ECAI 2020 | Multi-Label Learning with Deep Forest

Liang Yang and Xi-Zhu Wu and Yuan Jiang and Zhi-Hua Zhou

摘要

在许多现实问题中，数据是同时与多个标签相关联。多标签学习就是解决这样的问题，它在文本分类、场景分类、功能基因组学、视频分类、化学物质分类等问题中得到了广泛的应用。在大数据背景下，深度神经网络虽然可以被用来解决多标签问题，但是它通常需要大量的训练数据，而且它的成本也比较高。论文提出的 Multi-Label Deep Forest 模型 (MLDF) 使用不同的多标签树方法作为深度森林中的构建块，可以通过逐层表示学习来利用标签相关性。本工作 1) 复现了 MLDF 的基本结构；2) 复现了 MLDF 的两种优化机制；3) 对 MLDF 的分类性能在 9 个数据集上开展了充分的实验。结果表明 MLDF 的分类性能比传统的多标签方法更优越，另外 MLDF 在应用各种树模型时也具有很高的灵活性。

关键词： 多标签学习；多标签深度森林；深度森林

1 引言

在机器学习领域，处理多标签分类问题一直是一个重要而挑战性的课题。而深度森林是一种新兴的机器学习方法，它是一种基于树模型的深度学习框架，具有在各种任务上表现优异的潜力。多标签分类在现实问题中非常普遍，例如图像识别、文本分类、医学诊断等领域都需要处理多个标签的情况。深度学习方法在处理大规模数据和复杂任务方面表现出色，但它对于少样本多标签问题的表现可能不尽如人意。深度森林在保留传统决策树模型优势的同时，尝试结合深度学习框架，为多标签分类问题提供新的解决思路。通过结合深度森林和多标签学习，可能提高多标签分类任务的性能，并且为该领域带来新的创新和解决方案。

2 相关工作

2.1 传统的多标签学习方法

在多标签学习方法中，二值相关 [4]（Binary Relevance）是通过将多标签问题转化为每个标签的独立二分类问题的直接方法。虽然它旨在充分利用高性能的传统单标签分类器，但是当标签空间很大时，它的计算成本是特别高昂的。而且 BR 算法忽略了一个标签与标签之间的关系，这限制了它的预测性能。研究表明标签之间的相关性是提高多标签学习性能的关键。因此，越来越多的基于标签之间相关性的多标签学习方法被学者们提出 [5]。在多标签学习方法 [5] [6] [8] 中，数据样本标签之间的相关性有着十分重要的作用。

2.2 基于深度神经网络的多标签学习方法

深度神经网络模型与传统的多标签方法不同，它通常致力于学习一个新的特征空间，并在输出层使用多标签分类器。在较早使用网络结构的方法中，BP-MLL 方法 [7] 不仅将每个输出节点视为二分类任务，而且利用了依赖于网络结构本身的标签相关性。后来，提出了一种建立在 BP-MLL 基础上相对简单的神经网络方法，将成对排名损失替换为熵损失 [3]，它在大规模文本分类中取得了良好的效果。然而，深度神经模型通常需要大量的训练数据，因此它们通常不适合小规模数据集。

2.3 深度森林

通过认识到深度学习的本质在于逐层处理、模型内特征转换和足够的模型复杂度，Zhou 和 Feng 提出了深度森林 [10]。深度森林是建立在决策树上的深度集成模型，在训练过程中不使用反向传播。深度森林具有级联结构，它可以像深度神经模型一样进行表示学习。与深度神经网络（DNN）相比，深度森林更容易训练，因为它具有较少的超参数。它在广泛的任务中取得了优异的性能，例如大规模金融欺诈检测 [9]；并且已经发现森林模型可以实现一些被认为只有神经网络才拥有的重要属性，例如自动编码器能力 [2] 和分层分布式表示能力 [1]。虽然深度森林在分类任务中被发现是有用的 [10]，但在本论文的工作之前，还没有研究将其应用于多标签学习。

