Published in: F. Roli, J. Kittler and T. Windeatt (Eds.),
Proc. 5th Int. Workshop on Multiple Classifier Systems, Cagliari, Italy
Springer-Verlag, LNCS, **3077**, 2004, 1–15

Classifier Ensembles for Changing Environments

Ludmila I. Kuncheva

School of Informatics, University of Wales, Bangor
Bangor, Gwynedd, LL57 1UT, United Kingdom
l.i.kuncheva@bangor.ac.uk

Abstract. We consider strategies for building classifier ensembles for non-stationary environments where the classification task changes during the operation of the ensemble. Individual classifier models capable of online learning are reviewed. The concept of "forgetting" is discussed. Online ensembles and strategies suitable for changing environments are summarized.

Keywords: classifier ensembles, online ensembles, incremental learning, non-stationary environments, concept drift.

1 Introduction

> *"All things flow, everything runs, as the waters of a river, which seem to be the same but in reality are never the same, as they are in a state of continuous flow."*
>
> The doctrine of Heraclitus.

Most of the current research in multiple classifier systems is devoted to static environments. We assume that the classification problem is fixed and we are presented with a data set, large or small, on which to design a classifier. The solutions to the static task have marvelled over the years to such a perfection that the dominance between the classification methods is resolved by a fraction of percent of the classification accuracy. Everything that exists changes with time and so will the classification problem. The changes could be minor fluctuations of the underlying probability distributions, steady trends, random or systematic, rapid substitution of one classification task with another and so on.

A classifier (individual or an ensemble)[1], if intended for a real application, should be equipped with a mechanism to adapt to the changes in the environment. Various solutions to this problems have been proposed over the years. Here we try to give a systematic perspective on the problem and the current solutions, and outline new research avenues.

The paper is organized as follows. Section 2 sets the scene by introducing the concept of changing environment. Online classifier models are presented in Section 3. Section 4 details some ensemble strategies for changing environments.

[1] Unless specified otherwise, a *classifier* is any mapping $D : \Re^n \to \Omega$ from the feature space, $\Re^n$, to the set of class labels $\Omega = \{\omega_1, \ldots, \omega_c\}$.

2 The changing environment

Changing environments pose the main hurdle in many applications. One of the most acute examples is detecting and filtering out spam e-mail.[2] The descriptions of the two classes of e-mail, "spam" and "non-spam", evolve with time; they are user-specific, and user preferences may also evolve with time. Besides, the important discriminating variables used at time t to classify spam may be irrelevant at a future moment $t+k$. To make matters even worse, the variability of the environment in this scenario is hardly due to chance. The classifier will have to face an active opponent – the "spammers" themselves – who will keep coming up with ingenious solutions to trick the classifier into labeling a spam e-mail as legitimate.

2.1 Concept drift and types of changes

Viewed in a probabilistic sense, a classification problem may change due to the changes in [13]

- Prior probabilities for the c classes, $P(\omega_1), \ldots, P(\omega_c)$;
- Class-conditional probability distributions, $p(\mathbf{x}|\omega_i)$, $i = 1, \ldots, c$; or
- Posterior probabilities $P(\omega_i|\mathbf{x})$, $i = 1, \ldots, c$.

Not every change in the distribution is going to degrade the performance of a classifier. Consider the minimum-error classifier which labels $\mathbf{x}$ as the class index of the largest posterior probability $P(\omega_i|\mathbf{x})$. If the largest posterior probability for every $\mathbf{x} \in \Re^n$ keeps its class index, then the decision of the classifier for this $\mathbf{x}$ will guarantee the minimum error no matter what the changes in the distributions are. Kelly et al. [13] call the changes in the probabilities *population drift* and remark that a notion of *concept drift* used in machine learning literature is the more general one.

While in a natural system we can expect gradual drifts (e.g., seasonal, demographic, habitual, etc.), sometimes the class description may change rapidly due to *hidden contexts*. Such contexts may be, for example, illumination in image recognition and accents in speech recognition [24]. The context might instantly become highly relevant. Consider a system trained on images with similar illumination. If images with the same type of content but a different illumination are fed to the system, the class descriptions might change so as to make the system worse than a random guess. The type of changes can be roughly summarized as follows

- Random noise [1, 23]
- Random trends (gradual changes) [13]
- Random substitutions (abrupt changes) [23]
- Systematic trends ("recurring contexts") [23]

[2] The term SPAM is coined to denote unsolicited e-mail, usually of commercial or offensive nature.

