Reductions Between Classification Tasks

Alina Beygelzimer[1], Varsha Dani[2], Tom Hayes[3], John Langford[3], and Bianca Zadrozny[1]

[1] IBM TJ Watson
[2] University of Chicago
[3] TTI-Chicago

Abstract. A reduction converts a solver for one task into a solver for another task. We introduce a notion of reduction and analyze reductions between different classification tasks.

1 Introduction

There are many natural supervised learning tasks; even classification comes in a variety of different forms (binary, multiclass, cost-sensitive, importance weighted, etc.) There are two approaches that one can take: The first is to analyze all tasks independently and devise algorithms for each. The other approach is to use reductions to convert new, unexplored tasks to tasks that have been thoroughly analyzed, automatically transferring existing theory and algorithms from well-understood domains into new domains. We investigate the latter approach for classification problems. Reductions can be a powerful tool with a number of desirable properties:

1. They show how similar different tasks are. If two tasks are reducible to one another, it is reasonable to say that they are similar, modulo the efficiency of the reduction. The more efficient the reduction, the stronger the statement.
2. If task A can be reduced to task B, then techniques for solving B can be used to solve A, transferring results from one learning task to another.
3. Reductions help to extend existing theories to new tasks.

In addition to the above theoretical motivations, there appears to be significant empirical evidence (see [16], [6], [4]) that this style of analysis often produces learning algorithms that perform well in practice.

We give the basic definitions of our theoretical model, and discuss several known reductions from this standpoint. We then present two new reductions along with experimental evidence suggesting that analysis in this model is predictive of performance on real-world problems.

2 Basic Definitions

Definition 1. *A* supervised learning task *is a tuple* (K, Y, ℓ)*, where* K *is some information available at training time,* Y *is the space of predictions, and* $\ell : K \times Y \to [0, \infty)$ *is a loss function.*

As illustrative examples, we now specify (K, Y, ℓ) for several classification tasks.

Classification Task	K	Y	ℓ
Binary	$\{0,1\}$	$\{0,1\}$	$I(k \neq y)$
Multiclass	$\{1,\ldots,r\}$	$\{1,\ldots,r\}$	$I(k \neq y)$
Importance Weighted	$\{1,\ldots,r\} \times [0,\infty)$	$\{1,\ldots,r\}$	$k_2 I(k_1 \neq y)$
Subset	$2^{\{1,\ldots,r\}}$	$\{1,\ldots,r\}$	$I(y \notin k)$
Cost-Sensitive	$[0,\infty)^r$	$\{1,\ldots,r\}$	k_y

Note that the space K of information provided at training time, need not equal the prediction space Y. We sometimes need this additional flexibility (also used in [10]) to provide some information to the learning algorithm about the loss of different choices. In importance weighted classification, for example, we want to specify that predicting some classes correctly is more important than predicting others, while in cost-sensitive classification, K is used to associate a cost to each prediction $y \in Y$. In subset classification, K is the subset of correct predictions.

A task is (implicitly) a set of learning problems, each of which requires more information to be fully specified.

Definition 2. *A* supervised learning problem *is a tuple (D, X, K, Y, ℓ), where (K, Y, ℓ) is a supervised learning task, X is an arbitrary feature space, and D is a distribution over $X \times K$.*

The goal in solving a supervised learning problem is to find a hypothesis $h : X \to Y$ minimizing the expected loss $E_{(x,k)\sim D}\ell(k, h(x))$. For any particular hypothesis h, the loss rate $h_D = E_{(x,k)\sim D}\ell(k, h(x))$ is a fundamental quantity of interest. Finding h is difficult because we do *not* assume that D is known at training time.

Definition 3. *A* supervised learning algorithm for task (K, Y, ℓ) *is a procedure mapping any finite set of examples $(X \times K)^m$ to a hypothesis $h : X \to Y$.* [4]

Standard supervised learning algorithms such as support vector machines, neural networks, decision trees, and logistic regression fit this definition.[5]

2.1 Reductions

A reduction solves problems in one learning task using blackbox access to a solver for another task. We are interested in reductions that perform well whenever the solver performs well, so a reduction must map problems in one task to problems in another. To define a reduction, we must define this mapping from a *measure* D over original examples in $X \times K$ to a *measure* D' over generated examples in $X' \times K'$.

The reduction gets as input a set of examples S drawn from D. This set S induces a measure D_S over $X' \times K'$ defined by:

$$D_S(x', k') = E_{S' \sim S} \frac{\#(x', k')}{|S'|},$$

[4] Note that we do not require the learned hypothesis to belong to some predetermined class of hypotheses H; this is unnecessary. Of course, such an H exists implicitly as the set of all possible outputs of A.

[5] Note, however, that these algorithms do not actually produce a classifier for *all* learning problems in their task since they do not work for all input spaces. We assume that our idealized supervised learning algorithm does.

where the expectation is over all possible sets of examples S' generated by the reduction[6] on input S, and $\#(x', k')$ is the number of times (x', k') appears in S'. The induced measure $D' = E_{S \sim D} D_S$ is then defined by taking an expectation over sets S generated by D.

