Option Predictive Clustering Trees for Multi-label Classification

Tomaž Stepišnik, Dragi Kocev, Sašo Džeroski

Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
Jožef Stefan Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia
{tomaz.stepisnik, dragi.kocev, saso.dzeroski}@ijs.si

Abstract: In this work, we focus on the task of multi-label classification (MLC), where every example is associated with a set of labels. We present an algorithm for learning option predictive clustering trees (OPCTs) for MLC, based on the predictive clustering framework. The algorithm addresses the myopia of the standard tree induction algorithm by considering alternative splits in the internal nodes of the tree and introducing option nodes where appropriate. An option tree can be viewed as a compact representation of an ensemble, as well as, used as a pool of candidates from which a single tree can be extracted. This broadens the space of trees that is searched and reduces the myopia, compared to the standard tree induction. We evaluate the proposed OPCTs on 12 benchmark MLC datasets from different domains. Results show that OPCTs as ensembles can achieve performance similar to the bagging ensembles of PCTs, while the single trees extracted from OPCTs can outperform standard PCTs. We also perform parameter sensitivity analysis and provide avenues for future work.

Keywords: predictive clustering trees; option trees; multi-label classification; myopia

1 Introduction

The most widely studied machine learning task is binary classification where the goal is to predict whether an example belongs to a group/class or not. If the examples can belong to a single class from a given set of m classes ($m > 2$) the task is known as multi-class classification. In this work, we focus on the multi-label classification (MLC) task, where a single example can be assigned several labels (i.e., a subset of a given set of possible labels).

MLC is a well-established predictive modelling task. The methods addressing this task belong in two groups: problem transformation and algorithm adaptation methods [4]. The problem transformation methods transform the multi-label learning problem into one or more single-label classification problems, which a great number of machine learning algorithms are capable of solving.

The problem transformation methods can be further split into three categories: binary relevance, label power-set and pair-wise methods. Binary relevance methods use the one-against-all strategy to convert the problem into several binary classification problems. A closely related method is the classifier chain method and its ensemble extension [9]. Label power-set (LP) methods transform the problem into a single multi-class classification problem, where a separate class is created for every possible subset of original labels. In this way, LP based methods directly take into account the label correlations. Representative methods include HOMER [5] and RAkEL [3]. Pair-wise methods perform pair-wise or round robin classification, with binary classifiers using Q(Q - 1)/2 classifiers covering all pairs of labels [17]. To combine these classifiers, the pairwise classification method uses majority voting.

The algorithm adaptation methods customize existing machine learning algorithms for the task of MLC. There are extensions of the following machine learning algorithms: boosting, k-nearest neighbors, decision trees and neural networks. The extended methods are able to directly handle multi-label data. AdaBoost.MH and AdaBoost.MR [8] are two extensions of AdaBoost for multi-label data. While AdaBoost.MH is designed to minimize Hamming loss, AdaBoost.MR is designed to find a hypothesis which ranks the correct labels at the top. Several variants for multi-label learning (ML-kNN) of the popular *k*-Nearest Neighbors (kNN) lazy learning algorithm have been proposed [1]. Decision tree extension was proposed within the predictive clustering framework [22]. A single predictive clustering tree (PCT) is constructed by using a splitting criterion that considers all of the labels. The PCTs for MLC were also used in an ensemble setting [14]. Neural networks have been adapted for MLC by introducing a new error function that takes multiple labels into account [18].

An extensive experimental comparison [12] of 12 MLC methods on 11 datasets using 16 performance measures showed that ensembles of PCTs are a state-of-the-art method for the MLC task. However, as already pointed out, a common concern with decision tree based models is their myopia, resulting from the greedy induction algorithm used to learn them. For this reason, several alternatives were proposed, such as beam search induction [15] and option trees [19], aimed at reducing the myopia of the trees. Both approaches showed that they can improve the performance of standard decision trees, while [12] also showed that very large option trees achieve the performance of bagging ensembles of decision trees. Conversely, [10] argues that exhaustive searching of the model space often leads to an inferior generalization.

In this work, we extend predictive clustering trees (PCTs) for MLC with option nodes, thus forming option predictive clustering trees (OPCTs). An option tree can be seen as a condensed representation of an ensemble of trees which share a common substructure. For illustration, see Figures 2 and 6. We also examine different techniques for selecting the optimal embedded tree from the OPCT. In this way, we increase the space of trees searched by the algorithm. We evaluate

both OPCTs as ensembles and the best embedded trees extracted from OPCTs on several datasets from different domains. The goal of this paper is to see whether OPCTs as ensembles can achieve the performance of bagging ensembles for the MLC task, and if selecting the best embedded tree can reduce the myopia of the standard decision trees.

The remainder of this paper is organized as follows. Section 2 describes the algorithm for learning OPCTs for MLC. Next, Section 3 outlines the design of the experimental evaluation. Section 4 continues with a discussion of the results. Finally, we conclude and provides possible directions for further work.

2 Option Predictive Clustering Trees

The predictive clustering trees framework views a decision tree as a hierarchy of clusters. The top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. The PCT framework is implemented in the CLUS system [23] available at *http://clus.sourceforge.net*.

OPCTs extend the PCT framework by introducing option nodes into the tree building procedure. Option decision trees were first introduced as classification trees by Buntine [19] and then analyzed in more detail by Kohavi and Kunz [13]. Ikonomovska et al. [16] analyzed regression option trees in the context of data streams. We also evaluated OPCTs for the multi-target regression task [11] and hierarchical multi-label classification task [7].

The motivation for the introduction of option trees is to address the myopia of the *top-down induction of decision trees* (TDIDT) algorithm [20]. From the perspective of the predictive clustering framework, a PCT is a non-overlapping hierarchical clustering of the whole input space. Each node (subtree) corresponds to a clustering of a subspace and prediction functions are placed in the leaves, i.e., the lowest clusters in the hierarchy. In contrast, an OPCT allows the construction of an overlapping hierarchical clustering. This means that at each node of the tree several alternative hierarchical clusterings of the subspace can appear instead of a single one. When using TDIDT to construct a predictive clustering tree, all possible splits are evaluated by using a heuristic, and the best split is selected. However, other splits may have very similar heuristic values and the difference between them could be a consequence of noise or sampling that generated the data. In this case, selecting a different split could be optimal. To address this concern, the use of option nodes was proposed [13].