Depending on the type of changes, different strategies for building the classifier may be appropriate. The noise must not be modelled by the classifier but filtered out. On the other hand, if there are systematic changes whereby similar class descriptions are likely to reappear, we may want to keep past successful classifiers and simply reuse them when appropriate. If the changes are gradual, we may use a moving window on the training data. If the changes are abrupt we may choose to use a static classifier and when a change is detected, pause and retrain the classifier. This scenario is important when verification of the class labels of the streaming data is not easily available. In general, the more is known about the type of the context drift, the better the chances of devising a successful updating strategy.

2.2 Detecting a change

Unlabeled data. Sometimes online labeling is not straightforward. For example, in scanning mammograms for lesions, a verified diagnosis from a specialist is needed if we want to reuse the processed images for further learning. Another example is the credit application problem [13] where the true class label (good/bad) becomes known two years after the classification has taken place. In spam e-mail filtering, the user must confirm the legitimacy of every message if online training is intended.

In case of unknown labels of the streaming data the classifier should be able to signal a potential concept drift based on the unlabeled data. This is typically based on monitoring the unconditional probability distribution, $p(\mathbf{x})$. This problem is called *novelty detection* [18]. It is related to outlier detection in statistics.

One possible practical solution is to train an additional "classifier" to model $p(\mathbf{x})$ and compare the value for each input $\mathbf{x}$ with a threshold θ. If $\mathbf{x}$ is accepted to having come from the distribution of the problem ($p(\mathbf{x}) > \theta$), the system proceeds to classify it. Else we refuse to classify $\mathbf{x}$ and increment the count of novel objects. We can keep the count over a window of past examples. When the proportion of novel examples reaches a certain level, the system should either go to a halt or request a fresh labeled sample to retrain itself (if this option is incorporated). This addition to the original classifier requires 3 parameters to be specified by the user: θ, the threshold for the novelty; the threshold for the proportion of novel examples; and finally the size of the window. In the simplest case, the distance to the nearest neighbor from the current training data set can be used and θ could be a threshold distance. In theory this technique is equivalent to nonparametric modelling of $p(\mathbf{x})$. More sophisticated modelling approaches include Gaussian Mixture Modelling, Hidden Markov Models, kernel approximation, etc. (see the survey [18]).

Note that using only the unconditional distribution, $p(\mathbf{x})$, we may miss important changes in the class distributions which may alter the classification task but not $p(\mathbf{x})$.

Labeled data. The most direct indication that there has been an adverse change in the classification task is a consistent fall in the classification accuracy, either sudden or gradual.

2.3 Learn to forget

Suppose that the classifier has the ability to learn on-line. Instead of trying to detect changes, the classifier is kept constantly up-to-date, which is called "*any time learning*". This means that if the system is stopped at time t, we have the best classifier for the classification problem as it is at time t. The classifier should be able to learn new class descriptions and "forget" outdated knowledge, or "unlearn". The main problem is how to choose the rate of forgetting so that it matches the rate and the type of the changes [15, 23]. If the character of the changes is known or at least suspected, e.g., gradual seasonal changes, then an optimal forgetting strategy can be designed.

Forgetting by ageing at a constant rate. The most common solution is forgetting training objects at a constant rate and using the most recent training set to update (re-train) the classifier. The current classifier uses the past w objects. When a new object arrives, the classifier is updated so as if trained on a data set where the oldest observation is replaced by the newest one.

How large a window do we need? If the window is small, the system will be very responsive and will react quickly to changes but the accuracy of the classifier might be low due to insufficient training data in the window. Alternatively, a large window may lead to a sluggish but stable and well trained classifier. The compromise between the two is viewed as the "stability-plasticity dilemma" [15] whose solution lies with adjusting the "forgetting" parameters for the concrete problem.

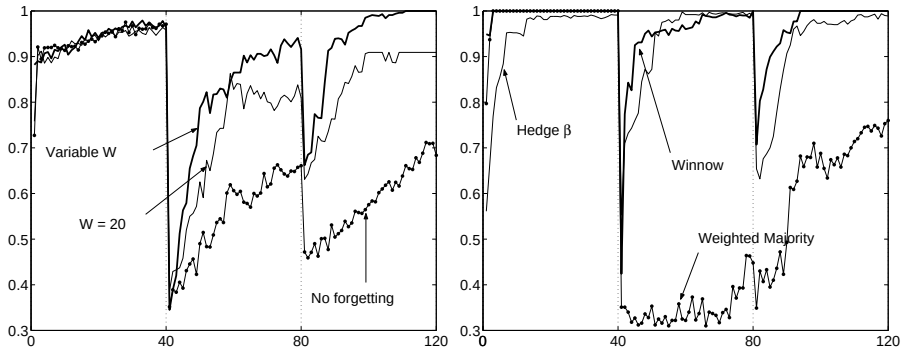
Forgetting by ageing at a variable rate. If a change is detected, the window is shrunk (past examples are forgotten).[3] For a static bout, the window is expanded to a predefined limit.

