Machine Learning Techniques—Reductions Between Prediction Quality Metrics

Alina Beygelzimer and John Langford and Bianca Zadrozny

Abstract Machine learning involves optimizing a loss function on unlabeled data points given examples of labeled data points, where the loss function measures the performance of a learning algorithm. We give an overview of techniques, called reductions, for converting a problem of minimizing one loss function into a problem of minimizing another, simpler loss function. This tutorial discusses how to create robust reductions that perform well in practice. The reductions discussed here can be used to solve any supervised learning problem with a standard binary classification or regression algorithm available in any machine learning toolkit. We also discuss common design flaws in folklore reductions.

1 Introduction

Machine learning is about learning to make predictions from examples of desired behavior or past observations. Learning methods have found numerous applications in performance modeling and evaluation (see, for example, [33, 22, 37, 41, 43, 39]). One natural example of a machine learning application is fault diagnosis: based on various observations about a system, we may want to predict whether the system is in its normal state or in one of several fault states. Machine learning techniques are preferred in situations where engineering approaches like hand-crafted models simply can not cope with the complexity of the problem. In the fault diagnosis prob-

Alina Beygelzimer
IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 e-mail: beygel@us.ibm.com

John Langford
Yahoo! Research, New York, NY e-mail: jl@yahoo-inc.com

Bianca Zadrozny
Fluminense Federal University, Brazil e-mail: bianca@ic.uff.br

lem, it is reasonably easy to collect examples of resolved faults, but writing robust diagnosis rules is very difficult.

A basic difficulty in applying machine learning in practice is that we often need to solve problems that don't quite match the problems solved by standard machine learning algorithms. In fault diagnosis, for example, the cost of misclassifying a faulty state as a normal state is sometimes much higher than the cost of misclassifying a normal state as a faulty state. Thus binary classification algorithms, which don't take misclassification costs into account, do not perform well on this problem.

Reductions are techniques that transform practical problems into well-studied machine learning problems. These can then be solved using any existing base learning algorithm whose solution can, in turn, be used to solve the original problem. Reductions have several desirable properties.

- They yield highly automated learning algorithms. Reductions convert *any* learner for the base problem into a learning algorithm for the new problem. Any future progress on the base problem immediately translates to the new problem.
- Reductions are modular and composable. A single reduction applied to N base learners gives N new learning algorithms for the new problem. Simple reductions can be composed to solve more complicated problems.
- The theory of learning has focused mostly on binary classification and regression. Reductions transfer existing learning theory to the new problem.
- Reductions help us organize and understand the relationship between different learning problems.

An alternative to reductions is designing new learning algorithms or modifying existing ones for each new problem. While this approach is quite attractive to learning algorithm designers, it is undesirable in some situations. For example, some algorithms cannot be easily modified to handle different learning problems, as evidenced, for example, by inconsistent proposals for extending Support Vector Machines to multiclass classification (see [30]). More generally, we can expect that people encountering new learning problems may not have the expertise or time for such adaption (or simply don't have access to the source code of the algorithm), implying that a reduction approach may be more desirable.

A critical question when comparing the two approaches is performance. Our experience is that both approaches can be made to work well. There is fairly strong empirical evidence that reductions analysis produces learning algorithms that perform well in practice (see, for example, [18, 10, 47, 13, 38]). This tutorial shows how reductions can be easily used by nonexperts.

2 Basic Definitions

Data points, called *examples*, are typically described by their values on some set of *features*. In fault diagnosis, for example, each event can be represented as a binary vector describing which observations have been made (ping latency from one node

to another, for example). The space that examples live in is called the *feature space*, and is typically denoted by X.

The *label* of an example is what we are trying to predict. The space of possible labels is denoted by Y. In fault diagnosis, Y corresponds to the set of system states.

A *learning problem* is some unknown data distribution D over $X \times Y$, coupled with a loss function $\ell(y', y)$ measuring the loss of predicting y' when the true label is y. (In some problems below, the loss function also depends on additional information about the example.)

A *learning algorithm* takes a set of labeled training examples of the form $(x, y) \in X \times Y$ and produces a predictor $f : X \to Y$. The goal of the algorithm is to find f minimizing the expected loss $\mathbf{E}_{(x,y)\sim D}\,\ell(f(x), y)$.

There are two base learning problems, defined for any feature space X. In binary classification, we want to classify examples into two categories.

Definition 1. A *binary classification* problem is defined by a distribution D over $X \times Y$, where $Y = \{0, 1\}$. The goal is to find a *classifier* $h : X \to Y$ minimizing the *error rate* on D,

$$e(h, D) = \mathbf{Pr}_{(x,y)\sim D}\left[h(x) \neq y\right].$$

By fixing an unlabeled example $x \in X$, we get a *conditional distribution* $D|x$ over Y.

Regression is another basic learning problem, where the goal is to predict a real-valued label Y. The loss function typically used in regression is the squared error loss between the predicted and actual labels.

Definition 2. A *regression problem* is defined by a distribution D over $X \times \mathbb{R}$. The goal is to find a function $f : X \to \mathbb{R}$ minimizing the *squared loss*

$$\ell(f, D) = \mathbf{E}_{(x,y)\sim D}(f(x) - y)^2.$$

Organization: Section 3 shows how to solve binary classification problems where some examples are more important to classify correctly than others. It covers problems where false positives and false negatives have different costs as a special case. Section 4 demonstrates how regression algorithms can be used to choose among more than two alternatives. Section 5 covers a very board set of learning problems, where a decision is not only over multiple choices, but each prediction has a different associated cost. In Section 6, we discuss how to compute quantiles with binary classifier learners. Section 7 closes with the problem of reducing ranking, as measured by the Area Under the Receiver Operating Characteristic Curve (AUC), to binary classification.

3 Importance-Weighted Classification

Standard classification algorithms are designed to minimize the probability of making an incorrect prediction, treating all errors as equally costly. In practice, however,

some errors are typically much more costly than others. For example, in credit card fraud detection the cost of an undetected fraud is much higher than the cost of an extra security check. The problem is complicated by the fact fraud is very rare. Ignoring the misclassification costs may produce a classifier that misses all the fraud by classifying every example as belonging to the more frequent non-fraud case. This classifier would be doing very well in terms of not making many mistakes, but it would have a very high cost. Thus a good metric is essential when training and evaluating a cost-sensitive learner.

Current techniques for cost-sensitive decision making fall into three categories: The first approach is to make particular classification algorithms cost-sensitive (see, for example, [15]). Doing this well is often nontrivial and requires considerable knowledge of the algorithm. The second approach uses Bayes risk theory to assign each example to its lowest risk class [14, 46, 32]. This requires estimating conditional class probabilities and, if costs are stochastic, estimating expected costs [46]. The third category concerns black-box reductions for converting arbitrary classification learning algorithms into importance-weighted algorithms [14, 47]. MetaCost [14] (implemented in Weka [45]), estimates conditional probability distributions, and thus also belongs to the Bayes risk minimization category above.

Before describing concrete methods for importance-weighted decision making, it is instructive to look at a simple theorem described below.

Motivating Theory

An *importance-weighted classification* problem is defined by a distribution D over $X \times Y \times C$, where X is some feature space, $Y = \{0,1\}$ is the label space, and $C \subset [0,\infty)$ is the importance (or cost) associated with mislabeling the corresponding example. The goal is to learn a classifier $h : X \to Y$ minimizing the *expected cost*

$$\mathbf{E}_{(x,y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)],$$

given training examples of the form $(x,y,c) \in X \times Y \times C$. Here $\mathbf{1}(\cdot)$ is the indicator function which evaluates to 1 if its argument is true, and to 0 otherwise. Since cost information is typically not available at prediction time, this is reflected in the model. If the cost is available, it can be included in the set of features.

