Probabilistic Aggregation of Classifiers for Incremental Learning[⋆]

Patricia Trejo[1], Ricardo Ñanculef[1], Héctor Allende[1], and Claudio Moraga[2,3]

[1] Universidad Técnica Federico Santa María,
Departamento de Informática, CP 110-V Valparaíso, Chile
{ptrejo,jnancu,hallende}@inf.utfsm.cl
[2] European Centre for Soft Computing 33600 Mieres, Asturias, Spain
[3] Dortmund University, 44221 Dortmund, Germany
mail@claudio-moraga.eu

Abstract. We work with a recently proposed algorithm where an ensemble of base classifiers, combined using weighted majority voting, is used for incremental classification of data. To successfully accommodate novel information without compromising previously acquired knowledge this algorithm requires an adequate strategy to determine the voting weights. Given an instance to classify, we propose to define each voting weight as the posterior probability of the corresponding hypothesis given the instance. By operating with priors and the likelihood models the obtained weights can take into account the location of the instance in the different class-specific feature spaces but also the coverage of each class k given the classifier and the quality of the learned hypothesis. This approach can provide important improvements in the generalization performance of the resulting classifier and its ability to control the stability/plasticity tradeoff. Experiments are carried out with three real classification problems already introduced to test incremental algorithms.

1 Introduction

Machine learning actually offers an interesting number of methods and tools for intelligent data analysis and knowledge discovery. Most of these techniques are developed for static environments, that is for scenarios where data to be mined is completely gathered before the analysis. This data is also supposed to be representative of all the patterns of interest. However, every time is more common to find applications where data become available over time in batches of observations, potentially containing new information that is necessary to discriminate and analyze to update the knowledge that has been obtained previously. These kind of situations also arise when we are dealing with "active learning" in which the sets of training data can be provided only one after the other. Please refer to [10], [9] and [8] for different approaches to incremental learning.

⋆ This work was supported in part by Research Grant Fondecyt (Chile) 1040365, 7060040 and 1070220. Partial support was also received from Research Grant BMBF (Germany) CHL 03-Z13 and Research Grant DGIP-UTFSM (Chile).

Recently, a methodology for incremental classification has been proposed in [8], [7] and [3] which consists in using an ensemble of base classifiers [5], combined using weighted majority voting [6]. Section 2 of this paper provides a brief description of the incremental classification problem and the main structure of this ensemble algorithm.

In the latter approach, the adequate definition of the voting weights is critical to control the appropriate tradeoff between *plasticity* or adaptation to new contexts and *stability* or preservation of existing knowledge (please see [4] and [8] for a discussion about the stability/plasticity dilemma). In section 3, we propose to define the voting weights of each hypothesis h as the posterior probability of the hypothesis given the instance to classify: $P(h|x)$. In this framework, class-specific probabilities can be introduced decomposing the likelihood $P(x|h)$ of the model in class conditional likelihoods $P(x|h,k)$ and then aggregating among classes using the so called *law of total probability.* Resulting voting weights take into account the location of the instance in the different class-specific feature spaces but also the prior probability of each class k given the classifier $P(k|h)$. Further generalizations of the proposed method can be obtained by manipulating priors or the likelihood models. For example, the traditional approach of using the accuracy for determining the voting weights can be introduced as priors $P(h)$ on the set of classifiers.

In the final section of this paper we discuss the experimental results obtained in three real classification problems whose results have been already reported in [3] and [7] using incremental algorithms.

2 An Ensemble Approach for Incremental Learning

In this paper we study the problem of incremental classification, where each example $z = (x, y)$ consists of a feature vector $x \in \mathbb{R}^n$ and a label y which can take a finite set of values $k = 1, 2, \ldots, K$ indicating the class of the instance. As against the standard approach to learning from examples, in an incremental environment, data is available over time t in batches or groups of observations S_t. Given this framework, the task of an incremental algorithm [8] , [7] , [3] is learning the new information contained in S_t without forgetting the knowledge previously acquired from $S_1, \ldots, S_{t-1}$. Re-learning using the entire training set $S_1 \bigcup S_2 \bigcup \ldots S_t$ is a possible but not very efficient solution. Hence a usual additional restriction over an incremental algorithm is to only require access to the actual dataset S_t and the actual model.

