RESEARCH ARTICLE OPEN ACCESS

Cyclic Classifier Chain for Cost-Sensitive Multi-Label Classification in Data Science for Real-World Applications

Kotadi chinnaiah[1], Akash saxena [2]

[1](Rescarch scholor, depaptment cse, sunrise university, alwar, rajasthan, india)

[2](Rescarch superviser, depaptment cse, sunrise university, alwar, rajasthan, india)

Abstract—

We propose a novel method, Cyclic Classifier Chain (CCC), for multilabel classification. CCC extends the classic Classifier Chain (CC) method by cyclically training multiple chains of labels. Three benefits immediately follow the cyclic design. First, CCC resolves the critical issue of label ordering in CC, and therefore reaches more stable performance. Second,CCC matches the task of cost-sensitive multilabel classifica-tion, an important problem for satisfying application needs. The cyclic aspect of CCC allows estimating all labels during training, and such estimates makes it possible to embed the cost information into weights of labels. Experimental results justify that cost-sensitive CCC can be superior to state-of-the-art cost-sensitive multilabel classification methods. Third, CCC can be easily coupled with gradient boosting to inherit the advantages of ensemble learning. In particular, gradient boosted CCC efficiently reaches promising performance for both linear and non-linear base learners. The three benefits, stability, cost-sensitivity and efficiency make CCC a competitive method for real-world applications.

I. Introduction

Multilabel classification is an important machine-learning problem that enjoys many real-world applications. It has many applications in text categorization [20], [25], image, video or sound tagging [4], [6], and bioinformatics [3], [11]. Multilabel classification allows an example to be associated with multiple classes simultaneously, which makes it very different from multiclass classification, an-other important problem. Multiclass classification allows only one class per example.

There are two types of methods for solving the mul-tilabel classification problem [21]: algorithm adaptation and problem transformation. Algorithm adaptation meth-ods modify or develop algorithms to specifically solve the problem, such as MLkNN [26] and BP-MLL [15]. Problem transformation methods transform the multilabel problem to other simpler problems we are familiar with, and most of them are binary classification or multiclass classification. The main benefit of problem transformation methods is to reuse the many mature and powerful tools to tackle multi-label classification problems, such as support vector machine, logistic regression, and decision tree. This paper proposes a problem transformation method, so we illustrate it more.

Arguably the simplest method in problem transforma-tion is called “Binary Relevance” (BR) [22]. BR simply transforms the multilabel classification problem to multiple binary classification problems, one for the existence of each class. The key benefit of BR is its efficiency. However, BR is often criticized for ignoring the relations that may exist among the labels. For example, if a table appears in a picture, it is highly possible that a chair is also in the same picture. BR cannot use such relations to improve performance. Therefore, it is usually the baseline method. Label powerset (LP) method is another simple idea [22]. It views each label set as a class, so the problem is transformed to be a 2K -classes multiclass classification, where K is the number of labels. However, 2K is usually very big, and many classes may not or seldom appear in the training data. Therefore, training using LP is extremely hard in practice. To solve the problems in BR and LP method, many methods are proposed, such as Random k-Labelset (RAkEL) [24], Classifier Chain [19], Conditional Principle Label Space Transformation (CPLST) [7], etc.

Classifier Chain (CC) is one of the most popular methods for multilabel classification because it is simple and can capture some relations among the labels. Many studies are trying to improve CC [8], [16], or use similar concepts to develop new methods [14]. Our proposed method also use

the similar concept. CC is a method that transforms the prob-lem into multiple binary classifications. We first determine the label order, and then train a classifier for each label one by one. Unlike in BR, when we train a classifier, CC can use the previously trained labels as features. Therefore, it can discover more relations of the labels. CC is usually as fast as BR if we train it serially, because they both train K classifiers. Though CC has more features for its classifiers, we can assume that the number of features is much more than the number of labels. Thus, the increasing number of features does not affect the time complexity much. If the assumption is not true, it is actually another problem known as extreme multilabel classification [18]. We will not discuss this problem in this study.

Though CC reaches promising results in many appli-cations, there is a problem: the label order affect the per-formance much. In addition, determining a good order is extremely hard. Ensemble Classifier Chain (ECC) [19] is one of the methods to reduce this problem. ECC builds multiple CCs with different label order and use the ensemble model of those CCs to predict. This paper proposes a novel method, Cyclic Classifier Chain (CCC), to solve the label ordering problem in another way. The root cause of the label order problem is that the classifiers near the tail of the chain get more information than the classifiers near the head get. Some of the labels may highly depend on some other labels, so it would be better if they are near the tail; some of the labels can be easily predicted using only the original features, so it would be better if they are near the head and help to predict the succeeding labels. Therefore, CCC tries to make every classifiers get all other labels during training. This is done by training many CCs repeatedly. Beginning at the second chain, we can get all other labels from the previous chains.

