Efficient Multi-class/ Multilabel Classification using Tree Structures

Outline

1. Introduction
2. HOMER
    1. Balanced k Means
    2. Performance
    3. Discussion
3. Label Embedding Trees
    1. Performance
    2. Discussion
4. Conclusion

1. Introduction

- Problem 1:
  - many examples, many labels, multiple labels per example → but low label density
  - e.g. text categorization, protein function classification
- Problem 2:
  - even more examples, even more labels, many features → but only one label per example
  - e.g. image annotation, web advertising

2. HOMER

- large set of labels L → tree- shaped hierarchy
- nodes contain:
  - similar labels $L_n \subseteq L$ (disjunction of labels in L_n = meta-label)
  - multilabel classifier (predicts meta-labels of children)
  - examples labeled with at least one label of L_n (used to train classifiers)

2. HOMER

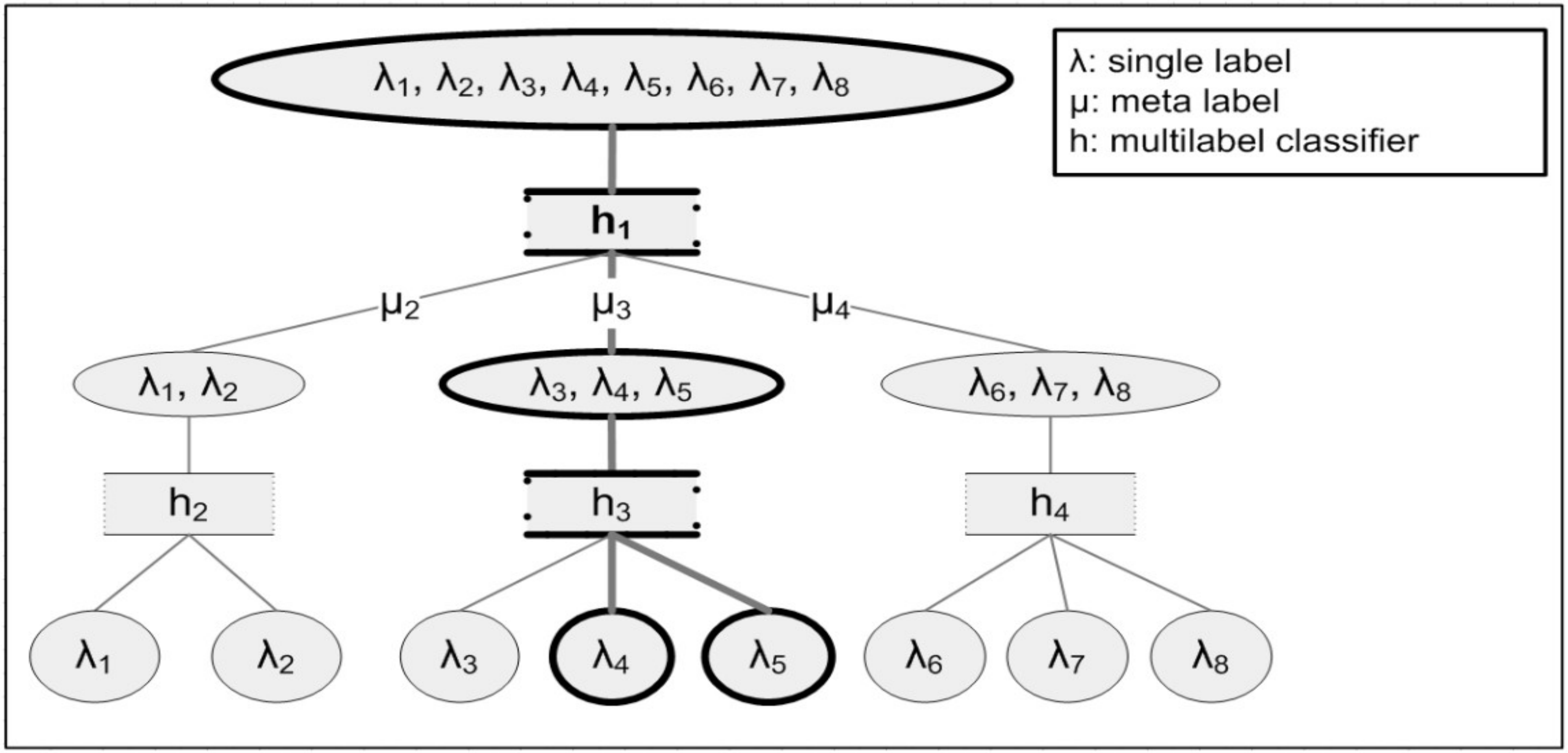


Fig. 1. Sample hierarchy for a multilabel classification task with 8 labels

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2. HOMER

- fewer training examples for each classifier
- even distribution of labels → more balanced training sets
- similarity-based distribution of labels → only few branches of tree activated

But how do we distribute the labels?

2.1. Balanced k Means

```
Input: number of clusters k, labels L_n, label data W_i, iterations it
