Learning and Inference in Probabilistic Classifier Chains with Beam Search

Abhishek Kumar[1⋆], Shankar Vembu[2⋆], Aditya Krishna Menon[1], and Charles Elkan[1]

[1] Department of Computer Science, UC San Diego, USA,
{abhishek,akmenon,elkan}@ucsd.edu
[2] Donnelly Centre for Cellular and Biomolecular Research, University of Toronto, Canada, shankar.vembu@utoronto.ca

Abstract. Multilabel learning is an extension of binary classification that is both challenging and practically important. Recently, a method for multilabel learning called *probabilistic classifier chains* (PCCs) was proposed with numerous appealing properties, such as conceptual simplicity, flexibility, and theoretical justification. However, PCCs suffer from the *computational* issue of having inference that is exponential in the number of tags, and the *practical* issue of being sensitive to the suitable ordering of the tags while training. In this paper, we show how the classical technique of *beam search* may be used to solve both these problems. Specifically, we show how to use beam search to perform tractable test time inference, and how to integrate beam search with training to determine a suitable tag ordering. Experimental results on a range of multilabel datasets show that these proposed changes dramatically extend the practical viability of PCCs.

1 Introduction

In the classical supervised learning task of binary classification, our goal is to learn some model that, given an input $x \in \mathcal{X}$, returns a single binary prediction $y \in \{0, 1\}$. This value y is considered to be a *label* of the example x, denoting whether it possesses some characteristic, or not. For example, x may represent an image by its pixel values, and y may denote whether or not the image contains a face. Multilabel learning is an extension of binary classification where the goal is to return *multiple* binary predictions, or equivalently a vector $y \in \{0, 1\}^K$. The label y now measures multiple characteristics of the example x, each of which we call a *tag*. For example, x may represent an image as before, and y could denote whether K specific people's faces appear in the image.

The recently proposed probabilistic classifier chains [1,2] (PCCs) are an attractive solution for multilabel classification for several reasons. First, it is based on a reduction of multilabel to binary classification, which allows us to leverage existing research on designing scalable and powerful binary classifiers. Second,

⋆ Contributed equally

it is a principled probabilistic model, and there is a theoretical understanding of how it may be used to produce Bayes optimal predictions for a variety of loss functions [1]. Third, it is computationally inexpensive to train (unlike e.g. structured prediction methods [3], which involve inference during training). Fourth, it is trained on the original label space without any prior transformations [4,5,6,7], which is important in certain settings and applications.

Despite the above positive characteristics, the current formulation of PCCs suffers from at least a couple of drawbacks. First, on the *computational* side, they are only applicable to multilabel data sets with a few number of tags. This is because to use a PCC at test time for an example with K possible tags, we need to evaluate all 2^K possible labellings, and pick the highest scoring one. This becomes quickly infeasible as K increases. Second, on the *performance* side, their accuracy depends on a pre-specified ordering of the tags. Different orderings result in solutions of different accuracy, and so a natural question is whether one can determine the ordering that yields the best performance. As with the previous issue, the current understanding of PCCs requires either picking a random ordering, or trying all $K!$ possibilities.

In this paper, we propose to address these shortcomings with PCCs using *beam search*, a classical AI search technique. In particular, we propose to use beam search to perform inference on PCCs at test time, changing the runtime from $O(2^K)$ to $O(bK)$, where b is a tunable beam width. As we shall demonstrate, in practice a beam size $b \ll 2^K$ achieves good performance. We also present an algorithm that integrates the search for the best ordering of tags with the learning algorithm. To avoid the burden of training a classifier for each ordering, we use *kernel target alignment* [8] to score the viability of a given ordering. Finally, we propose a richer feature representation for learning individual tag models than that is used in existing PCC solutions [1,2]. Experimental results on a range of multilabel data sets show that our scheme is able to improve on PCC, and extend its applicability to data sets with a large number of tags.

This paper is organized as follows. First, in Section 2, we analyze PCCs in detail and highlight some of the challenges in using them. Next, in Section 3, we show how one may use beam search to speed up test time inference of the method. In Section 4, we then show how beam search may be integrated during the learning phase to determine the tag ordering. Finally, we present a range of experimental results in Section 5.

2 Multilabel Learning and Probabilistic Classifier Chains

2.1 Multilabel learning

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the input space and $\mathcal{Y} = \{0,1\}^K$ be the label output space defined over a fixed set of K *tags*. Given a set of m training samples $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$ where $(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$, the goal of a multilabel classification algorithm is to learn a mapping $f : \mathcal{X} \to \mathcal{Y}$. We will use the notation $y_k^{(i)}$ to denote the kth tag of the ith example.

The naïve solution to the multilabel learning problem is to decompose it into K independent binary classification problems, one for each tag y_k. This method is known as *binary relevance*. In principle, this method is in fact optimal for certain loss functions, such as the Hamming and the ranking loss [1]. However, in practical situations where training data is limited, and for loss functions that take into account consistency of the entire tag sequence, it is intuitively necessary to exploit correlations between tags to make better predictions. This has motivated a slew of multilabel learning methods (see [9,10,11] for surveys).

Recently, Read *et al.* [2] proposed a simple decomposition method called the *classifier chain* (CC) that appears similar to binary relevance, but is able to exploit tag correlations. As with binary relevance, the idea is to train K separate models, one for each tag. The difference is that in the model for tag k, we use as input features not only the data point x but also the $(k-1)$ tags, $y_1, y_2, \ldots, y_{k-1}$, previously modeled. We thus attempt to use any relevant information in the previous tags to help simplify our model. Our focus in this paper is the related PCC method [1], which generalizes this scheme through a probabilistic framework.