This defines D' for a single invocation of the oracle. If the reduction makes several sequential calls, we need to define D' for each invocation. Suppose that the first (unresolved) invocation produces h. We replace this invocation with the oracle that always returns h, and use the above definition to find D' for the next invocation.

We can now state the formal definition.

Definition 4. *A* supervised learning reduction from task (K, Y, ℓ) to task (K', Y', ℓ') *is a learning algorithm A for (K, Y, ℓ) that has oracle access to a learning algorithm A' for (K', Y', ℓ'). The algorithm A must satisfy the following property for all X and for all distributions D over $X \times K$:*

Given a set of examples S drawn from D, let h be the hypothesis output by A on S, and let $h'_{D'}$ be the maximum loss rate over all hypotheses output by A' on inputs generated by the reduction (where D' is the corresponding induced measure on $X' \times K'$). Then

$$h_D \leq g(h'_{D'}),$$

for a continuous monotone nondecreasing function $g : [0, \infty) \to [0, \infty)$ with $g(0) = 0$. We call g the error limitation *function for the reduction.*

Note that none of the above definitions require that examples be drawn IID. Analysis in this model is therefore directly applicable to many real-world problems.

This definition is analogous to a "Turing reduction" in complexity theory. In both cases a reduction from problem A to problem B is a procedure that solves A using a solver for B as a black box. One critical difference is that in complexity theory reductions are typically viewed as tools for classifying relative hardness of problems. If A reduces to B, it is interpreted as "B is at least as hard as A". In machine learning, reductions are viewed as constructive tools for reusing algorithms for one problem to solve another (showing that "A is no harder than B"). Also, learning reductions defined here are *informational* rather than *computational*, i.e., the hardness of solving a learning problem is due to the lack of *information* (in particular D, which defines the optimal solution, is unknown) rather than the computational power available to the algorithm. Notice that this definition does not necessarily force the reduction to use A' in a nontrivial way. Some learning tasks may be (near) optimally solvable, so the reduction may never need to invoke the oracle.

2.2 Composition of Reductions

Since our motivation for studying reductions is to reduce the number of distinct problems which must be considered, it is natural to want to compose two or more reductions to obtain new ones.

[6] Note that the reduction can be randomized and the expectation occurs over that randomization.

Proposition 1. *(Reduction Composition) If R_{12} is a reduction from task $\mathcal{D}_1$ to $\mathcal{D}_2$ with error limitation g_{12} and R_{23} is a reduction from task $\mathcal{D}_2$ to $\mathcal{D}_3$ with error limitation g_{23}, then the composition $R_{12} \circ R_{23}$ is a reduction from $\mathcal{D}_1$ to $\mathcal{D}_3$ with error limitation $g_{12} \circ g_{23}$.*

Proof. Given oracle access to any learning algorithm A for $\mathcal{D}_3$, reduction R_{23} yields a learning algorithm for $\mathcal{D}_2$ such that for any X_2 and D_2, it produces a hypothesis with loss rate $h'_{D_2} \leq g_{23}(h''_{D_3})$, where h''_{D_3} is the maximum loss rate of a hypothesis h'' output by A (over all invocations). We also know that R_{12} yields a learning algorithm for $\mathcal{D}_1$ such that for all X_1 and D_1 we have:

$$h_{D_1} \leq g_{12}(h'_{D_2}) \leq g_{12}(g_{23}(h''_{D_3}),$$

where h'_{D_2} is the maximum loss rate of a hypothesis h' output by $R_{23}(A)$. Finally, notice that $g_{12} \circ g_{23}$ is continuous, monotone nondecreasing, and $g_{12}(g_{23}(0)) = 0$. ∎

2.3 Notions of Efficiency

If two tasks are each reducible to the other, it is reasonable to say that they are equivalent, *modulo the complexity of the reduction.* We consider several types of efficiency.

Error Efficiency of a reduction *for a given example* is defined as the maximum ratio of the loss of the hypothesis output by the reduction on this example and the total loss of the oracle hypotheses on the examples generated by the reduction.

Time Efficiency We want the transformation to be computationally efficient assuming that all oracle calls take unit time.

Call efficiency We quantify how extensively a reduction uses its oracle based on (1) the number of oracle calls it makes; (2) whether the calls are adaptive (i.e., depend on the answers to previous queries) or parallel (i.e., all queries can be asked before any of them is answered).

Note that parallel calls can be turned into a single call. Suppose that we make several parallel queries to the oracle. Let $\{S_i\}$ be the corresponding inputs (i.e., representing sample sets over the same feature space X), and let the corresponding hypotheses be $\{h_i\}$. We could instead create a new sample set $S = \{(\langle x, i\rangle, k) : \langle x, k\rangle \in S_i\}$, then call the oracle on S to get a hypothesis h, and define $h_i(x) = h(\langle x, i\rangle)$.

Independence Preservation Many common learning algorithms are built under the assumption that examples are drawn independently from some distribution D'. Reductions which take an independently drawn dataset, and construct dependent datasets for the oracle learning algorithm are often undesirable for this reason.

