Incremental Learning of New Classes in Unbalanced Datasets: Learn^{++}.UDNC

Gregory Ditzler, Michael D. Muhlbaier, and Robi Polikar[*]

Signal Processing and Pattern Recognition Laboratory
Electrical and Computer Engineering, Rowan University, Glassboro, NJ 08028 USA
ditzle53@students.rowan.edu, rpolikar@rowan.edu

Abstract. We have previously described an incremental learning algorithm, Learn^{++}.NC, for learning from new datasets that may include new concept classes without accessing previously seen data. We now propose an extension, Learn^{++}.UDNC, that allows the algorithm to incrementally learn new concept classes from unbalanced datasets. We describe the algorithm in detail, and provide some experimental results on two separate representative scenarios (on synthetic as well as real world data) along with comparisons to other approaches for incremental and/or unbalanced dataset approaches.

Keywords: Incremental Learning, Ensembles of Classifiers, Learn^{++}, Unbalanced Data.

1 Introduction

Incremental learning requires an algorithm that is capable of learning from new data that may introduce new concept classes, while retaining the previously acquired knowledge without requiring access to old datasets. The ability to learn new information and retaining existing knowledge are often conflicting in nature, which is commonly known as the stability-plasticity dilemma [1]. Ensemble based systems typically provide a good balance between the two by simultaneously increasing the memory (to aid stability) and learning capacity (to aid plasticity) of the learning algorithm. An ensemble based algorithm can add new classifiers to learn the new data and keep the previous classifiers to retain the existing knowledge. However, such an approach has its own shortcomings: there will always be classifiers trained on a subset of the concept classes, which are hence guaranteed to misclassify instances from classes on which they were not trained, potentially out-voting classifiers that were trained on such classes. While there are several ensemble-based incremental learning algorithms, such as those proposed in [2], [3], [4], this issue of out-voting is only explicitly addressed in Learn^{++}.NC (*N*ew *C*lass) [2], through a dynamic consult and vote approach. This approach allows each classifier to predict – based on the votes of others – whether it has been trained on a specific concept class, and withhold its vote on instances of classes that it predicts that it has not seen [2]. Learn^{++}.NC have previously been shown to provide favorable accuracy and parsimony properties compared to its predecessor

[*] Corresponding Author.

N. El Gayar, J. Kittler, and F. Roli (Eds.): MCS 2010, LNCS 5997, pp. 33–42, 2010.

Learn^{++}, as well as other ensemble based algorithms that are capable of incremental learning, such as dynamic weighted majority, bagging, AdaBoost and arc-x4. Learn^{++}.NC, however, was not designed to address –and hence could not handle – class imbalance, nor can it address the rare but pathological scenario of adding new classes while simultaneously removing existing ones on a subsequent dataset. Learning from unbalanced data was previously attempted in an algorithm called Learn^{++}.UD [5], however, that algorithm was not able to learn new classes. In this paper, we propose a new algorithm that is capable of learning new classes, even in the presence of relatively unbalanced data (including datasets with multiple minority classes).

Unbalanced data is a very real problem that draws growing interest [6] due to its prominence in several applications, such as fraud in financial transactions, spam detection, and weather prediction. Class imbalance occurs when a dataset does not have an equal number of exemplars from each class, which may be quite severe in some applications [7]. Common approaches to address class imbalance typically include oversampling or undersampling. The former involves increasing the number of minority class instances by creating copies of existing ones, which maybe prone to overfitting; whereas the latter involves removing majority class samples at random, which may cause loss of important information about the majority class. Perhaps one of the most unique, popular and unarguably successful approaches for class imbalance is the SMOTE algorithm [8], which modifies the feature space by creating a set of synthetic samples that lie on the line segment connecting two existing minority examples. The SMOTE algorithm was combined with the AdaBoost.M2 algorithm in [9] which allows for an ensemble approach to be combined with SMOTE.

The new member of the Learn^{++} family of algorithms described in this paper is the Learn^{++}.UDNC as it borrows techniques from both Learn^{++}.UD and Learn^{++}.NC. This version of Learn^{++} combines preliminary confidence measures in Learn^{++}.NC, with a transfer function that adjusts the voting weights of classifiers based on the number of instances seen from each class, as well as the class imbalance in each dataset. This approach works well in situations where an incremental learning is required to classify data coming from moderately imbalanced data distributions and new classes are being added and / or removed incrementally. It also works well in situations where classes are being added and removed at the same time from a database.