To illustrate the concepts being introduced we will keep along a synthetic example taken from [23]. There are 3 categorical features with three categories each: size $\in$ {small, medium, large}, colour $\in$ {red, green, blue} and shape $\in$ {square, circular, triangular}. There are three classification tasks to be learned. The first class to be distinguished is (size = small AND colour = red). 40 examples are generated randomly (with a uniform distribution over the possible values) and labeled according to the class description. These are fed to the classifier sequentially. An independent set of 100 objects labeled according to the current class description is generated for each of the 40 objects. The classifier is tested after each submission on the respective 100 examples. The class description is changed at step 40, so that objects 41 to 80 are labeled according to class (colour = green OR shape = circular). The testing objects are generated again from a uniform random distribution and labeled according to the current class

[3] FLORA 3 is an example of this group of methods [23].

description. Finally, a last set of 40 objects is generated (from 81 to 120) and labeled according to class (size = small OR size = large). The largest of the two prior probabilities for the first stage is 0.89, for the second stage is 0.56, and for the third stage, 0.67.

The importance of using a valid forgetting strategy is demonstrated in Figure 1(a). The Naive Bayes classifier is trained incrementally by updating the probabilities for the classes with each new data point (each example). The plot shows the accuracies of three classifiers versus the number of examples processed. The graph is the average of 10 runs. The classifier based on windows of variable size appears to be more versatile and able to recover from an abrupt change of class concept than the classifier with a constant window and the "no forgetting" classifier.



(a) Naive Bayes classifier without and with forgetting (fixed and variable window)

(b) Hedge β, Winnow and Weighted Majority for an ensemble of 3 ideal classifiers

Fig. 1. Classification accuracy for online algorithms.

Density-based forgetting. Sometimes older data points may be more useful than more recent points. Deleting training data based on the distribution is considered in adaptive nearest neighbors models [1, 20]. A weight is attached to each data point. The weight may decay with time or may be modified depending on how often the object has been found as a nearest neighbor.

3 Online learning

Online learning is becoming increasingly a center stage methodology in the current information era. Massive streams of data are being processed every day, exceeding by far the time and memory capacity of the ever improving modern computational technology. Examples of large data streams can be found in telecommunications, credit card transactions, Internet searches, etc. [6, 7, 21, 22].

Online learning, also termed incremental learning, is primarily focused on processing the data in a sequential way so that in the end the classifier is no worse than a (hypothetical) classifier trained on the batch data. A static environment is typically assumed, so forgetting of the learned knowledge is not envisaged. Each submitted data point is classified and at some later stage its true label is recovered. The data point and its label are added to the training set of the classifier. The classifier is updated, minimally if possible, to accommodate the new training point. In the simplest case the true label is known immediately. Adapting to the changing environment may come as an effortless bonus in online learning as long as some forgetting mechanism is put into place.

3.1 Desiderata for online classifiers

A good online classifier should have the following qualities [7, 21]

- *Single pass through the data.* The classifier must be able to learn from each example without revisiting it.
- *Limited memory and processing time.* Each example should be processed in a (small) constant time regardless of number of the examples processed in the past.
- *Any-time-learning.* If stopped at time t the algorithm should provide the best answer. The trained classifier should ideally be equivalent to a classifier trained on the batch data up to time t. An algorithm which produces a classifier functionally equivalent to the corresponding classifier trained on the batch data is termed a *lossless* online algorithm [19].

In fact, this list applies also for classifiers able to learn in changing environments. Any-time-learning accounts for the need for adapting the training in case of a concept drift.

3.2 Online classifier models and their extensions for changing environments

One of the oldest online training algorithms is the Rosenblatt's perceptron even though it only possesses the second one of the desired qualities.

Learning vector quantization (LVQ). LVQ is a simple and successful online learning algorithm originated also from the neural network literature [14]. The principle is as follows. We initialize (randomly or otherwise) a set of points labeled in the classes of the problem. The points are called prototypes. At the presentation of a new object, $\mathbf{x} \in \Re^n$, the prototypes' locations in the feature space are updated. The prototypes of the same class as $\mathbf{x}$ are pulled toward $\mathbf{x}$ while the prototypes from different classes are pushed away. In the simplest version only the nearest to $\mathbf{x}$ prototype is updated. The magnitude of the movement is controlled by a parameter of the algorithm, α, called the learning rate. Let $\mathbf{v} \in \Re^n$ be a prototype. The new value of the prototype is

$$\mathbf{v}^{\text{new}} = \begin{cases} \mathbf{v} + \alpha(\mathbf{x} - \mathbf{v}), & \text{if } \mathbf{v} \text{ has the same class label as } \mathbf{x} \\ \mathbf{v} - \alpha(\mathbf{x} - \mathbf{v}), & \text{otherwise} \end{cases}$$

For training a classifier in stationary environment the learning rate α is typically chosen as a decreasing function of the number of iterations so that at the end of the algorithm only small changes take place. We can regard α as a function of the accuracy of the classifier and enforce more aggressive training when a change in the environment is detected.