We would also like reductions to be *learning preserving*:

Definition 5. *A reduction R from (K, Y, ℓ) to (K', Y', ℓ') is* learning preserving *if there exists a reduction R' from (K', Y', ℓ') back to (K, Y, ℓ) such that for every distribution D' over $X' \times K'$ and every oracle A' for (K', Y', ℓ'), if A' has loss rate ϵ with respect to D', then $R'(R(A'))$ also has loss rate ϵ with respect to D'.*

Intuitively, a learning preserving reduction does not reduce easy problems to hard queries for the oracle, and it must use the oracle in a nontrivial manner. Appendix Theorem 5 shows that the tree reduction [7] is learning preserving.

Definition 4 and reduction composition do not generally imply that a reduction is learning preserving. For all distributions D' and all oracles A', the loss rate of the double reduction $R'(R(A'))$ on D' is bounded in terms of the loss rate of the oracle $R(A')$ on the *induced* (by R' on D') distribution D'', which is in turn bounded by the loss rate of the oracle A' on the *induced* (now by R on D'') distribution D'''. A reduction is *learning preserving* if the loss rate of the double reduction $R'(R(A'))$ on D' is the same as the loss rate of A' on the *same* distribution D' (not the induced D'''). In general, good performance on D''' says nothing about performance on D'.

3 Prior Work

Many people have worked on reductions in learning. One of our goals is to provide a unifying framework, allowing the objective comparison of different reductions according to the properties outlined in section 2.3.

Boosting algorithms [6, 11] are reductions from binary (sometimes multiclass) classification with a small error rate to importance weighted classification with a loss rate of nearly $\frac{1}{2}$, a rather surprising capability.

Bagging [3] can be seen as a self-reduction of classification to classification which turns learning algorithms with high "variance" (i.e., dependence on the exact examples seen) into voting classifiers with lower "variance". Bagging performance sometimes suffers because the examples that are fed into the classifier learning algorithm are not necessarily independently distributed according to the original distribution.

Error Correcting Output Codes (ECOC) [4] generalize the tree reduction [7] and the one-against-all reduction [13] used for solving multiclass classification given a solver for binary classification. The all-pairs reduction [9] also reduces multiclass classification to binary classification.

The "Costing" algorithm [16] is a reduction of importance weighted classification to classification.

Pitt and Warmuth [12] also introduced a form of a reduction between classification problems called *prediction-preserving reducibility*. The reductions presented here differ in several respects: (1) they are between arbitrary learning *tasks* rather than between restricted sets of classification problems; (2) they are representation independent; (3) prediction-preserving reducibility is a *computational* notion, while the reductions here are *informational*; (4) prediction-preserving reductions make only one call to the oracle (i.e., problem they reduce to) and output the oracle's answer as its own answer. We enforce no restriction on how the oracle is used, making the setting directly applicable to ECOC, Boosting, and other common transformations[7]; (5) prediction-preserving reductions are typically used to show negative, *non*-predictability results, while we show positive results transferring learning algorithms between tasks.

[7] Abe and Watanabe [1] defined a Turing analogue of the prediction preserving reducibility, and showed that new reductions are feasible in this setting.

We include a table of results and refer to the Appendix for details.

Name	$g(\epsilon)$	Time	#Calls	Indep. Preserve
Costing	ϵ	$O(T\|S\|)$	T parallel	Yes
1-vs-all	$(r-1)\epsilon$	$O(\|S\|r)$	r parallel	Yes
Tree	$\epsilon \log_2 r$	$O(\|S\|r)$	$r-1$ parallel	Yes
ECOC	4ϵ	$O(\|S\|r)$	$2r$ parallel	Yes

4 Reducing To Importance Weighted Binary Classification

In this section we define two new reductions to importance-weighted binary classification. Both can be composed with the "Costing" reduction from importance weighted classification to classification to yield new (or alternative) reductions to binary classification. We first present a weighted one-against-all reduction from subset classification to importance weighted classification. Our second reduction is a weighted all-pairs reduction from cost-sensitive classification to importance weighted classification. In the next sections, we prove the following efficiency guarantees:

Name	$g(\epsilon)$	Time	#Calls	Indep. Preserve
weighted 1-vs-all	$c(r,n)\epsilon$	$O(\|S\|r)$	1	No
weighted all-pairs	2ϵ	$O(r^2)$	1	No

where n is the number of labels for each example in subset classification and $c(r,n) = \frac{r}{n+1}$ if $r \le n^2+n$, otherwise $c(r,n) = \frac{r+1}{n+1} - \frac{n}{r}$.

We also present some experimental results for these reductions.

4.1 From Subset Classification: Weighted One-against-all

We present a weighted one-against-all reduction from subset classification to importance weighted binary classification. It appears to be new.