2 Learn^{++}.UDNC

The Learn^{++}.UDNC algorithm, whose pseudocode is shown in Fig. 1, receives subsequent dataset $\mathfrak{D}^k$ with m^k training examples, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{m^k}\}$, and class labels $Y = \{y_1, y_2, \dots, y_{m^k}\}$, a BaseClassifier to be used in ensemble generation, and T_k the number of classifiers to be generated from the kth dataset $\mathfrak{D}^k$. The algorithm works incrementally, hence never uses instances from previously seen datasets. Similar to that of AdaBoost [10], a distribution D is maintained according to which instances are drawn to train each classifier. Unlike AdaBoost, however, this distribution is updated based on the performance of the current ensemble, and not that of the previous classifier. The ensemble decision itself, on the other hand, uses a preliminary confidence measure and adjusts this confidence measures by the cardinality of each class in the training set.

```
Algorithm: Learn$^{++}$.UDNC
Input: Training dataset $\mathfrak{D}^k$, $k = 1,2,\dots,K$
 Sequence of input patterns $\{\mathbf{x}_i \in \mathbf{X}; y_i \in Y\}$ for $i = 1,2,\dots,m^t$
 Supervised learning algorithm: BaseClassifier
 $N_{k,c}$, the number of class-c instances in $\mathfrak{D}^k$
 Integer $T_k$, specifying the number of BaseClassifiers to create at $\mathfrak{D}^k$

Do for $k = 1,2,3 \dots, K$
  Initialize $D_1(i) = 1/m^1\ \forall i,\ eT_k = \sum_{j=1}^{k-1} T_j$
  If $k>1$,
      •  Update $D_t$ and number of classifiers in the ensemble
      •  Update $w_{t,c} = w_{t,c}\frac{\sum_{j=1}^{k-1} N_{j,c}}{\sum_{h=1}^{k} N_{h,c}}$,  $t = 1,2,\dots,eT_k$, $c = 1,2,\dots,C$
      •  Start the sub-ensemble generation from Step 5 of the following loop.
  Do for $t = 1, 2, \dots$
      1. Set $D_t(i) = D_t(i)/\sum_{j=1}^{m_k} D_t(j)$
      2. Call BaseClassifier, providing $\mathfrak{D}_t^k \in \mathfrak{D}^k$ drawn from $D_t$
      3. Obtain a hypothesis $h_t$: $\mathbf{X} \to Y$ and calculate its error
             $\epsilon_t = \sum_{j=1}^{m_k} D_t(j)[\![h_t(\mathbf{x}_j) \neq y_j]\!]$
         If $\epsilon_t > 1/2$, discard $h_t$ and go to step 2. Otherwise compute
         Normalized performance $p_t = 1 - 2\epsilon_t, 0 \le p_t \le 1$
      4. Compute class specific weights
             $w_{t,c} = p_t \frac{n_c}{\sum_{j=1}^{k} N_{j,c}}$
         where $n_c$ is the number of class-c instances in $\mathfrak{D}_t^k$
      5. Call EnsembleDecision to obtain sub-ensemble composite hypothesis $H_t$
      6. Compute Error on composite hypothesis $E_t = \sum_{j=1}^{m_k} D_t(j)[\![H_t(\mathbf{x}_j) \neq y_j]\!]$
      7. Set $\beta_t = E_t/(1 - E_t)$ and update the instance weights
             $D_{t+1} = D_t \times \begin{cases} \beta_t, & if\ H_t(\mathbf{x}_j) \neq y_j \\ 1, & otherwise \end{cases}$
  endfor
endfor
Call EnsembleDecision to obtain final hypothesis $H_{final}$
```

Fig. 1. Learn^{++}.UDNC Algorithm

The algorithm uses two loops for training, one indexed on k for subsequent datasets, and the other on *t* for individual classifiers to be generated for each training dataset. In step 1 of the inner loop (where we drop the superscript k, when the meaning is unambiguous, to avoid notational clutter), D_t is normalized to ensure a proper distribution, from which a training data subset $\mathfrak{D}_t^k$ is drawn is step 2. A hypothesis h_t is obtained from BaseClassifier in step 3, whose error is computed with respect to the current data distribution D_t. h_t is discarded if this error is greater than ½ and a new subset is drawn from D_t to create a new classifier.