Decision trees. To understand how the updating works, recall how a decision tree is built. Starting with a root node, at each node we decide whether or not the tree should be split further. If yes, a feature and its threshold value are selected for that node so as to give the best split according to a specified criterion. Upon receiving a new object $\mathbf{x}$, it is propagated down the tree to a leaf forming a path $P_{\mathbf{x}}$. The counts of training objects reaching the nodes on the path $P_{\mathbf{x}}$ are updated. For every internal node, the feature chosen for the split is confirmed along with the threshold value for the split. All necessary alterations are made so that the current tree is equivalent to a tree built upon the whole data set of all past $\mathbf{x}$'s.

Updating a tree may be an intricate job and may take longer than may be acceptable for an online system processing millions of records a day. Domingos and Hulten [6] propose a system for this case called VFDT (Very Fast Decision Trees). They use Hoeffding bound to guarantee that the VFDT will be asymptotically equivalent to the batch tree. The Hoeffding bound allows to calculate the sample size needed to estimate a confidence interval for the mean of the variable of interest, regardless of the distribution of the variable.[4] The importance of the bound is that after a certain number of input points has been processed, there is no need to update the tree further. The VFDT system is guaranteed to be asymptotically optimal for static distributions [6,7]. Hulten et al. [11] extend VFDT to Concept-adapting VFDT to cope with changing environment.

Naive Bayes. In the Naive Bayes model, the class-conditional probabilities are updated with each new $\mathbf{x}$. For simplicity, suppose that $\mathbf{x}$ consists of n categorical variables $x_1, x_2, \ldots, x_n$. For each variable, we keep a probability mass function over the possible categories for each of the classes, $P(x_k|\omega_i)$. When a new $\mathbf{x}$ is received, labeled in class ω_j, the probabilities for ω_j are updated. For example, let $\mathbf{x}$ = (small,blue,circular) and let the label of $\mathbf{x}$ be ω_1. Suppose that 99 objects from class ω_1 have been processed hitherto. Let $P(x_1 = \text{S} \mid \omega_1) = 0.2$, $P(x_1 = \text{M} \mid \omega_1) = 0.7$, and $P(x_1 = \text{L} \mid \omega_1) = 0.1$. The updated values for x_1 are

$$P(x_1 = \text{S}|\omega_1) = \frac{0.2 \times 99 + 1}{100} = 0.208 \qquad P(x_1 = \text{M}|\omega_1) = \frac{0.7 \times 99}{100} = 0.693$$

[4] The Hoeffding bound is also known as additive Chernoff bound [6]. It states that for any $\delta \in (0,1)$, with probability at least $1-\delta$ and irrespective of the true distribution of the random variable y with range R, the true mean of y is within ϵ of the mean $\bar{y}$ calculated from an i.i.d. sample of size n taken from the distribution of y where

$$\epsilon = \sqrt{\frac{R^2 \ln(2/\delta)}{2n}}.$$

$$P(x_1 = \mathrm{L}|\omega_1) = \frac{0.1 \times 99}{100} = 0.099.$$

Naive Bayes classifier is a lossless classifier. A forgetting mechanism can be incorporated by keeping a window and "unlearning" the oldest example in the window while learning $\mathbf{x}$.

Neural Networks. The training of neural networks is carried out by submitting the training set in a sequential manner a specified number of times (number of epochs). To train a NN online we abandon the epoch protocol and use the continuous stream of data instead. NN classifier is not a lossless model [19]. If we keep the training on-going with the data stream, the NN parameters will follow the concept drift. The responsiveness of the NN to changes will depend on the learning rate used in the backpropagation algorithm. A neural architecture called PFAM (Probabilistic Fuzzy ARTMAP neural network) is used as the base classifier for an ensemble suitable for online learning in [15].

