An Ensemble Method for Incremental Classification in Stationary and Non-stationary Environments*

Ricardo Ñanculef, Erick López, Héctor Allende, and Héctor Allende-Cid

Department of Informatics,
Federico Santa María University, Chile
{jnancu,elopez,hallende,vector}@inf.utfsm.cl

Abstract. We present a model based on ensemble of base classifiers, that are combined using weighted majority voting, for the task of incremental classification. Definition of such voting weights becomes even more critical in non-stationary environments where the patterns underlying the observations change over time. Given an instance to classify, we propose to define each voting weight as a function that will take into account the location of an instance to classify in the different class-specific feature spaces and also the prior probability of such classes given the knowledge represented by the classifier as well as its overall performance in learning its training examples. This approach can improve the generalization performance and ability to control the stability/plasticity trade-off, in stationary and non-stationary environments. Experiments were carried out using several real classification problems already introduced to test incremental algorithms in stationary as well as non-stationary environments.

Keywords: Incremental Learning, Dynamic Environments, Ensemble Methods, Concept Drift.

1 Introduction

It is important that machine learning systems be capable of dealing with new observations. Moreover, for large scale applications, it is unrealistic to think that a complete set of representative examples is available from the start, and hence algorithms able to learn from the observation of a sequence of examples delayed in time is crucial. A simple approach consists in using past and current observations to build a new model every time that new observations become available. However this solution is usually impractical or infeasible. An additional problem appears when the patterns underlying the observations change over time, that is the environment is not stationary. For example, in a document filtering problem, it is possible that the features defining a category are no longer valid because the preferences of user have changed. Ensemble methods are based on the idea

* This work was supported in part Research Grant DGIP-UTFSM (Chile).

C. San Martin and S.-W. Kim (Eds.): CIARP 2011, LNCS 7042, pp. 541–548, 2011.

of combining a set of simple predictors instead of using only one [4,7,16]. An interesting point is that with an appropriate design, the expected performance of the combined predictor can be better than the average performance of the individual predictors. This flexibility makes them particularly suitable for learning in changing environments. There are ensemble methods that have been proposed to address the problem of incremental learning such as in [2,5,10,11,13,15]. Originally these methods were proposed for stationary environments, but today have been extended for non-stationary ones. In this paper, we propose a strategy (based on [9,5,12,15]) for incremental learning in non-stationary environments that consist in using a set of base classifiers, combined using weighted majority voting, where voting weights of each hypothesis h will be a function that depends on the sample used to train classifier h.

2 Problem Definition

To obtain a formal definition of the incremental learning problem we follow a statistical approach. Throughout this work we suppose that observations z live in a space Z and are all drawn according to a probability measure $P(z)$. The observations are of the form $z = (x, y)$ where x represents some information about z and y a desired response or action. Given a sample of the form $S = z_1...z_n$, obtained sampling the distribution P, we are asked to recover a model h representing the relation between x and y. The problem in learning from examples is that instead of the measure P, we only have a finite sample of examples S. We select h such that it minimizes the so called *empirical risk*:

$$R_S(h) = \frac{1}{n} \sum_{i=1}^{n} Q(h(x_i), y_i) \tag{1}$$

Instead of a single sample, in incremental scenarios, we have to deal with a sequence of samples or batches of observations $S_1, S_2, ..., S_t$ which arrive continuously over time and possibly have different size. An exact definition of the learning task in such incremental scenarios is hence not straightforward.

A learning algorithm is called incremental if it is capable to generate hypotheses in steps, where each step starts with certain working hypothesis and a set of new data and ends with an appropriate updated hypothesis. Given a sequence of training sets $S_1, ..., S_t$ such algorithm is hence capable to generate a sequence of hypotheses $h_1, ..., h_t$, where h_t is obtained from h_{t-1} and S_t. We distinguish two possible objectives for the learning tasks: (a) **Stationary Environments**, the goal at time t is to obtain a hypothesis as close as the one obtained by training with a sample $S = S_1 \bigcup ... \bigcup S_t$. If $S_1, ..., S_t$ were drawn according to a distribution P, future cases (S_{t+1}) also appear according to the distribution given by P, so we can measure the performance of the algorithm by using a test set and also any of the partial samples $S_1, ..., S_t$. (b) **Non-Stationary Environments**, the underlying distribution of the new examples changes over time. The goal at time t is to obtain a hypothesis capable to decide well the next batch of observations, that is S_{t+1}.

