Reductions From Supervised Learning To Classification

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 problem into a solver for another problem. We discuss and analyze reductions which automatically convert classification solvers into solvers in other supervised learning domains.

1 Introduction

1.1 Motivation

There are many natural learning domains such as *classification*, *cost sensitive classification*, *ranking*, *reinforcement learning*, *outlier detection* and others. Two approaches are possible for dealing with this variety of learning domains. One possibility is to analyze every domain in great detail, and produce specific analysis and algorithms for each domain. The other possibility is to construct reductions from new or unexplored domains to a thoroughly analyzed domain, automatically transferring existing theory and algorithms from the well-understood domain into new domains. Here, we investigate the latter approach, which allows us to simultaneously:

1. Show how related these natural learning domains are. Two learning domains are related if we have a short efficient reduction from one to the other and vice versa.
2. Extend much of the hard work of existing theories into new domains.
3. Provide algorithms for solving these natural learning domains, when combined with a basic learning algorithm.

In addition to the above theoretical motivations, there appears to be some significant empirical evidence (see [9], [4], [3]) that this style of analysis often produces learning algorithms that perform well in practice.

1.2 Method

A natural question when working with reductions is: "What should we we reduce to and from?" Our answers are "to classification" and "from any formalized supervised learning domain". The choice of classification is motivated on three counts:

1. Classification has been previously studied in significant detail.
2. Classification seems to be the simplest possible supervised learning domain.

3. Classification performance is upper and lower boundable under the (occasionally met) assumption that data is drawn i.i.d. from some (unknown but fixed) distribution D.

Two reasonable alternatives are density estimation (which satisfies (1) but not (2) and (3)) and regression (which satisfies (1) and (3), but not (2)). We use classification because the strongest case can be made for it.

Another natural question is "What is a good reduction?" It is easy to agree that a good reduction has:

1. Good performance in the supervised learning domain's metric of success.
2. Low computational overhead.
3. Few invocations of the classifier learner.
4. Parallel invocations of the classifier learner.
5. Obedience of the invariants that the classifier learner needs to succeed.

Of the above, (1) is tricky. We *know* that classification is hard in general so it's difficult to say that any algorithm using a classifier learner as a subroutine can perform well. To address this we make an assumption:

For the classification problems created by our reduction, our classifier learning algorithm succeeds with error rate ϵ or less.

This assumption is powerful because it allows us to abstract away the difficulty of classification (impossible in the worst case, but often feasible in practice) and focus on the efficiency and error transformation characteristics of the reduction itself.

This assumption also invites abuse. Some very difficult problems (such as predicting plain text given examples of cipher text and plain text) can be phrased as learning problems. An analysis given this assumption may suggest that a reduction performs well on this problem. This is not so, simply because the assumption is not met in practice.

A more subtle form of assumption misuse occurs when an otherwise reasonable reduction fails to obey (5). One standard way this occurs is when the classification problem created is too noisy. If we considered Adaboost [4] in this framework, it sometimes fails (5) because it concentrates learning on a subset of examples which are mislabeled by noise. Another common misuse, which occurs in Bagging [2], is failing to produce a dataset which is drawn i.i.d. from the appropriate distribution.

1.3 Prior Work

There is quite a bit of significant prior work which is deeply related to the reductions analyzed here (perhaps this attests to the natural character of the notions considered). One of the goals of this paper is to unify these algorithms in a common framework.

Boosting algorithms [4] are reductions from (typically binary but 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. Some boosting variants have suffered with respect to criterion (5) above: they put too much emphasis on noisy data points (see [5] for an analysis).

Bagging [2] can be seen as a self-reduction of classification to classification. Bagging is useful because it turns learning algorithms with high "variance" (dependence on the exact examples seen) into voting classifiers with lower "variance". Like some Boosting variants, Bagging also suffers with respect to criterion (5): the examples that it feeds into the classifier learning algorithm are not independently distributed according to the original distribution.

Error Correcting Output Codes (ECOC) [3] is a technique for solving multi-class classification given a solver for binary classification (with significant positive empirical results).