Figure 1 presents the TDIDT algorithm modified for the induction of OPCTs. The function call *FindBestTests(E,O)* returns the *O* best tests according to the heuristic score in descending order (best first). Every test is represented as a triplet *(t,h,P)*,

where t is the actual test function, h is its heuristic value and P is the set of partitions produced by the test.

Procedure OptionPCT
Input: A data set E, parameter ε, maximum number of options O, current tree level l, maximum level for option nodes L
Output: An option predictive clustering tree
$candidates = \text{FindBestTests}(E, O)$
if $|candidates| > 0$ **then**
  if $|candidates| = 1$ **or** $l > L$ **then**
    $(t^*, h^*, \mathscr{P}^*) = candidates[0]$
    for each $E_i \in \mathscr{P}^*$ **do**
      $tree_i = \text{OptionPCT}(E_i, \varepsilon, O, l+1, L)$
    return $\text{node}(t^*, \bigcup_i \{tree_i\})$
  else
    $(t_0^*, h_0^*, \mathscr{P}_0^*) = candidates[0]$
    $nodes = \{\}$
    for each $(t_i^*, h_i^*, \mathscr{P}_i^*) \in candidates$ **do**
      if $\frac{h_i^*}{h_0^*} \geq 1 - \varepsilon$ **then**
        for each $E_j \in \mathscr{P}_i^*$ **do**
          $tree_j = \text{OptionPCT}(E_j, \varepsilon, O, l+1, L)$
        $nodes = nodes \cup \{\text{node}(t^*, \bigcup_j \{tree_j\})\}$
    if $|nodes| > 1$ **then**
      return $\text{option\_node}(nodes)$
    else
      return $nodes[0]$
else
  return $\text{leaf}(\text{Prototype}(E))$

Figure 1
The top down induction algorithm for option PCTs

The main component of the algorithm is the heuristic score used to evaluate the splits. For the MLC task, sets of labels are presented as binary vectors, where every component denotes the presence (1) or absence (0) of one label. For every component the Gini index is calculated, and their average is the final heuristic score. The algorithm introduces an option node into the tree when the best splits have similar heuristic values. Instead of selecting only the best split, we select every split s that satisfies the condition:

$$\frac{Heur(s)}{Heur(s_{best})} \geq 1 - \varepsilon \quad (1)$$

where s_{best} is the best split and ε determines how similar the heuristics must be. E.g., when ε=0.1, we are selecting only splits whose heuristics are within 10% of the best split. Typical values of ε are in the [0, 1] interval. After we have determined the candidate splits, we introduce an option node whose children are split nodes containing the selected splits. When no suitable test is found, a leaf node is created that stores the prototype of examples sorted to that leaf. For the MLC task, the *i*-th component of the prototype is the average value of the *i*-th components of examples in that leaf.

Introducing an option node with a large number of options is not advised [13] as it can lead to the explosion of model sizes. Therefore, we limit the maximum number of options included in a single option node (parameter *O*). To further limit the exponential growth of the tree, we also set the maximum depth in the tree where an option node can be introduced (parameter *L*). This means that on levels deeper than *L*, the tree is built following the standard TDIDT algorithm. This follows our intuition that splits lower in the tree, where clusters are already more homogeneous, are less important than splits higher in the tree.

Once an OPCT is learned, we use it to make predictions. In a regular PCT an example is sorted into a leaf (reached according to the tests in the nodes of the tree) where the prototype stored in that leaf is predicted. Traversing an example through an OPCT is the same for split nodes and leaves. When we encounter an option node, however, we traverse the example down each of the options. This means that in an option node an example is sorted to multiple leaves, where multiple predictions are produced. To obtain a single prediction in an option node, we aggregate the obtained predictions. For the MLC task, the aggregation is simple component-wise averaging of the vectors, as it is done in PCT ensembles. The resulting vector gives us pseudo-probabilities of every label, to which a threshold can be applied to select the actual labels.

An option tree is usually observed as a single tree, however, it can also be interpreted as a compact representation of an ensemble. We can extract *embedded trees* out of an option tree by replacing every option node with one of its options (Figure 2). A given OPCT is also an extension of the PCT learned on the same data. By definition, whenever we introduce an option node, the best split is one of the options included. Consequently, the PCT is an embedded tree in the OPCT, resulting from replacing all option nodes with the best option.

In addition to using an option tree as an ensemble of embedded trees, we can also extract a single embedded PCT out of it. This increases the space of trees searched by the TDIDT algorithm and directly addresses its myopia. The simplest way to select a single embedded PCT is to use the error on the training dataset as the selection criterion. This is an attractive option since it requires very little extra work and no extra data, but can lead to overfitting.

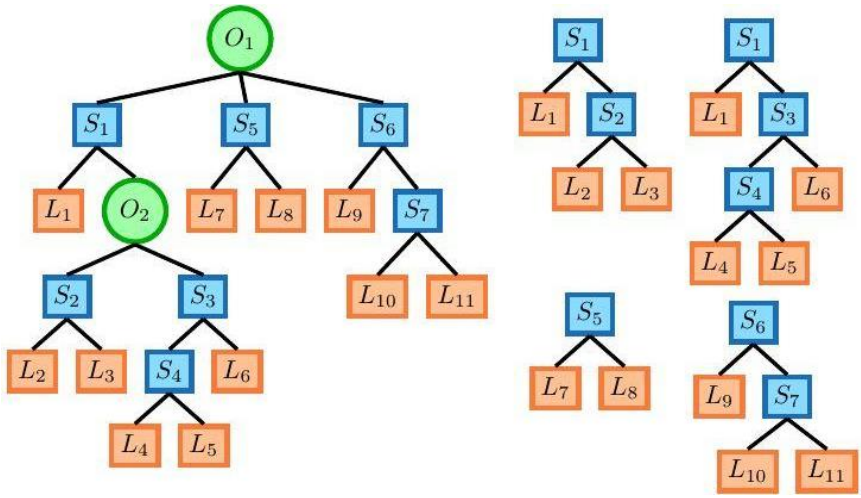

Figure 2

An option tree (left) and the ensemble of its embedded trees (right). O_i are option nodes, S_j split nodes and L_k leaf nodes.

