面向相似 App 推荐的列表式多核相似性学习算法①

卜 宁, 牛树梓, 马文静, 龙国平

(中国科学院 软件研究所, 北京 100190)

摘 要: 相似 App 推荐可以有效帮助用户发现其所感兴趣的 App. 与以往的相似性学习不同, 相似 App 推荐场景主要面向的是排序问题. 本文主要研究在排序场景下如何学习相似性函数. 已有的工作仅关注绝对相似性或基于三元组的相似性. 本文建模了列表式的相似性, 并将三元组相似性与列表式相似性用统一的面向排序场景的相对相似性学习框架来描述, 提出了基于列表的多核相似性学习算法 SimListMKL. 实验证明, 该算法在真实的相似 App 推荐场景下性能优于已有的基于三元组相似性学习算法.

关键词: 相似 App 推荐; 多核学习; 相对相似性; 相似性学习; 列表式学习

Listwise Multi-Kernel Similarity Learning Algorithm for Similar Mobile App Recommendation

BU Ning, NIU Shu-Zi, MA Wen-Jing, LONG Guo-Ping

(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Similar App recommendation is useful for helping users to discover their interested Apps. Different from existing similarity learning algorithms, the similar App recommendation focuses on presenting a ranking list of similar Apps for each App. In this paper, we put emphasis on how to learn a similarity function in a ranking scenario. Previous studies model the relative similarity in the form of triplets. Instead of triplets, we model the ranking list as a whole in the loss function, and propose a listwise multi-kernel similarity learning method, referred as SimListMKL. Experimental results on real world data set show that our proposed method SimListMKL outperforms the baselines approaches based on triplets.

Key words: similar App recommendation; multi-kernel learning; relative similarity; similarity learning; listwise learning

智能手机和移动互联网的广泛普及, 带动了移动应用(Mobile Application, 简称 App)的飞速发展, 越来越多的用户每天都要使用移动 App, 但是用户发现找到自己感兴趣的 App 越来越困难. 相似 App 推荐可以有效缓解这个问题. 例如, Google Play 应用商店在每个 App 的详情页面提供与该 App 相似的 App 列表. 然而, 如何定义两个 App 之间的相似程度成为相似 App 推荐中的关键问题.

移动 App 本质是一个应用程序, 但实际上公开可获得的只是其基本情况描述信息, 本文中称为 App 的元信息. 这些元信息通常存在于移动 App 商店中, 包括名称, 功能描述, 屏幕截图, 用户评论, 开发商, 用户评分, 大小等信息. 因此本文所关注的移动 App 的本质是一个异质信息的聚合体. App 相似性学习可以转换为多元异质信息间的相似性学习.

已有的相似性学习算法可以分为基于特征的学习算法与基于核函数的学习算法. 前者侧重于多元异质信息的表示学习, 后者更侧重于如何融合多个异质的核函数. 本文采用主流的基于多元核函数的方法来学习 App 相似性, 优势在于便于针对不同的信息采用合适的核函数, 有效的规避了信息间的异质性问题. 在面向排序的的相似性学习场景中, 给定目标对象 a, 我们需要一个按与 a 相似程度, 从由高到低排序的列表, 而非与其中某个对象相似程度的一个具体值, 例如相似 App 推荐. 相似 App 列表整体质量的好坏是核心. 已有一些工作[1,2]关注于三元组形式的相对相似性, 如

① 收稿时间:2016-04-14;收到修改稿时间:2016-05-12 [doi:10.15888/j.cnki.csa.005502]

(a,b,c)描述了对于对象 a 来说, b 比 c 更相似. 基于三元组的相似性学习算法将相似性学习的问题归约为对于对象 a, 对象 b 是否比 c 更相似的分类问题, 试图优化分类错误. 这样做只考虑了局部序的损失, 与我们所关心的排序列表的整体质量不一致, 因此本文提出直接建模排序列表(Listwise)质量, 将其作为优化目标, 来学习 App 相似性, 称为 SimListMKL. 在真实数据集上的实验结果表明, 本文提出的 SimListMKL 算法要优于主流的基于三元组的相对相似性学习算法.