Pitt and Warmuth [6] also constructed and analyzed a form of reduction between classification problems called *prediction-preserving reducibility*. The reductions presented here are similar in the general sense that the goal is to preserve accuracy, but there are several differences: (1) our reductions are between arbitrary supervised learning *domains* (see Section 2 for precise definitions) rather than between sets of classification problems; (2) our reductions are representation independent, while the notion of prediction-preserving reducibility defined in [6] inherently depends on the type of representation encoding the classification problems; (3) prediction-preserving reducibility is a *computational* notion, while we are more concerned with the type of *information* that can help in solving problems in a given domain, although we do consider the efficiency of our reductions; (4) prediction-preserving reductions mimic many-one reductions in complexity theory, which make only one call to the oracle (i.e., problem they reduce to) and output the oracle's answer as its own answer. We consider less restrictive Turing-type reductions, in which there is no restriction on how the oracle is used, making the setting directly applicable to ECOC, Boosting, and other common transformations[4]; (5) finally, the reductions of [6] were motivated by showing negative, *non*predictability results, while we are motivated by actually transfering learning algorithms between domains.

1.4 Layout

1. We first formally define several natural problems. These problems are the ones that we reduce from and to.
2. We state reductions to binary classification.
3. We state reductions to importance weighted classification.
4. We discuss the meaning of these results.

2 Basic Definitions

2.1 Supervised Learning

We first formalize some basic definitions which we rely upon throughout the paper.

Definition 21 *(Supervised Learning Domain) A supervised learning domain is specified by a set* (K, Y, l) *where* K *is some extra information available at training time and to the loss function,* Y *is the space of predictions and* $l : K \times Y \to [0, \infty)$ *is a loss function.*

[4] Abe and Watanabe [1] defined a Turing analogue of the prediction preserving reducibility of [6], and showed that it is strictly weaker than the original definition. More precisely, they constructed two prediction problems such that one is Turing reducible, but not many-one reducible to the other; thus one oracle call with no postprocessing of the answer is strictly less powerful than polynomially many calls. However, these problems are not natural in the sense that it cannot be efficiently determined whether a given instance belongs to any concept in the concept representation class encoding the problem.

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

Definition 22 *(Supervised Learning Problem) A supervised learning problem is a tuple (D, X, K, Y, l), where (K, Y, l) is a supervised learning domain, and D is a distribution over $X \times K$.*

A supervised learning problem is essentially a domain plus the distribution D (unknown at learning time) necessary to make the problem well specified. The goal in solving the supervised learning problem is to find a "hypothesis" $h : X \to Y$ which minimizes the loss with respect to D, i.e.

$$\arg\min_h E_{(x,k)\sim D} l(k, h(x)).$$

Naturally, for any particular hypothesis h, the loss rate:

$$h_D = E_{(x,k)\sim D} l(k, h(x))$$

is a fundamental quantity of interest.

Definition 23 *(Supervised Learning Algorithm) A supervised learning algorithm A takes as input a finite set of examples $(X \times K)^m$ and outputs a hypothesis $h : X \to Y$.*

Standard supervised learning algorithms such as support vector machines and logistic regression fit this definition. A "good" supervised learning algorithm approximately achieves the learning problem goal for natural learning problems in some domain. Notice that these algorithms do not actually work for *all* learning problems in their domain since they don't actually work for an arbitrary input space X. However our idealized supervised learning algorithm does as an abstraction.

We instantiate several domains next to give some intuition (and for future use).

Example 1. (Binary Classification) Binary classification is a supervised learning domain where $K = Y = \{0, 1\}$ and $l(k, y) = I(k \neq y)$.

The binary classification domain is appropriate when we would like to learn a predictor for "right" or "wrong" choice.

Example 2. (Multiclass Classification) Multiclass classification is a supervised learning domain where $Y = K = \{1, ..., \ell\}$ and $l(k, y) = I(k \neq y)$.

Multiclass classification is the obvious generalization of classification to situations with more than two choices.

Example 3. (Importance Weighted Classification) Importance weighted classification is a supervised learning domain where $Y = \{1, ..., \ell\}$, $K = \{1, ..., \ell\} \times [0, \infty)$ and $l((k, i), y) = iI(k \neq y)$. Since all importance weighted classification problems are unchanged under a rescaling of importance, we assume that $E_{(x,k,i)\sim D} i = 1$.

The importance weighted classification domain is appropriate when we want a predictor, but some of these predictions are more important than others.

Example 4. (Manyright Classification) Manyright classification is a supervised learning domain where $Y = \{1, ..., \ell\}$, $K = 2^{\{1,\ldots,\ell\}}$ and $l(k, y) = I(y \notin k)$.

Manyright classification is a generalization of multiclass classification to situations with multiple "right" choices.