Each classifier receives class specific weights, $w_{t,c}$, computed in step 4, based on the performance of h_t, the number of instances h_t is trained on for each class and the number of instances observed from a specific class. This method of computing the classifier weights limits the values of the classifier weights between 0 and 1, if sampling without replacement from D_t is used. Other weighting methods similar to those used in the original Learn^{++} (including that of AdaBoost) follow a $\log(1/\beta_t)$ form, which are bound between 0 and infinity. While optimal for learning from a single dataset, $\log(1/\beta_t)$ assigns weights to classifiers based on their performance on the entire dataset, and not on the ability of a classifier to predict a specific class. The proposed weighting approach, described in detail below, gives more weight to classifiers that have more experience with a specific class. This voting scheme also reduces the outvoting problem that occurs when classifiers not trained on a specific class vote – inevitably incorrectly – for instances that come from that class.

The *EnsembleDecision* function is called in step 5 to obtain the composite hypothesis H_t of the current subensemble. The error of the subensemble on $\mathfrak{D}^k$ is computed in step 6 and is used to update the instance weights, D_{t+1} (step 7), emphasizing the instances misclassified by the subensemble. This process is repeated until the subensemble has T_k classifiers, bring the total number of classifiers to eT_k. At any time, the current overall ensemble decision (final hypothesis on all datasets) can also be obtained by calling the *EnsembleDecision*.

When called within the inner loop, *EnsembleDecision* combines all classifiers generated for a subensemble and evaluates the subensemble performance using the dynamically weighted voting mechanism in conjunction with a reweighting transfer function that takes class imbalance into consideration. When called for the final hypothesis, *EnsembleDecision* combines all classifiers generated thus far.

```
Algorithm: EnsembleDecision
Input: Set of instances to be classified $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$
   Hypotheses $h_t$ trained for $\mathfrak{D}^k$ and hypothesis weight matrix $w_{t,c}$
   $N_{k,c}$, the number of class-c instances in $\mathfrak{D}^k$
   Integer $T_k$, specifying the number of BaseClassifiers to create at $\mathfrak{D}^k$
Initialization: Calculate $eT_k = \sum_{j=1}^{k-1} T_j$
   Compute sum of class specific weights for kth subensemble
      $Z_{k,c} = \sum_{j=eT_k}^{eT_k+T_k} w_{j,c}$
Do for $i = 1, 2, \dots, n$
   1. Obtain preliminary confidences
         $P_{k,c} = \sum_{t:h_t(\mathbf{x}_i)=c} w_{t,c}/Z_{k,c}$
      $t = eT_k, \dots, eT_k + T_k,\ k = 1,2,\dots,K,\ c = 1,2,\dots,C$
   2. Apply transfer function
         $\bar{P}_{k,c} = P_{k,c}^{\lambda_{k,c}},\ \lambda_{k,c} = \frac{N_{k,c}}{\min_c N_{k,c}}$
   3. Update sub-ensemble confidence
         $\hat{P}_{k,c} = \bar{P}_{k,c} \frac{N_{k,c}}{\sum_{j=1}^{K} N_{j,c}}$
   4. Compute ensemble decision
         $H_{final} = \arg\max_{c} \sum_{j=1}^{K} \hat{P}_{j,c}$
 endfor
```

Fig. 2. Ensemble Decision

The pseudocode for *EnsembleDecision* is shown Fig. 2. The inputs are a set of instances $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ to be classified, the hypotheses in the current subensemble (h_t) trained on $\mathfrak{D}^k$, the hypothesis weight matrix ($w_{t,c}$), and the number of instances, $N_{k,c}$, from each class in $\mathfrak{D}^k$. Initialization involves computing a normalization constant $Z_{k,c}$ as the total sum of class-specific weights for each class c seen by classifiers trained on $\mathfrak{D}^k$. A preliminary confidence $P_{k,c}$ is computed in step 1 for each class on which the classifiers in the kth subensemble were trained. $P_{k,c}$ represents the confidence of the kth subensemble in classifying class-c instances. This confidence is bound to be biased towards those classes with many instances. A transfer function is then applied in step 2 to reduce this bias on classes where training data is abundantly available. Step 3 updates the subensemble confidences $\bar{P}_{k,c}$ according to the cardinality of each class on which the subensemble was trained, relative to cardinality of the classes that all subensembles have seen. Finally, the ensemble confidence on a specific class is the sum of the subensembles confidences for that class, which form $\hat{P}_{k,c}$. The ensemble decision for $\mathbf{x}_i$ becomes the class corresponding to the largest sum of subensemble confidences.