1 相关工作

相似性学习是一种有监督的学习方式, 它从已有的相似关系的对象中学习相似性函数, 度量两个对象的相似程度[1]. 相似性学习所面向的任务有回归, 分类与排序等. 本文主要关注面向排序场景的相似性学习. 基于特征的方法在于学习一个特征映射函数, 然后根据该映射定义相似性函数. 基于核函数的方法包括单核函数的方法与多核函数方法, 其中单核函数方法与基于特征的方法类似, 多核函数方法一般要优于单核方法, 但计算代价高. 因此本文主要从基于特征的方法(单核方法)与基于多核函数的方法两方面阐述.

已有的基于特征的相似性学习的方法直接从成对的相似关系约束中学习相似性函数. 这些工作假设相似性度量的应用场景为回归或分类, 因此大多将相似性学习的问题归约为分类问题. 文献[3]将脸部识别问题归约为差分空间中的相似性判别二分类问题, 并采用 SVM 求解. 相似性神经网络[5](Similarity Neural Network)提出采用前向多层感知机来学习一个非负且对称的相似性度量, 其中非负性由输出层的 sigmoid 激活函数保证, 对称性由网络结构的权重共享机制决定. 文献[6]将相似性度量学习的问题转化为对象的特征权重的学习问题, 优化目标是经过特征权重的线性变换使得相似的两个对象距离尽可能小, 并约束不相似的两个对象间的距离尽量大. 成对相似性关系大多是一种绝对的表示, 但是相对的相似性关系的比较是表达领域知识的更为自然的方式, 因为相似性的知识通常是模糊的, 只有相对意义上的表示. 对象 a 与对象 b 相似度比对象 c 与对象 d 之间的更高, 这一点很容易确定. 对于对象 a 与对象 b 之间是否相似给一个绝对的值要困难一些. 有些基于特征的相似性学习方法关注从这种相对的相似性关系中学习相似性函数. 文献[7]考虑了将相对相似性约束应用到聚类问题中所面临的可行性, 完备性以及信息量. 文献[8]建模了这种相对相似性的领域知识, 将关于相对相似性的约束的均方残差作为优化目标, 有效改善了聚类准确率.

基于多核函数的方法是目前解决该问题的主流方法. 基于多核函数的相似性学习通过对多个核函数进行综合获得最优的描述相似性的核函数[9,10]. 文献[11]将学习多个核函数的组合权重的问题简化为分类问题, 采用类似于 AdaBoost 的方法求解. 考虑到相似度量的非负性, 文献[12]将成对对象作为训练实例, 这样每一个训练实例都用若干核函数来描述, 采用 epsilon 不敏感回归函数作为优化目标, 找到核函数的最优组合. 面向排序场景的多核函数方法大将相对相似性的比较作为监督信息来学习多个核函数的权重[1,2]. 考虑多核方法的计算代价高, SimApp[1]与 OASIS[2]均建模 hinge 损失函数, 前者采用在线梯度下降的优化算法得到核函数权重称为在线核函数权重学习(OKWL)方法, 后者采用在线的 Passive-Aggressive 算法求解.

2 面向排序的相似性学习框架

本文首先给出 App 相似性问题的形式化定义, 接下来基于多核学习思想, 提出了面向排序的相似性学习框架, 最后给出了具体实例.

2.1 App 相似性学习问题形式化

描述 App 的元信息包括多种类型, 如名称, 功能描述等为文本信息, 屏幕截图为图像信息, 用户评分为数值信息等. App 间的语义相似性直接由描述 App 的文本间, 图像间以及文本与图像间的语义相似性等决定. 因此本文将 App 相似性形式化为问题 1.

问题 1(App 相似性学习). 给定 App 的集合 A, $\forall a \in \mathrm{A}$, a 是多种信息的聚合体, 其中 $a(1)$表示名称, $a(2)$表示功能描述文本, $a(3)$表示屏幕截图等. $\forall a_i, a_j \in \mathrm{A}$, App 相似性学习问题旨在学到一个核函数 $f(a_i, a_j)$, 能够描述两个 App 之间的语义相似性.

