Deep Learning in Partially-labeled Data Streams

Jesse Read
Aalto University and HIIT
Helsinki, Finland
jesse.read@aalto.fi

Fernando Perez-Cruz
Univ. Carlos III de Madrid
Madrid, Spain
fernando@tsc.uc3m.es

Albert Bifet
Huawei Noah's Ark Lab
Hong Kong
bifet.albert@huawei.com

ABSTRACT

Of the considerable research on data streams, relatively little deals with classification where only some of the instances in the stream are labeled. Most state-of-the-art data-stream algorithms do not have an effective way of dealing with unlabeled instances from the same domain. In this paper we explore deep learning techniques that provide important advantages such as the ability to learn incrementally in constant memory, and from unlabeled examples. We develop two deep learning methods and explore empirically via a series of empirical evaluations the application to several data streams scenarios based on real data. We find that our methods can offer competitive accuracy as compared with existing popular data-stream learners.

1. INTRODUCTION

There is a trend towards working with dynamic data, both in the real world 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, monitoring (e.g., electricity) demand, 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.

On static datasets (where all training data is available before training commences), classifiers such as support vector machines (SVMs) have been very popular on account of their good performance. However, as these algorithms are not inherently incremental, they have not proven to be a competitive and viable data-stream learner. Two main approaches have surfaced in the literature. Some authors have adapted existing methods to the incremental setting, such as very fast decision trees (VFDT), e.g., [8] and lazy learners like k-nearest neighbors (kNN), e.g., [16]. Other authors have developed 'batch-incremental' methods to allow the application of methods like SVMs to data streams; where, generally, a new model is built when new data is available, and phasing out old models when memory fills up (the size each batch must be determined automatically or ad-hoc via a user parameter), as in [13].

VFDTs have often been considered the state of the art on many streams, [2], as are fast and accurate, although can require many (labeled) instances to build a good model and often require explicit detection of concept drift [14]. Likewise, kNN has also been shown to perform well in data-streams [16, 14], although as a lazy method, it can only store a finite 'window' of examples. Batch-incremental SVMs and Decision Trees only perform well on certain datasets with a well-chosen window size. It is notable that naturally incremental methods, like Naive Bayes and Neural Networks (assuming an online learning mechanism) do not usually feature in the data-stream literature as high-performing classifiers. We are not aware of any large-scale empirical study where these classifiers have out-competed with modern VFDTs or kNN approaches over a range of domains in a data-stream setting.

We note that the most renown methods up till now have focused primarily on dealing with high-speed data and detecting drift, under the fully-supervised scenario of a fully-labeled stream. The case of a partially-labeled stream has not been thoroughly addressed in data-stream literature (except a notable recent exceptions by [10, 12, 3] – which we discuss in Section 3). Although there are scenarios where the true label of each example is always known after prediction (such as predicting the weather, or electricity demand), in many contexts, labeled examples are expensive to obtain en masse (particularly when examples are labeled manually, such as e-mail and other textual domains), whereas a data-stream by definition has many data-instances arriving over time.

In recent years, in the static setting, neural networks have attracted renewed interest in the form of *deep learning* [6]; where layers of hidden units are stacked upon each other to model patterns in the data. These deep methods can outcompete SVMs in many problems. In particular with regard to data streams, we note important advantages; namely, they a) work with constant memory, b) can easily learn from unlabeled instances, and c) can obtain high accuracy

This paper investigates if deep learning can offer a new state of the art for data streams. We employ two deep learning methods for data-stream classification, and carry out a series of experiments – both in the context of a partially and fully-labeled stream. Our main contribution is 1) to improve the performance of several popular (and high-performing) data-stream methods, and 2) revive the status

April 13 - 17, 2014, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695871.

of neural network classifiers in data-stream classification.

2. DEEP LEARNING IN DATA STREAMS

A data instance $\mathbf{x}$ comes from some space $\mathbb{R}^d$ and can be represented by a vector of length d; i.e., $\mathbf{x} = [x_1, \ldots, x_d]$. Let us assume a data stream of instances $\mathbf{x}_t | t = 1, \ldots$ (of unknown length). Note we use the subscript of a vector to denote the step in time, and the subscript of a value to indicate a particular value of a vector (e.g., $x_k | 1 < k < d$).