3 本文方法

3.1 本文方法概述

图 2 展示了多标签深度森林 MLDF 的框架。在 MLDF 的每一层中，不同的多标签森林都集成在一起。从第 t 层 $layer_t$ 中，我们可以得到第 t 层的特征表示 H^t（黑色）。第 t 层的度量感知特征复用机制部分将接收第 t 层特征表示 H^t，并在不同性能度量的指导下，通过复用在第 t-1 层 $layer_{t-1}$ 中学习到的 t-1 层的特征表示 G^{t-1} 来更新它，更新后的特征表示是 G^t。然后，新的特征表示 G^t（绿色）将与原始输入的特征（红色）连接起来，然后进入下一层 t+1 层。

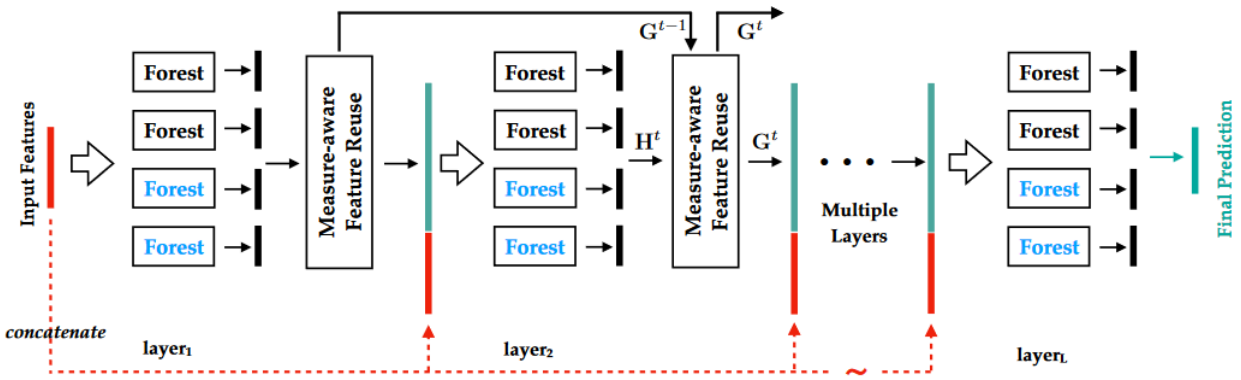

图 1. 多标签深度森林 MLDF 框架

在 MLDF 中，每个层都是一个森林集合。为了提高集成方法的性能，我们考虑了不同的树生长方法来丰富多样性，这是集成方法的成功的关键。在传统的多类问题中，gcForest 使用了极端随机树，它使用了全部特征用于拆分节点。对于多标签学习问题，我们也可以通过改变生成树时拆分节点的方式来丰富多样性。在 MLDF 中，我们使用 RF-PCT 作为森林块，在森林的树中生成节点方式中有两种不同的方法：一种考虑每个特征的所有可能的分割点，即 RF-PCT（黑色的），另一种随机考虑一个分割点，我们将其命名为 ERF-PCT（蓝色的）。当然，其他的多标签树方法也可以嵌入到每一层中，比如 RFML-C4.5。

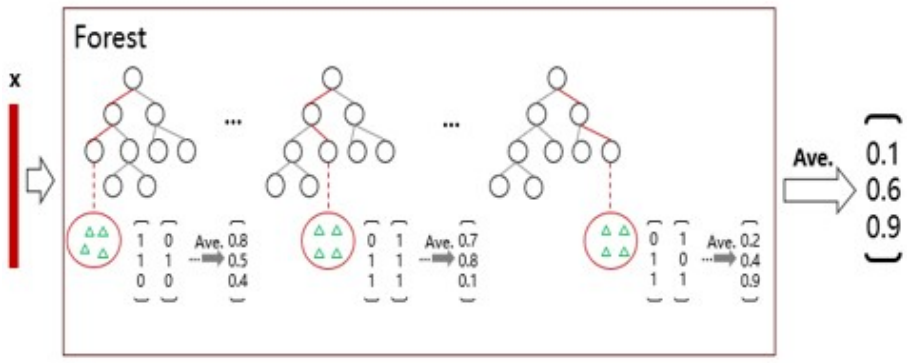

图 2. 多标签森林

假设所有基本森林都已拟合，预测过程可以总结如下。首先我们将实例预处理为标准矩阵 X，紧接着让实例矩阵 X 通过第一层。给定一个实例，森林将产生标签分布的估计值，即实例拥有每个标签的概率。因此，我们可以得到第一层的特征表示 H_1。通过采用度量感知特征复用机制，我们可以得到第一层新的特征表示 G_1。然后，我们将新的特征表示 G_1 与原始输入特征 X 连接起来，并让它们进入下一层。下一层将考虑丰富标签信息的特征表示 G_1，以方便 MLDF 更好地利用标签相关性。经过多层之后，我们得到了最终的预测。