There are many evaluation criteria for multilabel classi-fication. Therefore, another popular research problem exists for dealing with the evaluation criteria. We want to take the evaluation criteria into account in the algorithms and improve the results on those criteria. We call this problem Cost-sensitive multilabel classification (CSMLC). The pre-vious state-of-the-art CSMLC methods are all extensions of CC. Probabilistic Classifier Chain (PCC) [8] is a cost-sensitive method based on CC. If a CC use the classifiers that can predict probability (e.g., logistic regression), we can use a special inference rule during predicting, and the inference rule can actually be Bayes optimal for any cost. However, a general inference rule requires O(2K +1T) time, where T is the predicting time of each classifier. Therefore, a general inference rule is not feasible for use in real-world applications. For this reason, we must produce a special inference algorithm for each cost to optimize it efficiently, but it is extremely difficult to do that. Therefore, only the inference algorithms for Hamming loss, rank loss, and F1 score are designed and can be used [8], [9], [10].

Our proposed method also has a cost-sensitive version, and we do not need to design a special algorithm for each cost. The user only need to define how to calculate the cost given a label set and the ground truth of an example. Another state-of-the-art method known as Condensed Filter Tree (CFT) [14] can also achieve this. This method is also a chain-like method because its predicting process is exactly the same as in CC. It solves the cost-sensitive problem by a bottom-up training process, which trains from the tail of the chain to the head. With the bottom-up process, CFT can embed the cost within the weight of each example during training and decrease the cost. Though our proposed method do not have a bottom-up training process, we also can get the label information from all other labels, so we can use the similar example weighting method and then minimize the given cost.

ECC, and some cost-sensitive chain-like methods such as PCC and CFT. In Section 3, we illustrate our proposed method, CCC, and then extend CCC to the cost-sensitive and gradient-boosted versions. In Section 4, we use some real-world dataset to verify the performance of the proposed method, and compare with its variations and some related methods.

II. CLASSIC CLASSIFIER CHAIN

A multilabel classification problem assumes that K la-bels exist, and that the label set L = {1, 2, ..., K } is a finite set that comprises those K labels. Each example has a label set $y \subseteq L$. For convenience of mathematical operations, y is usually converted to a binary vector $y \in \{0, 1\}^K$. In this study, we call this a label vector. When the i-th element ofy (i.e., y[i]) equals 1, it means $l_i \in y$. Otherwise, it means $l_i \notin y$.

The formal definition of the multilabel classifica-tion problem is that when given training data $D = \{(x_n, y_n)\}_{n=1}^{N}$ (where x_n denotes the numeric features of the n-th example, y_n is the label vector of the n-th example, and N is the number of examples), we want to predict the label vector y of a new example given its features x.

2.1. Classifier Chain

A Classifier Chain (CC) divides the multilabel classifi-cation problem into multiple binary classification problems. Each classifier in the chain predicts a

corresponding label. Thus, in this study we call it a single-label classifier. All chain-like methods consist of multiple single-label classi-fiers, and a CC has K single-label classifiers g1, g2, ..., gK , where gk can predict the label y[k].

To train the K single-label classifiers, we first must determine the label order o = (o1, o2, ..., oK), where 1 ≤

oi ≤ K for all i = 1, 2, ..., K and oi = oj for all i = j. After the label order is set, we can train the single-label classifiers in the order of go1 , go2 , ..., goK . For convenience, we assume that the labels have been sorted by the determined order such that o = (1, 2, ..., K). Unlike in binary relevance, while training gk, we can use the features (x, y^[1...k − 1]), where y^ is the predicted label vector. We refer to the y[1...k − 1] labels as preceding labels because they are at the preceding positions in the chain. The preceding label predictions can be used because they can be predicted by the preceding single-label classifiers during testing. These predictions are used as additional features. Therefore, we refer to these as label features in this study. With these label features, CC can model the relations of the labels.

Algorithm 1 shows the details of the training process of CC. Some studies use the ground truth y[k] as the label feature instead of y^[k] during training. However, our proposed method is logical only when we use y^[k] as the label feature. Therefore, we use this as the standard version. Algorithm 2 shows the details of the testing process.