In [8] Polikar et al. proposed a new algorithm for incremental learning named *Learn++*, which is based in the now extensively analyzed *AdaBoost* algorithm [2] designed for batch classification. The main structure of this algorithm, improved subsequently in [7] and [3] , is sketched as algorithm (1). When a new set S_j of observations becomes available, a new set of classifiers is created with the purpose of learning the data presumably containing new knowledge. This is achieved resampling the new observations with weights proportional to the error of the existing model in the new data. Each classifier h_t just created is stacked with the classifiers generated previously to update the current ensemble H_t.

Algorithm 1. Structure of the Learn++ Algorithm

```
    Initialize $T = 0$
    foreach batch of observations $S_j$ of size $m_j$ do
INI     Initialize the sampling weights $d_0(i)$ of each example $i = 1, \dots, m_j$
        for $t = T+1, \dots, T+T_j$ do
            Set the sampling distribution to $D_t = d_t(i) / \sum_{j=1}^{m} d_t(j)$.
RES         Generate a set of examples $X_t$ sampling $S_j$ according to $D_t$.
            repeat
                Train a base classifier with $X_t$ to obtain $h_t$.
                Compute the weighted error of $h_t$ on $S_j$, $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.
            until $\epsilon_t < 1/2$
AGG         Compute the ensemble hypothesis $H_t(x)$ using an aggregation algorithm
            $\bigoplus$ over the set of classifiers $h_1, h_2, \dots, h_t$.
            Compute the weighted error of $H_t$ on $S_j$, $E_t = \sum_{i:H_t(x_i) \neq y_i} D_t(i)$
            Compute the confidence of $H_t$, $\alpha_t = \log((1 - E_t)/E_t)$
UPD         Update the sampling weights
                $d_{t+1}(i) = d_t(i) \times \begin{cases} e^{-\alpha_t} & \text{, if } H_t(x_i) = y_i \\ 1 & \text{, otherwise} \end{cases}$    (1)
        Recall the current number of classifiers $T = \sum_{i=1}^{j} T_i$.
AGG For any $x$, compute the final ensemble decision $H_T(x)$ applying an aggregation
    algorithm $\bigoplus$ over the complete set of classifiers $h_1, h_2, \dots, h_T$.
```

Initialization of the sampling weights d_t differs between different implementations of the algorithm. In [3] and [7] the error of the existing ensemble on the new batch of observations is computed and then the weight update rule of step with label (UPD) of algorithm (1) is applied. This is made to attempt that the algorithm focuses from the first classifier on the instances containing novel information. It should be recalled on the other hand, that the distribution update rule of AdaBoost is only dependent of the performance of the last created classifier h_t, whereas that of Learn++ is based on the performance of the entire ensemble H_t. This rule allows Learn++ to focus on the observations of the current batch that potentially contain novel information with respect to classifiers created for previous batches.

As described in [4], incremental learning usually represents a tradeoff between *stability* and *plasticity*. A completely stable classifier will not accommodate any new information but a completely plastic classifier will not conserve previous knowledge. A central issue of the algorithm defined above is hence, the aggregation procedure used to combine the classifiers. Although no classifiers are discarded, the ability of the algorithm to preserve previous knowledge and accommodate novel information strongly depends on the relative importance that each classifier has in the decisions taken by the final ensemble hypothesis. If for example we use a weighted majority voting algorithm that tends to assign small weights to the classifiers created for the last batch of data, we will obtain an extremely stable but poorly flexible algorithm, and viceversa.

3 Aggregation of Classifiers for Incremental Learning

This section is devoted to define a majority voting aggregation mechanism appropriate for incremental classification based on algorithm (1). In ensemble approaches that use a weighted voting mechanism for combining classifier outputs, each classifier h_t votes with a weight w_t on the class it predicts. The final decision is the class that cumulates the highest total weight from all the classifiers [5].

In [8], Polikar et al. proposed the AdaBoost aggregation strategy for using within the algorithm (1). Voting weights are computed as $\log((1-\eta_t)/\eta_t)$ where η_t is the training error of h_t. Although this procedure seems reasonable in a batch classification problem, in incremental environments this rule becomes not optimal. Since different batches will contain instances from different locations of the feature space, classifiers corresponding to different batches are modelling different patterns and hence the performances of these classifiers are not directly comparable.