3.2 度量感知特征复用机制

每个级联层的森林 PCT 的拆分准则与性能度量没有直接关系，在 $layer_t$ 的训练期间，表示张量 H_t 的形状为（num 森林，num 样本，num 标签），它与我们想要优化的性能度量无关。论文提出的度量感知特征复用机制是为了增强在不同性能度量指导下的特征表示。度量感知特征复用的关键思想是，如果当前层上的置信度低于训练中确定的阈值，则在当前层上部分复用前一层中更好的特征表示。因此，问题的关键在于如何按需定义多标签性能度量指标的置信度。受到多类（multi-class）学习问题中的准确度的置信度的启发，本论文中定义了 6 种多标签性能度量的置信度。

将预测矩阵与真实标签矩阵进行比较，汉明损失关心的是单个比特的正确性，单错误关心的是最接近 1 的元素，而其他则关心每行或每列的排名排列。一般来说，基于标签的度量和基于实例的度量是完全不同的，因此要对它们分别处理。图 3 通过考虑每个度量的内在含义总结了计算方法。

Measure	Confidence
◇hamming loss	$\frac{1}{m}\sum_{i=1}^{m} p_{ij}\mathbb{I}[p_{ij} > 0.5] + (1-p_{ij})\mathbb{I}[p_{ij} \leq 0.5]$
⋆one-error	$\max_{j=1,\ldots,l} p_{ij}$
⋆coverage	$1 - \frac{1}{l}\sum_{j=0}^{l}\left[j \cdot p_{ij} \prod_{k=j+1}^{l}(1-p_{ik})\right]$
⋆ranking loss	$\sum_{j=0}^{l}\prod_{k=1}^{j} p_{ik} \prod_{k=j+1}^{l}(1-p_{ik})$
⋆average precision	$\sum_{j=0}^{l}\prod_{k=1}^{j} p_{ik} \prod_{k=j+1}^{l}(1-p_{ik})$
◇macro-AUC	$\sum_{i=0}^{m}\prod_{k=1}^{i} p_{kj} \prod_{k=i+1}^{m}(1-p_{kj})$

图 3. 六种多标记度量的置信度计算方法

矩阵 P 是 H_t 在第一维度上（森林）的平均值，元素 p_{ij} 表示预测为正的标签 $P_r[\hat{y}_{ij}=1]$。在不失一般性的情况下，当度量是基于实例时，按降序排列 P 的每一行的元素；当度量是基于标签时，我们按降序排列 P 的每一列的元素。明确地说，对于汉明损失，我们计算每位的最大置信度。例如，某列的预测向量 $p_{\cdot j} = [0.9, 0.6, 0.4, 0.3]$，因此对于这一列的置信度是 $\alpha_j = \frac{1}{4}(0.9+0.6+0.6+0.7) = 0.7$。对于单错误，我们选择最可信的标签为正的概率。对于覆盖率，我们将能够覆盖所有相关标签的标签数进行概率求和，并将结果缩放到 $[0,1]$ 范围内。对于排序损失，我们计算排序损失为零的概率，即正标签先于负标签。例如，某行的预测向量 $p_{i\cdot} = [0.9, 0.6, 0.4, 0.3]$，则有 5 种的排列会导致排序损失为零：$\{0000, 1000, 1100, 1110, 1111\}$。每种情况的概率都很容易获得，例如.$P_r[1100] = 0.9 \cdot 0.6 \cdot (1-0.4) \cdot (1-0.3)$。因此，我们可以通过对这 5 种情况下的概率求和得到置信度。以类似的方式定义了 macro-AUC 和平均精度的置信度。

