An Ensemble of Classifiers for coping with Recurring Contexts in Data Streams

Ioannis Katakis, Grigorios Tsoumakas and **Ioannis Vlahavas**[1]

Abstract. This paper proposes a general framework for classifying data streams by exploiting incremental clustering in order to dynamically build and update an ensemble of incremental classifiers. To achieve this, a transformation function that maps batches of examples into a new *conceptual* feature space is proposed. The clustering algorithm is then applied in order to group different concepts and identify recurring contexts. The ensemble is produced by maintaining an classifier for every concept discovered in the stream[2].

1 INTRODUCTION

Recent advances in sensor, storage, processing and communication technologies have enabled the automated recording of data, leading to fast and continuous flows of information, referred to as data streams. The dynamic nature of data streams requires continuous or at least periodic updates of the current knowledge in order to ensure that it always includes the information content of the latest batch of data. This is important in applications where the concept of a target class and/or the data distribution changes over time. This phenomenon is commonly known as *concept drift*. A very special type of concept drift is that of *recurring contexts* [5]. In this case, concepts that appeared in the past may recur in the future. Although the phenomenon of reappearing concepts is very common in real world problems (weather changes, buyer habits etc) only few methods take it into consideration [3-5]. In this paper we propose an ensemble of classifiers that utilizes a new representation model for data streams suitable for problems with recurring contexts.

2 TRANSFORMATION FUNCTION

First, the data stream is separated into a number of small batches of examples. Each batch is transformed into a conceptual vector that is constructed out of a number of *conceptual feature* sets. Each feature set corresponds to a feature from the initial feature space. Let's assume that unlabeled (*U*) and labeled (*L*) examples are represented as vectors

$$\vec{x}_U = \left(x_1, x_2, ..., x_n\right) \text{ and } \vec{x}_L = \left(x_1, x_2, ..., x_n, c_j\right)$$

where x_i is the value of the feature f_i, and $c_j \in C$ with C being the set of available classes. Let B_U and B_L be a *batch* of unlabeled and labeled instances of size *b*,

$$B_U = \left\{\vec{x}_{U(k)}, \vec{x}_{U(k+1)}, ..., \vec{x}_{U(k+b-1)}\right\},\ B_L = \left\{\vec{x}_{L(k)}, \vec{x}_{L(k+1)}, ..., \vec{x}_{L(k+b-1)}\right\}$$

Every batch of examples (B_L) is transformed into a conceptual vector $\vec{Z} = \left(z_1, z_2, ..., z_n\right)$, where z_i are the conceptual feature sets. For every batch B_L and feature f_i of the original feature space the conceptual feature sets are calculated as follows:

$$z_i = \begin{cases} \left\{P^v_{i,j} : j = 1..m, v \in V_i\right\}, & \text{if } f_i \text{ is nominal} \\ \left\{\mu_{i,j}, \sigma_{i,j} : j = 1..m\right\}, & \text{if } f_i \text{ is numeric} \end{cases}$$

where $P^v_{i,j} = P(f_i = v \mid c_j)$ and $i \in [1,n], j \in [1,m],\ v \in V_i$, and V_i is the set of values of the nominal attribute f_i $P^v_{i,j}$ is considered to be equal to $n_{v,j} / n_j$, where $n_{v,j}$ is the number of samples of class c_j having the value *v* at attribute i in batch B_L and n_j is the number of samples belonging to c_j in batch B_L. For numeric attributes we use the mean (μ_{i,c_j}) and standard deviation (σ_{i,c_j}) of attribute f_i for samples of class c_j in batch B_L. The notion behind this representation is that every element of the conceptual vectors expresses in what degree a feature characterizes a certain class.

Consequently, conceptual distance between two batches $B_{L(\mu)}$ and $B_{L(\nu)}$ can be defined as the Euclidean distance of the corresponding Conceptual Vectors:

$$\mathit{ConDis}(B_{L(\mu)}, B_{L(\nu)}) = \mathit{Euclidean}(Z_{(\mu)}, Z_{(\nu)}) =$$
$$= \left\{dis(z_{1(\mu)}, z_{1(\nu)}) + ... + dis(z_{n(\mu)}, z_{n(\nu)})\right\}^{1/2}$$

Where, $dis(z_{i(\mu)}, z_{i(\nu)}) = \left(\zeta^1_{i(\mu)} - \zeta^1_{i(\nu)}\right)^2 + ... + \left(\zeta^l_{i(\mu)} - \zeta^l_{i(\nu)}\right)^2$ and $\zeta^j_{i(\mu)}$ is the *j-th* element of the *i-th* conceptual feature-set of the vector μ, whereas *l* is the length of the feature set.

This mapping procedure tries to ensure that the more similar two batches will be conceptually, the closer in distance their corresponding conceptual vectors will be. The definition of this distance will be also beneficial for the clustering algorithm of the framework we present in the following section.

3 THE CCP FRAMEWORK

The main components of the CCP (*C*onceptual *C*lustering and *P*rediction) framework (Fig. 1) are: a) *a mapping function (M)*, that transforms data into conceptual vectors, b) *an incremental clustering algorithm (R)*, that groups conceptual vectors into clusters and c) *an incremental classifier (h)* for every concept discovered. The pseudocode of the framework can be seen in Fig. 2. What is maintained as time (*t*) passes is a set of clusters $G_t = \{g_1, g_2, ..., g_q\}$ and a set of corresponding classifiers $H_t = \{h_1, h_2, ..., h_q\}$. Classifier h_i is trained from batches that belong conceptually to cluster g_i. Initially, $G_o = \varnothing$, $H_o = \varnothing$.