基于多核学习的思想, 本文将 App 相似性函数 $f(a_i, a_j)$ 简化为只考虑对应信息间的相似性. 对于任一类信息 d, $a_i(d)$ 与 $a_j(d)$ 的相似性都可以用最合适的核函数 $\mathrm{K}_d(a_i, a_j)$ 来表示. 例如, 名称间的相似性采用字符串核函数来度量, 功能描述间的相似性采用 LDA 得到的话题分布的相似来度量, 屏幕截图的相似性采用基于 SIFT 表示的 RBF 核函数来度量等. 因此

App 相似性函数可以简化为如公式(1)所示.

$$f(a_i,a_j)=\sum_{d=1}^{D}\omega_d \mathrm{K}_d(a_i,a_j) \tag{1}$$

2.2 面向排序的相似性学习框架

传统的相似性学习问题认为相似性是绝对的, 例如 a 与 b 相似记为 1, 不相似记为 0, 因此采用分类或回归算法求解相似性学习的问题. 在排序场景下, 相对相似性比绝对相似性要重要. 例如, 相似 App 推荐只需关心前 k 个 App 比 k 个之后的 App 相比, 与当前 App 更相似即可, 而不需要确定具体的相似性分值是 1 还是 0.

假设训练集 $\{a^q,\{a_i^q\}_1^{n_q},Y^q\}_1^Q$, 其中 a^q 为查询 App, 表示比较的基准, $\{a_i^q\}_1^{n_q}$ 表示待比较的 App 集合, 而 Y^q 表示真实的相似性标注, 主要有以下两种形式.

① 绝对相似性标注 常见的二值绝对标注, 如 $Y^q(a^q,a_i^q)=1$ 表示两个 App 相似, 否则不相似.

② 相对相似性标注 为了表达相对相似性, Y^q 既可以为三元组的形式, 如 $Y^q(a^q,a_i^q,a_j^q)=1$ 表示 a^q 与 a_i^q 相似程度高于 a^q 与 a_j^q, 反之亦然, 也可以表示为排序列表的形式, 如 $(a^q,a_{i_1}^q)\prec\cdots\prec(a^q,a_{i_{n1}}^q)$.

绝对相似性可以推导出相对相似性标注. 若 $Y^q(a^q,a_{i+}^q)=1$ 且 $Y^q(a^q,a_{j-}^q)=0$, 则可以推出三元组形式的相似性 $Y^q(a^q,a_{i+}^q,a_{j-}^q)=1$, 也可以推出列表形式的相似性 $\{(a^q,a_+^q)\}\prec\{(a^q,a_-^q)\}$.

无论是绝对相似性标注, 还是相对相似性标注, 采用相对相似性的形式进行组织训练实例, 由此定义损失函数 $L(f,\{a^q,\{a_i^q\}_1^{n_q},Y^q\})$ 如公式(2)所示, 来学习 App 相似性函数 $f(a_i,a_j)$ 中的核函数权重. 这就是面向排序的相似性学习框架. 以下是两个具体的实例来说明相对相似性在损失函数中的建模.

$$\sum_{q=1}^{Q} l(f,\{\mathrm{K}_{d=1..D}(a^q,a_i^q)\}_1^{n_q},Y^q) \tag{2}$$

2.3 基于三元组的相似性学习算法

本文以排序学习[13]中最常见的三类建模成对实例的方式来定义损失函数. 以一个查询 App 的损失函数的定义 $l(f,\{\mathrm{K}_{d=1..D}(a^q,a_i^q)\}_1^{n_q},Y^q)$ 为例来说明. 将代表 a^q 与 a_i^q 关系的 D 维核函数特征记为 $\mathrm{K}(a^q,a_i^q)$, 因此损失函数表示为 $l(f,\{\mathrm{K}(a^q,a_i^q)\}_1^{n_q},Y^q)$.

① 基于交叉熵的损失函数 SimTripletNet

$$\sum_{t=(a^q,a_+^q,a_-^q)\in T_q} -\bar{P}_t\log P_t(f)-(1-\bar{P}_t)\log(1-P_t(f)) \tag{3}$$

其中 T^q 表示 a^q 中所有满足 $Y^q(a^q,\cdot,\cdot)=1$ 的三元组. 若三元组满足 $Y^q(a^q,\cdot,\cdot)=1$, 则 $\bar{P}_t=1$, 否则为 0. $P_t(f)$ 采用 logistic 函数定义三元组的生成概率, 即 $\mathrm{s}(f(a^q,a_+^q)-f(a^q,a_-^q))$.