The task of data-stream learning is to learn some classifier h to provide a classification $\hat{\mathbf{y}}_{t+1}$ for each $\mathbf{x}_{t+1}$:

$$\hat{\mathbf{y}}_{t+1} = h(\tilde{\mathbf{x}}_{t+1})$$

This classification is often a single class $\hat{y} \in \{1, \ldots, C\}$, but we generalise to the multi-task/multi-dimensional case of multiple target variables $\hat{\mathbf{y}} \in \mathbb{N}_+^k$, each represented as vector $\hat{\mathbf{y}} = [y_1, \ldots, y_k]$.

Posterior to classification (time step t) the *true* classification of $\mathbf{x}_t$ may be provided: $\mathbf{y}_t$. Note that we denote time step $t+1$ for testing time and time step t for update time. At time step t a data-stream learner uses each pair $(\mathbf{x}_t, \mathbf{y}_t)$ as a training example to update its model. In the fully supervised case, the true classification $\mathbf{y}$ is *always* made available posterior to classification. In the semi-supervised / partially-labeled case, only a subset of $t = 1, \ldots$ have true labels; thus for some $\mathbf{x}_t$ there is no ground-truth classification available.

If we removed the unlabeled examples from the stream, the problem would be reduced to that of a standard fully-labeled data stream. But these unlabeled examples may contain useful information for learning. In this work we concern ourselves with both fully labeled and partially-labeled data streams and, in latter case, we improve accuracy by making use of all instances, whether they have labels or not.

2.1 Restricted Boltzmann Machines (RBMs)

A Boltzmann machine is a type of neural network that can be used to discover the underlying regularities of the training data. The *restricted* Boltzmann machine setting (RBM) [6] deals with two layers of features, a visible layer (the original feature-attribute space) and a hidden layer. Units (i.e., feature attributes) are fully connected between layers but unconnected *within* layers, making this setting tractable to larger numbers of units.

Figure 1 shows a graphical representation of an RBM. Given instances from some feature space, an RBM learns to project such instances into a new 'hidden' feature space. The visible units take the same values as the original instances $\mathbf{x} = [x_1, \ldots, x_d]$, and from these the RBM produces corresponding values for u hidden units $\mathbf{z} = [z_1, \ldots, z_u]$ (usually $u < d$ to create an information bottleneck). Each visible unit x_i is connected to every hidden unit z_j via a weight w_{ij}, and vice versa[1]. The RBM learns these weights. This process is completely *unsupervised*, i.e., the class label has not been considered.

The hidden variables can provide a compact representation of the underlying patterns and structure of the input. An RBM can capture 2^u input space regions, whereas standard clustering requires $O(2^u)$ parameters and examples to capture this much complexity. Ideally, the learned

[1]For clarity, we do not include bias units

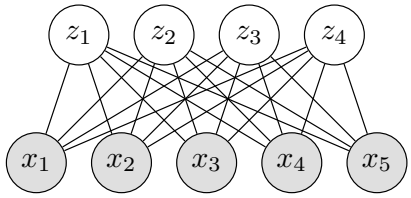


Figure 1: An RBM with $d = 5$ input units and $h = 4$ hidden units. Each edge is associated with a weight w_{ij}.

RBM would produce hidden variables that correspond directly (deterministically) to the label variables, and thus we could recover the label vector directly given any input vector. This is seldom the case, but we should expect the hidden layer of data to be more closely related to the labels than the original data (and hence be able to learn from it more easily).

RBMs are energy-based models, where the joint probability of visible ($\mathbf{x}$) and hidden units ($\mathbf{z}$) is proportional to the energy between them:

$$P(\mathbf{x}, \mathbf{z}) \propto e^{-E(\mathbf{x}, \mathbf{z})}.$$

Hence, by manipulating the energy E we can in turn generate the probability $P(\mathbf{x}, \mathbf{z})$. Specifically, we minimize the energy (i.e., find low energy states for the model) which maximizes the joint probability of the visible and hidden units. This is done by learning a weight matrix $\boldsymbol{W}$, since

$$E(\mathbf{x}, \mathbf{z}) = -\mathbf{x}\boldsymbol{W}\mathbf{z}.$$

We use *contrastive divergence* [5] to learn $\boldsymbol{W}$, an algorithm algorithmically similar to gradient descent and, as such, we can run it incrementally, one instance at-a-time. The update rule for $\boldsymbol{W}$ is:

$$\boldsymbol{W} \leftarrow \lambda \boldsymbol{W} \mathsf{cd}(\mathbf{x}_t) \tag{1}$$

where $\mathsf{cd}(\mathbf{x}_t)$ is the contrastive divergence of instance $\mathbf{x}_t$ and λ is the learning rate.

