multiboost.org: an implementation of cost-sensitive multi-class/multi-label AdaBoost.MH

Róbert Busa-Fekete\textsuperscript{1}, Norman Casagrande\textsuperscript{2}, and Balázs Kégl\textsuperscript{1}

\textsuperscript{1}University of Paris-Sud / CNRS
\textsuperscript{2}last.fm

ICML’10 open source workshop
June 25, 2010
reviewer 1: “using the program is marginally faster and much less instructive than writing your own version of these simple algorithms”

• Simple?

  • yes: binary AdaBoost with decision stumps is simple, but an average biologist will never sit down to implement it

  • no: implementing full cost-sensitive multi-class/multi-label AdaBoost.MH\(^1\) with decision trees/products/Haar filters is not simple even for an average computer scientist

  • WEKA contains a very suboptimal multi-class AdaBoost implementation, other than that there is no widely used implementation that we know of

---

A “simple” base learner for nominal features

\[
\text{INDICATOR BASE}(X, Y, W)
\]

1. \( \text{for } j \leftarrow 1 \text{ to } d \quad \triangleright \text{ all (nominal) features} \)
2. \( (a_j, v_j, u_j) \leftarrow \text{BEST INDICATOR}(x(j), Y, W, j(j)) \quad \triangleright \ x(j) \triangleq (x_1, \ldots, x_d) \)
3. \( j^* \leftarrow \arg\min_j E(a_j, v^j, u_j, W) \)
4. \( \text{return } (a_{j^*}, v^j, u^j, W) \)

\[
\text{BEST INDICATOR}(x, Y, W, I)
\]

1. \( \text{for } i \in I \)
2. \( \quad \text{for } \ell \leftarrow 1 \text{ to } K \)
3. \( \quad y_{i,\ell}^+ - y_{i,\ell}^- = 0 \)
4. \( w_{i,\ell} \leftarrow \text{RANDOM}(\pm 1) \)
5. \( \quad \text{for } i \leftarrow 1 \text{ to } n \text{ for } \ell \leftarrow 1 \text{ to } K \)
6. \( \quad \quad \text{if } w_{i,\ell} > 0 \text{ then} \quad \)
7. \( \quad \quad y_{i,\ell}^+ - y_{i,\ell}^- + w_{i,\ell} x_{i,\ell} \)
8. \( \quad \quad \text{else} \quad \)
9. \( y_{i,\ell}^+ - y_{i,\ell}^- - w_{i,\ell} x_{i,\ell} \)
10. \( \quad \alpha \leftarrow 0, v \leftarrow 0 \)
11. \( \quad \text{while TRUE} \quad \)
12. \( \quad \quad \upsilon_{\text{prev}} \leftarrow \alpha, v_{\text{prev}} \leftarrow v \quad \triangleright \text{ save current optimal } \alpha \text{ and } v \)
13. \( \quad \quad \text{for } \ell \leftarrow 1 \text{ to } K \)
14. \( \quad \quad \quad \upsilon_{\ell} \leftarrow \text{sign} \left( \sum_{i \in I} (y_{i,\ell}^+ - y_{i,\ell}^-) w_{i,\ell} \right) \quad \text{or } \upsilon_{\ell} \leftarrow -\frac{1}{2} \ln \frac{\sum_{i \in I} (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})}{\sum_{i \in I} (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})} \)
15. \( \quad \quad \alpha \leftarrow \frac{1}{2} \ln \frac{\sum_{i \in I} \sum_{\ell = 1}^K (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})}{\sum_{i \in I} \sum_{\ell = 1}^K (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})} \quad \text{or } \alpha \leftarrow -1 \)
16. \( \quad \quad \text{if } E(\alpha_{\text{prev}}, W) \geq E(\upsilon_{\text{prev}} v_{\text{prev}}, W) \text{ then} \quad \)
17. \( \quad \quad \quad \text{return } (\upsilon_{\text{prev}}, v_{\text{prev}}, \alpha) \quad \triangleright \text{ save current optimal } \alpha \text{ and } v \)
18. \( \quad \quad \upsilon_{\text{prev}} \leftarrow \alpha, v_{\text{prev}} \leftarrow v \quad \triangleright \text{ save current optimal } \alpha \text{ and } v \)
19. \( \quad \quad \text{for } i \in I \)
20. \( \quad \quad \quad \alpha \leftarrow -\frac{1}{2} \ln \frac{\sum_{i \in I} \sum_{\ell = 1}^K (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})}{\sum_{i \in I} \sum_{\ell = 1}^K (y_{i,\ell}^+ 1_{\{y_i > 0\}} + y_{i,\ell}^- 1_{\{y_i < 0\}})} \quad \text{or } \alpha \leftarrow -1 \)
21. \( \quad \quad \text{if } E(\alpha_{\text{prev}}, W) \geq E(\upsilon_{\text{prev}} v_{\text{prev}}, W) \text{ then} \quad \)
22. \( \quad \quad \quad \text{return } (\upsilon_{\text{prev}}, v_{\text{prev}}, \alpha) \quad \triangleright \text{ save current optimal } \alpha \text{ and } v \)
Cost sensitive multi-label/multi-class