② 基于指数函数的损失函数 SimTripletBoost

$$\sum_{(a^q,a_+^q,a_-^q)\in T_q} \exp(-(f(a^q,a_+^q)-f(a^q,a_-^q))) \tag{4}$$

③ 基于 Hinge 损失函数 SimTripletSVM

$$\sum_{(a^q,a_+^q,a_-^q)\in T_q} \max\{0,1-(f(a^q,a_+^q)-f(a^q,a_-^q))\} \tag{5}$$

已有的工作大多是采用的 Hinge 损失来从三元组中学习相对相似性[1,2], 根据采用不同的优化算法得到不同的算法. 本文中将他们统一称为 SimTripletSVM. 为了完备起见, 本文引入了其他可能的基于三元组的损失函数定义方式作为对比.

2.4 基于列表的相似性学习算法

基于三元组的相似性学习算法建模了三元组这样的局部序, 所带来的损失, 没有考虑排序列表整体的质量. 基于三元组的算法将整个相似性列表拆分为三元组的形式, 将每个三元组作为一个训练实例, 强制性的认为这些三元组服从独立同分布假设. 本文将整个排序列表视为一个训练实例, 采用基于位置的评价指标作为排序列表质量的度量, 将其作为优化目标, 采用 LambdaMART 定义损失函数的方式, 如公式(6)所示.

$$\sum_{a_i^q}-\sum_{a_j^q:a_i^q\Leftrightarrow a_j^q}|\Delta E_{ij}|\log s(f(a^q,a_i^q)-f(a^q,a_j^q)) \tag{6}$$

其中 $|\Delta E_{ij}|$ 表示交换预测序中 a_i^q 与 a_j^q 两个App的位置所带来的评价指标 E 的值的变化.

3 SimListMKL算法

本文采用 SimListMKL 算法来优化公式(6)所示的优化目标, 评价指标采用 MAP, 为基于所有查询 App 的 AP(Average Precision)的平均. 因此这里的 MAP 的变化应该对应到 AP 的变化. 公式(7)给出了对于一个查询 App 在已知其二值标注 Y 的情况下, 计算模型 f 产生的预测序的 AP 值.

$$AP(f,Y)=\frac{\sum_{r=1}^{n} rel(r)*P@r}{\sum_{r=1}^{n} rel(r)}$$

其中 $rel(r)=1$ 表示在预测序中第 r 个位置的 App 是否与当前查询 App 相似, 为 0 则不相似. $P@r$ 表示前 r

个位置中相似 App 的比例. 由此可以推知, 交换预测序中两个 App 的位置, 带来的 AP 变化如公式(8)所示.

$$|\Delta AP_{ij}| = (rel(r_i) - rel(r_j))(P@r_i - P@r_j) \quad (8)$$

SimListMKL 算法采用 MART 来优化目标函数公式(6). MART 是一类 Boosting 算法, 可以视为泛函空间的梯度下降算法.

SimListMKL 算法

Input: the number of trees N;
The number of leaves per tree L;
Learning rate η;
training samples $\{a^q, \{a_i^q\}_1^{n_q}, Y^q\}_1^Q$

Output: $f_N(a^q, a_i^q)$

1 For $q = 1\ to\ Q$ do
2 For $i = 1\ to\ n_q$ do
3 $f_0(a^q, a_i^q)$ = basemodel(K(a^q, a_i^q))
4 end
5 end
6 For t = 1 to N do
7 For q = 1 to Q do
8 For $i = 1\ to\ n_q$ do
9 $y_i^q = \sum_{a_j^q : a_i^q \Leftrightarrow a_j^q} \frac{-|\Delta AP_{ij}|}{1+\exp(f_{t-1}(a^q, a_i^q) - f_{t-1}(a^q, a_j^q))}$
10 $w_i^q = \frac{\partial y_i^q}{\partial f_{t-1}(a^q, a_i^q)}$
11 end
12 end
13 Obtain a L leaf tree from $\{\{a^q, a_i^q, y_i^q\}_1^{n_q}\}_1^Q$
14 Denoted as $\{R_l\}_{l=1}^L$, where each leaf value
15 is $\gamma_{lt} = \frac{\sum_{(a^q, a_i^q) \in R_{lt}} y_i^q}{\sum_{(a^q, a_i^q) \in R_{lt}} w_i^q}$
16 $f_t(a^q, a_i^q) = f_{t-1}(a^q, a_i^q) + \eta \sum_l \gamma_{lt} I((a^q, a_i^q) \in R_{lt})$
17 end

如算法 SimListMKL 所示, 第 1~5 行描述了模型的初始化过程, 第 6~17 行描述了每一个迭代步生成一个用来公式(6)中损失函数的梯度的回归树, 该树具有 L 个叶子节点, 叶子节点的值由 Newton-Raphon 更新步确定. 算法最终得到的 $f_N(a^q, a_i^q)$ 即为本文要求解的 App 相似性函数.