2.2 Deep Belief Networks (DBNs)

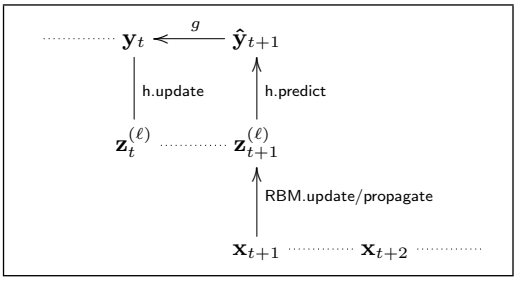
RBMs can be stacked on top of each other to form so-called DBNs [6]. This 'deep' nature provides important advantages in learning the underlying patterns of the data. The RBMs are trained greedily (and in the data-stream case, can be trained incrementally): the first RBM takes instances $\mathbf{x}_t$ from the input space and produces outputs $\mathbf{z}_t^{(1)}$, then the second RBM takes inputs $\mathbf{z}_t^{(1)}$ as the inputs, and produces $\mathbf{z}_t^{(2)}$, and so on and so forth until $\mathbf{z}_t^{(\ell)}$ (for ℓ layers).

Again, we point out that these deep Boltzmann machines function in an unsupervised and incremental fashion using only the input space. The final phase is to carry out supervised learning (whenever labels are available). In this work we use two strategies for this, to carry out learning and classification in a data-stream context.

`DBN-`h.

Many discriminative supervised methods can learn well simply by treating the top layer output as inputs [7]. Thus, we can build any data-stream model h directly from the top-layer instances $\mathbf{z}_t^{(\ell)} | t = 1, \ldots$ as if they were the original input space, with the labels $\mathbf{y}_t | t = 1, \ldots$ to create model. The DBN is trained using all $\mathbf{x}_t$, and when a $\mathbf{y}_t$ is available,

Figure 2: DBN-h: A data stream learner h learns from all $(\mathbf{z}_1^{(2)}, \mathbf{y}_1), \ldots, (\mathbf{z}_t^{(2)}, \mathbf{y}_t)$ and produces predictions $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_{t+1}$.



then h learns from pair $(\mathbf{z}_t, \mathbf{y}_t)$. It makes predictions $\hat{\mathbf{y}}_{t+1} = h(\mathbf{z}_{t+1}{}^{(\ell)}) = h(\text{DBN}(\mathbf{x}_{t+1}))$ where $\mathbf{z}_{t+1}^{(\ell)}$ is the output from the DBN (given input $\mathbf{x}_{t+1}$). See Figure 2.

DBN-BP.

In our second method, we use the network to predict the labels directly, by adding a final layer of equivalent dimension to the label space (of k units). To make the predictions correspond to the labels, we run the back propagation algorithm to fine-tune the weights of the network (with respect to label assignments), as in in [6]: each instance $\mathbf{x}_t$ is fed in at the bottom, and propagated upward to the final layer as $\mathbf{z}_t^{(\ell)} \equiv \hat{\mathbf{y}}$. The errors are then back propagated through the network, updating the weights (previously initialized by the RBMs) in the process. The only difference from DBN-h is that h is a linear layer, and the weights of the RBMs are updated.

Algorithm 1 illustrates the algorithm we use to bring together an RBM with any standard data-stream classifier (specified by the user). In the case of our DBN-BP method, this classifier (a back-propagated neural network) also updates the weights set by the RBM.

Algorithm 1 Update algorithm at timestep t, given an RBM or DBN and data-stream classifier.

1. transform $\mathbf{x}_t$ into $\mathbf{z}_t$ via RBM(s)
2. if this is a labeled example (i.e., $\exists \mathbf{y}_t$):
   - update the classifier h with example $(\mathbf{z}_t, \mathbf{y}_t)$
3. update the RBM with $\mathbf{x}_t$, see Eq. (1)

3. RELATED WORK

Work on dealing with partially-labeled data streams is a very small portion of the overall data-stream literature. [10] provides a neural network-based approach for semi-supervised data stream classification. [12] uses a micro-clustering technique. We remark again that DBMs, as we work with in this paper, provide a much more powerful framework than clustering. In [3] the main use of unlabeled examples is to help detect concept drift, although using the expectation-maximization algorithm (EM), they can also be used to update the classifier.