图 4 总结了度量感知的特征复用过程。由于基于标签的度量和基于实例的度量之间的多样性，我们需要分别对它们进行处理。具体来说，基于标签的度量在第三个维度上（H_t 的维度：样本）计算 H_t 的置信度，而基于实例的度量在第二个维度上（H_t 的维度：标签）计算 H_t 的置信度。在置信度计算之后，当置信度 α_t “低于” 阈值时，我们重新用之前的特征表示 G_{t-1}，用较好的特征表示部分来更新 G_t。

整个度量感知的特征复用过程不依赖于真实标签，论文是通过训练过程中确定的阈值来判断特征表示的优良性的。正如图 6 所示，当评估的性能度量在第 t 层变差时，我们将置信度 α_t 保存到集合 S 中。然后，基于集合 S 确定阈值 θ_t，为了方便起见，论文使用 S 的平均值作为阈值。

Algorithm 1 Measure-aware feature reuse

Input: measure M, forests' output $\mathbf{H}^t$, previous $\mathbf{G}^{t-1}$.
Output: new representation $\mathbf{G}^t$.
Procedure:
Initialize tensor $\mathbf{G}^t = \mathbf{H}^t$.
if Measure M is label-based **then**
  for $j = 1$ **to** l **do**
    compute confidence α_j^t on $\mathbf{H}_{\cdot\cdot j}^t$
    Update $\mathbf{G}_{\cdot\cdot j}^t$ to $\mathbf{G}_{\cdot\cdot j}^{t-1}$ when $\alpha_j^t < \theta_t$.
  end for
end if
if Measure M is instance-based **then**
  for $i = 1$ **to** m **do**
    compute confidence α_i^t on $\mathbf{H}_{\cdot i\cdot}^t$
    Update $\mathbf{G}_{\cdot i\cdot}^t$ to $\mathbf{G}_{\cdot i\cdot}^{t-1}$ when $\alpha_i^t < \theta_t$.
  end for
end if

图 4. 度量感知的特征复用过程

Algorithm 2 Determine threshold

Input: measure M, forests' output $\mathbf{H}^t$, ground-truth $\mathbf{Y}$, previous performance on layer_{t-1}.
Output: threshold θ_t.
Procedure:
Initialize confidence set $\mathcal{S} = \emptyset$.
if Measure M is label-based **then**
  for $j = 1$ **to** l **do**
    compute confidence α_j^t on $\mathbf{H}_{\cdot\cdot j}^t$.
    compute measure m_j^t on $(\mathbf{H}_{\cdot\cdot j}^t, \mathbf{Y}_{\cdot j})$.
    $\mathcal{S} = \mathcal{S} \cup \{\alpha_j^t\}$ when m_j^t is worse than m_j^{t-1}.
  end for
end if
if Measure M is instance-based **then**
  for $i = 1$ **to** m **do**
    compute confidence α_i^t on $\mathbf{H}_{\cdot i\cdot}^t$.
    compute measure m_i^t on $(\mathbf{H}_{\cdot i\cdot}^t, \mathbf{Y}_{i\cdot})$.
    $\mathcal{S} = \mathcal{S} \cup \{\alpha_i^t\}$ when m_i^t is worse than m_i^{t-1}.
  end for
end if
θ_t = Compute threshold on $\mathcal{S}$.

图 5. 阈值取值过程

3.3 度量感知层增长机制

度量感知的特征复用机制专注于表示学习，能够有效地增强由各种性能度量引导下的特征表示。同时，为了减少过拟合和控制模型复杂度，论文提出了度量感知的层增长机制。如果用同样的数据拟合森林，直接进行预测，会增加过拟合的风险。MLDF 使用 k 折交叉验证来缓解这一问题。对于每个 fold，我们根据其他 fold 中的样本来训练森林并预测当前 fold。

MLDF 是逐层构建的，图 6 总结了度量感知层增长过程，即 MLDF 的训练过程。输入为最大层数 T、评价指标 M 和训练数据 { X，Y }。在每一层中选择一个 RF-PCT 和一个

