职位薪资数据预测

王靖康	郑继来	王磊
515030910059	515030910114	515030910170
张晴钊	**沙金锐**	**谌仲威**
515030910484	515030910060	515030910069

Abstract

随着社会信息化和电子商务领域的快速发展，如何对海量数据进行有效的数据挖掘是各个行业的热点问题。当面对就业问题之时，对职位薪资数据进行有效的数据分析与处理会让人们的选择拥有更好考虑与选择的空间

本项目首先对数万条薪资数据进行了爬取处理，并根据爬取的属性进行了包括相关性分析及特性分布范围的特征分析；将数据按照数值数据、独热码数据以及多类别数据等不同种类分别进行预处理；最后对职位薪酬和职位类型分别进行了预测。

职位薪酬的预测本项目使用了 GradientBoost 算法进行分析，得到与薪资正相关的三个指标满足学历水平优于工作经验优于语言技能的结论，同时也得到经济发达城市薪资更高、工作城市对薪资的影响的优先级略低于学历水平等结论；职位类型预测本项目使用了随机森林算法进行分析，通过职位描述、职位所属类型和薪酬等因素预测职位类型。

关键词：数据挖掘；回归；分类；薪资数据

1 项目介绍

本项目对前程无忧网站上的职位薪资数据进行了有针对性的爬取处理，得到了 5 个职能范围和 5 个城市地址的数万条数据，并根据爬取数据的 17 个属性进行了特征的相关性分析、特征数值分布等。最后将数据按照不同种类分别预处理后，利用多种回归算法与分类算法进行对比，选择效果最优的算法，对职位薪酬和职位类型分别进行了预测。综上所示，本项目完成了以下内容：

- 使用 Python 和 Scrapy 进行数据爬取；使用 XPath 进行信息提取，得到 5 个职能范围、5 个城市地址的 7 万余条数据。
- 利用 3 种相关性分析系数得到各属性与薪资的相关程度，通过特征数值分布得到各属性数据分布特点。
- 对数值数据、独热码数据以及多类别数据分别进行预处理，以便后续进行数据分析和可视化处理。

- 使用多种回归算法对职位薪酬进行预测，经过误差比对，选择预测效果最优 Gradient Boost 算法进行职位薪酬预测。
- 使用多种分类算法对职位类型进行预测，经过分类耗时与分类效果等综合分析，采用效果最优的随机森林算法进行职位类型预测。

2 数据爬取

小组爬取的是前程无忧（51job）上的职位薪资数据。由于不同行业之间哪怕相同岗位，其薪资差距也可能很大。为了降低这种影响，我们最终选择了和专业更为相关的计算机行业，具体表现为前程无忧上**计算机软件**、**计算机硬件**、**互联网/电子商务/网游**、**IT-管理**、**IT-品管**、**技术支持及其他**这五个职能范围。同时为了进行地区间的对比，我们选择了**北京**、**上海**、**广州**、**深圳**、**杭州**五个城市，最终爬取到岗位薪资数据共计 **71641** 条。

2.1 爬取工具

我们使用了 Python 3.6 和 Scrapy 1.5 来进行爬取，爬取过程中的 Html 信息提取使用的是 XPath 方法。

Scrapy 是 Python 开发的一个快速、高层次的屏幕抓取和 web 抓取框架，用于抓取 web 站点并从页面中提取结构化的数据。其中封装了多种类型的爬虫基类，并且支持 CSS、XPath 等多种网站解析工具，可以方便地进行修改和扩展。

XPath 即为 XML 路径语言，它是一种用来确定 XML（标准通用标记语言的子集）文档中某部分位置的语言。它基于 XML 的树状结构，有不同类型的节点，包括元素节点，属性节点和文本节点，提供在数据结构树中找寻节点的能力，进而能方便地提取节点下的各种信息。

2.2 爬取流程

整个爬取过程主要分为三个步骤：搜索信息确定搜索结果页 url，从搜索结果页中提取每个职位详情页 url，从详情页中爬取所需信息。具体代码见7.2。

第一步，由搜索信息确定搜索结果页 url。在前程无忧高级搜索功能中输入之前的城市和职能范围限制条件，取得搜索页的第一页 url。然后选择其他页码，找到明显不同的一项即为页码表示项，穷举所有页码可得到所有搜索结果页 url。该对比可见图 1

第一页： https://search.51job.com/list/030000%252C070000%252C080000%252C090000%252C100000,000000,0000,00,9,99,%2B,2,1.html?lang=c&stype=1&postchannel=0000&workyear=99&cotype=99&degreefrom=99&jobterm=99&companysize=99&lonlat=0%2C0&radius=-1&ord_field=0&confirmdate=9&fromType=&dibiaoid=0&address=&line=&specialarea=00&from=&welfare=	**第二页**: https://search.51job.com/list/030000%252C070000%252C080000%252C090000%252C100000,000000,0000,00,9,99,%2B,2,2.html?lang=c&stype=1&postchannel=0000&workyear=99&cotype=99&degreefrom=99&jobterm=99&companysize=99&lonlat=0%2C0&radius=-1&ord_field=0&confirmdate=9&fromType=&dibiaoid=0&address=&line=&specialarea=00&from=&welfare=

图 1: 确定搜索结果页 url 页码位置

第二步，从搜索结果页中提取每个职位详情页的 url。在浏览器中通过 F12 查看网页结构，找到每个详情页对应的 url 位置，见图 2。可利用 XPath 规则’//div[@id=”resultList”]/div[@class=”el”]/p/span/a/@href’ 提取。

图 2: 确定职位详情页 url

第三步，在详情页中提取信息。利用爬虫访问每个详情页，在详情页中提取所需要的信息，并保存在一个对象中用于导出。所需要提取的信息见图3，图中的编号和表 1的顺序一致，该详情页来自`https://jobs.51job.com/guangzhou-lwq/103569280.html?s=01&t=0`。具体提取规则可见7.2。

图 3: 详情页提取信息

2.3 爬取结果属性

我们最终爬取到的数据集在未进行预处理的情况下共有 17 条属性，其具体情况见表 1。

3 特征分析

本节采用特征工程的分析方法，针对爬取的数据的各项特征进行比较全面的挖掘分析。我们首先采用多种相关性系数分析并量化特征相关性，用以指导后续的数据预处理过程。进一步地，我们分析并可视化特征间的关联关系，并选取关联度高的特征重点分析。

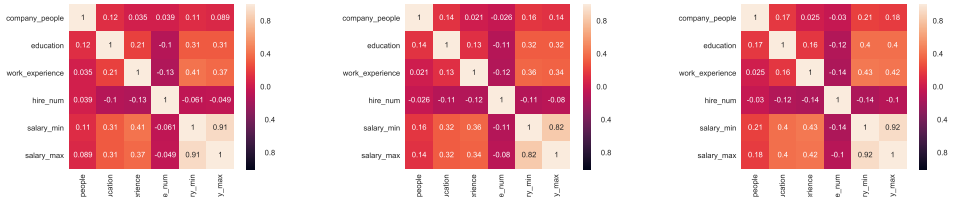
3.1 相关性分析

使用 pearson[3]、kendall[1]、spearman[5] 三种相关性分析方法对容易数值化的特征进行相关性系数的计算。图形化结果见图4，颜色越浅正相关程度越高；反之，反相关程度越高。次步相关性分析包括了公司规模、学历要求、工作经历要求、招聘人数等特征和工资下限和工资上限。