Recall that a training example in subset classification is labeled by some subset Q of r possible labels. The reduction maps each example (x, Q), $|Q| = n \le r$, to r examples of the form:

$$(\langle x, y\rangle, I(y \in Q), i_{I(y\in Q)}) \text{ for } y \in \{1, \ldots, r\},$$

where $i_0 = \frac{r}{n+1}$, and $i_1 = \frac{r}{n+1}$ if $r \le n^2+n$, otherwise $i_1 = \frac{r-n}{n}$. The oracle uses these examples to construct a binary classifier h. To construct a subset classifier from h, we do the following: If there exists a label $y \in \{1, \ldots, r\}$ such that $h(\langle x, y\rangle) = 1$, then predict y, breaking ties randomly; predict randomly otherwise.

The following theorem gives an error limitation bound for this reduction.

Theorem 1. (Error Limitation of Weighted One-against-all) *Let $c(r,n) = \frac{r}{n+1}$ if $r \le n^2+n$, otherwise $c(r,n) = \frac{r+1}{n+1} - \frac{n}{r}$. For any input space X and distribution D over $X \times 2^{\{1,\ldots,r\}}$, if the binary classifier has a normalized loss rate ϵ (on the induced distribution), then the weighted one-against-all reduction has loss rate at most:*

$$\epsilon E_{(x,Q)\sim D} c(r,n).$$

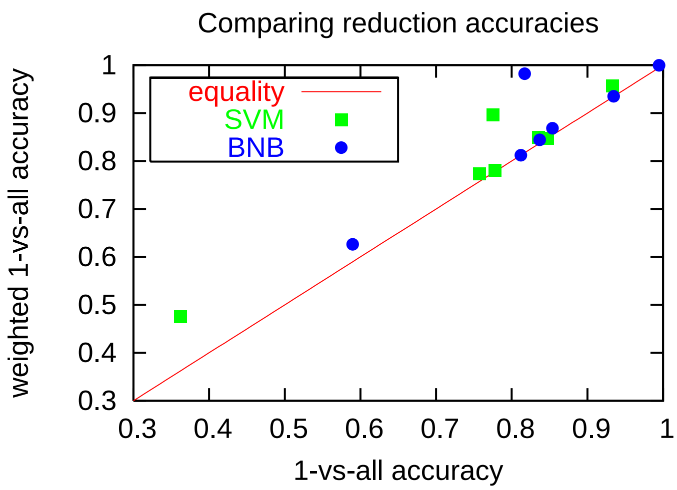

Fig. 4.1. A figure showing comparative performance of the weighted-one-against-all reduction (+ costing) and the one-against-all reduction on several multiclass datasets with Boosted Naive Bayes (BNB) and Support Vector Machine (SVM) classifiers. Note that for these experiments we used r classifiers rather than 1 as in the theorem.

The proof of this theorem is deferred to Appendix C. The theorem should be compared directly with the one-against-all reduction to binary classification which has error efficiency of $r-1$. Here, with $n = 1$, we get $c(r, n) = \frac{r+1}{2} - \frac{1}{r} = \frac{(r+2)(r-1)}{2r}$ implying error efficiency of about $\frac{r}{2}$, or about half the loss rate of the earlier reduction.

We tested the reduction on several UCI multiclass datasets using Boosted Naive Bayes and linear SVM as the oracle learning algorithms and obtained the average test set error rates shown in Figure 4.1. From these results, we see that there is a consistent benefit to using this reduction over the simple one-against-all reduction.

4.2 From Cost-Sensitive Classification: Weighted All-pairs

We reduce (multiclass) cost-sensitive classification to importance weighted (binary) classification using maps of the form:

$$(x, k_1, \ldots, k_r) \mapsto ((x, i, j), I(k_i < k_j), v_j - v_i),$$

for all pairs (i, j) with $i < j$. The values $v_1, \ldots, v_r$ are functions of $k_1, \ldots k_r$ defined below. The values are order-preserving (for all i, j, $k_i < k_j$ iff $v_i < v_j$).

Given m samples of training data for a cost-sensitive problem, we generate $m\binom{r}{2}$ training inputs for an importance weighted problem, by iterating through all $\binom{r}{2}$ possibilities for i and j. Now we use our oracle and this training data to generate a classifier, h. Given an input x for the cost-sensitive problem, we evaluate $h(x, i, j)$, for all the $\binom{r}{2}$ pairs $i, j \in [r]$, with $i < j$. Say label i "beats" label j for input x if either $i < j$ and $h(x, i, j) = 1$, or $i > j$ and $h(x, j, i) = 0$. Note that, if our classifier makes no errors and $k_i \neq k_j$, then label i beats label j exactly when $k_i < k_j$ (we are not concerned with the case $k_i = k_j$). Our algorithm outputs a label i which beats the maximum number of labels $j \neq i$. The mechanism for breaking ties does not affect our main result.

For $t \in [0, \infty)$, let $L(t) = \#\{j \mid k_j \leq t\}$. By shifting, we may assume that the minimum cost $k_{\min} = 0$, so that $t \geq 0$ implies $L(t) \geq 1$. Values v_i are defined by:

$$v_i = \int_0^{k_i} 1/L(t)\, dt.$$

This method of assigning importances is provably near-optimal for our algorithm.