ERF-PCT，并在每个森林中随机选择 $\sqrt{d}$ 个特征作为候选特征。MLDF 中训练森林的所有参数，如森林的数量和树木的深度，在训练前都是事先设定好的。由于希望每一层都能学习到不同的特征表示，所以我们可以设置随着第 i 层生长的森林中树的最大深度，树的数量也可以预先设置。在初始化步骤中，性能向量 q 记录了每层训练数据上的性能值，需要根据不同的度量进行初始化。在每一层中，首先拟合森林 (第 3 行)，得到特征表示 H_t(第 4 行)。然后确定阈值 θ_t(第 5 行)，通过度量感知的特征复用生成新的特征表示 G_t(第 6 行)。最后将层添加到模型集 C (第 14 行) 中。

层增长是可测量的，在拟合一层后，需要计算性能度量。当最近三层（第 11 行）的性能没有好转时，MLDF 停止增长。同时，应记录训练数据集中性能最佳的层索引，这对预测很有用。根据奥卡姆剃刀法则，当性能相似时，我们选择更简单的模型。当性能没有明显提高，最终的模型集应为 $C = \{layer_1, layer_2, ..., layer_L\}$，即删除 $layer_L$ 后面的层。

不同的衡量标准代表不同的用户需求。我们可以根据各种情况将 M 设置为所需的度量。因此，我们可以获得指定度量的相应模型。综上，度量感知层增长机制可以控制模型复杂度，帮助度量感知特征复用机制提升性能。

Algorithm 3 Measure-aware layer growth

Input: maximal depth T, measure M, training data $\{\mathbf{X}, \mathbf{Y}\}$.
Output: model set $\mathcal{C}$, threshold set Θ and final layer index L.
Procedure:

1: Initialize parameters:
   performance in each layer $\boldsymbol{q}[1:T]$,
   best performance on train set q_{best},
   the initial threshold $\theta_1 = 0$,
   the best performance layer index $L = 1$,
   the model set $\mathcal{C} = \emptyset$.
2: **for** $t = 1$ **to** T **do**
3: Train forests in layer_t and get classifier h_t.
4: Predict $\mathbf{H}_t = h_t([\mathbf{X}, \mathbf{G}_{t-1}])$.
5: $\theta_t =$ Determine Threshold (Algorithm 2) when $t > 1$.
6: $\mathbf{G}^t =$ measure-aware feature reuse (Algorithm 1).
7: Compute performance $\boldsymbol{q}[t]$ on measure M with $\mathbf{G}^t$.
8: **if** $\boldsymbol{q}[t]$ is better than q_{best} **then**
9: Update best performance $q_{\text{best}} = \boldsymbol{q}[t]$.
10: Update the layer index of best performance $L = t$.
11: **else if** q_{best} is not updated in recent 3 layers **then**
12: **break**
13: **end if**
14: Add layer_t to model set: $\mathcal{C} = \mathcal{C} \cup \text{layer}_t$.
15: **end for**
16: Keep $\{\text{layer}_1, \ldots, \text{layer}_L\}$ in model set $\mathcal{C}$ and drop others $\{\text{layer}_{L+1}, \ldots\}$.

图 6. 感知度量层增长过程

4 复现细节

4.1 与已有开源代码对比

本论文的相关代码已开源：源代码地址，本次复现工作在此开源代码的基础上进行。源代码并没有提供所有数据集，数据集需另行下载：数据集地址。

4.2 数据处理

下载得到的数据集格式都是 arff (Attribute-Relation File Format) 文件，这是一种 ASCII 文本文件，我们需要先将其转换成 npy 文件才方便用于我们的复现工作中。下面举 CAL500 数据集的处理为例子。

```
# 将目录中的CAL500数据集进行读取
data = arff.loadarff('./datasets/CAL500/CAL500.arff')
df = pd.DataFrame(data[0])
# 使用filter选择列
X = df.filter(regex="Mean|Std")
y = df.filter(regex="Emotion|Genre|Instrument|Song|Usage|Vocals")
y = y.applymap(lambda x:x if pd.isna(x) \
                          else str(x).replace('b',''))
for col in y.columns:
    y[col] = y[col].apply(lambda x: int(x.strip("'")))
X = X.values
y = y.values
file_path_X = "./data/CAL500_data.npy"
file_path_y = "./data/CAL500_label.npy"
np.save(file_path_X, X)
np.save(file_path_y, y)
```

