Multi-output and Structured-output Learning with Chaining Methods (Agent-based inference)

Alicia Fortes Machado Aloysio Galvão Lopes
Igor Albuquerque Silva Iago Martinelli Lopes

December 2020

1 Abstract

Classifier Chains appeared in the literature more than ten years ago and represent an effective method to attack the Multi-Label Classification problem. However, finding the right path in the associated probability tree is still a problem when we face an increasing number of labels. In order to tackle that, we rely on how similar problems have been dealt with such as Go [1] and graph path [2]. Thus, we propose a reinforcement learning approach that uses an RNN to encode states and value and policy neural networks to tackle the problem, as well as a Monte Carlo Tree Search for the policy improvement method.

2 Introduction

Multi-Label Classification (MLC) has attracted attention over the past few years due to its capacity of improving the prediction of problems as such movie genre, textual data and protein function classification. The key difference when compared to other classification problems is that the object can be classified in multiple labels, which aren't necessarily independent.

There are different methods in the literature to tackle this problem, such as RAkEL [3], Recurrent Neural Networks [4] and Classifier Chains (CC) [5]. The last one is of particular interest, given that it's one of the state of the art options for MLC and that it has been researched for a couple of years and showed improvements along this period of time. It's a more refined technique when compared to Binary Relevance (BR) one, since it considers that previously predicted variables might influence on others, while BR predicts each variable independently. Mathematically [5], a CC can be described as:

$$P(\mathbf{y}|\mathbf{x}) = P(y_1|x)\prod_{i=2}^{L} P(y_i|\mathbf{x}, y_1, ..., y_{i-1}),$$

where L, $\mathbf{y}$ and $\mathbf{x}$ correspond to the number of labels, the prediction and the input, respectively.

Notice that any permutation of the labels is valid. The discussion behind which order of labels to use in our CC and, consequently, in this tree, is not the focus of our work. However, we will briefly discuss this topic in Section 3. At first, we simply consider random choices of label orders in this work.

As the name says, a CC consists of a chain of binary classifiers. Once trained, the prediction step in this model can be seen as a probability tree, where each path from root to leaf represents

one combination of labels $\mathbf{y} \in \{0,1\}^L$. The probabilities in the non-leaf nodes are the predictions for the next label from the estimator considering the past actions.

However, even after considering an order for the labels and constructing the probability tree, we still need to find a way to go through the last one with a good quality-time tradeoff. Maximizing the probabilities of the chosen path in this tree is important to maximize the model's performance. Some simple methods to do that like Greedy and Exhaustive Search are described in Section 3, but they fail to balance both factors. Hence the need to find better methods to walk these trees.

Inspired by M-walk [2] and AlphaGo Zero [1], we decided to follow an approach that merges different techniques together: Deep Q-Learning, Policy Learning, Monte Carlo Search Tree (MCTS) and RNN. First, we want to use a RNN to encode the path followed until the current state. Then, for each state, we will use MCTS to search promising paths and then take the one with greatest expected value. The estimation of expected value and policy will be done using neural networks, as explained in section 4.

In Appendix A we also present our preliminary results, in which we reproduced other inference methods in order to understand them better and to have baseline results to compare with our model. We considered different methods such as Greedy, Exhaustive Search, Monte Carlo and ϵ-approximate inference. Our comparison was done by measuring subset 0/1 and Hamming losses, average number of explored nodes and average inference time for each approach.

Finally, one of the limitations that we might face is training power, since some datasets are large. And the other one is the coordination of different techniques in our model.

3 Related Work

3.1 Inference Methods

Inference methods are algorithms applied in the prediction step of a CC. Due to the inherent necessity to determine an order of the estimators in the chain, the prediction step may be done in different manners. An example of this is if in the prediction of an instance of the data the first estimator predicts its label with equal probabilities of `True` and `False`. It may be interesting in this case to analyze both possibilities when feeding the label value for the next estimators, because they may give a high probability if the first label is `True`, for example, and low probability if it is `False`.