表 1: 爬取数据集属性说明

属性名	含义	备注
area	工作地点	网站上有的只标注了城市，有的标注了区划
company	公司名称	
company_people	公司规模	以人数表示
company_service	公司所处行业	每个大类用逗号隔开，大类中的小类用/隔开
company_type	公司类型	“民营/国企/外资”等
demand_profession	招聘要求专业	可以为“无”
education	要求学历水平	
hire_num	该职位计划招聘人数	
job	职位名称	
job_info	职位具体信息	不同公司内容不同，主要有岗位描述和具体要求
job_keyword	职位关键字	可以为空
job_work_type	职位职能类型	可以为空
language	语言水平要求	可以为“无”
put_time	职位发布时间	
salary	职位薪酬	以上下限表示，计量单位（年月日）存在不同
welfare	提供的福利	
work_experience	要求工作经验	

三种相关性系数对于结果而言差异不大。观察相关性矩阵, 可以注意到招聘人数对其他特征的相关性系数很低，分析数据集我们认为主要原因是招聘人数在数据集中差异不大。与工资相关性最高的是学历要求和工作经历要求，达到 0.3 左右。其他特征和工资存在一定相关性但不明显。



(a) pearson 相关矩阵　(b) kendall 相关矩阵　(c) spearman 相关矩阵

图 4: 部分特征相关性分析图示

图 5用箱线图的方式，更具体地展示相对于学历要求的工资分布。基本可以得出学历要求越高，工资期望越高的结论。比如博士学位要求的职位无论是平均值还是四分位值都远远高于其他学历。次之是硕士学位和本科教育程度。出乎预料的是无学位要求的职位表现出高于本科学历以下要求的职位预期工资，可能是由于部分数据的缺失，在数据预处理部分我们会对这一部分数据进行处理。

图 6用箱线图的方式，更具体地展示相对于工作经历要求的工资分布。工作经验越多，工资期望越高，箱线图明显展示了这一特征。

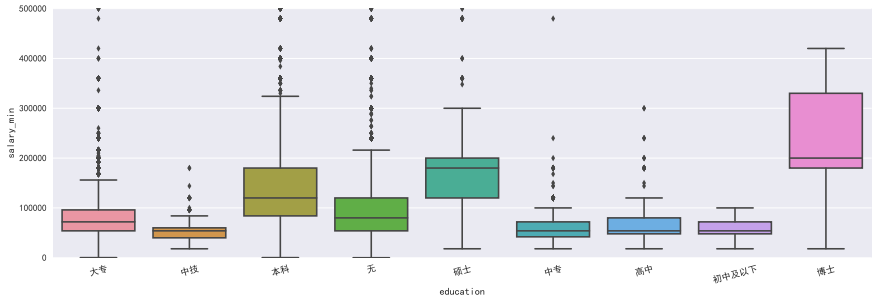


图 5: 不同学历要求对应最低工资分布图

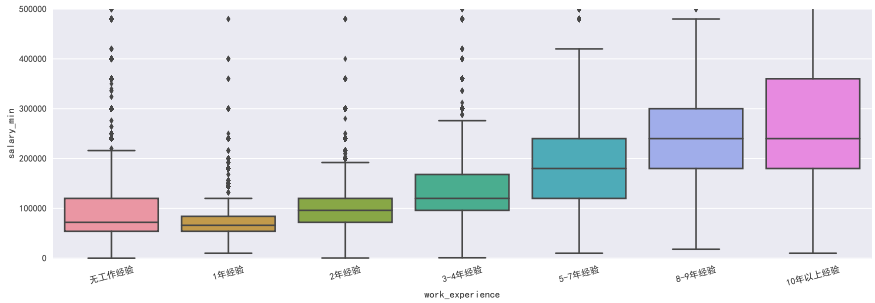


图 6: 不同工作经历要求对应最低工资分布图

此外，我们选取图4相关性分析中未包含的其他特征进行相关性的可视化，如公司类型和地域。图7使用箱线图展示公司类型对应的工资分布。总体而言，公司类型对于平均工资差异不大，欧美外资公司、上市公司和国企表现出稍高的工资倾向。图8展示的工资地域分布中，北上广深出人意料地并没有明显的工资优势，反而是珠海、天津、西安等地的职位平均工资较高。

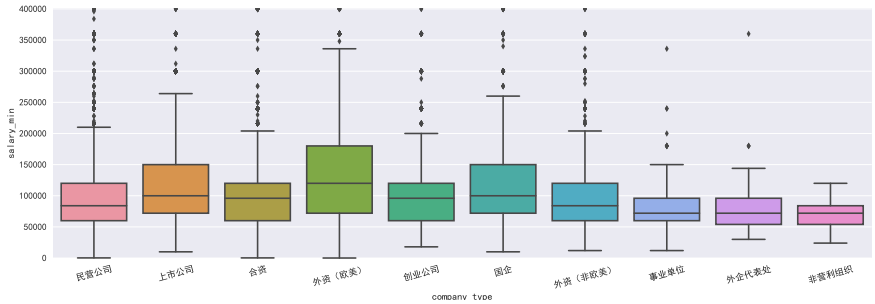


图 7: 不同公司类型对应最低工资分布图

最后我们分析特征之间的关系，并使用多变量图（图9）进行可视化。得出了学历要求和工作经历要求共同作用工资标准等一系列结论。

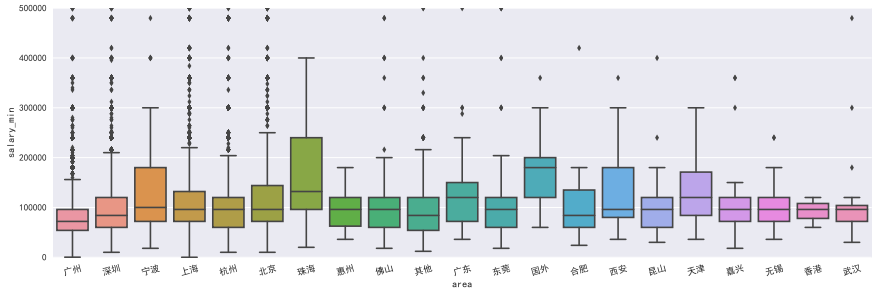


图 8: 不同地域对应最低工资分布图

图 9: 多变量相对最低工资关系图

3.2 特征数值分布

中小型公司占数据集主要部分。如图 10所示，爬取数据中 50-150 人的中小型公司占比最大。少于 50 人的小型公司和 150-500 人的中型公司个占比五分之一左右，这三类公司规模占据了数据集的 75% 左右。可见数据的采集保证了合理的随机分布，公司规模越大，数量越少。