2.2 The PCC model for multilabel learning

A probabilistic classifier chain [1] tries to estimate the conditional distribution $p(y \mid x)$ using the chain rule of probabilities:

$$p(y \mid x) = p(y_{\pi(1)} \mid x) \prod_{k=2}^{K} p(y_{\pi(k)} \mid x, y_{\pi(1)}, \ldots, y_{\pi(k-1)}) \ ,$$

where $\pi(\cdot)$ is some fixed permutation/ordering of tags. Thus, learning a multilabel classifier is reduced to learning K independent probabilistic binary classifiers. These independent base classifiers may be, for example, logistic regression models with a specialized feature representation:

$$p(y_{\pi(k)} \mid x, y_{\pi(1)}, \ldots, y_{\pi(k-1)}; \theta) \propto \exp(\langle \theta_{\pi(k)}, \phi_{\pi(k)}(x, y) \rangle), \quad \forall k \in \{2, \ldots, K\}.$$

In [1], the choice $\phi_k(x, y) = x \oplus (y_1, \ldots, y_{k-1})$ was used, where $a \oplus b$ is the concatenation of the vectors a and b, so that $\theta_k \in \mathbb{R}^{d+k-1}$. In total, this means we need to learn $\mathbb{R}^{dK+K(K-1)/2}$ parameters, as opposed to R^{dK} parameters with binary relevance. Suppose we write $\theta_k = \left[w_k; v_k\right]$, where $w_k \in \mathbb{R}^d$ and $v_k \in \mathbb{R}^{k-1}$. Then, it may be verified that the joint probability model with a logistic regression base learner is

$$p(y \mid x; \theta) = \frac{\exp\left(y^T W x + y^T V y\right)}{\prod_{k=1}^{K} (1 + \exp\left((Wx + Vy)_k\right))}, \tag{1}$$

where we have $W = \left[w_1 \ldots w_K\right]$ and $V = \left[v_1 \ldots v_K\right]$. The matrix V is lower-triangular, since we first model $y_{\pi(1)}$, then $y_{\pi(2)}$, *et cetera*. By contrasting this

to the joint model assumed for binary relevance,

$$p(y \mid x; \theta) = \frac{\exp\left(y^T W x\right)}{\prod_{k=1}^{K}(1 + \exp((Wx)_k))} \ ,$$

we see that the key difference in the joint probability model (1) is the tag correlation term $y^T V y$.

2.3 Advantages of using PCCs

PCCs have a number of attractive properties as a multilabel classification method.

(i) It is based on a reduction from multilabel to binary classification, which allows us to leverage existing research on designing scalable and powerful binary classifiers. Compared to the original CC method [2], which also uses decomposition, the key difference in this regard is that the decomposition is probabilistically motivated. Also, unlike CC, PCC does not use the model's *predictions* of the past tags during test time inference.

(ii) It is a principled probabilistic model with a theoretical understanding of how it may be used to produce Bayes optimal predictions for a variety of loss functions [1]. This is in contrast to several multilabel learning methods, where the statistical consistency of the algorithm is unclear. Further, the probabilistic underpinning gives a clear idea on how to modify the algorithm. For example, as we shall see later, the probabilistic setup allows one to design inference methods that are more efficient than the greedy inference described in [2].

(iii) Being a decomposition method, it is computationally inexpensive to train, requiring only marginally more effort than the binary relevance baseline. This is unlike e.g. structured prediction methods [3] which involve inference during training.

(iv) It is trained on the original label space without any prior transformations [4,5,6,7]. This is conceptually appealing and makes modifications much simpler. As an example, suppose we want to address the issue of class imbalance at the tag level. One way to do this is to appropriately modify the inputs to the models for each $p(y_k \mid x, y_1, \ldots, y_{k-1})$ by applying cost-sensitive weighting. By contrast, in transformation methods, since we lose the relationship to the original label space, it is not clear how modifications in the transformed space affect those in the original space.

2.4 Challenges with using PCCs

Thus far, we have not discussed two crucial questions in using PCCs: how to train them, and how to apply them at test time. At *training time*, one may maximize the log-likelihood of the given training set, which decomposes into K

distinct optimizations for each tag:

$$\begin{aligned}\mathcal{L}(\theta;\pi(\cdot)) &= \sum_{i=1}^{m} \log p(y^{(i)} \mid x^{(i)};\theta) \\ &= \sum_{i=1}^{m} \left[\log p(y^{(i)}_{\pi(1)} \mid x^{(i)};\theta) + \sum_{k=2}^{K} \log p(y^{(i)}_{\pi(k)} \mid x^{(i)}, y^{(i)}_{\pi(1)}, \ldots, y^{(i)}_{\pi(k-1)};\theta) \right] . \end{aligned} \tag{2}$$

The above hides a subtle issue: while in theory the chain rule applies regardless of the ordering of the tags, in practice, the ordering can make a big difference. The reason is that our model for each individual $p(y_{\pi(k)} \mid x, y_{\pi(1)}, \ldots, y_{\pi(k-1)})$ may be misspecified, in which case some orderings will be better modeled than others. Therefore, we can expect different solutions based on the choice of $\pi(\cdot)$. This prompts the natural question of what the *best* ordering $\pi(\cdot)$ is, in the sense of resulting in the highest possible value of $\mathcal{L}(\theta)$. It may seem that one should order the tags in order of "difficulty", but this may not be optimal: for example, a tag that is difficult to model may make subsequent tags considerably easier to model. Thus, some principled algorithmic solution is necessary.