Many CC inference methods have been proposed before. In this subsection we'll go over some of those, more precisely the greedy and exhaustive-search methods [6], ϵ-approximation inference [7], beam-search [8] and two Monte Carlo approaches [7, 9]. An overview and comparison of these algorithms was done in [10], and in Appendix A we'll present a reproduction of their results.

Greedy and Exhaustive Search (GS and ES)

The greedy method was first presented alone as CC, and was followed by the exhaustive search extension, the PCC. The former is the simplest and quickest way of doing an inference: each estimator will use the labels predicted by the previous ones in the chain to make their inference. The exhaustive search, however, will use every possibility of labels on each estimator and take the combination with the lowest loss to make its prediction.

A way to visualize this is by drawing a tree of labels, like the one in Figure 1. This tree represents a chain predicting only two labels, in which $\mathbf{x}$ is the test instance and the objective is to predict a vector $\mathbf{y}$. In a greedy search inference, the chosen path is always the ones following the highest conditional probability $\hat{P}(y_i|y_{i-1},...,y_0,\mathbf{x})$, which are the probabilities given by each individual estimator, in this case 0.6 and 0.5. The final probability is $\hat{P}(\mathbf{y}|\mathbf{x}) = 0.3$. An exhaustive search would transverse all of the four paths one by one and find a better probability of 0.36.

The exhaustive search can also maximize another expression: $\sum_i \hat{P}(y_i|y_{i-1}, ..., y_0, \mathbf{x})$, which is the sum of all of the conditional probabilities, instead of the multiplication. The difference in this is that it would minimize the Hamming loss, which is given by the ratio of incorrectly predicted labels over the total. In the example the exact-match (or subset 0/1) loss is minimized, which is 1 if all labels are correctly predicted and 0 otherwise [9].

ϵ-Approximation (ϵ-A)

The exhaustive search method is $\mathcal{O}(2^L)$, where L is the number of labels. Naturally, it is necessary to find alternatives to find better predictions then the greedy algorithm which can be applied to datasets with large number of labels. The first example of those is the ϵ-Approximation method.

The idea behind this algorithm is to start as an exhaustive search and stop looking through a path if its joint probability $\hat{P}(y_i|y_{i-1}, ..., y_0, \mathbf{x})$ becomes less than a value ϵ. This way, the number of visited paths is limited by $\mathcal{O}(L/\epsilon)$ [7, 10].

Beam Search (BS)

Another way of limiting the number of total paths is by specifying a number of maximum paths to explore at each level of the tree. The Beam Search algorithm transverses the tree in a breadth-first-search, limiting the number of paths being explored by a value b, called the beam [8]. The paths with lowest score are discarded (as in the exhaustive search, the score can be the one that minimizes the exact-match or Hamming loss.) in each level. This algorithm is $\mathcal{O}(Lb)$.

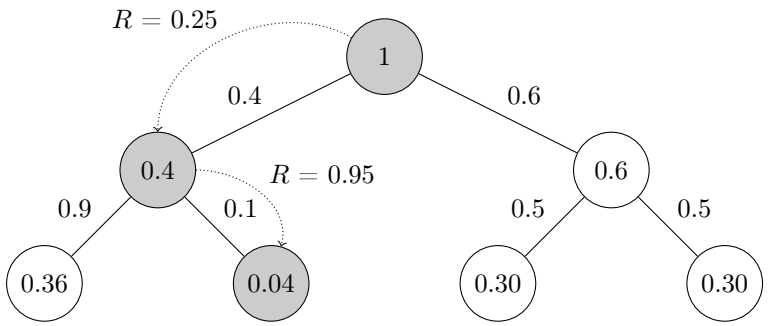
Monte Carlo Sampling (MC and EMC)

There are two different Monte Carlo techniques for inference, which are based on the same idea. In both of them a number k of paths are tried out, and they differ in the way the best path is chosen.

Each path will be transversed in a probabilistic way following the distributions of each estimator. This way, in each node of the tree a random number R is generated and the label will be `False` if $\hat{P}(y_i = \texttt{False}|y_{i-1}, ..., y_0, \mathbf{x}) < P$ and `True` otherwise. Figure 1 shows an example of a path explored in a CC with two estimators.