Nearest Neighbor. Nearest neighbour classifier is both intuitive and accurate. We can build the training set (the reference set from which the neighbors are found) by storing each labeled $\mathbf{x}$ as it comes. This model is called IB1 (instance-based learning, model 1) in [1]. IB1 is a lossless algorithm but it fails on the criteria for time and memory. IB2 accepts in the reference set only the $\mathbf{x}$'s which are misclassified using the reference set at the time they arrive. This is the online version of the Hart's *editing algorithm* [10]. Editing algorithms look for the minimal possible reference set with as high as possible generalization accuracy. IB3 introduces forgetting based on the usefulness of the $\mathbf{x}$'s in the reference set. There have been many studies on editing for the nearest neighbour algorithm [2,5]. Developing efficient online editing algorithms could be one of the important future directions.

4 Ensemble strategies for changing environments

When and why would an ensemble be better than a single classifier in a changing environment?

In massive data streams we are interested in simple models because there might not be time for running and updating an ensemble. On the other hand, Wang et al. argue that a simple ensemble might be easier to use than a decision tree as a single adaptive classifier [22].

When time is not of primary importance but very high accuracy is required, an ensemble would be the natural solution. An example is scanning mammograms for tissue lesions or cervical smear images for cell abnormalities. In these cases taking several minutes per image will be acceptable.

Various online ensemble solutions have been proposed for changing environments. We can group the approaches as follows

– Dynamic combiners (or "horse racing" algorithms). It this group we put the ensemble methods where the individual classifiers (experts) are trained in

advance and the changes in the environment are tracked by changing in the combination rule.
- Updated training data. The algorithms in this group rely on using fresh data to make online updates of the team members. The combination rule may or may not change in the process.
- Updating the ensemble members. Classifiers in the online ensemble can be updated online or retrained in a batch mode if blocks of data are available.
- Structural changes of the ensemble. "Replace the loser" is one possible strategy from this group. In the case of a change in the environment, the individual classifiers are re-evaluated and the worst classifier is replaced by a new classifier trained on the most recent data.
- Adding new features. The features will naturally change their importance along the life of the ensemble. There might be a need for incorporating new features without going through the loop of re-designing the entire ensemble system.

Some of the approaches are detailed below.

4.1 "Horse racing" ensemble algorithms

We will follow the horse racing analogy aptly used in [8] to introduce the Hedge-β algorithm and AdaBoost. Assume that you want to predict the outcome of a horse race and you have L expert-friends whose prediction you can take or ignore. Each $\mathbf{x}$ is a particular race and the class label is its outcome. You note which of the experts have been wrong in their prediction and update a set of weights to keep track on whose prediction is currently most trustworthy. The most famous representative of this group is the **Weighted Majority** algorithm by Littlestone and Warmuth [17], shown below.

1. Given is a classifier ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$.
   Initialize all L weights as $w_j = 1$.
2. *Operation.* For a new $\mathbf{x}$, calculate the support for each class ω_i as the sum of the weights of all classifiers D_i that suggest class label ω_i for $\mathbf{x}$.
   Label $\mathbf{x}$ to the class with the largest support.
3. *Training.* Observe the true label of $\mathbf{x}$. Using a pre-defined $\beta \in [0, 1]$, update the weights for all experts whose prediction was incorrect as $w_i \leftarrow \beta w_i$.
4. Continue from Step 2.

Hedge β operates in the same way as the Weighted Majority algorithm but instead of taking the weighted majority, one classifier is selected from the ensemble and its prediction is taken as the ensemble decision. The classifier is selected according to the probability distribution defined by the normalized weights.

Winnow is another algorithm under the horse race heading [16]. The algorithm was originally designed for predicting the value of a (static) Boolean function, called target function, $f : \{0, 1\}^n \rightarrow \{0, 1\}$, by getting rid of the irrelevant variables in the Boolean input. Translated into our ensemble terminology, Winnow

follows the weighted majority algorithm but uses a different updating Step 3. The weights are reconsidered only if the ensemble gives a wrong prediction for the current input $\mathbf{x}$. If classifier D_i gives the correct label for $\mathbf{x}$, its weight is increased as $w_i \leftarrow \alpha w_i$, where $\alpha > 1$ is a parameter of the algorithm. This is called a *promotion step*, rewarding the expert not so much for predicting correctly but for predicting correctly in times when the global algorithm should have listened to them more carefully [3]. If classifier D_i gives an incorrect label for $\mathbf{x}$, the *demotion step* takes place as $w_i \leftarrow w_i/\alpha$, ensuring that D_i takes a share of the blame for the ensemble error. Good results have been obtained for $\alpha = 2$ [16]. Machine learning literature abounds in proofs of theoretical bounds and further variants of Winnow and Weighted Majority [24].