3 An Ensemble Based System for Learning in Dynamic Environments

The overall structure of Ensemble Methods consist in generating a new hypothesis h_t when a new set of observations S_t becomes available. An updated model is obtained combining the individual hypotheses using majority voting.

The Learn++ algorithm proposed in [13] is based on AdaBoost [3]. The main steps of Learn++ (modified in [10] and [5]), are sketched as algorithm (1). When a new set S_j of observations becomes available, a training sample X_t is generated from S_j , sampling with weights given by a distribution d. A new set of classifiers is then created to learn X_t and stacked with the classifiers generated previously to update the current ensemble H_t.

Algorithm 1. Structure of the Learn++ Algorithm

Initialize $T = 0$
for each batch of observations S_j of size m_j **do**
  Initialize the sampling weights $d_0(i)$ of each example $i = 1, \ldots, m_j$
  for $t = T + 1, \ldots, T + T_j$ **do**
    Set the sampling distribution to $D_t = d_t(i) / \sum_{j=1}^{m} d_t(j)$.
    Generate a set of examples X_t sampling S_j according to D_t.
    while $\epsilon_t < 1/2$ **do**
      Train a base classifier with X_t to obtain h_t.
      Compute the weighted error of h_t on S_j, $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.
    end while
    Compute the ensemble hypothesis $H_t(x)$ using an aggregation algorithm $\bigoplus$ over the set of classifiers $h_1, h_2, \ldots, h_t$.
    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, $\alpha_t = \log((1 - E_t)/E_t)$
    Update the sampling weights

$$d_{t+1}(i) = d_t(i) \times \begin{cases} e^{-\alpha_t} & \text{, if } H_t(x_i) = y_i \\ 1 & \text{, otherwise} \end{cases} \tag{2}$$

  end for
  Recall the current number of classifiers $T = \sum_{i=1}^{j} T_i$.
end for
For any x, compute the final ensemble decision $H_T(x)$ applying an aggregation algorithm $\bigoplus$ over the complete set of classifiers $h_1, h_2, \ldots, h_T$.

The KBS-Stream algorithm proposed in [15] is similar to Learn++ but it is based on a sampling strategy named KBS (Knowledge-Based Sampling) [14]. Instead of using the error on the new observations to define the sampling weights d_t, KBS defines the concept of *Lift*, which measures the correlation (according to a given distribution) between a specific prediction and a specified true label.

4 An Aggregation Framework of Classifiers for Dynamic Environments

We define a majority voting aggregation mechanism appropriate for incremental classification based on algorithm (1). In these approach each classifier h_t votes

with a weight w_t on the class it predicts for a given instance x. The final decision is the class that cumulates the highest total weight from all the classifiers [8]. Defining ω_{tj} as 1 if the prediction of h_t corresponds to class j and 0 otherwise, the final decision can be expressed as

$$\widehat{\text{class}}(x) = \arg\max_j \sum_t w_t \omega_{tj}(x) \tag{3}$$

The preservation of previous knowledge and the accommodation of novel information, strongly depends on the relative importance of each classifier. In [13], Polikar et al. proposed the AdaBoost aggregation strategy [3] for algorithm (1). Voting weights are computed as $w_t = \log((1-\eta_t)/\eta_t)$ where η_t is the training error of h_t . In incremental environments this rule becomes not optimal, since classifiers corresponding to different batches might be model different patterns and hence the performances of these classifiers are not directly comparable. For example, it is possible that the batch S_t contains only instances of one class, say 1, so it is not difficult for a classifier to achieve a high accuracy, let say $\eta_t \sim 0$. If new classes appear in the next batch, the accuracy of the corresponding classifier could be significantly lower than say $\eta_t > 0$. The first classifier however is really not better than the second because it represents an incomplete knowledge of the environment.