A proof of concept for DBNs, using a conditional RBM, in an incremental setting was provided in [4], but little indication was given to its actual performance with regard to existing approaches on real-world concept-drifting data streams.

The task of *active learning* is similar to the semi-supervised case in that only a subset of instances are labeled. The main difference is that the algorithm chooses which instances it wants labels for; and this is the focus (for example, electing examples near the decision boundary) rather than trying to learn from the unlabeled examples. See, for example, [18].

4. EXPERIMENTS

We carry out experiments on some existing real-world datasets (see below) using the open-source MEKA[2] and MOA[3] frameworks (MEKA contains wrappers for MOA classifiers). Our source code will be made available.

Prior to each experiment, we remove the labels of a certain portion s of the instances (depending on the experiment) to simulate a partially-labeled stream; if $s = 0.8$, for example, then we remove the label from every fifth instance. In the case of methods which only deal with labeled examples, they quietly ignore any instances with no labels associated (once having predicted a label).

We split each dataset into 20 evenly-sized windows. We initially construct a model from the instances in the first window, and then update this model incrementally (one instance at a time) with each of the remaining instances. Prior to updating, we first gauge the accuracy of the classifier. Accuracy can be graded individually by window, or as an average overall.

4.1 Datasets

We select real-world datasets of an incremental nature:

- **Enron E-mail Subset** (Enron) A set of 1703 emails from the Enron corpus manually labeled into (an average of 3.4) $k = 53$ categories by the UC Berkeley Enron Email Analysis Project[4]. A small collection by data stream standards, but is time-ordered. Each instance is represented by 1000 binary word-presence feature attributes.

- **20 Newsgroups** (20NG) The classic *20 newsgroups* collection [9] of around 20,000 articles sourced from $k = 20$ newsgroups, with some articles (around 3%) overlapping between multiple newsgroups. Instances are ordered by date over several months. Each instance is represented by 1000 binary word-presence feature attributes.

- **Aviation Safety Reports** (TMC7) contains 28,596 instances of aviation safety reports labeled with problems being described by these reports ($k = 22$ possible problems, an average of 2.16 problems per report). The version of this dataset we used has 500 feature attributes.

- **Forest Cover Type** (CType) One of 7 types of forest cover is associated with cells based on 54 attributes. There are 581,012 instances in total. This dataset is commonly used in data-stream papers, e.g., [2].

[2] http://meka.sourceforge.net
[3] http://moa.cs.waikato.ac.nz/
[4] http://bailando.sims.berkeley.edu/enron_email.html

4.2 Evaluation

Because each of these datasets has separate categories, we create a classifier for each separately, and gauge accuracy as

$$\text{ACCURACY} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}_i \wedge \hat{\mathbf{y}}_i|}{|\mathbf{y}_i \vee \hat{\mathbf{y}}_i|}$$

where $\vee$ and $\wedge$ are the logical OR and AND operations, applied vector-wise. This measure is common to multi-label evaluation, for example [17, 15]. If there is only one label assigned, this measure will default to ordinary accuracy.

Therefore, each dataset really provides k separate problems. For Enron and TMC2007 we learn $h_1, \ldots, h_k$ classifiers separately for each problem of k categories (or labels). For 20NG and CType, due to the very sparse / absence of multi-labeling we learn a single multi-class classifier of k possible classes (and duplicate each $(\mathbf{x}_t, \mathbf{y}_t)$ where $\sum_{j=1}^{k} y_j > 1$, i.e., with multiple categories assigned). Both of these strategies has been used for some time in the multi-label literature. Although more competitive methods for dealing with multiple labels have appeared in recent times (a good survey is provided in [11]), we emphasise that we are only interested in the relative performances of methods in a data-stream scenario here, and are not trying to improve on the state-of-the-art in multi-label classification.

4.3 Results and Discussion

In all experiments we use RBMs with the following parameters: $u = 50$ hidden units, $\lambda = 0.1$ learning rate and 1000 epochs over the initial window. Except with DBN-BP, where we use $\lambda = 0.1$ for the BP neural network updates, and $\lambda = 0.01$ for the unsupervised RBM updates and (with the idea that the discriminative component should learn faster).

