environment is often characterized by large volumes of data that flow rapidly and continuously. They are processed in an *online* fashion to accommodate data that cannot reside in main memory. A streaming data environment is commonly used for tasks such as making recommendations for users on streaming platforms, and real-time analysis inside IoT devices. In such a stream, the underlying data distribution may change, and this phenomenon is referred to as *concept drift*. Formally, the posterior probability of a sample's class changes with time. Consequently, the method must also be able to adapt to the new distribution. To adapt to a new *concept*, the method may require supplemental or replacement learning. Tuning a model with new information is termed as supplemental learning. Replacement learning refers to the case when the model's old information becomes irrelevant, and is replaced by new information. A shift in the likelihood of observing a data point x within a particular class when class boundaries are altered, is called *real concept drift*. *Concept drift* without an overlap of true class boundaries, or an incomplete representation of the actual environment, is referred to as *virtual concept drift*. In *virtual concept drift*, one requires supplemental learning, while *real concept drift* requires replacement learning [1]. The other common way to categorize *concept drift* is determined by the speed with which changes occur [2]. Hence, drift may be *incremental*, *abrupt* or *gradual*. A *reoccurring drift* is one that emerges repeatedly. Thus, in order to handle *concept drift*, a model must be adaptive to non-stationary environments.

Several methods have been recently proposed to handle *concept drift* in a streaming environment. The most popular of these are ensemble learners [3, 4, 5, 6, 7, 8]. As the data stream evolves, an ensemble method selectively retains a few learners to maintain prior knowledge while discarding and adding new learners to learn new knowledge.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

Thus, an ensemble method is quite flexible, and maintains the *stability-plasticity* balance i.e. retaining the previous knowledge (*stability*) and learning new concepts (*plasticity*).

In this paper, we propose *Enhash*, an ensemble learner that employs projection hash to handle *concept drift*. For incoming samples, it generates a hash code such that similar samples tend to hash into the same bucket. A gradual forgetting factor weights the contents of a bucket. Thus, the contents of a bucket are relevant as long as the incoming stream belongs to the *concept* represented by them.

2 The Proposed Method

We propose Enhash, an ensemble learner, that employs hashing for *concept drift* detection. Let $x_t \in \mathbb{R}^d$ represent a sample from a data stream S at time step t and let $y \in \{1, 2, ..., C\}$ represent its corresponding *concept*, where C is the total number of *concepts*. Further, let us assume a family of hash functions H such that $\forall h_l \in H$, it maps x_t to an integer value. The hash code $h_l(x_t)$ is assigned to x_t by hash function h_l. A bucket is a set of samples with the same hash code; both these terms are used interchangeably. The total number of samples in bucket $h_l(x_t)$ is denoted by $N_{h_l(x_t)}$. Further assume that N samples have been seen and hashed from the stream so far. Amongst N, N_c samples belong to the *concept class* c such that $\sum_{c=1}^{c=C} N_c = N$. Based on the evidence from the data stream seen so far, the probability of bucket $h_l(x_{t+1})$ is given by $p(h_l(x_{t+1})) = \frac{N_{h_l(x_{t+1})}}{N}$ and prior for class c is given by $p(c) = N_c/N$. Assuming, $(N_{h_l(x_{t+1})})_c$ represents the samples of *concept* c in bucket $h_l(x_{t+1})$. Hence, the likelihood of x_{t+1} belonging to *concept* c in bucket $h_l(x_{t+1})$ is given by $p(h_l(x_{t+1})|c) = \frac{(N_{h_l(x_{t+1})})_c}{N_c}$ and the probability of x_{t+1} belonging to class c is given by

$$p(c|h_l(x_{t+1})) = \frac{p(h_l(x_{t+1})|c)p(c)}{p(h_l(x_{t+1}))} = \frac{(N_{h_l(x_{t+1})})_c}{N_{h_l(x_{t+1})}} \tag{1}$$

Equation (1) is simply the normalization of counts in bucket $h_l(x_{t+1})$.

