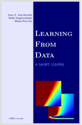
Cost-sensitive Classification: Techniques and Stories

Hsuan-Tien Lin
htlin@csie.ntu.edu.tw

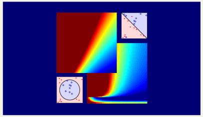
Professor
Dept. of CSIE, National Taiwan University

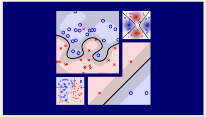
Machine Learning Summer School @ Taipei, Taiwan
August 2, 2021

About Me



- co-author of textbook '*Learning from Data: A Short Course*'
- instructor of two Coursera Mandarin-teaching ML MOOCs on Coursera



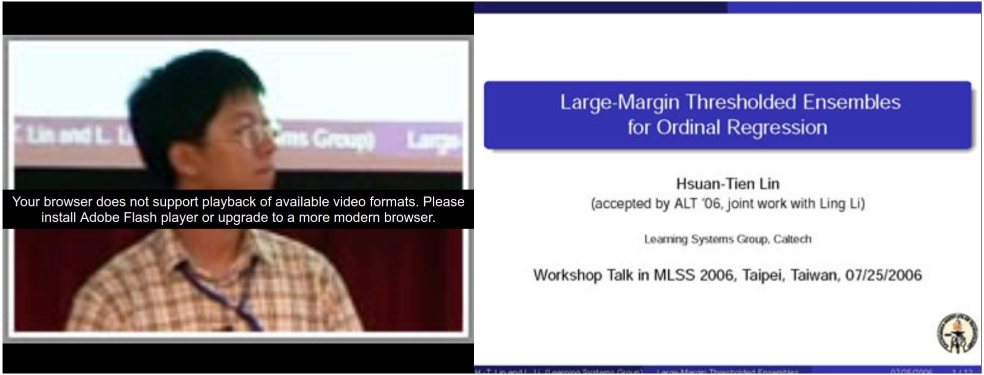


goal: make ML more realistic

- weakly supervised learning: in ICML '20, ICLR '21, . . .
- online/active learning: in ICML '12, ICML '14, AAAI '15, EMNLP '20, . . .
- **cost-sensitive classification**: in ICML '10, KDD '12, IJCAI '16, . . .
- **multi-label classification**: in NeurIPS '12, ICML '14, AAAI '18, . . .
- large-scale data mining: e.g. co-led KDDCup world-champion NTU teams 2010–2013

More About Me

attendant: MLSS Taipei 2006
student workshop talk: *Large-Margin Thresholded Ensembles for Ordinal Regression*



Disclaimer about Cost-sensitive Classification

materials mostly from "old" tutorials

- *Advances in Cost-sensitive Multiclass and Multilabel Classification*. KDD 2019 Tutorial, Anchorage, Alaska, USA, August 2019.
- *Cost-sensitive Classification: Algorithms and Advances*. ACML 2013 Tutorial, Canberra, Australia, November 2013.

- core techniques somewhat mature, compared to 10 years ago
- new research still being inspired, e.g.
  - *Classification with Rejection Based on Cost-sensitive Classification*, Charoenphakdee et al., ICML '21
  - *Cost-Sensitive Robustness against Adversarial Examples*, Zhang and Evans, ICLR '19
- will show one application story in the end

Outline

Which Digit Did You Write?

?

one (1)   two (2)   three (3)   four (4)

- a **multiclass classification** problem: grouping 'pictures' into different 'categories'

C'mon, we know about multiclass classification all too well! :-)

Performance Evaluation

?

- ZIP code recognition:
  1: wrong; 2: right; 3: wrong; 4: wrong
- check value recognition:
  1: one-dollar mistake; 2: no mistake;
  3: one-dollar mistake; 4: **two**-dollar mistake

different applications: **evaluate mis-predictions differently**

ZIP Code Recognition

?

1: wrong; 2: right; 3: wrong; 4: wrong

- **regular** multiclass classification: only right or wrong
- wrong cost: 1; right cost: 0
- prediction error of h on some $(\mathbf{x}, y)$:

$$\text{classification cost} = [\![y \neq h(\mathbf{x})]\!]$$

> regular multiclass classification:
>
> **well-studied**, many good algorithms

Check Value Recognition

?

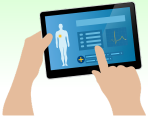
1: one-dollar mistake; 2: no mistake;
3: one-dollar mistake; 4: **two**-dollar mistake

- **cost-sensitive** multiclass classification: **different costs** for different mis-predictions
- e.g. prediction error of h on some $(\mathbf{x}, y)$:

$$\text{absolute cost} = |y - h(\mathbf{x})|$$

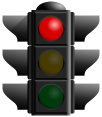
next: more about **cost-sensitive multiclass** classification (CSMC)

What is the Status of the Patient?

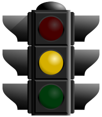


?

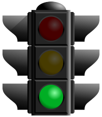
(image by mcmurryjulie from Pixabay)



bird flu



cold



healthy

(images by Clker-Free-Vector-Images from Pixabay)

- another **classification** problem: grouping ‘patients’ into different ‘status’

are all mis-prediction costs equal?

Patient Status Prediction

error measure = society cost

actual \ predicted	bird flu	cold	healthy
bird flu	0	1000	**100000**
cold	100	0	3000
healthy	100	30	0

- bird flu mis-predicted as healthy: **very high cost**
- cold mis-predicted as healthy: high cost
- cold correctly predicted as cold: no cost

human doctors consider costs of decision;
can computer-aided diagnosis do the same?

Setup: Class-Dependent Cost-Sensitive Classification

Given

N examples, each (input $\mathbf{x}_n$, label y_n) $\in \mathcal{X} \times \{1, 2, \ldots, K\}$

and cost matrix $\mathcal{C} \in \mathbb{R}^{K \times K}$ with $\mathcal{C}(y, y) = 0 = \min_{1 \le k \le K} \mathcal{C}(y, k)$

patient diagnosis with society cost

$$\mathcal{C} = \begin{pmatrix} 0 & 1000 & 100000 \\ 100 & 0 & 3000 \\ 100 & 30 & 0 \end{pmatrix}$$

check digit recognition with absolute cost

(cost function)

$$\mathcal{C}(y, k) = |y - k|$$

Goal

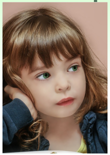
a classifier $g(\mathbf{x})$ that pays a small cost $\mathcal{C}(y, g(\mathbf{x}))$ on future **unseen** example $(\mathbf{x}, y)$

includes **regular classification** $\mathcal{C}_c$ like $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ as **special case**

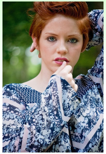
Which Age-Group?

?

infant (1)



child (2)



teen (3)

adult (4)

(images by Tawny van Breda, Pro File, Mieczysław Samol, lisa runnels, vontoba from Pixabay)

- small mistake—classify child as teen; **big mistake—classify infant as adult**
- cost matrix $\mathcal{C}(y, g(x))$ for embedding 'order': $\mathcal{C} = \begin{pmatrix} 0 & 1 & 4 & 5 \\ 1 & 0 & 1 & 3 \\ 3 & 1 & 0 & 2 \\ 5 & 4 & 1 & 0 \end{pmatrix}$

CSMC can help solve many other problems like **ordinal ranking**

Cost Vector

cost vector $\mathbf{c}$: a row of cost components

- society cost for a bird flu patient: $\mathbf{c} = (0, 1000, 100000)$
- absolute cost for digit 2: $\mathbf{c} = (1, 0, 1, 2)$
- age-ranking cost for a teenager: $\mathbf{c} = (3, 1, 0, 2)$
- 'regular' classification cost for label 2: $\mathbf{c}_c^{(2)} = (1, 0, 1, 1)$
- movie recommendation
  - someone who loves romance movie but **hates terror**:

$$\mathbf{c} = (\text{romance} = 0, \text{fiction} = 5, \textbf{terror} = \mathbf{100})$$

  - someone who loves romance movie but fine with terror:

$$\mathbf{c} = (\text{romance} = 0, \text{fiction} = 5, \text{terror} = 3)$$

cost vector:

representation of **personal preference** in many applications

Setup: Example-Dependent Cost-Sensitive Classification

Given

N examples, each (input $\mathbf{x}_n$, label y_n) $\in \mathcal{X} \times \{1, 2, \ldots, K\}$

and cost vector $\mathbf{c}_n \in \mathbb{R}^K$

—will assume $\mathbf{c}_n[y_n] = 0 = \min_{1 \le k \le K} \mathbf{c}_n[k]$

Goal

a classifier $g(\mathbf{x})$ that pays a small cost $\mathbf{c}[g(\mathbf{x})]$ on future **unseen** example $(\mathbf{x}, y, \mathbf{c})$

- will assume $\mathbf{c}[y] = 0 = c_{\min} = \min_{1 \le k \le K} \mathbf{c}[k]$
- note: y not really needed in evaluation

example-dependent $\supset$ class-dependent $\supset$ regular

Outline

Key Idea: Conditional Probability Estimator

Goal (Class-Dependent Setup)

a classifier $g(\mathbf{x})$ that pays a small cost $\mathcal{C}(y, g(\mathbf{x}))$ on future **unseen** example $(\mathbf{x}, y)$

if $P(y|\mathbf{x})$ known

Bayes optimal $g^*(\mathbf{x}) =$

$$\operatorname*{argmin}_{1 \le k \le K} \sum_{y=1}^{K} P(y|\mathbf{x}) \mathcal{C}(y, k)$$

if $q(\mathbf{x}, y) \approx P(y|\mathbf{x})$ well

approximately good $g_q(\mathbf{x}) =$

$$\operatorname*{argmin}_{1 \le k \le K} \sum_{y=1}^{K} q(\mathbf{x}, y) \mathcal{C}(y, k)$$

how to get conditional probability estimator q?

logistic regression, Naïve Bayes, . . .