The first way of selecting the best path between the ones explored is by taking the one which occurred the most, the mode [11]. In the second approach the best path is selected by taking the one with the highest score [9]. The advantage of the latter is that it doesn't require the calculation of the mode, which requires storing all of the unique explored paths.

Figure 1: An example of a Monte Carlo inference path in a chain with two estimators. Darker nodes represent visited nodes. The number on the edges represent $\hat{P}(y_i|y_{i-1}, ..., y_0, \mathbf{x})$ and the numbers inside the nodes represent $\prod_i \hat{P}(y_i|y_{i-1}, ..., y_0, \mathbf{x})$. Example based on [10].



3.2 Chain Structure

In the context of chain classification, a part as crucial as going through the chain is to construct it in the most optimal way possible, because different chain structures produce different results, directly affecting the quality of predictions, performance and interpretability.

An early approach to this problem was using ensembles of randomly selected chain structures [12], and then combine prediction results. However, previous work has shown that this method is much more efficient to reduce the effect of bad chains than to value the outcome of good chains [5]. Being an NP-Hard problem, a brute force approach is far from being satisfactory, so many methods have been developed over the years in order to give an efficient order for the chain. Among them, we will present below the Evolutionary approach [13], and the simulated-annealing approach [5].

Genetic Algorithm

This method uses Genetic Algorithm in order to find the most optimal labels order. By using the fitness function from Equation 3 for any individual, where EM is the Exact Match, ACC is the Accuracy and HL is the Hamming Loss, it quickly converges to a label order.

$$Fitness(i) = \frac{EM + ACC + (1 - HL)}{3} \tag{1}$$

It is important to notice that this algorithm will start with k random individuals, and after calculating the fitness for everyone, a winner will be selected among them, which will be the one with highest Fitness value. Afterwards, this individual will suffer random mutations in order to create the next generation, and the same steps above are repeated. After N generations, we will have an order for the labels which is a local maximum in relation to its fitness function.

Simulated-Annealing

This algorithm is a well-known technique to find global optimum of a given function. It works really well for problems in which the search space is discrete, like the traveling salesman problem, and of course, chain structure.

This method is an adaptation of the Metropolis-Hasting algorithm, being a technique that involves an internal temperature of the system, first by heating it and allowing movement, and slowly cooling and stabilizing at the global maximum.

The method works on a k_{max} number of iterations, and the final configuration tends to be relatively close to the optimal one. For a given temperature, iteration index and the optimal configuration until then, there is a probability that the system will suffer a random permutation, and if a better configuration is found, that will become the new optimal configuration. To work, it uses a decreasing temperature function, which reduces the transition speed of the system.

4 Proposed Method

4.1 General Idea

In simple terms, our objective is to learn how to traverse a binary tree. What makes learning algorithms interesting in this problem is that some complex patterns can be present in the best path and the search space, in general, is too big to be completely explored. Learning how to traverse a search space is a frequent problem in reinforcement learning. Furthermore, the key ideas of this approach have shown to benefit from the use of MCTS (monte carlo tree search) [11].

This way, we propose to use a framework similar to the one explained in the AlphaGo Zero paper [1] and in the original AlphaGo [14] paper. We propose to use a single neural network to generate value and policy estimation. For policy improvement we propose to use MCTS.

4.2 Problem Definition

From now on we refer to our search tree Γ as a set of nodes $n \in \Gamma$ connected only to its parent p_n and exactly two children c_1^n and c_2^n or exactly no children, a node with no children is called leaf node or terminal node. The root well be defined as r and is the only node with no parent. A path P is a sequence $n_0 n_1 ... n_f$ such that for every pair $n_i \in P$ and $n_j \in P$ if n_j comes strictly after n_i, then n_i is the parent of n_j. To each non terminal node n we associate a number p_n that is the probability $c_1^n = \hat{P}(y_i|y_{i-1}, ..., y_0, \mathbf{x})$, in the same way we can associate a probability to c_2^n as $1 - p_n$. We finally define the value V of a path as the product of the probabilities along that path from parent node to child node (we could also define it as the sum, but we'll focus on the product for now).