By classifying the current batch according to the classifier built from the cluster of the previous batch we make a kind of a locality assumption. We assume that successive batches (of small size) most of the time will belong to the same concept.

[1] Department of Informatics, Aristotle University of Thessaloniki, 54124 Greece, email:{katak, greg, vlahavas}@csd.auth.gr

[2] The full version of this paper as well as the datasets used for evaluation can be found at: http://mlkd.csd.auth.gr/concept_drift.html

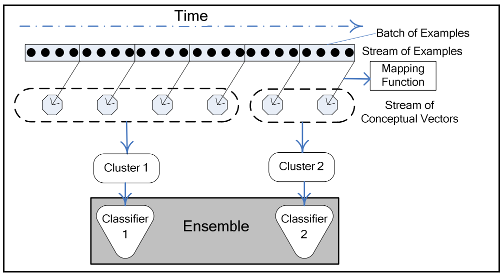

Fig. 1. Clustering conceptual vectors into concepts

```
CCP Framework
begin
for i=1 to infinity do
    $Z_{i-1}$=M.getconceptualVectorOf($B_{L(i-1)}$)
    $g'$ = R.getClusterOf($Z_{i-1}$)
    R.update($Z_{i-1}$)
    $h_{g'}$.update($B_{L(i-1)}$)
    $h_{g'}$.classify($B_{U(i)}$)
end
```

Fig. 2. The main operation of *CCP* framework

4 EVALUATION

Datasets The first two datasets (*usenet1*, *usenet2*) are based on the 20 newsgroups collection [1]. They simulate a stream of messages from different newsgroups that are sequentially presented to a user, who then labels them as *interesting* or *junk*, according to his/her personal interests**.** Table 1 shows which messages are considered interesting (+) or junk (-) in each time period. The third dataset is based on the Spam Assassin collection and contains both spam and legitimate messages.

Table 1. Dataset Usenet1 and Usenet2

	0-300	301-600	600-900	900-1200	1200-1500
			Usenet 1		
medicine	+	-	+	-	+
space	-	+	-	+	-
baseball	-	+	-	+	-
			Usenet 2		
medicine	+	-	-	-	+
space	-	+	-	+	-
baseball	-	-	+	-	-

Methods Evaluation involves the following methods:

Simple Incremental Classifier (SIC): It maintains only one classifier, which incrementally updates its knowledge.

Time Window (TW): It classifies incoming instances based on the knowledge of the latest *N* examples.

Weighted Examples (WE): It consists of an incremental classifier that supports weighted learning. Bigger weights are assigned to more recent examples in order to focus on new concepts.

An incremental naïve bayes classifier is used as base classifier for the above methods.

Our implementation of the CCP framework includes the mapping function discussed in section 2, the Leader-Follower algorithm described in [2] as the clustering component and an incremental Naive Bayes classifier. Preliminary experiments showed that a batch size around 50 instances is appropriate. Larger batches invalidate the locality assumption, whereas smaller batches do not suffice for calculating the summary probabilistic statistics. The experiments include a benchmark version of our framework (dubbed Oracle), where perfect clustering assignments are manually provided to the system. This allows the study of the maximum performance that can be achieved using the CCP framework.

Results Table 2 shows the results of the experiments in the three datasets. We notice that even a basic implementation of CCP achieves better performance than all other methods.

Fig. **3** shows the average accuracy over fifty instances for the CCP and WE method for the Usenet1 dataset. Note the sudden dives of WE's accuracy in drift time-points. In all cases, CCP manages to recover much faster from the drift. Most notably, at the last two drift point, CCP recognizes the recurrent theme and remains accurate. Finally, the performance of Oracle, strongly underlines the fact that there is room for improvement by using more advanced incremental clustering algorithms.

Table 2. Accuracy of the four methods in the three datasets

	Usenet1	**Usenet2**	**spam**
Simple Incremental	0.59	0.73	0.75
TimeWindow (w=100)	0.56	0.60	0.60
TimeWindow (w=150)	0.59	0.62	0.64
TimeWindow (w=300)	0.58	0.70	0.62
CCP (Oracle)	**0.81**	**0.80**	-
CCP (Leader-Follower)	0.75	0.77	0.93
Weighted Examples	0.67	0.75	0.91

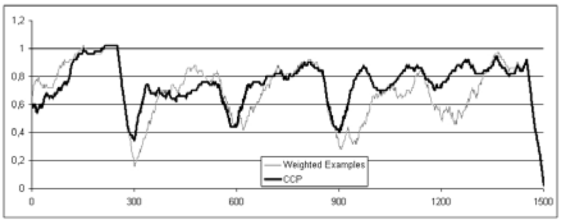

Fig. 3. Average accuracy over 50 instances for WE and CCP.

5 ACKNOWLEDGMENTS

This work was partially supported by a PENED program (EPAN M.8.3.1, No.03EΔ73), jointly funded by the European Union and the Greek Government (General Secretariat of Research and Technology).

REFERENCES

[1] Asuncion, A. and Newman, D.J., UCI Machine Learning Repository. 2007, University of California, School of Information and Computer Science [www.ics.uci.edu/~mlearn/MLRepository.html]: Irvine, CA.

[2] Duda, R.O., Hart, P.E., and Stork, D.G., Pattern Classification. 2000: Wiley-Interscience.

[3] Forman, G. Tackling Concept Drift by Temporal Inductive Transfer. in 29th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2006. Washington, USA: p. 252-259.

[4] Harries, M.B., Sammut, C., and Horn, K., Extracting Hidden Context. Machine Learning, 1998. 32(2): p. 101-126.

[5] Widmer, G. and Kubat, M., Learning in the Presense of Concept Drift and Hidden Contexts. Machine Learning, 1996. 23(1): p. 69-101.