哈爾濱工業大學

数据挖掘实验报告

题　　目　人岗智能匹配-以智联招聘为例

专　　业　计算机科学与技术

学　　号　19S103186

学　　生　赵朋飞

授 课 教 师　邹兆年

日　　期　2019.11.10

人岗智能匹配-以智联招聘为例

摘要

本文讨论一个关于人岗智能匹配的数据挖掘问题，根据智联招聘的求职者数据、职位信息、及部分求职者行为信息、用人单位反馈信息，训练模型，对求职者的职位候选集进行排序，尽可能使得双端都满意的职位（求职者满意以及用人单位满意）。在对数据集预处理，特征工程后，尝试使用基于梯度提升树（Gradient Boosting Decision Tree）算法的两种模型对其进行排序，最终较好的实现了求职者与岗位的匹配。为了达到更优的效果，利用两种模型的差异性，采用了不同的集成方式，集成两种模型。

关键词：人岗匹配、排序模型、梯度提升树、集成

1.问题描述

本文所解决的是第二届阿里巴巴大数据比赛的问题（https://tianchi.aliyun.com/competition/entrance/231728/information）。对于曝光给求职者的职位候选集里，假如求职者感兴趣会产生浏览职位行为，浏览职位后，如果求职者满意会产生主动投递行为。用人单位收到求职者的主动投递的简历后会给出是否满意的反馈信号。最终描述为解决两个问题，一是：预测用户会投递哪些岗位？二是：用户投递的哪些岗位会被认可？

预测到的用户投递岗位并且该投递岗位被用人单位认可，则该岗位将会获得较高的分数，在所有职位候选集中将会优先推荐给该用户。

2.评估指标

对于一个算法而言，评估算法的好坏，主要是由其在测试集上评估的分数决定，由问题描述中可以得知最终是解决两个问题，对于这两个问题的评估，将采用以下几种评估指标：

2.1 分类问题评估指标

训练得到的模型将会把职位候选集划分为两类，一个是被推荐的职位，一个是未被推荐的职位，由此可以使用常用分类评估指标对最后的结果进行评估。

2.2 排序问题评估指标

但是分类问题评估指标忽视了一个问题，它将推荐的职位等权重考虑，被推荐的每个职位将会赋予相同的分数，但实际应该是将最可能被推荐的职位排在最前面。这就需要采用排序问题评估指标。又由于涉及到两个问题的解决，所以采用加权的排序评估指标。以下将会详细叙述该指标：

测试数据由 n 组曝光的职位数据集组合，每组数据包含一个求职者和一序列曝光候选职位。需要对每组职位进行预测并排序给出排序后的职位序列。对 n 组排序后的职位序列。通过计算所有 n 组排序后的职位序列里，求职者投递（delivered）职位的 MAP 值以及用人单位中意（satisfied）职位的 MAP（Mean Average Precision），由最终的加权评价值：

$$MAP_{final} = MAP_{satisfied} * 0.7 + MAP_{delivered} * 0.3$$

作为评估指标，分数越高代表预测效果越好。

MAP 计算公式如下：

$$AP = \frac{\sum_{k=1}^{m}(P(k)*rel(k))}{COUNT(\text{正例})}$$

其中，m 为一个序列里结果条目数，k 为列表中的位置，P(k)为前 k 个结果的正例占比，rel(k)表示位置 k 的职位是否为正例，1 为正例，0 为负例。那么 MAP 则为：

$$MAP = \frac{\sum_{i=1}^{n} AP(\text{i})}{n}$$

3.数据分析

3. 1 数据集

该数据由阿里巴巴天池比赛数据平台获取，训练数据包含三个 csv 表格，分别是用户的简历描述表（table1_user）、岗位描述表（table2_job）、行为表（table3_action）（行为表表示的是当简历投放到相应的岗位后的行为），三张表的数据结构如下图所示：

用户标识	01234567890abcdef
居住城市	501
期望城市	601,701,801
期望行业	数据挖掘,生物学/植物学
期望职类	数据挖掘师,植物学家
期望薪水	70001100000
当前行业	生物学/植物学
当前职类	植物学家
当前薪水	5000170000
学历	本科
年龄	21
开始工作时间	201707
经验	竞赛|主宰|模型|植物分类|天池 |Lgb|10场|天才|无敌|女装|宗师|冠军|机器学习|统治|队友

职位标识	fedcba9876543210
标题	数据挖掘专家
城市	601
子类	数据挖掘工程师
需求人数	1
最低薪水	9999
最高薪水	99999
开始日期	20170101
结束日期	20190930
需要出差	1
工作年限	103
关键字	null
最低学历	本科
最高学历	
婚否	\N
语言需求	\N
描述	职责：负责利用数据设计牛逼的、有创新性的、有影响力的算法，并发表牛逼的、有创新性的、有影响力的论文。

浏览	1
投递	0
认可	0

图 1 训练数据中的三个 csv 表结构

为了简便期间，后面的字段均使用如下的英文代替：

用户标识	user_id
居住城市	live_city_id
期望城市	desire_jd_city_id
期望行业	desire_jd_industry_id
期望职类	desire_jd_type_id
期望薪水	desire_jd_salary_id
当前行业	cur_industry_id
当前职类	cur_jd_type
当前薪水	cur_salary_id
学历	cur_degree_id
年龄	birthday
开始工作时间	start_work_date
经验	experience

职位标识	jd_no
标题	jd_title
城市	city
子类	jd_sub_type
需求人数	require_nums
最低薪水	min_salary
最高薪水	max_salary
开始日期	start_date
结束日期	end_date
需要出差	is_travel
工作年限	min_years
关键字	key
最低学历	min_edu_level
最高学历	max_edu_level
婚否	is_mangerial
语言需求	resume_language_required
描述	job_description

图 2 中文字段对应的英文表示符

全部数据集中，table1_user 中有 4500 条用户数据，table2_job 中有 265690 条岗位数据，table3_action 中有 700938 条行为数据。最后将这三张表拼接得到一个总表，首先以 table3_action 为主表，"user_id"为索引拼接 table1_user 数据，拼接方式为"user_id"的交集。然后以拼接后的表为主表，"jd_no"为索引拼接 table2_job 数据，拼接方式为"jd_no"的交集。最终得到的数据（train_data）总计包含 506175 条数据。

3.2 数据预处理

a) 单一值、缺失值:

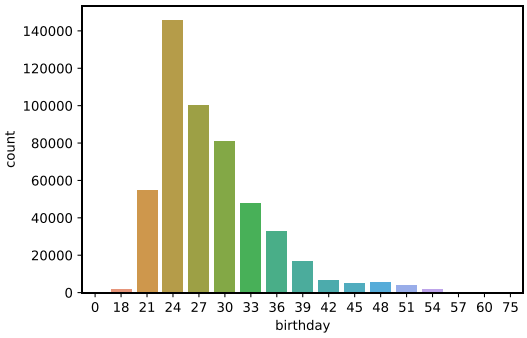
在数据集中有些字段，如"max_edu_level", "is_mangerial", "resume_language_required"全部数据中只存在‘\\N’数据，因此将这些字段剔除出去。然后存在一个字段"key"，在table2_job 表中，265690 条数据中，有 262634 条缺失数据，因此也予以剔除。

b) 数值、类别特征：

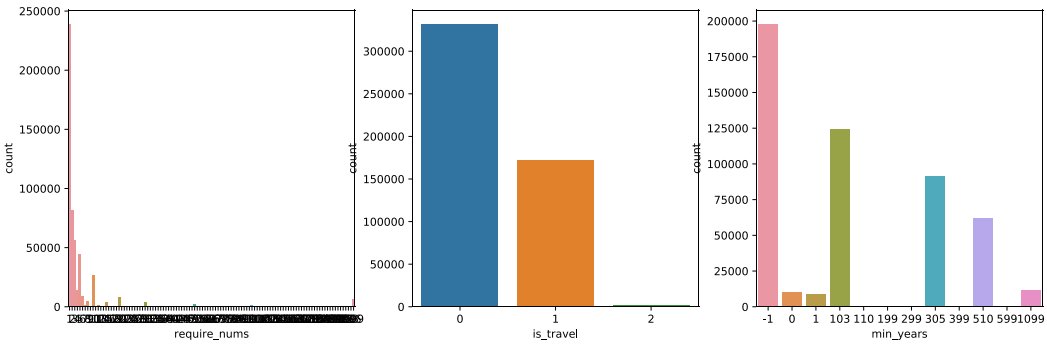
train_data 中的数据类型如图 3-1 所示：

```
Data columns (total 29 columns):
user_id                  506175 non-null object
jd_no                    506175 non-null object
browsed                  506175 non-null int64
delivered                506175 non-null int64
satisfied                506175 non-null int64
live_city_id             506175 non-null int64
desire_jd_city_id        506175 non-null object
desire_jd_industry_id    506175 non-null object
desire_jd_type_id        506175 non-null object
desire_jd_salary_id      506175 non-null int64
cur_industry_id          503176 non-null object
cur_jd_type              317760 non-null object
cur_salary_id            506175 non-null object
cur_degree_id            486690 non-null object
birthday                 506175 non-null int64
start_work_date          506175 non-null object
experience               503609 non-null object
jd_title                 506175 non-null object
city                     506175 non-null int64
jd_sub_type              506175 non-null object
require_nums             506175 non-null int64
max_salary               506175 non-null int64
min_salary               506175 non-null int64
start_date               506175 non-null object
end_date                 506175 non-null object
is_travel                506175 non-null int64
min_years                506175 non-null int64
min_edu_level            506175 non-null object
job_description          506175 non-null object
dtypes: int64(12), object(17)
```

