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
    
    \[
    \langle \text{name=} \text{Cindy} , \text{age=} 5 , \text{sex=} \text{F}\rangle, \quad \text{黄色}
    \]
    
    \[
    \langle \text{name=} \text{Marcia} , \text{age=} 15 , \text{sex=} \text{F}\rangle, \quad \text{红色}
    \]
    
    \[
    \langle \text{name=} \text{Bobby} , \text{age=} 6 , \text{sex=} \text{M}\rangle, \quad \text{蓝色}
    \]
    
    \[
    \langle \text{name=} \text{Jan} , \text{age=} 12 , \text{sex=} \text{F}\rangle, \quad \text{黄色}
    \]
    
    \[
    \langle \text{name=} \text{Peter} , \text{age=} 13 , \text{sex=} \text{M}\rangle, \quad \text{绿色}
    \]

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

\[
(x_1, +), (x_2, -), (x_3, +), \ldots \quad \rightarrow \quad A \quad \rightarrow \quad h \quad \rightarrow \quad h(x) \in \{+, -\}
\]
The problem (cont.)

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

\[(x_1, \square), (x_2, \square), (x_3, \square), \ldots\] → \(A\) → \(h(x) \in \{\text{ }, \text{ }, \text{ }, \text{ }\}\)
The one-against-all approach

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

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>✈️</td>
<td>🟢</td>
<td>🟡</td>
<td>🟠</td>
<td>🟡</td>
</tr>
</tbody>
</table>

- 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

- create one binary problem for each pair of classes

- not obvious how to combine predictions

- can be highly accurate, and even faster than one-against-all

[Friedman][Hastie & Tibshirani]
Error-correcting output codes

- reduce to binary using "coding" matrix $M$
- rows of $M \leftrightarrow$ code words

<table>
<thead>
<tr>
<th>$M$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>$x_1$</td>
<td>[green]</td>
<td>$x_1$</td>
<td>$x_1$</td>
<td>$x_1$</td>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>[red]</td>
<td>$x_2$</td>
<td>$x_2$</td>
<td>$x_2$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>[blue]</td>
<td>$x_3$</td>
<td>$x_3$</td>
<td>$x_3$</td>
<td>$x_3$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>[yellow]</td>
<td>$x_4$</td>
<td>$x_4$</td>
<td>$x_4$</td>
<td>$x_4$</td>
</tr>
<tr>
<td>$x_5$</td>
<td>[green]</td>
<td>$x_5$</td>
<td>$x_5$</td>
<td>$x_5$</td>
<td>$x_5$</td>
</tr>
<tr>
<td>$h_1$</td>
<td>$h_2$</td>
<td>$h_3$</td>
<td>$h_4$</td>
<td>$h_5$</td>
<td></td>
</tr>
</tbody>
</table>

- given $x$, choose closest row of $M$ to $h(x) = \langle h_1(x), \ldots, h_5(x) \rangle$
- e.g.: if $h(x) = \langle −, +, +, +, − \rangle$ then predict [red]
ECOC (continued)

- if rows of $\mathbf{M}$ 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 \( M \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( h_3 )</th>
<th>( h_4 )</th>
<th>( h_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
</tr>
</tbody>
</table>

- generalizes all three earlier approaches
- how to do decoding?
Hamming decoding

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

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

- e.g.: $\Delta(\langle + - 0 0 + \rangle, 
\langle + + - 0 + \rangle) = \frac{1}{5}(0 + 1 + \frac{1}{2} + \frac{1}{2} + 0)$

- to classify $x$
  - evaluate $h(x) = \langle h_1(x), \ldots, h_\ell(x) \rangle$
  - choose row $r$ closest to $h(x)$ 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:
  - $\text{sign}(h(x)) =$ predicted class ($-$ or $+$)
  - $|h(x)| =$ “confidence”

- margin of labeled example $(x, y)$ is $yh(x)$
  - $yh(x) > 0 \iff$ 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$L(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoost</td>
<td>$e^{-z}$</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>$\ln(1 + e^{-z})$</td>
</tr>
<tr>
<td>Least-squares regression</td>
<td>$(1 - z)^2$</td>
</tr>
<tr>
<td>Support-vector machines</td>
<td>$\max{1 - z, 0}$</td>
</tr>
<tr>
<td>C4.5</td>
<td>$\ln(1 + e^{-z})$</td>
</tr>
<tr>
<td>CART</td>
<td>$(1 - z)^2$</td>
</tr>
</tbody>
</table>
Loss-based decoding

- alternative to Hamming decoding when using margin-based learning algorithm
- to classify \( x \)
  - evaluate \( h(x) = \langle h_1(x), \ldots, h_\ell(x) \rangle \)
  - 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:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>0</td>
<td>−</td>
<td>0</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>−</td>
<td>−</td>
<td>+</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>0</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>0</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ h(x) = \langle 1.1, -3.1, 4.2, 0.7, 6.2 \rangle \]

- Then compute:

<table>
<thead>
<tr>
<th></th>
<th>[ L(0) + L(3.1) + L(0) + L(-0.7) + L(6.2) = 3.7 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ L(-1.1) + L(3.1) + L(4.2) + L(0) + L(6.2) = 3.1 ]</td>
</tr>
<tr>
<td></td>
<td>[ L(1.1) + L(0) + L(-4.2) + L(-0.7) + L(-6.2) = 15.1 ]</td>
</tr>
<tr>
<td></td>
<td>[ L(1.1) + L(-3.1) + L(0) + L(0.7) + L(0) = 6.4 ]</td>
</tr>
</tbody>
</table>

- Predict ■
Training error bounds

- $\varepsilon = \text{loss}$ averaged over all training examples and all $\ell$ binary problems

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

- $\rho = \text{minimum distance}$ 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 $\varepsilon$ small)
• want rows far apart from each other (so $\rho$ large)
  • $\rho \approx 1/2$ for ECOC
  • $\rho = 2/k$ for one-against-all
• but: making $\rho$ 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 $\varepsilon$
• 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