Algorithm 1 Training Classifier Chain

g_k is the single-label classifier for label k
$D = \{(x_n, y_n)\}_{n=1}^{N}$ is the training data with N exam-ples
x_n is the features of the n-th example
$y_n \in \{0, 1\}^K$ is the ground truth label vector of the n-th example
$\hat{y}_n \in \{0, 1\}^K$ is the label predictions of the n-th example
for each label k from 1 to K do
    $D \leftarrow \{\}$
    for each $(x_n, y_n) \in D$ do
        $D \leftarrow D \cup ((x_n, \hat{y}_n[1...k-1]), y_n[k])$
    end for
    train the classifier g_k using training data D
    for each $(x_n, y_n) \in D$ do
        $\hat{y}_n[k] \leftarrow$ use g_k and feature $(x_n, \hat{y}_n[1...k-1])$ to predict
    end for
end for
return $g_1, g_2, ..., g_K$

Algorithm 2 Testing Classifier Chain

g_k is the trained single-label classifier for label k
$D = \{x_n\}_{n=1}^{N}$ is the testing data with N examples
x_n is the features of the n-th example
$\hat{y}_n \in \{0, 1\}^K$ is the label predictions of the n-th example
for each label k from 1 to K do
    for each $x_n \in D$ do
        $\hat{y}_n[k] \leftarrow$ use g_k and feature $(x_n, \hat{y}_n[1...k-1])$ to predict
    end for
end for
return $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$

2.2. Ensemble Classifier Chain

Choosing the optimal label order for the CC is not easy, while it affects performance considerably. A method known as the Ensemble Classifier Chain (ECC) is proposed to solve this problem in the original study on CC [19]. The idea is to train many Classifier Chains independently using different label orders and different sampled training data. Both the randomness of label orders and training data can provide considerable diversity. Therefore, the ensemble framework works well. This tells us that the label order has considerable effect.

2.3. Probabilistic Classifier Chain

In real-world applications, different types of costs are needed to evaluate performance more effectively. One special type of cost, known as example-based cost, is often used. Example-based cost is a cost that can be calculated for each example. We only need to define a cost function $L(y, \hat{y})$, where y is the ground truth label vector and $\hat{y}$ is the label prediction. Given the ground truth and the prediction of an example, we can calculate its cost $L(y, \hat{y})$. Therefore,
the expected cost for all examples some commonly used costs are Hamming loss $Hamming(y, \hat{y}) = \frac{1}{K} \sum_{k=1}^{K} y[k] = \hat{y}[k]$, and the negative F1 score $F1(y, \hat{y}) = -\frac{2\, y \cap \hat{y}}{y_1 + \hat{y}_1}$. The goal of the cost-sensitive multilabel classification (CSMLC) problem for the example-based cost is to minimize the expected cost. Other types of metrics such as micro-F1 and macro-F1 scores are difficult to optimize. The example-based cost is effective in most cases. Therefore, in this study, we only discuss example-based costs.
The Probability Classifier Chain (PCC) tries to solve the cost-sensitive multilabel classification problem using the Bayes optimal decisions [8]:

$$g(\mathbf{x}) = \underset{\hat{\mathbf{y}}}{\operatorname{argmin}}\ \mathbb{E}_{\mathbf{y}|\mathbf{x}} L(\mathbf{y}, \hat{\mathbf{y}}). \quad (1)$$

To calculate the expectation, the following probability must be obtained:

P (y|x) = P (y[1]|x)P (y[2]|y[1], x)...P (y[K]|y[1...K−1], x).

The conditional probability P (y[k]|y[1...k − 1], x) can be estimated by means of the k-th single-label classifier in a normal CC. Note that the single-label classifier (e.g., logistic regression) must be able to estimate the probability. Therefore, we only need to train a normal CC, and then use the Bayes optimal decisions in (1) to infer the labels. However, a general inference rule requires $O(2^{K+1}T)$ time to enumerate all possible y and its probability P (y|x), where T is the predicting time of each single-label classifier. Therefore, a general inference rule is not feasible for use in real-world applications. For this reason, we require a special inference algorithm for each cost to calculate (1) efficiently. However, producing such an algorithm is extremely difficult. Therefore, only the inference algorithm for Hamming loss, rank loss, and F1 score is designed and can be used [8], [9], [10].

2.4. Condensed Filter Tree

Condensed Filter Tree (CFT) [14] is a general CSMLC method and can also minimize the example-based costs. Unlike in PCC, it does not require a special inference rule. Its predicting process is the same as in CC. Therefore, it is also a chain-like method, and is efficient in predicting regardless of the cost.