This way our probably can be formally defined as to maximize the value $V(P) \forall P \in \Gamma$ such that the first node $n_0 = r$ and the last node n_t is a terminal node. Our model should learn to maximize $V(P)$ for every tree generated by the classifier chain method in a given dataset.

4.3 Detailed approach

The base model

Our tree traversal will be similar to the one use used by AlphaGo Zero [11], we take the current state s_t, where t is the index in the current traversed path from the root node (therefore the number of nodes traversed). Then, using this state as base, we calculate two values, $v^\theta(s_t)$ and $p^\theta(s_t)$. This values represent, respectively, the value and the policy calculated by two networks in the following way: $v^\theta(s_t) = f_{\theta_v}(s_t)$ and $p^\theta(s_t) = f_{\theta_p}(s_t)$. The layer $f_\theta(\cdot)$ is a fully connected layer with rectified linear unit activations. The values θ are the model parameters.

The value $v^\theta(s_t)$ represents a prediction for the network of the maximum value V that can be achieved from the current state s_t until a terminal node. The policy $p^\theta(s_t)$ represents an approximation of the probability that you should select the child 1 for the next step to maximize the final $V(P)$, the value of the final path after the traversal.

Finally we describe s_t, this is a vector that encodes all of the history of the traversed path. This means that it should encode information about the probabilities that were selected, the path itself (directions of the edges) and finally about the nodes. Based on [2] we decided to use a GRU-RNN to encode our states, this way we can also learn a state representation using a network architecture that's well suited for that instead of imposing one. Inputting the probabilities defined in Γ and the path's directions (encoded as -1 and +1) is already enough to define the traversal completely. This way the input of this network is a sequence S of tuples (p_i, d_i) that defines the probability of each node and the direction taken in each step.

Training the model

We should now define how we'll approach training. Until now we have the following architecture, shown in Figure 2.

When training and evaluating the model, still based on [11] we won't sample an action (which represents choosing the child node to go next) based on the predicted policy. We'll use MCTS to calculate a new improved policy that we'll call $\pi(s_t)$. Finally, to train the model we'll run simulated traversals from root to leaf on the trees generated by the classifier chain method in a given dataset and use the improved policy $\pi(s_t)$ to decide at each step which direction to go. In this simulation we'll record the values of the tuples $(s_t, \pi(s_t), z_t)$, respectively the value of the state, the improved

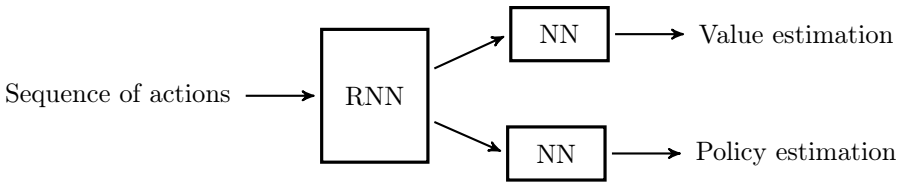


Figure 2: Resumed schema proposed model for policy and value estimation given a set of actions.

policy and the final value achieved after traverse a path completely. We have, now, all the necessary elements to calculate the loss which is described below in Equation 2.

$$l = \sum_t \left(\left(v^\theta(s_t) - z_t \right)^2 - \pi_t \log(p^\theta(s_t)) + c||\theta||^2 \right) \tag{2}$$

We then use this loss with gradient descent to train our model. To understand this loss, let's divide it in three parts. The first one is the $\left(v^\theta(s_t) - z_t\right)^2$ term, this represents that the value predictions should be updated to predict accurately the final value (the value $v^\theta(s_t)$ is obtained directly form the network placing the current state in the network). The second term, $\pi_t \log(p^\theta(s_t))$ represents that the predicted policy should converge to the distribution of the improved policy calculated by MCTS (in the same way, $p^\theta(s_t)$) can be obtained from the network. And finally the remaining term is a L2 weight regularization. Observe that we're recording the states and training the network based on the recorded states. We should also train the state RNN, to do that we proceed in the same manner but instead of the states we record directly the path (as sequence of tuples previously defined) and train the whole network as an ensemble. After the state network is trained we can freeze it and proceed recording only the states.