Output: k balanced clusters of labels
for i ← 1 to k do
    // initialize clusters and cluster centers
    C_i ← ∅ ;
    c_i ← random member of L_n ;
while it > 0 do
    foreach λ ∈ L_n do
        for i ← 1 to k do
            d_λi ← distance(λ, c_i, W_i)
        finished ← false;
        ν ← λ ;
        while not finished do
            j ← arg min_i d_νi;
            Insert sort (ν, d_ν) to sorted list C_j;
            if |C_j| > ⌈|L_n|/k⌉ then
                ν ← remove last element of C_j;
                d_νj ← ∞ ;
            else
                finished ← true;
    recalculate centers;
    it ← it − 1
return C_1, ..., C_k;
```

Fig. 2. Balanced k-Means Algorithm

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance

Table 1. Information and multilabel statistics for the data sets used in the experiments

Dataset	Examples Train	Examples Test	Attributes Numeric	Attributes Discrete	Labels	Label Cardinality	Label Density
delicious	12920	3185	0	500	983	19.020	0.019
mediamill	30993	12914	120	0	101	4.376	0.043

- training complexity: O(f(|L|)+|L|), with f(|L|)= complexity of balanced clustering
- testing: O($\log_k$(|L|)), instead of O(|L|)

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance

- HOMER-R: distributes labels evenly but randomly
- HOMER-K: uses k means
- HOMER-B: uses balanced k means
- BR: binary relevance method (one binary classifier for each label)