Without a special inference rule, the CFT requires a special training process to minimize the cost. CFT trains its single-label classifier from the tail of the chain to the head. This bottom-up process repeats M times. Therefore, we train M chains during the training process. Every time a chain is trained, we use it to predict the labels and then save it. Thus, when training a single-label classifier, we know the preceding label predictions by the ground truth and the label predictions from the previous chains. Because we have m possible preceding labels when we train the m-th chain, m examples will exist for each example in the original training data, and they may have different label features and example weights. If M is large enough, we can cover a sufficient number of patterns of the preceding labels.

The benefit of the bottom-up process is that we not only roughly know the preceding labels, but we also have the succeeding classifiers. When training the k-th single-label classifier in a chain, we can first assume that the k-th label prediction ŷ[k] = 0, and then use this label and the succeeding classifiers to predict the succeeding labels. With these succeeding labels, the cost c_0 for ŷ[k] = 0 can be calculated for each example. Similarly, we can assume that ŷ[k] = 1, and then obtain the cost c_1 for ŷ[k] = 1. With these two costs, we know that the ground truth for this example should be argmin c_i, and the example weight i

should be $|c_0 - c_1|$. The example weight should be $|c_0 - c_1|$ because if we wrongly predict this example, it will roughly generate $|c_0 - c_1|$ cost. This is known as a regret. CFT can minimize the cost because it can estimate the regret when a label is wrongly predicted. This enables us to train a single-label classifier to minimize the regret for each label. In other words, we can minimize the cost if we know all other labels while training a single-label classifier.

However, this training process is extremely space- and time-consuming. First, the training data grows linearly when we have more chains because we must use the preceding label predictions from the previous chains. Second, every time we train a single-label classifier, we must predict the succeeding labels twice for ŷ[k] = 0 and ŷ[k] = 1. In Section 3.2, we discuss the time complexity and compare it to our method.

3. Proposed Method

From CC and ECC [19], we learned that the label order is critical, and ECC works well because the random label order provides sufficient diversity. From CFT [14], we learned that if we can know all other labels while training a single-label classifier, we can embed the cost within the weight of each example in order to decrease the cost. Inspired by these studies, we propose a novel method that we call Cyclic Classifier Chain (CCC). CCC avoids the aforementioned problems while retaining many good properties from those studies. Section 3.1 illustrates the basic concept behind CCC that solves the label order problem. Section 3.2 describes an approach to make CCC cost-sensitive that embeds the cost as weight when other labels are known. In Section 3.3 and 3.4, we further improve the cost-sensitive version by applying the concept of gradient boosting.

3.1. Cyclic Classifier Chain

The label order is critical in CC because if the position of a single-label classifier is near the head, it will possess fewer label features than those of the classifier near the tail. Determining the label order is difficult because there are $K!$ possible orders. In addition, even if we can choose the best order, the classifier near the head clearly loses some information. A simple solution is to train an additional CC after we train the original CC. A single-label classifiers in the additional CC do not need to obtain all its label features from the preceding classifiers. Instead, it can use the label predictions from the previous CC. For example, while the additional CC trains the single-label classifier for label k, it uses the label predictions ŷ[k + 1...K] from the original CC, and ŷ[1...k − 1] from the additional CC as features. By contrast, the classic CC uses only its preceding label predictions ŷ[1...k − 1] as

features. Therefore, this method enables us to use the succeeding label predictions $\hat{y}[k+1...K]$ while training a single-label classifier.

Using this additional CC, we can lower the effect of the label order because all single-label classifiers have the same number of features. However, the effect of the label order remains because the label predictions from the first CC are affected, and the label features in the additional CC are also affected. Therefore, we add more CCs and also let them get their succeeding label predictions from their previous CC. The entire training process is similar to connecting the head and tail of a CC to form a cycle and cyclically training the single-label classifiers. Thus, we call this method a Cyclic Classifier Chain. After we train many cycles, the label predictions will converge and be sufficiently accurate to generate stable and improved predictions regardless of the label order.