Theorem 2. *(Error Limitation of Weighted All-pairs)*
upper bound: *If the importance weighted binary classifier has loss rate at most* ϵ, *then the weighted all-pairs reduction has expected cost at most* 2ϵ.
lower bound: *For any other assignments of importances* $w_{i,j}$ *to the sample points* (x, i, j) *in the above algorithm, there exists a distribution with expected cost* $\epsilon/2$.

The following observation is helpful in the proof:

Lemma 1. *Suppose label* i *is the winner. Then, for every* $j \in \{1, \ldots, i-1\}$, *there must be at least* $\lceil j/2 \rceil$ *pairs* (a, b), *where* $a \leq j < b$, *and* b *beats* a.

Proof (of Lemma 1). Let a be the winner of the tournament on $\{1, \ldots, j\}$.
Case 1: Suppose some a beats at least $\lceil j/2 \rceil$ of the others. If no label $b > j$ beat any label $c \leq j$, then a would beat at least $\lceil j/2 \rceil + 1$ more labels than any $b > j$. Thus, in order to have label $b > j$ beat as many labels as a, at least $\lceil j/2 \rceil$ edges of the form (a, c) or (d, b) must be reversed.
Case 2: There is no label $a \in \{1, \ldots, j\}$ beating $\lceil j/2 \rceil$ of the rest of $\{1, \ldots, j\}$. There must be a j-way tie with $(j-1)/2$ losses per label in $\{1, \ldots, j\}$. In this case, although every label beats $(j+1)/2$ more labels than any $b > j$, it is still necessary to reverse at least $(j+1)/2 \geq \lceil j/2 \rceil$ edges, in order to ensure that some $b > j$ beats as many labels as each of $\{1, \ldots, j\}$. ∎

Proof (of Theorem 2). For a particular input $(x, k_1, \ldots, k_r)$, suppose our algorithm chooses the wrong label. We show that this requires the classifier to incur a comparable loss. For notational simplicity, assume $k_1 \leq \cdots \leq k_r$ and that our algorithm chooses label $i > 1$.

Lemma 1 and the definition of v_i, imply the penalty incurred to make label i win is at least:

$$\int_0^{k_i} \frac{\lceil L(t)/2 \rceil}{L(t)} \mathrm{d}t \geq \int_0^{k_i} \frac{1}{2} \mathrm{d}t = \frac{k_i}{2}.$$

On the other hand, the total importance assigned to queries for this instance equals:

$$\sum_{i<j} v_j - v_i = \sum_{i<j} \int_{k_i}^{k_j} \frac{1}{L(t)} \mathrm{d}t = \int_0^{k_r} \frac{L(t)R(t)}{L(t)} \mathrm{d}t = \int_0^{k_r} R(t) \mathrm{d}t = \sum_{i=1}^{r} \int_0^{k_i} \mathrm{d}t = \sum_{i=1}^{r} k_i,$$

where $R(t) = r - L(t)$ is the number of labels whose value is greater than t and the second equality follows from switching the order of summation and counting the number of time a pair (i,j) satisfies $i < t < j$. The second-last equality follows by writing $R(t)$ as a sum of the r indicator functions for the events $\{k_j > t\}$, and then switching the order of summation.

Consequently, for every example $(x, k_1, \ldots, k_r)$, the total importance assigned to queries for x equals $\sum_i k_i$, and the cost incurred by our algorithm on instance x is at most twice the importance of errors made by the binary classifier on instance x. Averaging over the choice of x shows that the cost of the reduction is at most 2.

For the lower bound, consider examples $\left(x, 0, \frac{1}{r-1}, \ldots, \frac{1}{r-1}\right)$. Suppose we run our algorithm, using some $w_{i,j}$ as the importance for the query (x, i, j). Any classifier which errs on $(x, 1, i)$ and $(x, 1, j)$, where $i \neq j$, causes our algorithm to choose label 2 as the winner, thereby giving a cost of $1/(r-1)$, out of a total cost of 1. The importance of these two errors is $w_{1,i} + w_{1,j}$, out of a total importance of $\sum_{i,j} w_{i,j}$. Choosing i, j so that $w_{1,i} + w_{1,j}$ is minimal, the adversary's penalty is at most $2\sum_{i=2}^{r} w_{1,i}/(r-1)$, and hence less than $2/(r-1)$ times the total importance for x. This shows that the cost of the reduction cannot be reduced below 1/2 merely by improving the choice of weights. ∎

We did experiments comparing the weighted all-pairs reduction to the original all-pairs reduction [9] (which ignores the costs) and to a state-of-the-art multi-class cost-sensitive learning algorithm called MetaCost [5]. We used boosted Naive Bayes as the oracle classifier learner and applied the methods to five UCI multiclass datasets. Since these datasets do not have costs associated with them, we generated artificial costs in the same manner that was done in the MetaCost paper (except that we fixed the cost of classifying correctly at zero). We repeated the experiments for 20 different settings of the costs. The average and standard error of the test set costs obtained with each method are shown in the table below.