Figure 1(b) shows the average of 10 runs of Hedge β, The Weighted Majority and the Winnow algorithms for the artificial data described above. Here we are "cheating" in a way because we suppose that the ensemble consists of three perfect classifiers, one for each stage. For example, if classifier D_1 was always picked to make the decision for objects 1 to 40, classifier D_2 for objects from 41 to 80, and classifier D_3 for objects from 81 to 120, the classification accuracy would be 100% for all objects. As the plot shows, the Weighted Majority cannot recover from changing the class description at object 40 even though the ensemble consists of the three optimal experts. Hedge β would follow the same pattern if applied in its standard form. Here we modified it a little. Instead of updating all the weights for each $\mathbf{x}$, we update only the weight of the classifier selected to label $\mathbf{x}$. Winnow is the undisputed winner between the three algorithms in the plot. It quickly discovers the right classifier for each part. Obviously, involving the ensemble accuracy in the weight updating step makes all the difference.

Blum [3] describes the above algorithms as "learning simple things really well". The problem with the horse racing approach, in the context discussed here, is that the individual classifiers are not re-trained at any stage. Thus their expertise may ware off and the ensemble may be left without a single adequate expert for the new environment. Classifiers in the original algorithms are taken from the whole (finite) set of possible classifiers for the problem, so there is no danger of lack of expertise.

Mixture of experts type of training constitutes a special niche in the group of dynamic combiner methods. Originally developed as a strategy for training an ensemble of multi-layer perceptrons [12], the idea has a more generic standing as an on-line algorithm suitable for changing environments. The ensemble operates as an on-line dynamic classifier selection system and updates one classifier and the combination rule with each new example. Figure 2 shows a sketch of a training cycle upon a presentation of a new data point $\mathbf{x}$. The individual classifiers must be supplied with a forgetting mechanism so that outdated knowledge is "unlearned".

Good old combination rules. Any combination rule can be applied for the online ensemble. Although there is a marked preference for majority vote [21]

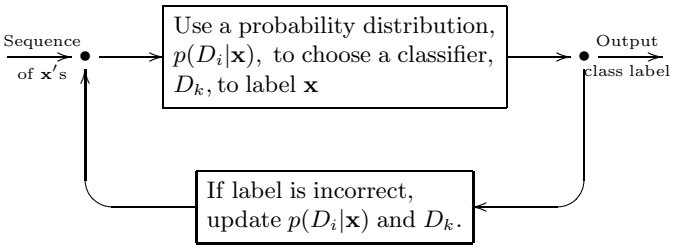


Fig. 2. A generic on-line ensemble construction method based on dynamic classifier selection.

and weighted majority, there is no reason why we should stop here. Naive Bayes and BKS combination rules are explored in [15]. Both can be updated online.

4.2 Updated training data for online ensembles

Reusing the data points. Oza proposes a neat **online bagging** algorithm which converges to the batch bagging as the number of training examples and the number of classifiers tend to infinity [19]. The idea is that the training samples for the L classifiers in the ensemble can be created incrementally. The base classifiers are trained using online (preferably lossless) classifier models. To form the training sample for classifier D_i, Oza observes that each training data point appears K times in a training set of size N sampled with replacement from the available data set of size N. K is a binomial random variable, so the probability that a certain $\mathbf{z}$ appears $K = k$ times in the ith training set is

$$P(K = k) = \binom{N}{k} \left(\frac{1}{N}\right)^k \left(1 - \frac{1}{N}\right)^{N-k}$$

When N is large the probability for selecting a particular data point is small, and the binomial distribution can be approximated by a Poisson distribution with $\lambda = 1$. Therefore the probability $P(K = k)$ is calculated as $P(K = k) = \frac{\exp(-1)}{k!}$. The probabilities for $k = 0 \ldots, 7$ are tabulated below. Practically it is very unlikely that any $\mathbf{z}$ will be present in a sample more than 7 times.

k	0	1	2	3	4	5	6	7
$P(K = k)$	0.3679	0.3679	0.1839	0.0613	0.0153	0.0031	0.0005	0.0001

The online bagging proposed by Oza is given below

1. Pick the number of classifier in the ensemble, L.
2. *Training.* For each new labeled data point $\mathbf{x}$, for each classifier D_i, $i = 1, \ldots, L$, sample from the Poisson distribution explained above to find k. Put k copies of $\mathbf{x}$ in the training set of D_i. Retrain D_i using an online training algorithm.

3. *Operation.* Take the majority vote on the ensemble to label $\mathbf{x}$.
4. Continue from Step 2.

