LEARN++ USING DYNAMIC WEIGHTING ENSEMBLES
G.T. PRASANNA KUMARI
Asso.Prof, Research Scholar, Dept of Computer Science and Engineering, Gokula Krishna College of Engg, Sullurpet-524121, AP, INDIA, tabi_prasanna@yahoo.com

ABSTRACT
In this paper, we propose an incremental learning model for ensemble classifier systems. Learn++, an ensemble of classifiers based algorithm originally developed for incremental learning, and now adapted for information/data fusion applications. Recognizing the conceptual similarity between incremental learning and data fusion, Learn++ follows an alternative approach to data fusion, i.e., Generates each classifier of the ensemble for each data set coming from a different source, and appropriately combine the classifier outputs to take advantage of the additional information in subsequent data sources using “weighted majority voting” (WMV) combination rule. Learn++ creates additional classifiers trained on the new data until the old classifiers do not Out-voted, when classifying an instance from a new class. In order to overcome the problem of out-voting when new data comes and to create new classifiers, we propose Learn++ using dynamic weighting ensemble. Learn++ suffers from the inherent “out-voting” problem when asked to learn new classes, which causes it to generate an unnecessarily large number of classifiers. This paper proposes a modified version of this algorithm, called Learn++MT that not only reduces the number of classifiers generated, but also provides performance improvements. This paper proposes an incremental learning algorithm framework using dynamic weighting ensembles to effectively learn new batches of data appearing over time without the need of retraining. This algorithm helps to learn additional information from new batches of data incrementally while preserving previously acquired knowledge. We demonstrate that the proposed incremental learning algorithm using dynamic weighting scheme has comparable performance with fixed weighting scheme. When a new batch of data becomes available, a new ensemble of basic classifiers is built solely on it so that the new information can be effectively extracted, without interfering with existing classifiers. Another advantage is that the training process has much better flexibility than the retraining strategy.

Keywords - Incremental learning, data fusion, learn++, weighted majority voting

I. INTRODUCTION
One of the biggest frustrations that many researchers working on classifiers face is that most classifiers cannot be “further trained” with new data without forgetting what has been learned earlier. This is particularly true for many of the most common neural network schemes, such as multilayer perceptron, radial basis function network, etc. Learning additional information provided by new data requires discarding the old network, combining old and new data and re-training from scratch. This solution not only forgets what has been learned earlier (also known as catastrophic forgetting), but it is useless if the original data is no longer available. The new algorithm, called Learn++, uses an ensemble of classifiers instead of a single classifier to learn incrementally, and it has shown to be effective in incremental learning, even when additional data introduce new classes.

In many applications that call for automated decision making, it is not unusual to receive data obtained from different sources that may provide complementary information. A suitable combination of such information is usually referred to as data or information fusion, and can lead to improved accuracy and confidence of the classification decision compared to a decision based on any of the individual data sources alone. Consequently, both incremental learning and data fusion involve learning from different sets of data. If the consecutive datasets that later become available are obtained from different sources and/or consist of different features, the incremental learning problem turns into a data fusion problem. Therefore a suitable modification of Learn++ can be used for data fusion, where a new ensemble of classifiers are generated for each source that generates a different database.

Classification algorithms usually require an adequate and representative set of training data to generate an appropriate decision boundary among different classes. This requirement still holds even for ensemble (of classifiers) based approaches that resample and reuse the training data. However, acquisition of such data for real-world applications is often expensive and time consuming. Hence, it is not uncommon for the entire data set to gradually become available in small batches over a period of time. In such settings, an existing classifier may need to learn the novel—or supplementary—information content in the new data without forgetting the previously acquired knowledge and without requiring access to previously seen data. The ability of a classifier to learn under these circumstances is commonly referred to as “incremental learning.” On the other hand, in many applications that call for automated decision making, it is not unusual to receive data obtained from different sources that may provide complementary information. A suitable combination of such information is known as “data” or “information” fusion, and can lead to improved accuracy of the classification decision.
compared to a decision based on any of the individual data sources alone. Consequently, both incremental learning and data fusion involve learning from different sets of data. If the consecutive data sets that later become available are obtained from different sources and/or consist of different features, the incremental learning problem turns into a data fusion problem. Recognizing this conceptual similarity, we propose an approach based on an ensemble of classifiers—originally developed for incremental learning—as an alternative and surprisingly well-performing approach to data fusion.