3-1 train_data 数据类型



3-2 "birthday"分布



3-3 "require_nums", "is_travel", "min_years"分布

图 3 数值、类别特征分布

从上图可以看出，"birthday"直接使用属性值即可，但是"require_nums"存在 100 多种数值情况，并且大部分极端数值出现频率极低，对于这些离群点采用截断处理，设定阈值，超过该阈值则使用一个固定值代替，低于的使用实际值。"min_yeras"、"salary"字段采用了一种类似的编码方式，如"min_yeras"=103，代表 1~3 年，后续的特征工程将会具体介绍如何处理该种字段。

此外存在一种高基数类别特征，如"city_id","jd_type_id"中就有高达上百中类型，对于此类字段采用均值编码方式（Target Encoding）转为数值特征。均值编码是一种参考目标值编码的方式，2001 年由 Micci-Barreca D[1] 提出。对于C分类问题，均值编码后只需要增加$C-1$个属性列，如果C远远小于高基数的类别，则相对于 one-hot 编码可以节省很多内存。其出发点是用概率$P(y=y_i|x=x_i)$代替属性值x，其中x表述属性值，y 表示类别值。但实际问题中经常会遇到$x=x_i$对应的样本比较少，导致对$(y=y_i|x=x_i)$的计算不准确。引入先验概率$P(y=y_i)$，公式转换成：

$$f(y_i,x_i)=\lambda(n_i)P(y=y_i\mid x=x_i)+(1-\lambda(n_i))\,\mathrm{P}(y=y_i)$$

其中$j \in [1, C)$，n_i是训练集中x_i的样本个数，$\lambda(n_i) \in [0,1]$负责计算这两个概率的可靠性。

c) 文本信息：

数据集中的"cur_industry_id", "cur_jd_type", "experience", "jd_sub_type", "job_description", "jd_title"均为中文描述，如下图所示：

	cur_industry_id
0	房地产/建筑/建材/工程
1	电气/电力/水利
2	医疗/护理/美容/保健/卫生服务
3	保险
4	零售/批发
5	汽车/摩托车
6	广告/会展/公关
7	教育/培训/院校
8	银行
9	物业管理/商业中心

cur_jd_type
土木/建筑/装修/市政工程
其他
销售业务
NaN
房地产开发/经纪/中介
建筑设计师
实习生/培训生/储备干部
人力资源专员/助理
人力资源
助理/秘书/文员

experience
停车|现场|凤凰|预算编制|建设|实习|专家|公园|预算软件|勘察|合同|知识|商务|单位|...
责任心|适应能力|标书制作
协助|市政施工|现场|市政|施工员|机械|施工队长
分析|秘书|材料|电话|主动性|收银|店铺|销售经理|自我评价|销售代表|促销|装饰|客户|...
园林|市政工程|资料|施工|总工|路桥|质检|吃苦|主管|招投标|自我评价|装修|建筑工程|...
客服|销售业务|售前|交易|房地产销售|技术支持|置业顾问|售后|租赁|谈判|房地产中介
房屋|校园|营业员资格证|材料|cad|招生|服务员|office|收银|自控|sketch...
互联网参考模型osi七层|建设|处理|office|责任心|单位|登记|协调|客服|敬业|安...
活动|综合|员工|前台|接待|组织|人寿|考勤|招聘|采购|沟通能力|办公室|人寿保险|内容...
后勤人员|对账|街道|核对|手续|保证|成本|核算|宣传|办公室|销售代表|社会工作者|登记...

	jd_sub_type
0	工程造价/预结算
1	项目招投标
2	房地产项目招投标
3	施工员
4	工程资料管理
5	行政专员/助理
6	文档/资料管理
7	\N
8	其他
9	内勤人员

job_description
1、能够独立完成工程概、预、结算的编制及审核工作；2、熟悉重庆地区相关计价原则及计价文件；3...
岗位职责：1、根据招标方要求编制招标文件，负责报建等招标代理项目流程；2、负责组织现场答疑会...
岗位职责：1.协助设计师完成方案图纸或施工图纸的绘制；2.协助设计师负责现场施工维护；3.协...
岗位职责：1、项目投资分析，进行日常成本测算，提供设计变更成本建议；2、负责对设计估算、施工...
要求：工程造价专业毕业，能够经常出差，常驻工地。能吃苦，肯学习。熟练掌握CAD、广联达等软件。
一、职位要求1.大专以上学历，工程类专业，具备工程预、决算和工程管理的的专业知识2、有工程造...
一)岗位职责：1、负责组织、编制施工项目的施工图概算；2、负责施工班组及业主单位项目的竣工结...
一、岗位职责1、学习实践工程造价职业技能；2、听从负责人制定的工作安排，并积极与负责人沟通完...
岗位职责1、关注每天各大招投信息网站的更新内容，及时发现与本公司相匹配的招标公告与业务部门及...
岗位职责：1.协助项目经理做好工程开工的准备工作。2.协助认真审核工程所需材料，并对进场材料...

jd_title
土建工程造价员
招标代理人员
设计师助理
预算造价师
预算员造价员
土建/安装/市政造价预算员
预算员
实习造价（资料）员
招投标资料员
实习施工员

图 4 文本信息

由于"cur_industry_id", "cur_jd_type", "experience", "jd_sub_type"存在大量的重复数据，因此本文直接使用 wiki 百科以 Word2Vector 方法构建训练的 300 维的向量，然后使用 jieba 分词生成单词，最后对所有词进行平均生成最终的 300 维向量。对于"job_description", "jd_title"数据，是类似于文本的表述，所以用 Doc2Vector 方法构建一个 100 维的向量。Word2Vector 常采用图 5 所示的两种方法，一是根据周围的词预测中心词，二是根据中心词预测周围的词。Doc2Vector 类似于 Word2Vector 方法，此处不再赘述。

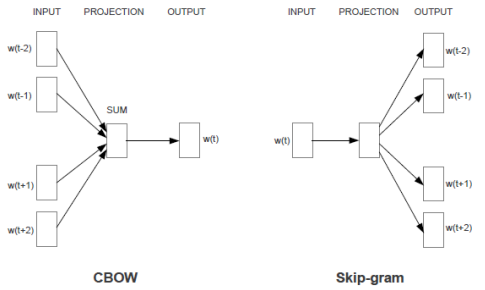


图 5 Word2Vector 方法

由于生成的文本信息字段词向量维度很大，因此为了减少特征维度，采用 SVD（奇异值分解）降维，将所有文本信息的向量降维至 10 维。因为 SVD 对于矩阵的分解不要求分解的矩阵是方阵，因此适用于该词向量的分解。

b) 数据清洗:

文本数据中提取学历信息，会发现 12%的学历和对应的学历字段值是不一致的（即职位

要求是本科，但是职位描述中会要求大专）。去除数据中的噪声，然后通过算法和模型学习其中的规律，才能将数据的价值最大化。

4.特征工程

4.1 基础特征

直接从数据上进行数值化，包括"desire_jd_sarlay_id"，"curr_jd_salary_id"，"curr_degree_id"，"max_salay"，"min_salay"等等。以及相关的判断，比如工作城市是否与期望的城市匹配等等，匹配值包括比值和差异值等等。下图表示对相关字段的数值，以及描述转为对应的数值特征：

```
min_year_dict = {
    -1:-1,
    0:0,
    1:1,
    103:1,
    110:1,
    199:1,
    299:2,
    305:3,
    399:3,
    510:5,
    599:5,
    1099:10
}

degree_dict = {
    "初中":1,
    "中技":2,
    "高中":3,
    "中专":3,
    "大专":4,
    "本科":5,
    "硕士":6,
    "博士":7,
    "EMBA":7,
    "MBA":6,
    "其他":0,
    "请选择":0,
    "\\N":0,
    "nan":0
}

min_salary_dict = {
    100002000:1000,
    400106000:4001,
    0:0,
    200104000:2001,
    600108000:6001,
    800110000:8001,
    1000115000:10001,
    2500199999:25001,
    1500125000:15001,
    3500150000:35001,
    70001100000:70001,
    1000:0,
    100001150000:100001,
    2500135000:25001,
    5000170000:50001
}

max_salary_dict = {
    100002000:2000,
    400106000:6000,
    0:0,
    200104000:4000,
    600108000:8000,
    800110000:10000,
    1000115000:15000,
    2500199999:99999,
    1500125000:25000,
    3500150000:50000,
    70001100000:100000,
    1000:1000,
    100001150000:150000,
    2500135000:35000,
    5000170000:70000
}
```