Dataset	All-Pairs	MetaCost	Weighted All-Pairs v.1	Weighted All-Pairs v.2
splice	59.8 ± 24	49.8 ± 3.05	197 ± 23	46.5 ± 3.5
solar	3989 ± 1415	5317 ± 390	24.7 ± 2.0	36.8 ± 11
anneal	310 ± 205	207 ± 43	29.9 ± 3.0	164 ± 43
kdd-99	234 ± 62	49.4 ± 9.3	0.956 ± 0.12	1.76 ± 0.66
satellite	137 ± 33	104 ± 6.4	428 ± 21	148 ± 8.5

We present results for two versions of this reduction. One is the exact version used in the proof above. In the other version, we learn one classifier per pair as is done in the original all-pairs reduction. More specifically, for each i and j (with $i < j$) we learn a classifier using the following mapping to generate training examples:

$$(x, k_1, \ldots, k_r) \mapsto (x, I(k_i < k_j), |v_i - v_j|).$$

In the results, we see that the first version appears to works well for some problems and badly for others, while the second version is more stable. This is probably caused by the fact that we make some problems more difficult for the classifier learner by bundling examples from different pairs together, while making other problems easier. Also, the training examples given to the learner in the first version are not drawn IID from a fixed distribution as learning algorithms often expect.

5 Concluding Remarks

Some of our experiments used a variant of the above reductions where we used a separate classifier for each pair (in the weighted all pairs) or label (in the weighted one-against-all). The theoretical impact of this design choice is that we gain independence preservation, but the error transformation $g(\epsilon)$ only holds when each classifier is trained on samples with the same expected importance.

The definition of a reduction we outline here covers a variety of existing supervised learning algorithms. The analysis of error limitation for these reductions suggests improved reductions, which appear to have superior empirical performance.

There are many open problems remaining. We do not yet have the most efficient reductions between the tasks analyzed in this paper. There are new natural tasks not discussed here which need to be formalized and connected via reductions.

Our definition of a reduction is limited to the supervised learning task. It nonetheless appears that a natural generalization capturing even broader tasks is possible.

References

1. Naoki Abe and Osamu Watanabe. Polynomially Sparse Variations and Reducibility among Prediction Problems, *IEICE Transactions on Communications, Electronics, Information, and Systems*, E75-D(4):449–458, 1992.
2. Erin Allwein, Robert Schapire, Yoram Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers, *Journal of Machine Learning Research*, 1:113–141, September 2001.
3. Leo Breiman. Bagging predictors, *Machine Learning*, 26(2):123–140, 1996.
4. Thomas G. Dietterich and Ghulum Bakiri. Solving Multiclass Learning Problems via Error-correcting Output Codes, *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.
5. Pedro Domingos. Metacost: A general method for make classifiers cost sensitive. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155-164. ACM Press, 1999.
6. Yoav Freund and Robert 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.
7. John Fox. *Applied Regression Analysis, Linear Models, and Related Methods*, Sage Publications, 1997.
8. Venkatesan Guruswami and Amit Sahai. Multiclass Learning, Boosting, and Error-correcting codes, *Proceedings of the 12th Annual Conference on Computational Learning Theory (COLT)*, 145–155, 1999.
9. Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems*, 507–513, MIT Press, 1998.
10. David Haussler. Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications, *Information and Computation*, 100:78–150, September 1992.
11. Adam Kalai and Rocco Servedio. Boosting in the Presence of Noise, *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing (STOC)*, 195–205, 2003.
12. Lenny Pitt and Manfred Warmuth. Prediction-Preserving Reducibility, *Journal of Computer and System Sciences*, 41:430–467, 1990.
13. Ryan Rifkin and Aldebaro Klautau. In Defense of One-Vs-All Classification, *Journal of Machine Learning Research*, 5:101–141, 2004.
14. Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
15. Robert E. Schapire. Using output codes to boost multiclass learning problems, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, 313–321, 1997.
16. Bianca Zadrozny, John Langford, and Naoki Abe. Cost Sensitive Learning by Cost-Proportionate Example Weighting, *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, 2003.

A Easy Directions

The learning tasks we consider have an "easy" direction for reduction related to whether a particular task is definable as a subset of another task. For example, solving binary classification with importance weighted binary classification is easy, because importance weighted binary classification with $k_2 = 1$ is identical to binary classification. Figure A.1 shows the easy direction for these reductions.

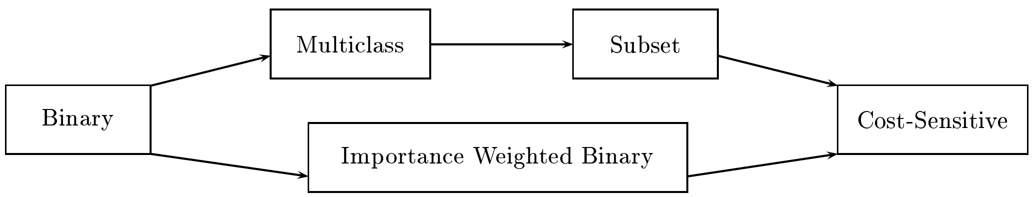

Fig. A.1. The "easy" direction for reductions between the learning problems discussed here.