To predict the *concept class* of x_{t+1}, an ensemble of L such hash functions can be employed and the weight for each *concept class* is computed as

$$\hat{p}_c = \sum_{l=1}^{l=L} \log\Big(1 + p(c|h_l(x_{t+1}))\Big) \tag{2}$$

and the *concept class* is predicted as

$$\hat{y} = \arg \max_{c \in \{1,..,C\}} \hat{p}_c \tag{3}$$

To accommodate an incoming sample of class c, the bucket is updated as

$$(N_{h_i(x_{t+1})})_c = 1 + (N_{h_i(x_{t+1})})_c \; \forall c \in \{1, .., C\} \tag{4}$$

Enhash utilizes a simple strategy as described above to build an ensemble learner for *concept drift* detection.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

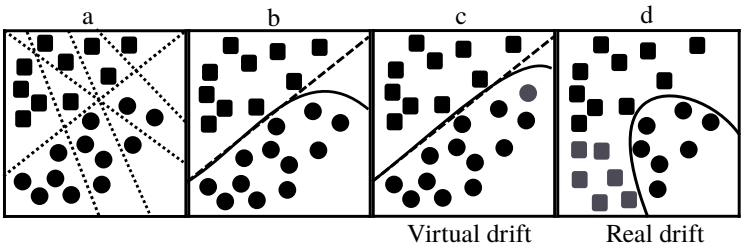

Fig. 1: Enhash accommodates both virtual and real drift.

In Enhash, the projection hash family is selected as a base learner. Here, a hash function involves the dot product of hyperplane $w^{(l)}$ and sample x_t. It is defined as

$$h_l(x_t) = \lfloor \frac{1}{\textit{bin-width}} \left(\sum_{j=1}^{j=d} \left(w_j^{(l)} * (x_t)_j \right) + \text{bias}^{(l)} \right) \rfloor \tag{5}$$

where, *bin-width* is quantization width, $w_j^{(l)}$ and $\text{bias}^{(l)}$ can be sampled from any desired distribution. In our implementation, $w_j^{(l)} \sim N(0,1)$ and
$\text{bias}^{(l)} \sim [-\textit{bin-width}, \textit{bin-width}]$.

Effectively, each h_l (5) divides the space into equally spaced unbounded regions of size *bin-width* (earlier referred to as bucket). Equation (1) computes the probability of each *concept class* in a region. An ensemble of hash functions makes all the regions bounded. The weight of a *concept class* in the bounded region is computed using (2). An absolute value of *concept class* is assigned to each region in (3). Figure 1a shows the arrangement of randomly generated hyperplanes. The solid line in Figure 1b shows the inferred decision boundary (learned distribution) after an absolute assignment of *concept class* to every bounded region. The dashed line in Figure 1b depicts the true decision boundary (true distribution).

Assuming at time step t, Figure 1b shows the current stage of learner. At step $t+1$, a new sample x_{t+1} arrives (gray sample in Figure 1c). After updating the bucket (4), the learned distribution shifts and moves towards the true distribution. Hence, Enhash accommodates virtual drift present in the data stream.

Figure 1d depicts the real drift when the true distribution evolves. This requires forgetting some of the previously learned information. Suppose that sample $x_{t+\Delta t}$ hashes to bucket $h_l(x_{t+\Delta t})$ at time say, $t + \Delta t$. Assume that previously, x_t from a different *concept*, was hashed to this bucket. In order to accommodate forgetting (6), Enhash employs a decay factor multiplier to $p(c|h_l(x_{t+\Delta t}))$ (2) while weighting a bounded region.

$$\hat{p}_c = \sum_{l=1}^{l=L} \log \left(1 + 2^{-\lambda \Delta t} p(c|h_l(x_{t+\Delta t})) \right) \tag{6}$$

where λ is the decay rate. The update rule for the bucket (3) is also changed to reflect

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

the new *concept class* as $(N_{h_l(x_{t+1})})_c = 1 + 2^{-\lambda\Delta t}(N_{h_l(x_{t+1})})_c$. Setting $\lambda = 0$ will reduce these equations to the base case.