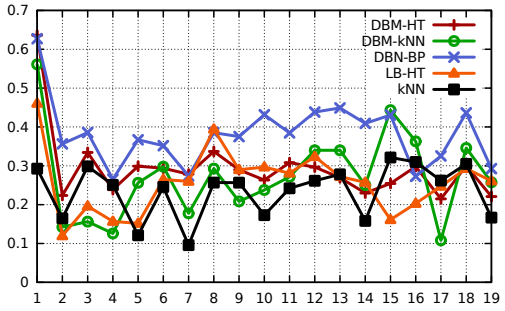
We use a small portion of the initial instances to begin the DBN (for a number of epochs), then update it thenceforth incrementally (where each $\mathbf{x}_t$ is propagated up to $\mathbf{z}_t^{(\ell)}$ before $\mathbf{x}_{(t+1)}$ arrives).

In the first experiment (results in Figure 3, Table 1), we take a couple of popular data-stream algorithms from the literature – k-nearest neighbours (kNN), and Hoeffding Trees (HT) – and compare their performance with and without an RBM-transformed feature space, for a range of rates of supervision $(1.0, \ldots, 0.2)$. With kNN we use a rolling batch of size 1000 (first in first out). We use HT with default parameters in MOA, having Naive Bayes at the leaves.

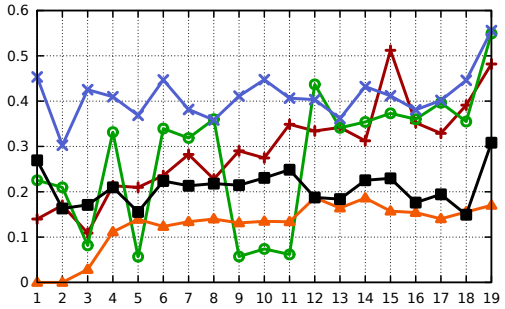
In a second experiment (results in Figure 4, Table 2), we compare the performance of popular kNN (a high-performer in data streams, as shown in [16, 14]) and state-of-the-art leveraging bagged HT (LB/HT) [2] with the performance of our DBNs: our deep-Boltzmann-machine with HT (DBN-h, where $h = k$NN or HT depending on the experiment) and our back-propagation tuned deep belief network DBN-BP.

RBMs can offer significant improvements in prediction performance, up to 10 percentage points. As expected, they are typically more resistant to low levels of supervision, as seen in Figures 3a, 3c, and 3e.

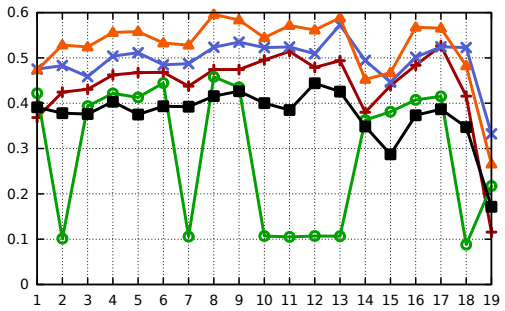
Our DBN-BP method performs strongly, notably on Enron and 20NG (best average accuracy). This is significant, since in general neural-network approaches have not often been taken seriously the data-stream literature (in terms of being 'state-of-the-art'). The DBN-h methods also performs very well overall; showing that deep learning methods can extract important structure from the original input space in a data