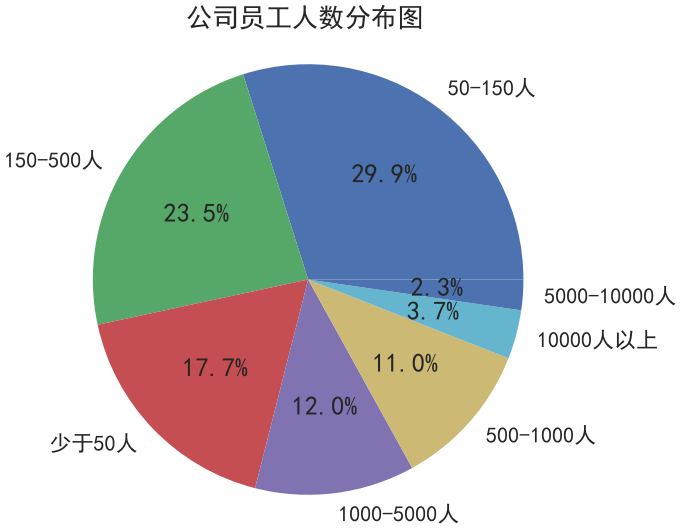
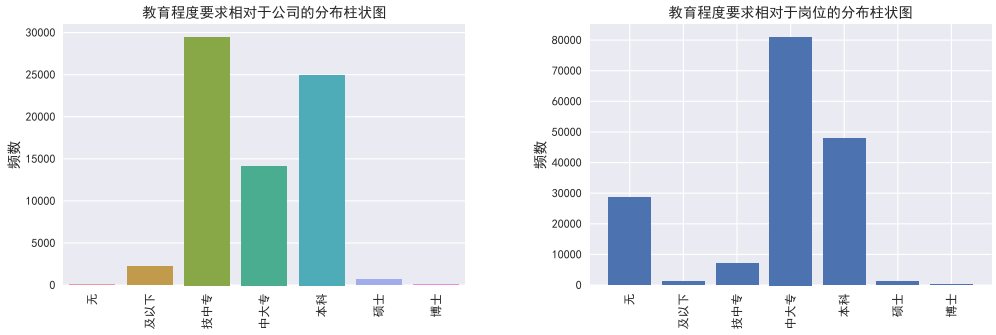


图 10: 公司规模分布饼图

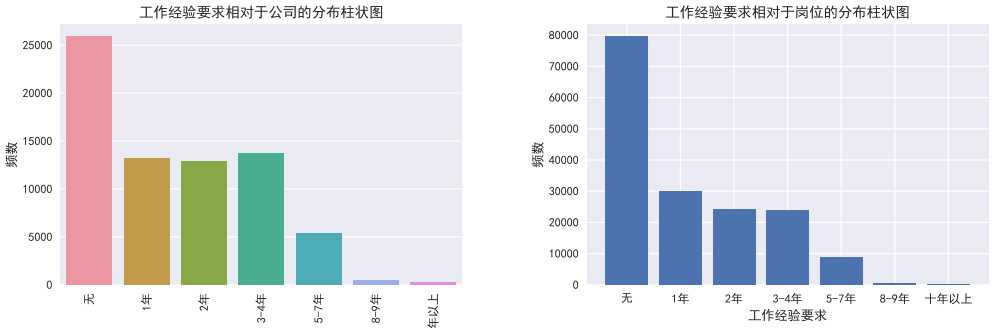
如图 11, 采集到的数据中，中技和中专以上的学历要求占比最大，本科的学历要求次之。然而考虑职位需求的数量，发现高中大专学历对应的岗位数量最多，无学历要求的岗位数量也非常可观。可见高中大专的学历要求虽然对应的招聘条目不及中专中技，但岗位数量需求大。无学历要求的岗位同理。可以预料地，要求硕士及以上学历的职位非常稀少，本科学历已经可以胜任大部分数据集中的职位。



(a) 教育经历要求的职位分布情况　　(b) 教育经历要求对应职位数量分布情况

图 11: 教育经历要求分布

对于工作经历要求，从图 12 可以看出，数据集中无工作经历要求的职位占大多数，工作经历要求越高的职位数量也越少。



(a) 工作经历要求的职位分布情况　　(b) 工作经历要求对应职位数量分布情况

图 12: 工作经历要求分布

数据集地域的分布明显呈现一线城市集中的特征。由图13，北京、上海、广州、深圳、杭州等一线城市占据大部分职位需求，其他城市仅占数据集的 1.3%。

对于公司的类型分布，由图14，数据集显示民营公司提供了接近 70% 的工作机会，其次是上市公司、合资公司、外企、国企等。

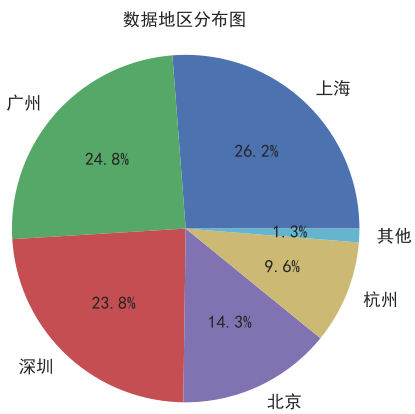


图 13: 地域分布饼图

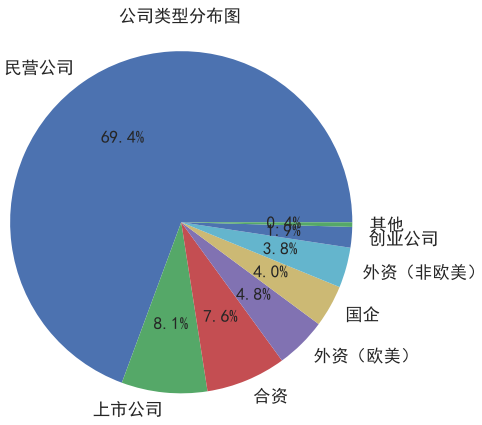


图 14: 公司种类分布饼图

散点图15横坐标表示工资下限，纵坐标表示工资上限，两者 pearson 相关性 0.91。由散点图可见，绝大部分职位分布在 40 万年薪以下。根据进一步分析，工资下限平均值在 10 万年薪左右，工资上限平均值在 15 万年薪左右。

最后绘制中国局部地区地图来可视化岗位地域分布以及对应的平均工资。如图16。

4　数据预处理

本次数据预处理的任务，主要是以我们利用 scarpy 爬虫在前程无忧网上爬取到的原始数据作为基础，进行一定的信息归整化与提取，从而能让后续数据可视化与数据分析的操作更加方便与准确。为了达成这一目的，我们采用了 python 的 pandas 库提供的 dataFrame 数据结构进行数据的组织，并采用一些正则表达式 re 等工具进行文本或数字的分析处理。