3 Experimental Results

The Learn^{++}.UDNC algorithm was tested on two different datasets, each with several scenarios. The first is a synthetic Gaussian data, which allows us to compute Bayes performances, and the second one uses a real-world 10-class dataset available from [11]. The class specific performances (recall) as well as overall performances of Learn^{++}.UDNC are compared to those of fuzzy ARTMAP, an incremental learning algorithm capable of learning new classes, and SMOTE, which can learn from unbalanced data.

3.1 Gaussian Data

The first set of experiments involves 2D Gaussian data with four classes. The means for the data sets were $\mu_1 = (0{,}0)^t$, $\mu_2 = (1{,}1)^t$, $\mu_3 = (-1{,}0)^t$ and $\mu_4 = (1,-1)$, where the subscript refers to the class label. All covariance matrices assume that the features are uncorrelated with variances $\sigma_2 = \sigma_3 = \sigma_4 = 0.35$ and $\sigma_1 = 0.15$. Fig. 3 shows the decision boundary of the Bayes classifier on this database. The BaseClassifier used in the Learn^{++}.UDNC was an MLP with 20 hidden nodes on a single layer trained with an error goal of 0.05 and logistic sigmoid activation functions. Fifteen classifiers were created for each database that was introduced to the algorithm. Table 1 contains the class distributions of two Gaussian experiments, indicating the introduction of new classes in subsequent datasets, as well as the class imbalance at a ratio of 1 to 50. Also, note that the TEST data includes instances from all classes. In the first experiment, a new class, ω_3, becomes a second minority class, whereas the second experiment completely removes a class from the current training set. The SMOTE algorithm was applied to both ω_1 and ω_3 for $\mathfrak{D}^4$ in experiment 1 since ω_3 became a minority class along with ω_1.

Table 1. Data Distribution for Experiments with Synthetic Gaussian Data

	Experiment 1				Experiment 2			
Class →	ω_1	ω_2	ω_3	ω_4	ω_1	ω_2	ω_3	ω_4
$\mathfrak{D}^1$	10	0	0	500	10	0	0	500
$\mathfrak{D}^2$	10	500	0	500	10	500	0	500
$\mathfrak{D}^3$	10	500	500	500	10	500	500	500
$\mathfrak{D}^4$	10	500	10	500	10	500	0	500
Test	200	200	200	200	200	200	200	200

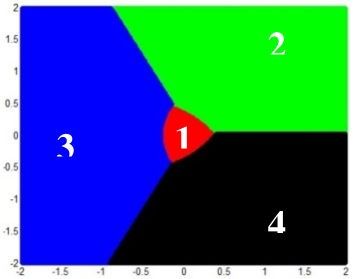

Fig. 3. Bayes Decision Boundary

The results from these two experiments are shown in Table 2 and Table 3. We observe that the Learn^{++}.UDNC outperforms the ARTMAP on all of the minority class examples (recall), as well as overall performance in both experiments. Also, Learn^{++}.UDNC performance on recall is comparable to that of SMOTE (with no significant difference between the two), but since Learn^{++}.UDNC is an incremental learning algorithm it can recall ω_3 better on $\mathfrak{D}^4$ when ω_3 is drastically reduced from the training set. On the other hand, SMOTE does a better job in recalling ω_1 for which the imbalance becomes progressively more severe (1 to 50 in $\mathfrak{D}^1$ and 1 to 100 in $\mathfrak{D}^4$). Even though ω3 is minority in $\mathfrak{D}$4, Learn^{++}.UDNC is able to recall its ω3 knowledge from $\mathfrak{D}^3$, whereas SMOTE does not have this opportunity. Hence Learn^{++}.UDNC will have seen more of the data space to accurately predict on ω3 while it may take additional data to learn the minority class (ω_1). SMOTE, on the other hand, can handle the more severe imbalance in any single setting.

It is perhaps a bit unfair to compare an ensemble based incremental learning algorithm to a single classifier based SMOTE, not designed for incremental learning. Hence we do not comment on the substantially better overall performance of Learn^{++}.UDNC over SMOTE, which is expected and not surprising; but rather we only comment on recall performance on the minority class, where the performances are more comparable, particularly for moderately unbalanced datasets. Furthermore, we merely point out that Learn^{++}.UDNC is capable of addressing unbalanced data (at least at a ratio of 1 to 50), while adding the capability of incremental learning of new classes, which themselves may be in minority.