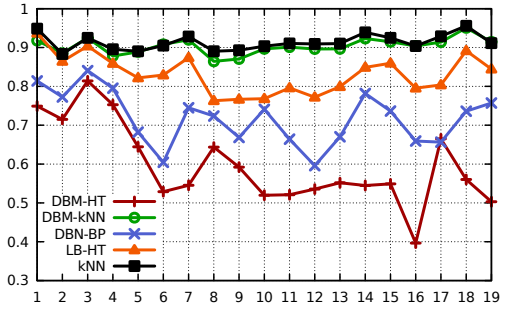
(a) Enron / 0.2 supervision



(b) 20NG / 0.2 supervision

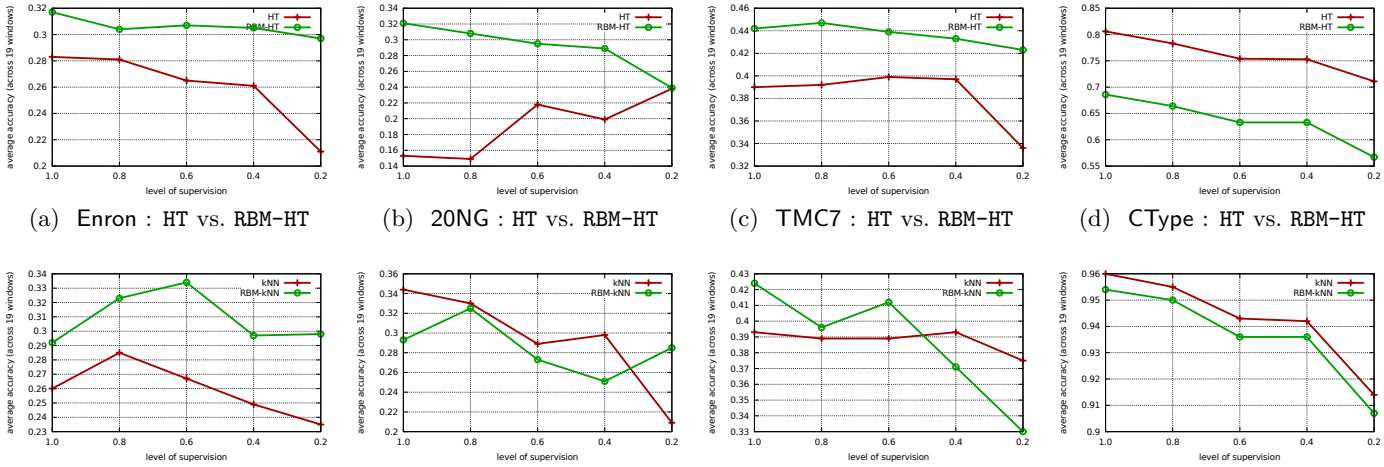


(c) TMC7 / 0.2 supervision



(d) CType / 0.2 supervision

Figure 4: Performance over time (19 windows) for our DBN-kNN, DBN-HT and DBN-BP compared with HT-LB and kNN. STD indicates the standalone method h (either kNN or HT, depending on the experiment).



(a) Enron : HT vs. RBM-HT (b) 20NG : HT vs. RBM-HT (c) TMC7 : HT vs. RBM-HT (d) CType : HT vs. RBM-HT

(e) Enron : kNN vs. RBM-kNN (f) 20NG : kNN vs. RBM-kNN (g) TMC7 : kNN vs. RBM-kNN (h) CType : kNN vs. RBM-kNN

Figure 3: Average predictive performance (over 19 windows) for varying supervision with and without RBMs; using both kNN and HT as classifiers, on the original and RBM-processed feature space.

Table 1: Total running times (rounded to the nearest second) for full supervision in format: build time on initial window + 19 (evaluated windows) × average time per evaluated window.

(a) Running times (s) for kNN

	kNN	RBM-kNN
Enron	$3 + 19 \times 107 = 2033$	$10 + 19 \times 9 = 181$
20NG	$1 + 19 \times 680 = 12920$	$490 + 19 \times 99 = 2371$
TMC7	$1 + 19 \times 653 = 12407$	$363 + 19 \times 162 = 3441$
CType	$0 + 19 \times 1 = 19$	$1611 + 19 \times 7 = 1744$

(b) Running times for HT

	HT	RBM-HT
Enron	$10 + 19 \times 1 = 29$	$524 + 19 \times 1 = 543$
20NG	$41 + 19 \times 1 = 60$	$16 + 19 \times 1 = 35$
TMC7	$3 + 19 \times 4 = 79$	$354 + 19 \times 1 = 374$
CType	$1 + 19 \times 1 = 20$	$1621 + 19 \times 1 = 1640$

Table 2: Average Accuracy over 19 windows and (rank).

Dataset	kNN	LB-HT	DBN-kNN	DBN-HT	DBN-BP
Enron	0.23 5	0.26 4	0.27 3	0.29 2	0.38 1
20NG	0.21 4	0.13 5	0.28 3	0.29 2	0.41 1
TMC7	0.38 4	0.52 1	0.29 5	0.44 3	0.49 2
CType	0.91 1	0.83 3	0.90 2	0.60 5	0.72 4
avg. rank	3.50	3.25	3.25	3.00	2.00

stream in real time, and that this has a positive impact on classification accuracy.