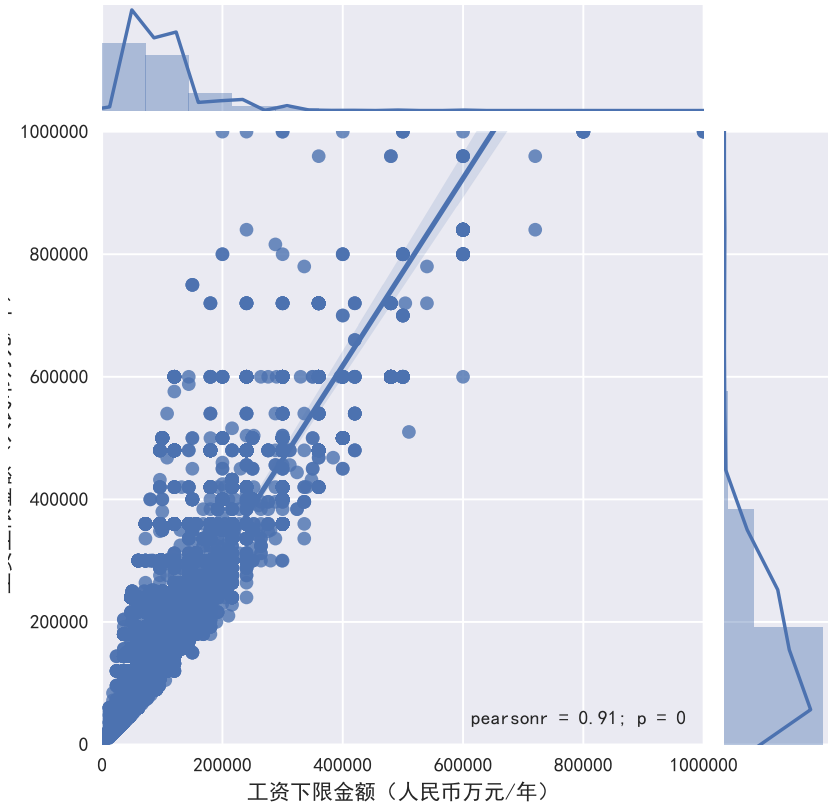
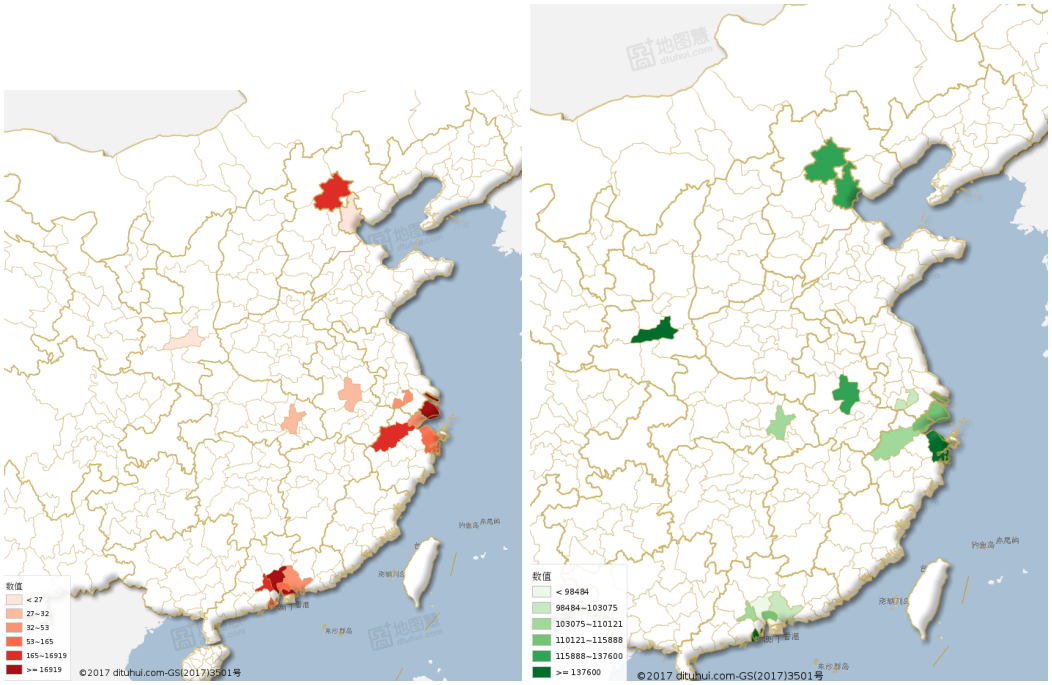


图 15: 工资分布散点图



(a) 职位数量地图分布情况　　(b) 平均工资下限地图分布情况

图 16: 职位与工资地图分布

值得一提的是，本项目中，由于数据可视化与数据分析所需要的数据略有不同，因此我们提供了一个适用于数据可视化的预处理版本 middleData.csv 以及一个进行了进一步数字化处理的预处理版本 quantityData.csv。由此，我们进一步细分了预处理的目标，让后续工作更加方便。

4.1 数值数据的预处理

事实上，本次实验中直接从前程无忧网上所爬取到的数据全部为字符串的表示形式，但其中一部分数据其实隐含了明确的数字化意义，因此我们需要将其全部进行数字化，从而方便后续的数据分析和可视化。

下面，我们将列举所有需要数字化的数据，并且突出其在 middleData.csv 与 quantityData.csv 中的表示方式的区别。

- **公司规模 company_people**

  对于公司规模 company_people 而言，原本爬取到的数据共有 7 类，分别是“少于 50 人”，“50-150 人”，“150-500 人”，“500-1000 人”，“1000-5000 人”，“5000-10000 人”，“10000 人以上”。在 middleData.csv 中，我们保留了原来的字符串类，以便进行作图。而我们又考虑到，一个公司的人数与这个公司的规模呈一定的正相关性，而公司规模又极大地影响薪水等因素。因此，我们需要在 quantityData.csv 中将这个字符串的类转化为数字形式，这里我们采取的是该类的下界（如 500-1000 人则数字化为 500）。

- **教育经历 education**

  对于教育经历 education 而言，原本爬取到的数据共有 9 类，分别是“无”，“初中及以下”，“中技”，“中专”，“高中”，“大专”，“本科”，“硕士”，“博士”。在 middleData.csv 中，为了作图方便，我们保留了原来的字符串类，以便进行作图。而在 quantityData.csv 中，由于学历存在高低之分，因此独热码并不适合这个属性。因此，我们根据学历的高低，将学历进行细化的数值化，越高的学历数字越高，例如“博士”对应 9，“中专”对应 1，而“无”则取中位数 3（因为可能无学历要求只是不写，不是没有要求）。

- **工作经历 work_experience**

  对于工作经历 work_experience 而言，原本爬取到的数据共有 7 类，分别是“无工作经验”，“1 年经验”，“2 年经验”，“3-4 年经验”，“5-7 年经验”，“8-9 年经验”，“10 年以上经验”。在 middleData.csv 中，为了作图方便，我们保留了原来的字符串类，以便进行作图。而在 quantityData.csv 中，我们认为工作经验的多少对于工资有一定正相关，因此我们将上述类别根据其工作经验进行数字化，如“3-4 年经验”变为 3.5。

- **最大最小薪水 salary_max 与 salary_min**

  最大薪水由于最小薪水本质上都是从原始数据中 salary 一列取得，因此我们在这里并为一谈。由于薪水是本项目中最重要的属性/标签，在作图与分析中均占极大因素。因此，我们在 middleData.csv 与 quantityData.csv 中均将其直接提取计算出来。

  计算过程的复杂在于，对于年薪这个数据，理论上应当是一个 int 值，但实际前程无忧网上提供了许多种表示方式，如元/日、千/月、万/年、千/年等，这就需要对不

同的格式进行不同的处理，需要考虑的因素非常多。在实际的数字化中，我们将其全部化为年薪。例如某人工资为 100 元/日，则其年薪计算公式为 100*300=30000 元/年；又某人工资为 18 千/月，则其年薪计算公式为 18*1000*12=216000 元/年。

4.2 独热码数据预处理

我们在前程无忧网上爬取到了很多分类的数据，其中一部分被我们数值化，而另一部分则被作为独热码处理。下面我们将简要描述这些独热码处理的数据，同时说明其在 middleData.csv 与 quantityData.csv 中的表示方式的区别。