The monte carlo tree search

An important part of the described model is the improved policy, it's what is used to make actual path decisions and also to define the loss for the predicted policy. We proceed, again, by using the same idea presented in the AlphaGo Zero [1] paper. For each state s we define the state-action value function $Q(s, a)$, which defines the benefit from taking action a (equivalent to choose a child to go along the path) at state s (which defines the current position on the tree traversal). During the simulation to calculate the improved policy $\pi(s)$ of a given state, we expand a tree with s as root and choose child nodes that maximize the state-action value plus a bonus $u(s, a)$ defined in Equation 3, this way we're discouraged to explore nodes that already have been explored.

$$u(s, a) = cte \frac{P(s, a)}{1 + N(s, a)} \tag{3}$$

At each state we choose a_t as $argmax(Q(s_t, a) + u(s_t, a))$ for all of the possible actions at s_t. $P(s, a)$ defines the policy calculated by the network and can be directly derived from $p^\theta(s)$. We initialize the action values Q and the counts N as zero and as we traverse the tree, we increase the count for each pair state action that is traversed. We also backpropagate the values calculated by the value network so that $Q(s, a)$ gets updated with the average between all of the states that it reaches during the simulation.

Finally, once some paths were simulated, the new improved policy can be calculated using the counts for each action that has been taken from the current state s. The formal definition of the new policy is presented on Equation 4.

$$\pi(s) = \frac{N(s, \cdot)}{\sum_i N(s, i)} \tag{4}$$

A final important remark is that to use the MCTS method we assume that Q and N are functions of discrete values a and s. While this is true for the action (take the left or the right path in the binary tree) the RNN encoding for the states make the state space infinite. This problem can be solved considering the states to be just a sequence of actions, and, for each explored sequence, we store the necessary values. If a value from the neural network is needed, you can convert this sequence to an input to the RNN. The main need for the RNN is to encode information about the probabilities stored in the network and also to generate a vector of fixed length that encodes the state. Within the framework of MCTS, we can just associate the values to paths directly as long as we keep always within the same tree - which is the case.

4.4 Follow Up directions

Given the complexity of the model explained above, we'll try to split the implementation in as many independent units as possible. One idea would be to separate the state representation from the value and policy estimation, this way we could try approaches that are simpler than the RNN to encode the current state and we could also work separately. This would also make testing a lot easier. The improved policy can also developed separately.

One possibility would be to initially to develop and train the agent using directly the base policy calculated by the neural network. In parallel we could develop and test the improved policy, which is also independent from the base network (it only needs value and policy estimations). This way, we can divide our efforts between state estimation, policy and value estimation and policy improvement via MCTS.

We also plan to first work with datasets that contain less labels such as `flags` and `image` so that we can ensure that the model works well. Then, we will ensure that our implementation works by testing with datasets that contain more labels such as `bibtex` and `corel5k`.

In practice, we will create our own environment using `OpenAI Gym` and will use one of the popular machine learning libraries `Pytorch` or `Tensorflow`.

Appendices

A Preliminary Results

In order to understand the idea behind other inference methods and comprehend their advantages and disadvantages, as well as to have baseline results to compare with our proposed method, we have implemented each of them and ran experiments, following the approach presented in [10].

Table 1 contains the properties of the datasets we used in our examples. The classifier chains are in random order and the estimators are logistic regressions, which had their parameter C optimized between the values 10^{-3}, 10^{-2}, ..., 10^{3}, as was detailed in [10].

Table 1: Properties of the tested datasets. Only a sample of the mediamill dataset was used.

Dataset	Type	Train size	Test size	Attributes	Labels	Cardinality
bibtex	Image	4500	500	499	374	3.52
birds	Audio	322	323	260	19	1.01
corel5k	Image	4500	500	499	374	3.52
emotions	Music	391	202	72	6	1.87
enron	Text	1123	579	1001	53	3.38
flags	Image	129	65	19	7	3.39
genbase	Biology	463	199	1186	27	1.25
image	Image	1800	200	135	5	1.24
mediamill*	Video	5000	5000	120	101	4.29
medical	Text	333	645	1449	45	1.25
scene	Image	1211	1196	294	6	1.07
yeast	Biology	1500	917	103	14	4.24

Tables 2 and 3 contain the results of our experiments. When possible (ES, BS and EMC), the inference method uses the specific score that minimizes the exact-match or the hamming loss, according to the one being used.