An idea to overcome this problem is to use non-constant voting weights but instance-dependent weights. In [3] , Gangardiwala et al. proposed to modify the original aggregation strategy of the algorithm (1) and to consider weights that depend on the location of the instance to classify in the feature space. The voting weights are heuristically computed as $w_t(x) = \min_k 1/\delta_{tk}(x)$, where, δ_{tk} is the class-specific *mahalanobis distance* of the test instance to the data used to train the classifier. If X_t is set of input instances used to train the classifier h_t and X_{tk} is the subset of X corresponding to the instances of class k, with $k = 1, \ldots, K$, the k-th class-specific distance of an input instance x to X_t is computed as

$$\delta_{tk}(x) = (x - \mu_{tk})' \cdot \mathbf{C}_{tk}^{-1} \cdot (x - \mu_{tk}) \tag{2}$$

where μ_{tk} is the mean and $\mathbf{C}_{tk}$ the covariance matrix of X_{tk}. This approach introduces important improvements with respect to the use of constant weights.

In this paper, we propose to define the voting weight $w_t(x)$ of the classifier h_t, for predicting the instance x, as the posterior probability of the classifier given the instance, that is $w_t(x) = P(h_t|x)$. Using the Bayes rule, this probability can be expressed in terms of a likelihood model $P(x|h_t)$ and a prior $P(h_t)$ on the classifier.

$$P(h_t|x) = \frac{P(x|h_t) \times P(h_t)}{P(x)} \tag{3}$$

To obtain a reasonable likelihood model we propose to decompose the conditional probability $P(x|h_t)$ partitioning between the different classes $k = 1, \ldots, K$ and introducing the class-conditional probabilities $P(x|h_t, k)$. By the so called law of total probability we obtain that

$$P(x|h_t) = \sum_{k=1}^{K} P(x|h_t, k)P(k|h_t) \tag{4}$$

That is, we compute the likelihood of x supposing that the event $A_k =$ "x is of class k" is true and then we average among the different events A_k with weights $P(k|h_t)$ corresponding to the the prior probability of the event A_k given the classifier h_t. For the class-specific likelihoods $P(x|h_t, k)$ we propose to use a simple class-specific gaussian model on the data used to train the classifier h_t. Using the notation introduced for equation (2) we can write

$$P(x|h_t, k) = N \times \exp\left(-(x - \mu_{tk})' \cdot \mathbf{C}_{tk}^{-1} \cdot (x - \mu_{tk})\right) = N \times \exp(-\delta_{tk}) \qquad (5)$$

where N is a normalizing constant. Note that this probability is inversely proportional to the class-specific mahalanobis distance $\delta_{tk}(x)$. Now, since the selected likelihood model depends on the data used to train the classifier, it seems reasonable to use as the prior $P(k|h_t)$ the fraction of such data that belongs to the class k, that is the relative coverage of this class with respect to the classifier h_t. If X_t denotes the set of input instances used to train the classifier and X_{tk} the subset of X corresponding to the instances of class k, this can be written as $P(k|h_t) = |X_{tk}|/|X_t|$ where $|\cdot|$ denotes cardinality.

To determine the weights $w_t(x)$ the only thing that is missing is to model the prior $P(h_t)$ on the set of classifiers. A simple choice is to use uniform priors, such that any classifier have the same prior probability of classifying well the instance x. After calculating (3) and discarding the constant terms, this leads to the following weights,

$$\hat{w}_t(x) = \sum_{k=1}^{K} \exp\left(-\delta_{tk}\right) \times \frac{|X_{tk}|}{|X_t|} \qquad (6)$$

Note that the resulting weights take into account not only the location of the instance in the different class-specific feature spaces, but also the coverage of each class k given the classifier. This can prevent the situation where a classifier h_t has been trained with instances X_{tk} of a given class k very similar to the instance to classify x_{test} but this has not seen enough examples of the class k to generalize well.