- **省市级地点 area 与区县级地点 area_detail**

  在前程无忧网上爬下来的原始数据中，area 字段同时包含了省市级字段与区县级字段（如“上海-闵行区”，“北京-海淀区”），这就导致了该列类别过多，且不够精细，还不适合于输入模型。

  因此，我们在第一轮处理（middleData.csv）中，就将原始的 area 属性拆解为省市级地点 area 与区县级地点 area_detail，即将原始数据以“-”为分割切成具有层次关系的两列。值得一提的是，我们在省市级地点 area 中，剔除掉了“哈尔滨”等出现次数过少的字段；而在更复杂的区县级地点 area_detail 中，我们仅仅保留了北上广深杭五地的区县。而在第二轮的处理（quantityData.csv）中，我们直接将上述两列利用 pandas 的 dataframe 类提供的 get_dummies 函数进行了独热码化。

- **公司种类 company_type**

  在前程无忧网上，公司种类一栏共有“民营公司”、“外企”、“国企”等约 10 类，并且这些公司种类之间也并没有什么高低贵贱之分，因此并不需要进行什么数字化。在 middleData.csv 中，为了方便作图，我们直接用一列字符串进行表示；在 quantityData.csv 中，我们直接用 pandas 的 dataframe 类提供的 get_dummies 函数进行了独热码化。

4.3 多类别数据的预处理

在前程无忧网上，存在一系列的多类别数据。所谓多类别，即该职业的某个属性可以同时属于多个类别，如某职位同时要求“德语”与“英语”，那么在 language 属性上就同时属于两个不同的类别（在前程无忧网站上，这类型数据体现为多选框）。在这一章，我们将详细解析这一类型数据的预处理方式。

- **公司服务类型 company_service**

  前程无忧网上，提供了关于公司服务类型的描述，例如“互联网/电子商务”，“贸易/进出口”，“通信/电信/网络设备”，“仪器仪表/工业自动化”等，共约 60 类。值得一提的是，前程无忧网设置了一个公司所属的服务类型的上限为 2，即一个公司最多属于两个服务类别。

  因此，在 middleData.csv 中，我们安排了两列数据 company_service_1 与 company_service_2，分别安放两个服务类别（字符串的形式）。而在 quantityData.csv 中，我们则安排了每一个可能的类别为一列，并使用 company_service_xxx 为列名（xxx 为类名）。若某职位的公司属于某一列的服务类别，该项设置为 1。因此，实际一个职位（一行）最多两个 1。

- **职位所属类别 job_work_type**

与公司服务类型 company_service 类似，前程无忧网上也提供了关于职务所属类别的描述，如“项目经理”，“电子商务专员”，“软件 UI 设计师/工程师”等。同样，同一职位只允许出现两个职务类别，我们在 middleData.csv 中就安排了 company_service_1 与 company_service_2 两列安放这两类职务。

然而，由于 company_service 不仅种类众多，有上万类，且常有特殊的自定义的职务类别，因此在输入模型前必须进行一定的正则匹配。由于时间原因，我们没有采用词向量或其他高端方案，仅仅手动提取了 30 个关键词，如“硬件”，“项目”，“专员”等，并在 quantityData.csv 中对每一个关键词分配一列。若某个职位的职位类别存在该关键词子串，则该项设置为 1。

- **职员福利 welfare**

  职员福利是最为难以处理的数据。在原始的数据中，一个职位可能存在任意多的职员福利（极端的甚至超过 10 种福利），具体如“五险一金”，“周末双休”，“交通便利”，“五险一金”，“绩效奖金”等。并且，福利的种类近万，极为难以操作。

  因此，我们首要需要做的是找出最主要的福利种类，这里我们搜索了所有的数据集，仅留下其中出现次数超过数据集数量 1% 的数据，共 70 类。之后，我们为每一类福利分配一列 0-1 编码的数值化属性。若某个各位的福利经过 re 的正则匹配存在该关键福利词子串，则该项设置为 1。

4.4 预处理总结

表 2总结了上述数据预处理过程。可以看到，我们对于不同数据的处理完全不同，均是根据需求和实际情况进行特化。由此可见，数据预处理部分确实有很大的的技术与难度，并没有想象中这么简单。

5 项目任务和实验结果

在基于对我们爬到的数据集的分析和处理后，我们进行了如下几项任务：1）职位薪酬（上限和下限）的预测; 2) 职位所属领域分类预测。

5.1 职位薪酬的预测

本次项目的主要任务为对进行统一处理后的职位薪酬（元/年）的预测，由于变量众多，我们不可能对将数据集中所有变量输入来预测不同职位的薪资，故需要选取一些较为重要的因素。

我们选取了八个项目, 包括公司规模、要求学历、工作经验、工作城市、公司类型、福利数目、工作类型、要求语言技能等，具体信息如表 3所示。

在该任务中, 我们采取了几种回归算法, 包括 AdaBoost[12], Random Forest[11], Bagging[4], Gradient Boost[7], SVR[2], Bayesian Ridge[9], Elastic Net[15] 和 MLP[14] 回归。通过实现分别测试了它们的回归准确率和运行时间。结果如表4所示。（注：由于 Random Forest 和 Bagging 算法无法拟合，故在表中未显示）

如表4所示，多种回归算法中，Gradient Boost 回归算法效果最好，R^2 达到 0.3237，平均误差为 34031.4 元/年，而 SVR 效果最差，平均误差为 43304.9 元/年，同时 SVR 的速度最慢，需要 350.08s。同时，可以发现惩罚项为 Ridge 的贝叶斯回归在速度上很有优势（比

表 2: 数据预处理总览表

数据名	数据含义	MiddleData 数据组织方式	QuantityData 数据组织方式
area	大地区（北京上海等）	列名 area，字符串形式	独热码，列名 area_xxx，仅在正确地区显示 1
area_detail	具体地区（黄浦区、闵行区等）	列名 area_detail，字符串形式	独热码，列名 area_detail_xxx，仅在正确地区显示 1
company_people	公司现有人数（规模）	列名 company_people，以字符串形式体现，约 7 类	已数字化，未归一化
company_service	公司的业务	两列，company_service_1 与 company_service_2，分别是公司的两类业务，总种类数约 60	多热码显示，每一列名 company_service_xxx，若有该公司业务两类则两列有 1
company_type	公司种类（私企国企等）	列名 company_type，共一列，直接字符串显示	独热码处理
education	要求的学历	列名 education，共一列，直接字符串显示	数字化，[" 无", " 初中及以下", " 中技", " 中专", " 高中", " 大专", " 本科"," 硕士"," 博士"] 分别对应 [3, 0, 1, 1, 2, 2, 4, 6, 9]
hire_num	招聘人数	列名 hire_num，数字化，999 代表不限人数	数字化，999（不限人数）被替换为中位数 1
Job_work_type	工作性质	两列，job_work_type_1 与 job_work_type_2，分别代表两种工作性质	多热码编码，采用 30 个关键字符串进行关键字匹配，如：“销售”，列名 job_work_type_ 销售。若包含该子串，则该列为 1（共 30 列，每一列代表一个关键词）
language	语言要求	两列，language _1 与 language _2，分别是两类语言（因为有的工作需要两门语言）	多热码，列名 language_xxx，如有两门语言则两列 1
work_experience	工作经历年限	列名 work_experience，字符串显示，共约 10 类	数字化，[" 无工作经验", "1 年经验", "2 年经验", "3-4 年经验", "5-7 年经验", "8-9 年经验", "10 年以上经验"] 分别对应数字 [0, 1, 2, 3.5, 6, 8.5, 11]
salary_max	年薪最大值	数字化	数字化
salary_min	年薪最小值	数字化	数字化