9 个数据集中还涉及到稀疏数据，对于是稀疏数据的数据集还需要进行特殊处理，下面是处理稀疏数据的函数。

```
def read_sparse_arrf(file):
    with open(file, encoding="utf-8") as f:
        header = []
        default_field = {}
        field_num = 0
        for line in f:
            if line.startswith("@attribute") or \
               line.startswith("@ATTRIBUTE"):
                _, name, field_type = line.split()
                header.append(name)
                if field_type.startswith("{"):
```

```
                    default_field[field_num] = \
                    field_type[1:-1].split(",")[0]
                field_num += 1
            elif line.startswith("@data") or \
                 line.startswith("@DATA"):
                break
        default_field_keys = set(default_field.keys())
        width = len(header)
        data = []
        for line in f:
            line = line.strip()
            if not line:
                continue
            tmp = [0]*width
            flags = set()
            for kv in line[1:-1].split(","):
                k, v = kv.split()
                k = int(float(k))
                if k not in default_field_keys:
                    v = int(float(v))
                flags.add(k)
                tmp[k] = v
            for k in default_field_keys-flags:
                tmp[k] = default_field[k]
            data.append(tmp)
    return pd.DataFrame(data, columns=header)
```

4.3 代码复现

先创建 python==3.6 的 conda 环境，再安装需要的第三方库：numpy、scipy、scikit-learn。特别注意，其中 scikit-learn 的版本选择 0.17.1 版本，因为官方提供的代码使用的是旧版本的 cross_validation.KFold 与新版本的 model_selection.KFold 差异比较大，为了方便，直接安装旧版 scikit-learn 即可。

```
pip install -U scikit-learn==0.17.1
```

还需修改 comb 的导入，因为它在新版本的 scipy 库的位置已经被修改。在 Anacanda 或 python 环境下，找到以下两个文件，并对代码进行修改。

文件 1: lib\site-packages\sklearn\model_selection\split.py

文件 2：lib\site-packages\sklearn\metrics\cluster\supervised.py

```
# 原来的代码
from scipy.misc import comb
```

```python
# 修改成以下代码
from scipy.special import comb
```

4.4 代码改进及创新点

在原论文框架中，每一级联层的每个森林的训练结果会进行取平均操作，平均之后的训练结果用于计算性能和计算置信度，再与阈值进行对比。取平均可能会带来一定的损失，所以我不做取平均的操作，而是将两个森林先分别进行度量感知特征复用机制。对于度量感知层增长机制，是对进行度量感知特征复用机制的两个森林产生的结果进行取平均操作，然后求性能决定是否进行层增长。

5 实验结果分析

5.1 代码复现实验结果

本次复现工作在 9 个数据集上，分别测试了 MLDF 的性能表现，并与原论文的结果数据进行比较，数据接近原论文的结果数据，说明了复现代码的有效性与 MLDF 的高效性。以下为实验结果展示，标星号的是原论文的数据结果，黑三角形是用两个机制的 MLDF，剩下的是没有度量感知特征复用机制的 MLDF。

