FastXML的模型细节

Saturday, July 13, 2019 16:34

typical multi-label metrics that we have used such as F1-score and Hamming loss would give equal weight to positive and negative labels. It is more important for us to correctly predict the positive labels vs the negative labels.

Fast extreme multi-label classification结构和思想

多标签分类方法有：1-vs-all、label-hierarchy（tree-based methods）；后者往往比前者好，但是基于tree的方法也有可能很昂贵，现在的state-of-art主要是label partitioning by sublinear ranking(LPSR),Multi-label random forest(MLRF)两种方法，LPSR除了要训练基本的分类器，还要学习一个层级化的标签系统，而MLRF在高维空间上也很昂贵。

本文主要是要实现一个1.快速训练、2.更加准确的极限多标签分类器；主要的贡献有：

1. 一个新的节点划分方法；
2. 一个高效的优化目标损失：rank sensitive loss function；LPSR采用的是clustering error，MLRF采用的是Gini index；

采用precision@k和nDCG@k的度量标准

FastXML is a tree based classifier algorithm that partitions the feature space instead of the label space. The intuition is that there are only a few labels associated with a region of the feature space. At any region of the feature space, the set of active labels are the union of the labels of all the region's training points. At each node, the parent feature space is partitioned into a left and right child space. In decision trees, Gini impurity, Information gain or clustering error is typically used to split each node.
As these measures are not suited for extreme multi-label applications, the authors have proposed to use nDCG instead as it is a ranking loss function. This will recommend better labels as relevant positive labels with the highest rank are returned. nDCG is optimized across all LL labels at the current node;

FastXML也要学习一个[illegible]这个层级化思想和LPSR和MLRF相似，但是这

Fast XML也要学习一个hierarchy，这个层级化思想和LPSR和MLRF相似，但是这个hierarchy是在特征空间学的，因为发现特征空间的每个区域只会有少数标签出现；因此只需要对一个数据点遍历整个层级化树，找到其特征空间，然后考虑这个特征空间下的标签就可以了。

- FastXML中一个特征空间下的标签集合由训练集中所有出现在该特征空间下的标签组成。
- FastXML学习的是**an ensemble of trees**，不需要依赖于基本分类器；
- 预测的时候：**多棵树预测的结果标签按出现频率排序，返回的是频率出现的最高的一些标签。**

节点划分方法

Training FastXML consists of recursively partitioning a parent's feature space between its children.

优化这个过程往往是优化一个全局的表现，例如叶节点的排序预测，但是有个缺点是这个树的所有节点都要同时优化，这样训练就比较昂贵；因此现有的一些方法进行只针对当前节点预测的局部优化，这样就允许从根节点到叶节点的逐个优化，更加高效；

FastXML不采用Gini index或者clustering error的局部表现，而是提出优化一个ranking loss function，也就是优化“the normalized Discounted Cumulative Gain(nDCG)”,这个损失函数有什么优点呢？

- nDCG is a measure which is sensitive to both ranking and relevance and therefore ensures that the relevant positive labels are predicted with ranks that are as high as possible. This cannot be guaranteed by rank insensitive measures such as the Gini index or the clustering error.
- can be optimized across all L labels at the current node thereby ensuring that the local optimization is not myopic.

下面再具体讲解下决策树实现的思路

其思想是基于随机森林的思想（bagging类的集成方法），在特征空间进行特征的划分，每个叶子节点只包含了与这个特征子空间相关的标签，这篇论文的创新点也在于节点的划分目标函数以及它提出了新的评价指标来评价分类器的一个好坏，也就是DCG（或nDCG, normalized discounted cummulative gain),为什么采用这个评价指标呢？文章里面说了两点：

a. nDCG对标签的排序和标签的相关性都有关，也就是说这个指标希望和相关的标签也排在前面；而Gini指数、分类错误率等都是rank insensitive的评

价标准。

b. 在训练的时候，这个指标是在所有节点上都计算了所有的标签，比只考虑topk的指标要能获得更多的信息；

那么nDCG具体是怎么构建的呢？

对于一个样本的标签$y \in \{0,1\}^L$,其标签索引的一个排列并取topk的结果为：

$$\mathrm{rank}_k(\mathbf{y}) = [i_1^{\mathrm{desc}}, \ldots, i_k^{\mathrm{desc}}]^\top$$

