Multiclass Classification
STAT 535 Lecture 4

Marina Meilã
mmp@stat.washington.edu

Department of Statistics
University of Washington
Outline

Intrinsic multiclass classifiers

Multiclass Support Vector Machine

Multiclass from Binary
  One versus all
  Pairwise voting
  Error Correcting Output Codes
Intrinsic multiclass classifiers

- decision trees
- K-nearest neighbor
- neural networks
- generative models
  - they estimate $P(X|Y = y)$ separately for each $y$
  - Naive Bayes (special case)
- other probabilistic models which estimate $P(Y|X)$ explicitly
**Multiclass Neural Network**

- Assume \( f(x) \in 1 : m \) induces a probabilistic model
  \[ P_f(x) \in [0, 1]^m = P[Y|X = x] \]
- Common example \( \tilde{\phi}(u) : \mathbb{R}^m \rightarrow (0, 1)^m \) the **softmax** function
  \[
  \tilde{\phi}_j(u) = \frac{e^{u_j}}{\sum_{i=1}^{m} e^{u_i}}, \quad u = [u_1 \ldots u_m]^T \in \mathbb{R}^m
  \]
  For any \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \), \( \tilde{\phi}_j(f(x)) = P_f(Y = j|X = x) \)
- **Loss** = \( L_{logit} \)
  \[
  L_{logit}(y, f(x)) = -\ln \tilde{\phi}_y(f(x)) = -\ln P_f(y|x)
  \]
- \( f(x) \) is a linear output neural network.
  E.g \( f_j(x) = \beta_j^T \phi(Wx) \), where \( W \in \mathbb{R}^{n \times p} \) is the matrix of weights of the first layer, \( \phi \) is the logistic function applied to each component of its vector argument, \( \beta_j \in \mathbb{R}^p \) is the weight vector of \( f_j \).
- **Exercise** Derive the gradient expression of \( \frac{\partial \hat{L}_{logit}}{\partial W} \) and \( \frac{\partial \hat{L}_{logit}}{\partial \beta} \)
Multiclass SVM

Idea: learn a \((w, b)\) pair for each class. Classify \(x\) by its longest projection onto the \((w_y, b_y)\) vectors.

Let \(m = \# \text{ classes}\)

\[
\begin{align*}
\min_{w_{1:m}, b_{1:m}, \xi} & \quad \frac{1}{2} \sum_{y=1}^{m} ||w_y||^2 + C \sum_{i=1}^{N} \xi_i \\
\text{s.t.} & \quad w_{y_i}^T x^i + b_{y_i} \geq 1 - \xi_i + w_{y'}^T x^i + b_{y'} \quad \text{for } y' \neq y^i, \ y' = 1 : m, \ i = 1 : N \\
& \quad \xi_i \geq 0, \ \text{for } i = 1 : N
\end{align*}
\]

Classify new \(x\) by

\[
\hat{y} = \arg\max_{y=1:m} w_y^T x + b_y
\]
Multiclass from Binary

- One class vs all (OVA)
- Pairwise voting (AVA)
- error correcting output codes (ECOC)
The general setup

- The class label $y$ is **encoded** as a *binary string* of length $L$
  
  $y \in \{1, 2, \ldots, m\}$ class label $\xrightarrow{h} h(y) \in \{\pm 1\}^L$

**Example** *sparse coding*

\[
  y \rightarrow h(y) = [-1 \ldots -1 \underbrace{1}_{i^{th}} -1\ldots -1]
\]

- Train $L$ binary classifiers $f_i$, $l = 1 : L$ with data of the form
  
  $D_l = \{(x^i, h_l(y^i)), i = 1 : N\}$.

- **Testing**
  
  - $x \rightarrow f(x) = [f_1(x) \ldots f_i(x) \ldots f_L(x)] \in \mathbb{R}^L \text{or} \{\pm 1\}$
  
  - Decode $f(x)$ into label $\hat{y}(x) \in 1 : m$
One versus all (OVA)

\[ h_l(y) = \begin{cases} 
1 & \text{if } y = l \\
-1 & \text{if } y \neq l 
\end{cases} \] (7)

- train \( f_l \) on \( D_l = \{ (x^i, h_l(y^i)) \}, l = 1 : m \)
  hence \( f_l \) is trained to discriminate class \( l \) from all the others
- decode \( \hat{y} = \arg \max_l f_l(x) \)
One vs. all: pros and cons

- Simple, easy to set up
- Many “intrinsic” multiclass classifiers can be viewed as 1-vs-all
  - Generative classifiers, logistic regression, neural nets, multiclass SVM
- Decision regions of $f_i$ relatively simple (compared to ECOC further on)
Pairwise voting (All Vs. All)

- **Idea** train a classifier \( f_{kl} \) for each pair of labels \( k, l \in 1: m, k < l \). Then \( f_{kl} \) will learn to discriminate between the two classes.