Another option is to use a part of the training data as a separated validation set, which would not be used for the initial learning of the OPCT, but would be utilized to determine which of the embedded trees has the best predictive performance. We can also use expert knowledge to select the embedded tree. Providing a domain expert with an option tree gives them a lot of choices with regards to the model. This approach also has the advantage that the domain expert need not be available for interaction when the model is learned, but can assess the OPCT and chose the preferred options later on.

3 Experimental Design

The experimental evaluation was performed on 12 datasets from biology, text classification and multimedia domains. They are described in Table 1. All datasets except *CAL500* were retrieved pre-divided into training and testing sets and we used them in their original format to facilitate easier comparison of the results. The *CAL500* dataset was randomly split on training and test sets. All 12 datasets can be found at *http://mulan.sourceforge.net/datasets-mlc.html*.

There are many measures of performance used for the MLC task [12]. In our comparison we focus on Area Under the Average Precision-Recall Curve ($\overline{AUPRC}$) [2]. It combines the pseudo-probabilities of all the different labels as if they belonged to a single binary classification problem, and calculates the area under the precision-recall curve from them. A nice feature of $\overline{AUPRC}$ is that it is

a threshold independent measure, so we do not have to worry about setting the threshold at which the labels are predicted in the leaves (as explained in Section 2). In the appendix we include the results for two additional performance measures (Ranking Loss and One Error [12]), from which similar conclusions can be made.

Table 1

Properties of the datasets used in the study: number of examples in the training/testing datasets (Ntr/Nte), number of descriptive attributes (discrete/continuous, *D*/*C*), the total number of labels (*Q*) and label cardinality (l_c)

	Ntr/Nte	*D*/*C*	*Q*	l_c
bibtex	4880/2515	1836/0	159	2.40
birds	322/323	2/258	19	1.01
CAL500	302/200	0/68	174	26.0
corel5k	4500/500	499/0	374	3.52
emotions	391/202	0/72	6	1.87
enron	1123/579	1001/0	53	3.38
flags	129/65	9/10	7	3.39
genbase	463/199	1185/0	27	1.25
medical	645/333	1449/0	45	1.25
scene	1211/1196	0/294	6	1.07
tmc2007-500	21519/7077	500/0	22	2.16
yeast	1500/917	0/103	14	4.24

We used the 12 benchmark datasets to evaluate OPCTs. Firstly, we wanted to see how the OPCTs as ensembles of embedded PCTs compare to the bagging ensembles of PCTs [21]. Specifically, we were interested in the trade-off between the performance and the size of the model. For the size of the model we looked at the total number of leaves in the tree(s). For the bagging ensembles of PCTs, the number of trees in the ensemble is the primary way to control the size of the model. We ran the experiments with 10, 25, 50, 100 and 125 PCTs in the ensemble.

For OPCTs, we can influence the number of option nodes introduced into the tree by using different parameter values. We considered different values for the ε parameter from the set {0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0}. When ε=1, option nodes with the best *O* splits are always introduced at the top *L* levels of the tree. We also tried different settings of the *O* and *L* parameters. Specifically, we used pairs (*O*, *L*) from the set {(10, 2), (5, 3), (3, 4)}. In addition to a better understanding of the performance vs. size trade-off, this also gives us insight into what is more valuable in OPCTs: more options at the top of the tree, or allowing option nodes lower in the tree. The pairs were selected in a way that at the maximum amount of option nodes introduced (e.g., at ε=1), the OPCTs would aggregate a similar amount of predictions compared to bagging ensembles of 100

PCTs. An important thing to note here is that different selections of parameters can still produce the same OPCT, if for a given dataset the same splits satisfy both criteria.

We also compared standard PCTs to the best embedded trees extracted from OPCTs (from here on referred to as BestEmbedded trees). We tried two approaches to extracting the BestEmbedded trees from OPCTs: selecting the embedded tree based on the training set performance, and based on the performance on a separate validation set not used to build the original OPCT. Both PCTs and BestEmbedded trees were post pruned using the training set in the first set of experiments, and using the validation set in the second set. This way, standard PCTs also benefited from the validation set, thus enabling a fair comparison. Note that BestEmbedded trees therefore use the validation set both for embedded tree selection and pruning. BestEmbedded trees were extracted from various OPCTs built with the same parameter settings described in the previous paragraph. This gave us insight into how different parameters influence the performance of the BestEmbedded trees. As a validation set, we randomly selected 20% of examples from the initial training set.

4 Results and Discussion

4.1 Comparison of OPCTs and Bagging Ensembles

Figure 3 shows the trade-off between the performance and size of the model for bagging ensembles of PCTs and OPCTs as ensembles. We can see that increasing the number of options improves the performance of OPCTs. In contrast to adding more trees to an ensemble, this is not guaranteed, since additional options include split nodes with lower heuristic scores. The performance improvement saturates at the largest OPCTs and often reaches the performance of bagging ensembles (on 7 datasets). All models seem to perform equally well on the *genbase* dataset, and much better than single PCTs (results on Figure 4).

Note that the comparison of the sizes of the trees is closely related to the comparison of running times. The most time consuming items when learning a tree are the evaluations of the candidate splits and then splitting the data. This is done in the same way in both PCTs (and ensembles thereof) and OPCTs, therefore, similar number of nodes in the trees indicates similar time needed for their induction.