表 3: 用于预测薪资的职位属性

属性名	数据含义	备注
company_people	公司员工规模	整数
education	要求学历水平	独热码，见数据预处理部分
work_experience	要求工作经验	实数，以年为单位
hire_number	招聘员工数	整数
city	工作城市	独热码（精确到城市）
company_type	公司类型	独热码（关键词匹配）
welfare_sum	福利数目	整数（福利数量）
job_type	工作类型	独热码，见数据预处理部分
language	要求语言技能	独热码，见数据预处理部分

表 4: 薪资预测结果

回归算法	R^2 value	平均误差（元/年）	运行时间
AdaBoost	0.2350	37483.9	0.79s
Gradient Boost	**0.3237**	**34031.4**	3.13s
SVR	0.0092	43301.9	350.08s
Bayesian Ridge	0.2667	36538.3	0.05s
Elastic Net	0.0426	44784.4	**0.03s**
MLP	0.2682	36207.3	19.29s

Gradient Boost 快 100 倍)，同时可以达到可最优算法 Gradient Boost 接近的效果（差距 5% 以内)。

使用训练好的 Gradient Boost 算法，示例预测结果如表所示。

表 5: 薪资预测示例（Gradient Boost）

公司规模	工作城市	工作经验	学历水平	语言技能	薪资预测 (下限)	真实薪资 (下限)
50	西安	0	本科	无	494345	48000
100	上海	3.5	博士	英语	315993	340000
50	北京	1	硕士	德语	87130	120000
50	武汉	0	本科	无	80869	96000
100	南京	4	硕士	无	274579	240000

5.2 职位类型预测

由于职位类型对于工作薪酬的影响较大，同时我们得到的数据集也包含了职位所属领域以及对于工作的描述，所以我们分别进行了根据职位描述预测职位所属类型和根据薪酬等其他因素预测职位类型。在爬取的数据中进行关键词匹配可以共 32 种工作类型，如美工、医学、测试、推广、总监、技术、实习等等。

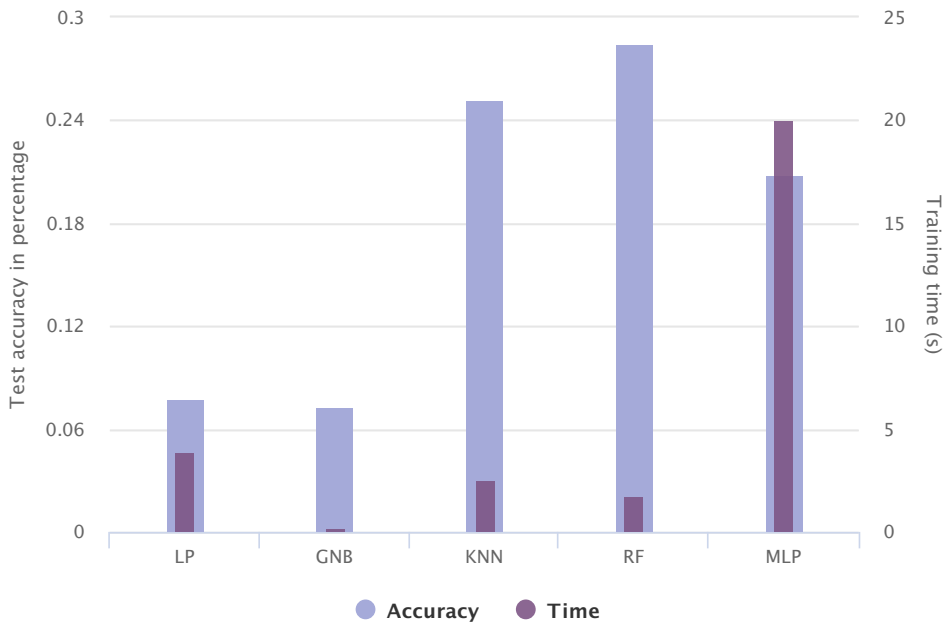
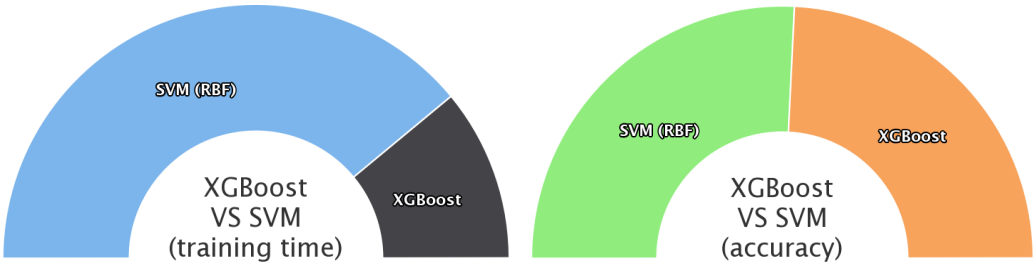


图 17: 几种算法在职位类型分类任务上的对比

同样地，我们使用了几种最具代表性的分类算法，包括支持向量机 SVM[8]，感知器 [13]，高斯贝叶斯 [9]，K 近邻 [10]、随机森林 [11]、XGBoost[6] 和神级网络多层感知器。具体结果如图 17和图 18所示。容易得到，在该任务上，随机森林（RF），支持向量机（RBF 核 SVM）和 XGBoost 的效果最好，可达到 28.4%；而 LP 和 GNB 分类效果相对较差。图 17和18几个算法在性能上差别很大 MLP、SVM 和 XGBoost 算法训练时间较长。特别地，XGBoost 和 SVM 的训练时间分别为 332.05s 和 1177.8s。因此，综合上述分析，在该数据集上，随机森林既能取得很好的效果，同时所消耗的训练时间也较少，是一个比较好的选择。



(a) 时间对比　　(b) 准确率对比

图 18: SVM 和 XGBoost 性能比较

值得说明的是，由于数据爬取的时候具有较强的随机性和不确定性，而由于时间有限我们并未针对数据做进一步审查，所以导致数据分布差异性较大，同时，由于工作类别较多（共有 32 类），所以该任务难度较大。将 32 类进一步归结后，我们可以在该数据集上得到更好的效果。

另外，另一种较为可靠的解决该问题的思路是针对数据集中的职位描述信息，进行特征提取，然后进行语义的分析从而更好地完成职位类型的精准预测/评估。具体地，我们可以先将文本离散化处理，可以使用该数据集作为词向量的训练空间，自行训练该语境下的词向量

分布；也可以使用 google 提供的训练好的 word2vec 将文本转化成定长向量。在抽取 vector 之后，我们有两种解决方案。一种是对每个词向量归一化，进行词向量叠加，从而获取职位描述的向量描述。另一种是使用循环神级网络（RNN, 一般 LSTM 效果更好），抽取特征，后使用全连接对抽取的特征进行合并。最后，使用 softmax 层或 crf 层进行 label 预测。值得说明的是，由于时间有限，我们并未完成本部分的实验。

本项目所有代码、数据、文档、图片等在 Github 上开源，仓库地址为`https://github.com/wangjksjtu/Data-Mining-51Job`。