When the output space is binary, this formulation of cost-sensitive learning in terms of one number per example is more general than the commonly used cost matrix formulation [16, 14]. A cost matrix specifies the cost c_{ij} of predicting label i when the true label is j. If the label is binary, the costs are associated with false negatives (c_{01}), false positives (c_{10}), true negatives (c_{00}), and true positives (c_{11}). Given a cost matrix and an example (x,y), only two cost entries (c_{1y}, c_{0y}) are relevant for that example. These two numbers can be further reduced to one, $|c_{1y} - c_{0y}|$, because it is the difference in costs which controls the importance of correct classification. This difference is the importance c we use here. The formulation we use is more general because it allows the costs to be example dependent. For example, the cost

of a fraudulent charge can depend on the transaction amount. The multilabel case is covered by the more general cost-sensitive classification in Section 5. This section covers the binary case.

The following simple theorem is quite instructive.

Theorem 1. (Translation Theorem [47]) *For any importance-weighted distribution D, there exists a constant $\langle c \rangle = \mathbf{E}_{(x,y,c)\sim D}[c]$ such that for any classifier h,*

$$\mathbf{E}_{(x,y,c)\sim D'}[\mathbf{1}(h(x) \neq y)] = \frac{1}{\langle c \rangle}\mathbf{E}_{(x,y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)],$$

where D' is defined by $D'(x,y,c) = \frac{c}{\langle c \rangle}D(x,y,c)$.

Proof. Assuming for simplicity that X is finite,

$$\begin{aligned}\mathbf{E}_{(x,y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)] &= \sum_{x,y,c} D(x,y,c) \cdot c \cdot \mathbf{1}(h(x) \neq y) \\ &= \langle c \rangle \sum_{x,y,c} D'(x,y,c) \cdot \mathbf{1}(h(x) \neq y) \\ &= \langle c \rangle \, \mathbf{E}_{(x,y,c)\sim D'}[\mathbf{1}(h(x) \neq y)].\end{aligned}$$

∎

Despite its simplicity, this theorem is very useful because the right-hand side expresses the expected cost we want to control via the choice of h, and the left-hand side is the error rate of h under a related distribution D'. Thus choosing h to minimize the error rate under D' is equivalent to choosing h to minimize the expected cost under D.

The prescription for coping with cost-sensitive problems is now straightforward: re-weight the distribution in your training set according to the importances, so that the training set is effectively drawn from D'. Doing this in a correct and general manner is more challenging than it may seem.

There are two basic methods: (1) Transparent Box: supply the costs of the training data as example weights to the classifier learning algorithm; (2) Black Box: resample the training data according to these same weights.

The transparent box approach cannot be applied to arbitrary classifier learners, but it can be applied to those which use the data only to estimate expectations of the form $\mathbf{E}_{x,y\sim D}[f(x,y)]$ for some query function $f : X \times Y \to \{0,1\}$. Whenever a learning algorithm can be rewritten to fit this statistical query model [24], there is a simple recipe for using the weights directly. To get an unbiased estimate of the expectation with respect to D', one can use

$$\frac{1}{\sum_{(x,y,c)\in S} c} \sum_{(x,y,c)\in S} cf(x,y),$$

instead of using the sample mean of $f(x,y)$, where S is a set of training examples drawn independently from D. Such mechanisms for realizing the transparent box approach have been used for a number of weak learners used in boosting [18].

Neural networks, decision trees and Naive Bayes can all be expressed in this model, although it may require some understanding of the algorithms to see that. Support vector machines do not fit the model, because the produced classifier is explicitly dependent upon individual examples rather than on statistics derived from the entire sample. But there are still ways to incorporate importance weights directly (see, for example, [47]).

The black box approach has the advantage that it can be applied to any classifier learner, without requiring any knowledge of the learning algorithm.

Black Box: Sampling methods

Suppose that we do not have transparent box access to the learner. In this case, sampling is the obvious method to alter the distribution of examples, in order to use Theorem 1.

Simple sampling strategies: Sampling with replacement is a sampling scheme where each example (x,y,c) is drawn according to the distribution $p(x,y,c) = c/\sum_{(x,y,c)\in S} c$. A number of examples are drawn to create a new dataset S'. It may seem at first that this method is useful because every example is effectively drawn from the distribution D'. In fact, it can result in severe overfitting due to the fact that examples in S' are not drawn *independently* from D'. Also, as shown in Elkan [16], creating duplicate examples has little effect on classifiers produced by standard Bayesian and decision tree algorithms.

Sampling without replacement is also not a solution to this problem. In this scheme, an example (x,y,c) is drawn from the distribution $p(x,y,c) = c/\sum_{(x,y,c)\in S} c$, and the drawn example is removed from S. This process is repeated, drawing from an increasingly smaller set according to the weights of the examples remaining in the set. To see how this method fails, simply note that sampling m examples from a set of size m results in the original set, which by assumption is drawn from the distribution D, instead of D' as desired.

Cost-proportionate rejection sampling: We will present another sampling scheme based on rejection sampling [34], which allows one to draw examples independently from the distribution D' given examples drawn independently from D. In rejection sampling, examples from D' are obtained by first drawing examples from D, and then keeping the example with probability proportional to D'/D. In our case, $D'/D \propto c$, so we accept each c-important example with probability c/Z, where Z is a normalizing constant satisfying $\max_{(x,y,c)\in S} c \leq Z$. [1] Rejection sampling results in

[1] In practice, we choose $Z = \max_{(x,y,w)\in S} c$ so as to maximize the size of S'. A data-dependent choice of Z is not formally allowed for rejection sampling, but the introduced bias appears small when $|S| \gg 1$.

a set S' which is generally smaller than S. Notice that if examples in S are drawn independently from D, then examples in S' are going to be distributed independently according to D'.

A simple corollary of Theorem 1 says that any classifier achieving approximate error rate minimization on D' is guaranteed to produce approximate cost-minimization on D.

Corollary 1. *For all importance-weighted distributions D and all binary classifiers h, if*

$$\mathbf{E}_{(x,y,c)\sim D'}[\mathbf{1}(h(x) \neq y)] \leq \varepsilon,$$

then

$$\mathbf{E}_{(x,y,c)\sim D}[c \cdot \mathbf{1}(h(x) \neq y)] \leq \langle c \rangle \varepsilon,$$

where $\langle c \rangle = \mathbf{E}_{(x,y,c)\sim D}[c]$.

Proof. Follows immediately from Theorem 1. ∎

Cost-proportionate rejection sampling with aggregation (Costing): Given the same original training sample, different runs of cost-proportionate rejection sampling will produce different training subsamples. Since rejection sampling produces small subsamples allowing us to learn quickly, we can take advantage of the runtime savings to run the base learner on multiple draws of subsamples and average over the resulting classifiers. This method is called Costing [47].

Algorithm 1: Costing (learning algorithm A, training set S, count t)

for $i = 1$ *to* t **do**
  S' rejection sample from S with acceptance probability c/Z.
  Let $h_i = A(S')$
return $h(x) = \text{sign}\left(\sum_{i=1}^{t} h_i(x)\right)$

The goal in averaging is to improve performance. There is significant empirical evidence that averaging can considerably reduce overfitting suffered by the learning algorithm, despite throwing away a fraction of the samples. There are also several theoretical explanations of the empirical success of averaging methods (see, for example, [17]). In fact, Bagging [10] and Boosting [18] can both be viewed as reductions. Boosting algorithms [18, 23] reduce from classification with a small error rate to importance weighted classification with a loss rate of nearly $\frac{1}{2}$. Bagging [10] is a self-reduction of classification to classification, which turns learning algorithms with high variance (i.e., dependence on the exact examples seen) into a classifier with lower variance.

Since most learning algorithms have running times that are superlinear in the number of examples, the overall computational time of costing is generally much

smaller than that of a learning algorithm using the original sample set S. Costing was shown to have excellent predictive performance and dramatic savings of computational resources (see [47]), which is especially important in applications that involve massive amount of data such as fraud and intrusion detection, and targeted marketing.

4 Multiclass Classification

Multiclass classification is just like binary classification, except that there are more than two choices available. Naturally, there are many applied problems where a decision over more than two possibilities is necessary, with such examples as optical character recognition, textual topic classification, and phoneme recognition.