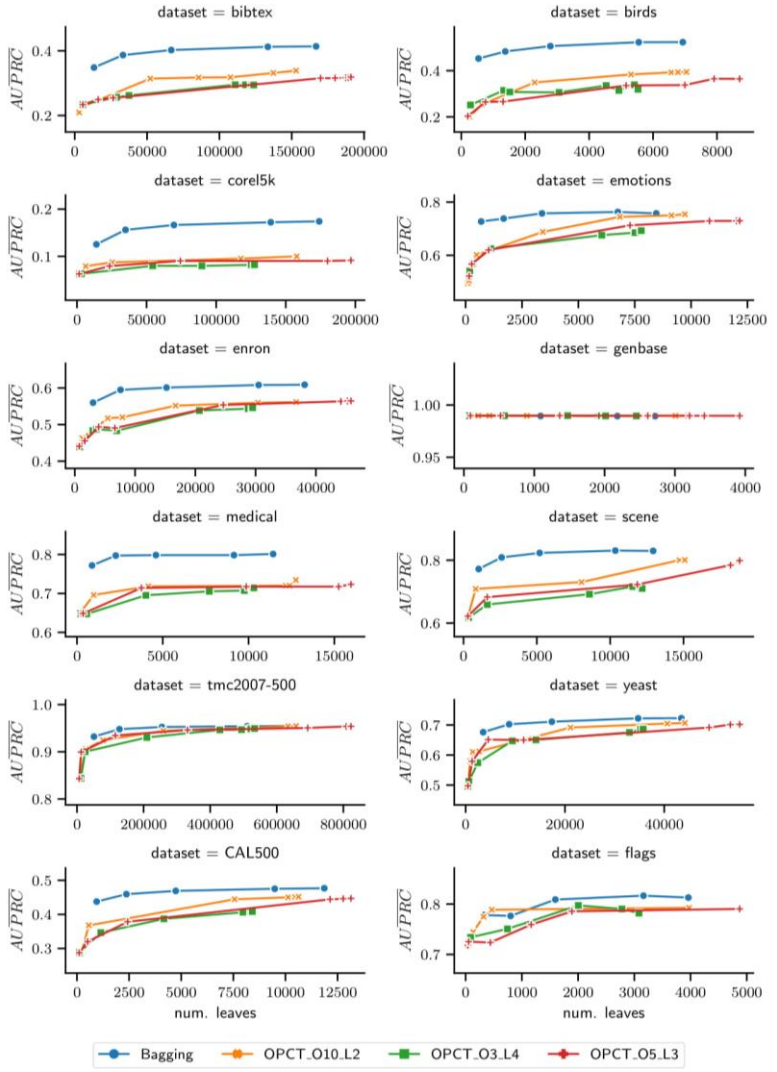

Figure 3

Trade-off between size and performance of bagging ensembles of PCTs and OPCTs. The size of ensembles is influenced by the number of trees, whereas the size of OPCTs is mainly influenced by the number of option nodes introduced into the tree. Note that every graph is on a different scale.

By comparing the results of OPCTs with different settings of parameters O and L, we can see that the pair $(O, L) = (10, 2)$ offers the best performance, followed by $(O, L) = (5, 3)$. This confirms our intuition that option nodes lower in the tree are less useful and do not offer enough improvement in performance to justify the increased model size. This is especially evident on the *bibtex*, *birds*, *emotions* and *scene* datasets.

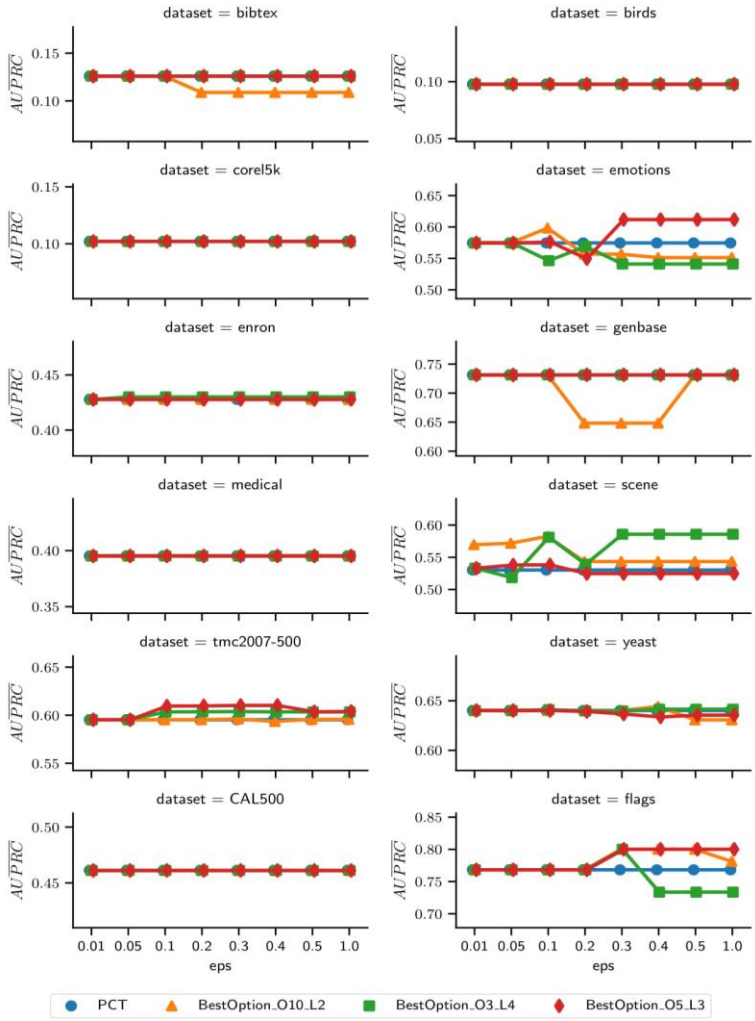

Figure 4

Performance of BestEmbedded trees produced by different parametrizations of OPCTs and selected based on training set performance, compared to the performance of a single PCT. Note that every graph is on a different scale.

4.2 Comparison of BestOptions and Standard PCTs

Figures 4 and 5 show the comparison of standard PCTs and the BestEmbedded trees extracted from OPCTs with different parameters. Figure 4 shows the results for BestEmbedded trees selected based on the training set performance, and Figure 5 based on the validation set performance. First thing we can notice is that

the BestEmbedded trees and PCTs often have very similar or identical performance. There are usually no significant differences in size as well. In fact, often the BestEmbedded tree is the same as the standard PCT, especially when it is selected based on the training set. This is not surprising, since standard PCTs choose splits that perform best on the training set at each step. We can also see that having a separate validation set for pruning can greatly improve the performance of both standard PCTs and BestEmbedded trees (the *bibtex*, *genbase*, *medical* and *tmc2007* datasets).