In order to break ties in $p(c|h_l(x_t))$, the class weight in the region is also weighted by the distance of the sample x_t from the mean of class samples in bucket $h_l(x_t)$, i.e. $\text{mean}^c_{h_l(x_t)}$

$$\text{dist}_c(x_t) = \sqrt{\sum_{i=1}^{i=d} \Big((x_t)_i - (\text{mean}^c_{h_l(x_t)})_i\Big)^2}$$

$$\hat{p}_c = \sum_{l=1}^{l=L} \log\Big(1 + \frac{2^{-\lambda\Delta t} p(c|h_l(x_t))}{dist_c(x_t)})\Big)$$

and remaining ties are broken arbitrarily. Here, $(x_t)_i$ denotes the i-th feature of sample x_t.

3 Experimental Setup

Enhash's performance was compared with some of the widely used ensemble learners. These include Learn^{++}.NSE [3], Accuracy-Weighted Ensemble (AWE) [7], Additive Expert Ensemble (AEE) [8], Online Bagging-ADWIN (OB) [9], Leveraging Bagging (LB) [5], Online SMOTE Bagging (OSMOTEB) [4], ARF [6], Online CSB2 [4], Online RUSBoost [4], and Streaming Random Patches (SRP) [10]. The implementation of these methods is available in `scikit-multiflow` `python` package [11]. A fixed value of *estimators* was used for all the methods. For methods AWE, Learn^{++}.NSE, LB, and OB, the maximum size of window was set $\min(5000, 0.1 * n)$, where n is the total number of samples. For the rest of the parameters, the default value was used for all the methods.

4 Experimental Results

For all methods, the number of $estimators$ is considered as 10. In addition, for Enhash the *bin-width* was set to $\{0.1, 0.01\}$ and λ was set to 0.015. Table 1 compares the performances in terms of error and KappaM using Interleaved Test-Train strategy. For these measures, the performance of the proposed method was superior to AWE, AEE, and Online RUSBoost on 8 data sets, Online CSB2, Learn^{++}.NSE, OSMOTEB, and OB on 7 data sets, Learn++ and LB on 6 data sets, ARF on 5 data sets, and SRP on 4 data sets. The performance of Enhash supersedes all other methods for 3 data sets.

Other evaluation criteria are speed and RAM-hours (Table 2). For all data sets, Enhash takes the least time. In terms of speed, Learn^{++}.NSE is the closest competitor of Enhash. In terms of accuracy, however, Enhash supersedes Learn^{++}.NSE on the majority of the data sets. Our findings were also consistent for error and KappaM. The overall closest competitors of Enhash in terms of error and KappaM are SRP, ARF, LB, and OB. Enhash's speed and RAM-hours' requirement are almost insignificant when

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7. Available from http://www.i6doc.com/en/.