Formally, a *k-class classification problem* is defined by a distribution D over $X \times Y$, where X is some feature space and $Y = \{1, \ldots, k\}$ is the set of possible labels. The goal is to find a *classifier* $h : X \to Y$ minimizing the *error rate* on D, $e(h, D) = \mathbf{Pr}_{(x,y)\sim D}[h(x) \neq y]$.

There are several methods for solving multiclass classification problems directly [42, 44, 9, 11, 30, 31, 19]. These approaches *can* be made to work well, but the corresponding optimization problems are often fairly involved.

Given that we have many good binary learning algorithms and many multiclass problems, it is tempting to create meta-algorithms which use binary classifiers to make multiclass predictions.

4.1 One-Against-All

Perhaps the simplest such scheme is *one-against-all* (OAA). The OAA reduction creates a binary classification problem for each of the k classes. The classifier for class i is trained to predict whether the label is i or not, distinguishing class i from all other classes (Algorithm 2).

Algorithm 2: OAA-TRAIN (set of k-class training examples S, binary classifier learning algorithm A)

```
Set S' = ∅
for all (x, y) ∈ S do
    for all i ∈ {1, ..., k} do
        add a binary example (⟨x, i⟩, 1(y = i)) to S'
return h = A(S').
```

Predictions are done by evaluating each binary classifier and randomizing over those which predict "yes," or over all k labels if all answers are "no" (Algorithm 3).

Algorithm 3: OAA-TEST (binary classifier h, test example x)

output $\arg\max_i h(\langle x,i\rangle)$ for $i \in \{1,\ldots,k\}$, breaking ties randomly

Notice that we do not actually have to learn k separate classifiers in Algorithm 2. We can simply augment the feature space with the index of the classifier and then learn a single combined classifier on the union of all training data. The implication of this observation is that we can view the reduction as a machine that maps multiclass examples to binary examples, transforming the original multiclass distribution D into an *induced* binary distribution D'. To draw a sample from D', we simply draw a multiclass example (x,y) from D and a random index $i \in \{1,\ldots,n\}$, and output $(\langle x,i\rangle, \mathbf{1}(y=i))$.

The lemma below bounds the error rate of Algorithm 3 on D in terms of the error rate of h on D'. Such a statement is called an *error transformation* of a reduction.

Lemma 1. (One-against-all error efficiency [2, 20, 8]) *For all k-class distributions D and binary classifiers h, $e(\mathrm{OAA}_h, D) \leq (k-1)e(h,D')$, where OAA_h is the multiclass classifier produced by* OAA *using h.*

Proof. We analyze how false negatives and false positives produced by the binary classifier lead to errors in the multiclass classifier. A false negative produces an error in the multiclass classifier a $\frac{k-1}{k}$ fraction of the time (assuming that all the other classifiers are correctly outputting 0), because we are choosing randomly between k labels and only one is correct. The other error modes to consider involve (possibly multiple) false positives. If there are m false positives, the error probability is either $\frac{m}{m+1}$ or 1 if there is also a false negative. The efficiency of these three modes in creating errors (i.e., the maximum ratio of the probability of a multiclass error to the fraction of binary errors) is $k-1$, $\frac{k}{m+1}$, and $\frac{k}{m+1}$, respectively. Taking the maximum, $k-1$, we get the result. ∎

The proof actually shows that the multiclass error can be as high as $(k-1)e(h,D')$. By noticing that a false negative is more disastrous than a false positive, we can put more importance on positive examples, first reducing the multiclass problem to an importance-weighted binary problem, and then composing this reduction with the Costing reduction from Section 3 to remove the importances. As shown in [8], doing this halves the worst-case error theoretically, and improves over OAA empirically. This gives an example of a practical improvement directly influenced by analysis.

Inconsistency and Regret Transforms

There is an essential problem with OAA.

Definition 3. A reduction is said to be *inconsistent* if for some distribution D, the reduction does not produce $\arg\min_f e(f,D)$ given an optimal base predictor $\arg\min_h e(h,D')$ for the induced distribution D'.

Consistency is a very natural and desirable property of any reduction. If the auxiliary problems are solved optimally, the reduction should in turn yield an optimal predictor for the original problem. Unfortunately, OAA is inconsistent.

Theorem 2. *The One-Against-All reduction is inconsistent.*

The proof is simple and illustrative. A similar theorem statement and proof applies to ECOC reductions [13].

Proof. Consider a distribution D which puts all the probability mass on some example x. Let $k = 3$ and define the conditional probability as $D(y = 1 \mid x) = 0.5 - \gamma$ and $D(y = 2 \mid x) = D(y = 3 \mid x) = 0.25 + \frac{\gamma}{2}$, for some $0 < \gamma < 1/4$. Thus there is no majority class occurring at least half the time. OAA creates a binary problem for each class $y \in \{1,2,3\}$, distinguishing y (binary label 1) from the two remaining classes (binary label 0). The probability of label 1 for the three binary problems is $0.5 - \gamma$, $0.25 + \frac{\gamma}{2}$, and $0.25 + \frac{\gamma}{2}$ respectively, which implies that the optimal prediction is 0 for each binary problem. When every binary predictor predicts 0, the induced multiclass predictor can't do better than randomize among the three possible classes, resulting in an error rate of $\frac{2}{3}(0.5 - \gamma) + \frac{4}{3}(0.25 + \frac{\gamma}{2}) = \frac{2}{3}$. However, the optimal multiclass predictor always chooses class 1 achieving the error rate of $0.55 < 2/3$. ∎

Practical implications of the inconsistency theorem 2 are known and understood by practitioners. It is one of the reasons why practical implementations of one-against-all reductions use internal prediction quantities such as the margin of a predictor rather than the actual binary classification. A basic question is whether such soft versions of OAA work. Some do and some don't. Ideal soft quantities are good class probability estimates, but for many classifiers, these estimates are very poor. For example, the efficacy of Platt scaling [35] (i.e., fitting a sigmoid to a margin to get a probabilistic prediction) can be thought of as strong empirical evidence of the deficiency of margins as probability estimates. Rifkin and Klautau [38] argue that the soft version OAA can be made to work as well as other techniques, if some effort is put into optimizing the binary classifiers.

Regret reductions The fundamental noise in the distribution D may make optimal performance not be equivalent to zero error rate, motivating the notion of regret.

Definition 4. The *regret* of a classifier h on distribution D is defined as

$$r(h,D) = e(h,D) - \min_{h^*} e(h^*,D),$$

where the minimum is taken over all classifiers h^* (of the same type as h).

Thus *regret* is the difference between the incurred loss and the lowest achievable loss on the same problem. Regret can be defined with respect to any loss function. A statement showing how the regret of the base classifier on the induced problem controls the regret of the resulting predictor on the original problem is called a *regret transformation* of the reduction. Regret statements are more desirable than statements relating error rates, because regret analysis separates excess loss from the unavoidable noise in the problem, making the statement nontrivial even for noisy problems. For example, if the binary error rate is 10% due to noise and 5% due to the errors made by the classifier, then the multiclass error rate depends only on the 5%. Any reduction with a multiplicative regret transform is consistent.

4.2 Error-Correcting Coding (ECOC) Approaches

Another obvious problem with the OAA reduction is the lack of robustness. If just one out of k binary classifiers errs, the multiclass classifier errs. This section presents approaches based on error correcting codes, where we can even have a constant fraction of classifiers err (independent of k) without mispredicting the multiclass label.

The ECOC approach, popularized by Dietterich and Bakiri [13], learns binary classifiers for deciding membership in different subsets of the labels. The reduction can be specified by a binary-valued matrix C with n rows and k columns, where n is the number of binary problems created by the reduction. Each row i in the matrix defines a binary classification problem: predict $C(i,y)$, where y is the correct label given input x. Given C, each label corresponds to a binary string defined by the inclusion of this label in the sequence of subsets.

For two n-bit binary vectors, the *Hamming distance* between them is the number of bit positions on which they differ. A multiclass prediction is made by finding the codeword closest in Hamming distance to the sequence of binary predictions on the test example.

