Reducing Multiclass to Binary

A Unifying Approach for Margin Classifiers

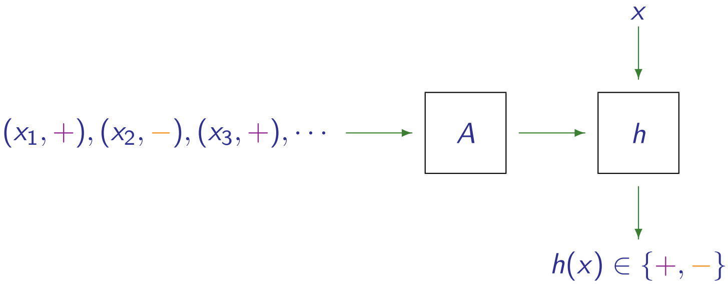
Erin Allwein
Rob Schapire
Yoram Singer

The problem

- Given: multiclass data
    - e.g.: predict children's favorite colors

⟨`name=Cindy  , age=5 , sex=F`⟩, 🟨
⟨`name=Marcia, age=15, sex=F`⟩, 🟥
⟨`name=Bobby  , age=6 , sex=M`⟩, 🟦
⟨`name=Jan    , age=12, sex=F`⟩, 🟨
⟨`name=Peter  , age=13, sex=M`⟩, 🟩

- Given: binary (2-class) learning algorithm A



$(x_1, +), (x_2, -), (x_3, +), \cdots$

A

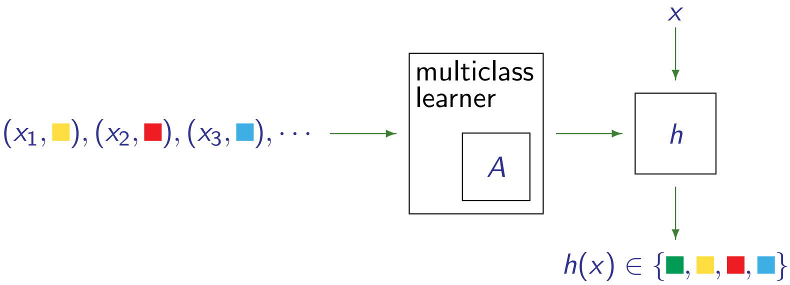
h

x

$h(x) \in \{+, -\}$

The problem (cont.)

- **Goal**: use binary learning algorithm to build classifier for multiclass data



$(x_1,$ 🟨$), (x_2,$ 🟥$), (x_3,$ 🟦$), \cdots$

multiclass learner

A

x

h

$h(x) \in \{$🟩, 🟨, 🟥, 🟦$\}$

The one-against-all approach

[folklore]

- break k-class problem into k binary problems and solve each separately

			🟩	🟨	🟥	🟦
x_1	🟨		x_1 $-$	x_1 $+$	x_1 $-$	x_1 $-$
x_2	🟥		x_2 $-$	x_2 $-$	x_2 $+$	x_2 $-$
x_3	🟦	$\Rightarrow$	x_3 $-$	x_3 $-$	x_3 $-$	x_3 $+$
x_4	🟨		x_4 $-$	x_4 $+$	x_4 $-$	x_4 $-$
x_5	🟩		x_5 $+$	x_5 $-$	x_5 $-$	x_5 $-$
			$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
			h_1	h_2	h_3	h_4

- how to combine predictions?
  - evaluate all h's and hope exactly one is $+$
  - e.g.: if $h_1(x) = h_2(x) = h_4(x) = -$ and $h_3(x) = +$ then predict 🟥
- problem: will give incorrect prediction if only one h is wrong

The all-pairs approach

[Friedman][Hastie & Tibshirani]

- create one binary problem for each pair of classes

			🟩 vs. 🟨	🟩 vs. 🟥	🟥 vs. 🟦	🟦 vs. 🟨	🟩 vs. 🟦	🟥 vs. 🟨
x_1	🟨		x_1 $-$			x_1 $-$		x_1 $-$
x_2	🟥			x_2 $-$	x_2 $+$			x_2 $+$
x_3	🟦	$\Rightarrow$			x_3 $-$	x_3 $+$	x_3 $-$	
x_4	🟨		x_4 $-$			x_4 $-$		x_4 $-$
x_5	🟩		x_5 $+$	x_5 $+$			x_5 $+$	
			$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
			h_1	h_2	h_3	h_4	h_5	h_6

- not obvious how to combine predictions
- can be highly accurate, and even faster than one-against-all

[Fürnkranz]

Error-correcting output codes

[Dietterich & Bakiri]