An idea to overcome this problem is to use instance-dependent weights. In [5], Gangardiwala et al. proposed to obtain $w_t(x)$ as $\min_k 1/\delta_{tk}(x)$, 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 the 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 $k = 1, ..., K$, the k-th class-specific distance of an input instance x to X_{tk} is computed as

$$\delta_{tk}(x) = (x - \mu_{tk})' \cdot \mathbf{C}_{tk}^{-1} \cdot (x - \mu_{tk}) \tag{4}$$

where μ_{tk} is the mean and $\mathbf{C}_{tk}$ the covariance matrix of X_{tk}.

We propose to define the voting weight $w_t(x)$ of the classifier h_t, for predicting label x, as a function that depends on the Mahalanobis distance between the instance x and each class-specific subset X_{tk}. If Mahalanobis distance between x and class-specific subset is zero, hypothesis should have greater weighting, otherwise, the weight decreases, penalizing divergence:

$$\widehat{w}_t(x) = \sum_{k=1}^{K} \exp(-(x - \mu_{tk})' \cdot \mathbf{C}_{tk}^{-1} \cdot (x - \mu_{tk})) = \sum_{k=1}^{K} \exp(-\delta_{tk}) \tag{5}$$

In addition, the coverage that the classifier for each class has, is considered. Suppose that 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 been trained with enough examples of the class k to generalize well. Consider the event $A_k =$ "x is of class k", then $P(k|h_t)$ corresponds to the prior probability of the event A_k given the classifier h_t. Since the knowledge acquired depends on the data,

it seems reasonable to use as the prior $P(k|h_t)$ the fraction of such data that belongs to the class k.

$$\widehat{w}_t(x) = \sum_{k=1}^{K} \exp(-\delta_{tk}) \times P(k|h_t) = \sum_{k=1}^{K} \exp(-\delta_{tk}) \times \frac{|X_{tk}|}{|X_t|} \tag{6}$$

where $|\cdot|$ denotes cardinality, X_t is set of input used to train h_t and X_{tk} the subset of X corresponding to the instances of class k.

Finally, it should be mentioned that knowledge represented by a classifier depends on the classes it was able to learn, that is, the proposed framework should consider how reliable is the knowledge it represents. Then we will use $P(h_t)$ as the probability how much good is the classifier h_t.

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

The determination of this probability may be through the classifier accuracy, even in literature, the accuracy is used for determining the weight of h_t, but it is not the only strategy possible. If suppose, we take a uniform distribution for $P(h_t)$, weight $w_t(x)$ is equal (6). If we consider $P(h_t)$ proportional to accuracy, $P(h_y)$ can be defined as $P(h_y) = \log((1-\eta_t)/\eta_t)$ where η_t is the training error of h_t with S_t. It should be noted that the whole set of classifiers originated after a new batch of observations that arrive to the system are generated to learn the new information contained in these observations. Resampling steps after the first one, make that different classifiers work with partially different data sets.

Hence it makes sense to compute the weight $w_t(x)$ only once 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, which 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 (7) with the set of observations X_t obtained after step 14 of algorithm (1) has been applied for the first time with the current batch.

In non-stationary environments, Scholz [15] used an aggregation strategy capable to follow the dynamic of the drifting observations. Just like the instance selection methods based on sliding windows or example weighting [6,18], where aggregation strategy is biased towards the last observations, our voting strategy can be adapted similarly. Instead of computing the accuracy of the classifier h_j in S_j, we can use the last batch of observations as a better approximation of the distribution of future observations. Then, $P(h_t)$ is recomputed immediately after a new batch of observations become available as $\log((1-\eta_t)/\eta_t)$, but now η_t is the training error of h_t on the most recent sample of examples.