For the ECOC reduction, a basic statement can be made [20]: with a good code, the error rate of the multiclass classifier is at most 4 times the average error rate of the individual binary classifiers. The proof of this statement is essentially the observation that there exist codes in which the distance between any two codewords is at least $\frac{1}{2}$. Consequently, at least $\frac{1}{4}$ of the classifiers must err to induce a multiclass classification error, implying the theorem. In general, if d is the smallest distance between any pair of columns in C, the loss rate for this reduction is at most $2n\varepsilon/d$, where ε is the average loss rate of the n binary classifiers.

We mention several coding matrices of particular interest. The first is when the columns form a subset of the columns of a Hadamard matrix, an $n \times n$ binary matrix with any two columns differing in exactly $n/2$ places. Such matrices are easy to construct recursively when $n = 2^m$ is a power of 2:

$$C_1 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \qquad C_{m+1} = \begin{pmatrix} C_m & C_m \\ C_m & \overline{C}_m \end{pmatrix},$$

where $\overline{C}_m$ is C_m with 0s and 1s exchanged. It is clear from the construction that all columns are at the same distance 2^m. Notice that the distance property is preserved when any column or row is complemented. We will use this code in our constructions. Thus, for Hadamard codes, the number of classifiers needed is less than $2k$ (since a power of 2 exists between k and $2k$), and the loss rate is at most 4ε.

If the codewords form the $k \times k$ identity matrix, the ECOC reduction corresponds to the one-against-all reduction.

As OAA, ECOC was modified [2] to consider margins of the binary classifiers, numbers internal to some classification algorithms that provide a measure of confidence in a binary prediction. Decoding proceeds in the same way as for ECOC except a loss-based distance is used instead of the Hamming distance, where the loss is defined by the optimization function used internally by the binary learning algorithm. Instead of working with margins, we define binary classification problems for which the optimal solution computes the relative expected cost (rather than the margin) of choices. This approach can be applied to arbitrary classifier learners rather than margin-based learners. Finally, we can generalize the approach to tackle all cost-sensitive problems rather than just multiclass problems (see Section 5). While we talk about predefined, data-independent output codes here, there are methods that *learn* coding matrices based on training data [12].

Probabilistic ECOC

Unfortunately, the ECOC reduction is inconsistent [26]. We describe a probabilistic variant of ECOC [26], which deals with the inconsistency problem by reducing to squared loss regression rather than binary classification. (The same approach can be applied to OAA, but the resulting solution would still not be robust.) If a reduction to binary classification is desired, this approach can be composed with the Probing reduction [28] from squared loss regression to binary classification.

As ECOC, the PECOC reduction is defined by a binary coding matrix C with k columns corresponding to multiclass labels and n rows corresponding to prediction problems. For each row, we form a squared-loss regression problem to predict the probability that the label is in one subset or the other. We write $\mathbf{E}_i$ to denote an expectation over i drawn uniformly from the rows of C.

There are two ways to use the reduction: either for estimating the probability of a class label as in Algorithm 5 or for hard prediction of a class label as in Algorithm 6.

PECOC Theorem

As in earlier reductions, Algorithm 4 transforms the original multiclass distribution D into an induced distribution D' over real-valued examples. As in OAA, we can

Algorithm 4: PECOC-Train (set of k-class multiclass examples S, squared loss regressor B)

for *each subset i defined by the rows of C* **do**
  Let $S_i = \{(x, C(i,y)) : (x,y) \in S\}$
  Let $b_i = \mathrm{B}(S_i)$.
return $\{b_i\}$

Algorithm 5: PECOC-Predict (classifiers $\{b_i\}$, example $x \in X$, label y whose probability we want to predict)

return $\mathrm{ppecoc}_b(x,y) = 2\mathbf{E}_i\left[C(i,y)b_i(x) + (1-C(i,y))(1-b_i(x))\right] - 1$

Algorithm 6: PECOC-Hard-Predict (classifiers $\{b_i\}$, example $x \in X$)

return $\mathrm{pecoc}_b(x) = \arg\max_{y \in \{1,\ldots,k\}} \mathrm{PECOC\text{-}Predict}(\{b_i\}, x, y)$

assume that we have a single combined regressor $b(x,i) = b_i(x)$ by augmenting the feature space with the index of the classifier. The theorem below is quantified for all regressors, including b learned in Algorithm 4.

Theorem 3. (PECOC Regret Transform [26]) *For any k-class distribution D, any regressors b, and any label* $y \in \{1,\ldots,k\}$

$$\mathbf{E}_{x\sim D}\left(\mathrm{ppecoc}_b(x,y) - D(y|x)\right)^2 \leq 4r(D',b),$$

where ppecoc_b *is as defined in Algorithm 5.*

This theorem relates the average regret of the created regressors to the relative estimation error.

Proof. We first analyze what happens when no regret is suffered, and then analyze the case with regret. For any i, let $D(i)$ be the distribution on $X \times \{0,1\}$ induced by drawing (x,y) from D and outputting $(x, C(i,y))$. For any choice of i, the optimal squared loss regressor is given by

$$\begin{aligned} b_i^* &= \arg\min_b \mathbf{E}_{(x,y')\sim D(i)}\left[(b(x)-y')^2\right] \\ &= \arg\min_b \mathbf{E}_{(x,y')\sim D(i)}\left[b(x)^2\right] - 2\mathbf{E}_{(x,y')\sim D(i)}\left[y'b(x)\right] \\ &= \arg\min_{x,b}\left(b(x)^2 - 2\mathbf{E}_{y'\sim D(i|x)}\left[y'b(x)\right]\right). \end{aligned}$$

For each x, the optimal value of $b(x)$ can be found by taking the derivative and setting it equal to zero, because squared loss is convex. This yields $b(x) = \mathbf{E}_{y'\sim D(i|x)}\left[y'\right]$ which can also be written as

$$b_i(x) = \mathbf{Pr}_{y\sim D|x}[C(i,y) = 1].$$

Since decoding is symmetric with respect to all labels, we need analyze only one label y. Furthermore, since complementing all subsets not containing y does not change the decoding properties of the code, we can assume that y is in every subset. Consequently,

$$\mathbf{E}_i\left[b_i^*(x)\right] = \mathbf{E}_i\mathbf{Pr}_{y'\sim D|x}[C(i,y') = 1] = \frac{1}{2}(D(y|x)+1),$$

where the third equality follows from the fact that every label y' other than y appears in i half the time, in expectation over i. Consequently, $\text{ppecoc}_b(x,y) = D(y|x)$ for each x and y, when the classifiers are optimal.

Now we analyze the regret transformation properties. The remainder of this proof characterizes the most efficient way that any adversary can induce estimation regret with a fixed budget of squared loss regret.

First, notice that PECOC-predict (Algorithm 5) is linear in the base predictions $b_i(x)$, but the squared loss regret of bad predictions is quadratic, according to $(b_i(x) - b_i^*(x))^2$. Consequently, it is cheapest for the adversary to have a small equal disturbance for each i rather than a large disturbance for a single i. (The cost any adversary pays for disturbing the overall expectation can be monotonically decreased by spreading errors uniformly over subsets i.) Thus the optimal strategy for an adversary wanting to disturb the output of PECOC-Predict by Δ is to disturb the expectation for each i by $\Delta/2$. The regret of the base regressors is given by $\Delta^2/4$, implying the theorem. ∎

Hard Prediction

We can use Algorithm 6 to make hard multiclass predictions. For this special case, a simple corollary of the soft prediction analysis holds.

Corollary 2. (Multiclass Classification Regret Transform) *For any k-class distribution D and any regressor b,*

$$r(D, \text{pecoc}_b(x)) \leq 4\sqrt{r(D', b)},$$

where pecoc_b *is defined as in Algorithm 6.*

Proof. The regret of a multiclass prediction is proportional to the difference in probability of the best prediction and the prediction made. Weakening Theorem 3 gives, for all y,

$$\mathbf{E}_{(x,y)\sim D}\left|\text{ppecoc}_b(x,y) - D(y|x)\right| \leq 2\sqrt{r(D', b)},$$

since for all Z, $\sqrt{E(Z)} \geq E\sqrt{Z}$. When doing a hard prediction according to these outputs, our regret at most doubles because the probability estimate of the correct

class can be reduced by the same amount that the probability estimate of the wrong class increases. ∎

As shown in [26], PECOC consistently performs better (or as well) as ECOC and OAA, across different datasets and base learners.