4 实验结果

为了验证 SimListMKL 算法的有效性, 本文首先介绍了实验设置. 接下来介绍了基于三元组相似性的多核学习算法与本文提出的基于列表相似性的多核学习算法, 在相似 App 推荐场景下的性能对比. 最后对比了各类算法性能受不同类型数据集的影响以及各类算法的运行时间.

4.1 实验设置

实验数据集采用的是 SimApp[1]中所用的训练数据集. 该数据集来源于 Google Play 的真实数据, 包括约 13,000 个查询 App, 每个查询 App 下约 20 个 App, 其中相似 App 的比例分布如图 1 所示. 横轴表示相似 App 的比例范围, 如 0.1 对应的是相似 App 比例在 0~0.1 之间的查询 App 的比例. App 之间的相似性由 10 个核函数来刻画, 包括名称, 类别, 描述, 开发商, 屏幕截图, 大小, 用户评分, 用户评论, 更新信息, 许可.

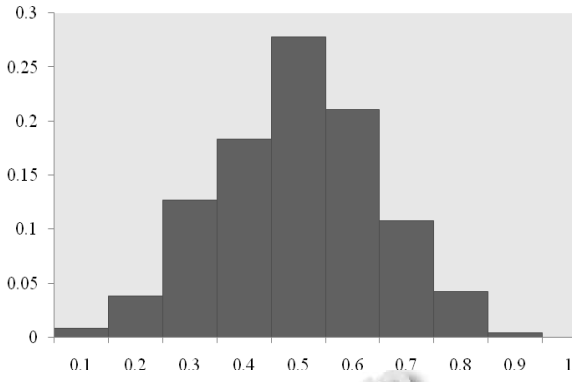

图 1 相似 App 比例分布

本文的主要基准算法是基于三元组的相似性学习算法, 包括 SimTripletNet, SimTripletBoost, SimTripletSVM. 尽管采用了相同的优化目标, 可是由于优化算法不同也有许多不同的变种, 如 SimTripletSVM 对应的 OKWL 算法与 OASIS 算法, 这里将它们统一从优化目标的角度命名. 基于交叉熵损失函数的 SimTripletNet 采用梯度下降优化算法求解, 基于指数函数损失的 SimTripletBoost 采用 boosting 优化算法求解, 基于 hinge 损失函数的 SimTripletSVM 采用切平面优化算法求解.

为了便于调参, 本文将该数据划分为 5 份, 采用 5 折交叉验证, 选取最优超参数, 即最后表中与图中所展示的结果. 所有方法的最大迭代次数均设置为 100 次. SimTripletNet 与 SimListMKL 中需要设置的学习步长从 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5} 中选取. SimTripletSVM 中的 C 从 10^3, 10^2, 10, 1, 10^{-1} 到 10^{-5} 中选取. 本文提出的

算法 SimListMKL 中的叶子数从 2 到 20 中选取. 实验采用的基于位置的评价指标是 Precision@1~10, 简记为 P@1~10 以及 MAP.

4.2 性能分析

本文将五折交叉验证的结果分别展示在表 1 和图 2 中. 表 1 描述了基于 P@1~10 的性能对比, 图 2 描述了 MAP 作为评价指标的性能对比. 基于 t-test 的假设检验证明以下表和图中的对比差异是显著的.

表 1 Precision 性能对比

	P@1	P@2	P@3	P@4	P@5
SimTripletBoost	0.9882	0.9684	0.9448	0.9176	0.88474
SimTripletNet	0.9809	0.9602	0.9382	0.9116	0.87966
SimTripletSVM	0.9873	0.9677	0.9452	0.9174	0.8854
SimListMKL	**0.9896**	**0.9703**	**0.9471**	**0.9190**	**0.88628**
	P@6	P@7	P@8	P@9	P@10
SimTripletBoost	0.8492	0.8122	0.7748	0.7379	0.7019
SimTripletNet	0.8444	0.8074	0.7707	0.7343	0.6988
SimTripletSVM	0.8499	0.8128	0.7754	0.7382	0.7020
SimListMKL	**0.8508**	**0.8137**	**0.77628**	**0.7393**	**0.7032**