Example 5. (Cost Sensitive Classification) Cost sensitive classification is a supervised learning domain where $Y = \{1, ..., \ell\}$, $K = [0, \infty)^\ell$ and $l(k, y) = k_y$. A cost sensitive classification problem is unchanged under two transformations: (1) for each example subtracting a constant so that the minimal cost is zero; (2) rescaling the costs. Using these properties, we assume that $\min_y k_y = 0$ and $E_{x,k_1,...,k_\ell \sim D} \sum_y k_y = 1$.

Cost sensitive classification is appropriate when each choice has a varying value. The binary version of this domain is essentially equivalent to importance weighted binary classification.

2.2 Reductions

We propose the notion of error limiting reducibility between learning domains. Here, one learning domain reduces to another if we can solve problems in the first domain using a black box that solves problems in the second. The meta-algorithm that does this is the reduction. We are fundamentally interested in reductions that perform well whenever the oracle learning algorithm performs well. This leads to the following definition.

Definition 24 *(Supervised Learning Reduction) A learning reduction from domain (K, Y, l) to domain (K', Y', l') is a learning algorithm for (K, Y, l) that has oracle access to a learning algorithm A' for (K', Y', l'). The reduction must satisfy the following property for some monotone nondecreasing continuous function $g : [0, \infty) \to [0, \infty)$ with $g(0) = 0$.*

(Error Limitation) For all X, and for all distributions D over $X \times K$,

$$E_{(x,k)\sim D} l(k, h(x)) \le g(\epsilon),$$

where h is the hypothesis output by the reduction, ϵ is given by

$$\epsilon = \max_{induced\ D'} E_{(x',k')\sim D'} l'(k', h_{D'}(x')),$$

where the maximum is taken over distributions D' induced on $X' \times K'$ by the reduction (see a precise description below), and $h_{D'}$ is the output of an invocation of A' on examples drawn from D'.

To relate the performance of the reduction to the performance of the oracle A', we must specify a distribution D' for each invocation of A'. A set of examples $S = \{(x_1, k_1), \ldots, (x_m, k_m)\}$ input to the reduction induces a measure D_S over $X' \times K'$. This distribution is given by

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

where the expectation is over possible sets of examples S' generated by the reduction from S, and $\#(x', k')$ is the number of times (x', k') appears in S'. The measure D_S is a measure on a discrete set of points. We extend this to a more general measure by taking the expectation over S. More precisely, $D' = E_S D_S$. This gives us a well defined distribution to measure performance with respect to, given a distribution on the inputs of the reduction.

One issue arises: What is D' for the second sequential invocation of the oracle? Suppose that the first invocation produces a function h. We replace the first invocation with the oracle which always returns h, and use the above definition to find D' for the second invocation.

One important thing to notice here is that while we talk about "distributions" and "measures" over samples, none of the above definitions require that samples be drawn independently.

It is not hard to see that reductions compose.

Theorem 1. *(Reduction Composition) If R_{12} is a reduction from domain $\mathcal{D}_1$ to $\mathcal{D}_2$ with error limitation g_{12} and R_{23} is a reduction from domain $\mathcal{D}_2$ 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 $g_{21} \circ g_{23}$.*

Proof. Given oracle access to any learning algorithm A for $\mathcal{D}_3$, R_{23} yields a learning algorithm for $\mathcal{D}_2$ such that for all X_2 and D_2 (subscripts denote the association with the corresponding domains), we have $E_{(x,k)\sim D_2} l(k, h'(x)) \leq g_{23}(\epsilon')$, where ϵ' is the maximum expected loss over the invocations of A, as described in Definition 24. We also know that R_{12} (when given oracle access to any learning algorithm for $\mathcal{D}_2$, and in particular to $R_{23}(A)$) yields a learning algorithm for $\mathcal{D}_1$ such that for all X_1 and D_1, the following holds: $E_{(x,k)\sim D_1} l(k, h(x)) \leq g_{12}(\epsilon)$, where ϵ is the maximum expected loss over the invocations of the oracle $R_{23}(A)$; but since for any X_2 and D_2 (in particular the ones induced by R_{12}), this expected loss is bounded by $g_{23}(\epsilon')$, and g_{12} is continuous and monotone, we actually have $E_{(x,k)\sim D_1} l(k, h(x)) \leq g_{12}(g_{23}(\epsilon'))$, as desired. Finally, notice that $g_{12} \circ g_{23}$ is continous, monotone nondecreasing and $g_{12}(g_{23}(0)) = 0$; moreover, if both g_{12} and g_{23} are polynomials, then so is $g_{12} \circ g_{23}$.