2.2. Performance

- BR: F-Measure: 0.081, Loss: 0.282

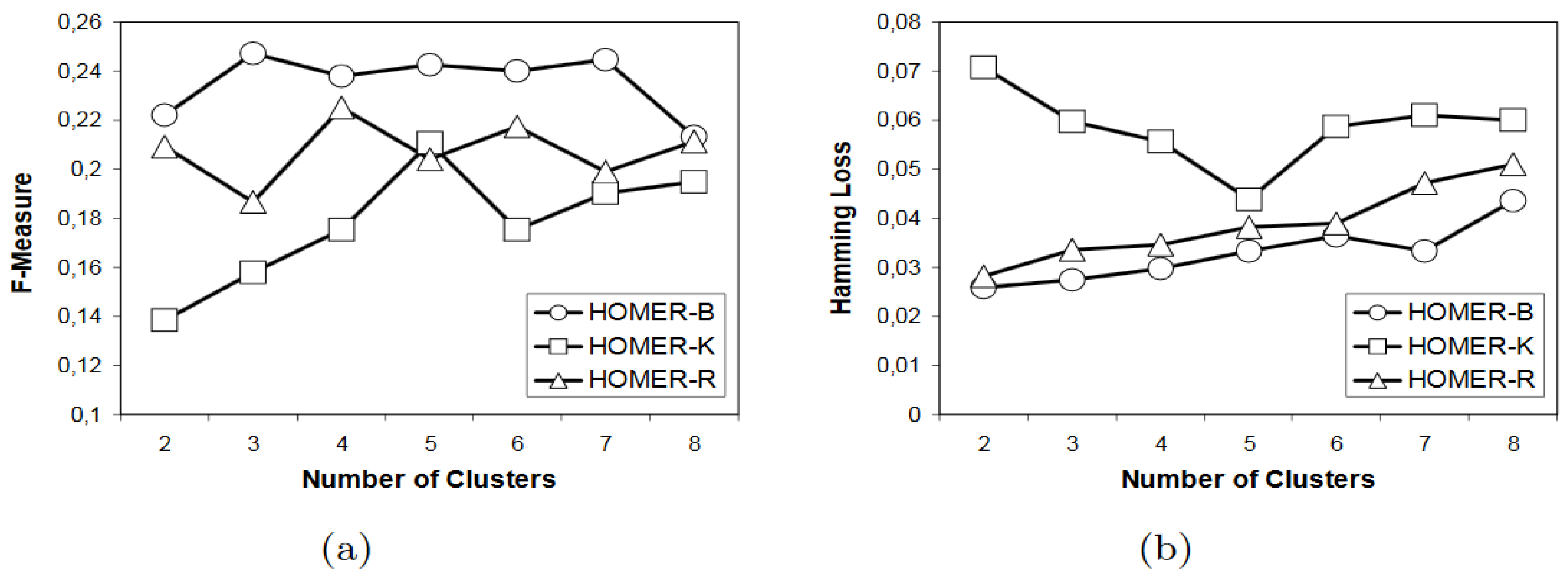


Fig. 3. Predictive performance of HOMER and variations in delicious

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance

- BR: F-Measure: 0.157, Loss: 0.331

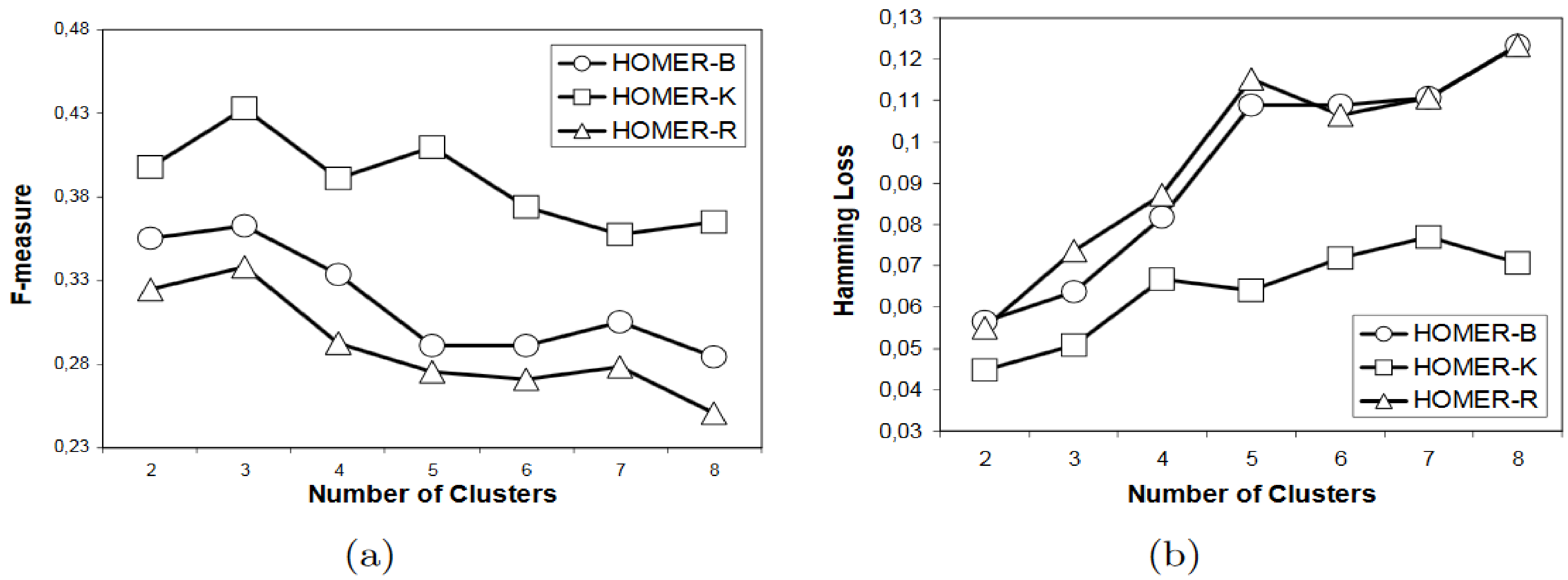