At *test time*, the problem becomes one of estimating, for a given feature vector x, the most likely set of tags under the learned parameters $\hat{\theta}$:

$$\hat{y} = \operatorname*{argmax}_{y \in \{0,1\}^K} p(y \mid x;\hat{\theta}) \ .$$

This inference is unfortunately computationally intractable. The proposal in [1] is to simply perform brute-force enumeration of all possible labels.

To summarize, we see that there are two main issues with using PCCs in practice. On the *computational* side, the method proposed for test time inference in [1] requires that we enumerate all 2^K possible candidate labellings, and evaluate them. Indeed, existing applications of PCCs have been restricted to data sets with relatively few number of tags. A general purpose multilabel method should of course handle a large number of tags. On the *accuracy* side, the choice of ordering the tags while training can make a difference in generalization performance. One might hope to do significantly better than just a random ordering. While Dembczyński *et al.* [1] proposed taking several random orderings to create an ensemble of PCCs, we would like a more principled procedure, that searches more intelligently.

We note that there are related schemes that deal with the above problems for PCCs. An inference algorithm was proposed in [12] which makes assumptions on the joint probability distribution of labels to guarantee polynomial-time convergence of the algorithm. However, it does not address the problem of learning tag orderings. Our algorithm based on beam search is generic in the sense that it is possible to accommodate a variety of scoring functions into the search algorithm thereby allowing us to solve both the inference problem and the problem of learning tag orderings. In [13], an algorithm is proposed to learn an undirected network of dependencies between the tags, which is intractable in general, and

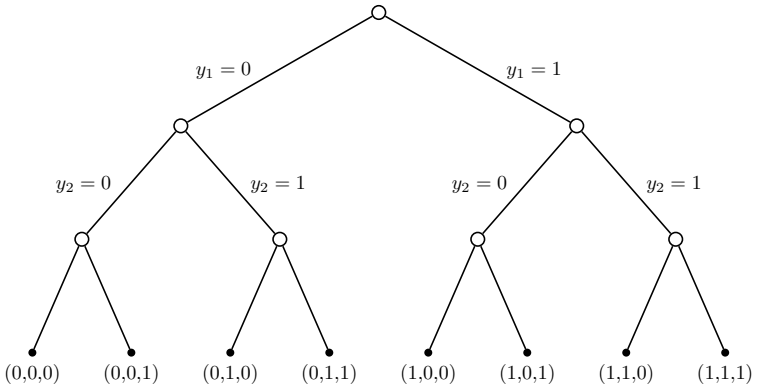

Fig. 1. Binary tree used in beam search for inference with $K = 3$ tags.

therefore approximates the structure learning problem using the Chow-Liu algorithm to learn a tree dependency structure between tags. However, this tree structure is unlikely to represent many real-world scenarios and it is an empirical question whether such an approximation is good or not.

3 Label Inference using Beam Search

Recall that the inference problem in PCC is $\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}} \; p(y \mid x; \hat{\theta})$, i.e., we wish to find the maximum scoring label vector. Assuming our probability model is correctly specified, the resulting solution will give the Bayes optimal prediction for subset 0/1 loss [1]. This inference task is equivalent to finding the optimal path in a rooted, complete binary tree of height K, where each internal vertex at level k denotes a possible *partial label vector* of length k, so that the leaf nodes represent all possible 2^K label vectors (see Figure 1). Thus, the inference problem is one of finding the optimal path from the root to one of the leaves in this binary tree, where the score of a vertex v at level k with a corresponding partial label $y^{(v)}$ is equal to the partial probability:

$$s_k(v; \hat{\theta}) = p(y_1^{(v)} \mid x; \hat{\theta}) \cdot \prod_{j=2}^{k} p(y_j^{(v)} \mid x, y_1^{(v)}, \ldots, y_{j-1}^{(v)}; \hat{\theta}) \;, \tag{3}$$

which can be computed recursively. The inference algorithm in the original classifier chain [2] greedily searches for the optimal path by deciding at each level of the tree whether to traverse in the left or the right direction. However, this may not result in finding the optimal label vector [1].

We propose an inference algorithm using *beam search* [14], which is a heuristic search technique. A*[15] and similar search algorithms could also be used as more sophisticated alternatives to beam search. The basic idea is that we will keep b candidate solutions at each level of the tree, where b is a user-defined parameter known as the beam width, which represent the best partial solutions seen thus

far. We then explore the tree in a breadth-first fashion using these solutions. Greedy search is recovered for the case of $b = 1$.

Our inference procedure for PCCs is described in Algorithm 1. At each level of the tree, we maintain a list of best-scoring candidate vertices of size at most b, where b is the beam width. We traverse down the tree by considering the children of only those vertices that are in this list, sort them in increasing order of their partial probabilities (3), and prune all the vertices that are not in the top-b list.

Algorithm 1 Inference using beam search.