B Reduction from Multiclass Classification to Binary Classification

In this section we analyze several well known reductions for efficiency. Before we start, it is important to note that there are (see [15],[14]) boosting algorithms for solving multiclass classification given weak binary importance weighted classification. Some of these methods require extra-powerful classification algorithms, but others are genuine reductions, as defined here. A reductionist approach for creating a multiclass boosting algorithm is to simply compose any one of the following reductions with *binary* boosting.

The one-against-all reduction In the one-against-all reduction, we learn r classifiers h_i, using the mapping $(x, y) \to (x, I(y = i))$. In order to construct a multiclass classifier from the binary classifiers, we use the following procedure: If there exists a label i such that $h_i(x) = 1$, then predict i, breaking ties randomly; predict randomly otherwise. Note that the trick of combining many calls into one described in Section 2.3 applies here.

Theorem 3. *(One-against-all error efficiency) If the binary classifiers have error rate ϵ, the one-against-all reduction has error rate at most*

$$(r - 1)\epsilon.$$

Proof. We analyze how false negatives (predicting 0 when the correct label is 1) and false positives (predicting 1 when the correct label is 0) produced by the binary classifiers lead to errors in the multiclass classifier. A false negative produces an error in the multiclass classifier a $\frac{r-1}{r}$ fraction of the time (assuming all the other classifiers are correctly outputting 0), because we are choosing randomly between l

labels and only one is correct. The other error modes to consider involve (possibly multiple) false positives. If there are n false positives, the error probability is either $\frac{n}{n+1}$ or 1 if there was also a false negative. The efficiency of these modes in creating errors (i.e., the ratio of the probability of a multiclass error to the number of binary errors) is $\frac{\frac{r-1}{r}}{1} = \frac{r-1}{r}$, $\frac{\frac{n}{n+1}}{n} = \frac{1}{n+1}$, and $\frac{1}{n+1}$, respectively. Taking the maximum, we get $\frac{r-1}{r}$. Multiplying by r (since we have r opportunities to err, one for each classifier), we get the bound. ∎

The one-against-all reduction above is not learning preserving, but two simple modifications would make it learning preserving without affecting the error limitation bound. First, we learn binary classifiers only for those labels that actually appear in the training set; i.e., if there was no example with label i, we do not learn h_i. Similarly for prediction; if no learned h_i predicts 1, we predict randomly among the labels seen during training. Second, if we train a pair of binary classifiers on sets of examples with complementary labels, we ignore one of the classifiers. A moment's thought shows that the error limitation bound still holds.

The modified reduction is learning preserving. For every $r > 1$, there exists a reduction R from binary classification to r-class classification such that for every input space X, distribution D over $X \times \{0, 1\}$, and every binary learning algorithm A, if A has loss rate ϵ on D, then R(one-against-all(A)) has loss rate ϵ on D.

The back reduction R takes a set of examples with binary labels and simply feeds it to the oracle for r-class classification. If the oracle's prediction is not in $\{0, 1\}$, predict randomly; otherwise answer with the oracle's prediction.

Let $S \sim D$ be a binary training set input to R. Under the reduction, R feeds S to the oracle one-against-all(A), which in turn feeds S to its oracle A, making the loss rates of A and R(one-against-all(A)) equal. On the other hand, an adversarial A can easily make the unmodified one-against-all reduction not learning preserving.

The Tree Reduction In the tree reduction (which is well known, see chapter 15 of [7]), we construct $r - 1$ binary classifiers, which distinguish between the labels using a binary tree. The tree has depth $\log_2 r$, and each internal node is associated with a set of labels that can reach the node, assuming that each classifier above it predicts correctly. The set of labels at that node is split in half, and a classifier is trained to distinguish between the two sets of labels. In particular, we start with the set of all labels $\{1, \ldots, r\}$ at the top most node, and split the labels into ordered intervals $\{1, \ldots, r/2 - 1\}, \{r/2, r\}$ at each node. Predictions are made by following a chain of classifications down to the leaves, each of which is associated with a unique label.

Note that the trick of combining many calls into one described in Section 2.3 applies here as well. Also, instead of $r - 1$ parallel, one could use $\log_2 r$ queries.

Theorem 4. *(Tree error efficiency) If the binary classifiers have loss rate ϵ, the tree reduction has loss rate at most*

$$\epsilon \log_2 r.$$

Proof. A multiclass error occurs only if some binary classifier on the path from the root to the leaf errs. ∎

We show that the tree reduction is learning preserving.

Theorem 5. *The tree reduction TREE is learning preserving. For every $r > 1$, there exists a reduction R from binary classification to r-class classification such that for every binary learning problem L and every binary learning algorithm A, if A has error rate ϵ on L, then $R(TREE(A))$ has error ϵ on L.*

Proof. The converse reduction R takes a set of examples S with binary labels, and feeds the set $\{(x, (r-1)y+1) | (x, y) \in S\}$ to the oracle for r-class classification. The binary classifier h is constructed from the resulting r-class classifier h_r as $h(x) = I(h_r(x) > r/2 - 1)$.