The strengths and limitations of RBMs are clear: RBMs work best with a large and complex input space, and achieve much higher results in these contexts; on both Enron and 20NG, DBN-BP obtains over 10 percentage points over the competitive LB-HT. With smaller, simpler and already fine-tuned input spaces, the advantages are weaker. This is especially apparent on CType: standalone kNN performs best. Clearly, time and memory benefits of RBMs cannot be obtained here either, since the feature space is already small. This makes sense: deep learning finds structure and patterns in a complex input space, but in an already well-structured feature space, it ends up simply recasting the input into another dimension. If the input space of CType were raw pixel data, we could expect greater gains by using deep learning strategies.

We note that DBN-h may be somehow overfitting the data in some cases. For example on Figure 4b, 4c; it performs very well on some windows, and on other windows its accuracy falls dramatically.

It is unusual to see in (Figure 3) that on 20NG with HTs, the RBM's performance degrades with less supervision (not in itself surprising) while the performance of HT actually improves.

Table 1 shows that, concerning kNN, although there is a higher initial investment wrt running time in initialising RBMs, this can quickly pay off by having a smaller input space (where $u < d$). Of course standalone kNN's initialisation is almost instantaneous since it only needs to store the instances in the initial window in a buffer. HTs are so efficient that relatively little time savings can be offered by RBMs, but even under 20NG we see that time savings can be made. Along with time savings, we can often expect memory savings. For example, kNN on 20NG will maintain up to $w \times (d + k) = 1,020,000$ values in memory (for a moving window of size $w = 1000$), compared with $(w \times (u + k)) + (h * u) + (u * u) = 103,500$ values when using two-layer DBN-kNN and only $(h * u) + (u * u) + (u * k) = 53,500$

values for three-layer DBN-BP. The size of a tree is difficult to estimate a-priori but a tree constructed on an input space of $u = 50$ attributes will usually be more compact in many cases than one constructed on $d = 1000$ attributes (for example).

5. CONCLUSIONS

We have investigated the use of deep learning methods to incrementally provide a better representation of data-streams, and thus can improve the accuracy of popular existing data-stream methods (we look at k-nearest neighbours and incremental decision trees) by over 10 percentage points in some cases. At the same time, they can offer (especially in the case of kNN) significant reductions in running time, even taking into account the time spent learning from unlabeled examples.

Furthermore we develop two deep learning approaches, employing existing classifiers on top of RBMs, and fine-tuning them using back propagation (and later direct discriminative prediction), respectively. On higher-dimensional datasets, both methods obtained the highest results overall, compared with popular and competitive methods from the literature.

Acknowledgments

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

6. REFERENCES

[1] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.

[2] Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *ECML PKDD'10*, pages 135–150, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] Hanen Borchani, Pedro Larrañaga, and Concha Bielza. Mining concept-drifting data streams containing labeled and unlabeled instances. In *IEA/AIE 2010 : 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*, pages 531–540, 2010.

[4] Roberto Calandra, Tapani Raiko, Marc Peter Deisenroth, and Federico Montesino-Pouzols. Learning deep belief networks from non-stationary streams. In *Int. Conf. on Artificial Neural Networks*, pages 379–386. Springer, 2012.

[5] Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711âĂŞ1800, 2000.

[6] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[7] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[8] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, pages 97–106, 2001.

[9] Ken Lang. The 20 newsgroups dataset. "http://people.csail.mit.edu/jrennie/20Newsgroups/", 2008.

[10] Daniel Leite, Pyramo Costa Jr., and Fernando Gomide. Evolving granular neural network for semi-supervised data stream classification. In *IJCNN '10: International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2010.

[11] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Saso Dzeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recogn.*, 45(9):3084–3104, 2012.

[12] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 929–934, Washington, DC, USA, 2008. IEEE Computer Society.

[13] Wei Qu, Yang Zhang, Junping Zhu, and Qiang Qiu. Mining multi-label concept-drifting data streams using dynamic classifier ensemble. In *sian Conference on Machine Learnin*, volume 5828 of *Lecture Notes in Computer Science*, pages 308–321. Springer, 2009.

[14] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *11th Int. Symposium on Intelligent Data Analysis*, 2012.

[15] Jesse Read, Bernhard Pfahringer, Geoffrey Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[16] Ammar Shaker and Eyke Hüllermeier. Instance-based classification and regression on data streams. In *Learning in Non-Stationary Environments*, pages 185–201. Springer New York, 2012.

[17] Grigorios Tsoumakas and Ioannis Katakis. Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[18] Indre Zliobaite, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Moa concept drift active learning strategies for streaming data. *Journal of Machine Learning Research - Proceedings Track*, 17:48–55, 2011.