Approximate Bayes-Optimal Decision

if $q(\mathbf{x}, y) \approx P(y|\mathbf{x})$ well

(Domingos, 1999)

$$\text{approximately good } g_q(\mathbf{x}) = \operatorname*{argmin}_{1 \le k \le K} \sum_{y=1}^{K} q(\mathbf{x}, y)\mathcal{C}(y, k)$$

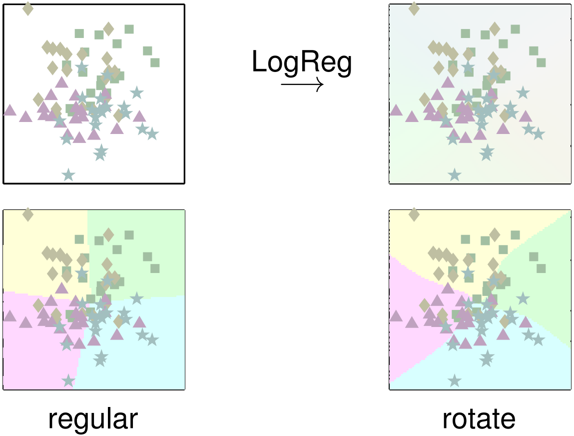
Approximate Bayes-Optimal Decision (ABOD) Approach

1. use your favorite algorithm on $\{(\mathbf{x}_n, y_n)\}$ to get $q(\mathbf{x}, y) \approx P(y|\mathbf{x})$
2. for each new input $\mathbf{x}$, predict its class using $g_q(\mathbf{x})$ above

ABOD: probability estimate + Bayes-optimal decision

ABOD on Artificial Data

1. use your favorite algorithm on $\{(\mathbf{x}_n, y_n)\}$ to get $q(\mathbf{x}, y) \approx P(y|\mathbf{x})$
2. for each new input $\mathbf{x}$, predict its class using $g_q(\mathbf{x})$ above



		$g(\mathbf{x})$			
		1	2	3	4
y	1	0	1	2	4
	2	4	0	1	2
	3	2	4	0	1
	4	1	2	4	0

ABOD for Binary Classification

Given N examples, each (input $\mathbf{x}_n$, label y_n) $\in \mathcal{X} \times \{-1, +1\}$
and weights w_+, w_- representing two entries of cost matrix

		$g(\mathbf{x})$ +1	$g(\mathbf{x})$ -1
y	+1	0	w_+
	-1	w_-	0

if $q(\mathbf{x}) \approx P(+1|\mathbf{x})$ well

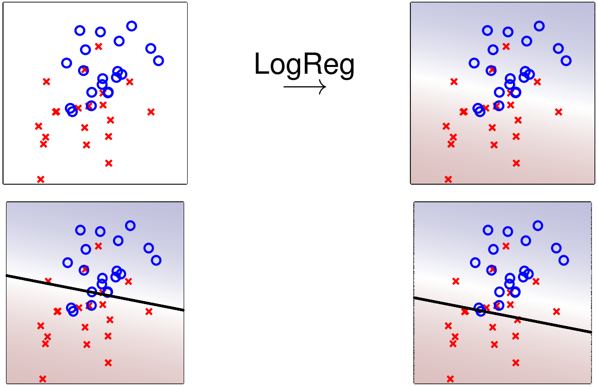
approximately good $g_q(\mathbf{x}) = \text{sign}\Big(w_+ q(\mathbf{x}) - w_-(1 - q(\mathbf{x}))\Big)$, i.e. (Elkan, 2001),

$$g_q(\mathbf{x}) = +1 \iff w_+ q(\mathbf{x}) - w_-(1 - q(\mathbf{x})) > 0 \iff q(\mathbf{x}) > \frac{w_-}{w_+ + w_-}$$

ABOD for binary classification:
probability estimate + threshold changing

ABOD for Binary Classification on Artificial Data

1. use your favorite algorithm on $\{(\mathbf{x}_n, y_n)\}$ to get $q(\mathbf{x}) \approx P(+1|\mathbf{x})$
2. for each new input $\mathbf{x}$, predict its class using $g_q(\mathbf{x}) = \text{sign}\left(q(\mathbf{x}) - \frac{w_-}{w_+ + w_-}\right)$



regular | positive emphasis

		$g(\mathbf{x})$: +1	$g(\mathbf{x})$: -1
y	+1	0	10
	-1	1	0

Pros and Cons of ABOD

Pros

- optimal if good probability estimate q
- prediction easily adapts to different $\mathcal{C}$ without modifying training (probability estimate)

Cons

- 'difficult': good probability estimate often more difficult than good multiclass classification
- 'restricted': only applicable to class-dependent setup —need 'full picture' of cost matrix
- 'slower prediction' (for multiclass): more calculation at prediction stage

can we use any multiclass classification algorithm for ABOD?

MetaCost Approach

Approximate Bayes-Optimal Decision (ABOD) Approach

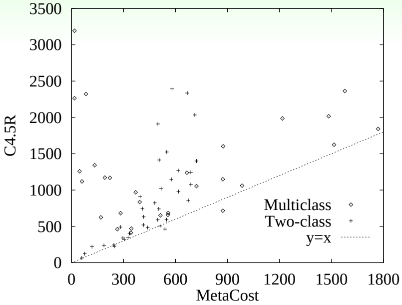
1. use your favorite algorithm on $\{(\mathbf{x}_n, y_n)\}$ to get $q(y, \mathbf{x}) \approx P(y|\mathbf{x})$
2. for each new input $\mathbf{x}$, predict its class using $g_p(\mathbf{x})$

MetaCost Approach (Domingos, 1999)

1. use your favorite multiclass classification algorithm on **bootstrapped** $\{(\mathbf{x}_n, y_n)\}$ **and aggregate** the classifiers to get $q(y, \mathbf{x}) \approx P(y|\mathbf{x})$
2. for each given input $\mathbf{x}_n$, **relabel it to** y'_n using $g_q(\mathbf{x})$
3. run your favorite multiclass classification algorithm on **relabeled** $\{(\mathbf{x}_n, y'_n)\}$ to get final classifier g
4. for each new input $\mathbf{x}$, predict its class using $g(\mathbf{x})$

pros: any multiclass classification algorithm can be used

MetaCost on Semi-Real Data



(Domingos, 1999)

- some ‘artificial’ cost with UCI data
- MetaCost+C4.5: cost-sensitive
- C4.5: regular

not surprisingly,

considering the cost properly does help

Outline

Key Idea: Cost Transformation

(heuristic) **relabeling** useful in MetaCost: a more **principled** way?

Yes, by Connecting Cost Vector to Regular Costs!

$$\underbrace{\begin{pmatrix}1 & 0 & 1 & 2\end{pmatrix}}_{\mathbf{c}\text{ of interest}} \xrightarrow{\text{shift equivalence}} \underbrace{\begin{pmatrix}3 & 2 & 3 & 4\end{pmatrix}}_{\text{shifted cost}} = \underbrace{\begin{pmatrix}1 & 2 & 1 & 0\end{pmatrix}}_{\text{mixture weights } u_\ell} \cdot \underbrace{\begin{pmatrix}0 & 1 & 1 & 1\\ 1 & 0 & 1 & 1\\ 1 & 1 & 0 & 1\\ 1 & 1 & 1 & 0\end{pmatrix}}_{\text{regular costs}}$$

i.e. $\mathbf{x}$ with $\mathbf{c} = (1, 0, 1, 2)$ equivalent to

a weighted mixture $\{(\mathbf{x}, y, u)\} = \{(\mathbf{x}, 1, 1), (\mathbf{x}, 2, 2), (\mathbf{x}, 3, 1)\}$

cost equivalence (Lin, 2014): for any classifier h,

$$\mathbf{c}[h(\mathbf{x})] + \text{constant} = \sum_{\ell=1}^{K} u_\ell \, [\![\ell \neq h(\mathbf{x})]\!]$$

Meaning of Cost Equivalence

$$\mathbf{c}[h(\mathbf{x})] + \text{constant} = \sum_{\ell=1}^{K} u_\ell \, [\![\ell \neq h(\mathbf{x})]\!]$$

on one $(\mathbf{x}, y, \mathbf{c})$:

wrong prediction charged by
$\mathbf{c}[h(\mathbf{x})]$

on all $\{(\mathbf{x}, \ell, u_\ell)\}$:

wrong prediction charged by
total weighted classification error
of **relabeled data**

	$\min_h$ expected LHS	(original CSMC problem)
$=$	$\min_h$ expected RHS	(weighted classification when $u_\ell \geq 0$)

Calculation of u_ℓ

Smallest Non-Negative u_ℓ's (Lin, 2014)

when constant $= (K-1) \max_{1 \le k \le K} \mathbf{c}[k] - \sum_{k=1}^{K} \mathbf{c}[k]$,

$$u_\ell = \max_{1 \le k \le K} \mathbf{c}[k] - \mathbf{c}[\ell]$$

$$\text{e.g. } \underbrace{\begin{pmatrix} 1 & 0 & 1 & 2 \end{pmatrix}}_{\mathbf{c}\text{ of interest}} \rightarrow \underbrace{\begin{pmatrix} 1 & 2 & 1 & 0 \end{pmatrix}}_{\text{mixture weights } u_\ell}$$

- **largest $\mathbf{c}[\ell]$**: $u_\ell = 0$ (least preferred relabel)
- smallest $\mathbf{c}[\ell]$: $u_\ell =$ largest (**original label** & most preferred relabel)

ℓ's and u_ℓ's **embed the cost**

Data Space Expansion Approach

Data Space Expansion (DSE) Approach (Abe, 2004)

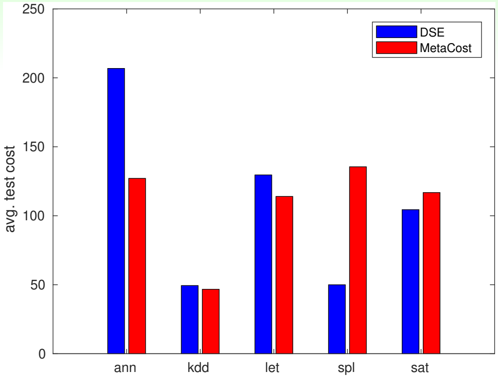
1. for each $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ and ℓ, let $u_{n,\ell} = \max_{1 \le k \le K} \mathbf{c}_n[k] - \mathbf{c}_n[\ell]$
2. apply your favorite multiclass classification algorithm on the weighted mixtures $\bigcup_{n=1}^{N} \{(\mathbf{x}_n, \ell, u_{n,\ell})\}_{\ell=1}^{K}$ to get $g(\mathbf{x})$
3. for each new input $\mathbf{x}$, predict its class using $g(\mathbf{x})$

- by **cost equivalence**,

  good g for new (weighted) regular classification problem
  $=$ good g for **original cost-sensitive classification problem**
- weighted regular classification: special case of CSMC
  but more easily solvable by, e.g., sampling + regular classification (Zadrozny, 2003)

pros: any multiclass classification algorithm can be used

DSE versus MetaCost on Semi-Real Data



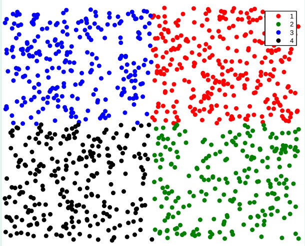
(Abe, 2004) some 'artificial' cost with UCI data

- use sampling + C4.5 for weighted regular classification

DSE **competitive to** MetaCost

Cons of DSE: Unavoidable Noise

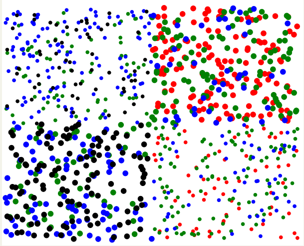
Original Cost-Sensitive Classification Problem



individual examples without noise

+ absolute cost =

New Regular Classification Problem



mixtures with relabeling noise

- cost embedded as weight + **noisy labels**
- new problem usually **harder** than original one

need robust multiclass classification algorithm to deal with noise

Key Idea: Design Robust Multiclass Algorithm

One-Versus-One: A Popular Classification Meta-Method

- for all different class pairs (i, j),
  1. take all examples $(\mathbf{x}_n, y_n)$
     - that $y_n = i$ or j **(original one-versus-one)**
     - that $u_{n,i} \neq u_{n,j}$ with the larger-u label and weight $|u_{n,i} - u_{n,j}|$ **(robust one-versus-one)**
  2. train a binary classifier $\hat{g}^{(i,j)}$ using those examples
- return $g(\mathbf{x})$ that predicts using the votes from $\hat{g}^{(i,j)}$