Indicator	CAL500	enron	image	scene	yeast
⋆ hamming loss	0.136±0.001	0.046±0.000	0.148±0.003	0.082±0.002	0.190±0.003
▲ hamming loss	0.138±0.000	0.050±0.000	0.150±0.000	0.082±0.000	0.202±0.000
hamming loss	0.136±0.000	0.049±0.000	0.150±0.000	0.080±0.000	0.197±0.000
⋆ one-error	0.122±0.009	0.216±0.009	0.239±0.008	0.188±0.005	0.223±0.010
▲ one-error	0.112±0.000	0.314±0.000	0.251±0.001	0.195±0.000	0.238±0.000
one-error	0.122±0.000	0.232±0.000	0.253±0.000	0.191±0.000	0.227±0.000
⋆ coverage	0.741±0.006	0.223±0.003	0.159±0.004	0.064±0.003	0.434±0.004
▲ coverage	0.752±0.000	0.245±0.000	0.159±0.000	0.065±0.000	0.442±0.000
coverage	0.7446±0.000	0.233±0.000	0.161±0.000	0.065±0.000	0.440±0.000
⋆ ranking loss	0.176±0.002	0.077±0.001	0.129±0.005	0.059±0.004	0.160±0.006
▲ ranking loss	0.187±0.000	0.119±0.004	0.131±0.001	0.062±0.000	0.169±0.000
ranking loss	0.179±0.000	0.082±0.000	0.133±0.000	0.062±0.000	0.166±0.000
⋆ average precision	0.512±0.003	0.696±0.004	0.842±0.005	0.891±0.008	0.770±0.005
▲ average precision	0.490±0.000	0.623±0.000	0.837±0.001	0.886±0.000	0.755±0.000
average precision	0.504±0.000	0.673±0.002	0.834±0.000	0.887±0.000	0.762±0.000
⋆ macro-AUC	0.568±0.006	0.742±0.014	0.885±0.003	0.956±0.003	0.732±0.010
▲ macro-AUC	0.500±0.000	0.705±0.000	0.883±0.000	0.952±0.000	0.713±0.000
macro-AUC	0.565±0.001	0.730±0.001	0.882±0.000	0.953±0.000	0.762±0.000

Indicator	corel16k-s1	corel16k-s2	eurlex-sm	mediamill
⋆ hamming loss	0.018±0.000	0.017±0.000	0.006±0.001	0.027±0.001
▲hamming loss	0.019±0.000	0.017±0.000	0.008±0.000	0.029±0.000
hamming loss	0.019±0.000	0.018±0.000	0.006±0.000	0.030±0.000
⋆ one-error	0.640±0.003	0.639±0.004	0.138±0.001	0.147±0.005
▲ one-error	0.727±0.000	0.761±0.000	0.431±0.000	0.168±0.000
one-error	0.675±0.000	0.677±0.000	0.190±0.000	0.165±0.000
⋆ coverage	0.262±0.002	0.274±0.005	0.066±0.001	0.128±0.001
▲ coverage	0.330±0.000	0.355±0.000	0.093±0.000	0.139±0.000
coverage	0.296±0.000	0.284±0.000	0.045±0.000	0.144±0.000
⋆ ranking loss	0.143±0.002	0.138±0.002	0.014±0.001	0.034±0.001
▲ ranking loss	0.182±0.000	0.202±0.000	0.059±0.000	0.038±0.000
ranking loss	0.152±0.000	0.145±0.000	0.021±0.000	0.040±0.000
⋆ average precision	0.347±0.002	0.342±0.004	0.840±0.002	0.732±0.007
▲ average precision	0.288±0.000	0.257±0.000	0.583±0.000	0.706±0.000
average precision	0.319±0.000	0.311±0.000	0.788±0.000	0.698±0.000
⋆ macro-AUC	0.728±0.001	0.737±0.007	0.930±0.002	0.842±0.002
▲ macro-AUC	0.667±0.000	0.634±0.000	0.872±0.000	0.822±0.000
macro-AUC	0.713±0.000	0.718±0.000	0.923±0.000	0.823±0.000

5.2 分析代码复现中的问题

在小数据集上，无论有没有度量感知机制结果都很近似原论文结果；在大数据集上，没有度量感知特征复用机制的结果接近原论文结果，但有细微差距，而且优于使用了度量感知特征复用机制的结果，因此设置消融实验对原因进行说明原因并分析：

MLDF 随机性太强；多标签随机森林的随机性极强，而随机种子的取值可以是任意的整数。我们从数据集和算法两个方面考虑：1. 数据集：不同的划分可能带来不同分布的数据集，在整个框架中存在的数据集上的随机性的地方有预处理阶段数据集的划分和每个级联层的五折交叉验证过程。2. 算法：随机森林的随机性，每个级联层包含两种森林，特别是完全随机森林，它是随机挑选所有特征中的一个进行分裂生成，所可能生成的森林是变化莫测的。