Input: Query point x, learned model parameters $\hat{\theta}$, beam width b
Output: Estimate $\hat{y}$ for $\operatorname{argmax}_y p(y \mid x; \hat{\theta})$

$B^{(0)} = \{(1, 0)\}$ {initialize beam}
for $j = 1 \dots K$ **do**
  $B^{(j)} = \{\}$
  for $(\text{parentTags}, \text{parentScores}) \in B^{(j-1)}$ **do**
    for $z \in \{0, 1\}$ **do**
      if $p(y_j = z \mid x, \text{parentTags}; \hat{\theta}) > \min\{v : (\cdot, v) \in B^{(j)}\}$ **then**
        $B^{(j)} \leftarrow B^{(j)} \cup (\text{parentTags} \cup \{z\}, p(y_j = z \mid x, \text{parentTags}; \hat{\theta}))$
        $B^{(j)} \leftarrow \text{Top-}b(B^{(j)})$
      end if
    end for
  end for
end for
$\hat{v} = \operatorname{argmax}_v\{v : (\cdot, v) \in B^{(K)}\}$ {highest score}
return $\hat{y} : (\hat{y}, \hat{v}) \in B^{(K)}$

The greedy inference algorithm used in classifier chain [2] is recovered as a special case with beam width $b = 1$ and performing inference by exhaustively enumerating all possible labels is equivalent to doing beam search with $b = \infty$. Thus, by tuning b, we can control the tradeoff between computation time and accuracy of our method. The hope is that for real-world multilabel datasets, we can use a relatively small value of b and get performance that is significantly better than the greedy approach, and commensurate with exhaustive enumeration. This is a question that we will answer empirically in Section 5.

4 Learning to Order Tags

Recall that training a PCC involves picking a particular ordering $\pi(\cdot)$ of the tags, based on which we apply the chain rule decomposition. The tag ordering problem is to find the best $\pi(\cdot)$ in the sense of yielding the maximum log-likelihood $\mathcal{L}(\theta; \pi(\cdot))$ in Equation 2. It is easy to see that if each individual tag model is misspecified, this quantity varies based on the choice of the permutation $\pi(\cdot)$.

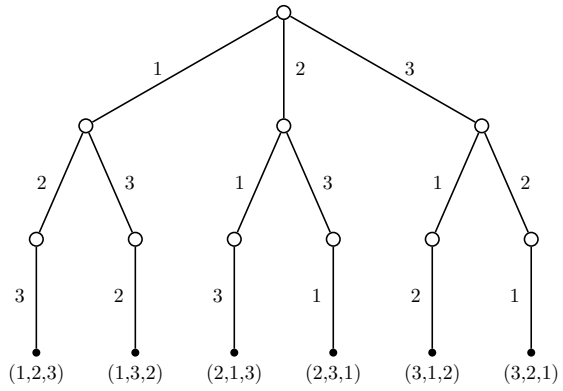

Fig. 2. Example of ordering tree for $K = 3$ tags.

Even if the model is correctly specified, the optimal solution may vary due to finite sample effects; for example, suppose that a tag y_k is extremely rare; then, on any finite sample, we may seriously misestimate $p(y_k = 1 \mid x)$, even if in the infinite sample case we will discover the correct probability.

Intuitively, one may expect the optimal ordering to progressively involve picking the easiest tag to model given the previously picked tags. But it may alternately be the case that given a "difficult" tag, subsequent tags are easy to model. A basic question then is to how to determine a suitable tag ordering without resorting to heuristics, or performing exhaustive enumeration over all $K!$ possible orderings.

We propose to use beam search to solve the problem of determining a suitable tag ordering. We do so by casting it as a search problem over a tree. Instead of a complete binary tree used in the inference algorithm, for the ordering problem we have a tree of height K, where every vertex at level t has $(K - t)$ children, as shown in Figure 2. Given such a tree, our goal is again to find the optimal path from the root to one of the leaf vertices.

Our procedure to learn tag orderings for PCC is described in Algorithm 2. Similar to the beam search algorithm used for inference, we use a beam of fixed width b, maintain a list of best-scoring candidate vertices of size at most b and prune all the vertices that are not in the top-b list. We now need to determine the scoring function used to prune the vertices. One possible scoring function is the validation error of classifier, i.e., for every candidate vertex in the tree, we train a (partial) PCC. More specifically, the score of a vertex v at level t is given by:

$$s_t(v; \hat{\theta}) = \sum_{(x,y) \in \mathcal{V}} \log p(y_{\pi_v(t)} \mid x, y_{\pi_v(1)}, \ldots, y_{\pi_v(t-1)}; \hat{\theta}) \ ,$$

where $\hat{\theta}$ are the parameters of the (partial) PCC that is being evaluated on a validation set of examples $\mathcal{V}$, and the partial tag ordering specified by $\pi_v(\cdot)$ is the directed path from the root to the vertex v. However, this results in training a (partial) PCC at every vertex of the tree which can be prohibitively

Algorithm 2 Learning to order tags using beam search.

Input: Training set $\mathcal{T} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, beam width b
Output: Model parameters $\hat{\theta}$

$B^{(0)} = \{(1, 0)\}$ {initialize beam}
for $j = 1 \dots K$ **do**
  $B^{(j)} = \{\}$
  for (parentTags, parentScores) $\in B^{(j-1)}$ **do**
    for $i \in \{1, \dots, K\} \backslash$parentTags **do**
      $\mathcal{K} \leftarrow k\left(\phi(x, y_{[\text{parentTags}]}), \phi(x', y'_{[\text{parentTags}]})\right)$
      $\ell = [y_i^{(1)}; y_i^{(2)}; \dots; y_i^{(m)}]$
      if $\text{KTA}(\mathcal{K}, \ell) > \min\{v : (\cdot, v) \in B^{(j)}\}$ **then**
        $B^{(j)} \leftarrow B^{(j)} \cup (\text{parentTags} \cup \{i\}, \text{KTA}(\mathcal{K}, \ell))$
        $B^{(j)} \leftarrow \text{Top-}b(B^{(j)})$
      end if
    end for
  end for
end for
$\hat{v} = \text{argmax}_v\{v : (\cdot, v) \in B^{(K)}\}$ {highest score}
Return $\hat{\theta}$ learned by training a PCC using the ordering specified by $\hat{\pi} : (\hat{\pi}, \hat{v}) \in B^{(K)}$