- **un-shifting** inside the meta-method to **remove noise**
- **robust step** makes it suitable for DSE

cost-sensitive one-versus-one: DSE + robust one-versus-one

Cost-Sensitive One-Versus-One (CSOVO)

Cost-Sensitive One-Versus-One (Lin, 2014)

- for all different class pairs (i, j),
  1. **robust one-versus-one** + calculate from $\mathbf{c}_n$: take all examples $(\mathbf{x}_n, y_n)$ that $\mathbf{c}_n[i] \neq \mathbf{c}_n[j]$ with smaller-$\mathbf{c}$ label and weight $u_n^{(i,j)} = |\mathbf{c}_n[i] - \mathbf{c}_n[j]|$
  2. train a binary classifier $\hat{g}^{(i,j)}$ using those examples
- return $g(\mathbf{x})$ that predicts using the votes from $\hat{g}^{(i,j)}$

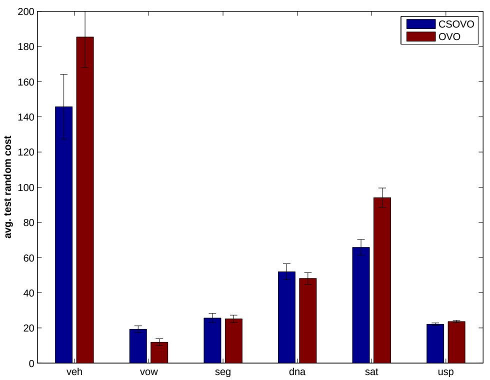
- comes with **good theoretical guarantee**:

$$\text{test cost of } g \leq 2 \sum_{i<j} \text{test cost of } \hat{g}^{(i,j)}$$

- sibling to Weighted All-Pairs (WAP) approach: even tighter guarantee (Beygelzimer, 2005) with more sophisticated construction of $u_n^{(i,j)}$

physical meaning: each $\hat{g}^{(i,j)}$ answers yes/no question 'prefer i or j?'

CSOVO on Semi-Real Data



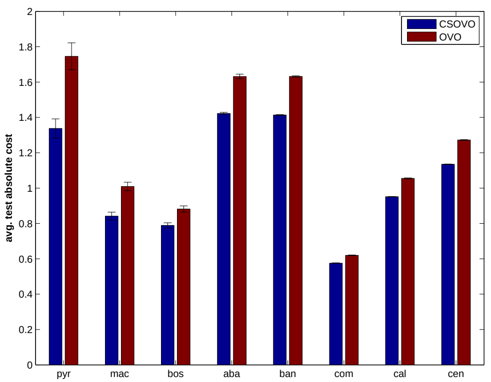
(Lin, 2014) some 'artificial' cost with UCI data

- CSOVO-SVM: cost-sensitive
- OVO-SVM: regular

not surprisingly again,

considering the cost properly does help

CSOVO for Ordinal Ranking



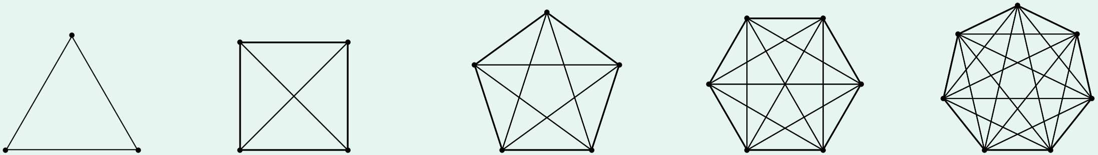
(Lin, 2014) absolute cost with benchmark ordinal ranking data

- CSOVO-SVM: cost-sensitive
- OVO-SVM: regular

CSOVO significantly better for ordinal ranking

Cons of CSOVO: Many Binary Classifiers

K classes $\xrightarrow{\text{CSOVO}}$ $\frac{K(K-1)}{2}$ binary classifiers



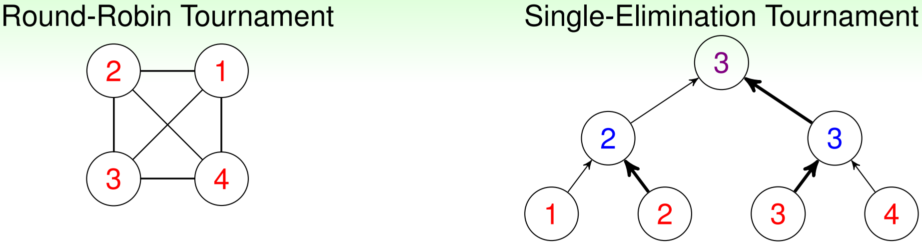
time-consuming in both

- training, especially with **many different $\mathbf{c}_n[i]$ and $\mathbf{c}_n[j]$**
- prediction

—parallization helps a bit, but **generally not feasible for large K**

CSOVO: a simple meta-method for median K only

Key Idea: OVO $\equiv$ Round-Robin Tournament

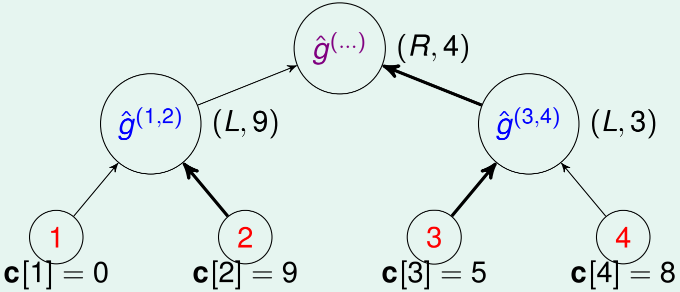


- prediction $\equiv$ deciding tournament winner for each $\mathbf{x}$
- (CS)OVO: $\frac{K(K-1)}{2}$ games for prediction (and hence training)
- single-elimination tournament (for $K = 2^\ell$):
  - $K - 1$ games for prediction via bottom-up: real-world
  - $\log_2 K$ games for prediction via top-down: computer-world :-)

next: single-elimination tournament for CSMC

Filter Tree (FT) Approach

Filter Tree (Beygelzimer, 2009) Training: from bottom to top



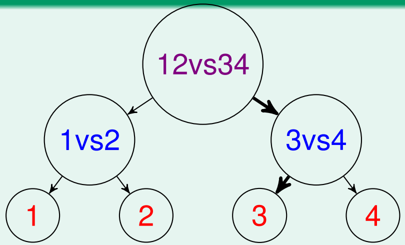
- $\hat{g}^{(1,2)}$ and $\hat{g}^{(3,4)}$ trained like CSOVO: smaller-**c** label and weight $u_n^{(i,j)} = |\mathbf{c}_n[i] - \mathbf{c}_n[j]|$
- $\hat{g}^{(\ldots)}$ trained with (k_L, k_R) filtered by sub-trees —smaller-**c** sub-tree direction and weight $u_n^{(\ldots)} = |\mathbf{c}_n[k_L] - \mathbf{c}_n[k_R]|$

FT: top classifiers aware of bottom-classifier mistakes

Pros and Cons of FT

Pros

- efficient:
  $O(K)$ training, $O(\log K)$ prediction
- **strong theoretical guarantee**:
  small-**regret** binary classifiers
  $\Longrightarrow$ small-**regret** CSMC classifier



Cons

- 'asymmetric' to labels: **non-trivial structural decision**
- 'hard' **sub-tree dependent** top-classification tasks

next: **other reductions** to (weighted) binary classification

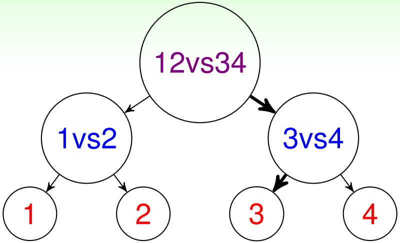
Other Approaches via Weighted Binary Classification

FT: with regret bound (Beygelzimer, 2009)

the lowest achievable cost within $\{1, 2\}$ or $\{3, 4\}$?

Divide&Conquer Tree (TREE): without regret bound (Beygelzimer, 2009)

the lowest ideal cost within $\{1, 2\}$ or $\{3, 4\}$?



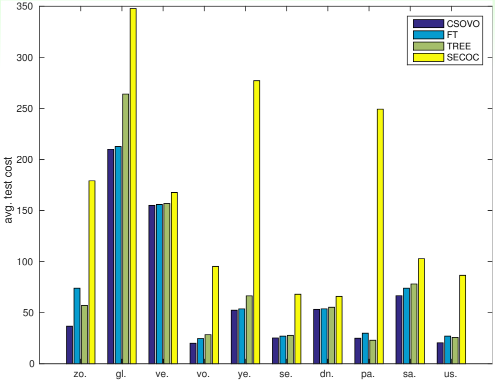
Sensitive Err. Correcting Output Code (SECOC): with regret bound (Langford, 2005)

$\mathbf{c}[1] + \mathbf{c}[3] + \mathbf{c}[4]$ greater than some θ?

training time:

$$\text{SECOC } (O(T \cdot K)) > \text{FT } (O(K)) \approx \text{TREE } (O(K))$$

Comparison of Reductions to Weighted Binary Classification



(Lin, 2014) couple all meta-methods with SVM

- round-robin tournament (CSOVO)
- single-elimination tournament (FT, TREE)
- error-correcting-code (SECOC)

CSOVO often among the best; FT somewhat competitive

Outline

Key Idea: Cost Estimator

Goal

a classifier $g(\mathbf{x})$ that pays a small cost $\mathbf{c}[g(\mathbf{x})]$ on future **unseen** example $(\mathbf{x}, y, \mathbf{c})$

if every $\mathbf{c}[k]$ known

optimal

$$g^*(\mathbf{x}) = \text{argmin}_{1 \le k \le K} \mathbf{c}[k]$$

if $r_k(\mathbf{x}) \approx \mathbf{c}[k]$ well

approximately good

$$g_r(\mathbf{x}) = \text{argmin}_{1 \le k \le K} r_k(\mathbf{x})$$

how to get cost estimator r_k? **regression**

Cost Estimator by Per-class Regression

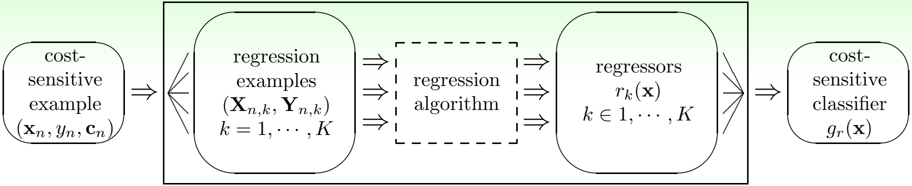
Given

N examples, each (input $\mathbf{x}_n$, label y_n, cost $\mathbf{c}_n$) $\in \mathcal{X} \times \{1, 2, \ldots, K\} \times R^K$

input	$\mathbf{c}_n[1]$	input	$\mathbf{c}_n[2]$	...	input	$\mathbf{c}_n[K]$
$\mathbf{x}_1$	0,	$\mathbf{x}_1$	2,		$\mathbf{x}_1$	1
$\mathbf{x}_2$	1,	$\mathbf{x}_2$	3,		$\mathbf{x}_2$	5
...						
$\mathbf{x}_N$	6,	$\mathbf{x}_N$	1,		$\mathbf{x}_N$	0
r_1		r_2			r_K	

want: $r_k(\mathbf{x}) \approx \mathbf{c}[k]$ for all future $(\mathbf{x}, y, \mathbf{c})$ and k