- reduce to binary using "coding" matrix **M**
- rows of **M** ↔ code words

M	1	2	3	4	5
🟩	+	−	+	−	+
🟨	−	−	+	+	+
🟥	+	+	−	−	−
🟦	+	+	+	+	−

			1	2	3	4	5
x_1	🟨		x_1 −	x_1 −	x_1 +	x_1 +	x_1 +
x_2	🟥		x_2 +	x_2 +	x_2 −	x_2 −	x_2 −
x_3	🟦	⇒	x_3 +	x_3 +	x_3 +	x_3 +	x_3 −
x_4	🟨		x_4 −	x_4 −	x_4 +	x_4 +	x_4 +
x_5	🟩		x_5 +	x_5 −	x_5 +	x_5 −	x_5 +
			⇓	⇓	⇓	⇓	⇓
			h_1	h_2	h_3	h_4	h_5

- given x, choose closest row of **M** to $\mathbf{h}(x) = \langle h_1(x), \ldots, h_5(x) \rangle$
  - e.g.: if $\mathbf{h}(x) = \langle -, +, +, +, - \rangle$ then predict 🟦

ECOC (continued)

- if rows of **M** far from one another,
  will be highly robust to errors
- potentially much faster when k (# of classes) large
- disadvantage:
  binary problems may be unnatural and hard to solve

This work

- unified framework for studying reductions from multiclass to binary
  - particularly for “margin-based” binary learners
- general “decoding” methods for combining predictions
- general training error bounds
- margins-based analysis of generalization error when binary learner is AdaBoost [omitted]
- experiments

A (slightly) more general approach

- choose $\{-1, 0, +1\}$ matrix $\mathbf{M}$

$\mathbf{M}$	1	2	3	4	5
🟩	0	−	0	−	+
🟨	−	−	+	0	+
🟥	+	0	−	−	−
🟦	+	+	0	+	0

			1	2	3	4	5
x_1	🟨		x_1 −	x_1 −	x_1 +		x_1 +
x_2	🟥		x_2 +		x_2 −	x_2 −	x_2 −
x_3	🟦	⇒	x_3 +	x_3 +		x_3 +	
x_4	🟨		x_4 −	x_4 −	x_4 +		x_4 +
x_5	🟩			x_5 −		x_5 −	x_5 +
			⇓ h_1	⇓ h_2	⇓ h_3	⇓ h_4	⇓ h_5

- generalizes all three earlier appproaches
- how to do decoding?

Hamming decoding

- (generalized) Hamming distance between $\mathbf{u}, \mathbf{v} \in \{-1, 0, +1\}^{\ell}$:

$$\Delta(\mathbf{u}, \mathbf{v}) = \frac{1}{\ell} \sum_{s=1}^{\ell} \begin{cases} 0 & \text{if } u_s = v_s \wedge u_s \neq 0 \wedge v_s \neq 0 \\ 1 & \text{if } u_s \neq v_s \wedge u_s \neq 0 \wedge v_s \neq 0 \\ 1/2 & \text{if } u_s = 0 \vee v_s = 0. \end{cases}$$

- e.g.: $\begin{aligned} \Delta(& \langle + \quad - \quad 0 \quad 0 \quad + \rangle, \\ & \langle + \quad + \quad - \quad 0 \quad + \rangle\ ) \end{aligned}$

$$= \tfrac{1}{5}(0 + 1 + \tfrac{1}{2} + \tfrac{1}{2} + 0)$$

- to classify x
  - evaluate $\mathbf{h}(x) = \langle h_1(x), \ldots, h_{\ell}(x) \rangle$
  - choose row r closest to $\mathbf{h}(x)$ in Hamming distance
- weakness: ignores "confidence" of predictions

Margin-based binary learning algorithms

- typical binary classifier produces real-valued predictions
  i.e., $h(x) \in \mathbb{R}$ rather than $h(x) \in \{-, +\}$
- interpretation:
  - $\mathrm{sign}(h(x))$ = predicted class ($-$ or $+$)
  - $|h(x)|$ = "confidence"
- margin of labeled example (x, y) is $yh(x)$
  - $yh(x) > 0 \Leftrightarrow$ correct prediction
  - $|yh(x)| = |h(x)|$ = "confidence"
- many learning algorithms attempt to find classifier h minimizing

$$\sum_{i=1}^{m} L(y_i h(x_i))$$

where $(x_1, y_1), \ldots, (x_m, y_m)$ is given training set
L is loss function of margin

Examples of margin-based learning algorithms

algorithm	$L(z)$
AdaBoost	e^{-z}
logistic regression	$\ln(1 + e^{-z})$
least-squares regression	$(1 - z)^2$
support-vector machines	$\max\{1 - z, 0\}$
C4.5	$\ln(1 + e^{-z})$
CART	$(1 - z)^2$