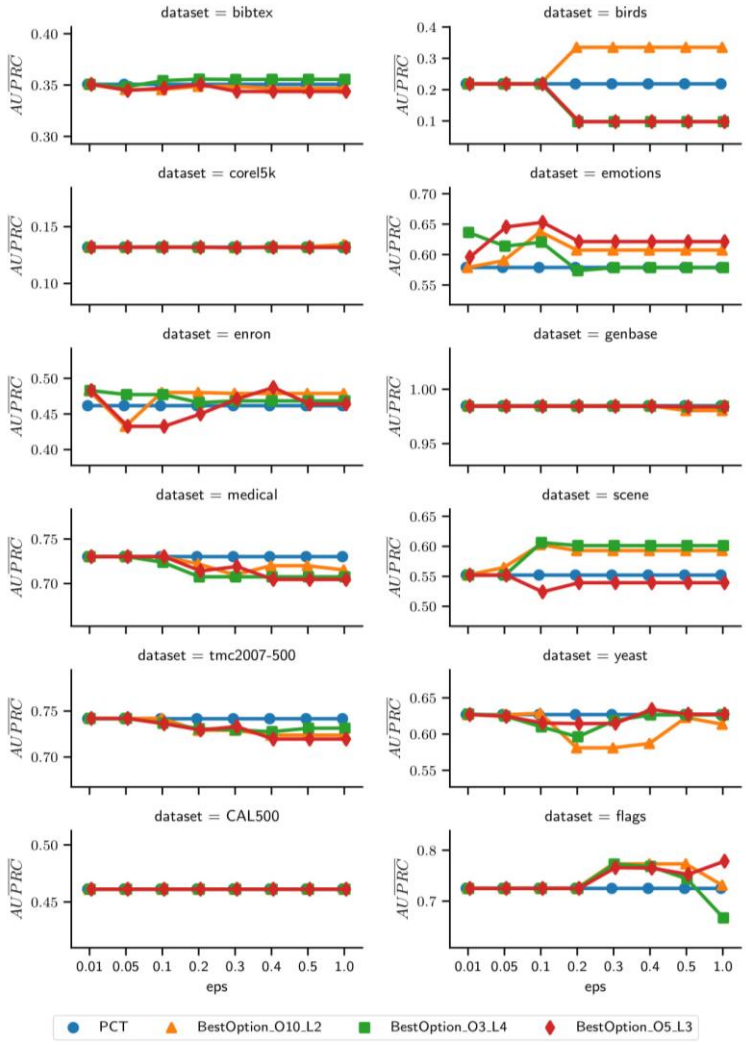

Figure 5
Performance of BestEmbedded trees produced by different parametrizations of OPCTs and selected based on validation set performance, compared to the performance of a single PCT. Note that every graph is on a different scale.

Another thing that the results show is that for extracting BestOptions, option nodes deeper in the tree are still helpful. While the parameter values (O, L) = (10, 2) always provided the best results for OPCTs as ensembles, even parameter values (O, L) = (3, 4) often lead to the best selection of BestEmbedded trees.

With larger OPCTs a larger space of trees is searched when selecting BestEmbedded trees, and that is where there are more differences compared to standard PCTs. Even when selected based on the training set performance, BestEmbedded trees often perform better than standard PCTs (on the *scene*, *tmc2007* and mostly *flags* datasets). But it also frequently happens that the BestEmbedded tree generalizes worse than the standard PCT (the *bibtex*, *emotions*, *genbase* and *flags* datasets).

Using a validation set to select the BestEmbedded tree mostly helps in selecting a tree that generalizes better, when compared to the train set selection. This can be nicely seen on the *bibtex*, *emotions*, *enron*, *scene* and *flags* datasets. However, in some cases it leads to a worse selection relatively to the standard PCT, as seen on the *medical*, *tmc2007* and *yeast* datasets. The cause for this may be oversearching, as described in [10]. By searching the space of trees more exhaustively, we have a higher chance of discovering "fluke" theories that fit the data well, but generalize poorly. As opposed to what is commonly understood as overfitting, these "fluke" theories are not overly complex. For example, when selecting the BestEmbedded tree, the standard PCT is always one of the options. It often happens that the tree with the best performance on the validation set is smaller (represents a simpler theory) than the standard PCT, but it still has a poorer performance on the test set.

4.3 Use Case Demonstration

We demonstrate the usefulness of OPCTs on the *emotions* dataset [6]. The examples in the *emotions* dataset are 30 seconds long music samples extracted from 100 different songs from different genres. The goal is to label the music samples with the emotions present in the music. There are 6 possible labels corresponding to 6 emotional clusters. The music samples are described with 8 rhythmic features and 64 timbre features.

Figure 6 shows an example of an OPCT trained on the *emotions* dataset for illustrative purpose. It was pruned to reduce the tree size to a humanly manageable proportions and contains only one option node with three options.

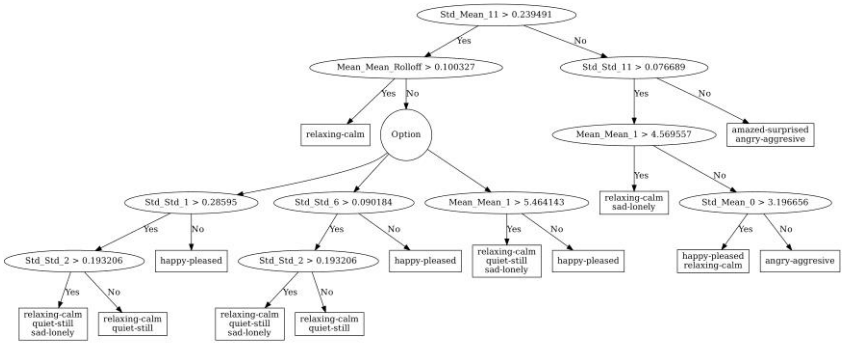

Figure 6

A heavily pruned OPCT constructed on the *emotions* dataset. Option nodes are represented as circles, standard split nodes have oval shape and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Figure 7 shows a PCT trained using the standard top down induction algorithm.