图 6 工作年限、教育程度、薪资对应的转换字典

在薪资要求字段中，需要按照上图转换为对应的数值，才能更好的表达信息，如 100002000 表示最低薪资为 1000 最高薪资为 2000，对于所有涉及薪资的都应按照上述的字典进行转换。工作经验、薪资、学历特征的构建如下图所示：

```
"""工作经验"""
df["work_years"] = 2019 - df["start_work_date"].apply(lambda x: 2019 if x == "-" else int(x))
df["min_years"] = df["min_years"].apply(lambda x: min_year_dict[x])
df["work_years_statisfied"] = df["work_years"].astype(int) > df["min_years"]

"""薪资"""
df["desire_min_salary"] = df["desire_jd_salary_id"].apply(lambda x: min_salary_dict[x])
df["desire_max_salary"] = df["desire_jd_salary_id"].apply(lambda x: max_salary_dict[x])
df["desire_salary_diff"] = df["desire_max_salary"] - df["desire_min_salary"]
df["cur_salary_min"] = df["cur_salary_id"].apply(lambda x: min_salary_dict[int(x if str.isnumeric(x) else "0")])
df["cur_salary_max"] = df["cur_salary_id"].apply(lambda x: max_salary_dict[int(x if str.isnumeric(x) else "0")])

df["cur_min_salary_larger_job_min"] = df["cur_salary_min"] > df["min_salary"]
df["cur_min_salary_larger_job_max"] = df["cur_salary_min"] > df["max_salary"]
df["cur_max_salary_larger_job_min"] = df["cur_salary_max"] > df["min_salary"]
df["cur_max_salary_larger_job_max"] = df["cur_salary_max"] > df["max_salary"]

df["desire_min_salary_larger_job_min"] = df["desire_min_salary"] > df["min_salary"]
df["desire_min_salary_larger_job_max"] = df["desire_min_salary"] > df["max_salary"]
df["desire_max_salary_larger_job_min"] = df["desire_max_salary"] > df["min_salary"]
df["desire_max_salary_larger_job_max"] = df["desire_max_salary"] > df["max_salary"]

"""学历"""
df["cur_degree_id"] = df["cur_degree_id"].fillna("nan").apply(lambda x: degree_dict[x.strip()])
df["min_edu_level"] = df["min_edu_level"].fillna("nan").apply(lambda x: degree_dict[x.strip()])
df["edu_satisfied"] = df["cur_degree_id"] >= df["min_edu_level"]
```

图 7 工作经验、薪资、学历特征的构建

城市，行业，工种的特征构建如下图所示：

```
"""行业"""
df["desire_jd_industry_id_1"] = df["desire_jd_industry_id"].apply(
    partial(extract_desire_jd_industry, index=0))
df["desire_jd_industry_id_2"] = df["desire_jd_industry_id"].apply(
    partial(extract_desire_jd_industry, index=1))
df["desire_jd_industry_id_3"] = df["desire_jd_industry_id"].apply(
    partial(extract_desire_jd_industry, index=2))
df["desire_jd_industry_num"] = np.sum(
    df[["desire_jd_industry_id_1", "desire_jd_industry_id_2", "desire_jd_industry_id_3"]] != -1, axis=1)
df["desire_jd_industry_real_num"] = df["desire_jd_industry_id"].apply(lambda x: len(x.split(",")))
```

```
"""工种"""
df["desire_jd_type_id_1"] = df["desire_jd_type_id"].apply(
    partial(extract_desire_jd_industry, index=0))
df["desire_jd_type_id_2"] = df["desire_jd_type_id"].apply(
    partial(extract_desire_jd_industry, index=1))
df["desire_jd_type_id_3"] = df["desire_jd_type_id"].apply(
    partial(extract_desire_jd_industry, index=2))
df["desire_jd_type_num"] = np.sum(
    df[["desire_jd_type_id_1", "desire_jd_type_id_2", "desire_jd_type_id_3"]] != -1, axis=1)
df["desire_jd_type_real_num"] = df["desire_jd_type_id"].apply(lambda x: len(x.split(",")))
```

```
"""城市特征"""
df["desire_jd_city_1"] = df["desire_jd_city_id"].apply(partial(extract_city, index=0))
df["desire_jd_city_2"] = df["desire_jd_city_id"].apply(partial(extract_city, index=1))
df["desire_jd_city_3"] = df["desire_jd_city_id"].apply(partial(extract_city, index=2))
df["desire_jd_city_nums"] = df["desire_jd_city_id"].apply(city_nums)
df["live_city_in_desire"] = df.apply(
    lambda row: city_in_desire(row["live_city_id"], row["desire_jd_city_id"]), axis=1)
df["job_live_city_in_desire"] = df.apply(
    lambda row: city_in_desire(row["city"], row["desire_jd_city_id"]), axis=1)
```

图 8 城市、行业、工种特征的构建

日期特征的构建：

```
"""日期"""
df["start_date"][df["start_date"] == "\\N"] = "20190109"
df["end_date"][df["end_date"] == "\\N"] = "20190209"
curr = pd.to_datetime(df["start_date"].astype(str))
end = pd.to_datetime(df["end_date"].astype(str))
df['continue_days'] = (end - curr).values / np.timedelta64(1, 'D')
```

图 9 日期特征的构建

有些数值特征很大，如薪资等，这就需要对这些特征进行转换，如下图所示：

```
#特征转换
df["continue_days_log"] = df["continue_days"].apply(lambda x:  np.log(x))
df["cur_salary_min"] = df["cur_salary_min"].apply(lambda x: np.round(x/1000))
df["cur_salary_max"] = df["cur_salary_max"].apply(lambda x: np.round(x/1000))
df["desire_max_salary"] = df["desire_max_salary"].apply(lambda x: np.round(x/1000))
df["desire_min_salary"] = df["desire_min_salary"].apply(lambda x: np.round(x/1000))
df["desire_salary_diff"] = df["desire_salary_diff"].apply(lambda x: np.round(x/1000))
df["max_salary"] = df["max_salary"].apply(lambda x: np.round(x/1000))
df["min_salary"] = df["min_salary"].apply(lambda x: np.round(x/1000))
df["require_nums"] = df["require_nums"].apply(lambda x: x if x <= 100 else 100)
```

图 10 特征转换

4.2 交叉统计特征

因为每个工作的"browsed"，"delivered"，"satisfied"的均值，方差等统计特征有很大的差异性。"satisfied"比率高的工作，处理新用户时，将有很大可能被推送，但是如果直接统计全部训练数据集的每个工作的这些字段的统计特征，然后再重新赋予训练集，将会造成数据泄露，因为特征中将包含标签信息。这对后续的模型训练而言，将会是一个巨大的灾难！

因此，为了有效利用该特征，将数据分为七折，以其中的六折数据统计上述字段信息，然后将统计后的信息赋予另外一折，这样就可以避免数据泄露问题。七折的选取，一是为了减少运算的压力，另一个是为了较好的代表整体数据的分布。

本文对统计信息主要采用两个，sum，mean 交叉统计特征的构建如下所示：

```
#五折交叉统计特征
folds = KFold(n_splits=7, shuffle=True, random_state=2019)
kfold_features = set()
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train_data, train_data["satisfied"])):
    data_trn = train_data.iloc[trn_idx, :]
    data_val = train_data.iloc[val_idx, :]

    for feat in ["jd_no"]:
        nums_columns = ["browsed", "delivered", "satisfied"]
        for f in nums_columns:
            statisticals = ["mean", "sum"]
            for statistical in statisticals:
                colname = feat + '_' + f + '_kfold_{}'.format(statistical)
                kfold_features.add(colname)
                if statistical == "mean":
                    order_label = data_trn.groupby([feat])[f].mean()
                elif statistical == "sum":
                    order_label = data_trn.groupby([feat])[f].sum()
                train_data.loc[val_idx, colname] = data_val[feat].map(order_label)
```

图 11 交叉统计特征的构建

4.3 转化率特征

转化率特征主要构建三种，分别是浏览后投递率，浏览后满意率，投递后满意率，构建如下所示：

```
def construct_ratio(x, y):
    if np.isnan(x) == False and np.isnan(y) == False:
        return x/(y+1e-8)
    else:
        return np.nan

#转化率特征
train_data["delivered_divided_browsed"] = train_data.apply(lambda row:
                                                           construct_ratio(
                                                           row["jd_no_delivered_kfold_sum"],
                                                           row["jd_no_browsed_kfold_sum"]), axis=1)
train_data["satisfied_divided_browsed"] = train_data.apply(lambda row:
                                                           construct_ratio(
                                                           row["jd_no_satisfied_kfold_sum"],
                                                           row["jd_no_browsed_kfold_sum"]), axis=1)
train_data["satisfied_divided_delivered"] = train_data.apply(lambda row:
                                                           construct_ratio(
                                                           row["jd_no_satisfied_kfold_sum"],
                                                           row["jd_no_delivered_kfold_sum"]), axis=1)
```

5.模型构建

排序是对一组物品列表按照某种方式进行排序，来最大化整个列表的效用过程，广泛应用于搜索引擎、推荐系统、机器翻译、对话系统甚至计算生物学。一些监督机器学习技术经常被广泛应用在这些问题中，这些技术称作排序学习技术。接下来将简要介绍该领域的相关工作，然后介绍在智联招聘问题中所使用的模型。

5.1 相关工作

排序学习（Learning to Rank，LTR）最早兴起于信息检索领域。经典的信息检索模型包括布尔模型、向量空间模型 、 概率模型、语言模型以及链接分析等。这些在不同时期提出的模型都是无监督排序方法，其共同特点是利用一些简单的特征进行排序，例如词频、逆文档频率等。这些传统排序方法的优点在于容易进行经验参数的调整，得到最优的参数，用以对检索文档按照一定标准(往往是查询与文档之间的相关性)进行排序。