4.3 Approaches Based on Pairwise Comparisons

Another class of multiclass to binary reductions are based on comparing only pairs of classes [21, 36, 5, 6, 7].

The All-Pairs reduction [21] starts by constructing $\binom{k}{2}$ binary classifiers, one for every pair of classes. Given a training dataset $S = \{(x,y)\}$, the binary classifier for the (i,j)-class pair is trained with dataset $\{(x, \mathbf{1}(y=i)) : (x,y) \in S \text{ and } y=i \text{ or } y=j\}$ to discriminate between classes i and j. Given a test example, each of the binary classifiers predicts a winner amongst its two classes, and the class with the highest number of wins is chosen as the multiclass prediction, with ties broken randomly.

The All-Pairs reduction *is* consistent (as an application of a theorem in [2]). It is frail theoretically, but it works fairly well in practice. Platt, Cristianini, and Shawe-Taylor [36] proposed a DAG method which is identical to All-pairs at training time. At test time, the labels play a sequential single-elimination tournament, requiring only $k-1$ classifier evaluations instead of $k(k-1)/2$, making the approach substantially faster. Error-correcting tournament approaches [7] perform multiple-elimination tournaments over the labels, yielding robust regret transforms, independent of k. A computational advantage of pairwise reductions, which is especially important in applications that involve massive amounts of data, is that individual classifiers are run on datasets that are smaller than the original dataset (unlike OAA and ECOC approaches).

5 Cost-Sensitive Classification

This section presents a reduction from cost-sensitive classification to binary classification. Cost-sensitive k-class classification is just like k-class classification, only each prediction now has an associated cost.

Definition 5. A *cost-sensitive* k-class classification problem is defined by a distribution D over $X \times [0,\infty)^k$. The goal is to find a classifier $h : X \to \{1,...,k\}$ minimizing the expected cost $e(h,D) = \mathbf{E}_{(x,c)\sim D}\left[c_{h(x)}\right]$. Here $c \in [0,\infty)^k$ gives the cost of each of the k choices for x.

Since cost-sensitive classification can express any bounded-loss finite-choice supervised learning task, the reduction shows that *any* such task can be solved using a binary classification oracle.

We present a reduction, Weighted All-Pairs (WAP) [5], which reduces k-class cost-sensitive classification to importance weighted binary classification. It can be composed with the Costing reduction from Section 3 to yield a reduction to binary classification.

Weighted All Pairs: The Algorithm

WAP is a weighted version of the All-Pairs reduction [21] discussed above. The algorithms specifying the reduction are given below.

Algorithm 7: WAP-Train (set of k-class cost sensitive examples S, importance weighed binary classifier learning algorithm B)

Set $S' = \emptyset$
for *all examples* $(x, c_1, \ldots, c_k) \in S$ **do**
  for *all pairs* (i, j) *with* $1 \leq i < j \leq k$ **do**
    Add an importance weighted example $(\langle x, i, j \rangle, I(c_i < c_j), |v_j - v_i|)$ to S'.
return $h = B(S')$

The values v_i used by the reduction are defined as follows: For a given cost-sensitive example $(x, c_1, \ldots, c_k)$, let $L(t)$ be the function $L(t) = |\{j \mid c_j \leq t\}|$, for $t \in [0, \infty)$. By shifting, we may assume that the minimum cost is 0, so that $t \geq 0$ implies $L(t) \geq 1$. Optimizing the loss of the shifted problem is equivalent to optimizing the loss of the original problem. The values v_i are defined as $v_i = \int_0^{c_i} 1/L(t)dt$. Note that the values are order-preserving: $c_i < c_j$ iff $v_i < v_j$ for all i and j.

We say that label i *beats* label j for input x if either $i < j$ and $h(\langle x, i, j \rangle) = 1$, or $i > j$ and $h(\langle x, j, i \rangle) = 0$.

Algorithm 8: WAP-Test (classifier h, example x)

for *all pairs* (i, j) *with* $1 \leq i < j \leq k$ **do**
  Evaluate $h(\langle x, i, j \rangle)$
output $\text{wap}_h(x) = \text{argmax}_i |\{j \mid i \text{ beats } j\}|$

Note that if h makes no errors and $c_i \neq c_j$, then label i beats label j exactly when $c_i < c_j$. WAP-Test outputs the label which beats the maximum number of other labels, with ties broken arbitrarily.

Before stating the error transform, we define the importance-weighted binary distribution D' induced by the reduction. To draw a sample from this distribution, we first draw a cost sensitive sample $(x, c_1, \ldots, c_k)$ from the input distribution D and

then apply WAP-Train to the singleton set $\{(x, c_1, \ldots, c_k)\}$ to get a sequence of $\binom{r}{2}$ examples for the binary classifier. Now we just sample uniformly from this set.

Theorem 4. (WAP error efficiency [5]) *For any cost-sensitive distribution D and any importance-weighted classifier h,*

$$e(\mathrm{wap}_h, D) \leq 2e(h, D'),$$

where wap_h *is defined in Algorithm 8.*

This theorem states that the cost sensitive loss is bounded by twice the importance weighted loss on the induced importance weighted learning problem.

For notational simplicity, assume that $c_1 \leq \cdots \leq c_k$. Note that no generality is lost since the algorithm does not distinguish between the labels. The following lemma (from [5]) is the key to the proof.

Lemma 2. *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. Consider the restricted tournament on $\{1, \ldots, j\}$.
Case 1: Suppose that some w beats at least $\lceil j/2 \rceil$ of the others. If no label $b > j$ beats any label $a \leq j$, then w would beat at least $\lceil j/2 \rceil + 1$ more labels than any $b > j$; in particular, w would beat at least $\lceil j/2 \rceil + 1$ more labels than i. Thus, in order to have label i beat as many labels as w, at least $\lceil j/2 \rceil$ edges of the form $(w, b), b > j$ or $(a, i), a \leq j$ must be reversed.
Case 2: There is no label $w \in \{1, \ldots, j\}$ beating $\lceil j/2 \rceil$ of the rest of $\{1, \ldots, j\}$. This can only happen if j is odd and there is 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 $b > j$, in particular i, it is still necessary to reverse at least $(j+1)/2 \geq \lceil j/2 \rceil$ edges, in order to ensure that $i > j$ beats as many labels as each of $\{1, \ldots, j\}$. ∎

Proof. (Theorem 4) Suppose that our algorithm chooses the wrong label i for a particular example $(x, c_1, \ldots, c_k)$. We show that this requires the adversary to incur a comparable loss.

Lemma 2 and the definition of v_i imply that the penalty incurred to make label i win is at least

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

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

$$\begin{aligned} \sum_{i<j} v_j - v_i &= \sum_{i<j} \int_{c_i}^{c_j} \frac{1}{L(t)} \mathrm{d}t = \int_0^{c_k} \frac{L(t)R(t)}{L(t)} \mathrm{d}t \\ &= \int_0^{c_k} R(t) \mathrm{d}t = \sum_{i=1}^{k} \int_0^{c_i} \mathrm{d}t = \sum_{i=1}^{k} c_i, \end{aligned}$$

where $R(t) = k - L(t)$ 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 times a pair (i, j) satisfies $i < t < j$. The second to last equality follows by writing $R(t)$ as a sum of the k indicator functions for the events $\{c_j > t\}$, and then switching the order of summation.

Consequently, for every example $(x, c_1, \ldots, c_k)$, the total importance assigned to queries for x equals $\sum_i c_i$, 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 is at most 2. ∎

This method of assigning importances is provably near-optimal, as shown in [5].