- **Training**
  - For \((k, l) \in \{(1, 2), (1, 3), \ldots (m - 1, m)\}\),

  \[
  h_{kl}(y) = \begin{cases} 
  1 & \text{if } y = k \\
  -1 & \text{if } y = l \\
  \text{undefined} & \text{if } y \neq k, l
  \end{cases}
  \]  

  (8)

- Train \( f_{kl} \) on \( D_{kl} = \{(x^i, h_{kl}(y^i)) | y^i \in \{k, l\}\} \)

- **Decoding** (many variants proposed!)
  - \( f_{kl} = 1 \) is vote for \( k \)
  - \( f_{kl} = 1 \) is vote against \( l \)
  - **pairwise coupling** \( f_{kl} \approx P[y = k | y = k \text{ or } l] \)

- **Remark:** boundaries even simpler than (OVA)
Count each $f_{kl}$ as “vote for” $k$ or $l$

\[
\hat{y}(x) = \arg\max_k \sum_{l \neq k} f_{kl}(x)
\]
or
\[
\hat{y}(x) = \arg\max_k \sum_{l \neq k} \text{sgn} f_{kl}(x)
\]
we set by convention $f_{lk} = -f_{kl}$

Remark: if $y = k$ and $f_{kl} = 1$ for all $l \neq k$ then $\hat{y} = y$ no matter what values $f_{k'l'}$ take
Count each $f_{kl}$ as “vote against” $k$ or $l$

- **Idea** When $f_{kl} = 1$, $Pr(Y = l|X)$ is decreased, and $Pr(Y = l'|X)$ increased for all $l' \neq l$.
- **Input parameters:** $p_l = P(Y = l)$ prior probability, $\varepsilon_{kl}$ decay parameter (set by the user)
- When $f_{kl} = 1$, $P[Y = l|f_{kl}] = p_l \varepsilon_{kl}$, $P[Y = l'|f_{kl}] = p_{l'} \frac{1 - \varepsilon_{kl} p_l}{1 - p_l}$ for $l' \neq l$
- We now write the probability of $Y = l$, assuming the values $\{f_{kl}\}$ are mutually independent given $Y = l$.

\[
P[l|\{f_{k'l'}\}] \propto p_l \prod_{k',l'} P[f_{k'l'}|Y = l] = p_l \prod_{k',l'} \frac{P[Y = l|f_{k'l'}]P[f_{k'l'}]}{p_l}
\]

\[
\ln P[l|\{f_{k'l'}\}] = \left(1 - \frac{m(m-1)}{2}\right) \ln p_l + \sum_{k',l'} \ln P[Y = l|f_{k'l'}] + \text{const.}
\]

\[
= \left(1 - \frac{m(m-1)}{2}\right) \ln p_l + \sum_{k',l' \text{against } l} (\ln p_l + \ln \varepsilon_{k'l'}) + \text{const.}
\]

- **Decode by** $\hat{y} = \arg\max \sum_l \ln P[l|\{f_{k'l'}\}]$
- **Remark:** can work with a subset of all $k, l$ pairs
- **Variation:** **Tournament:** Fixed elimination scheme between classes.
Pairwise coupling

- Idea 1. View \( f_{kl} \approx P[y = k | y = k \text{ or } l] \). Let \( p_k = P[Y = k | X = x] \) and \( N_{kl} = |D_{kl}| \). Then

\[
 f_{kl} = \frac{p_k}{p_k + p_l} \overset{\text{def}}{=} r_{kl} \quad \text{and} \quad \ln r_{kl} = \ln p_k - \ln(p_k + p_l) \quad (12)
\]

- Solve for \( p_{1:m} \) so that \( r_{kl} \approx f_{kl} \) for all \((k, l)\). For this
  - Assume \( N_{kl} r_{kl} \equiv (N_{kl} p_k, N_{kl} p_l) \sim \text{Binomial}(N_{kl}, f_{kl}) \).
  - Minimize

\[
 \min_{p_{1:m}} E[KL(f, r)] \approx \min_{p_{1:m}} \sum_{k < l} N_{kl} f_{kl} \ln \frac{f_{kl}}{r_{kl}} + N_{kl} f_{kl} \ln \frac{1 - f_{kl}}{1 - r_{kl}} \quad (13)
\]