随着搜索引擎需要处理的数据量呈指数增长，人为凭经验优化参数的过程变得越来越复

杂。这些经典的模型往往偏重某一方面的因素而忽略了其他可以用于排序的重要因素，例如概率模型和语言模型都没有考虑网页链接、网页 pagerank 值等互联网结构对排序的影响。在这种情况下，排序学习应运而生。排序学习的定义为：基于机器学习中用于解决分类与回归问题的思想，提出利用机器学习方法解决排序的问题。排序学习的目标在于自动地从训练数据中学习得到一个排序函数，使其在文本检索中能够针对文本的相关性、重要性等衡量标准对文本进行排序。

排序学习方法按照输入样例的不同，一般可分为 3 类：点级(pointwise)、对级(pairwise)、列表级(listwise)。点级方法输入包括用户、物品等特征，根据其对待排序的不同方式划分为回归和分类两种方法，它们分别将排序问题退化为回归与分类问题求解；对级方法考虑物品对之间的偏序关系，将问题转换为有序分类问题，更接近排序问题的实质。列表级方法输入所有相关联的物品集合，更加全面地考虑了不同物品的序列关系，因而成为近年被研究较多的方法。

Pointwise 方式中，单个物品对于某个用户而言，要么是正样本、要么是负样本。对于正样本，用户实际的反馈操作包括点击、浏览、收藏、下单、支付等，不同操作反映了该物品对用户需求的不同匹配程度，对应的样本可以看做正样本，且赋予不同的权重。对于负样本，为了模拟用户的真实浏览行为(从上往下)，可以采取 Skip Above 策略，即用户点过的物品之上，采样没有点过的物品作为负样本。Pairwise 方式中，主要构造物品 pair 对，对单个用户而言，没有绝对的偏好，只有对某个物品的相对偏好(e.g., 比起 B，用户更喜欢 A)。Listwise 方式中，会考察列表级别的排序。例如，列表的 NDCG 指标融入到目标函数中，目标函数能够优化列表级别的排序指标。

接下来主要介绍几种常用的排序模型：

a) 逻辑回归模型（Logistic Regression）:

LR 模型，用于建模 Pointwise 方式的数据。解释性强，并且通过权重可以解释推荐的内容，找到模型的不足之处。实际优化过程中，会给样本添加不同权重，例如根据反馈的类型点击、收藏、下单、支付等，依次提高权重，优化如下带权重的损失：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} w_i \cdot (y_i \log(\sigma_\theta(x_i)) + (1-y_i)\log(1-\sigma_\theta(x_i)))$$

其中w_i是样本的权重，y_i是样本的标签。另外，LR 是个线性分类模型，要求输入是线性独立的特征。我们使用的稠密的特征（维度在几十到几百之间）往往都是非线性的，并且具有依赖性，因此需要对特征进行转换。虽然 LR 模型很简单，解释性强，不过在特征特别多的情况下，劣势是显而易见的。

b) 梯度提升树（Gradient Boosting Decision Tree）:

GBDT 是一种迭代的决策树算法，该算法由多个基分类器组成，所有基分类器的结果累加起来做预测。多个基分类器的联合表达形式如下所示：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F$$

其中$f_k(x_i)$是第 k 个基分类器，x_i是输入特征，$\hat{y}_i$是预测结果。最终模型训练的目标是：

$$obj(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

其中$\Omega(f_k)$表示的是模型的复杂度。在训练第 t 个模型时，则：

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

则目标函数可以重写为：

$$obj^t = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$
$$= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const$$

对上式一阶泰勒展开后为：

$$obj^t = \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + const$$
$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}), \quad h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

移除目标函数当中所有的常量后，最终的优化目标为：

$$obj^t = \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

GBDT 一般采用决策树作为基分类器，FaceBook 2014 年的文章[2]介绍了通过 GBDT 解决 LR 的特征组合问题。核心思想是利用 GBDT 训练好多的多棵决策树，某个样本从根节点开始遍历，可以落到不同树的不同叶子节点中，将叶子节点的编号作为 GBDT 提取到的高阶特征，该特征将作为 LR 模型的输入。

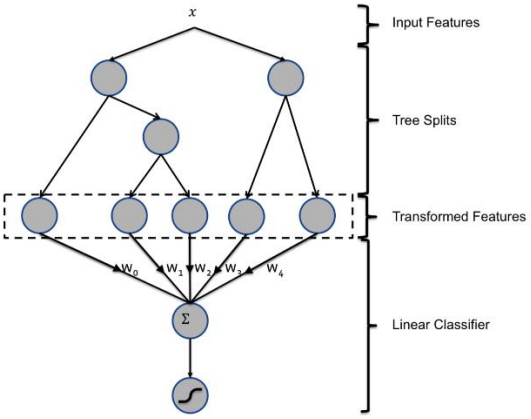


图 12 GBDT+LR

c) LambdaMART:

LambdaMART[3]由微软于 2010 年提出，主要是基于 LambdaRank 算法和 MART (Multiple Additive Regression Tree) 算法，将排序问题转化为回归决策树问题。MART 其实就是 GBDT 算法，GBDT 的核心思想是在不断的迭代中，新一轮迭代产生的回归决策树模型拟合损失函数的梯度，最终将所有的回归决策树叠加得到最终的模型。LambdaMART 使用一个特殊的 Lambda 值来代替上述梯度，也就是将 LambdaRank 算法与 MART 算法结合起来，是一种能够支持非平滑损失函数的学习排序算法。

常见的排序评价指标都无法求梯度，因此没法直接对评价指标做梯度下降，因此将他们转化为对偏序概率的交叉熵损失函数的优化问题，从而适用梯度下降方法，偏序概率的定义如下：

$$P_{ij} = P(x_i \rhd x_j) = \frac{1}{1+\exp(-\sigma \cdot (s_i - s_j))}$$

$$s_i = f(x_i; w),\ \ s_j = f(x_j; w)$$

其中s_i为x_i的得分，上述偏序函数表示x_i得分大于x_j得分的偏序概率。再定义交叉熵损失函数：

$$\begin{aligned}
L_{ij} &= -\bar{P}_{ij} \log P_{ij} - (1-\bar{P}_{ij}) \log(1-P_{ij}) \\
&= -\frac{1}{2}(1+S_{ij}) \log P_{ij} - \frac{1}{2}(1-S_{ij}) \log(1-P_{ij}) \\
&= -\frac{1}{2} S_{ij} (\log P_{ij} - \log(1-P_{ij})) - \frac{1}{2}(\log P_{ij} + \log(1-P_{ij})) \\
&= -\frac{1}{2} S_{ij} (\log \frac{P_{ij}}{1-P_{ij}}) - \frac{1}{2}(\log P_{ij}(1-P_{ij})) \\
&= -\frac{1}{2} S_{ij} \cdot \sigma \cdot (s_i - s_j) - \frac{1}{2} \log(\frac{\partial P_{ij}}{\partial(\sigma \cdot (s_i - s_j))}) \\
&= \frac{1}{2}(1-S_{ij}) \cdot \sigma \cdot (s_i - s_j) + \log\left\{1+\exp(-\sigma \cdot (s_i - s_j))\right\}
\end{aligned}$$

上式是当个样本的损失，$S_{ij} = \{1, -1, 0\}$是标签，$\bar{P}_{ij}$是物品 i 比 j 排序靠前的真实概率，即$i \rhd j\ (S_{ij} = 1)$，则为 1，$i \rhd j\ (S_{ij} = -1)$为 0，否则为 1/2。因此$\bar{P}_{ij} = (S_{ij} + 1)/2$。最后可以上述 loss 进行梯度下降，其中算法函数$f(x, w)$可以取任何函数，如 GBDT、DNN、LR 等。

接下来以一张图来表述上述的作用过程：

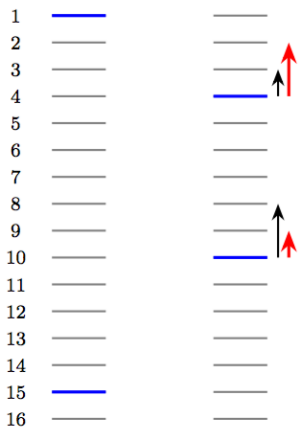


图 12 排序表示

这里每条横线代表一个物品，其中蓝色的表示相关的物品，灰色的则表示不相关的物品。在某次迭代中，物品的顺序从左边变成了右边。于是我们可以看到：

I. 梯度下降表现在结果的整体变化中是逆序对的下降。左图中，2~14 不相关物品都排在了 15 号相关物品之前，这些不相关物品和 15 号物品构成的逆序对共 13 个，因此损失等价于为 13；而右图中，将 1 号相关物品降到了 4 号，15 号相关物品上升到了 10 号，此时逆序对的数量为 3+8=11 个，因此损失等价于 11。

II. 对于一些强调最靠前的 TopN 个物品的排序指标(NDCG、ERR 等)而言，上述优化不是理想的。例如，右图下一次迭代，在 Ranknet 中梯度优化方向如黑色箭头所示，此时损失

可以下降到 8；然而对于 NDCG 指标而言，我们更愿意看到红色箭头所示的优化方向（此时 Ranknet 同样是 8，但是 NDCG 指标相比前一种情况上升了）。

此时 Lambda 梯度登场，它不是通过显示定义损失函数再求梯度的方式对排序问题进行求解，而是分析排序问题需要的分数 s 的梯度的物理含义，直接定义梯度，即 Lambda 梯度。只希望 L 对 s 的梯度可以按照理想 NDCG 指标的方向走。