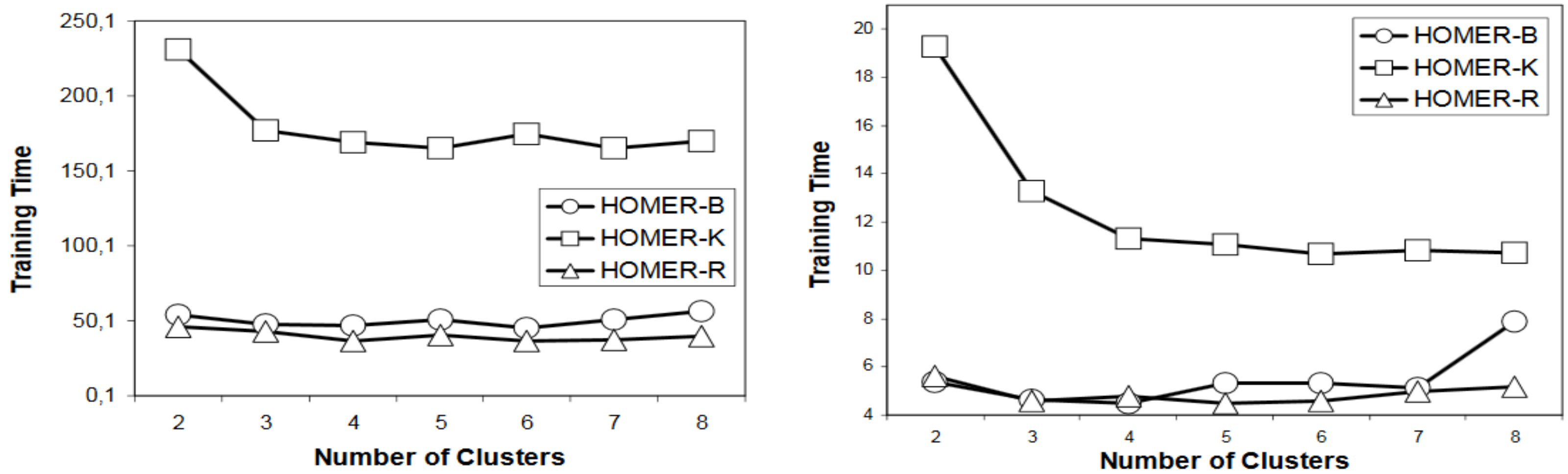


Fig. 4. Predictive performance of HOMER and variations in mediamill

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance



(a) delicious (b) mediamill

Fig. 7. Training time of HOMER and variations

- BR: 24.6 min delicious, 10.1 min mediamill
- measured in wall time!