Theorem 5. (Lower bound) *For any other assignments of importances $w_{i,j}$ to the points (x, i, j) in the above algorithm, there exists a distribution with expected cost $\varepsilon/2$.*

Proof. Consider examples $\left(x, 0, \frac{1}{r-1}, \ldots, \frac{1}{k-1}\right)$. Suppose that 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 the total cost of 1. The importance of these two errors is $w_{1,i} + w_{1,j}$, out of the total importance of $\sum_{i,j} w_{i,j}$. Choosing i and j so that $w_{1,i} + w_{1,j}$ is minimal, the adversary's penalty is at most $2\sum_{i=2}^{k} w_{1,i}/(k-1)$, and hence less than $2/(k-1)$ 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. ∎

The PECOC reduction from Section 4.2 can be extended to cost-sensitive classification [26]. Tournaments reductions [6, 7] give the tightest known analysis, with dependence on the expected sum of cost differences instead of the sum of costs as in the cost-sensitive variant of PECOC and WAP.

6 Predicting Conditional Quantiles

Recall the definition of a regression problem.

Definition 6. A *(least squares) regression* problem is defined by a distribution D over $X \times Y$, where $Y = \mathbb{R}$. The goal is to find a *predictor* $f : X \to Y$ minimizing the expected *squared error loss*

$$e(f, D) = \mathbf{E}_{(x,y)\sim D}\left[(y - f(x))^2\right].$$

One standard justification for this choice is that the minimizer $f^* = \arg\min_f e(f, D)$ is the conditional mean (conditioned on x): $f^*(x) = \mathbf{E}_{y\sim D|x}[y]$. However, there are many important applications for which mean estimates are either irrelevant or insufficient, and *quantiles* (also known as general order statistics) are the main quantities of interest. For instance, consider trying to assess the risk of a business proposal.

Estimates of the lower quantiles of the conditional return distribution would give a better indication of how worthwhile the proposal is than a simple estimate of the mean return (which could be too high because of very unlikely high profits).

The process of estimating the quantiles of a conditional distribution is known as *quantile regression*. More specifically, the goal of quantile regression is to obtain estimates of the q-quantiles of the conditional distribution $D|x$. Intuitively, q-quantiles for different q describe different segments of the conditional distribution $D|x$ and thus offer more refined information about the data at hand.

Definition 7. (Conditional q-quantile, $0 \leq q \leq 1$). The *conditional q-quantile* (or *conditional q-order statistic*) for a distribution D is a function $f = f(x)$ such that for every $x \in X$, $D(y \leq f(x) \mid x) \geq q$ and $D(y \geq f(x) \mid x) \geq 1-q$. The $1/2$-quantile is also known as the *median*.

Note that the q-quantile may not be unique when the conditional distribution has regions with zero mass.

It is well-known that the optimal estimator for the absolute-error loss is the median [25]. In other words, for every distribution D over $X \times \mathbb{R}$,

$$\arg\min_f \mathbf{E}_{(x,y)\sim D}|y - f(x)| \text{ is the (conditional) median.}$$

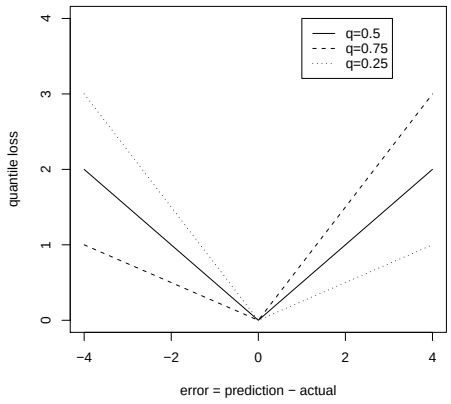

Fig. 1 Loss functions which induce quantile regression.

The generalization of absolute-error loss for arbitrary order statistics is the quantile loss function, also known as the pinball loss [40]. Pictorially, this is a tilted absolute loss as in figure 1. Mathematically, this is $\mathbf{E}_{(x,y)\sim D}\ell_q(y, f(x))$, where

$$\ell_q(y, f(x)) = q(y - f(x))\mathbf{1}(y \geq f(x)) + (1-q)(f(x) - y)\mathbf{1}(y < f(x)),$$

where $\mathbf{1}(\cdot) = 1$ if its argument is true and 0 otherwise.

Definition 8. A *quantile regression* problem is defined by a distribution D over $X \times Y$, where $Y = \mathbb{R}$, and $0 \leq q \leq 1$. The goal is to find a *predictor* $f : X \to Y$ minimizing $\mathbf{E}_{(x,y)\sim D}\ell_q(y, f(x))$.

The Quanting Reduction

We show that the quantile regression problem can be reduced to standard binary classification via an algorithm called Quanting [27]. For any $q \in [0,1]$, the Quanting algorithm estimates the q-th quantile of the conditional distribution $D|x$ using any importance weighted classification algorithm A. Using the costing reduction from Section 3, we can also do Quanting via any binary classification algorithm.

The Quanting algorithm has two parts. Given a set of training examples S of the form (x,y) with $y \in [0,1]$ (we can always renormalize S so that all labels are in $[0,1]$), Algorithm 9 uses any given importance-weighted binary classification algorithm A to learn a family of classifiers, parameterized by a threshold $t \in [0,1]$. Each classifier h_t attempts to answer the question "Is the q-quantile above or below t?" To train each h_t, Algorithm 9 creates an importance-weighted training set by adding a weight w to each example in S. Positive examples with $y \geq t$ receive weight q, while negative examples with $y < t$ receive weight $(1-q)$.

Algorithm 9: Quanting-train (importance-weighted classifier learning algorithm A, training set S, quantile q)

for $t \in [0,1]$ **do**
  $S_t = \emptyset$
  for *each* $(x,y) \in S$ **do**
    $S_t = S_t \cup \{(x, \mathbf{1}(y \geq t), q \cdot \mathbf{1}(y \geq t) + (1-q)\mathbf{1}(y < t))\}$
  $h_t = A(S_t)$
return the set of classifiers $\{h_t\}$

In reality, of course, one cannot find a different classifier for each $t \in [0,1]$. Constructing classifiers h_t for t in a discrete mesh $\{0, 1/n, 2/n, \ldots, (n-1)/n, 1\}$ will add a $1/n$ term to the error bound.

Using the learned classifiers, Algorithm 10 produces a prediction of the q-quantile for each x in a given test set S'.

Algorithm 10: Quanting-test (set of classifiers $\{h_t\}$, test set S')

for *each* $x \in S'$ **do**
  $Q(x) = \mathbf{E}_{t\sim U(0,1)}[h_t(x)]$

In the (idealized) scenario where A is perfect, one would have $h_t(x) = 1$ if and only if $t \leq q(x)$ for a q-quantile $q(x)$, hence Algorithm 10 would output $\int_0^{q(x)} dt = q(x)$ exactly. The analysis below shows that if the error of A is small on average over t, the quantile estimate is accurate.

Quanting Reduction Analysis

The lemma we prove next relates the average regret of the classifiers h_t (how well the classifiers do in comparison to how well they could do) to the regret of the quantile loss incurred by the Quanting algorithm. For each x, the output produced by the quanting algorithm is denoted by $Q(x)$, whereas $q(x)$ is a correct q-quantile. In this analysis, we use the standard one-classifier trick [28]: instead of learning different classifiers, we learn one classifier $h = \{h_t\}$ with an extra feature t used to index classifier h_t.

The quanting reduction induces an importance weighted binary classification problem, which we denote as D'.

Lemma 3. (Quanting Regret Transform [27]) *For all distribution D over $X \times \mathbb{R}$, and all classifiers h,*

$$\mathbf{E}_{(x,y)\sim D}[\ell_q(y, Q(x))] - \mathbf{E}_{(x,y)\sim D}[\ell_q(y, q(x))] \leq e(D', h) - \min_{h'} e(D', h')$$

where $e(D', h)$ is the expected importance weighted binary loss of h on the induced distribution D'.