It should be noted that the whole set of classifiers generated after a new batch of observations have arrived to the system are generated to learn the new information contained in these observations. Resampling makes that different classifiers work with partially different data sets, however this behavior is the strategy used to distribute the original problem between the different classifiers that, as a group, are learning the same underlying data. Hence it makes sense to compute the probabilities $P(h_t|x)$ only one time per batch, immediately after the first resampling of the data, that has the task of identifying the observations that presumably contain new information. In this approach, we call *Global Probabilistic*, all the classifiers created for a given batch of data S_k receive the same weight, that is computed using the equation (6) with the set of observations X_t obtained after step with label (RES) of algorithm (1) has been applied for the first time with the actual batch.

The proposed framework to determine the voting weights can be easily modified by manipulating the likelihood models and the priors on the clases or the set of classifiers. For example, we could introduce priors proportional to the accuracy of the classifier h_t as in the AdaBoost algorithm. By defining $P(h_t) = \log((1-\eta_t)/\eta_t)$ where η_t is the training error of h_t, we obtain the weights:

$$\hat{w}_t(x) = \left(\sum_{k=1}^{K} \exp\left(-\delta_{tk}\right) \times \frac{|X_{tk}|}{|X_t|} \right) \times \log((1-\eta_t)/\eta_t) \tag{7}$$

Priors can also be used to model the dynamics of the learning problem. For example, we could impose priors inversely proportional to the age of the models.

4 Experiments and Conclusions

In this section we present the results obtained with three incremental classification problems already studied in [7] and [3]. The benchmarks are the *Vehicle Silhouettes* database, the *Wine Recognition* database and the *Optical Character Recognition* database from UCI [1]. Multilayer perceptrons were used as base classifiers trained using the backpropagation algorithm, as in previous works with algorithm (1). In the tables we use the following names for the algorithms: *Learn++* for algorithm (1) as proposed in [3], where weights are computed using the inverse of the mahalanobis distance as defined in equation (2); *Probabilistic Uniform* for algorithm (1) but using the voting weights we have proposed in equation (6) and *Probabilistic with accuracy-based Priors* when using the voting weights we have proposed in equation (7). The approach named *Global Probabilistic* was discussed in the previous section.

The **Vehicle Silhouettes Database** consists of 4 classes and 18 attributtes. To simulate incremental learning, the database was split in three batches S_1, S_2, S_3 of training observations and one test dataset S_{test} following exactly the same class distribution proposed in [3]. The **Wine Recognition Database** consists in 3 classes and 13 attributes. This database was split in two batches S_1, S_2 of training observations and one test dataset S_{test} following the class distribution proposed in [7]. Finally, the **Optical Digits Database** (OCR) consists in 10 classes and 64 attributes. This database was split in three batches S_1, S_2, S_3 of training observations and one test dataset S_{test} following the guidelines of [7] for determining the class distribution. In the three benchmarks, data distributions were deliberatively designed to test the ability of the algorithm in dealing with the stability/plasticity tradeoff.

Tables (1), (2) and (3) show 95%-confidence intervals for the **percentage of correct classification** obtained with the different algorithms in the three benchmarks. Each row of this table corresponds to the performance obtained in the corresponding dataset (S_1, S_2, S_3, S_{test}) after the training session indicated in the columns. Recall that in the training session i the algorithm has seen the training sets $S_1, \dots, S_{i-1}$ and has to incrementally learn the new "incoming batch" S_i (results in $S_{i+1}, S_{i+2}, \dots$ are provided to observe the relative improvement in the $i+1, i+2, \dots$ learning session). In the first two problems, the three proposed algorithms show a better ability to accommodate novel information without catastrophically forgetting previous knowledge. This behavior finally leads to better generalization performance as can be observed comparing the last row of these tables. In the last data set however, our algorithms do not improve the performance of *Learn++* but show comparable results.

Table 1. Best performance results obtained with the **Vehicle Silhouettes Database**. These results were achieved with 4 classifiers and 10 neurons for Learn++, 10 classifiers and 5 neurons for *Probabilistic Uniform*, 10 classifiers and 5 neurons for *Probabilistic with accuracy-based Priors*; and 6 classifiers and 20 neurons for *Global Probabilistic.*