Table 2. Gassian Expierment 1 Results on TEST dataset

		ω_1	ω_2	ω_3	ω_4	Overall
ARTMAP	$\mathfrak{D}^1$	41.60±16.53	0	0	99.55±0.20	35.29±4.11%
	$\mathfrak{D}^2$	41.75±12.86	87.60±6.21	0	96.00±2.05	56.45±3.50%
	$\mathfrak{D}^3$	14.85±6.12	90.70±2.07	92.50±1.22	96.30±1.11	73.59±1.53%
	$\mathfrak{D}^4$	18.95±7.04	91.35±0.83	85.90±2.74	95.65±1.32	72.96±1.62%
SMOTE	$\mathfrak{D}^1$	92.10±3.00	0	0	97.90±1.20	47.50±0.71%
	$\mathfrak{D}^2$	78.95±5.68	94.95±0.78	0	96.45±1.09	67.59±1.39%
	$\mathfrak{D}^3$	60.00±5.62	94.44±1.09	91.70±2.71	95.90±1.59	85.50±1.26%
	$\mathfrak{D}^4$	69.35±6.43	94.45±0.76	78.25±7.28	96.15±1.55	84.55±1.87%
Learn^{++}. UDNC	$\mathfrak{D}^1$	91.55±2.25	0	0	98.40±0.55	47.49±0.50%
	$\mathfrak{D}^2$	78.65±3.53	91.55±1.48	0	98.90±0.33	67.28±0.96%
	$\mathfrak{D}^3$	59.15±4.33	93.70±0.93	90.30±1.63	98.60±0.37	85.44±1.00%
	$\mathfrak{D}^4$	56.75±4.56	94.75±0.57	90.35±1.61	98.75±0.35	85.15±1.07%
Bayes						93.1%

Table 3. Gassian Expierment 2 Results on TEST dataset

		ω_1	ω_2	ω_3	ω_4	Overall
ARTMAP	$\mathfrak{D}^1$	39.90±13.65	0	0	99.80±0.25	34.93±3.38%
	$\mathfrak{D}^2$	33.95±9.78	97.55±6.27	0	96.60±1.08	52.52±2.73%
	$\mathfrak{D}^3$	16.05±2.41	86.10±3.29	93.85±1.55	94.70±1.15	72.68±0.93%
	$\mathfrak{D}^4$	19.05±4.69	86.40±3.90	91.05±2.29	94.20±1.12	72.68±1.92%
SMOTE	$\mathfrak{D}^1$	92.30±3.11	0	0	97.10±0.84	47.35±0.60%
	$\mathfrak{D}^2$	78.75±4.81	96.10±0.98	0	95.35±0.79	67.55±1.23%
	$\mathfrak{D}^3$	67.50±5.15	95.60±0.73	90.65±1.83	94.30±0.42	87.01±0.99%
	$\mathfrak{D}^4$	79.15±5.17	95.85±1.39	0	94.05±2.95	67.26±1.37%
Learn^{++}. UDNC	$\mathfrak{D}^1$	91.45±2.90	0	0	95.35±1.12	46.70±0.49%
	$\mathfrak{D}^2$	79.05±3.85	89.70±1.65	0	96.20±0.76	66.24±0.87%
	$\mathfrak{D}^3$	52.80±4.14	92.10±1.14	91.40±1.13	95.65±0.51	82.99±1.08%
	$\mathfrak{D}^4$	50.30±3.86	93.25±1.06	91.15±1.14	95.75±0.51	82.61±1.01%
Bayes						93.1%

3.2 OCR Data

The Optical Character Recognition (OCR) database consists of numerical characters 0~9 in a 8x8 matrix with two features removed due to zero variance of these feature. The training and testing class distributions are shown in Table 4, which has multiple new classes being introduced and removed at the same time in subsequent datasets. In addition to four datasets being introduced incrementally, Table 4 also shows a dataset $\mathfrak{D}^*$ which replaced $\mathfrak{D}^4$ in the OCR experiment. This new dataset, $\mathfrak{D}^*$, makes two of the previously seen classes minority class but includes instances from all classes. The results are shown in Table 5, which compares Learn^{++}.UDNC to ARTMAP.