II. INCREMENTAL LEARNING

Broadly speaking, the phrase “incremental learning” refers to classification tasks where only part of the training data are available in the initial stage and the classifier has to keep learning from new data whenever possible. In this situation, it is desirable for the classifier to be able to learn new patterns and never forget old memories (knowledge) due to the learning of new patterns. Within this scope, various algorithms have been proposed focusing on the growing or pruning of classifier architectures, selection of the most informative training samples, or the modification of classifier weights. In addition to the above two conditions, algorithms should not require access to the original data and should be able to accommodate new classes that may be introduced from the new data. Under these conditions, the authors propose the algorithm Learn++, which inherits a popular ensemble method AdaBoost to solve incremental learning problems. It iteratively generates an ensemble of classifiers for each data set that becomes available, and combines them using weighted majority voting.

An incremental algorithm Learn++, which learns novel information from consecutive data sets by generating an ensemble of classifiers with each data set, and combining them by weighted majority voting. However, Learn++ suffers from an inherent outvoting problem when asked to learn a new class introduced by a subsequent data set, as earlier classifiers not trained on this class are guaranteed to misclassify new instances. The collective votes of earlier classifiers, for an inevitably incorrect decision, then outweigh the votes of the new classifiers' correct decision on new instances—until there are enough new classifiers to counteract the unfair outvoting. This forces Learn++ to generate an unnecessarily large number of classifiers. Learn++.NC, specifically designed for efficient incremental algorithm of multiple new classes using significantly fewer classifiers. To do so, Learn++.NC introduces dynamically weighted vote and vote (DW-CAV), a novel voting mechanism for combining classifiers: individual classifiers consult with each other to determine which ones are most qualified to classify a given instance, and decide how much weight, if any, each classifier's decision should carry. Experiments on real-world problems indicate that the new algorithm performs remarkably well with substantially fewer classifiers, not only as compared to its predecessor Learn++, but also as compared to several other algorithms recently proposed for similar problems.

Later on, a series of extensions and improvements of Learn++ have been introduced, including Learn++ for concept drift. Learn++ for data fusion and Learn++ for imbalanced data. The original Learn++ suffers from the “out-voting” problem, which means that when a new class is introduced, the decisions of latter classifiers that recognize it are out-voted by previous classifiers that do not recognize it, until a sufficient number of new classifiers are generated. Some modified versions, such as Learn++.MT using Dynamic Weighted Voting (DWV) and Learn++.NC using Dynamically Weighted Consult and Vote (DW-CAV), are proposed to address this issue. In order to avoid relying on the previous data, the class-specific Mahalanobis distance metric is introduced to compute the distance between the training data and the unknown instance for each classifier in order to calculate the dynamic weights.

Learn++ is an ensemble approach, inspired primarily by the AdaBoost algorithm. Similar to AdaBoost, Learn++ also creates an ensemble of (weak) classifiers, each trained on a subset of the current training dataset, and later combined through weighted majority voting. Training instances for each classifier are drawn from an iteratively updated distribution. The main difference is that the distribution update rule in AdaBoost is based on the performance of the previous hypothesis, which focuses the algorithm on difficult instances, whereas that of Learn++ is based on the performance of the entire ensemble, which focuses this algorithm on instances that carry novel information. This distinction gives Learn++ the ability to learn new data, even when previously unseen classes are introduced. As new data arrive, Learn++ generates additional classifiers, until the ensemble learns the novel information. Since no classifier is discarded, previously acquired knowledge is retained. Learn++ works rather well on a variety of real world problems, though there is much room for improvement. An issue of concern is the relatively large number of classifiers required for learning instances coming from a new class. This is because, when a new dataset introduces a previously unseen class, new classifiers are trained to learn the new class; however, the existing classifiers continue to misclassify instances from the new class. Therefore, the decisions of latter classifiers that recognize the new class are out-voted by the previous classifiers that do not recognize the new class, until a sufficient number of new classifiers are generated that recognize the new class. This leads to classifier proliferation.
III. LEARN++.MT

In ensemble approaches that use a voting mechanism for combining classifier outputs, each classifier votes on the class it predicts. The final classification is then determined as the class that receives the highest total vote from all classifiers.