其中上述的梯度表述，总体表述如下：

$$\frac{\partial L}{\partial w_k}=\sum_{(i,j)\in P}\frac{\partial L_{ij}}{\partial w_k}=\sum_{(i,j)\in P}\frac{\partial L_{ij}}{\partial s_i}\frac{\partial s_i}{\partial w_k}+\frac{\partial L_{ij}}{\partial s_j}\frac{\partial s_j}{\partial w_k}$$

并且，存在以下关系：

$$\frac{\partial L_{ij}}{\partial s_i}=\sigma\bullet\left[\frac{1}{2}(1-S_{ij})-\frac{1}{1+\exp(\sigma\bullet(s_i-s_j))}\right]=-\frac{\partial L_{ij}}{\partial s_i}$$

当$S_{ij}=1$时，于是有简化：

$$\lambda_{ij}=-\frac{\sigma}{1+\exp(\sigma\bullet(s_i-s_j))}$$

对于$S_{ij}=1$，希望λ_{ij}越大越好。因此，此时的目标是最大化某个效用函数 C。此时就需要梯度提升。关于 LambdaMART 的算法不再具体介绍，可以参见 CIKM2018 的论文[4]。

5.2 本文的模型

本文主要采用两个模型，一个是 Pointwise 方式的 LR 模型（基于 GBDT）,该模型以下称为回归模型，一个是 Listwise 方式的 LambdaMART 模型（基于 GBDT），该模型以下称为排序模型，两种模型实现方式都采用 LightGBM[5]。它是个快速的、分布式的、高性能的基于决策树算法的 GBDT 框架。可用于排序、分类、回归以及很多其他的机器学习任务中。

在数据挖掘任务中，XGBoost[6]算法非常热门，它是一种优秀的 GBDT 框架，但是在如今的大样本和高维度的环境下，传统的 boosting 似乎在效率和可扩展性上不能满足现在的需求了，主要的原因就是传统的 boosting 算法需要对每一个特征都要扫描所有的样本点来选择最好的切分点，这是非常耗时的。为了解决这种大样本高维度数据下的耗时问题，LightGBM 使用了如下两种解决办法：一是基于梯度的单边采样（Gradient-based One-Side Sampling），不是使用所有样本点来计算梯度，而是对样本进行采样来计算梯度；二是互斥特征捆绑（Exclusive Feature Bundling），这里不是使用所有的特征来进行扫描来获得最佳的切分点，而是将某些特征捆绑在一起来降低特征的维度，使寻找最佳切分点的消耗减少。这样大大的降低处理样本的时间复杂度，但在精度上，通过实验证明，在某些数据集上使用 Lightgbm 并不损失精度，甚至有时还会提升精度。

a) 排序模型：

首先将数据根据"user_id"划分为五折，这样可以保证同一个用户的所有求职信息都会划分到对应的训练集或者验证集中，也就是说一个用户的信息不会即出现在训练集中又出现在验证集中，避免了数据的泄露。

LGBMRanker 是 LGBMModel 的子任务，它用于 ranking 任务。其中主要参数如下所示：

```
class lightgbm.LGBMRanker(boosting_type='gbdt', num_leaves=31, max_depth=-1,
   learning_rate=0.1, n_estimators=10, max_bin=255, subsample_for_bin=200000,
   objective=None, min_split_gain=0.0, min_child_weight=0.001, min_child_samples=20,
   subsample=1.0, subsample_freq=1, colsample_bytree=1.0, reg_alpha=0.0,
   reg_lambda=0.0, random_state=None, n_jobs=-1, silent=True, **kwargs)
```

以下将简短的介绍上述参数的作用：

boosting_type: 一个字符串，指定了基学习器的算法，默认为"gbdt"

num_leaves: 一个整数，给出一棵树上叶子节点的个数，默认为 31

max_depth: 一个整数，限制了树模型的最大深度，默认值为-1

learning_rate: 一个浮点数，给出学习率，默认值为-1

n_estimators: 一个整数，给出了 boosted trees 的数量，默认为 10

min_child_samples: 一个整数，表示一个叶子节点上包含的最小样本数量，默认值是 20

min_child_gain: 一个浮点数，表示执行切分的最小增益，默认为 0

subsample: 一个浮点数，表示训练样本的采样比例

colsample_bytree: 一个浮点数，表示特征的采样比例

reg_alpha: 一个浮点数，表示 L1 正则化系数，默认为 0

reg_lambda: 一个浮点数，表示 L2 正则化系数，默认为 0

eval_at：一个整数列表，给出了 metric 的 evaluation position

简短的介绍模型的初始化参数，对于模型的训练参数如下：

```
fit(X, y, sample_weight=None, init_score=None, group=None, eval_set=None,
    eval_names=None, eval_sample_weight=None, eval_init_score=None,
    eval_group=None, eval_metric='ndcg', eval_at=[1], early_stopping_rounds=None,
    verbose=True, feature_name='auto', categorical_feature='auto', callbacks=None)
```

x: 一个 array_like 对象，其形状为（n_samples, n_features）表示训练样本集

y: 一个 array_like 对象，其形状为（n_samples, ），给出了标签值

group: 给出了每个训练样本的分组

eval_set: 给出验证集，用于早停

eval_group：给出每个验证集中，每个样本的分组

early_stopping_rounds：早停轮数

verbose：每隔多少轮，输出 evaluation 信息

最终构建的模型训练训练代码如下所示：

```
for idx, (train_idx, valid_idx) in enumerate(folds.split(train_users)):
    t_user = train_users[train_idx]
    v_user = train_users[valid_idx]

    x_train = train_data[train_data["user_id"].isin(t_user)]
    x_valid = train_data[train_data["user_id"].isin(v_user)]
    x_train.sort_values("user_id",inplace=True)
    x_valid.sort_values("user_id",inplace=True)

    curr_oof = x_valid[["user_id", "jd_no", "delivered", "satisfied"]]

    g_train = x_train.groupby("user_id",as_index=False).count()["satisfied"].values
    g_valid = x_valid.groupby("user_id",as_index=False).count()["satisfied"].values


    result = lbl_emb(target, x_train, x_valid, test_data)
    t_x,t_y = result[0]
    v_x,v_y = result[1]
    test_x,_ = result[2]

    model = lgb.LGBMRanker(**param)
    print("Fold",idx,"-"*30)
    model.fit(t_x,t_y,group=g_train,
              eval_set=[(t_x, t_y),(v_x,v_y)],
              eval_group=[g_train, g_valid],
              early_stopping_rounds=100,
              verbose=50,
              eval_at=[200]
             )
```

图 13 模型训练

训练参数如下所示（参数调优采用网格搜索，耗时极长，具体细节此处不再赘述）：

```
ranker(param=dict(n_estimators=3000,metric="map",learning_rate=0.1,num_leaves=31,
 max_depth=7,min_data_in_leaf=20, min_sum_hessian_in_leaf=0.002,
 reg_alpha=0.5,reg_lambda=1,colsample_bytree=0.8,subsample=0.8,
 device='gpu',gpu_platform_id=1,gpu_device_id=0)
  ,n_folds=5,target="target")
```

训练采用 mAP 的 metric，其中 target 是根据"satisfied"和"delivered"转化而来的目标，当"satisfied"=1 且"delivered"=1 时，"target"=3；当"satisfied"=0 且"delivered"=1 时，"target"=1；当"satisfied"=0 且"delivered"=0 时，"target"=0。