Table 4. Database Distribution with Removal of Classes

Class →	0	1	2	3	4	5	6	7	8	9
$\mathfrak{D}^1$	0	257	248	0	0	248	248	0	0	252
$\mathfrak{D}^2$	0	0	247	257	0	0	247	252	0	0
$\mathfrak{D}^3$	248	0	0	256	0	0	0	252	248	0
$\mathfrak{D}^4$	247	256	0	0	252	247	0	0	247	252
$\mathfrak{D}^*$	20	20	250	250	250	250	250	250	250	250
Test	50	58	66	62	59	55	62	63	54	58

An MLP was used as the BaseClassifier with a 62x20x10 architecture with sigmoidal activation functions and an error goal of 0.05. Five classifiers were created with each dataset. The single classifier with SMOTE was not used with the OCR data because we wanted to see the effect of the class imbalance within a challenging incremental learning setting, where SMOTE – not designed to handle incremental data – would naturally be unsuccessful, and hence result in an unfair comparison. This is because Learn^{++}.UDNC will always be able to predict on previously seen classes that are not present in the current dataset, whereas SMOTE cannot predict on instances from classes not seen on the current training data.

Table 5. Results on the OCR Database

		0	1	2	3	4	5	6	7	8	9	All
ARTMAP	$\mathfrak{D}^1$	0	85.7	90.2	0	0	85.7	98.9	0	0	87.2	45.0±0.7%
	$\mathfrak{D}^2$	0	85.6	90.0	77.2	0	73.7	99.1	93.2	0	63.8	58.1±1.7%
	$\mathfrak{D}^3$	98.8	60.0	79.3	76.1	0	63.4	91.1	87.1	70.2	52.9	67.9±1.9%
	$\mathfrak{D}^4$	96.1	83.7	72.0	62.8	80.2	77.1	79.3	82.3	74.3	76.4	78.5±1.5%
	$\mathfrak{D}^5$	97.2	80.9	88.7	91.1	94.6	87.7	94.9	94.6	87.5	86.6	90.3±0.9%
Learn^{++}.UDNC	$\mathfrak{D}^1$	0	93.5	98.0	0	0	97.6	99.3	0	0	95.2	48.6±0.2%
	$\mathfrak{D}^2$	0	84.7	97.6	92.1	0	85.7	99.8	96.3	0	48.5	60.3±1.4%
	$\mathfrak{D}^3$	99.8	70.4	96.9	92.6	0	88.1	97.5	97.1	80.7	63.1	78.7±1.1%
	$\mathfrak{D}^4$	99.5	88.4	96.9	91.9	78.7	89.4	98.9	98.4	85.7	68.2	89.6±0.5%
	$\mathfrak{D}^5$	99.8	96.1	88.0	93.9	76.9	88.9	99.8	99.8	87.5	72.4	90.3±0.6%

Table 5 shows the results for the Learn^{++}.UDNC and ARTMAP applied to the incremental learning problem described in Table 4. A large performance boost is observed for Learn^{++}.UDNC as $\mathfrak{D}^4$ is introduced, resulting in fuzzy ARTMAP being outperformed by a large margin. The Learn^{++}.UDNC maintains the best recall of the minority classes when $\mathfrak{D}^*$ is introduced, however the overall performances here are no longer significantly different. Of the two minority classes, while the recall of character 0 for both algorithms are very close, the recall for character 1 with the Learn++.UDNC is significantly better than that of ARTMAP.

An additional experiment was also created with the OCR data that has a single minority class (character 0), with 40 instances of this class presented with each dataset, otherwise using the same class introduction and removal shown in Table 4. Therefore, the minority class is always present in the data and the incremental learning algorithm needs to continue to learn new parts of the minority and majority classes. ($\mathfrak{D}^*$ is not present in this new experiment).