Learn++ uses weighted majority voting, where each classifier receives a voting weight based on its training performance. This works well in practice for most applications. However, for incremental learning problems that involve introduction of new classes, the voting scheme proves to be unfair towards the newly introduced class: since none of the previously generated classifiers can pick the new class, a relatively large number of new classifiers that recognize the new class are needed, so that their total weight can out-vote the first batch of classifiers on instances of the new class. This in return populates the ensemble with an unnecessarily large number of classifiers. Learn++.MT is specifically designed to address the classifier proliferation issue. The novelty in Learn++.MT is the way by which the voting weights are determined. Learn++.MT also uses a set of voting weights based on the classifiers’ performances, however, these weights are then adjusted based on the classification of the specific instance at the time of testing, through dynamic weight voting (DWV). For any given test instance, Learn++.MT compares the class predictions of each classifier and cross-references them against the classes on which they were trained. If a subsequent ensemble overwhelmingly chooses a class it has seen before, then the voting weights of those classifiers not trained with that class are proportionally reduced.

As an example, assume that an ensemble has seen classes 1 and 2, and a second ensemble has seen classes 1, 2 and 3. For a given instance, if the second ensemble (trained on class 3) picks class 3, the classifiers in the first ensemble (which has not seen class 3) reduce their voting weights in proportion to the confidence of the second ensemble. In other words, when the algorithm detects that the new classifiers overwhelmingly choose a new class on which they were trained, the weights of the other classifiers which have not seen this new class are reduced. The Learn++.MT algorithm is given in Figures 1 and 2, and explained in detail below.

For each dataset (D_k) that becomes available to Learn++.MT, the inputs to the algorithm are (i) a sequence of m_k training data instances x_i and their correct labels y_i, (ii) a classification algorithm BaseClassifier, and (iii) an integer T_k specifying the maximum number of classifiers to be generated using that database. If the algorithm is seeing its first database (k=1), a data distribution (D_1) – from which training instances will be drawn – is initialized to be uniform, making the probability of any instance being selected equal. If k>1 then the distribution is updated from the previous step based on the performance of the existing ensemble on the new data. The algorithm then adds T_k classifiers to the ensemble starting at t = cT_k+1 where cT_k denotes the number of classifiers that currently exist in the ensemble. For each iteration t, the instance weights, w_i, from the previous iteration are first normalized (step 1) to create a weight distribution D_t. A hypothesis, h_t, is generated from a subset of D_t that is drawn from D_t (step 2). The error, e_t, of h_t is then calculated; if t > \frac{1}{2}, the algorithm deems the current classifier, h_t, to be too weak, discards it, and returns to step 2, otherwise, calculates the normalized error β_t (step 3). The class labels of the training instances used to generate this hypothesis are then stored as CTr_t (step 4). The dynamic weight voting (DWV) algorithm is called to obtain the composite hypothesis, H_t, of the ensemble (step 5). H_t represents the ensemble decision of the first t hypotheses generated thus far. The error of the composite hypothesis, E_t is then computed and normalized (step 6). The instance weights wt are finally updated according to the performance of H_t (step7) such that the weights of instances correctly classified by H_t are reduced (and those that are misclassified are effectively increased). This ensures that the ensemble focus on those regions of the feature space that are not yet learned, paving the way for incremental learning. The inputs to the dynamic weight voting algorithm are (i) the current training data (during training) or any test instance, (ii) classifiers h_t, (iii) β_t, normalized error for each h_t, and (iv) the vector CTr_t containing the classes on which h_t has been trained.

Classifier weights are first initialized (step 1), where each classifier receives a standard weight that is inversely proportional to its normalized error β_t so that those classifiers that performed well on their training data are given higher voting weights. A normalization factor Z_e is then created as the sum of the weights of all classifiers trained on instances from class c (step 2).

For each instance, a preliminary per-class confidence factor 0<P_e<1 is generated (step 3). P_e is the sum of weights of all the classifiers that choose class c divided by the sum of the weights of all classifiers trained with class c (which is Z_e). In effect, this can be considered as the ensemble assigned confidence of the instance for belonging to each of the c classes. Then, again for each class, the weights are adjusted for classifiers that have not been trained with that class, that is, the weights are lowered proportional to the ensemble’s preliminary confidence on that class (step 4). The final / composite hypothesis is then calculated as the maximum sum of the weights that chose a particular class (step 5). Learn++ draws its inspiration from AdaBoost, which in turn relies on an ensemble of classifiers trained using adaptive bootstrap techniques. It is iteratively updated by new sets of data, possibly containing new classes. A modification called Learn++.MT, proposed by Muhlbaier et al. (2004),
improves the performance when new classes are added.