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance

- BR: 983 classifiers activated, 69.4 min testing time

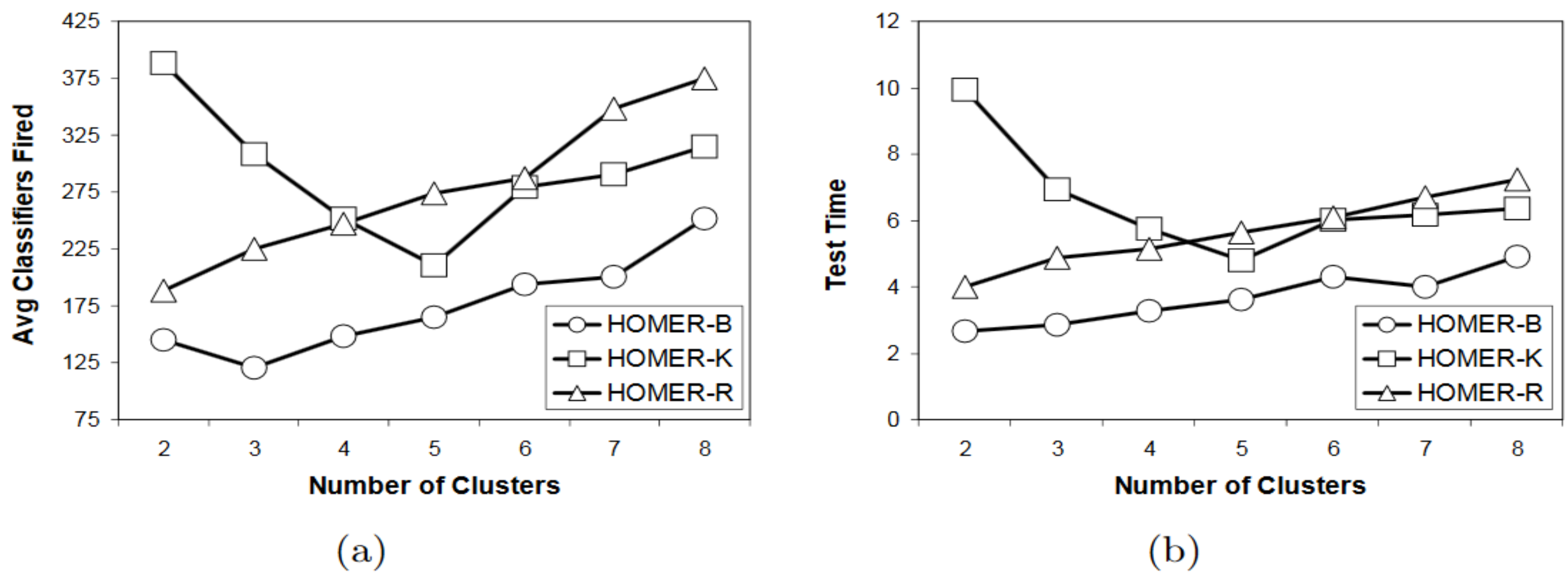


Fig. 8. Average classifiers fired and testing time of HOMER and variations in delicious