Algorithm 3 Training Cyclic Classifier Chain

1: $g_{c,k}$ is the single-label classifier for label k in the c-th cycle
2: $D = \{(x_n, y_n)\}_{n=1}^{N}$ is the training data with N exam-ples
3: x_n is the features of the n-th example
4: $y_n \in \{0, 1\}^K$ is the ground truth label vector of the n-th example
5: $\hat{y}_n \in \{0, 1\}^K$ is the label predictions of the n-th example
6: run Algorithm 1 and Algorithm 2 to get the initial classifiers $g_{1,1}, g_{1,2}, ..., g_{1,K}$ and the initial predictions $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$
7: for each cycle c from 2 to C do
8: for each label k from 1 to K do
9: $D \leftarrow \{\}$
10: for each $(x_n, y_n) \in D$ do
11: $D \leftarrow D \cup ((x_n, \hat{y}_n[1...k-1], \hat{y}_n[k+1...K]), y_n[k])$
12: end for
13: train $g_{c,k}$ using training data D
14: for each $(x_n, y_n) \in D$ do
15: $\hat{y}_n[k] \leftarrow$ use $g_{c,k}$ and feature $(x_n, \hat{y}_n[1...k-1], \hat{y}_n[k+1...K])$ to predict
16: end for
17: end for
18: end for
19: return $g_{c,k}$ for $c = 1, 2, ..., C$ and $k = 1, 2, ..., K$

Algorithm 4 Testing Cyclic Classifier Chain

1: $g_{c,k}$ is the single-label classifier for label k in the c-th cycle
2: $D = \{x_n\}_{n=1}^{N}$ is the training data with N examples
3: x_n is the features of the n-th example
4: $\hat{y}_n \in \{0, 1\}^K$ is the previous label prediction of the n-th example
5: run Algorithm 2 with the classifiers $g_{1,1}, g_{1,2}, ..., g_{1,K}$ and get the initial predictions $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$
6: for each cycle c from 2 to C do
7: for each label k from 1 to K do
8: for each $x_n \in D$ do
9: $\hat{y}_n[k] \leftarrow$ use $g_{c,k}$ and feature $(x_n, \hat{y}_n[1...k-1], \hat{y}_n[k+1...K])$ to predict
10: end for
11: end for
12: end for
13: return $\hat{y}_1, \hat{y}_2, ..., \hat{y}_N$

Algorithm 3 shows the details of the training process. This algorithm involves two stages. The first stage involves training the initial classifiers $g_{1,k}$, which is the same as that for a classic CC. The second stage involves training $C-1$ cycles of CCs. Algorithm 4 shows the details of the testing process. It can be derived by simply removing the training part of the single-label classifiers.

3.2. Cost-Sensitive Cyclic Classifier Chain

After the first cycle of the CC, the single-label classifiers can use all other label predictions as features. Furthermore, this additional information not only can be used as features, but also can provide some information about the cost. In Section 2.4, we describe the manner in which CFT can optimize any given example-based cost. The main property that makes it easily achieve this also appears in CCC. While training a single-label classifier in a CFT, we can calculate the cost of each class for every example, and simply use the difference of the cost as the example weight. We can thus minimize the provided cost. Similarly, this can also be accomplished in CCC because we also know all other labels while training a single-label classifier. We call this variation of CCC the Cost-Sensitive Cyclic Classifier Chain (C^4).

While training the single-label classifier for label k, we use the preceding label predictions $\hat{y}[1...k-1]$ from the current cycle, as well as the succeeding label predictions $\hat{y}[k+1...K]$ from the last cycle. Therefore, we can calculate the cost of predicting label k as 0:

$c_0 = L(y_n, (\hat{y}_n[1...k-1], 0, \hat{y}_n[k+1...K]))$, and the cost of predicting label k as 1:

$c_1 = L(y_n, (\hat{y}_n[1...k-1], 1, \hat{y}_n[k+1...K]))$,

where L is the cost function. As in CFT, we then use the regret $|c_0 - c_1|$ as the example weight to minimize the expected cost.

Algorithm 5 Training Cost-Sensitive Cyclic Classifier Chain

1: assign each example a weight by replacing the line 11 in Algorithm 3 with the following lines:
2: $c_0 = L(y_n, (\hat{y}_n[1...k-1], 0, \hat{y}_n[k+1...K]))$
3: $c_1 = L(y_n, (\hat{y}_n[1...k-1], 1, \hat{y}_n[k+1...K]))$
4: $w \leftarrow |c_0 - c_1|$
5: $y \leftarrow \operatorname{argmin}_i c_i$
6: $D \leftarrow D \cup ((x_n, \hat{y}_n[1...k-1], \hat{y}_n[k+1...K]), y, w)$

Algorithm 5 shows the details of the training process. We only need to calculate the example weight according to the costs for each example, and then use this weight to train the single-label classifiers in Algorithm 3. The testing process is the same as the cost-insensitive version in Algorithm 4.