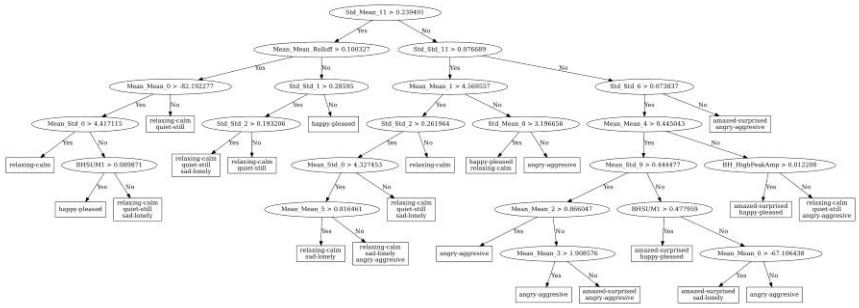

Figure 7

A standard PCT constructed on the *emotions* dataset. Split nodes have oval shapes and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Figure 8 shows the BestEmbedded tree selected from OPCT_O5_L3 with ε=0.2. They were both pruned using the validation set. The standard PCT has 22 leaf nodes and it achieved 0.57 $AU\overline{PRC}$, while the BestEmbedded tree is much smaller with only 9 leaves and has better predictive performance with 0.62 $AU\overline{PRC}$. We can see that they differ already at the root node.

This demonstrates the ability of OPCTs to help us find better performing trees, which can even be smaller than standard trees. Ideally, the BestEmbedded trees can be selected by experts, looking at the OPCT and selecting the options that are consistent with their knowledge.

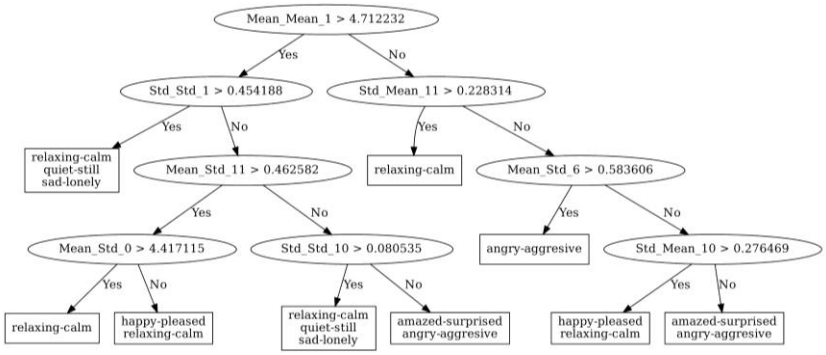

Figure 8
The BestEmbedded tree selected from OPCT_O5_L3 with ε=0.2 constructed on the *emotions* dataset. Standard split nodes have oval shapes and leaf nodes are shown as boxes containing the labels predicted in that leaf.

Conclusions

In this work, we present an algorithm for learning option predictive clustering trees for the task of multi-label classification, where every example is associated with a set of labels. OPCTs are global models that predict the entire label sets simultaneously and belong in the algorithm adaptation group of methods. The main purpose of the proposed algorithm is to reduce the myopia of the standard greedy algorithm for learning PCTs. During tree construction, if multiple tests have heuristic scores similar to that of the best test, an option node is created that contains a set of alternative sub-nodes, called options, i.e., a set of the best performing splits. When predicting the labels of an example, each option produces a prediction, and predictions from different options are then aggregated in the option node. This leads us to consider option trees as condensed representations of an ensemble of trees.

In addition to the pseudo-ensemble aspect of OPCTs, we can also look at them as a set of possible PCTs (embedded trees), and select one of them as the final model. This broadens the search over the space of possible trees and decreases the myopia of the algorithm. There are multiple ways of selecting the best embedded trees. In our experiments, we used the performance on the training set, and the performance on a validation set. Another option is to use expert knowledge to select the best options in the option tree.

We performed an experimental evaluation of the OPCT method and the BestEmbedded trees, and compared them to standard PCTs and bagging ensembles of PCTs. The evaluation was performed on 12 datasets appropriate for the MLC task, and we measured model performance with $AU\overline{PRC}$. The results show that large OPCTs can often reach the performance of tree ensembles, and that BestEmbedded trees can outperform standard trees. Unfortunately, we can also fall in the trap of oversearching and find BestEmbedded trees with worse

predictive performance compared to standard PCTs. We also looked into how different parameters affect the performance of OPCTs and BestEmbedded trees. We found that allowing more option nodes higher in the tree, while limiting them on the lower levels (parameter values $(O, L) = (10, 2)$) works best for OPCTs as ensembles. For BestEmbedded trees, option nodes lower in the tree are still useful and can help find better trees. We demonstrated the potential of OPCTs on a music emotion labelling dataset, where the BestEmbedded tree was much smaller than the standard PCT, yet had a better predictive performance.

There are several paths for further work. We would like to evaluate OPCTs and BestEmbedded trees in a domain where an expert would help select the best splits in the option nodes. Additionally, we plan to investigate the use of OPCTs for feature ranking and selection for MLC datasets.

Acknowledgement

We acknowledge the financial support of the Slovenian Research Agency via the grants P2-0103 and a young researcher grant to TS, as well as the European Commission, through the grants MAESTRA (Learning from Massive, Incompletely annotated, and Structured Data) and HBP (The Human Brain Project), SGA1 and SGA2. SD and DK also acknowledge support by Slovenian Research Agency (via grants J4-7362, L2-7509, N2-0056 and J2-9230), the European Commission (the LANDMARK project). The computational experiments presented here were executed on the computing infrastructure from the Slovenian Grid (SLING) initiative.

References

[1] M. L. Zhang and Z. H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognition*, Vol. 40, pp. 2038-2048, 2007

[2] C. Vens, J. Struyf, L. Schietgat, S. Džeroski and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Machine Learning*, Vol. 73, pp. 185-214, 2008

[3] G. Tsoumakas and I. Vlahavas, "Random k-Labelsets: An Ensemble Method for Multilabel Classification," in *Proc. of the 18th European conference on Machine Learning*, 2007

[4] G. Tsoumakas, I. Katakis and I. Vlahavas, "Mining Multi-label Data," in *Data Mining and Knowledge Discovery Handbook*, Springer Berlin / Heidelberg, 2010, pp. 667-685

[5] G. Tsoumakas, I. Katakis and I. Vlahavas, "Effective and Efficient Multilabel Classification in Domains with Large Number of Labels," in *Proc. of the ECML/PKDD Workshop on Mining Multidimensional Data*, 2008