- Algorithm to find \( p_{1:m} \)
  - Start with guess \( p_{1:m} \), compute \( \{r_{kl}\} \)
  - Repeat until convergence
    1. for \( k = 1 : m \), \( p_k \leftarrow p_k \frac{\sum_{k \neq l} N_{kl} f_{kl}}{\sum_{k \neq l} N_{kl} r_{kl}} \)
    2. Normalize \( \{p_k\} \), recalculate \( \{r_{kl}\} \)

- Decode \( \hat{y}(x) = \arg\max_k p_k \)

- Drawback: it is possible to have \( f_{kl} > 0.5 \) for all \( l \neq k \), yet to converge to a \( p_k \) that is not the largest.
Error Correcting Output Codes (ECOC)

- **Idea** Each binary classifier $f_{kl}(x)$ provides a bit of information about the true class. But some of the $f_{kl}(x)$ values may be incorrect, and others may be irrelevant (e.g. $f_{12}$ is irrelevant when true class is 4) or providing information that we aren’t prepared to use. How to best handle these noisy bits? Use Information theory, specifically Error Correcting Codes.

- **Error Correcting (Linear) Code**
  - Input dictionary $v^k \in \{0, 1\}^q$.
  - Coding matrix $M \in \{0, 1\}^{q \times (L-q)}$ (carefully designed). $L > q$ the output size is a design choice.
  - Output dictionary $\mathcal{W} = \{w^k = [I \ M]^T v^k\} \subset \{0, 1\}^L$.
  - Decoding When word $w' \in \{0, 1\}^L$ is received
    - Find $w \in \mathcal{W}$ closest to $w'$ in Hamming distance.
    - Return $v = [I \ M]w$.
  - Remarks: In real decoding algorithms, the two decoding steps are performed by a single linear operation. The set $\mathcal{W}$ is never represented explicitly.
Example: Naive ECOC

- This “code” corrects one error
- $v = [v_1 \ldots v_q] \rightarrow w = [v_1 v_1 v_1 v_2 v_2 v_2 \ldots v_q v_q v_q] \in \mathcal{W}$. E.g $v = [1 0 1] \rightarrow [1 1 1 0 0 0 1 1 1]$.
- 1 error on any bit of $w$, e.g $\tilde{w} = [1 0 1 0 0 0 1 1 1]$. The nearest $w$ in the dictionary $\mathcal{W}$ to $\tilde{w}$ is the original [1 1 1 0 0 0 1 1 1]. Hence we can recover $v$ correctly.
ECOC – continued

So, how to use ECC for classification?

1. Choose a code matrix\(^1\) \(M \in \{\pm 1\}^{q \times L}\)

2. For each column \(l\) of \(M\), define \(h_l(y) = M_{y_l}\).

Construct training set \(\mathcal{D}_l = \{(x^i, y^i_l), i = 1 : N\}\). (In other words, we partition the \(m\) classes into two sets, according to the values in column \(l\) of \(M\).)

3. Train \(f_l\) on \(\mathcal{D}_l\). (I.e. \(f_l\) will discriminate between the two groups of classes corresponding to column \(l\) of \(M\)). Let \(f = [f_l]_{l=1:L} \in \{\pm 1\}^L\)

4. Test/Decode Calculate \(\hat{v} = \text{sgn} Mf\). Output \(\hat{y} = \arg\max_k v^T v^k\) (or equivalently, choose \(v^k\) closest in Hamming distance to \(v\).)

- Many variants exist.
  - \(M_{kl} \in \{0, \pm 1\}\) (not all classes included in each \(\mathcal{D}_l\).
  - Loss-based decoding instead of Hamming distance.

\(^1\)Here \(M\) stands for \([I M]^T\) from frame 14.
Example: Encoding digits

<table>
<thead>
<tr>
<th>Class</th>
<th>f0</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
<th>f5</th>
<th>f6</th>
<th>f7</th>
<th>f8</th>
<th>f9</th>
<th>f10</th>
<th>f11</th>
<th>f12</th>
<th>f13</th>
<th>f14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Classifier $l = 1$ separates digits $\{0, 2, 4, 6, 8\}$ from $\{1, 3, 5, 7, 9\}$,
  $l = 14$ separates 1 from all other digits. Note that the separations have nothing to do with the similarities/dissimilarities of the handwritten digits.

- Code corrects up to three errors
General empirical experience

- OVA, AVA with simple decoding work well enough (i.e. little evidence that more complicated classifiers improve on them)
- ECOC can improve classification error, but only slightly