expensive. As a computationally cheaper alternative, we propose to instead use *kernel target alignment* (KTA) [8] as a measure to score vertices. We want to measure to what extent similar training examples agree on a single given binary tag. Let $y \in \{0, 1\}^m$ be a vector containing the value of this tag for each of the m training examples. The matrix $yy^\top$ is a kernel matrix based on this tag. Let $\mathcal{K}$ be the kernel matrix based on the feature vector representing each of the m examples. Let $\langle A, B\rangle_F = \sum_{ij} A_{ij} B_{ij}$ denote the Frobenius inner product between two matrices A and B. The kernel target alignment between $\mathcal{K}$ and $yy^\top$ is

$$\text{KTA}(\mathcal{K}, y) = \frac{\langle \mathcal{K}, yy^\top \rangle_F}{\sqrt{\langle \mathcal{K}, \mathcal{K} \rangle_F \langle yy^\top, yy^\top \rangle_F}} \ .$$

In practice, the KTA score may be much more efficient to compute than training a (partial) PCC. (Indeed, this was our experience in the empirical study reported in Section 5.) Note that there are also hidden costs with training a classifier, such as performing cross-validation to determine regularization and other hyperparameters. Intuitively, the KTA score can be considered as a proxy for the accuracy of a classifier trained on the same input features and outputs and therefore it is reasonable to expect the KTA scores to correlate positively with the accuracies of a classifier.

The KTA score of a vertex v at level t is computed by constructing a kernel matrix whose entries are $k\left(\phi(x, y_{\pi_v([t-1])}), \phi(x', y'_{\pi_v([t-1])})\right)$, where $k(\cdot, \cdot)$ is the kernel function, $\phi(x, z) = x \otimes z$, i.e., the Kronecker product of x and z, for *cross-product features* and $\phi(x, z) = x \oplus z$ for *concatenated features*, and

Table 1. Details of benchmark multilabel data sets [16].

Data set	# training inst. (m)	# test inst.	# features (d)	# tags (K)
Emotions	391	202	72	6
Scene	1211	1196	294	6
Yeast	1500	917	103	14
Genbase	463	199	1186	27
Medical	333	645	1449	45
Enron	1123	579	1001	53

$y_{\pi_v([t-1])} = (y_{\pi_v(1)}, \ldots, y_{\pi_v(t-1)})$. For linear kernels, the kernel matrix factorizes into the product of the kernel matrix defined on the input features and the kernel matrix defined on the output labels. Note that earlier, the log-likelihood scoring function led to a naturally additive objective function. While the same can be done with KTA, it is an empirical question whether this will be appropriate or not. Observe in particular that an alternative is to use the product of KTA scores, which treats the KTA as a surrogate for the raw probability score itself.

5 Experimental Results

5.1 Data sets and methods

We report experiments on benchmark multilabel data sets [16] listed in Table 1. We selected all the data sets with fewer than 100 tags and fewer than 10K training instances so that all models can be trained easily using batch optimization methods. Note that on the majority of these data sets, inference by exhaustive enumeration is either computationally expensive or intractable. All data sets have a pre-defined test set, and our reported results are on this set. We compare the following *reduction*-based algorithms:

(a) Binary relevance (BR): This is the baseline algorithm where we train separate independent logistic regressors for each tag.
(b) Kernel dependency estimation (KDE): This is the algorithm described in [4]. Here, a (linear) transformation using PCA is applied to the original label matrix in order to decorrelate the tags. Then, independent regressors are trained in the transformed label space.
(c) Probabilistic classifier chain (PCC): We use the original formulation as described in [1] but with beam search as inference and the original ordering of tags found in the data sets.
(d) PCC with logistic regression using beam search for *both* learning the tag ordering and inference. To learn the tag ordering, we use kernel target alignment as the scoring function in beam search. Note that, in this setting, the output of beam search for learning is the tag ordering which is then used at a later stage to train a PCC.

We evaluate the performance of algorithms using the following loss functions:

(i) **Subset** $0/1$ **loss:**

$$\ell_{0/1}(y, \hat{y}) = [\![y \neq \hat{y}]\!] \ ,$$

(ii) **Hamming loss:**

$$\ell_h(y, \hat{y}) = \sum_{i=1}^{K} [\![y_i \neq \hat{y}_i]\!] \ , \quad \text{and}$$

(iii) **Ranking loss:**

$$\ell_r(y, \hat{y}) = \sum_{(i,j):y_i > y_j} \left([\![\hat{y}_i < \hat{y}_j]\!] + \frac{1}{2} [\![\hat{y}_i = \hat{y}_j]\!] \right) \ ,$$

where y and $\hat{y}$ are the target and the predicted labels respectively. It has been noted previously that BR is a strong baseline on a number of loss functions [2,1]. (It is in fact theoretically optimal for Hamming and ranking losses [1].) We have found this to be especially true if the base classifier for each tag is regularized. Some previous studies, such as [1], use an unregularized base classifier, for which BR may be misleadingly sub-optimal. In all our experiments, we used *regularized* linear models and tuned the regularization parameter using cross-validation.

5.2 Results and analysis

Q1: What is the effect of beam width used in inference and learning tag orderings on the performance of PCC?

For three of the data sets, namely, *Emotions*, *Scene* and *Yeast*, it is possible to do inference by exhaustive enumeration of all possible labels. We compared the performance of (i) binary relevance (BR) and (ii) probabilistic classifier chain (PCC) with the original tag ordering and using beam search for inference for several values of beam width b. Figure 3 shows the performance of the algorithms measured in terms of subset $0/1$ loss, Hamming loss and ranking loss with varying beam width.