The **online boosting** algorithm proposed next by Oza [19] draws upon a similar idea as his online bagging. The number of times that a data point appears in the training set of classifier D_i is again guided by the Poisson distribution. However, the parameter of the distribution, λ will change from one classifier to the next. We start with $\lambda = 1$ to add copies of $\mathbf{x}$ to the first sample and train classifier D_1 on it. If D_1 misclassifies $\mathbf{x}$, λ increases so that $\mathbf{x}$ is likely to feature more often in the training set of D_2. The subsequent λs are updated in the same manner until all L classifiers are retrained.

Filtering. Breiman suggests to form the training sets for the consecutive classifiers as the data flows through the system, called **Pasting small votes**. [4]. The first N data points are taken as the training set for D_1. The points coming next are filtered so that the next classifier is trained on a sample such that the ensemble built so far (just D_1 for now) will misclassify approximately half of it. To form the sample for classifier D_{k+1}, run each new data point through the current ensemble. If misclassified, add it to the new training set. If correctly classified, add it to the training set with probability $\frac{e_k}{1-e_k}$, where e_k is an estimate of the error of the ensemble on the training set for classifier D_k. To calculate this estimate, Breiman suggests to use the smoothed version

$$e_k = 0.75 \times e_{k-1} + 0.25 \times r_k,$$

where e_{k-1} is the *generalization* ensemble error[5] up to classifier D_{k-1} and r_k is the ensemble error found during building the training set for classifier D_k. The smoothing is needed because if N is small, then the estimate $r(k)$ will be noisy. In this way, L classifiers are subsequently trained on diverse data sets.

Breiman proposes to use e_k as a stopping criterion. In case of static environment, the ensemble error will comfortably level off at some L and no more ensemble members will be needed. If, however, the environment changes, e_k will start increasing again. Changes in the ensembles will be needed at this point. We can, for example, replace ensemble members selected through a certain criterion. Alternatively, we can keep an ongoing process of building classifiers for the ensemble and "forget" the oldest member each time a new member is added, thus keeping the ensemble size constant at L.

The methods in this subsection have been proposed as variants of the corresponding batch methods for stationary environments. When faced with changing environments, we have to introduce a forgetting mechanism. One possible solution would be to use a window of past examples, preferably of variable size, as discussed earlier.

Using data blocks or chunks. In this model we assume that the data come as blocks of data points at a time. The ensemble can be updated using batch mode

[5] This error may be measured on an independent testing set or on the so called "out-of-bag" data points. For details see [4].

training on a "chunk" of data [9, 21, 22]. The blocks can be treated as single items of data in the sense that we may train the ensemble on the most recent block, on a set of past blocks or on the whole set of blocks. A forgetting strategy needs to be incorporated in the model, e.g., a window of blocks. Ganti et al. [9] propose a change detection algorithm based on difference between blocks of data. They also consider selective choices of past blocks so that a certain pattern is modelled. For example, if the blocks come once a day and contain all the data for that day, it might be interesting to build an ensemble for the Wednesday's classification problem and apply it on the subsequent Wednesdays.

Figure 3 gives an illustration of the three data handling approaches.

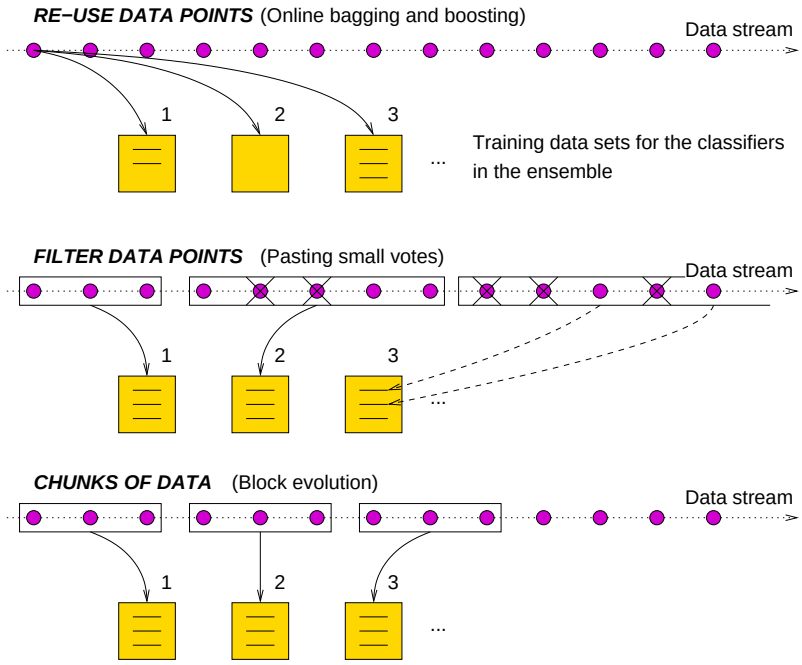


Fig. 3. Illustration of the three data handling approaches for building online classifier ensembles.