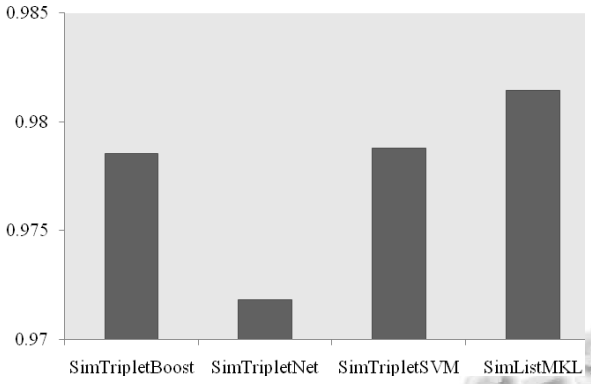

图 2 MAP 性能对比

表 1 的结果表明, 本文提出的 SimListMKL 方法普遍好于基于三元组的相似性学习方法. 以 P@5 为例, SimListMKL 算法优于最好的 SimTripletSVM 算法约 0.1%, 比最差的 SimTripletNet 算法约 0.7%. 因此, 从 Precision 的度量角度来看, 基于列表建模损失函数在面向排序的相似性学习场景中是较好的选择.

图 2 展示的结果可以直观的表明 SimListMKL 算法的优势. 该算法高出 SimTripletBoost 约 0.3%, 高出 SimTripletNet 约 1%, 高出 SimTripletSVM 约 0.3%. 因此, 从 MAP 的度量角度来看, 基于列表建模损失函数在面向排序的相似性学习场景中是较好的选择.

4.3 敏感分析

性能分析的结果显示, 算法在实验数据集上的性能普遍较好. 根据已有的排序学习算法性能受数据集影响的研究[14]表明, 这主要归因于训练数据中正负例的比例均衡.

换句话说, 即每个查询 App 对应的训练实例集合中, 相似 App 的数量占一半左右.

本实验试图打破这种均衡, 研究在不理想的数据集上算法性能的变化. ratio 表示每个查询 App 对应的待比较 App 集合中相似 App 的数量比例. 按这个比例抽取查询 App, 分别构建 ratio 小于 0.1, 0.2, 0.3, 0.4, 0.5 的数据集, 对比各算法在这些数据集上的性能表现, 如图 3 所示.

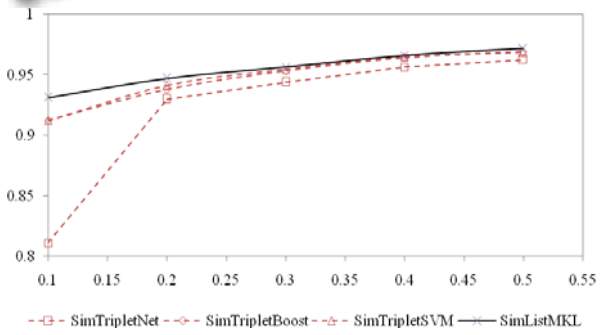

图 3 MAP 随 ratio 变化

图 3 展示了不同算法在 ratio 不同的数据集上 MAP 的变化曲线. SimListMKL 算法对数据集是最不敏感的, 其从 0.1 到 0.5, 仅变化了 4.4%. 而变化最大的 SimTripletNet 变化幅度为 18.6%. SimTripletBoost 与 SimTripletSVM 居中, 变化约为 6.2%. 由此可以看出, SimListMKL 是一个稳定的算法, 不容易受数据集好坏的影响.

4.4 运行时间分析

本部分实验对比了三元组的相似性学习算法与基于列表的相似性学习算法的训练时间. 图 4 展示了在 Win7-64 位操作系统, 4GB 内存, i7-2600 CPU, 主频 3.4GHZ 的环境下, 两组算法分别采用五折交叉验证时, 平均每折的运行时间(单位: 秒).

由图 4 可得, SimTripletSVM 运行时间最短约为 2.8 秒, 这一时间要远远好于同属于基于三元组的相似性学习算法的 SimTripletBoost(约 14.2 秒) 与 SimTripletNet(约 28 秒)算法. SimTripletSVM 算法的高效一部分原因在于采用的是 C 实现, 而其他所有算法

都采用的是 Java 实现.