The Reduction-to-Regression Framework



1. encode: transform cost-sensitive examples $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to regression examples $(\mathbf{x}_{n,k}, Y_{n,k}) = (\mathbf{x}_n, \mathbf{c}_n[k])$
2. learn: use your favorite algorithm on regression examples to get estimators $r_k(\mathbf{x})$
3. decode: for each new input $\mathbf{x}$, predict its class using $g_r(\mathbf{x}) = \text{argmin}_{1 \le k \le K} r_k(\mathbf{x})$

the reduction-to-regression framework:

systematic & easy to implement

Theoretical Guarantees (1/2)

$$g_r(\mathbf{x}) = \operatorname*{argmin}_{1 \le k \le K} r_k(\mathbf{x})$$

Theorem (Absolute Loss Bound)

For any set of cost estimators $\{r_k\}_{k=1}^{K}$ *and for any example* $(\mathbf{x}, y, \mathbf{c})$ *with* $\mathbf{c}[y] = 0$*,*

$$\mathbf{c}[g_r(\mathbf{x})] \le \sum_{k=1}^{K} \Big| r_k(\mathbf{x}) - \mathbf{c}[k] \Big|.$$

low-cost classifier $\Longleftarrow$ accurate estimators

Theoretical Guarantees (2/2)

$$g_r(\mathbf{x}) = \operatorname*{argmin}_{1 \le k \le K} r_k(\mathbf{x})$$

Theorem (Squared Loss Bound)

For any set of cost estimators $\{r_k\}_{k=1}^{K}$ *and for any example* $(\mathbf{x}, y, \mathbf{c})$ *with* $\mathbf{c}[y] = 0$,

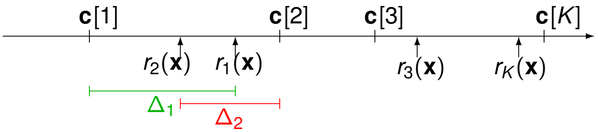
$$\mathbf{c}[g_r(\mathbf{x})] \le \sqrt{2 \sum_{k=1}^{K} (r_k(\mathbf{x}) - \mathbf{c}[k])^2}.$$

applies to common **least-square regression**

A Pictorial Proof

$$\mathbf{c}[g_r(\mathbf{x})] \le \sum_{k=1}^{K} \left| r_k(\mathbf{x}) - \mathbf{c}[k] \right|$$

- assume $\mathbf{c}$ ordered and not degenerate: $y = 1$; $0 = \mathbf{c}[1] < \mathbf{c}[2] \le \cdots \le \mathbf{c}[K]$
- assume mis-prediction $g_r(\mathbf{x}) = 2$: $r_2(\mathbf{x}) = \min_{1 \le k \le K} r_k(\mathbf{x}) \le r_1(\mathbf{x})$

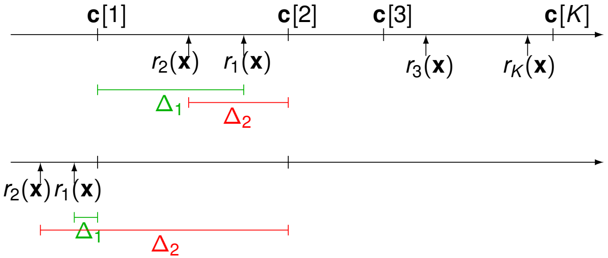


$$\mathbf{c}[2] - \underbrace{\mathbf{c}[1]}_{0} \le \left|\Delta_1\right| + \left|\Delta_2\right| \le \sum_{k=1}^{K} \left| r_k(\mathbf{x}) - \mathbf{c}[k] \right|$$

An Even Closer Look

let $\Delta_1 \equiv r_1(\mathbf{x}) - \mathbf{c}[1]$ and $\Delta_2 \equiv \mathbf{c}[2] - r_2(\mathbf{x})$

1. $\Delta_1 \geq 0$ and $\Delta_2 \geq 0$: $\mathbf{c}[2] \leq \Delta_1 + \Delta_2$
2. $\Delta_1 \leq 0$ and $\Delta_2 \geq 0$: $\mathbf{c}[2] \leq \Delta_2$
3. $\Delta_1 \geq 0$ and $\Delta_2 \leq 0$: $\mathbf{c}[2] \leq \Delta_1$



$$\mathbf{c}[2] \leq \max(\Delta_1, 0) + \max(\Delta_2, 0) \leq |\Delta_1| + |\Delta_2|$$

Tighter Bound with One-sided Loss

Define **one-sided loss** $\xi_k \equiv \max(\Delta_k, 0)$

$$\text{with} \quad \Delta_k \equiv \Big(r_k(\mathbf{x}) - \mathbf{c}[k]\Big) \text{ if } \mathbf{c}[k] = c_{\min} = 0$$

$$\Delta_k \equiv \Big(\mathbf{c}[k] - r_k(\mathbf{x})\Big) \text{ if } \mathbf{c}[k] \neq c_{\min}$$

Intuition

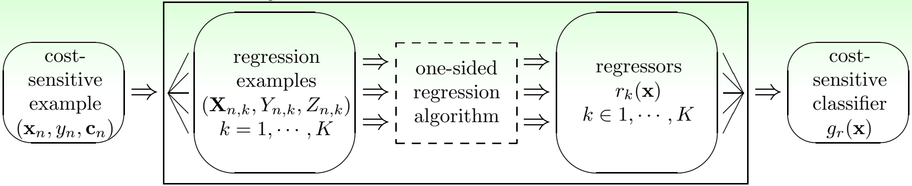
- $\mathbf{c}[k] = c_{\min}$: wish to have $r_k(\mathbf{x}) \leq \mathbf{c}[k]$
- $\mathbf{c}[k] \neq c_{\min}$: wish to have $r_k(\mathbf{x}) \geq \mathbf{c}[k]$

—**both wishes same as** $\Delta_k \leq 0 \Longleftrightarrow \xi_k = 0$

One-sided Loss Bound:

$$\mathbf{c}[g_r(\mathbf{x})] \leq \sum_{k=1}^{K} \xi_k \leq \sum_{k=1}^{K} \Big|\Delta_k\Big|$$

The Improved Reduction Framework



(Tu, 2010)

1. encode: transform cost-sensitive examples $(\mathbf{x}_n, y_n, \mathbf{c}_n)$ to
   one-sided regression examples $\left(\mathbf{x}_n^{(k)}, Y_n^{(k)}, Z_n^{(k)}\right) = (\mathbf{x}_n, \mathbf{c}_n[k], 2 [\![\mathbf{c}_n[k] = 0]\!] - 1)$
2. learn: use a one-sided regression algorithm to get estimators $r_k(\mathbf{x})$
3. decode: for each new input $\mathbf{x}$, predict its class using $g_r(\mathbf{x}) = \text{argmin}_{1 \le k \le K}\, r_k(\mathbf{x})$

the reduction-to-OSR framework:

need a good OSR algorithm

Regularized One-Sided Hyper-Linear Regression

Given

$$\left(\mathbf{x}_{n,k}, Y_{n,k}, Z_{n,k}\right) = \left(\mathbf{x}_n, \mathbf{c}_n[k], 2\left[\!\left[\mathbf{c}_n[k] = 0\right]\!\right] - 1\right)$$

Training Goal

all training $\xi_{n,k} = \max\left(\underbrace{Z_{n,k}\left(r_k(\mathbf{x}_{n,k}) - Y_{n,k}\right)}_{\Delta_{n,k}}, 0\right)$ small

—**will drop** k

$$\min_{\mathbf{w},b} \quad \frac{\lambda}{2}\langle \mathbf{w}, \mathbf{w}\rangle + \sum_{n=1}^{N} \xi_n$$

to get $\quad r_k(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x})\rangle + b$

One-Sided Support Vector Regression

Regularized One-Sided Hyper-Linear Regression

$$\min_{\mathbf{w},b} \quad \frac{\lambda}{2}\langle \mathbf{w}, \mathbf{w}\rangle + \sum_{n=1}^{N} \xi_n$$

$$\xi_n = \max\left(Z_n \cdot (r_k(\mathbf{x}_n) - Y_n), 0\right)$$

Standard Support Vector Regression

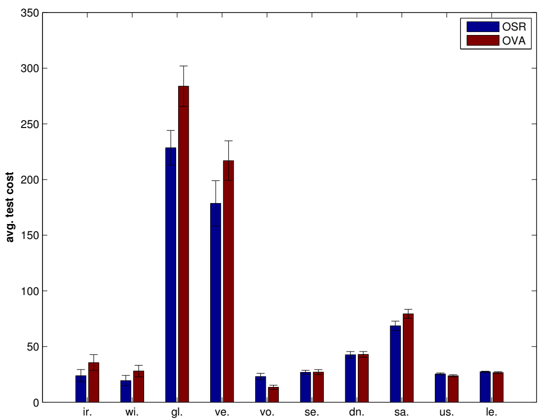
$$\min_{\mathbf{w},b} \quad \frac{1}{2C}\langle \mathbf{w}, \mathbf{w}\rangle + \sum_{n=1}^{N} (\xi_n + \xi_n^*)$$

$$\xi_n = \max\left(+1 \cdot (r_k(\mathbf{x}_n) - Y_n - \epsilon), 0\right)$$

$$\xi_n^* = \max\left(-1 \cdot (r_k(\mathbf{x}_n) - Y_n + \epsilon), 0\right)$$

OSR-SVM = SVR + $(\epsilon \leftarrow 0)$ + (keep ξ_n or ξ_n^* by Z_n)

OSR-SVM on Semi-Real Data

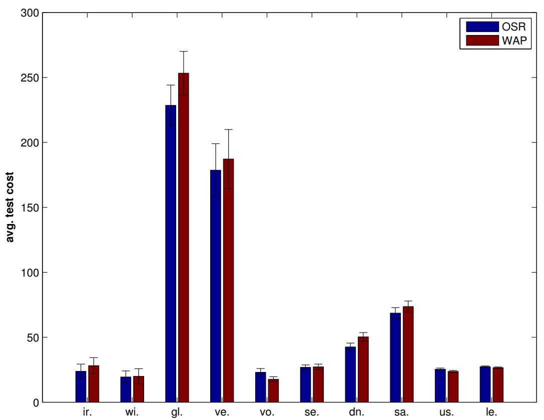


(Tu, 2010) some 'artificial' cost with UCI data

- OSR: cost-sensitive SVM
- OVA: regular one-versus-all SVM

OSR often significantly better than OVA

OSR versus WAP on Semi-Real Data



(Tu, 2010) some 'artificial' cost with UCI data

- OSR (per-class): $O(K)$ training, $O(K)$ prediction
- WAP $\approx$ CSOVO (pairwise): $O(K^2)$ training, $O(K^2)$ prediction