As explained in [10], some results are inconsistent with what is theoretically expected. This happens because the estimators are not perfect, and probabilities predicted by them are not the true probabilities. We plan to attack this problem by trying to improve the estimators performances, either by optimizing the chain structure and/or changing the base estimators, and we also plan to try to use larger datasets. Solving this issue is important for us to correctly analyze the performance of our algorithm, since the quality of the estimators are directly related to the quality of the inference.

Table 2: Subset 0/1 loss and Hamming loss for tested datasets, in percentage values. The former is displayed on the first row and the latter in the second row, for each dataset. Exhaustive search results weren't calculated for datasets with more than 15 labels, because its complexity increases exponentially. The best result(s) in each row is in bold.

Datset	GS	ES	ϵ-A (0.25)	BS (2)	BS (3)	BS (10)	MC (10)	MC (100)	EMC (10)	EMC (100)
bibtex	**82.23**	-	82.35	82.39	82.43	82.43	85.73	82.90	83.26	82.43
	1.20	-	1.21	1.22	1.22	1.22	1.47	1.22	1.24	1.22
birds	**49.85**	-	**49.85**	**49.85**	**49.85**	**49.85**	63.16	52.94	57.28	50.15
	4.61	-	**4.61**	**4.61**	**4.61**	**4.61**	7.43	5.41	5.74	4.69
corel5k	**99.00**	-	**99.00**	**99.00**	**99.00**	**99.00**	99.60	99.60	99.60	99.20
	0.93	-	**0.93**	0.94	0.94	0.94	1.59	1.05	1.13	0.94
emotions	71.78	**68.32**	69.31	68.81	**68.32**	**68.32**	79.70	70.79	72.28	**68.32**
	21.70	21.78	**21.70**	22.69	22.03	21.78	27.72	22.36	**21.70**	21.78
enron	85.84	-	**83.59**	83.94	83.94	83.94	94.82	86.70	87.22	83.94
	4.43	-	4.45	4.51	4.56	4.57	6.42	4.78	4.95	4.57
flags	80.00	80.00	**78.46**	80.00	80.00	80.00	89.23	84.62	81.54	80.00
	25.49	25.27	**24.62**	25.27	25.27	25.27	30.11	28.35	26.15	25.27
genbase	**1.51**	-	**1.51**	**1.51**	**1.51**	**1.51**	13.57	4.52	**1.51**	**1.51**
	0.06	-	**0.06**	**0.06**	**0.06**	**0.06**	0.54	0.17	0.07	**0.06**
image	69.50	69.50	69.50	69.50	69.50	69.50	76.50	**69.00**	71.00	69.50
	19.10	19.40	19.40	19.40	19.40	19.40	23.60	19.20	19.60	19.40
mediamill	90.84	-	90.60	**90.44**	90.58	90.56	97.68	91.50	93.34	90.54
	3.32	-	**3.30**	3.37	3.50	3.74	4.87	3.47	3.73	3.61
medical	**37.05**	-	**37.05**	**37.05**	**37.05**	**37.05**	40.00	37.67	**37.05**	**37.05**
	1.12	-	1.13	1.13	1.13	1.13	1.25	1.14	1.13	1.13
scene	41.22	38.80	38.80	**38.63**	38.80	38.80	44.73	38.88	**38.63**	38.80
	10.35	9.88	10.05	**9.81**	9.87	9.88	11.19	9.88	9.84	9.88
yeast	79.83	76.88	79.28	**76.77**	77.21	76.88	93.02	80.15	81.35	76.99
	20.13	20.68	20.38	21.01	20.65	20.68	26.97	21.37	21.90	20.66

Table 3: Average number of explored nodes and average inference time in milliseconds for tested datasets. The former is displayed on the first row and the latter in the second row, for each dataset.