Dataset	Learn^{++}.NSE	LB	OB	OSMO-TEB	AWE	AEE	ARF	Enhash	Online CSB2	Online RUSBoost	SRP
(a) Error (in %)											
transientChessboard	**3.67**	13.13	29.62	24.28	89.78	85.37	27.56	18.84	19.80	23.95	39.53
rotatingHyperplane	24.13	24.65	**15.08**	19.75	16.27	18.10	16.40	32.72	18.16	28.06	17.69
mixedDrift	39.46	16.71	23.63	20.30	77.79	81.17	19.79	**12.88**	18.78	16.80	19.69
movingSquares	68.74	55.69	65.42	61.12	67.51	67.23	58.54	**13.29**	64.43	35.34	19.40
interchangingRBF	75.43	4.61	37.67	22.05	83.49	82.58	3.22	2.72	3.66	97.37	**2.69**
interRBF20D	76.36	5.62	37.02	21.00	84.90	82.35	**2.39**	11.30	3.48	95.73	2.92
airlines	42.92	43.36	39.23	40.66	38.49	39.17	33.09	41.66	43.04	44.41	**32.62**
elec2	34.63	18.64	23.26	24.60	39.64	26.71	**11.59**	17.34	22.94	18.95	11.81
NEweather	29.20	27.20	**20.84**	22.74	30.72	30.78	21.43	29.27	23.27	29.20	21.23
outdoorStream	-	9.33	34.15	21.25	78.55	42.45	26.12	**8.75**	33.85	62.73	34.62
(b) KappaM											
transientChessboard	**0.96**	0.85	0.66	0.72	-0.03	0.02	0.68	0.78	0.77	0.73	0.55
rotatingHyperplane	0.52	0.51	**0.70**	0.60	0.67	0.64	0.67	0.34	0.64	0.44	0.65
mixedDrift	0.54	0.80	0.72	0.76	0.09	0.05	0.77	**0.85**	0.78	0.80	0.77
movingSquares	0.08	0.26	0.13	0.19	0.10	0.10	0.22	**0.82**	0.14	0.53	0.74
interchangingRBF	0.18	0.95	0.59	0.76	0.09	0.10	0.96	**0.97**	0.96	-0.06	**0.97**
interRBF20D	0.17	0.94	0.60	0.77	0.08	0.10	**0.97**	0.88	0.96	-0.04	**0.97**
airlines	0.04	0.03	0.12	0.09	0.14	0.12	0.26	0.06	0.03	0.00	**0.27**
elec2	0.18	0.56	0.45	0.42	0.07	0.37	**0.73**	0.59	0.46	0.55	0.72
NEweather	0.07	0.13	**0.34**	0.28	0.02	0.02	0.32	0.07	0.26	0.07	0.32
outdoorStream	-	0.90	0.65	0.78	0.19	0.56	0.73	**0.91**	0.65	0.36	0.64

Table 1: For a given data set, the method with the best metric value is in boldface. Due to implementation constraint, Learn^{++}.NSE could not run for the outdoorStream data set.