OSR faster and competitive performance

From OSR-SVM to AOSR-DNN

OSR-SVM

$$\min_{\mathbf{w},b} \quad \frac{\lambda}{2}\langle \mathbf{w}, \mathbf{w} \rangle + \sum_{n=1}^{N} \xi_n$$

with

$$r_k(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b$$

$$\xi_n = \max\left(Z_n \cdot (r_k(\mathbf{x}_n) - Y_n), 0\right)$$

Appro. OSR-DNN

$$\min_{\text{NNet}} \quad \text{regularizer} + \sum_{n=1}^{N} \delta_n$$

with

$$r_k(\mathbf{x}) = \text{NNet}(\mathbf{x})$$

$$\delta_n = \ln\left(1 + \exp\left(Z_n \cdot (r_k(\mathbf{x}_n) - Y_n)\right)\right)$$

AOSR-DNN (Chung, 2016a) = Deep Learning + OSR +

smoother upper bound $\delta_n \geq \xi_n$ because $\ln(1 + \exp(\bullet)) \geq \max(\bullet, 0)$

From AOSR-DNN to CSDNN

Cons of AOSR-DNN

c affects both classification **and feature-extraction** in DNN
but hard to do effective **cost-sensitive feature extraction**

idea 1: pre-training with **c**

- layer-wise pre-training with cost-sensitive autoencoders

  loss = reconstruction + AOSR

- CSDNN (Chung, 2016a) = AOSR-DNN + cost-sens. pre-training

idea 2: auxiliary cost-sensitive nodes

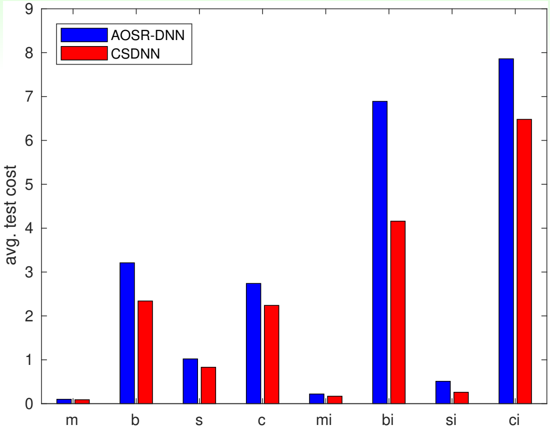
- auxiliary nodes to predict costs per layer

  loss = AOSR for classification + AOSR for auxiliary

- applies to any deep learning model (Chung, 2020)

CSDNN: world's **first successful CSMC deep learning model**

AOSR-DNN versus CSDNN

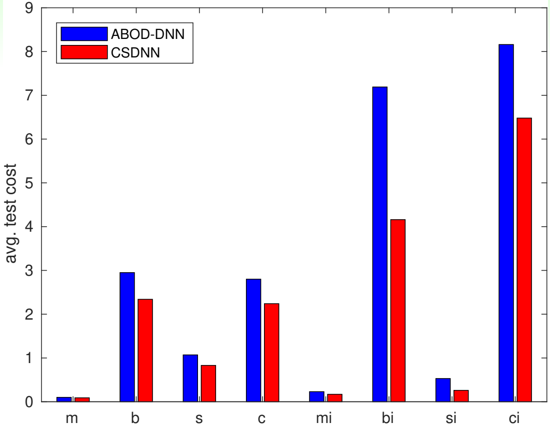


(Chung, 2016a)

- AOSR-DNN: cost-sensitive training
- CSDNN: AOSR-DNN + cost-sensitive feature extraction

CSDNN wins, justifying **cost-sensitive feature extraction**

ABOD-DNN versus CSDNN



(Chung, 2016a)

- ABOD-DNN: probability estimate + cost-sensitive prediction
- CSDNN: cost-sensitive training + cost-sensitive feature extraction

CSDNN still wins, hinting **difficulty of probability estimate** without cost-sensitive feature extraction

Outline

Which Fruit?

?

(image by Robert-Owen-Wahl from Pixabay)

apple

orange

strawberry

kiwi

(images by Pexels, PublicDomainPictures, 192635, Rob van der Meijden from Pixabay)

multiclass classification:
classify input (picture) to **one category** (label), **remember? :-)**

Which Fruits?

?: {apple, orange, kiwi}

(image by Michal Jarmoluk from Pixabay)

apple

orange

strawberry

kiwi

(images by Pexels, PublicDomainPictures, 192635, Rob van der Meijden from Pixabay)

multilabel classification:
classify input to **multiple (or no)** categories

Label Powerset: Multilabel Classification via Multiclass (Tsoumakas, 2007)

multiclass w/ $L = 4$ classes

4 possible outcomes
{a, o, s, k}

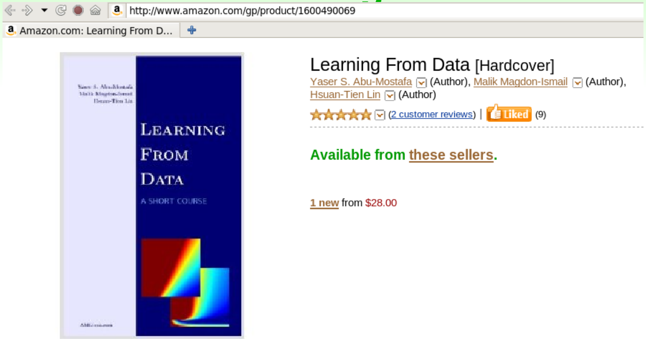
multilabel w/ $L = 4$ classes

$2^4 = 16$ **possible outcomes**
$2^{\{a, o, s, k\}}$
$\Updownarrow$
{ ϕ, a, o, s, k, ao, as, ak, os, ok, sk, aos, aok, ask, osk, aosk }

- **Label Powerset (LP)**: reduction to multiclass classification
- difficulties for large L:
  - **computation**: 2^L extended classes
  - **sparsity**: no or few example for some combinations

LP: feasible only for **small L**

What **Tags**?



?: {machine learning, ~~data structure~~, data mining, ~~object oriented programming~~, artificial intelligence, ~~compiler~~, ~~architecture~~, ~~chemistry~~, textbook, ~~children book~~, ~~. . . etc.~~ }

another **multilabel** classification problem:
tagging input to multiple categories

Binary Relevance: Multilabel Classification via Yes/No

binary classification

{yes, no}

multilabel w/ L classes: L yes/no questions

machine learning (Y), data structure (N), data mining (Y), OOP (N), AI (Y), compiler (N), architecture (N), chemistry (N), textbook (Y), children book (N), *etc.*

- **Binary Relevance (BR)**: reduction to **multiple isolated binary classification**
- disadvantages:
  - **isolation**—hidden relations not exploited (e.g. ML and DM highly correlated, ML subset of AI, textbook & children book disjoint)
  - **unbalanced**—few yes, many no

BR: simple (& strong) benchmark with known disadvantages

Multilabel Classification Setup

Given

N examples (input $\mathbf{x}_n$, labelset $\mathcal{Y}_n$) $\in \mathcal{X} \times 2^{\{1,2,\cdots L\}}$

- `fruits`: $\mathcal{X} =$ encoding(pictures), $\mathcal{Y}_n \subseteq \{1, 2, \cdots, 4\}$
- `tags`: $\mathcal{X} =$ encoding(merchandise), $\mathcal{Y}_n \subseteq \{1, 2, \cdots, L\}$

Goal

a multilabel classifier $g(\mathbf{x})$ that **closely predicts** the labelset $\mathcal{Y}$ associated with some **unseen** inputs $\mathbf{x}$ (by exploiting hidden relations/combinations between labels)

multilabel classification:
hot and important with many real-world applications

From Labelset to Coding View

	labelset	apple	orange	strawberry	**binary code**
	$\mathcal{Y}_1 = \{\text{o}\}$	0 (N)	1 (Y)	0 (N)	$\mathbf{y}_1 = [0, 1, 0]$
	$\mathcal{Y}_2 = \{\text{a}, \text{o}\}$	1 (Y)	1 (Y)	0 (N)	$\mathbf{y}_2 = [1, 1, 0]$
	$\mathcal{Y}_3 = \{\text{o}, \text{s}\}$	0 (N)	1 (Y)	1 (Y)	$\mathbf{y}_3 = [0, 1, 1]$
	$\mathcal{Y}_4 = \{\}$	0 (N)	0 (N)	0 (N)	$\mathbf{y}_4 = [0, 0, 0]$

(images by PublicDomainPictures, Narin Seandag, GiltonF, nihatyetkin from Pixabay)

subset $\mathcal{Y}$ of $2^{\{1,2,\cdots,L\}}$ $\Longleftrightarrow$ **length-L binary code $\mathbf{y}$**

LP Approach: What Performance Measure?

Goal: a classifier $g(\mathbf{x})$ that **closely predicts** the labelset $\mathcal{Y}$ (**code y**) associated w/ $\mathbf{x}$

LP Approach

1. encode: transform multilabel examples $(\mathbf{x}_n, \mathbf{y}_n)$ to multiclass examples $(\mathbf{x}_n, Y_n)$, where Y_n = binary number of $\mathbf{y}_n$

   $\mathbf{y} \to Y \mid [0,0,0] \to 0 \quad [0,0,1] \to 1 \quad [0,1,0] \to 2 \quad [0,1,1] \to 3$
2. learn: use any (regular) algorithm on multiclass examples to get classifier $\hat{g}(\mathbf{x})$
3. decode: for each new input $\mathbf{x}$, predict its code using

   $$g(\mathbf{x}) = \text{binary representation of } \hat{g}(\mathbf{x})$$

Measuring 'Closely Predict'

- **regular** multiclass algorithm: optimizes $[\![Y \neq \hat{g}(\mathbf{x})]\!]$
- LP: correspondingly optimizes $[\![\mathbf{y} \neq g(\mathbf{x})]\!]$, called **subset** $0/1$ **error**

subset $0/1$ error: a **strict** measure for multilabel classification

BR Approach: What Performance Measure?