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.2. Performance

- BR: 101 classifiers activated, 7.6 min testing time

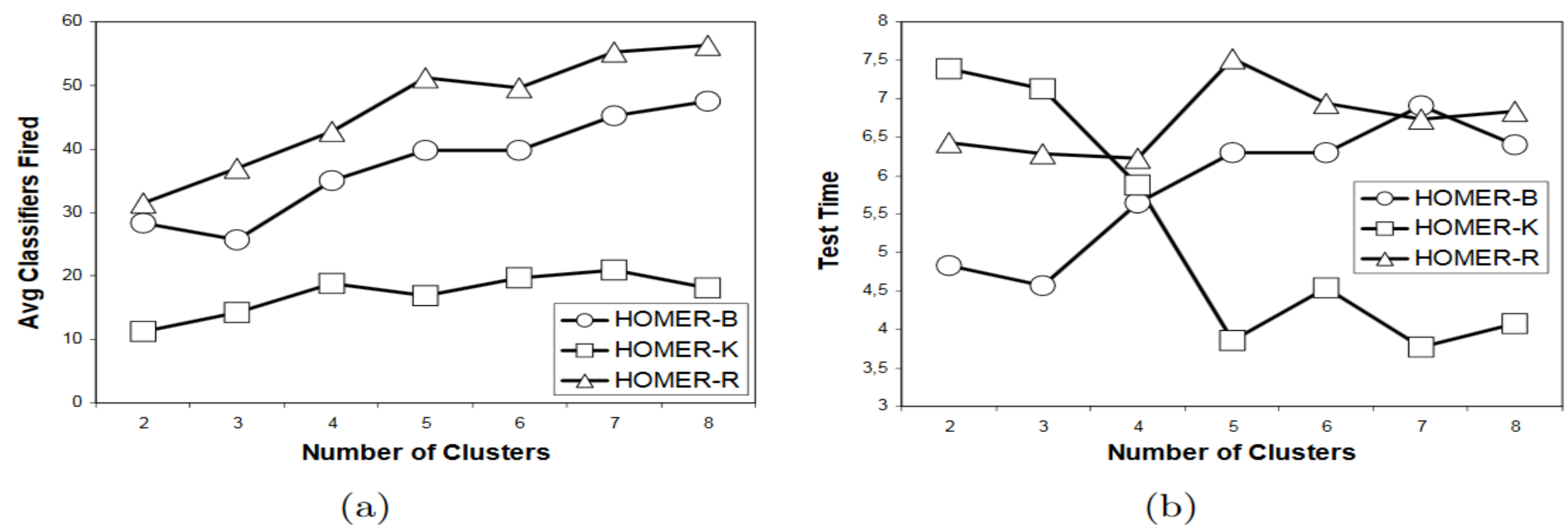


Fig. 9. Average classifiers fired and testing time of HOMER and variations in mediamill with respect to the number of clusters

Figure by G. Tsoumakas, I. Katakis and I. Vlahavas

2.3. Discussion

Quality of Prediction

Training Time

Questions? Ideas?

Balanced k Means

Testing Time

HOMER

3. Label Embedding Trees

- $T = (N, E, F, L)$
  - indexed nodes $N = \{0, \ldots n\}$
  - edges E
  - label predictors $F = \{f_1, \ldots, f_n\}$ (scoring)
  - label sets $L = \{l_0, \ldots, l_n\}$
- label embeddings

→ goal: minimize tree loss

3. Label Embedding Trees

$$R_{emp}(f_{tree}) = \frac{1}{m}\sum_{i=1}^{m} \max_{j \in B(x)} I(y_i \notin \ell_j)$$

m = #examples, B(x) = indices of "best" nodes

- minimize approximation of empirical loss over variables F:

  a) count errors of all nodes independently

  b) count errors of nodes jointly (check if node containing true label is ranked highest of siblings)

3. Label Embedding Trees

- minimize overall tree loss over N, E, L:

I. Train k One-vs-Rest classifiers independently

II. Compute confusion matrix on validation set

III. For each internal node: partition label set between children's by choosing subsets that have max confusion of labels in the subset

3. Label Embedding Trees

How do we predict using the learnt tree?

Algorithm 1 Label Tree Prediction Algorithm

Input: test example x, parameters T.
Let $s = 0$. — *- Start at the root node*
repeat
  Let $s = \text{argmax}_{\{c:(s,c)\in E\}} f_c(x)$. — *- Traverse to the most confident child.*
until $|\ell_s| = 1$ — *- Until this uniquely defines a single label.*
Return ℓ_s.

Figure by Samy Bengio, Jason Weston and David Grangier

3. Label Embedding Trees

$$f_{embed}(x) = \text{argmax}_{i=1,\ldots,k}\ S\left(Wx, V\phi(i)\right)$$

- V is a $d_e \times k$ matrix, k = #labels
- W is a $d_e \times d$ matrix, d = #features
- S(*,*) = measure of similarity
- $\Phi(i)$ is a k-dimensional vector with a 1 at the ith position and 0 otherwise

How do we learn V and W?

3. Label Embedding Trees

a) first learn V, so that similar classes have small distance between their label embedding vectors → learn W, by minimizing approximation of empirical loss (convex problem)

b) learn W and V jointly, by directly minimizing approximation of empirical loss (non-convex problem)

3. Label Embedding Trees

- potentially $O(d_e\,(d + \log(k))$ testing speed

Algorithm 3 Label Embedding Tree Prediction Algorithm

Input: test example x, parameters T.
Compute $z = Wx$. *- Cache prediction on example*
Let $s = 0$. *- Start at the root node*
repeat *- Traverse to the most*
  Let $s = \text{argmax}_{\{c:(s,c)\in E\}} f_c(x) = \text{argmax}_{\{c:(s,c)\in E\}} z^\top \mathcal{E}(c)$. *confident child.*
until $|\ell_s| = 1$ *- Until this uniquely defines a single label.*
Return ℓ_s.