Let S_r be the set of examples induced by R on the inputs to the oracle $TREE(A)$. Under the tree reduction, these examples are converted according to $\{(x, I(y_r \geq r/2-1) : (x, y_r) \in S_r\}$. Consequently, we have $S = S_r$ and so the loss rate of A on S is exactly the loss rate of the root classifier in $TREE(A)$. Since the loss rates of all other classifiers are irrelevant under the mapping $h(x) = I(h_r(x) > r/2 - 1)$, we know that the loss rate of the classifier given by $R(TREE(A))$ equals the loss rate of A. ∎

The error correcting output code (ECOC) reduction. Let $C \subseteq \{0, 1\}^n$ be an error-correcting code with l codewords a minimum distance of d apart. Let $C_{i,j}$ denote the i'th bit of the j'th codeword of C. The ECOC reduction corresponding to C learns n classifiers: the i'th classifier predicts $C_{i,y}$ where y is the correct label given input x. The loss rate for this reduction is at most $2n\epsilon/d$, as we shall prove in Theorem 6 below.

We mention three codes of particular interest. The first is when the codewords are a subset of the rows of a Hadamard matrix (an $n \times n$ $\{0, 1\}$-matrix for which any two rows differ in exactly $n/2$ places). Such matrices exist and are easy to construct when n is a power of 2. (The famous Hadamard conjecture states that they exist whenever n is a multiple of 4.) Thus, for Hadamard codes, the number of classifiers needed is less than $2r$, and the loss rate is at most 4ϵ.

A second code of interest is when the codewords are the binary representations of $0, 1, \ldots, r-1$. In this case, the ECOC reduction is the same as the tree reduction above.

A third code of interest is the $r \times r$ identity matrix, for which the ECOC reduction is the same as the one-against-all reduction.

Also note that the "many calls to one call" trick in section 2.3 (call efficiency) can be applied here.

Theorem 6. *(ECOC loss efficiency) If the binary classifiers have loss rate ϵ, the ECOC reduction has loss rate less than:*

$$\frac{2n}{d}\epsilon.$$

This theorem is identical to Lemma 2 of [8] (which was generalized in [2]), except for one small difference. Here, we are concerned with the loss rate on the distribution D generating examples rather than the loss rate on a training set.

Sometimes ECOC is analyzed assuming the errors are independent. This gives much better results, but seems less justifiable in practice. For example, in any setting with label noise, errors are highly *dependent*.

Proof. When the loss rate is zero, the correct label is consistent with all n classifiers, and wrong labels are consistent with at most $n-d$ classifiers, since C has minimum distance d. In order to wrongly predict the multiclass label, at least $d/2$ binary classifiers must err simultaneously. Since every multiclass error implies at least a fraction $d/2n$ of the binary classifiers erred, a binary loss rate of ϵ can yield at most a multiclass loss rate of $2n\epsilon/d$. ∎

C Proof of The Weighted One-Against-All Reduction

Proof. (of Theorem 1) There are two forms of errors, false positives (predicting 1 when the correct label is 0) and false negatives (predicting 0 when the correct label is 1). We show that an adversarial binary classifier trying to induce subset errors with maximal efficiency has three possible strategies, then we analyze these strategies.

First, notice that it is never more efficient to have multiple false positives because the expected probability of a subset error grows sublinearly. For any number p of true positives, the probability of a subset error with two false positives is $\frac{2}{p+2}$, which is less than $\frac{2}{p+1}$, the probability of erring on two subset examples when investing one false positive in each. For $p = 0$, only one false positive is required to always err.

Now let f be the number of false negatives. If $f < n$ (note that f can be at most n), there must be one false positive; otherwise the error rate would be 0. For f false negatives and one false positive, we have a loss rate of $\frac{1}{n-f+1}$ with the adversary paying importance $fi_1 + i_0$. To improve error efficiency, it is beneficial for the adversary to increase f if

$$\frac{\frac{1}{n-(f+1)+1}}{(f+1)i_1 + i_0} > \frac{\frac{1}{n-f+1}}{fi_1 + i_0},$$

or equivalently if $i_0 > i_1(n-2f)$. Otherwise, it is more efficient to decrease f. Thus an optimal adversary must choose either $f = 0$ or $f = n$.

For the $f = 0$ case we have error efficiency $\frac{\frac{1}{n+1}}{i_0}$.

For the $f = n$ case, the adversary can have 0 or 1 false positives. These cases have loss rates of $\frac{r-n}{r}$ and 1 with importance consumption of ni_1 or $ni_1 + i_0$, respectively.

Thus the adversary's most efficient strategy is given by

$$\max\left\{\frac{r-n}{rni_1}, \frac{1}{ni_1 + i_0}, \frac{1}{(n+1)i_0}\right\} = \frac{1}{r}.$$

Since the maximal error efficiency is $\frac{1}{r}$ and there are r classifications, a binary importance weighted loss of ϵ implies a subset loss rate of ϵ. However, the importance weighted loss is unnormalized since $\frac{n}{r}i_1 + \frac{r-n}{r}i_0 = \frac{r}{n+1}$ for $r \leq n^2 + n$ or $(\frac{r+1}{n+1} - \frac{n}{r})$ for $r > n^2 + n$. Taking an expectation over n according to D, we get the result. ∎