Goal: a classifier $g(\mathbf{x})$ that **closely predicts** the labelset $\mathcal{Y}$ (**code y**) associated w/ $\mathbf{x}$

BR Approach

1. encode: transform multilabel examples $(\mathbf{x}_n, \mathbf{y}_n)$ to binary examples $(\mathbf{x}_n, \mathbf{y}_n[\ell])$
2. learn: use any algorithm on binary classification examples to get classifier $\hat{g}_\ell(\mathbf{x})$
3. decode: for each new input $\mathbf{x}$, predict its code using

$$g(\mathbf{x}) = [\hat{g}_1(\mathbf{x}), \hat{g}_2(\mathbf{x}), \ldots, \hat{g}_L(\mathbf{x})]$$

Measuring 'Closely Predict'

- **regular** binary classification algorithm: optimizes $[\![\mathbf{y}[\ell] \neq \hat{g}_\ell(\mathbf{x})]\!]$
- BR: correspondingly optimizes $\frac{1}{L}|g(\mathbf{x}) \triangle \mathcal{Y}|$, called **Hamming error**

Hamming error: a **simple** measure for multilabel classification

Evaluating Multilabel Classifiers

Different Approaches Optimizes Different Measure

- LP: subset 0/1 error
- BR: Hamming error

Different (Assumed) Dependence Associated with Different Measure

(Dembczyński, 2012)

- strong conditional dependence: subset 0/1 error (need 'joint' optimization)
- no conditional dependence: Hamming error (can use 'marginal' optimization)

Different Applications Needs Different Measure

- information retrieval: F1 score (harmonic mean of precision & recall)
- tag recommendation: ranking error

Cost-Sensitive Multilabel Classification (CSML):
design one approach for 'any' measure, just like CSMC

Setup: Cost-Sensitive Multilabel Classification (CSML)

Given

N examples, each (input $\mathbf{x}_n$, code $\mathbf{y}_n$) $\in \mathcal{X} \times \{1, 2, \ldots, L\}$

and cost function $\mathcal{C} \in \mathbb{R}^{2^L \times 2^L}$ with $\mathcal{C}(\mathbf{y}, \mathbf{y}) = 0 = \min_{\mathbf{k} \in \{0,1\}^L} \mathcal{C}(\mathbf{y}, \mathbf{k})$

Goal

a classifier $g(\mathbf{x})$ that pays a small cost $\mathcal{C}(\mathbf{y}, g(\mathbf{x}))$ on future **unseen** example $(\mathbf{x}, \mathbf{y})$

- called **instance-based** CSML: each instance evaluated separately
  —more complicated to solve other kinds of CSML (Hsieh, 2018)
- possible extension to **example-dependent costs** $\mathcal{C}_{\mathbf{x}}$ like CSMC

will focus on **'class'-dependent instance-based CSML**

Outline

Approximate Bayes-Optimal Decision Revisited for CSML

if $q(\mathbf{x}, \mathbf{y}) \approx P(\mathbf{y}|\mathbf{x})$ well

AOBD for CSML

$$\text{approximately good } g_q(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{k}\in\{0,1\}^L} \sum_{\mathbf{y}\in\{0,1\}^L} q(\mathbf{x}, \mathbf{y})\mathcal{C}(\mathbf{y}, \mathbf{k})$$

difficulty of directly using AOBD

- difficulty in probability estimation: 2^L outputs per $\mathbf{x}$ for $q(\mathbf{x}, \mathbf{y})$
- difficulty in cost calculation: 2^L possible $\mathbf{y}$ in $\sum$ to compute per $\mathbf{k}$
- difficulty in inference: argmin over 2^L possible candidates $\mathbf{k}$

ABOD: **even harder** for CSML than CSMC

Key Idea: Estimate Probability with Decomposition

$$\begin{array}{ccccccccc} P(\mathbf{y} \mid \mathbf{x}) & = & P(\mathbf{y}[1] \mid \mathbf{x}) & \cdot & P(\mathbf{y}[2] \mid \mathbf{x}, \mathbf{y}[1]) & \cdot & P(\mathbf{y}[3] \mid \mathbf{x}, \mathbf{y}[1], \mathbf{y}[2]) & \cdot & \cdots \\ \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \\ q(\mathbf{x}, \mathbf{y}) & & q_1(\mathbf{x}) & & q_2(\mathbf{x}, \mathbf{y}[1]) & & q_3(\mathbf{x}, \mathbf{y}[1], \mathbf{y}[2]) & & \cdots \end{array}$$

- $q_\ell(\mathbf{x}, \mathbf{y}[1], \ldots, \mathbf{y}[\ell - 1])$: estimates $P\Big(\mathbf{y}[\ell] = 1 \mid \mathbf{x}, \mathbf{y}[1, \ldots, (\ell - 1)]\Big)$
  —learned with your favorite estimation algorithm, such as **logistic regression**
- if each q_ℓ accurate, multiplied q also accurate

$$q(\mathbf{x}, \mathbf{y}) = \prod_{\ell=1}^{L} q_\ell^{\mathbf{y}[\ell]} (1 - q_\ell)^{(1 - \mathbf{y}[\ell])}$$

Probabilistic Classifier Chain (PCC) (Dembczyński, 2010):
estimate q_ℓ's to conquer difficulty in probability estimation

Key Idea: Sparsify $q(\mathbf{x}, \mathbf{y})$ with Representative $\mathbf{y}$'s

conjecture: $P(\mathbf{y}|\mathbf{x})$ usually small for most $\mathbf{y}$'s, hence many $q(\mathbf{x}, \mathbf{y})$ also small

draw typical $\mathbf{y}$ from $q(\mathbf{x}, \mathbf{y})$

(Dembczyński, 2011)

- Monte Carlo sampling from q_1, q_2, ..., q_ℓ sequentially

keep most probable $\mathbf{y}[1, \ldots, \ell]$

(Kumar, 2013)

- beam search: keep B most probable predictions

calculate **necessary costs/statistics with representative y**
to conquer difficulty in cost calculation

Key Idea: Derive Efficient Inference Rule for Specific $\mathcal{C}$

AOBD for CSML: approximately good $g_q(\mathbf{x}) = \underset{\mathbf{k} \in \{0,1\}^L}{\text{argmin}} \sum_{\text{representative } \mathbf{y}} q(\mathbf{x}, \mathbf{y}) \mathcal{C}(\mathbf{y}, \mathbf{k})$

—even with representative $\mathbf{y}$, still exponentially many $\mathbf{k}$

some $\mathcal{C}$ allows efficient inference

- subset 0/1 error: $\mathcal{C}(\mathbf{y}, \mathbf{k}) = 0$ iff $\mathbf{y} = \mathbf{k}$ & 1 otherwise

  optimal $\mathbf{k} = \text{argmax}_{\mathbf{y}}\, q(\mathbf{x}, \mathbf{y})$ over representative $\mathbf{y}$

- Hamming error: $\mathcal{C}(\mathbf{y}, \mathbf{k}) = \frac{1}{L} \sum_{\ell=1}^{L} [\![\mathbf{y}[\ell] \neq \mathbf{k}[\ell]]\!]$

  optimal $\mathbf{k}[\ell] =$ majority bit of $\mathbf{y}[\ell]$ over representative $\mathbf{y}$

AOBD for CSML: generally **restricted to specific $\mathcal{C}$**
where difficulty in inference can be resolved

Mini-Summary of Key Ideas

AOBD for CSML: if $q(\mathbf{x}, \mathbf{y}) \approx P(\mathbf{y}|\mathbf{x})$ well

$$\text{approximately good } g_q(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{k} \in \{0,1\}^L} \sum_{\mathbf{y} \in \{0,1\}^L} q(\mathbf{x}, \mathbf{y}) \mathcal{C}(\mathbf{y}, \mathbf{k})$$

difficulty of directly using AOBD revisited

- difficulty in probability estimation: 2^L outputs per $\mathbf{x}$ for $q(\mathbf{x}, \mathbf{y})$
- difficulty in cost calculation: 2^L possible $\mathbf{y}$ in $\sum$ to compute per $\mathbf{k}$
- difficulty in inference: argmin over 2^L possible candidates $\mathbf{k}$

corresponding key ideas

- estimate probability with decomposition
- sparsify probability estimation with representative $\mathbf{y}$'s
- derive efficient inference rule for specific $\mathcal{C}$

next: **concrete approaches** that combine key ideas

Putting It All Together: PCC for Subset 0/1 Error

- training: calculate $q_1, q_2, \ldots, q_L$,
  where q_ℓ learned with extended inputs $(\mathbf{x}_n, \mathbf{y}_n[1, \ldots, \ell-1])$ and outputs $\mathbf{y}_n[\ell]$
- inference: for each $\mathbf{x}$,
  - get B most representative $\mathbf{y}$ by beam search (Kumar, 2013)
  - return $g(\mathbf{x}) = \underset{\text{representative } \mathbf{y}}{\text{argmax}} \; q(\mathbf{x}, \mathbf{y})$

Cons of PCC

- 'assymetric' to labels: **non-trivial structural decision** of label order
  —often coupled with uniform aggregation (Ensemble PCC) to improve performance
- somewhat time consuming during inference

special case of $B = 1$:
a classic approach called Classifier Chain (CC) (Read, 2009)

Putting It All Together: PCC for F1 Score

- training: calculate $q_1, q_2, \ldots, q_L$,
  where q_ℓ learned with extended inputs $(\mathbf{x}_n, \mathbf{y}_n[1, \ldots, \ell - 1])$ and outputs $\mathbf{y}_n[\ell]$
- inference: for each $\mathbf{x}$,
  - get B most representative $\mathbf{y}$ by **sampling** (Dembczyński, 2011)
  - estimate necessary statistics

$$\delta_{\ell,k} = \mathbb{E}\Big\{\mathbf{y}[\ell] \text{ given } |\mathbf{y}| = k\Big\}$$

  - return $g(\mathbf{x})$ with exact inference from $\delta_{\ell,k}$ using $O(L^3)$ computation

strength depends on whether $\delta_{\ell,k}$ estimated well enough
with probability estimation + representative sampling

Mini-Summary of the PCC Family

with Efficient Inference Rules

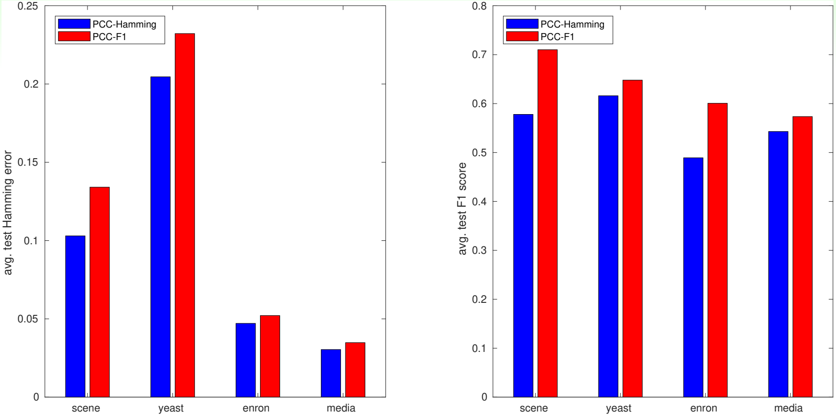
measure	inference rule
subset 0/1	mode of q
Hamming	threshold of 'marginal' q
ranking	sorted 'marginal' q
$\mathcal{C}(\mathbf{y}, \mathbf{k}) = -\frac{2\|\mathbf{y} \cap \mathbf{k}\|}{\|\mathbf{y}\| + \|\mathbf{k}\|}$: F1	statistics $\delta_{\ell,k}$ from q

without Efficient Inference Rules

measure	equation
accuracy	$\mathcal{C}(\mathbf{y}, \mathbf{k}) = -\frac{\|\mathbf{y} \cap \mathbf{k}\|}{\|\mathbf{y} \cup \mathbf{k}\|}$
multi. objective combination	$\mathcal{C}(\mathbf{y}, \mathbf{k}) = \mathcal{C}_1(\mathbf{y}, \mathbf{k}) + \mathcal{C}_2(\mathbf{y}, \mathbf{k})$