Our definition of a learning reduction is analogous to the definition of a Turing reduction in complexity theory. In both cases a reduction from problem A to problem B is a procedure that solves A using B as a black box. (In the standard definition of a Turing reduction, both A and B are sets, and a reduction decides the membership in A using a membership oracle for B.) However, there are several important differences between reductions in machine learning and complexity theory. First, in complexity theory reductions are viewed mainly as means for classifying the relative hardness of problems. This is motivated by the fact that showing any explicit problem to be hard is seemingly difficult, so one turns to tools for showing relative hardness. If A reduces to B, it is interpreted as saying that B is at least as hard as A; in particular, if A is firmly believed to be hard, then a reduction indicates that B is also hard (indeed, otherwise A would be easy).

In the context of machine learning, reductions are actually viewed as constructive tools for reusing algorithms for one problem to solve another (showing that A is no harder than B). Another major difference is that for a Turing reduction to be meaningful, it has to be computationally limited. If a reduction yields a solution to A, it should not be due to significant computations hidden in the reduction itself. Thus restricting the power of a reduction is a way of enforcing that the reduction does something qualitatively different than directly solving A, i.e., uses its oracle in a meaningful way. In the context of learning, on the other hand, reductions without a restriction of the computational power are still meaningful. The hardness of solving a learning problem is not so much due to the lack of computational power as to the lack of *information* available to the algorithm when it is run. (The learning domain is typically known, but the learning *problem* is given only in the form examples.) Computational analogs of learning reductions (capturing efficient transformations of algorithms) were investigated elsewhere by Pitt and Warmuth [6].

The error limitation in our definition of a reduction states that for any D, the loss incurred by the reduction is upper bounded by a function of the loss incurred by the oracle on the derived problems D'. Notice that this does not necessarily force the reduction to use A' in a nontrivial way; for some choices is may be the case that a learning domain is (near) optimally solvable, and the reduction does well regardless of the performance of the oracle. Reductions from easy, solvable domains may simply be uninteresting. In complexity theory, such uninteresting cases are ruled out by considering only "efficient reductions" that are in some computational sense weaker than the best known algorithms for the problems they reduce from.

2.3 Properties of Reductions

We now discuss some desirable properties of reductions.

Invertibility A reduction R from (Y, K, l) to (Y', K', l') is *learning preserving* if there exists a reduction R' from (Y', K', l') back to (Y, K, l) such that for every distribution D' over $X' \times K'$, every oracle A', if A' has loss rate at most ϵ with respect to D', then $R'(R(A'))$ has loss rate at most $f(\epsilon)$ with respect to D', for some well-behaved function f.

Notice that invertibility does not immediately follow from the fact that (Y, K, l) and (Y', K', l') are reducible to one another in the sense of Definition 24. By the definition, 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'''. We stress that invertibility requires the loss rate of the double reduction $R'(R(A'))$ on D' to be bounded in terms of the loss rate of A' on the *same* distribution D' (not the induced D'''). Intuitively, it forces the reduction not to reduce easy problems to hard queries for the oracle. Indeed, the only reason why $R'(R(A'))$ could have a large error rate on D' (despite A' doing well on D') is because A' could be doing badly on D'''.

Efficiency If two domains are reducible, it is reasonable to say that they are similar, *modulo the complexity of the reduction.* The more restrictive the reduction, the stronger the statement. Thus it is useful to quantify the power of reductions. We analyze the computational power of a reduction and its form of oracle access.

Time Efficiency: We want the transformation to be computatioanally efficient assuming that all calls to the oracle 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 parallelized (i.e., all queries but all must be asked before any of them is answered); (3) the maximum expected number of samples input to the oracle. It is worth noting that parallel calls to the oracle can be turned into a single call (at least theoretically). Suppose that we want to 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 $S = \{(\langle x, i\rangle, k) : \langle x, k\rangle \in S_i\}$, call the oracle on S to get a hypothesis h, and define $h_i(x) = h(\langle x, i\rangle)$. It is unclear whether this is a better mechanism in practice.

Independence Preservation Many common learning algorithms are built under the assumption that samples are drawn independently. Reductions which take an independently drawn dataset, and construct dependent datasets for the oracle learning algorithm are sometimes undesirable for this reason.