Loss-based decoding

- alternative to Hamming decoding when using margin-based learning algorithm
- to classify x
  - evaluate $\mathbf{h}(x) = \langle h_1(x), \ldots, h_\ell(x) \rangle$
  - for each class r
    - compute total loss that would have been suffered if x were labeled r

$$\sum_{s=1}^{\ell} L(M(r, s)\ h_s(x))$$

  - choose class r that minimizes total loss

Loss-based decoding (example)

- suppose:

$$\mathbf{M} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline \text{🟩} & 0 & - & 0 & - & + \\ \text{🟨} & - & - & + & 0 & + \\ \text{🟥} & + & 0 & - & - & - \\ \text{🟦} & + & + & 0 & + & 0 \end{array}$$

$$\mathbf{h}(x) = \langle \quad 1.1, \quad -3.1, \quad 4.2, \quad 0.7, \quad 6.2 \quad \rangle$$

- then compute:

	total loss for each class
🟩	$L(\ 0\ )+L(\ 3.1\ )+L(\ 0\ )+L(-0.7)+L(\ 6.2\ ) = \ 3.7$
🟨	$L(-1.1)+L(\ 3.1\ )+L(\ 4.2\ )+L(\ 0\ )+L(\ 6.2\ ) = \ 3.1$
🟥	$L(\ 1.1\ )+L(\ 0\ )+L(-4.2)+L(-0.7)+L(-6.2) = 15.1$
🟦	$L(\ 1.1\ )+L(-3.1)+L(\ 0\ )+L(\ 0.7\ )+L(\ 0\ ) = \ 6.4$

- predict 🟨

Training error bounds

- $\varepsilon =$ loss averaged over all training examples and all ℓ binary problems

$$\varepsilon = \frac{1}{m\ell} \sum_{i=1}^{m} \sum_{s=1}^{\ell} L(\ M(y_i, s)\ h_s(x_i)\ )$$

- $\rho =$ minimum distance between pairs of rows of **M**

$$\rho = \min\{\ \Delta(\ M(r_1, \cdot), M(r_2, \cdot)\ ) : r_1 \neq r_2\ \}$$

- assume $L(0) = 1$ and other benign technical conditions
- Theorem: For loss-based decoding

$$\text{training error} \leq \frac{\varepsilon}{\rho}$$

- Theorem: For Hamming decoding

$$\text{training error} \leq \frac{2\varepsilon}{\rho}$$

Trade-offs in code design

- want good performance on binary problems (so that ε small)
- want rows far apart from each other (so ρ large)
  - $\rho \approx 1/2$ for ECOC
  - $\rho = 2/k$ for one-against-all
- but: making ρ large may mean making binary problems more difficult to solve
- many zeros in matrix makes binary problems easier and faster to solve
- but: adding zeros tends to increase ε
- update: better bounds for sparse matrices (e.g. all-pairs) achievable by ignoring 0 entries during decoding

Experiments

- tested SVM on 8 benchmark problems using both decoding methods and 5 different coding matrices
- overall, loss-based decoding is better than Hamming decoding
- one-against-all...
  - sometimes as good as others
  - usually not the best
  - never substantially better than any other method
- best method seems highly problem dependent
- however, [Rifkin & Klautau] later argued one-against-all as good as others if binary SVM's have been well tuned

When output coding might help

- might be helpful when...
    - very large number of classes
        - speed-up possible using few columns (ideal: $O(\log k)$) or many 0 entries
    - using codes that incorporate background knowledge to balance:
        - robustness/efficiency of code
        - "naturalness" of binary learning problems
- e.g. when classes can be arranged in hierarchy or described naturally by features/attributes
- same setting as "structured prediction" — very large number of highly structured classes (e.g.: class = sequence of tags)
    - many recent algorithms and advances:
        - [Taskar, Guestrin & Koller]
        - [Tsochantaridis, Joachims, Hofmann & Altun]
        - [Collins, Globerson, Koo, Carreras & Bartlett]

        ⋮

A (small) sampling of more recent work

- automatic design of codes [Crammer & Singer]
    - provably intractable
    - but efficiently solvable using real-valued codes
      → multiclass version of SVM
- more general decoding schemes and improved analyses
  [Klautau, Jevtic & Orlitsky] [Escalera, Pujol & Radeva]
- theoretically optimal reduction of multiclass to binary using "error-correcting tournaments"
  [Beygelzimer, Langford & Ravikumar]

Concluding perspectives

- binary classification was right place to start, but extension to multiclass generally not straightforward
- myriad ways to deal with multiclass in general
  - makes problem both rich and difficult
- this work: attempted to take early steps in developing unified tools and theory
- multiclass problems present same critical issues as elsewhere in machine learning:
  - balancing of trade-offs
  - incorporation of prior knowledge
  - computational efficiency
- continuing progress will depend on how these issues are addressed