PCC: CSML approach '**in principle**'

Cost-Sensitivity of PCC



(Dembczyński, 2011) cost-sensitive behavior:

PCC-Hamming better (↓) for Hamming; PCC-F1 better (↑) for F1

Outline

A Naïve CSML Approach: Cost-Sensitive Label Powerset

Label Powerset (LP) Approach

$$\text{multilabel} \xrightarrow{\text{regular cost}} \text{multiclass}$$

Cost-Sensitive LP (CSLP)

$$\text{CSML} \xrightarrow{\text{cost function } c} \text{CSMC}$$

Cons of CSLP

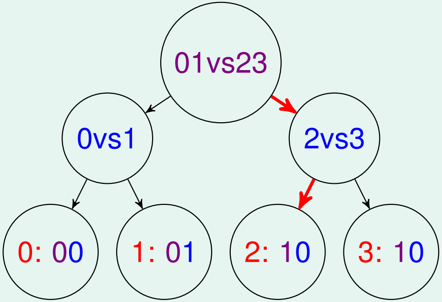
- **complexity**, just like LP,

		conquered by
CSLP + ABOD	: $O(2^L)$ estimates	decomposition + special inference
CSLP + CSOVO	: $O(2^L \cdot 2^L)$ classifiers	sampling + efficient decoding (Yang, 2018)
CSLP + FT	: $O(2^L)$ internal nodes	**sampling + node sharing** (Li, 2014)

next: CSLP + FT

CSLP + FT for Prediction

(CS)LP: Y = binary number of **y**

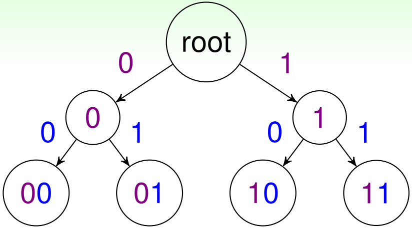


- with 'binary number encoding' (proper ordering):

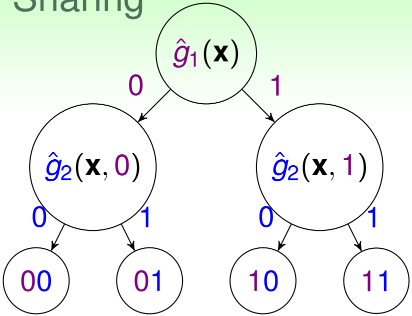
  ℓ-th layer nodes (classifier) $\Leftrightarrow$ ℓ-th label
- FT: $O(\log K)$ prediction, $O(K)$ training
  - $\log_2(2^L) = L$ **predictions only :-)**
  - still $O(2^L)$ training complexity
  - actually, $2^L - 1$ internal nodes

next: representing $2^L - 1$ internal nodes efficiently

Key Idea: Classifier Sharing



$\Longrightarrow$



$2^L - 1$ nodes $\Longrightarrow$ L classifiers

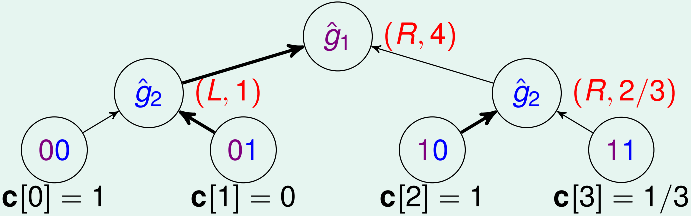
- root node $\hat{g}_1(\mathbf{x})$: just like BR on 1-st label
- 2-nd layer nodes 'shared' in $\hat{g}_2(\mathbf{x}, \tilde{\mathbf{y}}[1])$, where $\tilde{\mathbf{y}}[1] = \hat{g}_1(\mathbf{x})$

CSLP + FT similar to PCC in prediction

PCC	$q_1(\mathbf{x})$	$q_2(\mathbf{x}, \tilde{\mathbf{y}}[1])$	$q_3(\mathbf{x}, \tilde{\mathbf{y}}[1], \tilde{\mathbf{y}}[2])$	$\cdots$
CSLP + FT	$\hat{g}_1(\mathbf{x})$	$\hat{g}_2(\mathbf{x}, \tilde{\mathbf{y}}[1])$	$\hat{g}_3(\mathbf{x}, \tilde{\mathbf{y}}[1], \tilde{\mathbf{y}}[2])$	$\cdots$

CSLP + FP Training

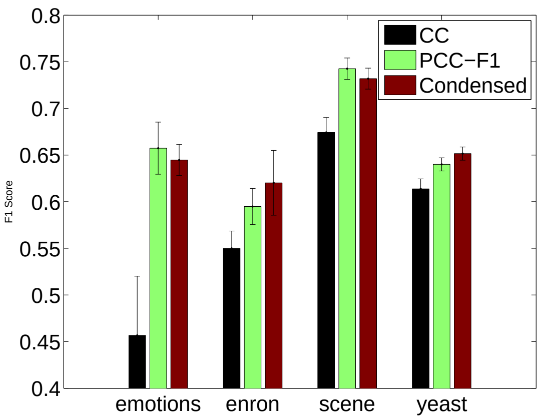
even with classifier sharing, $2^L - 1$ weighted binary examples per $(\mathbf{x}_n, \mathbf{y}_n)$ in FT



- not all binary examples relevant to training $\hat{g}_\ell$
  —prediction goes through **one path** anyway
- Condensed FT (CFT) (Li, 2014) :
  keeping only those examples **near prediction path** for training $\hat{g}_\ell$

CFT = CSLP + FT
+ Proper Ordering + Classifier Sharing + Example Sampling

CFT versus PCC on F1 Score



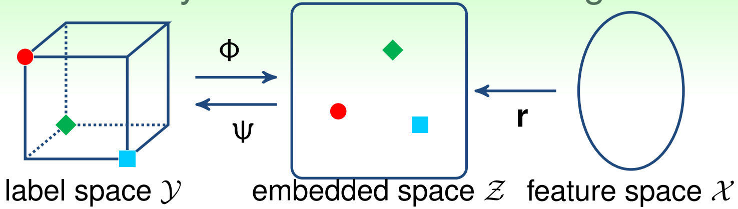
(Li, 2014)

- CFT certainly better than CC
- CFT can be better than PCC

CFT competitive within 'chaining approaches' for CSML

Outline

Key Idea: Label Embedding



Training Stage

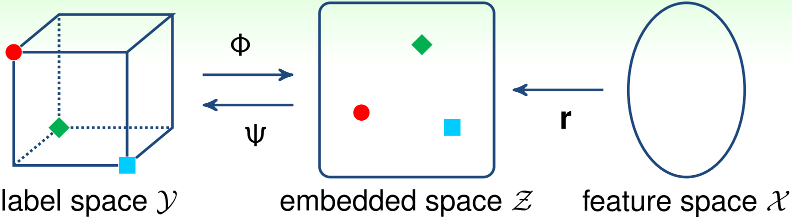
- embedding function Φ: label vector $\mathbf{y} \to$ embedded vector $\mathbf{z}$
- learn a regressor $\mathbf{r}$ from $\{(\mathbf{x}_n, \mathbf{z}_n)\}_{n=1}^{N}$

Predicting Stage

- for testing instance $\mathbf{x}$, predicted embedded vector $\tilde{\mathbf{z}} = \mathbf{r}(\mathbf{x})$
- decoding function Ψ: $\tilde{\mathbf{z}} \to$ predicted label vector $\tilde{\mathbf{y}}$

label embedding: popular for extracting **joint** information of labels

Cost-Sensitive Label Embedding

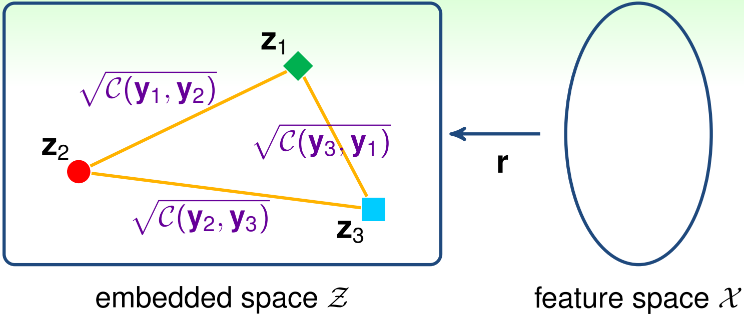


Existing Works

- **label embedding**: PLST (Tai, 2012), CPLST (Chen, 2012), FaIE (Lin, 2014), RAKEL (Tsoumakas 2007), etc.
- **cost-sensitivity**: CFT (Li, 2014), PCC (Dembczyński, 2010), etc.
- **cost-sensitivity + label embedding**: this work

Cost-Sensitive Label Embedding: considering $\mathcal{C}$ in Φ and Ψ

Cost-Sensitive Encoding

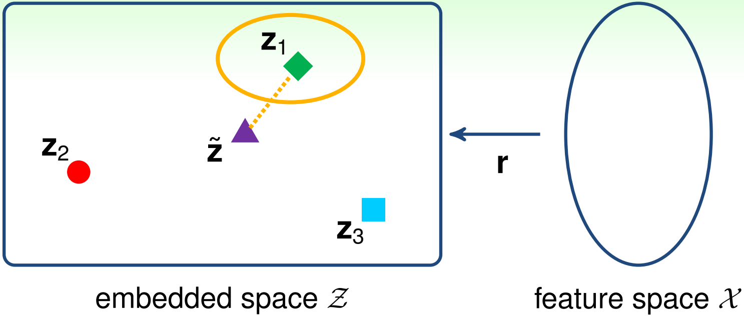


Training Stage

- distances between embedded vectors $\Leftrightarrow$ cost information
- larger distance $d(\mathbf{z}_i, \mathbf{z}_j) \Leftrightarrow$ higher cost $\mathcal{C}(\mathbf{y}_i, \mathbf{y}_j)$

$d(\mathbf{z}_i, \mathbf{z}_j) \approx \sqrt{\mathcal{C}(\mathbf{y}_i, \mathbf{y}_j)}$: by multidimensional scaling (Huang, 2017) or deep learning (Chiu, 2018)

Cost-Sensitive Decoding



Predicting Stage

- for testing instance $\mathbf{x}$, predicted embedded vector $\tilde{\mathbf{z}} = \mathbf{r}(\mathbf{x})$
- find nearest embedded vector $\mathbf{z}_q$ of $\tilde{\mathbf{z}}$

cost-sensitive decoding: $g(\mathbf{x})$ = corresponding $\mathbf{y}_q$

Theoretical Explanation

Cost Bound Theorem (Huang, 2017)