3 Reductions, the easy way

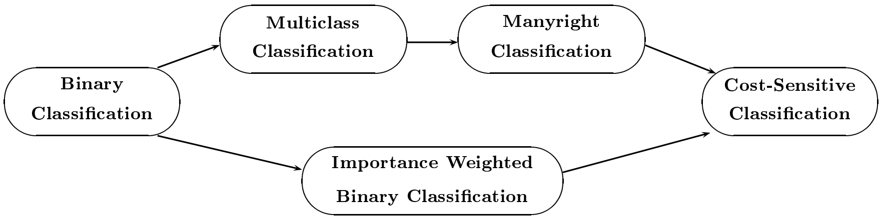

Fig. 3.1. A figure showing the "easy" direction for reduction amongst the learning problems discussed here.

The learning problems above have an "easy" direction for reduction related to whether a particular problem is definable as a subset of other problems.

For example, solving binary classification with importance weighted binary classification is easy, because importance weighted binary classification for $i = 1$ is identical to binary classification. Figure 3.1 shows the easy direction for these reductions. The remainder of this paper is devoted to reductions in the other, "hard" direction.

4 Reductions to Binary Classification

In this section we discuss the characteristics of several reductions to the binary classification domain.

4.1 Reduction from Importance Weighted Classification to Classification

The "Costing" algorithm [9] is a reduction of importance weighted classification to classification with the following characteristics:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
$\epsilon E_{(x,y,i)\sim D} i$	No	$O(Tm)$	T parallel	Yes

4.2 Reduction from Multiclass Classification to Binary Classification

There are several well known methods of reduction which we analyze for error efficiency next. Before we start, it's important to note that there are (see [8],[7]) 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 ℓ classifiers h_i, using the mapping:

$$(x, y) \to (x, I(y = i))$$

In order to construct a multiclass classifier from the binary classifier we use the following procedure:
Predict(h, x)

1. If there exists a label i such that $h_i(x) = 1$ then predict i, breaking ties with randomization.
2. Otherwise predict randomly.

This reduction has the following properties.

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
$(\ell - 1)\epsilon$	No	$O(\|S\|\ell)$	ℓ parallel	Yes

Note, that the many-calls to one call trick in section 2.3 (call efficiency) applies here.

Theorem 41 *(One-against-all error efficiency) If the binary classifiers have error rate ϵ, the one-against-all reduction has error rate at most:*

$$(\ell - 1)\epsilon$$

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 $\frac{\ell-1}{\ell}$ fraction of the time (assuming all the other classifiers are correctly outputing 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 creating errors is: $\frac{\frac{\ell-1}{\ell}}{1} = \frac{\ell-1}{\ell}$, $\frac{\frac{n}{n+1}}{n} = \frac{1}{n+1}$, and $\frac{1}{n+1}$. Taking the maximum, we get $\frac{\ell-1}{\ell}$. Multiplying by ℓ (because we have ℓ opportunities to err, one for each classifier) we get the bound.

The Tree Reduction In the tree reduction we construct $\ell - 1$ binary classifiers which distinguish between the labels using a binary tree. The tree has depth $\log_2 \ell$, and each internal node is associated with a set of labels that can reach the node, assuming each classifier above it predicts correctly. The set of labels at that node is split into two sets of half the size, and a classifier is learned to distinguish between the two sets of labels.

Predictions are made by following the chain of classifications down to the leaves, which are associated with a unique label. This reduction has the following properties:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
$\epsilon \log_2 \ell$	No	$O(\|S\|\ell)$	$\ell - 1$ parallel	Yes

Note, that the many-calls to one call trick in section 2.3 (call efficiency) can be applied here.

Theorem 42 *(Tree error efficiency) If the binary classifiers have error rate ϵ, the tree reduction has error rate at most:*

$$\epsilon \log_2 \ell$$

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

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 error rate for this reduction is at most $2n\epsilon/d$, as we shall prove in Theorem 43 below.

We mention three codes of particular interest. The first is when the codewords are a subset of the rows of an 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 2ℓ, and the error rate is at most 4ϵ.

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

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

For the Hadamard code, we get the following:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
4ϵ	No	$O(\ell\|S\|)$	2ℓ parallel	Yes

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

Theorem 43 *(ECOC error efficiency) If the binary classifiers have error rate ϵ, the ECOC reduction has error rate less than:*

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

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 error 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 error rate of ϵ can yield at most a multiclass error rate of $2n\epsilon/d$.