C^4 operates in a reversed way comparing to CFT though they are very similar in the concept of example weighting. While C^4 knows exactly the prediction of the preceding labels, CFT only roughly knows it. By contrast, the former only roughly knows the prediction of the succeeding labels, whereas the latter knows it exactly. We divided the training process into "rough prediction" and "exact prediction" to determine the differences between them.

For the rough prediction, CFT hides all the label predictions of the previous trees in the training data. Therefore, it knows many possible predictions of the preceding labels. However, this causes the training data to grow considerably at the end of every round. If we train M rounds of CFT, the number of training examples will grow to MN in the last round, where N is the original number of training examples. Therefore, the time complexity for training a CFT is

$$O\Big(T_{tree}(N) + T_{tree}(2N) + \ldots + T_{tree}(MN)\Big), \quad (2)$$

where $T_{tree}(\cdot)$ is the training time for a round of CFT if we consider only the number of examples as variable. This

complexity is actually extremely large (which we discuss later). Therefore, M can only be extremely small (e.g., M = 8 in the experiments in [14]). CFT costs much to know the preceding label predictions, while C^4 simply knows its succeeding label predictions from the previous round but do not know any information about the succeeding classifiers in this round.

For the exact prediction, C^4 knows the exact predictions of the preceding labels because this knowledge is inherent in CC. However, CFT requires considerable time for this because it must predict the succeeding labels every time it calculates the example weights. If we assume that the number of features is much greater than the number of labels, the time complexity for training a round of CFT is given by:

$$\begin{aligned} &O\Big(K \cdot T_{train}(N') + \\ &(T_{predict}(N') + 2 \cdot T_{predict}(N') + \ldots + K \cdot T_{predict}(N'))\Big) \\ &= O\Big(K \cdot T_{train}(N') + K^2 \cdot T_{predict}(N')\Big), \end{aligned} \quad (3)$$

where K is the number of labels, N is the number ofexamples in this round, and $T_{train}(\cdot)$ and $T_{predict}(\cdot)$ are the training and predicting time of a single-label classifier respectively. The K^2 in the complexity means that training a CFT for many labels is extremely difficult.

We next combine the effect of these two time-consuming processes. Assume that we use a single-label classifier with $T_{train}(N) = O(N^2)$ and $T_{predict}(N) = O(N)$. The time complexity for training a round (3) then becomes $O(KN^2 + K^2N)$, and the time complexity for training a CFT (2) becomes $O(KM^3N^2 + K^2M^2N)$, while only

$$O\Big(KC(T_{train}(N) + T_{predict}(N))\Big) = O(KCN^2)$$

is used for a C^4 with C cycles. If $T_{train}(\cdot)$ and $T_{predict}(\cdot)$ are greater than we assumed previously, the difference between the two complexity will be greater because M also affects $T_{train}(N)$ and $T_{predict}(N)$. Therefore, C^4 is more scalable than CFT in training.

3.3. Gradient Boosted C^4

For some cost or score metrics such as F1 score and accuracy, if most labels are predicted incorrectly, the example weights will be extremely sparse. In other words, most examples have the same cost regardless of whether the single-label classifier predicts the label as being 0 or 1. Therefore, only some of the examples have non-zero weights. This means a single-label classifier cannot learn much when the predictions of other labels are not sufficiently accurate. In addition, the results will not be stable because the classifier only use a few examples to train in each cycle. Therefore, we propose a stable method based on gradient boosting to enable a single-label classifier to cooperate with the classifiers for the same label in previous cycles.

is a model that iteratively trains multiple base learners (i.e., single-label classifier). Therefore, using the boosting technique is suitable. Gradient Tree Boosting is a popular model in machine learning competitions and re-cently has won several first place awards [1], [2]. Therefore, we use the same idea proposed by Friedman [12], [13]. The basic idea of a general Gradient Boosting Machine is to train C base learners iteratively, and assign weight to each of them. The model then obtains a real value prediction given the feature x:

$$F_C(\mathbf{x}) = const + \sum_{c=1}^{C} \gamma_c g_c(\mathbf{x}),$$

where g_i is the i-th base learner, and γ_i is the weight for the i-th base learner. In our case, for the C^4 with C cycles, the k-th label prediction of the n-th example becomes

$$F_{C,k}(\mathbf{x}_n) = const_k + \sum_{c=1}^{C} \gamma_{c,k} g_{c,k}(\mathbf{x}_{n,c,k}).$$