Proof. For any function $f = f(x)$, $\mathbf{E}_{(x,y)\sim D}[\ell_q(y, f(x))]$ is given by eqn. (6):

$$q\mathbf{E}_{(x,y)\sim D}[(y - f(x))\mathbf{1}(y - f(x) > 0)] + (1-q)\mathbf{E}_{(x,y)\sim D}[(f(x) - y)\mathbf{1}(f(x) - y > 0)].$$

It is known that $\mathbf{E}[X \cdot \mathbf{1}(X > 0)] = \int_0^{+\infty} \mathbf{Pr}(X \geq t)dt = \int_0^{+\infty} \mathbf{Pr}(X > t)dt$ for any random variable X, so we rewrite

$$\begin{aligned}\mathbf{E}_{(x,y)\sim D}[\ell_q(y, f(x))] &= q\mathbf{E}_x \int_0^{\infty} D(y - f(x) \geq t_1 | x)dt_1 + (1-q)\mathbf{E}_x \int_0^{\infty} D(f(x) - y > t_2 | x)dt_2 \\ &= q\mathbf{E}_x \int_{f(x)}^{1} D(y \geq u | x)\, du + (1-q)\mathbf{E}_x \int_0^{f(x)} D(y < u | x)\, du.\end{aligned}$$

Applying this formula to $f(x) = Q(x)$ and $f(x) = q(x)$ and taking the difference yields

$$\begin{aligned}
&\mathbf{E}_{(x,y)\sim D}[\ell_q(y,Q(x)) - \ell_q(y,q(x))] \\
&\quad = \mathbf{E}_x \int_{Q(x)}^{q(x)} [qD(y \geq u|x) - (1-q)D(y < u|x)]\, du \\
&\quad = \mathbf{E}_x \int_{Q(x)}^{q(x)} [q - qD(y < u|x) - (1-q)D(y < u|x)]\, du \\
&\quad = \mathbf{E}_x \int_{Q(x)}^{q(x)} [q - D(y < u|x)]\, du. \qquad (1)
\end{aligned}$$

We will show that $e(D',h) - \min_{h'} e(D',h')$ is at least this last expression. The expected importance-weighted error incurred by the classifiers $\{h_t\}$ is

$$\begin{aligned}
e(D',h) &= \mathbf{E}_{(x,y)\sim D} \int_0^1 \begin{bmatrix} q\mathbf{1}(y \geq t)(1-h_t(x)) \\ +(1-q)\mathbf{1}(y < t)h_t(x) \end{bmatrix} dt \\
&= \mathbf{E}_x \int_0^1 \begin{bmatrix} qD(y \geq t|x) \\ +(D(y < t|x) - q)h_t(x) \end{bmatrix} dt \\
&= q\mathbf{E}_x[y] + \mathbf{E}_x \int_0^1 [D(y < t|x) - q]h_t(x)\, dt \qquad (2) \\
&\geq q\mathbf{E}_x[y] + \mathbf{E}_x \int_0^{Q(x)} [D(y < t|x) - q] dt. \qquad (3)
\end{aligned}$$

Here only the last line is non-trivial, and it follows from the fact that $D(y < t|x) - q$ is increasing in t. Thus the smallest possible value for the integral in (2) is achieved by placing as much "weight" $h_t(x)$ as possible on the smallest t while respecting the constraints $\int_0^1 h_t(x)\, dt = Q(x)$ and $0 \leq h_t(x) \leq 1$. This corresponds precisely to setting $h_t(x) = \mathbf{1}(t \leq Q(x))$, from which (3) follows.

Inequality (3) is in fact an equality when instead of $\{h_t\}$ we use the (optimal) classifiers

$$\{h_t^*(x) = \mathbf{1}(D(y \leq t|x) \leq q)\}$$

and substitute $q(x)$ for $Q(x)$. Therefore,

$$\begin{aligned}
e(D',h) - e(D',h^*) \geq \quad & \mathbf{E}_x \int_{q(x)}^{Q(x)} [D(y < t|x) - q] dt \\
&= \mathbf{E}_{(x,y)\sim D}[\ell_q(y,Q(x)) - \ell_q(y,q(x))],
\end{aligned}$$

using (1). This finishes the proof. ∎

We now show how to reduce q-quantile estimation to unweighted binary classification using the results in Section 3. To apply *rejection sampling*, we feed a binary classifier learner samples of the form $((x,t), \mathbf{1}(y \geq t))$, each of the samples being independently discarded with probability $1 - w(\mathbf{1}(y \geq t))$, where $w(b) = qb + (1-q)(1-b)$ is the example's weight.

Corollary 3. (Quanting to Binary Regret [27]) *For any D as above and any binary classifier $g = \{g_t\}$,*

$$\mathbf{E}_{(x,y)\sim D}[\ell_q(y,Q(x))] - \mathbf{E}_{(x,y)\sim D}[\ell_q(y,q(x))] \leq e(\tilde{D},g) - \min_{g'} e(\tilde{D},g'),$$

where $\tilde{D}$ is the distribution produced by rejection sampling.

Proof. Let $h = \{h_t\}$ be the importance-weighted classifier induced by the rejection sampling reduction. Theorem 1 implies that

$$e(D,h) - \min_{h'} e(D,h') = e(\tilde{D},h) - \min_{g'} e(\tilde{D},g')$$

and the result follows from Lemma 3. ∎

Experimental results: The Quanting reduction compares favorably with two existing direct methods for quantile regression: linear quantile regression [25] and kernel quantile regression [40]. See [27] for details on the performed experiments.

7 Ranking

Finally, we consider the problem of ranking a set of instances. In the most basic version, we are given a set of unlabeled instances belonging to two classes, 0 and 1, and the goal is to rank all instances from class 0 before any instance from class 1. A common measure of success for a ranking algorithm is the area under the ROC curve (AUC). The associated loss, $1 - \text{AUC}$, measures how many pairs of neighboring instances would have to be swapped to repair the ranking, normalized by the number of 0s times the number of 1s. The loss is zero precisely when all 0s precede all 1s; one when all 1s precede all 0s. It is greater for mistakes at the beginning and the end of an ordering, which satisfies the intuition that an unwanted item placed at the top of a recommendation list should have a higher associated loss than when placed in the middle.

At first, this problem appears very different from binary classification. A misclassified instance in classification incurs the same loss independently of how other instances are classified, while the AUC loss depends on the whole (ranked) sequence of instances. However, it turns out that we don't need different algorithms to optimize these two loss functions: there is a robust mechanism for translating any binary classifier learning algorithm into a ranking algorithm.

Balcan et al. [4] present a simple deterministic reduction with a guarantee that any binary classifier with regret r on the induced problem implies AUC regret at most $2r$, for arbitrary distributions over instances. This is the best possible with any deterministic algorithm. This is also a large improvement over naive approaches such as ordering according to regressed scores.

In a subsequent paper, Ailon and Mohri [1] describe a randomized quick-sort reduction, which guarantees that AUC loss is bounded by binary loss, in expectation over the randomness of the algorithm. When there are more than two labels, the expected generalized AUC loss is bounded by twice the binary loss. The quick-sort algorithm is quick efficient, requiring only $O(n \log n)$ classifier evaluations at test time, which makes it practical in larger settings.

8 Conclusion

There are certain key concepts that we want to emphasize for readers interested in studying or building further reductions.

1. Regret vs. Error Reductions. An error reduction simply states that the error rate on an induced problem bounds the error rate on the original problem. While an error reduction might be a good first-pass approach, it has certain undesirable properties which are removed by a regret reduction. For example, all regret reductions are necessarily consistent.
2. Prediction Minimality. Embedded in the logic of reductions is a preference for systems which don't make unnecessary ancillary predictions. If there is a core set of n predictions to make, adding an unnecessary extra prediction always makes the regret bound worse by a factor of $(n+1)/n$.
3. Importance weighting. Many reductions use importance weighting of some sort to carefully control how much they care about one prediction versus another. Mastering the use of importance weighting is essential.
4. Thresholding. When a continuous parameter needs to be predicted as with the Quanting reduction, setting up a continuous family of classification problems appears necessary.
5. Orthogonal Prediction. The PECOC analysis relies deeply on the ability to setup orthogonal prediction problems which happen to cancel out in just the right way to achieve good performance.