<table>
<thead>
<tr>
<th>Figure 1. Learn++.MT Algorithm</th>
<th>Figure 2. Dynamic Weight Voting Algorithm for Learn++.MT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn++.MT meets the compact and scalable requirements, and was used to create our visual object model. A brief description of it follows.</td>
<td></td>
</tr>
</tbody>
</table>

### IV. LEARN++.MT ALGORITHM

For each new dataset $D_k$, the inputs to Learn++.MT are:

1. A sequence of training data instances $x_i$ and their correct labels $y_i$, (2) the classification algorithm BaseClassifier, and (3) $T_k$, the maximum number of classifiers. As in typical boosting learning algorithms, data is drawn according to a distribution $D_1$. For the first set, $D_1$ is initialized as a uniform distribution. From the second set onwards, this distribution is updated according to the performance of the ensemble on the new data. $T_k$ classifiers are added to the ensemble when a new dataset is added. For each new classifier, a subset of $D_k$ is drawn, according to $D_k$, and evaluated against the new classifier to obtain the hypothesis $h_t$. The classifier error is estimated by $e_t = \frac{1}{n} \sum_{i=1}^{n} D_k(i)$ and if $e_t > \frac{1}{2}$, a new subset is drawn, discarding the classifier. A dynamic weight voting (DWV) algorithm is then called to obtain a composite hypothesis $H_t$. It represents the ensemble decision of all classifiers trained until now. The distribution $D_k$ is updated according to the performance of $H_t$. This process is repeated until all new $T_k$ classifiers have been trained. The essential difference from Learn++.MT to its parent method is the voting scheme. As in Learn++, voting is based on the weights assigned to each classifier but with DWV these weights are modified according to the classification of the specific testing instance. This is achieved by adjusting weights of classifiers that have not been trained with a given class. The adjustment is proportional to the ensemble’s confidence on that class. The goal of object classification is to attribute a known label to each input object frame comprising that object’s track. This can be seen as a multiclass classification problem, with a variable number of classes. As described previously, we use the Learn++.MT algorithm to handle class variability, and opted to use Support Vector Machines (SVMs) (Burges, 1998) as its BaseClassifier. SVMs are commonly used in machine learning problems, especially with large dimensional input spaces - which is the case for the object classification problem. Standard SVMs rely on margin optimization to learn a decision function $h(x)$, such that, if $x$ belongs to the target class, $h(x)$ is large and positive; otherwise, $h(x)$ is negative. SVMs are thus binary classifiers but can be adapted to multiclass problems; we opted for the one-against-one approach, where $n(n-1)/2$ models are constructed for a $n$-class problem.

Additionally, we used linear kernel SVMs, mainly for processing speed purposes. Specific kernels could alternatively be used, such as the pyramid match kernel proposed by Grauman and Darrell (2005) for multiresolution histograms.
V. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated the question of how to use ensemble algorithms to solve incremental learning problems where the training data become available in batches. The motivation was to show that ensemble algorithms, when applied smartly, are able to learn additional information from new data and preserve previously acquired knowledge.

In this paper we proposed Learn++.MT, a modified version of our previously introduced incremental learning algorithm, Learn++. The novelty of the new algorithm is its use of preliminary confidence factors in assigning voting weights, based on a cross-reference of the classes that have been seen by each classifier during training. Specifically, if a majority of the classifiers that have seen a class votes on that class, the voting weights of those classifiers who have not seen that class are reduced in proportion to the preliminary confidence. This allows the algorithm to dynamically adjust the voting weights for each test instance. The approach overcomes the outvoting problem inherent in the original version of Learn++ and prevents proliferation of unnecessary classifiers. The new algorithm also provided substantial improvements on the generalization performance on all datasets we have tried so far. We note that these improvements are more significant in those cases where one or several new classes are introduced with subsequent datasets.

It is also worth noting that, Learn++.MT is more robust than its predecessor. One of the reasons why Learn++ is having difficulty in learning a new class when first presented is due to difficulty in choosing the strength of the base classifiers. If we choose too weak classifiers, the algorithm is unable to learn. If we choose too strong classifiers, the training data are learned very well, resulting in very low β values which then causes very high voting weights, and hence even a more difficult outvoting problem. This explains why Learn++ requires larger number of classifiers or repeated training to learn the new classes. Learn++.MT, by significantly reducing the effect of the out-voting problem, improves the robustness of the algorithm, as the new algorithm is substantially more resistant to more drastic variations in the classifier architecture and parameters (error goal, number of hidden layer nodes, etc.).

REFERENCES


Author’s Profile

Mrs G.T.Prasanna Kumari is an Associate Professor in the Department of Computer Science and Engineering, Gokula Krishna College of Engineering, Sullurpet. She is pursuing Ph.D., in Computer Science in the area of Ensemble-Based Classifiers. She is in teaching since 1999. She presented papers at National and International Conferences and published articles in national & international journals.