为了验证以上的想法，我做了多组消融实验来说明，以下是在 image 数据集上进行实验，第一列表示的是哪些地方设置了随机种子（"random_state=42"，s 代表 shuttle）结果如下表。通过多组消融实验发现，有无设置随机种子、在哪设置随机种子对 image 数据集的各个指标的性能表现是有一定的影响的。

	hamming loss	one-error	coverage	ranking loss	average precision	macro-auc
kf, forest	0.1430	0.2292	0.1554	0.1233	0.8469	0.8881
s, forest	0.1504	0.2522	0.1610	0.1341	0.8343	0.8803
s, kf	0.1456	0.2234	0.1536	0.1206	0.8501	0.8879
s	0.1476	0.2324	0.1566	0.1247	0.8449	0.8859
kf	0.1560	0.2578	0.1631	0.1369	0.8313	0.8786
forest	0.1462	0.2438	0.1593	0.1325	0.8387	0.8817
none	0.1524	0.2668	0.1666	0.1407	0.8268	0.8773
all	0.1459	0.2274	0.1552	0.1234	0.8475	0.8874

5.3 代码创新实验结果

在 image 数据集和 scene 数据集上进行创新实验，实验结果如下。

image	hamming loss	one-error	coverage	ranking loss	average precision	macro-auc
创新前	0.1524	0.2668	0.1666	0.1407	0.8268	0.8773
创新后	0.1481	0.2494	0.1601	0.1318	0.8367	0.8835
差值	0.43%	1.74%	0.65%	0.89%	0.99%	0.62%

scene	hamming loss	one-error	coverage	ranking loss	average precision	macro-auc
创新前	0.0840	0.1930	0.0662	0.0628	0.8863	0.9522
创新后	0.0819	0.1907	0.0652	0.0611	0.8885	0.9526
差值	0.21%	0.23%	0.10%	0.89%	0.99%	0.62%

通过实验可以发现，在 image 数据集上的提升比较明显，但是在 scene 数据集上的提升比较微小，但是这里微小的提升是以训练时间代价换取的，所以还是需要进一步考虑这两者之间的平衡。

6 总结与展望

这篇论文首先将深度森林框架引入到多标记学习中，提出了多标记深度森林 (Multi-Label Deep Forest, MLDF)。设计的多层结构使 MLDF 能够利用标签之间的相关性。由于度量感知的两种机制，度量感知的特征复用和度量感知的层增长，我们的方案可以根据用户的需求优化不同的多标签度量，降低过拟合的风险，并在一组基准数据集上取得了最好的结果。实验表明，在广泛的基准数据集上取得了优异的性能。未来可以考虑将森林替换成分类器链，来进行多标签分类。这样不仅保留了森林的前向传播，而且增加了向后传播，进一步学习神经网络的结构，但是又避免了神经网络过高的复杂度和过多的参数。

参考文献

[1] Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. NIPS'18, page 3555–3565, Red Hook, NY, USA, 2018. Curran Associates Inc.

[2] Ji Feng and Zhi-Hua Zhou. Autoencoder by forest. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 04 2018.

[3] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. Large-scale multi-label text classification —revisiting neural networks. In *Machine Learning and Knowledge Discovery in Databases*, page 437–452, Berlin, Heidelberg, 2014. Springer-Verlag.

[4] Grigorios Tsoumakas and Ioannis Manousos Katakis. Multi-label classification: An overview. *Int. J. Data Warehous. Min.*, 3:1–13, 2007.

[5] Grigorios Tsoumakas, Ioannis Manousos Katakis, and Ioannis P. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, 2010.

[6] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2285–2294, 2016.

[7] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[8] Qian-Wen Zhang, Yun Zhong, and Min-Ling Zhang. Feature-induced labeling information enrichment for multi-label learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[9] Ya-Lin Zhang, Jun Zhou, Wenhao Zheng, Ji Feng, Longfei Li, Ziqi Liu, Ming Li, Zhiqiang Zhang, Chaochao Chen, Xiaolong Li, Yuan (Alan) Qi, and Zhi-Hua Zhou. Distributed deep forest and its application to automatic detection of cash-out fraud. *ACM Trans. Intell. Syst. Technol.*, 10(5), sep 2019.

[10] Zhi-Hua Zhou and Ji Feng. Deep forest. *National Science Review*, 6(1):74–86, 10 2018.