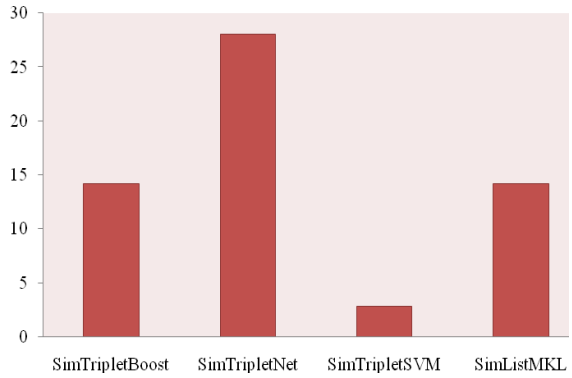

图 4 平均运行时间对比

本文提出的算法 SimListMKL 运行时间约为 14.2 秒, 与基于三元组的相似性学习算法 SimTripletBoost 相当, 远远好于 SimTripletNet. SimListMKL 与 SimTripletBoost 均采用了 Boosting 的优化算法, 而 SimTripletNet 采用了梯度下降算法. 可见, 影响运行时间的因素主要取决于优化算法的具体实现. 本文提出的 SimListMKL 优化目标虽然复杂, 并没有给优化算法带来太多额外的时间开销.

5 结语

针对排序场景中的相似性学习问题, 本文总结出了一个基于相对序的相似性学习框架, 并提出了基于列表的多核相似性学习算法 SimListMKL. 与已有的基于三元组相对相似性学习的算法相比, SimListMKL 算法采用了一个更为概括的形式, 即列表的相对相似性, 并基于此建模了损失函数, 并采用 MART 算法优化. 实验结果表明, SimListMKL 是一种更好的建模相对相似性的方式, 提升了相似 App 推荐的性能, 同时保证较高的时间效率. 多核相似性学习方法的显著问题是计算复杂度高, 下一步的研究工作主要是在线 SimListMKL 算法.

参考文献

1 Chen N, Hoi S C H, Li S, et al. SimApp: A framework for detecting similar mobile applications by online kernel learning. Proc. of the 8th ACM International Conference on Web Search and Data Mining. ACM. 2015. 305–314.

2 Chechik G, Sharma V, Shalit U. Large scale online learning of image similarity through ranking. Journal of Machine Learning Research, 2010: 1109–1135.

3 https://en.wikipedia.org/wiki/Similarity_learning#cite_note-4.

4 Guo G, Li S, Chan K. Support vector machines for face recognition. Image and Vision Computing, 2001, 19(9): 631–638.

5 Melacci S, Sarti L, Maggini M, et al. A neural network approach to similarity learning. IAPR Workshop on Artificial Neural Networks in Pattern Recognition. Springer Berlin Heidelberg. 2008. 133–136.

6 Xing EP, Ng AY, Jordan MI, et al. Distance metric learning with application to clustering with side-information. Advances in Neural Information Processing Systems, 2003, 15: 505–512.

7 Liu E Y, Zhang Z, Wang W. Clustering with relative constraints. Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. 2011. 947–955.

8 Liu E Y, Guo Z, Zhang X, et al. Metric learning from relative comparisons by minimizing squared residual. 2012 IEEE 12th International Conference on Data Mining. IEEE. 2012. 978–983.

9 Lanckriet GRG, Cristianini N, Bartlett P, et al. Learning the kernel matrix with semidefinite programming. Journal of Machine Learning Research, 2004, 5(Jan): 27–72.

10 Ying Y, Zhou DX. Learnability of Gaussians with flexible variances. Journal of Machine Learning Research, 2007, 8(Feb): 249–276.

11 Tang Y, Li L, Li X. Learning similarity with multikernel method. IEEE Trans. on Systems, Man, and Cybernetics, 2011, 41(1): 131–138.

12 Chen LB, Wang YN, Hu BG. Kernel-based similarity learning. Proc. 2002 International Conference on Machine Learning and Cybernetics. IEEE. 2002, 4. 2152–2156.

13 Liu TY. Learning to rank for information retrieval. Foundations and Trends in Information Retrieval, 2009, 3(3): 225–331.

14 Niu S, Lan Y, Guo J. Which noise affects algorithm robustness for learning to rank. Information Retrieval Journal, 2015, 18(3): 215–245.