具体表达形式上面已经叙述，训练的过程中的日志信息如下所示：

```
Fold 0 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's map@200: 0.264371    valid_1's map@200: 0.250926
[100]   training's map@200: 0.281211    valid_1's map@200: 0.256072
[150]   training's map@200: 0.293513    valid_1's map@200: 0.25857
[200]   training's map@200: 0.303797    valid_1's map@200: 0.261193
[250]   training's map@200: 0.311442    valid_1's map@200: 0.264324
[300]   training's map@200: 0.319948    valid_1's map@200: 0.263448
[350]   training's map@200: 0.325617    valid_1's map@200: 0.264786
[400]   training's map@200: 0.331224    valid_1's map@200: 0.265055
[450]   training's map@200: 0.336588    valid_1's map@200: 0.267598
[500]   training's map@200: 0.341668    valid_1's map@200: 0.267945
[550]   training's map@200: 0.346901    valid_1's map@200: 0.267749
[600]   training's map@200: 0.353237    valid_1's map@200: 0.268997
[650]   training's map@200: 0.358361    valid_1's map@200: 0.269362
[700]   training's map@200: 0.364514    valid_1's map@200: 0.271672
[750]   training's map@200: 0.369229    valid_1's map@200: 0.271416
[800]   training's map@200: 0.373839    valid_1's map@200: 0.271466
Early stopping, best iteration is:
[701]   training's map@200: 0.364577    valid_1's map@200: 0.271713
Fold 1 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's map@200: 0.26788     valid_1's map@200: 0.242666
[100]   training's map@200: 0.28402     valid_1's map@200: 0.245277
[150]   training's map@200: 0.296436    valid_1's map@200: 0.248883
[200]   training's map@200: 0.307798    valid_1's map@200: 0.251852
[250]   training's map@200: 0.315596    valid_1's map@200: 0.25019
[300]   training's map@200: 0.323171    valid_1's map@200: 0.250893
Early stopping, best iteration is:
[200]   training's map@200: 0.307798    valid_1's map@200: 0.251852
Fold 2 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's map@200: 0.263379    valid_1's map@200: 0.257354
[100]   training's map@200: 0.278846    valid_1's map@200: 0.262823
[150]   training's map@200: 0.291801    valid_1's map@200: 0.263216
[200]   training's map@200: 0.302351    valid_1's map@200: 0.26367
[250]   training's map@200: 0.312162    valid_1's map@200: 0.265146
[300]   training's map@200: 0.319199    valid_1's map@200: 0.265662
[350]   training's map@200: 0.327157    valid_1's map@200: 0.266882
[400]   training's map@200: 0.335411    valid_1's map@200: 0.267173
[450]   training's map@200: 0.341474    valid_1's map@200: 0.268016
[500]   training's map@200: 0.347597    valid_1's map@200: 0.26848
[550]   training's map@200: 0.35243     valid_1's map@200: 0.269301
[600]   training's map@200: 0.356711    valid_1's map@200: 0.270224
[650]   training's map@200: 0.362602    valid_1's map@200: 0.270608
[700]   training's map@200: 0.367235    valid_1's map@200: 0.272577
[750]   training's map@200: 0.371914    valid_1's map@200: 0.275248
[800]   training's map@200: 0.376309    valid_1's map@200: 0.276318
[850]   training's map@200: 0.381294    valid_1's map@200: 0.276817
[900]   training's map@200: 0.386658    valid_1's map@200: 0.276611
Early stopping, best iteration is:
[832]   training's map@200: 0.379552    valid_1's map@200: 0.277073
Fold 3 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's map@200: 0.268965    valid_1's map@200: 0.236049
[100]   training's map@200: 0.286075    valid_1's map@200: 0.239154
[150]   training's map@200: 0.298777    valid_1's map@200: 0.243216
[200]   training's map@200: 0.308643    valid_1's map@200: 0.245576
[250]   training's map@200: 0.317409    valid_1's map@200: 0.248192
[300]   training's map@200: 0.324283    valid_1's map@200: 0.251196
[350]   training's map@200: 0.329783    valid_1's map@200: 0.25076
[400]   training's map@200: 0.337688    valid_1's map@200: 0.251329
[450]   training's map@200: 0.345299    valid_1's map@200: 0.252462
[500]   training's map@200: 0.351049    valid_1's map@200: 0.253022
[550]   training's map@200: 0.357314    valid_1's map@200: 0.254521
[600]   training's map@200: 0.362869    valid_1's map@200: 0.255853
[650]   training's map@200: 0.367907    valid_1's map@200: 0.255996
[700]   training's map@200: 0.37384     valid_1's map@200: 0.256636
[750]   training's map@200: 0.379771    valid_1's map@200: 0.257275
[800]   training's map@200: 0.385638    valid_1's map@200: 0.258714
[850]   training's map@200: 0.390988    valid_1's map@200: 0.259832
[900]   training's map@200: 0.39522     valid_1's map@200: 0.261204
[950]   training's map@200: 0.399058    valid_1's map@200: 0.262861
[1000]  training's map@200: 0.403978    valid_1's map@200: 0.263093
[1050]  training's map@200: 0.409329    valid_1's map@200: 0.264142
[1100]  training's map@200: 0.41359     valid_1's map@200: 0.264623
[1150]  training's map@200: 0.418316    valid_1's map@200: 0.26404
Early stopping, best iteration is:
[1063]  training's map@200: 0.410674    valid_1's map@200: 0.265273
Fold 4 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's map@200: 0.265639    valid_1's map@200: 0.248048
[100]   training's map@200: 0.282005    valid_1's map@200: 0.253179
[150]   training's map@200: 0.294784    valid_1's map@200: 0.254907
[200]   training's map@200: 0.305225    valid_1's map@200: 0.256133
[250]   training's map@200: 0.31562     valid_1's map@200: 0.258006
[300]   training's map@200: 0.323585    valid_1's map@200: 0.258173
[350]   training's map@200: 0.329879    valid_1's map@200: 0.25982
[400]   training's map@200: 0.337641    valid_1's map@200: 0.260089
[450]   training's map@200: 0.343542    valid_1's map@200: 0.260776
[500]   training's map@200: 0.349603    valid_1's map@200: 0.261288
[550]   training's map@200: 0.355382    valid_1's map@200: 0.263361
[600]   training's map@200: 0.361042    valid_1's map@200: 0.26406
[650]   training's map@200: 0.366167    valid_1's map@200: 0.264216
[700]   training's map@200: 0.371091    valid_1's map@200: 0.263709
Early stopping, best iteration is:
[606]   training's map@200: 0.361637    valid_1's map@200: 0.264296
mean score 0.2660414089053319
satisfiedMAP 0.2784286249795223
deliveredMAP 0.26414054658256764
0.2741422014604359
```

图 14 训练日志数据

根据以上日志信息，其中 mAP 分数对前 200 个进行排序然后计算指标，satisfiedMAP 对 satisfied 目标进行评估得分 0.2784，deliveredMAP 对 delivered 目标进行评估得分 0.2641，最后的评估分数为两者的加权分数 0.2741，接下来简要的说明这些指标的运算：

假设一个序列 1 有 10 个职位，其中 3 个职位被投递，最终排序这 3 个职位的 rank 为 1，3，5 则投递$AP_{delivered1} = \frac{\frac{1}{1}+\frac{2}{3}+\frac{3}{5}}{3} = 0.756$；序列 2 有 20 个职位，其中五个职位被投递，最终的排序 rank 为 1，2，3，4，5，则$AP_{delivered2} = 1$，所以$MAP_{delivered} = \frac{0.756+1}{2} = 0.878$。其中序列 1 有一个职位被 HR 中意，rank 为 3，则$AP_{satisfied1} = \frac{\frac{1}{3}}{1} = 0.333$；序列 2 中有两个职位被 HR 中意，rank 为 1 和 4，则$AP_{satisfied2} = \frac{\frac{1}{1}+\frac{2}{4}}{2} = 0.75$，所以$MAP_{satisfied} = \frac{0.333+0.75}{2} = 0.542$。最终，$MAP_{final} = MAP_{satisfied} * 0.7 + MAP_{delivered} * 0.3 = 0.642$。

最后对所有的特征重要性进行排序显示如下（由于有多达上百维的特征，仅部分显示）：

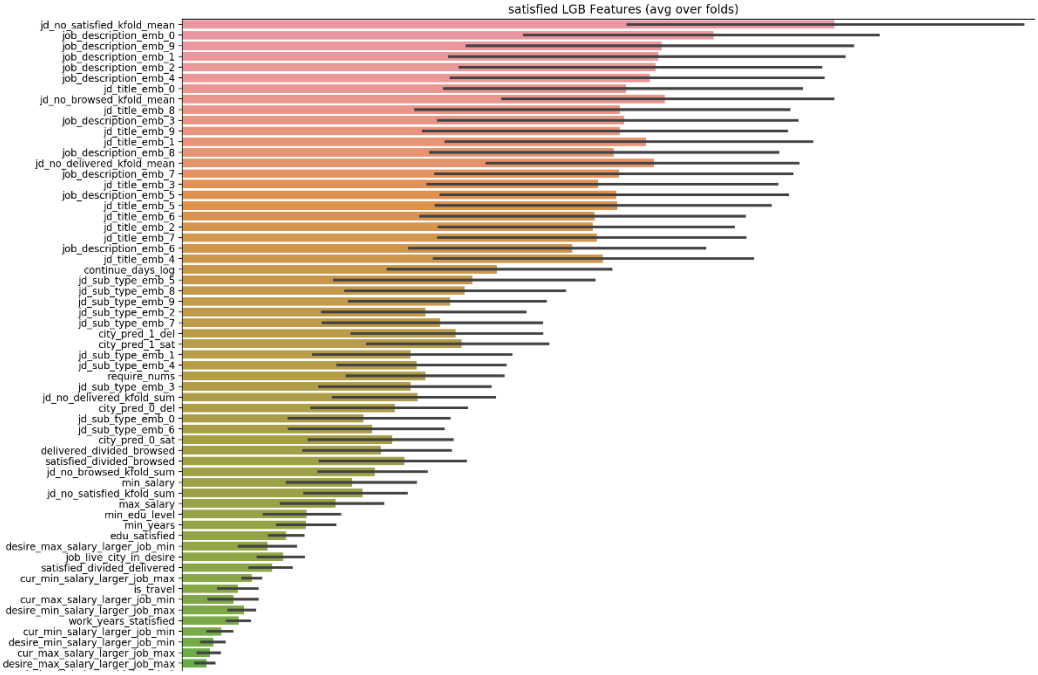


图 15 特征重要性排序结果

从上图可以看到，职位满意平均值最重要，这符合我们的直观认识，后续的一些 word2vector 和 doc2vector 编码的文本信息依次排序。

b) 回归模型：

使用 Pointwise 方式的回归模型来处理，不考虑列表之间的职位交互，模型容易理解。回归的模型构建如下：

```
for idx, (train_idx, valid_idx) in enumerate(folds.split(train_data, train_data[target])):
    x_train = train_data.iloc[train_idx, :]
    x_valid = train_data.iloc[valid_idx, :]

    curr_oof = x_valid[["user_id", "jd_no", "delivered", "satisfied"]]

    result = lbl_emb(target, x_train, x_valid, test_data)
    t_x, t_y = result[0]
    v_x, v_y = result[1]
    test_x, _ = result[2]

    model = lgb.LGBMRegressor(**param)
    print("Fold", idx, "-"*30)
    model.fit(t_x, t_y,
              eval_set=[(t_x, t_y), (v_x, v_y)],
              early_stopping_rounds=100,
              verbose=50,
             )
```