那么对于一个样本，它的标签的理想排列为：所有和这个样本相关的label都排在不相关的label的前面，然后我们构建DCG@k(就是说取前k个标签)指标为：

$$\mathcal{L}_{\mathrm{DCG@}k}(\mathbf{r}, \mathbf{y}) = \sum_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)}$$

其中，y_{rl} 表示标签索引的排列为r，其第l个索引的标签值为y_{rl}, 它的取值为0或者1，l的取值为1到k，也就是只计算前k个标签，从这个指标的构建我们可以发现：指标利用了和这个数据相关的标签，且这个相关的标签排在越前面越它的值越大，因此这个公式也能表达我们希望得到的标签排列。同样，我们得到了归一化的nDCG@k的值：

$$\mathcal{L}_{\mathrm{nDCG@}k}(\mathbf{r}, \mathbf{y}) = I_k(\mathbf{y}) \sum_{l=1}^{k} \frac{y_{r_l}}{\log(1+l)}$$

$$\text{where } I_k(\mathbf{y}) = \frac{1}{\sum_{l=1}^{\min(k, \mathbf{1}^\top \mathbf{y})} \frac{1}{\log(1+l)}}$$

一方面，我们用nDCG作为我们的评价指标，另一方面，我们也借助nDCG来构建节点划分的目标函数，类似Gini系数、分类错误率等。

目标函数

决策树的每一个节点都是有权重的，它是一个线性分类器w，一个样本经过每个节点的时候，我们需要判断是将其分到左边的特征空间还是右边的特征空间，比如这个样本是$X^i \in R^D$,$w \in R^D$,那么当wx^i大于0时，我们分到右子树，否则分到左子树，那么我们如何选择这样的w呢？我们构建了这样的目标函数：

$$\min \quad \|\mathbf{w}\|_1 + \sum_i C_\delta(\delta_i) \log(1 + e^{-\delta_i \mathbf{w}^\top \mathbf{x}_i})$$

$$- C_r \sum \tfrac{1}{2}(1+\delta_i) \mathcal{L}_{\mathrm{nDCG@}L}(\mathbf{r}^+, \mathbf{y}_i)$$

$$- C_r \sum_{i}^{i} \frac{1}{2}(1-\delta_i)\mathcal{L}_{\mathrm{nDCG@}L}(\mathbf{r}^-, \mathbf{y}_i)$$

$$\text{w.r.t.} \quad \mathbf{w} \in \mathcal{R}^D, \boldsymbol{\delta} \in \{-1,+1\}^L, \mathbf{r}^+, \mathbf{r}^- \in \Pi(1,L)$$

我们要选择使得这个目标函数最小的w，这个目标函数里面需要优化的参数就只有w，我们来看看这个目标函数表达的含义是什么呢？

- $||W||_1$表示的是结构风险；
- $\sum_i C_\delta(\delta_i)\log(1+e^{-\delta_i w x^i})$ 表示的是一个数据点x^i被分到右分支还是分到做分支了，wx^i的结果越大，就说明分到某一边的确定性越大，这样的w就是好的w，而这一部分的损失就小；注意这里$\delta_i = \mathrm{sign}(wx^i)$
- 第三项和第四项分别表示和样本相关的标签的nDCG值，以及和样本不相关标签的nDCG值，那样本不相关时的计算方法应该表示为：
- $L_{nDCG_{@k}}(r^-, y) = I_k(y)\sum_{l=1}^{k}\frac{1-y_{rl}}{\log(1+l)}$

这个目标函数经过分步优化之后就能得到比较好的参数了，这个模型训练停 止的条件就是r^+, r^-, δ, w均不再变化。

那测试时我怎么预测每个数据的标签呢？

因为我们使用的是集成的方法，我们每棵树都能将我们的测试数据分到一个页节点上，而每个叶节点都对应了一部分标签子集，我们通过统计每棵树预测的标签，并统计频率，最后再求一个均值，最后再排序取topk的结果，也就是我们的rank@k（...）的结果：

$$\mathbf{r}(\mathbf{x}) = \mathrm{rank}_k\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{P}_t^{\mathrm{leaf}}(\mathbf{x})\right)$$