5 To Importance Weighted Classification

In this section we analyze and discuss reductions to weighted binary classification, a less explored possibility. Given the "Costing" reduction from importance weighted classification to classification, the results here compose to produce new (or alternative) reductions to binary classification.

5.1 Boosting

The algorithm Adaboost [4] run for T rounds can be thought of as a reduction from classification to importance weighted classification with the following properties:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
$e^{-2T(1/2-\epsilon)^2)}$	No	$O(\|S\|T)$	T Adaptive	Yes

The $g(\epsilon)$ entry above deserves some explanation. Adaboost is typically analyzed with respect to weak learning on an importance weighted *training set* rather than weak learning with respect to the importance weighted distribution constructed by randomizing over the samples input to the boosting algorithm. Nonetheless, the same analysis holds when we assume weak learning on the importance weighted distributions created for the oracle ("weak") learning algorithm.

5.2 Manyright Classification

The One-against-all importance Reduction This reduction is an optimized one-against-all reduction for manyright classification to importance weighted binary classification. It appears to be new.

In the one-against-all reduction, we learn ℓ classifiers, one for each possible output label using data from the mapping:

$$(x, y_1, y_2, ..., y_n) \rightarrow (x, I(\ell \in \{y_1, ..., y_n\}), i_{I(\ell \in \{y_1,...,y_n\})})$$

where $i_0 = \frac{\ell}{n+1}$ and $i_1 = \frac{\ell}{n+1}$ if $\ell \leq n^2+n$ or $i_1 = \frac{\ell-n}{n}$ for $\ell \geq n^2+n$. We use the same prediction function as in section 4.2.

The reduction has the following characteristics:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
$\frac{\ell}{n+1}\epsilon$	No	$O(\|S\|\ell)$	ℓ parallel	Yes

The value of $g(\epsilon)$ is actually slightly better than for the standard one-against-all reduction for multiclass to binary. 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 table 1. From these results, we see that there is a consistent benefit to using this reduction over the simple one-against-all reduction. Next, we prove the $g(\epsilon)$ claim.

	Boosted Naive Bayes		SVM	
dataset	one-ag-all	one-ag-all-imp	one-ag-all	one-ag-all-imp
splice	0.0652	0.0652	0.222	0.219
solar	0.163	0.156	0.153	0.153
anneal	0.00535	0.000376	0.0669	0.0438
sat	0.183	0.181	0.243	0.227
letter	0.410	0.374	0.638	0.525
pendigits	0.146	0.139	0.165	0.151
soybean	0.188	0.188	0.225	0.104

Table 1. Test set error rates by the one-against-all and one-against-all importance reductions for seven UCI multiclass datasets.

Theorem 51 *(Manyright classification error efficiency) Let $c_{\ell n} = \frac{\ell}{n+1}$ for $\ell \leq n^2 + n$ or $c_{\ell n} = \frac{\ell+1}{n+1} - \frac{n}{\ell}$ otherwise. If the binary classifiers have (normalized) loss rate ϵ, the one-against-all reduction has error rate at most:*

$$\epsilon E_{x,y_1,\ldots,y_n \sim D} c_{\ell n} \leq \epsilon E_{x,y_1,\ldots,y_n \sim D} \frac{\ell}{n}$$

This theorem should be compared directly with the one-against-all reduction to binary classification which had an error efficiency of $\ell - 1$. Here, with $n = 1$, we get $c_{\ell n} = \frac{\ell+1}{2} - \frac{1}{\ell} = \frac{(\ell+2)(\ell-1)}{2\ell}$ implying an error efficiency of about $\frac{\ell}{2}$, or about half the error rate of the earlier reduction. Reducing to importance weighted binary classification has an improved efficiency.

Proof. Outline: For a particular example with n "right" labels out of ℓ possible labels. We first show that there are 3 possible "most efficient" methods for a binary classifier to induce a multiclass error. These possibilities are: 1) a single false positive error, 2) all possible false negative errors, or 3) (1) and (2). After showing these are the only possibilities, we analyze which is the worst, and then take an expectation over n and ℓ.

There are two forms of errors, false positive and false negative. It is never more efficient to have multiple false positive errors. This is because for any number $n \geq p > 0$ of true positives, $\frac{2}{p+2} < \frac{1}{p+1} + \frac{1}{p+1}$ and for $p = 0$, only one false negative is required to err with probability 1.