训练参数如下所示（参数调优采用网格搜索，耗时极长，具体细节此处不再赘述）：

```
reg(param=dict(n_estimators=3000,metric=["auc"],learning_rate=0.1,num_leaves=31,
min_data_in_leaf=20, min_sum_hessian_in_leaf=0.002,
max_depth=5,reg_alpha=4.0,reg_lambda=5.0,colsample_bytree=0.8,subsample=0.8)
,n_folds=5,target="satisfied")
```

训练日志如下所示（训练的metric为auc），不同于排序模型，回归模型的训练目标是"satisfied"：

```
Fold 0 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.74281 valid_1's auc: 0.72874
[100]   training's auc: 0.759965        valid_1's auc: 0.733462
[150]   training's auc: 0.772483        valid_1's auc: 0.736373
[200]   training's auc: 0.783409        valid_1's auc: 0.737533
[250]   training's auc: 0.793493        valid_1's auc: 0.738891
[300]   training's auc: 0.802456        valid_1's auc: 0.739805
[350]   training's auc: 0.811142        valid_1's auc: 0.740125
[400]   training's auc: 0.818723        valid_1's auc: 0.740803
[450]   training's auc: 0.826278        valid_1's auc: 0.741979
[500]   training's auc: 0.833048        valid_1's auc: 0.742044
[550]   training's auc: 0.838936        valid_1's auc: 0.7423
[600]   training's auc: 0.845284        valid_1's auc: 0.741956
Early stopping, best iteration is:
[523]   training's auc: 0.835834        valid_1's auc: 0.742476
Fold 1 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.743229        valid_1's auc: 0.725265
[100]   training's auc: 0.760033        valid_1's auc: 0.730681
[150]   training's auc: 0.772761        valid_1's auc: 0.733039
[200]   training's auc: 0.784313        valid_1's auc: 0.734459
[250]   training's auc: 0.794057        valid_1's auc: 0.735663
[300]   training's auc: 0.803214        valid_1's auc: 0.736725
[350]   training's auc: 0.811968        valid_1's auc: 0.736777
[400]   training's auc: 0.819411        valid_1's auc: 0.736672
Early stopping, best iteration is:
[325]   training's auc: 0.807447        valid_1's auc: 0.736897
Fold 2 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.74374 valid_1's auc: 0.726809
[100]   training's auc: 0.760099        valid_1's auc: 0.731736
[150]   training's auc: 0.773296        valid_1's auc: 0.734443
[200]   training's auc: 0.784214        valid_1's auc: 0.735914
[250]   training's auc: 0.794413        valid_1's auc: 0.736491
[300]   training's auc: 0.803438        valid_1's auc: 0.736563
[350]   training's auc: 0.811713        valid_1's auc: 0.737572
[400]   training's auc: 0.819285        valid_1's auc: 0.737843
[450]   training's auc: 0.826425        valid_1's auc: 0.738289
[500]   training's auc: 0.83305 valid_1's auc: 0.738152
[550]   training's auc: 0.839632        valid_1's auc: 0.738046
[600]   training's auc: 0.845897        valid_1's auc: 0.737817
Early stopping, best iteration is:
[518]   training's auc: 0.835391        valid_1's auc: 0.738389
Fold 3 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.741927        valid_1's auc: 0.73202
[100]   training's auc: 0.758132        valid_1's auc: 0.737319
[150]   training's auc: 0.771847        valid_1's auc: 0.738958
[200]   training's auc: 0.782868        valid_1's auc: 0.740217
[250]   training's auc: 0.792884        valid_1's auc: 0.741077
[300]   training's auc: 0.802073        valid_1's auc: 0.741908
[350]   training's auc: 0.810397        valid_1's auc: 0.742763
[400]   training's auc: 0.81792 valid_1's auc: 0.742778
[450]   training's auc: 0.825635        valid_1's auc: 0.742884
[500]   training's auc: 0.83224 valid_1's auc: 0.743316
[550]   training's auc: 0.838498        valid_1's auc: 0.74297
[600]   training's auc: 0.84464 valid_1's auc: 0.742966
Early stopping, best iteration is:
[537]   training's auc: 0.836737        valid_1's auc: 0.743401
Fold 4 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.74375 valid_1's auc: 0.72241
[100]   training's auc: 0.760106        valid_1's auc: 0.727558
[150]   training's auc: 0.773166        valid_1's auc: 0.730255
[200]   training's auc: 0.78438 valid_1's auc: 0.732512
[250]   training's auc: 0.793671        valid_1's auc: 0.733541
[300]   training's auc: 0.802951        valid_1's auc: 0.734465
[350]   training's auc: 0.812174        valid_1's auc: 0.734494
[400]   training's auc: 0.819581        valid_1's auc: 0.734411
Early stopping, best iteration is:
[340]   training's auc: 0.810253        valid_1's auc: 0.734709
```


```
[350]   training's auc: 0.810397        valid_1's auc: 0.742763
[400]   training's auc: 0.81792 valid_1's auc: 0.742778
[450]   training's auc: 0.825635        valid_1's auc: 0.742884
[500]   training's auc: 0.83224 valid_1's auc: 0.743316
[550]   training's auc: 0.838498        valid_1's auc: 0.74297
[600]   training's auc: 0.84464 valid_1's auc: 0.742966
Early stopping, best iteration is:
[537]   training's auc: 0.836737        valid_1's auc: 0.743401
Fold 4 ------------------------------
Training until validation scores don't improve for 100 rounds.
[50]    training's auc: 0.74375 valid_1's auc: 0.72241
[100]   training's auc: 0.760106        valid_1's auc: 0.727558
[150]   training's auc: 0.773166        valid_1's auc: 0.730255
[200]   training's auc: 0.78438 valid_1's auc: 0.732512
[250]   training's auc: 0.793671        valid_1's auc: 0.733541
[300]   training's auc: 0.802951        valid_1's auc: 0.734465
[350]   training's auc: 0.812174        valid_1's auc: 0.734494
[400]   training's auc: 0.819581        valid_1's auc: 0.734411
Early stopping, best iteration is:
[340]   training's auc: 0.810253        valid_1's auc: 0.734709
```


```
satisfiedMAP 0.2325547386955517
deliveredMAP 0.23233567323785792
0.23248901905824354
```

图 16 训练日志数据

从上图可以看到，当采用回归模型训练数据时，训练的结果没有比采用排序模型训练的效果好。这是因为回归模型只是将所有数据看作一个个单独的个体，然后根据特征回归最终的目标，最终根据回归的分数对结果排序。这种考虑是不充分的，因为回归模型没有考虑个体与个体之间的关系，而排序模型较好的解决了这个问题，它将数据集划分为一个个组，每个组的数据都是同一个用户的投递职位，它优化的目标是全局优化，考虑每个组内的所有数据。

satisfiedMAP 评估得分 0.2325，deliveredMAP 评估得分 0.2323，最后的评估分数为两者的加权分数 0.2324，然后对所有特征进行排序，排序的结果如下图所示（显示部分特征）：

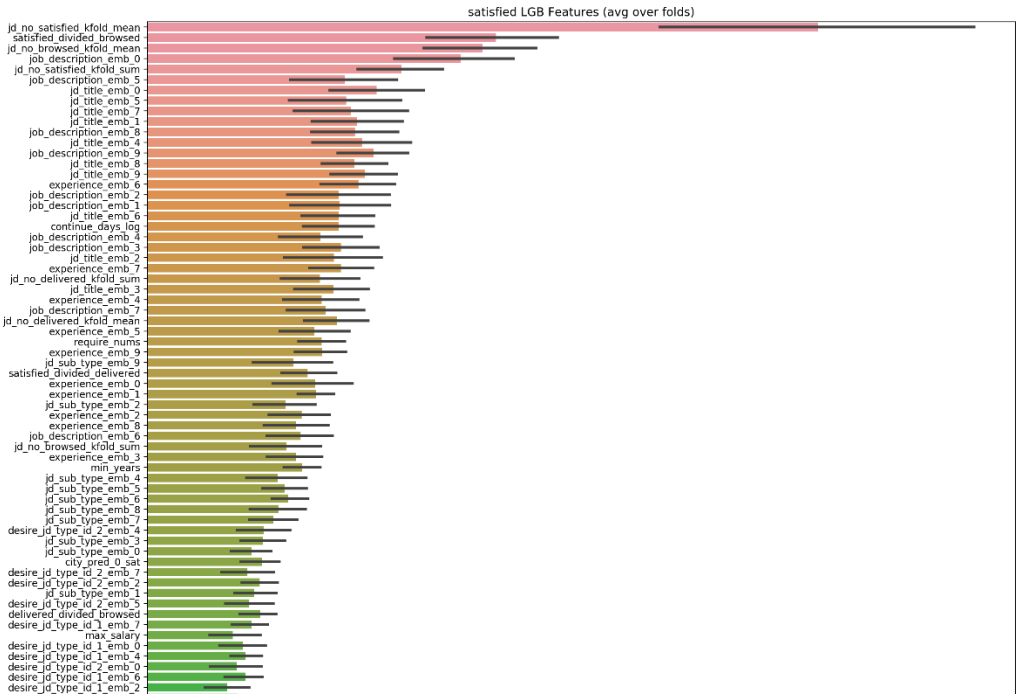


图 17 特征重要性排序结果

从上图可以看到相对于排序模型，回归模型过分依赖于某一特征，没有对特征很好的结合，没有较好的挖掘特征深层次的关联，好的特征排序应该是右侧边缘较为平滑。

6.后续工作（模型集成）

生成两个模型的 oof 结果，oof 结果由五折交叉训练产生，每次选取其中四折进行训练，然后另外一折测试，最后五次测试拼接成 oof 结果，两个模型的 oof 结果如下图所示（oof 结果有 50 万条数据，部分显示）：

user_id	jd_no	delivered	satisfied	predict_target	sigmoid
0009f81c68cddb4db035042dc395e756	abae0ee167656ea9d6b99e81feb3d674	0	0	-0.163390	0.459243
0009f81c68cddb4db035042dc395e756	96a34d3c7294a1da3a2501ca338c61f5	0	0	-0.622863	0.349131
0009f81c68cddb4db035042dc395e756	781e3faf3b37c47d893001b162ea9436	0	0	-0.659688	0.340810
0009f81c68cddb4db035042dc395e756	fd9520b5afa2ecd887ec758eb5f651f8	0	0	-0.217200	0.445913
0009f81c68cddb4db035042dc395e756	7a60b73aa09b8db5e20a7ba7b4e010f9	0	0	-0.176016	0.456109