Note that x is changed to $x_{n,i,k}$ because we use different label features for different labels and cycles. Because we want to conduct binary classification for each label, we use the following logistic function to transform the real value prediction into a probability:

$$P_{C,k}(y = 1|\mathbf{x}_n) = \frac{e^{F_{C,k}(\mathbf{x}_n)}}{1 + e^{F_{C,k}(\mathbf{x}_n)}},$$

and we use the negative log-likelihood as the loss function:

$$\begin{aligned} L(\mathbf{y}_n[k], F_{C,k}(\mathbf{x}_n)) &= -\log P_{C,k}(y = \mathbf{y}_n[k]|\mathbf{x}_n) \\ &= -\log \frac{e^{\mathbf{y}_n[k] F_{C,k}(\mathbf{x}_n)}}{1 + e^{F_{C,k}(\mathbf{x}_n)}}. \end{aligned}$$

Because we want to solve a cost-sensitive problem here, we must calculate the weighted sum of the loss using:

$$\sum_{n=1}^{N} w_{n,C,k} L(\mathbf{y}_n[k], F_{C,k}(\mathbf{x}_n)) = -\sum_{n=1}^{N} w_{n,C,k} \log \frac{e^{\mathbf{y}_n[k] F_{C,k}(\mathbf{x}_n)}}{1 + e^{F_{C,k}(\mathbf{x}_n)}},$$

where $w_{n,C,k}$ is the example weight of the n-th example in the C-th cycle for the k-th label. We can then formulate the optimization problem. When we add a new base learner $g_{k,c}$ for the k-th label in the c-th cycle, the new prediction becomes:

$$F_{c,k}(\mathbf{x}) = F_{c-1,k}(\mathbf{x}) + \left(\operatorname{argmin}_{g \in \mathcal{H}} \sum_{n=1}^{N} w_{n,c,k} L(\mathbf{y}_n[k], F_{c-1,k}(\mathbf{x}_n) + g_{c,k}(\mathbf{x}_{n,c,k}))\right)(\mathbf{x}).$$

A greedy approach [12] to solve the argmin is to use the steepest descent. We first train a base learner to predict the gradient:

$$g_{c,k} : \mathbf{x} \rightarrow \nabla_F L(\mathbf{y}[k], F_{c-1,k}(\mathbf{x})), \qquad (4)$$

where

$\nabla_F L(\mathrm{y}[k], F_{c-1,k}(\mathrm{x})) = \mathrm{y}[k] - P_{c-1,k}(y = 1|\mathrm{x})$, and then find a γ that minimizes the loss of $F_{c,k}$:

$$\gamma_{c,k} = \frac{\sum_{n=1}^{N} w_{n,c,k}(\mathbf{y}_n[k] - p_n) g_{c,k}(\mathbf{x}_{n,c,k})}{\sum_{n=1}^{N} w_{n,c,k} p_n (1 - p_n) g_{c,k}(\mathbf{x}_{n,c,k})^2}, \qquad (5)$$

where $p_n = P_{c,k}(y = 1|\mathrm{x}_{n,c,k})$. We can iteratively train the (4) and $\gamma_{c,k}$ (5) from $c = 1$ to $c = C$. We should also iterate over the labels in each cycle in the same manner as described in Section 3.2. A basic Gradient Boosted C^4 is then built. A simple regularization strategy is proposed [12] to scale the contribution of each base learner by a learning rate ν such that a prediction of the c-th cycle becomes:

$$F_{c,k}(\mathbf{x}_n) = const_k + \sum_{i=1}^{c} \nu \gamma_{i,k} g_{i,k}(\mathbf{x}_{n,i,k}).$$

Finally, the constant $const_k$ can be learned by simply calculating the log odds ratio

$$\log \frac{\sum_{n=1}^{N} \mathbf{y}_n[k]}{\sum_{n=1}^{N} (1 - \mathbf{y}_n[k])}.$$

3.4. Gradient Tree Boosted C^4

If Regression Tree [5] is used as the base learner, a special modification can be used [12]. In Regression Tree, each leaf has a predicted value. Because the number of leaves is finite, we can actually train multiple γ for each leaf to fit the loss. To calculate the γ for each leaf, we simply use (5), and modify the summation to be over examples in a leaf. This technique can speed up the training process.