6 总结

本项目经过数据爬取、特征分析、数据预处理等前期分析处理，最后对职位薪酬和职位所属领域分类分别进行了预测。

职位薪酬的预测采用了多种回归算法，并选择效果最好的 Gradient Boost 算法进行分析，得到了不同城市、不同经验、不同学历下薪资的预测结果。可以看出，学历水平、工作经验、语言技能都与薪资呈正相关的趋势，其中对薪资影响的优先级为学历水平优于工作经验优于语言技能，同时也可以发现，诸如上海南京等经济发达城市的薪资更高，但工作城市对薪资的影响的优先级略低于学历水平。

职位类型预测采用了多种分类算法，并选择效果最好、训练时间最少的随机森林算法进行了根据职位描述预测职位所属类型和根据薪酬等其他因素预测职位类型的分析。

7 附录

7.1 个人感想

7.1.1 王靖康

本次大作业是一个比较贴近真实应用场景的项目，需要我们从数据获取，处理，再到任务设计，算法设计，最后得到在该数据集上一个较好的模型结果。相信在完整地体验过这一流程后，我们会对数据挖掘的流程和基本操作有了更为深刻的认识。

在本次项目中，我主要负责前程无忧网薪资预测和工作类别分类两个任务; 另外，我还参与了部分数据预处理及可视化的工作。在本次实验中，最另我印象深刻的对数据的不同处理方式。与常用的标准数据集不同，我们爬下来的数据由于未经人工审核，可能存在很多不准确，乃至虚假/错误的信息，这对于模型的预测结果影响是非常大的。比如，由于薪资差异很大（部分数据偏离过大），一些归一化的操作将会收到一些极端值的影响，因此我们有必要进行一些数据预处理，舍去或替代明显的离群点 (outlier)。同时，本次项目又使我对各种回归算法和评价指标有了较为全面的理解，也更加熟悉了 python 的 sklearn 和 pandas 库。特别是 pandas 库，其在数据预处理，可视化，数据清理等方面非常有效，也是我之前没有接触的。

另外，由于本次项目时间较为紧张，小组成员较多，又正值期末考试时间，所以如何进行任务分工，多人任务如何有效衔接，如何落实进度都是十分重要的。作为小组长，我在本次项目中学习到了如何在一个团队中尽可能调动多方积极性，充分发挥小组成员的优势。另外，及时的沟通和完整的文档（如爬取数据文档、预处理文档）都是保证小组沟通和任务衔接的必要条件。

最后，非常感谢范老师在一个学期来的帮助和指导，使我对数据挖掘的经典流程和算法都有整体的把握和认识。

7.1.2 王磊

在本次大作业中我负责的是前程无忧网站数据爬取部分，主要使用的是 Python 的 Scrapy 爬虫框架。虽然之前有使用过 Scrapy 的经历，但没有像这次一样爬取大型网站如此大量的数据。因此在这次大作业中对于爬虫框架和 XPath 方法都有了更加深刻的理解，也对数据挖掘的第一步数据采集有了更为直观的感受。

爬取过程中的首要问题是如何从网站纷繁复杂的结构中准确写出提取信息的 XPath 表达式。由于之前 XPath 的使用没有此次如此复杂多变的情况，因此着实遇到了一些困难。包括如何查找所有子孙节点的信息并进行拼合、如何结合属性以及节点进行查找等等。爬取过程中参照了不少网上的资料，对于 XPath 在网页信息提取上对比正则匹配的优劣，有了更加清晰的认识。

在爬取过程中我充分感受到了数据的杂乱不一，在前程无忧这种对于发布信息有结构化要求的网站上仍然有很多信息非常杂乱，最典型的就是职位名称，有的公司会为了吸引人而在名称上加入各种各样的福利薪资等信息，使得从中提取出真正的职位信息变得相当困难。除此之外，自主性较高的信息能否使用、如何处理，也让我有了一些思考。

数据挖掘，数据先行，数据采集和预处理的过程无比重要。感谢老师能提供这样一个实际操作的机会，感谢我的各位队友在实践过程中对我的帮助。谢谢大家！

7.1.3 郑继来

本次项目中我主要负责的是前程无忧网上爬下的数据的预处理部分，主要使用的是 python 的 pandas 库提供的 dataFrame 数据结构进行数据的组织，并采用一些正则表达式 re 等工具进行文本或数字的分析处理。

在预处理的过程中，我有很多印象深刻的点。首当其冲的，就是数据的不规整性超乎我的想象，预处理工作量非常之大。譬如年薪这个数据，理论上应当是一个 int 值，但实际前程无忧网上提供了许多种表示方式，如元/日、千/月、万/年、千/年等，这就需要对不同的格式进行不同的处理，需要考虑的因素非常多（甚至还会出现 null、nan 等诡异错误，都需要手动一一处理）。此外，还有如职位描述与职位种类等数据，直接很多都是前程无忧网的用户自定义的字符串，完全无法通过直接的分类或独热等方法进行预处理，因此只能用 re 字符串匹配等方式，首先提取特征，再进行预处理。除此之外，前程无忧网数据集的数据属性也分为数值型、独热型、多类别型等不同属性类别，对于不同类的数据要使用完全不同的预处理方式，同一类之间也要根据实际情况进行一定的区别对待，考虑到因素非常复杂并且现实，收获很大。

还有一点很重要，就是我对于数据挖掘，尤其是数据预处理的看法有了极大的改观。我此前一直觉得算法是最重要的，因此自学和上课听讲学习了很多各类机器学习的方法，但却一直看不起或者忽略数据的预处理这一块。经过本项目的实践，我逐渐意识到，在实际的项目中，数据预处理的重要性和难度丝毫不低于算法，算法很大程度上也只是对预处理提取出来的信息的进一步拟合和提取罢了。因此，数据学科中切不可眼高手低，要脚踏实地，真正实践，才能成为数据挖掘的高手。

最后，还要谢谢范磊老师和我的队友提供的无私的帮助，谢谢各位！

7.1.4 张晴钊

本次作业我主要负责对抓取到的数据做数据分析，包括特征工程、相关性分析等。

任务的主要工作量在于数据的可视化处理。一方面需要使用工具绘制各种各样的图表，另一方面需要从未处理完成的数据中分析并选择存在关联的特征。对于可视化处理，主要以 python 工具为主，包括 pandas 做统计，matplotlib 和 seaborn 绘制图表等。另外对于其他特殊的图表，比如地图的绘制，则利用了 echart 等尚不熟悉的工具。在数据分析方面，首先从相关系数分析开始，快速锁定学历要求和工作经验要求两个特征与预期工资相关性高，从而进一步分析其分布。另外多特征的相关性分析也能得到有趣的结论。另外，我的一个主要工作是分析原始数据的数值分布，从而了解数据集的具体情况，可以作为后续预处理和模型训练的参考。

完成作业过程中也遇到了诸多困难并最终克服。除了工具使用还不够熟悉以外，尚未数值化、归一化的数据存在分析的难度。比如地域分布特征，数据集的地域精确到区县一级，种类庞杂，分析和绘图过程中只使用了市一级的数据。再比如工资下限和工资上限，即使统一到年薪，仍然存在如实习生的底薪和接近千万的高薪。在分析之前我去除了这些离群点和异常点，进而可以得到可靠的工资分布。