From the figures, we see that the test set performance of PCC converges rapidly with $b < 15$ to the performance obtained with exhaustive enumeration, especially for the subset $0/1$ loss. We also observed that with $b = 15$, more than 90% of the correct labels were predicted within the candidate labels found by beam search, even if the single best label found by beam search was not exactly correct. Note that for Hamming and ranking losses, the loss at certain values of beam width is lower than the loss obtained by exhaustive enumeration which is surprising at first glance since exhaustive enumeration should ideally provide a lower bound for the test set loss. However, note that in beam search, labels are scored according to the joint probability $p(y \mid x)$ which may not necessarily correspond to the optimal result. Indeed, one of the points made in [1] was that taking $\text{argmax}_{y \in \mathcal{Y}} p(y \mid x)$ gives the optimal result for subset $0/1$ loss, but for Hamming and ranking losses the optimal result is for tag k, i.e.,

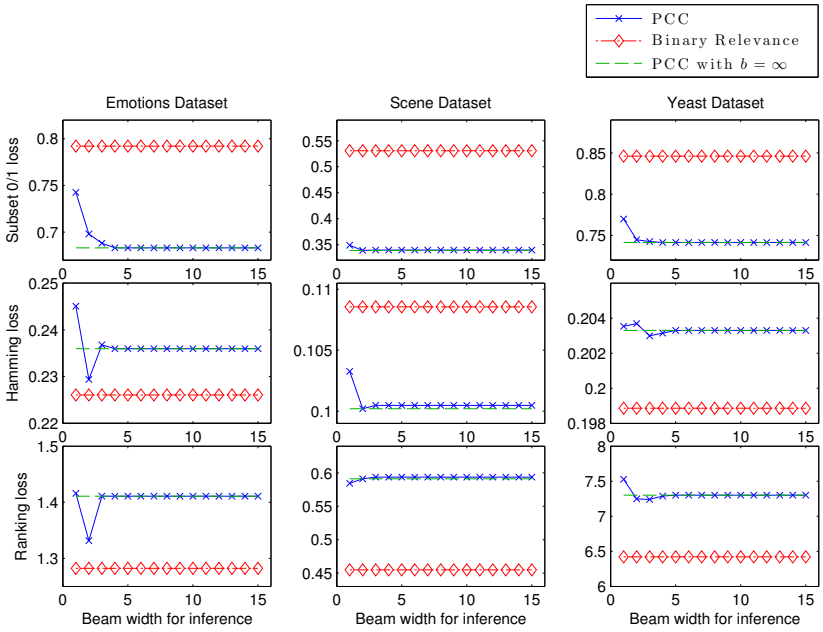

Fig. 3. Effect of beam width used in inference on the performance of classifiers.

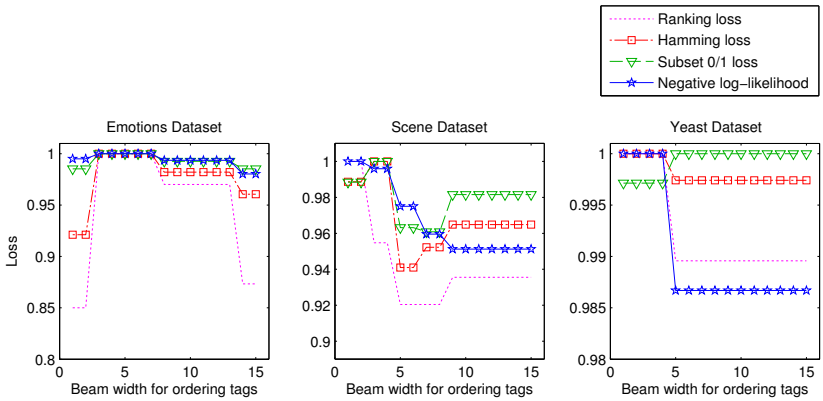

Fig. 4. Effect of beam width used to learn the tag ordering on the performance of PCCs. All losses have been scaled to $[0, 1]$ for the sake of legibility.

$\text{argmax}_{b \in \{0,1\}} p(y_k = b \mid x)$, which could be different in the case of misspecified models.

We also analyzed the effect of increasing beam width used in beam search to determine the tag ordering on the classifier performance. Figure 4 shows the performance of PCC with varying beam width on the three data sets, *Emotions*, *Scene* and *Yeast*. The test time inference was done by exhaustive enumeration of all possible labels. From the figure, we see that there is no clear pattern with

Table 2. Test set performance of binary relevance (BR), kernel dependency estimation (KDE) and probabilistic classifier chain (PCC) trained with *cross-product features* measured w.r.t. subset 0/1 loss (*top*), Hamming loss (*middle*) and ranking loss (*bottom*) on the benchmark data sets. Numbers in subscript and superscript indicate beam width used to learn tag ordering and for inference respectively. PCC_{or} is PCC trained with original tag ordering. The last row in each table shows the ranking of algorithms averaged across all the data sets, with lower ranks being better.