Figure by Samy Bengio, Jason Weston and David Grangier

3.1. Performance

Table 1: **Summary Statistics of the Three Datasets Used in the Experiments.**

Statistics	ImageNet	Product Descriptions	Product Images
Task	image annotation	product categorization	image annotation
Number of Training Documents	2518604	417484	417484
Number of Test Documents	839310	60278	60278
Validation Documents	837612	105572	105572
Number of Labels	15952	18489	18489
Type of Documents	images	texts	images
Type of Features	visual terms	words	dense image features
Number of Features	10000	10000	1024
Average Feature Sparsity	97.5%	99.6%	0.0%

Figure by Samy Bengio, Jason Weston and David Grangier

3.1. Performance

Table 2: **Flat versus Tree Learning Results** Test set accuracies for various tree and non-tree methods on three datasets. Speed-ups compared to One-vs-Rest are given in brackets.

Classifier	Tree Type	ImageNet	Product Desc.	Product Images
One-vs-Rest	None (flat)	2.27% [1×]	37.0% [1×]	12.6% [1×]
Filter Tree	Filter Tree	0.59% [1140×]	14.4% [1285×]	0.73% [1320×]
Conditional Prob. Tree (CPT)	CPT	0.74% [41×]	26.3% [45×]	2.20% [115×]
Independent Optimization	Random Tree	0.72% [60×]	21.3% [59×]	1.35% [61×]
Independent Optimization	Learnt Label Tree	1.25% [60×]	27.1% [59×]	5.95% [61×]
Tree Loss Optimization	Learnt Label Tree	2.37% [60×]	39.6% [59×]	10.6% [61×]

Table 3: **Label Embeddings and Label Embedding Tree Results**

		ImageNet			Product Images		
Classifier	Tree Type	Accuracy	Speed	Memory	Accuracy	Speed	Memory
One-vs-Rest	None (flat)	2.27%	1×	1.2 GB	12.6%	1×	170 MB
Compressed Sensing	None (flat)	0.6%	3×	18 MB	2.27%	10×	20 MB
Seq. Convex Embedding	None (flat)	2.23%	3×	18 MB	3.9%	10×	20 MB
Non-Convex Embedding	None (flat)	2.40%	3×	18 MB	14.1%	10×	20 MB
Label Embedding Tree	Label Tree	2.54%	85×	18 MB	13.3%	142×	20 MB

Figure by Samy Bengio, Jason Weston and David Grangier

3.2. Discussion

Quality of Prediction

Tree Loss

<u>Questions? Ideas?</u>

Label Trees

Speed-up

Label Embeddings

4. Conclusion

+ tree structures offer reduction of testing time and better predictions in comparison to flat structures

- all predictable labels have to be known before training

- longer training time (may be reduced by optimization)

→ could try building tree with e.g. WordNet for text classification

Sources

- G. Tsoumakas, I. Katakis, and I. Vlahavas: Effective and efficient multilabel classification in domains with large number of labels. ECML/PKDD 2008 Work-shop on Mining Multidimensional Data, 2008.
- S. Bengio, J. Weston, and D. Grangier: Label embedding trees for large multi-class tasks. NIPS, pages 163–171. Curran Associates, Inc., 2010
- A.Y. Ng, M.I. Jordan, and Y. Weiss: On spectral clustering: Analysis and an algorithm. Advances in neural information processing systems, 2:849–856, 2002.
- M. Belkin and P. Niyogi: Laplacian eigenmaps and spectral techniques for embedding and clustering. Advances in neural information processing systems, 1:585–592, 2002.
- D. Hsu, S. Kakade, J. Langford, and T. Zhang: Multi-label prediction via compressed sensing. In Neural Information Processing Systems (NIPS), 2009.
- Bennett, K., Bradley, P., , Demiriz, A.: Constrained k-means clustering. TechnicalReport TR-2000-65, Microsoft Research (2000).
- L´eon Bottou. Stochastic learning: In Olivier Bousquet and Ulrike von Luxburg, editors, Advanced Lectures on Machine Learning, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.
- https://en.wikipedia.org/wiki/Hinge_loss