[6] K. Trochidis, G. Tsoumakas, G. Kalliris and I. Vlahavas, "Multi-label classification of music into emotions," 2008

[7] T. Stepišnik Perdih, A. Osojnik, S. Džeroski and D. Kocev, "Option Predictive Clustering Trees for Hierarchical Multi-label Classification," in *Discovery Science*, 2017

[8] R. E. Schapire and Y. Singer, "BoosTexter: A Boosting-based System for Text Categorization," *Machine Learning,* Vol. 39, No. 2, pp. 135-168, 2000

[9] J. Read, B. Pfahringer, G. Holmes and E. Frank, "Classifier chains for multi-label classification," *Machine Learning,* Vol. 85, pp. 333-359, 2011

[10] J. R. Quinlan and R. M. Cameron-Jones, "Oversearching and Layered Search in Empirical Learning," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, San Francisco, CA, USA, 1995

[11] Osojnik, S. Džeroski and D. Kocev, "Option predictive clustering trees for multi-target regression," in *Discovery Science: 19th International Conference (DS 2016)*, 2016

[12] G. Madjarov, D. Kocev, D. Gjorgjevikj and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition,* Vol. 45, pp. 3084-3104, 2012

[13] R. Kohavi and C. Kunz, "Option Decision Trees with Majority Votes," in *Proceedings of the 14th International Conference on Machine Learning,* San Francisco, CA, USA, 1997

[14] D. Kocev, C. Vens, J. Struyf and S. Džeroski, "Tree ensembles for predicting structured outputs," *Pattern Recognition,* Vol. 46, pp. 817-833, 2013

[15] D. Kocev, J. Struyf and S. Džeroski, "Beam search induction and similarity constraints for predictive clustering trees," in *Proc. of the 5th Intl Workshop on Know. Disc. in Inductive Databases KDID - LNCS 4747*, 2007

[16] E. Ikonomovska, J. Gama, B. Zenko and S. Dzeroski, "Speeding-Up Hoeffding-Based Regression Trees With Options," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011

[17] J. Fürnkranz, "Round robin classification," *Journal of Machine Learning Research,* Vol. 2, pp. 721-747, 2002

[18] K. Crammer and Y. Singer, "A family of additive online algorithms for category ranking," *Journal of Machine Learning Research,* Vol. 3, pp. 1025-1058, 2003

[19] W. Buntine, "Learning classification trees," *Statistics and Computing*, Vol. 2, pp. 63-73, 1992

[20] L. Breiman, J. Friedman, R. A. Olshen and C. J. Stone, Classification and Regression Trees, Chapman & Hall/CRC, 1984

[21] L. Breiman, "Bagging Predictors," *Machine Learning*, Vol. 24, pp. 123-140, 1996

[22] H. Blockeel, L. D. Raedt and J. Ramon, "Top-down induction of clustering trees," in *Proceedings of the 15th International Conference on Machine Learning*, 1998

[23] H. Blockeel and J. Struyf, "Efficient algorithms for decision tree cross-validation," *Journal of Machine Learning Research*, Vol. 3, pp. 621-650, 2002

Appendix – additional results

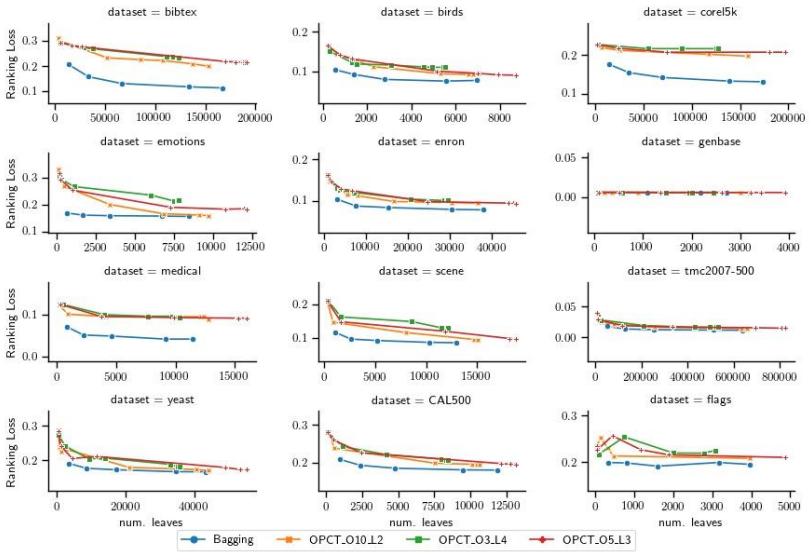

Figure 9

Comparison of bagging and OPCTs using the ranking loss measure. For details see Figure 3.

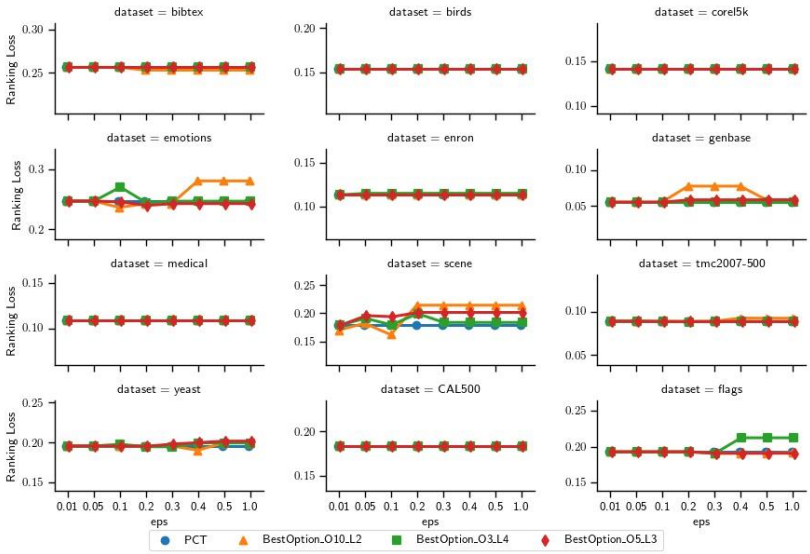

Figure 10
Comparison of PCTs and BestOption trees selected based on the training set performance using the ranking loss measure