Learn++				*Probabilistic Uniform*			
	1	**2**	**3**		**1**	**2**	**3**
$\mathbf{S_1}$	66.3 ∼ 74.5	64.1 ∼ 70.2	62.7 ∼ 68.3	$\mathbf{S_1}$	75.2 ∼ 84.1	73.6 ∼ 79.9	73.0 ∼ 78.4
$\mathbf{S_2}$	73.4 ∼ 81.9	81.8 ∼ 89.9	74.7 ∼ 83.3	$\mathbf{S_2}$	87.6 ∼ 100.	99.0 ∼ 100.	95.5 ∼ 99.6
$\mathbf{S_3}$	89.1 ∼ 96.4	89.0 ∼ 95.9	93.1 ∼ 96.6	$\mathbf{S_3}$	87.2 ∼ 100.	86.7 ∼ 100.	99.3 ∼ 100.
$\mathbf{S_{test}}$	70.4 ∼ 74.4	71.1 ∼ 74.7	70.0 ∼ 73.8	$\mathbf{S_{test}}$	72.6 ∼ 78.7	71.8 ∼ 79.1	76.3 ∼ 77.9
Probabilistic with accuracy-based Priors				*Global Probabilistic*			
	1	**2**	**3**		**1**	**2**	**3**
$\mathbf{S_1}$	82.4 ∼ 88.9	81.2 ∼ 86.8	80.1 ∼ 84.7	$\mathbf{S_1}$	71.0 ∼ 77.8	70.7 ∼ 75.3	70.4 ∼ 74.9
$\mathbf{S_2}$	87.7 ∼ 100.	98.1 ∼ 100.	93.7 ∼ 100.	$\mathbf{S_2}$	88.7 ∼ 100.	98.3 ∼ 100.	97.8 ∼ 100.
$\mathbf{S_3}$	87.2 ∼ 100.	85.5 ∼ 100.	97.9 ∼ 100.	$\mathbf{S_3}$	87.0 ∼ 100.	87.9 ∼ 100.	98.3 ∼ 100.
$\mathbf{S_{test}}$	74.4 ∼ 80.5	73.6 ∼ 80.4	77.6 ∼ 80.1	$\mathbf{S_{test}}$	75.0 ∼ 81.8	75.9 ∼ 81.5	79.1 ∼ 80.8

Table 2. Best performance results obtained with the **Wine Recognition Database**. These results were achieved with 2 classifiers and 5 neurons for *Learn++*, 2 classifiers and 20 neurons for *Probabilistic Uniform*, 2 classifiers and 10 neurons for *Probabilistic with accuracy-based Priors*; and 8 classifiers and 10 neurons for *Global Probabilistic.*

Learn++			*Probabilistic Uniform*		
	1	**2**		**1**	**2**
$\mathbf{S_1}$	99.5 ∼ 100.	92.8 ∼ 96.3	$\mathbf{S_1}$	99.5 ∼ 100.	97.0 ∼ 98.8
$\mathbf{S_2}$	48.5 ∼ 49.3	97.8 ∼ 99.2	$\mathbf{S_2}$	48.4 ∼ 49.2	98.5 ∼ 99.6
$\mathbf{S_{test}}$	70.2 ∼ 71.8	93.6 ∼ 96.9	$\mathbf{S_{test}}$	70.7 ∼ 72.1	94.7 ∼ 97.5
Probabilistic with accuracy-based Priors			*Global Probabilistic*		
	1	**2**		**1**	**2**
$\mathbf{S_1}$	98.9 ∼ 99.9	97.1 ∼ 98.9	$\mathbf{S_1}$	99.4 ∼ 100.	97.2 ∼ 98.9
$\mathbf{S_2}$	48.5 ∼ 49.2	98.4 ∼ 99.7	$\mathbf{S_2}$	49.1 ∼ 49.4	97.9 ∼ 99.6
$\mathbf{S_{test}}$	70.2 ∼ 72.1	93.9 ∼ 97.4	$\mathbf{S_{test}}$	71.2 ∼ 72.5	96.7 ∼ 99.0

The results described above correspond to the best test performance obtained after experimenting with different number of neurons $n_H = 5, 10, 20$ and base classifiers $M = 2, 4, 6, 8, 10$. The testing results for the different combinations of this parameters are omitted due to space limitations, however, the improvements in the first two data sets and the comparable results in the third data set are uniform among different combinations of parameters. We indicate in each table the number of neurons and classifiers used within each algorithm.