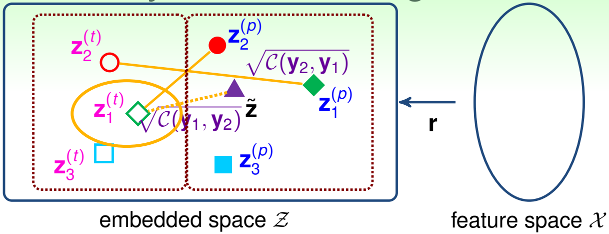
$$\mathcal{C}(\mathbf{y}, \tilde{\mathbf{y}}) \leq 5\Big(\underbrace{\left(d(\mathbf{z}, \mathbf{z}_q) - \sqrt{\mathcal{C}(\mathbf{y}, \mathbf{y}_q)}\right)^2}_{\text{embedding error}} + \underbrace{\|\mathbf{z} - \mathbf{r}(\mathbf{x})\|^2}_{\text{regression error}} \Big)$$

Optimization

- embedding error $\rightarrow$ multidimensional scaling
- regression error $\rightarrow$ any regression $\mathbf{r}$

challenge: **asymmetric cost function** vs. **symmetric distance**?
i.e. $\mathcal{C}(\mathbf{y}_i, \mathbf{y}_j) \neq \mathcal{C}(\mathbf{y}_j, \mathbf{y}_i)$ vs. $d(\mathbf{z}_i, \mathbf{z}_j)$

Key Idea: Mirroring Trick



- two roles of $\mathbf{y}_i$: **ground truth role $\mathbf{y}_i^{(t)}$** and **prediction role $\mathbf{y}_i^{(p)}$**
  - $\sqrt{\mathcal{C}(\mathbf{y}_i, \mathbf{y}_j)} \Rightarrow$ predict $\mathbf{y}_i$ as $\mathbf{y}_j \Rightarrow$ for $\mathbf{z}_i^{(t)}$ and $\mathbf{z}_j^{(p)}$
  - $\sqrt{\mathcal{C}(\mathbf{y}_j, \mathbf{y}_i)} \Rightarrow$ predict $\mathbf{y}_j$ as $\mathbf{y}_i \Rightarrow$ for $\mathbf{z}_i^{(p)}$ and $\mathbf{z}_j^{(t)}$
- learn **regression function r** from $\mathbf{z}_i^{(p)}, \mathbf{z}_2^{(p)}, \ldots, \mathbf{z}_L^{(p)}$
- find **nearest embedded vector** of $\tilde{\mathbf{z}}$ from $\mathbf{z}_1^{(t)}, \mathbf{z}_2^{(t)}, \ldots, \mathbf{z}_L^{(t)}$

mirroring trick: handle asymmetric cost with **embedding flexibility**

Cost-Sensitive Label Embedding with Multidimensional Scaling (CLEMS)

training stage of CLEMS (Huang, 2017)

- given training instances $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ and cost function $\mathcal{C}$
- apply mirroring trick to set up $\mathbf{z}_n^{(t)}$ and $\mathbf{z}_n^{(p)}$ for label vector $\mathbf{y}_n$
- compute embedding function $\Phi\colon \mathbf{y}_n \to \mathbf{z}_n^{(p)}$ by multidimensional scaling such that $d(\mathbf{z}_m^{(t)}, \mathbf{z}_n^{(p)}) \approx \sqrt{\mathcal{C}(y_n, y_m)}$
- learn a regression function $\mathbf{r}$ from $\{(\mathbf{x}_n, \mathbf{z}_n^{(p)} = \Phi(\mathbf{y}_n))\}_{n=1}^{N}$

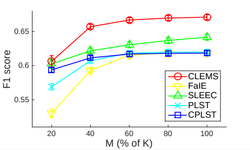
predicting stage of CLEMS

- given the testing instance $\mathbf{x}$
- obtain the predicted embedded vector by $\tilde{\mathbf{z}} = \mathbf{r}(\mathbf{x})$
- prediction $\tilde{\mathbf{y}} = \Psi(\tilde{\mathbf{z}}) = \Phi^{-1}(\text{nearest neighbor}) = \Phi^{-1}\left(\operatorname{argmin} d(\mathbf{z}_n^{(t)}, \tilde{\mathbf{z}})\right)$

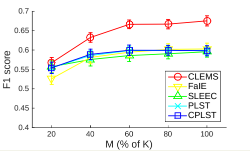
minor details: embed **subset of**, rather than 'all', $\{0, 1\}^L$ for efficiency

Comparison with Label Embedding Approaches

F1 score (↑)

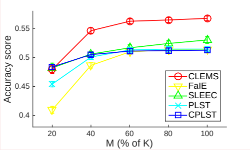


yeast

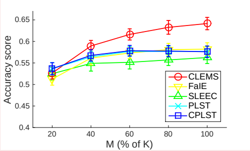


birds

Accuracy score (↑)

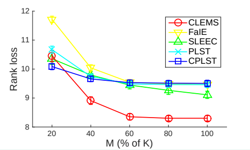


yeast

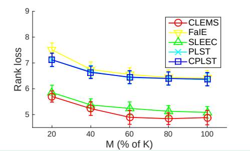


birds

Rank loss (↓)



yeast



birds

CLEMS: best label embedding approach across different criteria

Comparison with Cost-Sensitive Algorithms

data	F1 score (↑)			Accuracy score (↑)			Rank loss (↓)		
	CLEMS	CFT	PCC	CLEMS	CFT	PCC	CLEMS	CFT	PCC
emot.	**0.676**	0.640	0.643	**0.589**	0.557	–	1.484	1.563	**1.467**
scene	**0.770**	0.703	0.745	**0.760**	0.656	–	0.672	0.723	**0.645**
yeast	**0.671**	0.649	0.614	**0.568**	0.543	–	**8.302**	8.566	8.469
birds	**0.677**	0.601	0.636	**0.642**	0.586	–	4.886	4.908	**3.660**
med.	**0.814**	0.635	0.573	**0.786**	0.613	–	5.170	5.811	**4.234**
enron	**0.606**	0.557	0.542	**0.491**	0.448	–	29.40	26.64	**25.11**
lang.	**0.375**	0.168	0.247	**0.327**	0.164	–	31.03	34.16	**19.11**
flag	**0.731**	0.692	0.706	**0.615**	0.588	–	2.930	3.075	**2.857**
slash	**0.568**	0.429	0.503	**0.538**	0.402	–	4.986	5.677	**4.472**
CAL.	**0.419**	0.371	0.391	**0.273**	0.237	–	1247	1120	**993**
arts	**0.492**	0.334	0.349	**0.451**	0.281	–	9.865	10.07	**8.467**
EUR.	**0.670**	0.456	0.483	**0.650**	0.450	–	89.52	129.5	**43.28**

- **generality for CSML**: CLEMS $=$ CFT $>$ PCC
- **performance**: CLEMS $\approx$ PCC $>$ CFT
- **speed**: CLEMS $\approx$ PCC $>$ CFT

CLEMS: very **competitive** for CSML

Work on CSMC

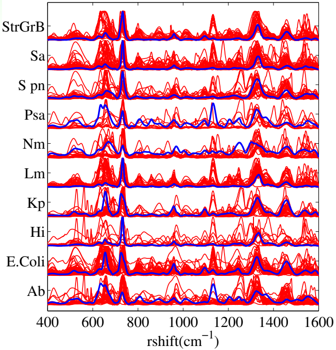
	binary	multiclass
regular	well-studied	well-studied
cost-sensitive	known (Zadrozny, 2003)	past 10 years (our works, among others)

selected works of ours

- cost-sensitive SVM (Tu, ICML '10) via one-sided regression
- cost-sensitive one-versus-one (Lin, ACML '14)
- cost-sensitive deep learning (Chung, IJCAI '16) via one-sided regression

why are people not
using those cool ML works for their applications?

Issue 1: Where Does Cost Come From?



automatic classification from spectrum to bacterium (Jan, 2011)

are all mis-prediction costs equal?

A Real Medical Application: Bacteria Classification

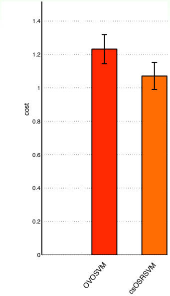
The Problem

- Gram-positive as Gram-positive: small cost
  Gram-positive as Gram-negative: **big cost**
- cost matrix averaged from two doctors:

	Ab	Ecoli	HI	KP	LM	Nm	Psa	Spn	Sa	GBS
Ab	0	1	10	7	9	9	5	8	9	1
Ecoli	3	0	10	8	10	10	5	10	10	2
HI	10	10	0	3	2	2	10	1	2	10
KP	7	7	3	0	4	4	6	3	3	8
LM	8	8	2	4	0	5	8	2	1	8
Nm	3	10	9	8	6	0	8	3	6	7
Psa	7	8	10	9	9	7	0	8	9	5
Spn	6	10	7	7	4	4	9	0	4	7
Sa	7	10	6	5	1	3	9	2	0	7
GBS	2	5	10	9	8	6	5	6	8	0

issue 2: is cost-sensitive classification really useful?

Cost-Sensitive vs. Traditional on Bacteria Data

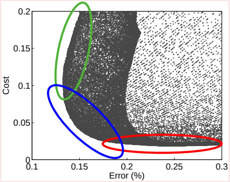


(Jan, 2011)

cost-sensitive better than traditional;
but why are people **still not**
using those cool ML works for their applications? :-)

Issue 3: Error Rate of Cost-Sensitive Classifiers

The Problem



- cost-sensitive classifier: low cost but high error rate
- traditional classifier: low error rate but high cost
- how can we get the blue classifiers?: low error rate and low cost

cost-and-error-sensitive:
more suitable for real-world medical needs

Improved Classifier for Both Cost and Error

(Jan, 2012)

Cost

iris	≈
wine	≈
glass	≈
vehicle	≈
vowel	◯
segment	◯
dna	◯
satimage	≈
usps	◯
zoo	◯
splice	≈
ecoli	≈
soybean	≈

Error

iris	◯
wine	◯
glass	◯
vehicle	◯
vowel	◯
segment	◯
dna	◯
satimage	◯
usps	◯
zoo	◯
splice	◯
ecoli	◯
soybean	◯

now, are people using those cool ML works for their applications? :-)

Lessons Learned from CSMC Research in Applications

?

bird flu

cold-infected

healthy

1. more realistic (generic) in academia $\neq$ more realistic (feasible) in application
   e.g. the 'cost' of inputting a cost matrix? :-)
2. cross-domain collaboration important
   e.g. getting the 'cost matrix' from domain experts
3. not easy to win human trust
   —humans are somewhat multi-objective

important yet **challenging**
to use CSMC/CSML in practical applications

Summary

- **cost-sensitive multiclass classification**: **class/example-dependent**
  - Bayesian: MetaCost (Domingos, 1999)
  - non-Bayesian: Data Space Expansion (Abe, 2004) (to multiclass), Cost-Sensitive One-Versus-One (Lin, 2014), Filter Tree (Beygelzimer, 2009), . . . (to binary), One-Sided Regression (Tu, 2010) (to regression)

—some SVM implementations here:

http://www.csie.ntu.edu.tw/~htlin/program/cssvm/

- **cost-sensitive multilabel classification**:
  - Bayesian: PCC (Dembczyński, 2010)
  - non-Bayesian: Condensed Filter Tree (Li, 2014) (to binary), CLEMS (Huang, 2017) (to regression)
- **application story**:
  - cost-and-error-sensitive learning for bacteria classification (Jan, 2012)

Thank you. Questions?