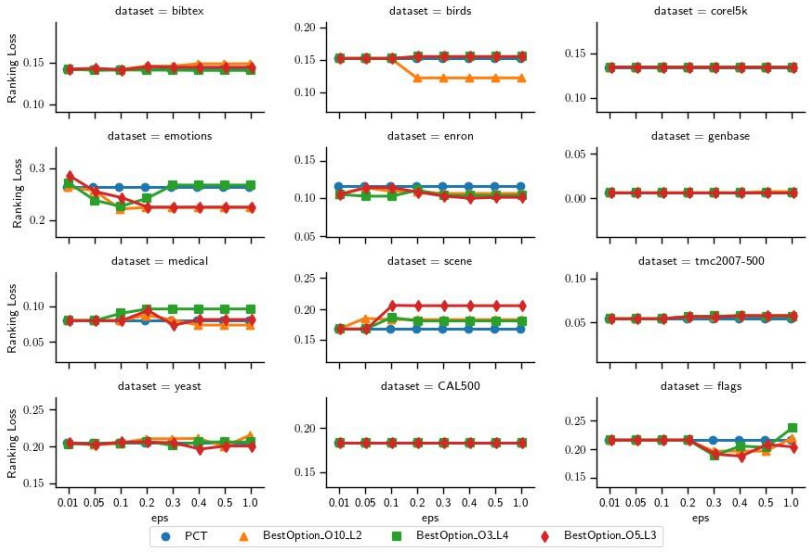

Figure 11
Comparison of PCTs and BestOption trees selected based on the validation set performance using the ranking loss measure

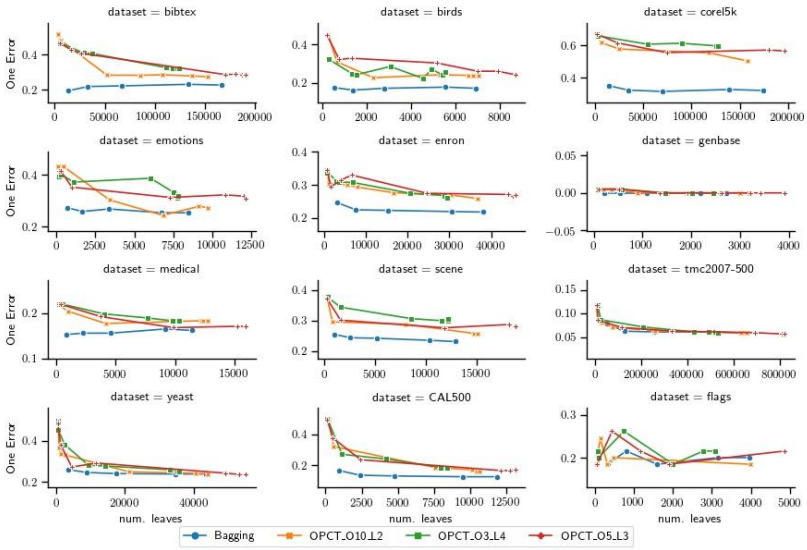

Figure 12

Comparison of bagging ensembles and OPCTs using the one error measure. For details see Figure 3.

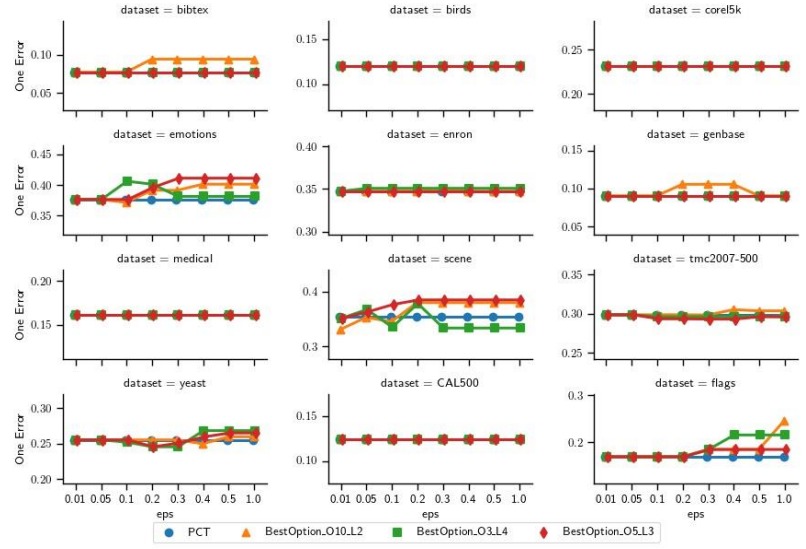

Figure 13

Comparison of PCTs and BestOption trees selected based on the training set performance using the one error measure

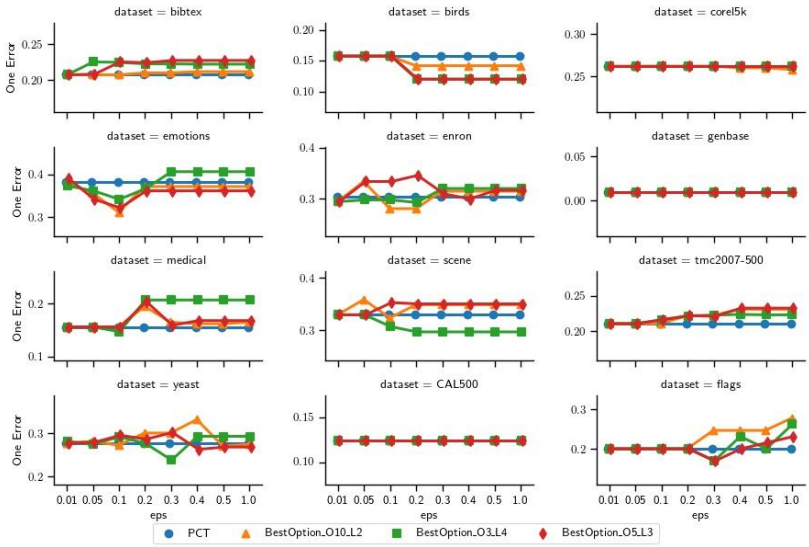

Figure 14

Comparison of PCTs and BestOption trees selected based on the validation set performance using the one error measure