Since a particular batch of observations could be an incomplete description of a stationary frame in the dynamics of the environment, this strategy could be improved if we were able to detect the step in which a drift takes place and if we use the performance of the classifiers in the set of batches after the drift to compute the prior $P(h_t)$. The effect however is attenuated because our

aggregation strategy remains sensitive to the location of the instances to classify in the feature space.

5 Experiments

In [17] we provided comparative results, for problems already studied in [10] and [5] for incremental learning, between our algorithm and the Learn++ algorithm as defined in [5]. Here, we study the behavior of our algorithm in non-stationary environments using one classification problem and three different concept drift scenarios proposed in [15].

The benchmark used to test our algorithm in non-stationary environments consist in the Satellite Image Data obtained from the UCI library [1]. Since it does not contain a known concept drift and in order to allow a comparison, we used the experimental setup proposed in [15]: the data was randomly ordered into a stream and split in 20 batches of equal size (321 examples per batch). Since the KBS-Stream algorithm [15] is designed to deal with binary classification, 2 of the original 6 classes were marked as relevant (class 1) and the other as non-relevant (class 2). The same three concept drift scenarios proposed in [15] were simulated:

1. Scenario A corresponds to an abrupt drift from the first to the second class in batch 10. That is, after the batch 10 examples marked as relevant become not-relevant and viceversa.
2. Scenario B corresponds to a gradual drift from the first to the second class between batches 8 and 12. That is, a linearly increasing fraction of the examples of class 1 becomes of class 2 and viceversa, beginning with 0 in batch 8 and finishing with 1 in batch 12.
3. In Scenario C, an abrupt drift occurs in batch 9, as in scenario A, but it is abruptly reversed in batch 11.

Table (1) shows the best results obtained in the scenarios A, B and C with our framework (7) using accuracy of the classifier without considering concept drift (named *Static Priors*) and using the framework designed for concept drift (named *Adaptive Priors*). The base classifier used is the same on [17]. Tables include the mean and variance of the classification error computed after 20 experimental runs with different random permutations of the examples. Rows of the table correspond to different parameter configurations: number of classifiers per batch (C) and number of neurons (N).

From the results reported above we can conclude that the voting strategy defined to deal with concept drift introduces significant improvements with respect to our original static accuracy based strategy to define the priors over the classifiers. In Scenario A we reduce the misclassification error around 4%, in Scenario B around 3% and in Scenario C, which represents a more complex type of drift, around 2%. Improvements seem independent of the parameter configuration of the algorithms, that is number of classifiers and the number of neurons in the base learners. Moreover, an important reduction of variance is observed

Table 1. Best Results in Non-Stationary Environments

Results in Scenario A				
	Static Priors		*Adaptive Priors*	
Combinations	**Mean Error**	**Variance**	**Mean Error**	**Variance**
20N-2C	14.819315	5.788040	**9.581776**	3.440245
20N-6C	**13.628505**	5.689071	9.853583	3.193592
Results in Scenario B				
	Static Priors		*Adaptive Priors*	
Combinations	**Mean Error**	**Variance**	**Mean Error**	**Variance**
10N-2C	15.162773	4.416926	**11.483645**	3.030324
20N-6C	**14.123832**	3.654565	11.693925	2.787356
Results in Scenario C				
	Static Priors		*Adaptive Priors*	
Combinations	**Mean Error**	**Variance**	**Mean Error**	**Variance**
10N-2C	**11.746885**	4.834248	9.409657	6.536859
20N-4C	12.666667	4.659173	**9.326324**	6.640003

in scenarios A and B (around the half of the original variance). The curves representing the behavior of the algorithm as new batches of observations become available, also show that the algorithm designed for changing environments has a stronger ability to recover for an abrupt or gradual drift. This occurs because the static algorithm can only recover from a drift creating more classifiers representing the new knowledge than the classifiers representing the old knowledge. The dynamic algorithm, on the other hand, is capable to rapidly and selectively reuse old knowledge structures and hence can respond more quickly. This observation, in fact, explains the closer difference between the algorithms in Scenario C.