4.3 Changing the ensemble structure

The simplest strategy here can be named **replace the oldest**. We remove the oldest classifier in the ensemble and train a new classifier to take its place. The three methods of data handling described above can be run within this framework.

We may adopt a more sophisticated strategy for pruning of the ensemble. Wang et al. [22] propose to evaluate all classifiers using the *most recent* "chunk" of data as the testing set. The classifiers whose error exceeds a certain threshold are discarded from the ensemble. We call this strategy **replace the loser**. Street and Kim [21] consider a "quality score" for replacing a classifier in the ensemble based on its merit to the ensemble, not only on the basis of its individual accuracy. This quality score is a soft version of the Winnow approach for updating the weights of the classifiers. Suppose we are testing the ensemble members on a new data set. We start with equal weights for all ensemble members. For each $\mathbf{x}$, let P_1 be the proportion of votes for the majority class, P_2 be the proportion of votes for the second most voted for class, P_c be the proportion for the correct class of $\mathbf{x}$, and P_i be the proportion for the class suggested by classifier D_i. The score for D_i is updated as $w_i \leftarrow w_i + \delta(\mathbf{x})$, where $\delta(\mathbf{x})$ is

$$\begin{array}{ll} 1-|P_1-P_2|, & \text{if both the ensemble and } D_i \text{ are correct} \\ 1-|P_1-P_c|, & \text{if the ensemble is wrong and } D_i \text{ is correct} \\ 1-|P_i-P_c|, & \text{if } D_i \text{ is wrong (regardless of the ensemble)} \end{array}$$

Most of the methodologies for building ensembles considered hitherto have built-in methodologies for controlling the ensemble size. Alternatively, the ensemble size can be left as a parameter of the algorithm or a designer's choice.

5 Conclusions

Knowing that ensemble methods are accurate, flexible and sometimes more efficient than single classifiers, the purpose of this study was to explore classifier ensembles within changing classification environments. "Concept drift" has long been an ongoing theme in machine learning. Ensemble methods have been recently probed in this line. The frontier of advanced research in multiple classifier systems which we are reaching now is likely to bring new fresh solutions to the changing environment problem.

The ensemble can be perceived as a living population – expanding, shrinking, replacing and retraining classifiers, taking on new features, forgetting outdated knowledge. Ideas can be borrowed from evolutionary computation and artificial life. Even if we are still a long way from putting together a toolbox and a theoretical fundament, the road ahead is fascinating.

References

1. D. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
2. J.C. Bezdek and L.I. Kuncheva. Nearest prototype classifier designs: An experimental study. *International Journal of Intelligent Systems*, 16(12):1445–1473, 2001.
3. A. Blum. Empirical support for Winnow and weighted-majority based algorithms: results on a calendar scheduling domain. In *Proc. 12th International Conference on Machine Learning*, pages 64–72, San Francisco, CA., 1995. Morgan Kaufmann.

4. L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36:85–103, 1999.
5. B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1990.
6. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
7. P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12:945–949, 2003.
8. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
9. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations Newsletter*, 3:1–10, 2002.
10. P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 16:515–516, 1968.
11. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *In Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106. ACM Press, 2001.
12. R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
13. M. G. Kelly, D. J. Hand, and N. M. Adams. The impact of changing populations on classifier performance. In *Proc 5th ACM SIGDD International Conference on Knowledge Discovery and Data Mining*, pages 367–371, San Diego, CA, 1999. ACM Press.
14. T. Kohonen. *Self-Organizing Maps*. Springer, New York, 3rd edition, 2000.
15. C. P. Lim and R. E. Harrison. Online pattern classification with multiple neural network systems: An experimental study. *IEEE Transactions on Systems, Man, and Cybernetics*, 35:235–247, 2003.
16. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.
17. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inform. Computation*, 108:212–261, 1994.
18. M. Markou and S. Singh. Novelty detection: a review. Part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.
19. N. C. Oza. *Online Ensemble Learning*. PhD thesis, University of California, Berkeley, 2001.
20. M. Salganicoff. Density-adaptive learning and forgetting. In *Proceedings of the 10th International Conference on Machine Learning*, pages 276–283, 1993.
21. W. N. Street and Y. S. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382. ACM Press, 2001.
22. H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM Press, 2003.
23. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.
24. G. Widmer and M. Kubat. Special Issue on Context Sensitivity and Concept Drift. *Machine Learning*, 32, 1998.