If the optimal error rate occurs for $f < n$ false negatives then there must be 1 false positive (since the error rate is 0 otherwise). For $f < n$ false negatives and 1 false positive, we have an error rate of $\frac{1}{n-f+1}$ using an importance $fi_1 + i_0$ giving us an efficiency of $\frac{\frac{1}{n-f+1}}{fi_1+i_0}$. Since

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

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

$$\Leftrightarrow (fi_1 + i_0)(n-f+1) > ((f+1)i_1 + i_0)(n-f)$$

$$\Leftrightarrow fi_1 + i_0 > i_1(n-f)$$
$$\Leftrightarrow i_0 > i_1(n-2f)$$

We can always improve the error efficiency by increasing f whenever $i_0 > i_1(n-2f)$ and by decreasing f otherwise. Consequently, the number of false negative errors is either $f=0$ or $f=n$.

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

For the $f=n$ case there are again two possibilities: one or zero false positives. These cases have an error rate of $\frac{\ell-n}{\ell}$ and 1 with an importance consumption of ni_1 or ni_1+i_0 implying error efficiencies of $\frac{\frac{\ell-n}{\ell}}{ni_1}$ and $\frac{1}{ni_1+i_0}$.

The next step is determining which of these failure modes is "most efficient" in the sense that it induces the largest number of multiclass error for the least cost in "importance".

The maximal error efficiency is $\max\left\{\frac{\ell-n}{\ell ni_1}, \frac{1}{ni_1+i_0}, \frac{1}{(n+1)i_0}\right\}$. There are two regimes for this maximum, $\ell \geq n^2+n$ and $\ell \leq n^2+n$.

For $\ell \leq n^2+n$, we get:$\max\left\{\frac{\ell-n}{\frac{\ell^2 n}{n+1}}, \frac{1}{\ell}, \frac{1}{\ell}\right\} = \frac{1}{\ell}$.

For $\ell \geq n^2+n$, we get: $\max\left\{\frac{1}{\ell}, \frac{1}{\ell-n+\frac{\ell}{n+1}}, \frac{1}{\ell}\right\} = \frac{1}{\ell}$.

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

5.3 Multiclass cost sensitive classification

Here we attempt to reduce multiclass cost sensitive classification to simpler problems.

The One-wrong reduction Consider the reduction which maps $(x, \ell_1, \ell_2, ..., \ell_l) \to (x, y, \max_i \ell_i - \ell_y)$ for a uniformly chosen $y \in \{1, ..., l\}$. This dataset is fed into the importance weighted classification solver resulting in a classifier $c : X \to \{1, ..., l\}$ which is used to solve the classification problem.

The difficulty with this reduction is that it can be simply impossible to produce a predictor with a small error rate. In particular, it's only for lucky distributions where just one of the costs is large[5] that the problem we reduce to can have a small error rate. Nonetheless, in this lucky case, a very strong statement can be made.

The characteristics of this reduction are given in the following table:

$g(\epsilon)$	Learn Preserve	Time	Call	Indep. Preserve
ϵ	No	$O(\|S\|)$	1	Yes

Theorem 52 *(error transformation) If the importance weighted classifier has loss rate ϵ, then the cost sensitive classifier has expected cost less than ϵ.*

[5] Thus the name.

Proof. Assume that

$$\epsilon = E_{x,y,i\sim D^*} iI(c(x)\neq y)$$

$$= E_{x,\ell_1,\ldots,\ell_l\sim D, y\sim\{1,\ldots,l\}}(\max_i \ell_i - \ell_y)I(c(x)\neq y)$$

$$= E_{x,\ell_1,\ldots,\ell_l\sim D, y\sim\{1,\ldots,l\}} \max_i \ell_i I(c(x)\neq y) - E_{x,\ell_1,\ldots,\ell_l\sim D, y\sim\{1,\ldots,l\}} \ell_y I(c(x)\neq y)$$

$$= \frac{l-1}{l} E_{x,\ell_1,\ldots,\ell_l\sim D} \max_i \ell_i - E_{x,\ell_1,\ldots,\ell_l\sim D} \frac{\sum_i \ell_i - \ell_{c(x)}}{l}$$

$$= E_{x,\ell_1,\ldots,\ell_l\sim D} \frac{(l-1)\max_i \ell_i - \sum_i \ell_i + \ell_{c(x)}}{l}$$

$$= E_{x,\ell_1,\ldots,\ell_l\sim D} \frac{\ell_{c(x)} - \max_i \ell_i}{l} + \max_i \ell_i - E_{i\sim\{1,\ldots,l\}} \ell_i$$