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.5309	0.6204	0.3487	0.3395	0.3395	0.3829	0.3620	0.3495	0.3478
Yeast	0.8462	0.8397	0.7699	0.7416	0.7416	0.7666	0.7579	0.7590	0.7601
Emotions	0.7921	0.7822	0.7426	0.6832	0.6832	0.6832	0.6733	0.6733	0.6634
Enron	0.8774	0.9016	0.8273	0.8169	0.8169	0.8394	0.8048	0.8100	0.8100
Medical	0.4170	0.4310	0.3891	0.3643	0.3643	0.3798	0.3597	0.3597	0.3597
Genbase	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201
Avg. Rank	7.83	8	6	3.67	3.67	6	3.25	3.5	3.08

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.1086	0.1204	0.1033	0.1005	0.1005	0.1189	0.1105	0.1044	0.1058
Yeast	0.1989	0.1984	0.2035	0.2033	0.2035	0.2154	0.2106	0.2094	0.2098
Emotions	0.2261	0.2236	0.2450	0.2360	0.2360	0.2351	0.2302	0.2302	0.2211
Enron	0.0463	0.0465	0.0488	0.0506	0.0508	0.0507	0.0516	0.0507	0.0517
Medical	0.0122	0.0127	0.0148	0.0132	0.0132	0.0122	0.0114	0.0114	0.0114
Genbase	0.0010	0.0008	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010
Avg. Rank	3.67	3.5	5.67	4.83	5.58	6.42	5.83	4.58	4.92

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.4548	0.5084	0.5844	0.5936	0.5936	0.6112	0.5920	0.5610	0.5284
Yeast	6.4209	6.4384	7.5267	7.3021	7.3010	7.6194	7.5463	7.4297	7.4526
Emotions	1.2822	1.4307	1.4158	1.4109	1.4109	1.5347	1.4851	1.4851	1.2970
Enron	12.9378	14.7219	16.1105	16.8929	16.9326	16.4447	16.9197	16.8765	16.6123
Medical	1.6271	1.3659	3.6922	3.7186	3.7202	2.9093	3.0783	3.0682	3.0682
Genbase	0.1709	0.0452	0.1910	0.1910	0.1910	0.1834	0.1834	0.1884	0.1884
Avg. Rank	1.33	2.33	5.83	6.33	6.67	6.25	6.5	5.42	4.33

varying beam width for subset 0/1, Hamming and ranking losses. However, we note that negative log-likelihood is non-increasing with increasing beam width. This confirms that beam search using KTA as the scoring function is successful at finding orderings that lead to good negative log-likelihood; as the amount of search done by beam search increases, better orderings are found.

Q2: Does learning to order tags improve the performance of PCC when compared to PCC using a random or otherwise pre-defined ordering?

The results are shown in Table 2. All variants of PCC consistently outperform or are in par with binary relevance for the subset 0/1 loss. Note that PCC^1_{or} is the

variant of PCC which uses the original tag ordering in the data sets and greedy search for inference, i.e., beam search with $b = 1$. All variants of PCC that uses beam search for inference and/or beam search to determine the tag ordering outperform PCC^1_{or} which clearly demonstrates the advantages of using beam search for PCC. On a majority of the data sets, variants of PCC that used beam search to determine the tag ordering using KTA as the scoring function gave the best results. For Hamming and ranking losses, we found that binary relevance is a strong baseline and outperformed PCC on average thus also confirming the results reported in [1]. Nevertheless, as for the subset 0/1 loss, we found that PCC using beam search to determine the tag ordering performed, on average, better than PCC that used the original tag ordering.

An interesting observation from the results reported in [1] is that for Hamming and ranking losses, PCC performs worse than binary relevance on average, but an ensemble of PCCs that were created from a random subsample of tag orderings performed better than binary relevance. As noted in [1], comparing a non-ensemble method with ensemble methods is not fair and we suspect that the improvements in performance from using an ensemble of PCCs may be due to the *ensemble* effect. Due to these reasons, we do not report results for ensembles of our method. The goal in this paper is to conduct a fair, controlled comparison of our beam search approach to the original PCC approach. Note that it is possible to create an ensemble of PCCs in our approach by selecting a subset of best scoring leaf vertices from the beam search tree used to determine the tag ordering.

Q3: How much of an impact does cross-product features have on the performance of PCC when compared to concatenated features?

The results are shown in Table 3. The relative performance of different algorithms is similar to those reported in Table 2. For the subset 0/1 loss, we found improvements in performance when using cross-product features, $\phi(x, y) = x \otimes y$. However, for Hamming and ranking losses, cross-product features seem to degrade the performance of classifiers when compared to features formed by concatenating labels, $\phi(x, y) = x \oplus y$. The fraction of experiments (an experiment is an entry in Table 2 or 3) where cross-product features performed better than concatenated features were 0.82, 0.44 and 0.31 (with ties broken at random) for subset 0/1, Hamming and ranking loss respectively. We suspect this rather surprising negative result for Hamming and ranking losses may be due to overfitting with increased number of features in the cross-product feature representation. However, the interplay between choice of loss functions and the choice of feature representations is unclear and we leave this as an open question.

6 Concluding Remarks

Empirical results clearly demonstrate the benefit of using beam search for test time inference and to learn the tag ordering. We believe these are important extensions to probabilistic classifier chains. Regarding directions for future work,

Table 3. Test set performance of binary relevance (BR), kernel dependency estimation (KDE) and probabilistic classifier chain (PCC) trained with *concatenated features* measured w.r.t. subset 0/1 loss (*top*), Hamming loss (*middle*) and ranking loss (*bottom*) on the benchmark data sets. The last row in each table shows the ranking of algorithms averaged across all the data sets, with lower ranks being better.