Table 6. OCR Database Results with Single Minority Class

		0	1	2	3	4	5	6	7	8	9	All
ARTMAP	$\mathfrak{D}^1$	68.6	85.9	88.3	0	0	92.3	96.9	0	0	83.9	51.8±0.7%
	$\mathfrak{D}^2$	82.9	83.8	88.0	78.4	0	67.3	96.2	96.4	0	54.6	64.5±1.6%
	$\mathfrak{D}^3$	87.7	58.4	77.8	78.1	0	63.7	89.6	93.4	72.9	44.8	66.2±3.0%
	$\mathfrak{D}^4$	85.1	81.7	70.2	66.1	86.8	79.5	81.1	88.6	73.5	71.4	78.3±1.9%
Learn^{++} .UDNC	$\mathfrak{D}^1$	97.9	97.2	94.4	0	0	90.7	97.1	0	0	95.17	57.1±0.5%
	$\mathfrak{D}^2$	99.6	88.8	94.0	93.9	0	71.6	97.8	97.9	0	57.9	69.8±1.4%
	$\mathfrak{D}^3$	99.6	70.0	89.1	95.6	0	71.3	92.0	97.9	86.1	62.7	75.5±1.5%
	$\mathfrak{D}^4$	99.4	83.2	89.3	94.9	88.9	81.5	97.3	98.2	89.6	69.8	88.9±0.9%

Table 6 shows that the Learn^{++}.UDNC is able to consistently outperform fuzzy ARTMAP, not only on the minority class (0), but also on classes 2,3,6,7,8, with generally higher (but not statistically significant) performance on others.

4 Conclusions and Future Work

We described an incremental learning algorithm, Learn^{++}.UDNC, that combines several novelties of different algorithms within the Learn^{++} family, including a class specific weighting method, normalized preliminary confidence measures and a new transfer function that is used to reduce the confidence bias of a sub-ensemble trained on a majority class. Preliminary results indicate that Learn^{++}.UDNC is able to consistently outperform fuzzy ARTMAP under a variety of incremental learning scenarios and with a wide margin on unbalanced data problems. This was observed with both synthetic and real-world incremental learning problems. While not quite as effective as SMOTE on severely unbalanced data, we have shown that the Learn^{++}.UDNC performs comparably to SMOTE on minority class recall on moderately unbalanced datasets, but with the added advantage of learning incrementally, without applying any oversampling (or undersampling). This algorithm has shown the ability to perform well on a broad spectrum of incremental learning problems where the previous members of the Learn^{++} algorithms are not able to be reliable predictors on all classes. Current and future work include evaluating the algorithm on more severe unbalanced data on synthetic and real-world datasets, as well as integrating SMOTE and Learn^{++}.

Acknowledgements

This material is based on work supported by the National Science Foundation, under Grant No: ECCS-0926159.

References

1. Grossberg, S.: Nonlinear neural networks: principles, mechanisms, and architectures. Neural Networks 1, 17–61 (1988)
2. Muhlbaier, M., Topalis, A., Polikar, R.: Learn++.NC: Combining Ensembles of Classifiers with Dynamically Weighted Consult-and-Vote for Efficient Incremental Learning of New Classes. IEEE Transactions on Neural Networks 20(1), 152–168 (2009)
3. Carpenter, G., Grossberg, S., Markuzon, N., Reynolds, J.H., Rosen, D.: Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. IEEE Transactions on Neural Networks 3, 698–713 (1992)
4. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: an ensemble method for drifting concepts. Journal of Machine Learning Research 8, 2755–2790 (2007)
5. Muhlbaier, M., Topalis, A., Polikar, R.: Incremental learning from unbalanced data. In: Proc. of Int. Joint Conference on Neural Networks (IJCNN 2004), Budapest, Hungary, July 2004, pp. 1057–1062 (2004)
6. Chawla, N., Japkowicz, N., Kotcz, A.: Editorial: special issue on learning from imbalanced data sets. ACM SIGKDD Explorations Newsletter 6(1), 1–6 (2004)
7. Kubat, M., Holte, R.C., Matwin, S.: Machine Learning for the Detection of Oil Spills in Satellite Radar Images. Machine Learning 30, 195–215 (1998)
8. Chawla, N., Bowyer, K., Hall, L., Kegelmeyer, W.: SMOTE: Synthetic Minority Oversampling Technique. Journal of Artificial Intelligence Research 16, 321–357 (2002)
9. Chawla, N., Lazarevic, A., Hall, L., Bowyer, K.: SMOTEBoost: Improving Prediction of the Minority Class in Boosting. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 107–119. Springer, Heidelberg (2003)
10. Freund, Y., Schapire, R.E.: Decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 55(1), 119–139 (1997)
11. Asuncion, A., Newman, D.J.: UCI Repository of Machine Learning (November 2009), http://www.ics.uci.edu/~mlearn/MLReporitory.html