6 Conclusions

In this paper we have introduced a new voting strategy to incremental learning using an ensemble of classifiers. This strategy identifies a voting weight with two fundamental pieces: (1) a function which depends on the instance to classify and the knowledge represented by the classifier and (2) a prior which represents an instance-independent belief about the ability of the classifier to deal with the environment. By defining both pieces we can obtain an aggregation mechanism with different properties. This paper has examined a model which explores the knowledge cumulated by the classifier in the different class-specific feature spaces and different types of priors. Using priors depending on the overall performance of the classifier in its training set we have obtained an algorithm capable to accommodate new knowledge without compromising previously acquired knowledge and capable to detect the most suitable knowledge substructures to predict a given instance. Experiments in well-known benchmarks show that this algorithm can introduce important improvements or at least competitive results with respect to similar algorithms. Introducing a simple modification on the priors again, we can obtain a new version of the algorithm capable to deal with non-stationary environments. Further improvements could probably be obtained if we were able to detect the specific step in which a drift takes place, and if we

use the performance of the classifiers in the set of batches after the drift to recompute the prior.

References

1. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
2. Fern, A., Givan, R.: Online ensemble learning: An empirical study. Machine Learning 53(1-2), 71–109 (2003)
3. Freud, Y., Schapire, R.: A short introduction to boosting. Journal of Japanese Society for Artificial Intelligence 14(5), 771–780 (1999)
4. Fumera, G., Roli, F.: A theoretical and experimental analysis of linear combiners for multiple classifier systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(6), 942–956 (2005)
5. Gangardiwala, A., Polikar, R.: Dynamically weighted majority voting for incremental learning and comparison of three boosting based approaches. In: Joint Conf. on Neural Networks (IJCNN 2005), pp. 1131–1136 (2005)
6. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intelligent Data Analysis 8(3), 281–300 (2004)
7. Kuncheva, L.I., Bezdek, J.C., Duin, R.P.W.: Decision templates for multiple classifier fusion: An experimental comparison. Pattern Recognition 34(2), 299–314 (2001)
8. Kuncheva, L.: Combining pattern classifiers: Methods and algorithms. Wiley InterScience (2004)
9. Littlestone, N., Warmuth, M.: The weighted majority algorithm. Information and Computation 108(2), 212–261 (1994)
10. Muhlbaier, M., Topalis, A., Polikar, R.: Learn++.MT: A new approach to incremental learning. In: Roli, F., Kittler, J., Windeatt, T. (eds.) MCS 2004. LNCS, vol. 3077, pp. 52–61. Springer, Heidelberg (2004)
11. Oza, N.C.: Online bagging and boosting. In: IEEE International Conference on Systems, Man and Cybernetics, vol. 3, pp. 2340–2345 (2005)
12. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits and Systems 24(4), 21–45 (2006)
13. 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)
14. Scholz, M.: Knowledge-based sampling for subgroup discovery. In: Morik, K., Boulicaut, J.-F., Siebes, A. (eds.) Local Pattern Detection. LNCS (LNAI), vol. 3539, pp. 171–189. Springer, Heidelberg (2005)
15. Scholz, M., Klinkenberg, R.: Boosting classifiers for drifting concepts. Intelligent Data Analysis, Special Issue on Knowledge Discovery from Data Streams 11(1), 3–28 (2007)
16. Todorovski, L., Dzeroski, L.: Combining classifiers with meta decision trees. Machine Learning 50(223), 249 (2003)
17. Trejo, P., Ñanculef, R., Allende, H., Moraga, C.: Probabilistic aggregation of classifiers for incremental learning. In: Sandoval, F., Prieto, A.G., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 135–143. Springer, Heidelberg (2007)
18. Widmer, K., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23, 69–101 (1996)