图 18 排序模型预测结果

user_id	jd_no	delivered	satisfied	predict_satisfied	sigmoid
04edcbfd579b127171e32d7e334f9368	e9959d3b5fd8b26da07e8b8a3b80489a	1	0	0.095294	0.523806
17e1b9f107dd1214bd78dec6d91593a4	dda657439fe72a592b523596453f0290	0	0	0.050720	0.512677
314a3eb69400238af1be9f5d144b4659	7de64630e8869c50f2e3aa5b7c4fe86c	0	0	0.141624	0.535347
03851d836a7c6de623bdfa3e03696c15	7de64630e8869c50f2e3aa5b7c4fe86c	0	0	0.126778	0.531652
0d4787ca3f0be4a49b620aecb86cccaf	7de64630e8869c50f2e3aa5b7c4fe86c	0	0	0.119295	0.529788

图 19 回归模型预测结果

其中 predict_target，predict_satisfied 分别为排序模型和回归模型的预测结果，为了对两个结果较好的集成，所以采用 sigmoid 函数对预测值转换。

a) 平均集成：

平均集成就是将 sigmoid 激活后的值简单的平均，最终处理如下图所示：

delivered	satisfied	predict_target	sigmoid_x	predict_satisfied	sigmoid_y	simple_average
0	0	-0.163390	0.459243	0.039327	0.509831	0.484537
0	0	-0.163390	0.459243	0.041697	0.510423	0.484833
0	0	-0.263562	0.434488	0.039327	0.509831	0.472159
0	0	-0.263562	0.434488	0.041697	0.510423	0.472456
0	0	-0.622863	0.349131	0.012249	0.503062	0.426096

其中 simple_average 就是平均结果，最后集成的分数 satisfiedMAP 0.2783，deliveredMAP 0.2660，MAP 0.2746。

b) 加权集成：

加权平均采用的思想是每个模型赋予不同的权重，权重的选择根据模型训练分数求得，最终的加权平均表达式如下：

$$Score = (\sum_{i=1}^{k} w_i * score_i) / (\sum_{i=1}^{k} w_i)$$

其中，Score 为最终集成的得分，本次取的两个模型的权重分别为：0.2741 和 0.2324，取自它们单模训练的分数，最终处理如下图所示：

delivered	satisfied	predict_target	sigmoid_x	predict_satisfied	sigmoid_y	simple_average	weight_average
0	0	-0.163390	0.459243	0.039327	0.509831	0.484537	0.482454
0	0	-0.163390	0.459243	0.041697	0.510423	0.484833	0.482726
0	0	-0.263562	0.434488	0.039327	0.509831	0.472159	0.469058
0	0	-0.263562	0.434488	0.041697	0.510423	0.472456	0.469330
0	0	-0.622863	0.349131	0.012249	0.503062	0.426096	0.419760

其中 weight_average 就是平均结果，最后集成的分数 satisfiedMAP 0.2789，deliveredMAP 0.2666，MAP 0.2752。

c) 排序集成：

上述的模型忽略组内的排序信息，客观上来说还是 Pointwise 方式的集成，故采用以下公式利用排序信息，进行集成：

$$Score = \sum_{i=1}^{k} w_i / \ln(rank_i + eps) * score_i$$

其中 rank 代表模型 score 在组内的排序，w 为加权平均集成中的权重。该集成思路主要是将被某个模型排序较低，而另外一个模型排序较高的职位推上去，但是同时也要考虑这个模型的置信度，即权重。对排序值取对数函数，即前排职位推上去更重要，对排序值进行一个非线性变换。最终处理如下图所示：

predict_target	sigmoid_x	predict_satisfied	sigmoid_y	simple_average	weight_average	rank_x	rank_y	rank_average
-0.163390	0.459243	0.039327	0.509831	0.484537	0.482454	20.0	54.0	0.071721
-0.163390	0.459243	0.041697	0.510423	0.484833	0.482726	21.0	48.0	0.071987
-0.263562	0.434488	0.039327	0.509831	0.472159	0.469058	34.0	55.0	0.063339
-0.263562	0.434488	0.041697	0.510423	0.472456	0.469330	35.0	49.0	0.063976
-0.622863	0.349131	0.012249	0.503062	0.426096	0.419760	116.0	126.0	0.044305

其中 rank_x 代表排序模型的排序值，rank_y 代表回归模型的排序值，rank_averag 代表最终集成的排序分数，最后集成的分数 satisfiedMAP 0.2801，deliveredMAP 0.2654，MAP 0.2756。

三种集成方式的分数如下所示：

表 1 模型集成结果

	satisfiedMAP	deliveredMAP	MAP
排序模型	0.2784	0.2641	0.2741
回归模型	0.2325	0.2323	0.2324
平均集成	0.2783	0.2660	0.2746
加权集成	0.2789	**0.2666**	0.2752
排序集成	**0.2801**	0.2654	**0.2756**

从上表可以看出，当使用集成的方式后，最终的结果比所有的单模结果都要好，其中采用排序集成方式，MAP 评估结果更优。

7.总结

本文使用机器学习来解决现实生活中的一个问题，在海量的求职者数据、职位数据、交互数据中，挖掘最有价值的信息，尽可能的在求职者与用人单位之间构建一种推荐关系，使得双方都满意。

为此，首先对数据全方面的分析、预处理、清洗数据；然后从业务角度出发，构建不同的特征，用户特征、职位特征、用户职位交互特征等等；接着，基于构建的特征，训练了两个不同的模型，排序模型、回归模型，这两种模型基于不同的思路；最后，将训练后的两个差异性模型进行集成，互补缺陷，将模型的效能发挥最大。

该数据集将所有文件拼接起来，接近 50 万条数据，并且数据字段存在两个突出问题，一是存在很多的高基数特征字段，若是简单的 one-hot 编码，将会产生非常高的维度，并且特征较为稀疏，不利于训练，因此引入均值编码建立特征与目标之间的关系；二是存在很多文本数据，怎么为这些数据构建特征也是一个难点？为此，使用 NLP 中常用的手段 Word2Vecto 和 Doc2Vecto 对这些文本数据构建向量表达，然后再使用 SVD 对生成的向量降维，缓解计算压力，删除冗余特征。

特征的构建上，创新性的引入交叉统计特征、转化率特征，该特征的思路主要来源于广告场景中点击通过率（CTR），平均点击价格（ACP），最后的实验结果也证明，交叉统计特征、转化率特征具有相当高的重要性。

模型构建上，训练了两种不同的模型，一种是 pointwise 方式的回归模型，一种是 listwise 方式的排序模型，两个模型都是基于 GBDT，但是出发点不同。实验结果显示，排序模型效果更优，因为排序模型综合考虑了用户组内职位之间的关系，优化的是全局目标。后续为了结合两种模型的差异性，将效能发挥最大，创新性的引入一种排序集成方法，实验结果显示，该集成方法比其它集成方法较优。

在评估方法上，回归模型训练过程中对于验证数据集采用 Acc 准确率评估，但是这种评估具有一定的缺陷，因为本文解决的问题是一个推荐问题，相对而言，排序指标更重要。因此，后续模型的对比均采用 mAP 进行评估。

参考文献

[1] Micci-Barreca D. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems[J]. ACM SIGKDD Explorations Newsletter, 2001, 3(1): 27-32

[2] He X, Pan J, Jin O, et al. Practical lessons from predicting clicks on ads at facebook[C]//Proceedings of the Eighth International Workshop on Data Mining for Online Advertising. ACM, 2014: 1-9.

[3] Wu Q, Burges C J C, Svore K M, et al. Adapting boosting for information retrieval measures[J]. Information Retrieval, 2010, 13(3): 254-270.

[4] Wang X, Li C, Golbandi N, et al. The lambdaloss framework for ranking metric optimization[C]//Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 2018: 1313-1322.

[5] Ke G, Meng Q, Finley T, et al. Lightgbm: A highly efficient gradient boosting decision tree[C]//Advances in Neural Information Processing Systems. 2017: 3146-3154.

[6] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016: 785-794.