Rewriting, we get:

$$E_{x,\ell_1,\ldots,\ell_l\sim D}\ell_{c(x)} = \epsilon - l[E_{x,\ell_1,\ldots,\ell_l\sim D} \max_i \ell_i - E_{i\sim\{1,\ldots,l\}}\ell_i] + E_{x,\ell_1,\ldots,\ell_l\sim D} \max_i \ell_i$$

$$= \epsilon - E_{x,\ell_1,\ldots,\ell_l\sim D}(l-1)\max_i \ell_i - \sum_i \ell_i$$

$$\leq \epsilon$$

Where the last inequality follows because on of the ℓ_i is 0.

5.4 The weighted all pairs reduction

We learn $\binom{\ell}{2}$ classifiers $h_{i,j}$ $(j > i)$ using the mapping

$$(x, k_1, k_2, \ldots, k_\ell) \to (x, I(k_i \leq k_j), \sum_{n=1}^{\ell} k_n).$$

We say label i beats label j if $h_{i,j}$ outputs 1, otherwise j beats i. The cost sensitive classifier predicts the label that beats the maximum number of other labels (breaking ties at random).

Theorem 53 *If the binary classifiers have error rate at most ϵ then the cost sensitive classifier defined by the above reduction has expected cost at most $\frac{\ell}{2}\epsilon$.*

Proof. Suppose $x \in X$ and $0 = k_{i_1} \leq k_{i_2} \leq \ldots \leq k_{i_\ell}$ are the true costs for x, where $i_1, i_2, \ldots i_\ell$ is some permutation of the labels. Let $C = \sum_{j=1}^{\ell} k_j$.

In order for label i_n to win the tournament outright (over label i_1 in particular), at least $n-1$ of the $2\ell - 3$ classifiers subscripted with i_1 and/or i_n must err. If at least $n-2$ of these classifiers err, it is possible for labels i_1 and i_n to tie (possibly in a 3-way tie with label i_2, however this is not an efficient use of errors for our adversary). Thus, if n errors are made by the binary classifiers, and $1 \leq n \leq \ell - 2$, then the cost incurred by our cost sensitive classifier is at most $\max(k_{i_{n+1}}, k_{i_{n+2}}/2)$.

Since the sum of the costs is C, for any n the largest possible value of k_{i_n} is achieved when

$$0 = k_{i_1} = k_{i_2} = \cdots = k_{i_{n-1}} < k_{i_n} = \ldots = k_{i_\ell} = C/(\ell + 1 - n).$$

So whenever $1 \leq n < \ell$ errors occur, the cost incurred is at most $C/(\ell - n)$, which is $\leq Cn/(\ell - 1)$. The same upper bound holds for $n \geq \ell$, since $\ell - 1$ errors are already enough to produce the maximum possible output.

Then by definition,

$$\underset{(x,k_1,k_2,\ldots,k_\ell)\sim D}{\mathrm{E}}(Cn) = \sum_{i<j} E_{(x,k)\sim D'} l(k, h_{i,j}(x)) \leq \binom{\ell}{2}\epsilon.$$

It follows that the expected cost for our cost-sensitive classifier is at most $\frac{1}{\ell-1}\binom{\ell}{2}\epsilon = \frac{\ell}{2}\epsilon$.

6 Conclusion

The definitions of reduction we outline here cover 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 characteristics.

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

Our definition of reduction here is limited to the supervised learning domain. It nonetheless appears that natural generalizations capturing even broader domains is possible. More work on basic definitions must be done.

References

1. N. Abe and O. Watanabe. TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems, 1992.
2. L. Breiman. Bagging predictors. Machine Learning, 26(2):123-140, 1996.
3. 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.
4. 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.
5. Adam Kalai and Rocco Servedio, "Boosting in the Presence of Noise" To appear in Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing (STOC 2003).
6. Lenny Pitt and Manfred Warmuth, "Prediction-Preserving Reducibility", Journal of Computer and System Sciences, 41:430–467, 1990.
7. Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297–336, 1999.
8. Robert E. Schapire. "Using output codes to boost multiclass learning problems". In Machine Learning: Proceedings of the Fourteenth International Conference, 1997 (ICML 1997).
9. Bianca Zadrozny, John Langford, and Naoki Abe. "Cost Sensitive Learning by Cost-Proportionate Example Weighting", ICDM 2003.