This tutorial has covered a number of different methods for reducing general learning problems to core problems, including essentially all supervised learning problems. There are at least three directions of future progress:

1. Extending the scope of learning reductions to new learning problems.
2. Improving existing reductions.
3. Shifting the foundations. Existing reductions theory finds a happy medium between the provable, practical, and useful, but there is no proof that it is canonical. A reexamination of the foundations may yield new directions of research.

Learning reductions are an effective tool for designing automated solutions to learning problems. They also tell us something about the organization of learning problems and have a remarkably clean analysis. Reductions are a basic tool which make a handy component in a tool-chest of solutions.

References

1. N. Ailon and M. Mohri (2007) An Efficient Reduction of Ranking to Classification, *New York University Technical Report*, TR-2007-903.
2. E. Allwein, R. Schapire, and Y. Singer (2000) Reducing multiclass to binary: A unifying approach for margin classifiers, *Journal of Machine Learning Research*, 1:113–141.

3. A. Asuncion, D. Newman (2007) UCI Machine Learning Repository, http://mlearn.ics.uci.edu/MLRepository.html, University of California, Irvine.
4. N. Balcan, N. Bansal, A. Beygelzimer, D. Coppersmith, J. Langford, and G. Sorkin (2007) Robust reductions from ranking to classification, *Proceedings of the 20th Annual Conference on Learning Theory* (COLT), Lecture Notes in Computer Science 4539: 604–619.
5. A. Beygelzimer, V. Dani, T. Hayes, J. Langford, and B. Zadrozny (2005) Error limiting reductions between classification tasks, *Proceedings of the 22nd International Conference on Machine Learning* (ICML), 49–56.
6. A. Beygelzimer, J. Langford, and P. Ravikumar (2008) Filter trees for cost sensitive multiclass classification.
7. A. Beygelzimer, J. Langford, and P. Ravikumar (2008) Error Correcting Tournaments.
8. A. Beygelzimer, J. Langford, B. Zadrozny (2005) Weighted One-Against-All, *Proceedings of the 20th National Conference on Artificial Intelligence* (AAAI), 720–725.
9. E. Bredensteiner and K. Bennett (1999) Multicategory classification by Support Vector Machines, *Computational Optimization and Applications*, 12(1-3): 53–79.
10. L. Breiman (1996) Bagging predictors, *Machine Learning*, 26(2):123–140.
11. K. Crammer and Y. Singer (2001) On the algorithmic implementation of multiclass mernel-based vector machines, *Journal of Machine Learning Research* 2: 265–292.
12. K. Crammer and Y. Singer (2002) On the learnability and design of output codes for multiclass problems, *Machine Learning*, 47, 2-3: 201–233.
13. T. Dietterich and G. Bakiri (1995) Solving multiclass learning problems via error-correcting output codes, *Journal of Artificial Intelligence Research*, 2: 263–286.
14. P. Domingos (1999) MetaCost: A general method for making classifiers cost-sensitive, *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining* (KDD), 155–164.
15. C. Drummond and R. Holte (2000) Exploiting the cost (in)sensitivity of decision tree splitting criteria, *Proceedings of the 17th International Conference on Machine Learning* (ICML), 239–246.
16. C. Elkan (2001) The foundations of cost-sensitive learning, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (IJCAI), 973–978.
17. Y. Freund, Y. Mansour and R. Schapire (2004) Generalization bounds for averaged classifiers, *The Annals of Statistics*, 32(4): 1698–1722.
18. Y. Freund and R. Schapire (1997) A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences*, 55(1): 119–139.
19. Y. Guermeur, A. Elisseeff, and H. Paugam-Moisy (2000) A new multi-class SVM based on a uniform convergence result, *Proceedings of the IEEE International Joint Conference on Neural Networks* 4, 183–188.
20. V. Guruswami and A. Sahai (1999) Multiclass learning, boosting, and error-correcting codes, *Proceedings of the 12th Annual Conference on Computational Learning Theory* (COLT), 145–155.
21. T. Hastie and R. Tibshirani (1998) Classification by pairwise coupling, *Advances in Neural Information Processing Systems* (NIPS), 507–513.
22. L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. Joseph, and N. Taft (2007) Communication-efficient online detection of network-wide anomalies, *Proceedings of the 26th Annual IEEE Conference on Computer Communications* (INFOCOM), 134–142.
23. A. Kalai and R. Servedio (2003) Boosting in the presence of noise, *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing* (STOC), 195–205.
24. M. Kearns (1998) Efficient noise-tolerant learning from statistical queries, *Journal of the ACM*, *45:6*, 983–1006.
25. R. Koenker and K. Hallock (2001) Quantile regression, *Journal of Economic Perspectives*, 15, 143–156.
26. J. Langford and A. Beygelzimer (2005) Sensitive Error Correcting Output Codes, *Proceedings of the 18th Annual Conference on Learning Theory* (COLT), 158–172.

27. J. Langford, R. Oliveira and B. Zadrozny (2006) Predicting conditional quantiles via reduction to classification, *Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence* (UAI).
28. J. Langford and B. Zadrozny (2005) Estimating class membership probabilities using classifier learners, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*.
29. J. Langford and B. Zadrozny (2005) Relating reinforcement learning performance to classification performance, *Proceedings of the 22nd International Conference on Machine Learning* (ICML), 473–480.
30. Y. Lee, Y. Lin, and G. Wahba (2004) Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data, *Journal of American Statistical Association*, 99: 67–81.
31. C. Hsu and C. Lin (2002) A comparison of methods for multi-class support vector machines, *IEEE Transactions on Neural Networks*, 13, 415–425.
32. D. Margineantu (2002) Class probability estimation and cost-sensitive classification decisions, *Proceedings of the 13th European Conference on Machine Learning*, 270–281.
33. M. Mesnier, M. Wachs, R. Sambasivan, A. Zheng, and G. Ganger (2007) Modeling the relative fitness of storage, *International Conference on Measuremen and Modeling of Computer Systems* (SIGMETRICS), 37–48.
34. J. von Neumann (1951) Various techniques used in connection with random digits, *National Bureau of Standards*, *Applied Mathematics Series*, *12*: 36–38.
35. J. Platt (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, 61–74.
36. J. Platt, N. Cristiani and J. Shawe-Taylor (2000) Large margin DAGs for multiclass classification, *Advances of Neural Information Processing Systems*, 12: 547–553.
37. J. Platt, E. Kiciman and D. Maltz (2008) Fast variational inference for large-scale internet diagnosis, *Advances in Neural Information Processing Systems* 20.
38. R. Rifkin and A. Klautau (2004) In defense of one-vs-all classification, *Journal of Machine Learning Research*, 5: 101–141.
39. I. Rish, M. Brodie and S. Ma (2002) Accuracy versus efficiency in probabilistic diagnosis, *Proceedings of National Conference on Artificial Intelligence* (AAAI), 560–566.
40. I. Takeuchi, Q. Le, T. Sears, and A. Smola (2006) Nonparametric quantile estimation, *The Journal of Machine Learning Research*, 7, 1231–1264.
41. G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson, and C. Lefurgy (2008) Managing power consumption and performance of computing systems using reinforcement learning, *Advances in Neural Information Processing Systems* 20.
42. V. Vapnik (1998) *Statistical Learning Theory*, John Wiley and Sons.
43. H. Wang, J. Platt, Y. Chen, R. Zhang, and Y. Wang (2004) Automatic Misconfiguration Troubleshooting with PeerPressure, *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, (2004). Also in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems*, SIGMETRICS 2004, 398–399.
44. J. Weston and C. Watkins (1998) Multiclass support vector machines, *Proceedings of the 11th European Symposium on Artificial Neural Networks*, 219–224.
45. I. Witten and E. Frank (2000) *Data Mining: Practical machine learning tools with Java implementations*, Morgan Kaufmann, `http://www.cs.waikato.ac.nz/ml/weka/`.
46. B. Zadrozny and C. Elkan (2001) Learning and making decisions when costs and probabilities are both unknown, *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining* (KDD), 203–213.
47. B. Zadrozny, J. Langford and N. Abe (2003) Cost-sensitive learning by cost-proportionate example weighting, *Proceedings of the 3rd IEEE International Conference on Data Mining* (ICDM), 435–442.

Index