Datset	GS	ES	ϵ-A (0.25)	BS (2)	BS (3)	BS (10)	MC (10)	MC (100)	EMC (10)	EMC (100)
bibtex	159	-	182.6	317	474	1565	1590	15900	1590	15900
	0.61	-	1.68	1.71	2.60	8.66	6.82	65.31	6.40	64.13
birds	19	-	23.1	37	54	165	190	1900	190	1900
	0.01	-	0.12	0.05	0.07	0.20	0.40	2.40	0.21	2.09
corel5k	374	-	383.5	747	1119	3715	3740	37400	3740	37400
	0.49	-	1.94	3.62	5.48	18.22	7.81	68.61	6.50	67.19
emotions	6	63	9.2	11	15	35	60	600	60	600
	0.01	0.07	0.03	0.02	0.02	0.04	0.21	0.98	0.08	0.75
enron	53	-	75.5	105	156	505	530	5300	530	5300
	0.11	-	0.54	0.30	0.46	1.44	1.49	13.02	1.24	12.54
flags	7	127	12.2	13	18	45	70	700	70	700
	0.01	0.18	0.03	0.02	0.03	0.07	0.20	1.26	0.12	1.16
genbase	27	-	29.8	53	78	245	270	2700	270	2700
	0.06	-	0.25	0.16	0.24	0.71	1.01	7.93	0.73	7.51
image	5	31	7.7	9	12	25	50	500	50	500
	0.01	0.04	0.02	0.01	0.02	0.04	0.21	0.92	0.07	0.70
mediamill	101	-	117.9	201	300	985	1010	10100	1010	10100
	0.04	-	0.11	0.28	0.43	1.43	0.79	5.79	0.50	5.23
medical	45	-	49.1	89	132	425	450	4500	450	4500
	0.13	-	0.39	0.30	0.44	1.34	1.55	13.95	1.39	13.98
scene	6	63	7.7	11	15	35	60	600	60	600
	0.00	0.04	0.02	0.01	0.01	0.03	0.14	0.53	0.05	0.42
yeast	14	16383	19.4	27	39	115	140	1400	140	1400
	0.00	7.08	0.02	0.02	0.03	0.08	0.19	0.89	0.07	0.70

Bibliography

[1] David Silver et al. Mastering the game of go without human knowledge. 550(7676):354–359. Number: 7676 Publisher: Nature Publishing Group.

[2] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search, 2018.

[3] Grigorios Tsoumakas, Ioannis Katakis, and I. Vlahavas. Random k-labelsets for multi-label classification. *IEEE Trans. Knowl. Data Eng.*, 23:1079–1089, 07 2011.

[4] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5413–5423. Curran Associates, Inc., 2017.

[5] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains: A review and perspectives, 2020.

[6] Jesse Read, Pfahringer, Bernhard, Holmes, Geoffrey, and Eibe Frank. Classifier chains for multi-label classification. volume 85, pages 254–269, 08 2009.

[7] Krzysztof Dembczyński, Willem Waegeman, and Eyke Hüllermeier. An analysis of chaining in multi-label classification. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, page 294–299, NLD, 2012. IOS Press.

[8] Abhishek Kumar, Shankar Vembu, Aditya Krishna Menon, and Charles Elkan. Beam search algorithms for multilabel learning. *Machine Learning*, 92(1):65–89, Jul 2013.

[9] Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo optimization for multi-label classifier chains. *CoRR*, abs/1211.2190, 2012.

[10] Deiner Mena, Elena Montañés, José Ramón Quevedo, and Juan José del Coz. An overview of inference methods in probabilistic classifier chains for multilabel classification. *WIREs Data Mining and Knowledge Discovery*, 6(6):215–230, 2016.

[11] Tom Vodopivec, Spyridon Samothrakis, and Branko Ster. On monte carlo tree search and reinforcement learning. 60:881–936.

[12] J. Read, B. Pfahringer, Holmes, G., and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85 (3), 2011.

[13] E. C. Goncalves, A. Plastino, and A. A. Freitas. A genetic algorithm for optimizingthe label ordering in multi-label classifier chains. *IEEE 25th International Conference on Tools with Artificial Intelligence*, page 469–476, 2013.

[14] David Silver et al. Mastering the game of go with deep neural networks and tree search. 529:484–489.