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.5309	0.6204	0.4022	0.3863	0.3863	0.4022	0.3813	0.3855	0.3813
Yeast	0.8462	0.8397	0.7895	0.7634	0.7634	0.8070	0.7612	0.7601	0.7601
Emotions	0.7921	0.7822	0.7475	0.6832	0.6832	0.7228	0.6634	0.6634	0.6634
Enron	0.8774	0.9016	0.8670	0.8497	0.8497	0.8566	0.8411	0.8411	0.8428
Medical	0.4170	0.4310	0.4093	0.4000	0.4000	0.4124	0.4031	0.4031	0.4031
Genbase	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201	0.0201
Avg. Rank	7.83	8	6.25	4.08	4.08	6.25	2.83	2.83	2.83

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.1086	0.1204	0.1145	0.1113	0.1113	0.1104	0.1086	0.1095	0.1056
Yeast	0.1989	0.1984	0.2131	0.2092	0.2092	0.2204	0.2113	0.2106	0.2106
Emotions	0.2261	0.2236	0.2368	0.2261	0.2261	0.2228	0.2112	0.2129	0.2129
Enron	0.0463	0.0465	0.0465	0.0462	0.0462	0.0461	0.0461	0.0461	0.0461
Medical	0.0122	0.0127	0.0122	0.0120	0.0120	0.1216	0.1196	0.1196	0.1196
Genbase	0.0010	0.0008	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010
Avg. Rank	4.58	4.92	7.08	4.92	4.92	5.83	4.25	4.5	4

	BR	KDE	PCC^1_{or}	PCC^5_{or}	PCC^{15}_{or}	PCC^1_1	PCC^5_1	PCC^5_5	PCC^{15}_{15}
Scene	0.4548	0.5084	0.6120	0.5978	0.5978	0.5397	0.5485	0.5293	0.5000
Yeast	6.4209	6.4384	7.7121	7.5071	7.5071	7.5474	7.6619	7.6314	7.6336
Emotions	1.2822	1.4307	1.3911	1.3218	1.3218	1.3317	1.3564	1.3366	1.3366
Enron	12.9378	14.7219	13.0743	13.0846	13.0846	13.0708	13.0656	13.0639	13.0639
Medical	1.6271	1.3659	1.7798	1.7752	1.7752	1.7395	1.7333	1.7333	1.7333
Genbase	0.1709	0.0452	0.1859	0.1859	0.1859	0.1884	0.1884	0.1884	0.1884
Avg. Rank	1.33	4.17	7.5	5.42	5.42	5.42	6.08	4.92	4.75

one general issue with multilabel classification is that most data sets are highly imbalanced at the tag level, i.e., every tag has very few positive instances. Using logistic regression gives biased probability estimates on imbalanced datasets [17]. Since probabilistic classifier chains rely on predicting accurate probabilities for each classifier in the chain, such biased estimates may hamper overall performance. At a minimum, we believe better results can be obtained by post-processing the scores by isotonic regression [18,19].

Another issue is concerned with using cross-product features for large number of tags where learning linear models may pose scalability issues. To circumvent this problem, we may compute the cross-product (linear) kernel matrix, and, if the number of training instances is not high, use a kernel method (kernel SVM, kernel logistic regression). Otherwise, we can compute a low-dimensional representation of the feature space given the kernel matrix using, for example,

kernel PCA. An alternative (approximation) is to treat the rows (or columns) of the kernel matrix as features – the so-called empirical kernel map [20] – and train a linear SVM or a linear logistic regression using these features.

References

1. Dembczyński, K., Cheng, W., Hüllermeier, E.: Bayes optimal multilabel classification via probabilistic classifier chains. In: ICML. (2010)
2. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine Learning **85**(3) (2011) 333–359
3. Finley, T., Joachims, T.: Training structural SVMs when exact inference is intractable. In: ICML. (2008)
4. Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., Vapnik, V.: Kernel dependency estimation. In: NIPS. (2002)
5. Rai, P., Daumé III, H.: Multi-label prediction via sparse infinite CCA. In: NIPS. (2009)
6. Hsu, D., Kakade, S., Langford, J., Zhang, T.: Multi-label prediction via compressed sensing. In: NIPS. (2009)
7. Bi, W., Kwok, J.T.: Multilabel classification on tree- and DAG-structured hierarchies. In: ICML. (2011)
8. Cristianini, N., Shawe-Taylor, J., Elisseeff, A., Kandola, J.S.: On kernel-target alignment. In: NIPS. (2001)
9. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining **3**(3) (2007) 1–13
10. Tsoumakas, G., Katakis, I., Vlahavas, I.P.: Mining multi-label data. In: Data Mining and Knowledge Discovery Handbook. Springer (2010) 667–685
11. Sorower, M.S.: A literature survey on algorithms for multi-label learning. Technical report, Oregon State University., Corvallis, OR, USA (December 2010)
12. Dembczyński, K., Waegeman, W., Hüllermeier, E.: Joint mode estimation in multi-label classification by chaining. In: ECML Workshop - CoLISD. (2011)
13. Zaragoza, J., Sucar, L., Morales, E.: Bayesian chain classifiers for multidimensional classification. In: IJCAI. (2011)
14. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edition edn. Prentice-Hall, Englewood Cliffs, NJ (2003)
15. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2) (1968) 100–107
16. Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., Vlahavas, I.: Mulan: A java library for multi-label learning. Journal of Machine Learning Research **12** (2011) 2411–2414
17. King, G., Zeng, L.: Logistic regression in rare events data. Political Analysis **9**(2) (Spring 2001) 137–163
18. Zadrozny, B., Elkan, C.: Transforming classifier scores into accurate multiclass probability estimates. In: KDD. (2002)
19. Menon, A.K., Jiang, X., Vembu, S., Elkan, C., Ohno-Machado, L.: Predicting accurate probabilities with a ranking loss. In: ICML. (2012)
20. Schölkopf, B., Mika, S., Burges, C.J.C., Knirsch, P., Müller, K.R., Rätsch, G., Smola, A.J.: Input space versus feature space in kernel-based methods. IEEE Transactions on Neural Networks **10**(5) (1999) 1000–1017