Finally, the computational complexity of our algorithms is comparable to that of Learn++. The only additional computation are the class-specific priors $P(k|h_t)$. The algorithm *Global Probabilistic*, on the other hand, does not compute the weights one time per classifier but one time per batch and hence, the complexity is clearly lower. It is interesting to note that this faster algorithm is the most effective of the three algorithms we have proposed, except in the third data set.

Table 3. Best performance results for analyzed algorithms in the **OCR Database**. These results were achieved with 2 classifiers and 20 neurons for the four algorithms.

Learn++				*Probabilistic Uniform*			
	1	**2**	**3**		**1**	**2**	**3**
$\mathbf{S_1}$	99.9 $\sim$ 100.	98.7 $\sim$ 99.1	99.3 $\sim$ 99.4	$\mathbf{S_1}$	99.9 $\sim$ 100.	98.5 $\sim$ 98.9	98.0 $\sim$ 98.4
$\mathbf{S_2}$	53.6 $\sim$ 53.8	99.9 $\sim$ 100.	99.2 $\sim$ 99.5	$\mathbf{S_2}$	53.6 $\sim$ 53.8	99.9 $\sim$ 100.	99.0 $\sim$ 99.3
$\mathbf{S_3}$	21.2 $\sim$ 21.3	40.7 $\sim$ 41.1	99.8 $\sim$ 99.9	$\mathbf{S_3}$	21.2 $\sim$ 21.3	40.7 $\sim$ 41.0	99.9 $\sim$ 100.
$\mathbf{S_{test}}$	59.4 $\sim$ 59.6	77.6 $\sim$ 78.0	97.6 $\sim$ 97.8	$\mathbf{S_{test}}$	59.4 $\sim$ 59.6	77.6 $\sim$ 77.9	96.3 $\sim$ 96.7
Probabilistic with accuracy-based Priors				*Global Probabilistic*			
	1	**2**	**3**		**1**	**2**	**3**
$\mathbf{S_1}$	99.9 $\sim$ 100.	98.5 $\sim$ 98.9	97.9 $\sim$ 98.4	$\mathbf{S_1}$	99.7 $\sim$ 99.8	98.0 $\sim$ 98.4	97.2 $\sim$ 97.6
$\mathbf{S_2}$	53.6 $\sim$ 53.8	100. $\sim$ 100.	99.1 $\sim$ 99.3	$\mathbf{S_2}$	53.6 $\sim$ 53.8	99.1 $\sim$ 99.4	98.0 $\sim$ 98.5
$\mathbf{S_3}$	21.2 $\sim$ 21.3	40.6 $\sim$ 40.9	99.9 $\sim$ 100.	$\mathbf{S_3}$	21.2 $\sim$ 21.3	40.4 $\sim$ 40.8	97.9 $\sim$ 98.1
$\mathbf{S_{test}}$	59.5 $\sim$ 59.6	77.5 $\sim$ 78.0	96.3 $\sim$ 96.8	$\mathbf{S_{test}}$	59.4 $\sim$ 59.5	77.5 $\sim$ 77.8	96.2 $\sim$ 96.6

References

1. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
2. Freud, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and application to boosting. Journal of Computer and System Sciences 55(1), 119–137 (1997)
3. Gangardiwala, A., Polikar, R.: Dynamically weighted majority voting for incremental learning and comparison of three boosting based approaches, Joint Conf. on Neural Networks (IJCNN 2005), pp. 1131–1136 (2005)
4. Grossberg, S.: Nonlinear neural networks: principles, mechanisms and architectures. Neural Networks 1(1), 17–61 (1988)
5. Kuncheva, L.: Combining pattern classifiers: Methods and algorithms, Wiley InterScience (2004)
6. Littlestone, N., Warmuth, M.: The weighted majority algorithm. Information and Computation 108(2), 212–261 (1994)

7. Muhlbaier, M., Topalis, A., Polikar, R.: Learn++.mt: A new approach to incremental learning. In: MCS 2004. LNCS, vol. 3077, pp. 52–61. Springer, Heidelberg (2004)
8. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks, IEEE Transactions on systems, man, and cybernetics Part C: applications and reviews, 31(4), 497–508 (2001)
9. Vijayakumar, S., Ogawa, H.: RKHS based functional analysis for exact incremental learning. Neurocomputing 29, 85–113 (1999)
10. Widmer, K., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23, 69–101 (1996)