Conclusions

This paper proposes a novel method, Cyclic Classifier Chain, for multilabel classification based on the concept of Classifier Chain. It tries to solve the label ordering problem in Classifier Chain, and is extended to deal with cost-sensitive multilabel classification. Similar to Condensed Filter Tree, it can also optimize any given example-based cost. Its performance is better than Probabilistic Classifier Chain, and is competitive with Condensed Filter Tree, while we have shown that its training time complexity is much smaller than Condensed Filter Tree. To improve the stability of the prediction, we further propose Gradient Boosted Cyclic Classifier Chain, and it slightly improves Cyclic Classifier Chain. It is extremely easy to add the gradient boosting concept to our method, so the training process is also efficient. Because Gradient Boosted Cyclic Classifier Chain can be trained efficiently, we further replace the base learner with regression tree, and make it be similar to the popular Gradient Tree Boosting method. The regression tree is non-linear, so the Gradient Tree Boosted Cyclic Classifier Chain becomes a non-linear method, and can outperform very much versus other linear methods on some datasets. This means our method can also be easily extended to a non-linear version to significantly improve the performance, and can be trained efficiently.

The difference between Cyclic Classifier Chain and Condensed Filter Tree is that they use different methods to estimate all other labels during training a single-label classifier. The success of Cyclic Classifier Chain means that we can try any methods to estimate other labels, and then use them to optimize any example-based costs. Therefore, this study also gives us a direction of future research on the general cost-sensitive multilabel classification.

References

[1] Kdd cup 2015 winner report. http://kddcup2015.com/information-winners.html. Accessed: 2017-06-04.

[2] Loan default prediction winner report. https://www.kaggle.com/c/loan-default-prediction/details/winners. Accessed: 2017-06-04.

[3] Z. Barutc¸uoglu, R. E. Schapire, and O. G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, pages 830– 836, 2006.

[4] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, pages 1757–1771, 2004.

[5] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[6] F. Briggs, Y. Huang, R. Raich, K. Eftaxias, Z. Lei, W. Cukierski, S. F. Hadley, A. Hadley, M. Betts, X. Z. Fern, J. Irvine, L. Neal, A. Thomas, G. Fodor, G. Tsoumakas, H. W. Ng, T. N. T. Nguyen, H. Huttunen, P. Ruusuvuori, T. Manninen, A. Diment, T. Virtanen, J. Marzat, J. Defretin, D. Callender, C. Hurlburt, K. Larrey, and M. Milakov. The 9th annual MLSP competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–8, 2013.

[7] Y.-N. Chen and H.-T. Lin. Feature-aware label space dimension reduction for multi-label classification. In *NIPS*, pages 1538–1546, 2012.

[8] K. Dembczynski, W. Cheng, and E. Hullermeier¨. Bayes optimal multilabel classification via probabilistic classifier chains. In *ICML*, pages 279–286, 2010.

[9] K. Dembczynski, W. Kotlowski, and E. Hullermeier¨. Consistent multilabel ranking through univariate losses. In *ICML*, 2012.

[10] K. Dembczynski, W. Waegeman, W. Cheng, and E. Hullermeier¨. An Exact Algorithm for F-Measure Maximization. In *NIPS*, pages 1404– 1412. 2011.

[11] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *NIPS*, pages 681–687, 2001.

[12] J. H. Friedman. Greedy function approximation: A gradient boosting machine., Oct. 2001.

[13] J. H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, pages 367–378, 2002.

[14] C.-L. Li and H.-T. Lin. Condensed filter tree for cost-sensitive multi-label classification. In *ICML*, pages 423–431, 2014.

[15] M. ling Zhang, Z. hua Zhou, and S. Member. Multi-label neural networks with applications to functional genomics and text categorization. In *IEEE Transactions on Knowledge and Data Engineering*, 2008.

[16] W. Liu and I. Tsang. On the optimality of classifier chain for multi-label classification. In *NIPS*, pages 712–720, 2015.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, pages 2825–2830, 2011.

[18] Y. Prabhu and M. Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *KDD*, 2014.

[19] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. In *ECML*, pages 254–269, 2009.

[20] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, pages 135–168, 2000.

[21] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *IJDWM*, pages 1–13, 2007.

[22] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining Multi-label Data. In *Data Mining and Knowledge Discovery Handbook*, chapter 34, pages 667–685. 2010.

[23] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: a java library for multi-label learning. *JMLR*, 2011.

[24] G. Tsoumakas and I. P. Vlahavas. Random *k* -labelsets: An ensemble method for multilabel classification. In *ECML*, pages 406–417, 2007.

[25] N. Ueda and K. Saito. Parametric mixture models for multi-labeled text. In *NIPS*, pages 721–728, 2002.

[26] M. Zhang and Z. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, pages 2038–2048, 2007.