- **Input** vector \( \mathbf{x} \in \mathcal{X}^d \): \( x^{(j)} \) is either real-valued or nominal

- **Label** vector \( \mathbf{y} \in \{-1, 1\}^K \)

- **Cost** (initial weight) vector \( \mathbf{w}^{(1)} \in (\mathbb{R}^+ \cup \{0\})^K \)

  - for example, **classical multi-class**

\[
    w_{i,\ell}^{(1)} = \begin{cases} 
      \frac{1}{2n} & \text{if } \ell \text{ is the correct class of } x_i \text{ (if } y_{i,\ell} = 1), \\
      \frac{1}{2n(K-1)} & \text{otherwise (if } y_{i,\ell} = -1). 
    \end{cases}
\]
Cost sensitive multi-label/multi-class

- AdaBoost.MH learns a vector-valued discriminant function
  \[ f(x) : \mathcal{X}^d \rightarrow \mathbb{R}^K \]

by minimizing the weighted Hamming loss

\[
R(f^{(T)}, w^{(1)}) = \sum_{i=1}^{n} \sum_{\ell=1}^{K} w_{i,\ell} \mathbb{I}\{f^{(\ell)}(x_i) y_{i,\ell} < 0\}
\]
• Strong learners
  • AdaBoost.MH$^2$
  • FeatureBoost$^3$

• Weak learners
  • Decision stump for real-valued features
  • Selector and subset indicator$^4$ for nominal features
  • Haar filter$^5$ for image input
  • Decision tree and decision product$^4$ for combining simple base-classifiers
  • Easy to add new base learners without affecting the main boosting engine

---


• Other features

  • Bandit boosting\(^6\): adaptive feature selection to accelerate training

  • Efficient multi-platform C++ implementation

  • Training and test data are input in ARFF format

  • Support for sparse data and/or label matrix

  • The classifiers are saved in XML format

  • Test/training error and other iteration-wise statistics are saved in a tab separated data file

---

Bechmark and challenge results

- **MIREX’05**
  
  www.music-ir.org/evaluation/mirex-results
  
  winner in genre classification track and runner up in the artist Identification track

- **MNIST**
  
  yann.lecun.com/exdb/mnist
  
  Boosting decision product of stumps is the best reported no-domain-knowledge algorithm on MNIST after Hinton and Salakhutdinov’s deep belief nets

- **Yahoo - Learning to Rank Challenge (ICML’10 workshop)**
  
  learningtorankchallenge.yahoo.com
  
  6th place in track 1 and 11th place in track 2. The difference between our calibrated AdaBoost.MH approach and the winners was less then 0.003 in both tracks.

---