Dataset	Learn^{++}.NSE	LB	OB	OSMO-TEB	AWE	AEE	ARF	Enhash	Online CSB2	Online RUSBoost	SRP
Time (in hrs)											
transientChessboard	0.287	8.374	0.983	13.623	0.181	0.570	0.481	**0.099**	1.081	1.008	1.238
rotatingHyperplane	0.199	14.022	6.957	114.615	0.198	0.660	1.318	**0.067**	2.917	4.109	1.635
mixedDrift	0.627	27.408	7.916	185.501	1.673	17.149	1.875	**0.339**	5.708	4.428	5.456
movingSquares	0.179	9.015	4.180	28.028	0.142	0.398	7.082	**0.068**	1.641	2.202	0.812
interchangingRBF	0.185	8.298	1.106	7.424	0.376	1.048	0.411	**0.132**	1.411	1.346	2.005
interRBF20D	0.166	19.167	1.292	7.604	2.091	2.373	2.940	**0.106**	1.757	1.447	8.569
airlines	0.571	35.839	13.088	403.378	0.744	4.357	2.405	**0.182**	6.553	8.906	3.388
elec2	0.020	7.575	1.777	14.511	0.016	0.059	0.171	**0.015**	0.585	0.871	0.306
NEweather	0.008	0.590	0.183	1.326	0.013	0.020	0.080	**0.006**	0.219	0.282	0.140
outdoorStream	-	0.069	0.093	0.162	0.047	0.129	0.067	**0.004**	0.191	0.182	0.293
RAM-hours											
transientChessboard	8.1e-05	8.0e-02	3.3e-04	5.5e-01	2.3e-04	**5.5e-05**	1.2e-03	4.4e-04	7.5e-04	7.7e-04	3.3e-04
rotatingHyperplane	**5.6e-05**	3.5e-01	1.7e-01	4.6e+01	5.8e-04	1.1e-04	3.4e-02	7.8e-05	1.7e-02	2.4e-02	1.2e-01
mixedDrift	**4.9e-04**	2.7e-01	2.9e-02	3.9e+01	2.5e-03	2.9e-03	1.3e-02	5.3e-03	1.3e-02	2.1e-03	2.3e-03
movingSquares	4.9e-05	8.7e-02	3.9e-02	2.3e+00	1.6e-04	2.3e-05	3.0e+00	**1.2e-05**	4.8e-03	6.4e-03	1.4e-03
interchangingRBF	**5.2e-05**	8.2e-02	6.1e-04	2.4e-01	5.6e-04	1.8e-04	7.6e-04	9.9e-05	1.1e-03	6.1e-04	6.9e-04
interRBF20D	**1.7e-04**	7.8e-01	2.4e-03	1.1e+00	1.8e-02	3.3e-03	2.5e-03	1.0e-03	3.5e-03	1.5e-03	5.8e-01
airlines	1.3e-03	6.9e-01	2.4e-01	3.0e+02	1.7e-03	**5.2e-04**	9.3e-02	1.6e-01	3.1e-02	4.2e-02	1.7e-01
elec2	**3.7e-06**	1.4e-01	3.3e-02	1.3e+00	3.2e-05	8.0e-06	1.4e-03	1.8e-05	2.7e-03	4.4e-03	5.5e-03
NEweather	**7.0e-07**	4.6e-03	1.4e-03	3.1e-02	1.2e-05	2.6e-06	1.2e-03	1.7e-04	1.1e-03	1.3e-03	4.5e-03
outdoorStream	-	2.6e-04	1.4e-03	6.0e-03	1.4e-04	4.5e-04	9.4e-05	**3.4e-05**	5.8e-03	3.2e-03	6.5e-04

Table 2: (a) The method with the fastest speed is highlighted for every data set. (b) The memory consumption is measured in terms of RAM-hours. The method with the least value of RAM-hours is highlighted for every data set.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

compared with other methods. OSMOTEB is the slowest when compared with all other methods. The remaining methods AWE and AEE have inadequate performances as compared to Enhash.

In summary, LB and ARF suffer in detecting incremental drifts. However, LB and ARF have an overall satisfactory performance. Enhash performs relatively well on all data sets. Enhash has a superior performance on a data set consisting of three different kinds of drifts, namely incremental, virtual, and abrupt drifts.

5 Conclusions

We conclude that Enhash supersedes other methods in terms of speed since the algorithm effectively requires only $\mathcal{O}(1)$ running time for each sample on a given estimator. In addition, the performance of Enhash in terms of error and KappaM is better or comparable to others for majority data sets. These data sets constitute abrupt, gradual, virtual, and reoccurring drift phenomena. The closest competitor of Enhash in terms of performance is ARF and SRP. Notably, Enhash requires, on an average 10 times lesser RAM-hours than that of ARF and SRP.

References

[1] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

[2] Roberto Souto Maior de Barros and Silas Garrido T de Carvalho Santos. An overview and comprehensive comparison of ensembles for concept drift. *Information Fusion*, 52:213–244, 2019.

[3] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[4] Boyu Wang and Joelle Pineau. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3353–3366, 2016.

[5] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 135–150. Springer, 2010.

[6] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

[7] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235. AcM, 2003.

[8] Jeremy Z Kolter and Marcus A Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd international conference on Machine learning*, pages 449–456. ACM, 2005.

[9] Nikunj C Oza. Online bagging and boosting. In *2005 IEEE international conference on systems, man and cybernetics*, volume 3, pages 2340–2345. Ieee, 2005.

[10] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. Streaming random patches for evolving data stream classification. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 240–249. IEEE, 2019.

[11] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5, 2018.