本次作业让我体会到数据分析时需要注意的细节和可能造成的误差。比如不同城市的平均工资，珠海平均工资远超北上广，是出乎我们常识预料的。实际上珠海的职位信息只有二十余例，而北上广深职位上万，相比之下，珠海的平均工资数据并不可靠。诸如此类。

大作业的完成离不开老师的指导和同组同学的密切配合谢谢老师和各位同学。

7.1.5 沙金锐

本项目中我主要负责论文撰写部分。由于已经完成了第一个大作业，自己对论文的整体架构和逻辑顺序也都有着一定的熟练度。本学期的两次大作业让自己第一次接触数据挖掘方面相关的文章，以前更多是在数模比赛的论文上的个别地方用到，寥寥数笔进行介绍，没有对数据预处理、特征性分析、特征值分布等多个方面进行如此细致的可视化处理。完成作业期间，自己也寻找了一些会议的相关 paper，去看他们的文章结构是如何完成的，都要在各个方面完成哪些内容。

通过本学期两次大作业的完成，自己也知道了数据挖掘领域的文章的整体的架构应如何撰写，让自己未来在接触这个领域内的 paper 时能更加熟悉。最后非常感谢范老师一个学期的辛勤指导，让我在这门课上收获很多，同时也感谢队友们对我的帮助以及彼此间默契的配合，让大作业完成的更好。

7.1.6 谌仲威

本项目中我主要负责论文撰写、项目的总结以及展示的制作。在本次项目中，我收获最大的是数据爬取和数据预处理的相关步骤。由于此前没有接触过数据爬取，对数据的爬取了解很浅，通过王磊同学的爬取工作和源码讲解，我对数据爬取的流程和 scrapy 的使用有了一定程度的了解。由于此前一次的作业中，数据预处理的内容相比于这次项目远远较少，本次数据预处理的步骤和内容较为复杂，数值型、独热型以及多类别型数据的考虑及预处理步骤让我对数据预处理有了进一步的理解。

通过本学期两次作业，我对数据挖掘了基本步骤以及各步骤的实现均有了一定程度的了解，感谢小组其他成员给予的帮助和教导。尤其感谢王靖康组长，他在任务的分配及工作流程

安排中让小组成员密切合作，各自发挥长处起到了重大的作用。最后感谢范老师在课上知识的传授与课后的帮助。

7.2 程序源代码

由于完整代码较多，这里仅列出部分核心代码。

网络爬虫

```
import scrapy
from ..items import Job51SpiderItem

class A51jobSpider(scrapy.Spider):
    name = '51job'

    def start_requests(self):
        urls = []
        # 生成搜索结果页所有的url
        for i in range(2000):
            urls.append(
                'https://search.51job.com/list/010000%252C020000%252
                    C030200%252C040000%252C080200,000000,0100%252C2400
                    %252C2500%252C2600%252C2700,00,9,99,%2B,2,'+str(i+1)+
                    '.html?lang=c&stype=1&postchannel=0000&workyear=99&
                    cotype=99&degreefrom=99&jobterm=99&companysize=99&
                    lonlat=0%2C0&radius=-1&ord_field=0&confirmdate=9&
                    fromType=&dibiaoid=0&address=&line=&specialarea=00&
                    from=&welfare='
            )

        # url访问
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        # 获得搜索结果页中每一个职位详情页url并访问
        divs = response.xpath('//div[@id="resultList"]/div[@class="el"]/p
            /span/a/@href').extract()
        for div in divs:
            yield scrapy.Request(url=div, callback=self.getDetails)

    def getDetails(self, response):
        try:
            # 在详情页中利用XPath抓取各类属性信息
            item = Job51SpiderItem()
            item['job']=response.xpath('//div[@class="tHeader tHjob"]//h1
                /text()')[0].extract()
            item['area']=response.xpath('//div[@class="tHeader tHjob"]//
                span[@class="lname"]/text()')[0].extract()
```

```
            item['company']=response.xpath('//p[@class="cname"]/a/text()'
                )[0].extract()

            tmp = response.xpath('//p[@class="msg ltype"]/text()').
                extract_first().split('|')
            item['company_type']=tmp[0].strip()
            item['company_people']=tmp[1].strip()
            item['company_service']=tmp[2].strip()
            item['salary']=response.xpath('//div[@class="tHeader tHjob
                "]//div[@class="cn"]/strong/text()')[0].extract()
            item['welfare']=response.xpath('//p[@class="t2"]/span/text()'
                ).extract()
            item['work_experience']=response.xpath('//div[@class="t1"]//
                em[@class="i1"]/../text()').extract_first("无")
            item['education']=response.xpath('//div[@class="t1"]//em[
                @class="i2"]/../text()').extract_first("无")
            item['hire_num']=response.xpath('//div[@class="t1"]//em[
                @class="i3"]/../text()').extract_first("无")
            item['put_time']=response.xpath('//div[@class="t1"]//em[
                @class="i4"]/../text()').extract_first("无")
            item['language']=response.xpath('//div[@class="t1"]//em[
                @class="i5"]/../text()').extract_first("无")
            item['demand_profession']=response.xpath('//div[@class="t1
                "]//em[@class="i6"]/../text()').extract_first("无")

            item['job_info'] = ''.join(response.xpath(
                '//div[@class="bmsg job_msg inbox"]/p/text() | //div[
                    @class="bmsg job_msg inbox"]/div/text()').extract())
            item['job_info'] = item['job_info'].strip()
            item['job_work_type'] = response.xpath(
                '//div[@class="bmsg job_msg inbox"]/div[@class="mt10"]//
                    span[contains(text(),"职能类别")]/../span[@class="el
                    "]/text()').extract()
            item['job_keyword'] = response.xpath(
                '//div[@class="bmsg job_msg inbox"]/div[@class="mt10"]//
                    span[contains(text(),"关键字")]/../span[@class="el"]/
                    text()').extract()
            yield item

        except:
            pass
```

数据可视化

```
def drawpearson(corr):
    colormap = plt.cm.viridis
    plt.figure(figsize=(12,12))
    plt.title('Pearson Correlation of Features', y=1.05, size=15)
    sns.heatmap(
```

```
        corr,linewidths=0.1,vmax=1.0, square=True, cmap=colormap,
            linecolor='white', annot=True)
    plt.savefig("test1.pdf")
```

数据预处理-生成年薪

```
    # append the maximum salary per year and minimum salary per year
    minimum = []
    maximum = []
    for i in range(len(dataFrame['salary'])):
        tmp = dataFrame['salary'][i]
        if re.compile('年').search(tmp):
            yearTag = 1    # year salary
        elif re.compile('天').search(tmp):
            yearTag = 300    # day salary
        else:
            yearTag = 12   # month salary

        if re.compile('千').search(tmp):
            multiTag = 1000   # count as 1000
        elif re.compile('元').search(tmp):
            multiTag = 1   # count as 1
        else:
            multiTag = 10000   # count as 10000

        numberIndex = max(tmp.find("千"), tmp.find("元"), tmp.find("万"))
        numberString = tmp[0:numberIndex]
        numberList = numberString.split("-")
        if len(numberList) == 1:
            minimum.append(int(float(numberList[0]) * multiTag * yearTag)
                )
            maximum.append(int(float(numberList[0]) * multiTag * yearTag)
                )
        else:
            minimum.append(int( float(numberList[0]) * multiTag * yearTag
                ))
            maximum.append(int( float(numberList[1]) * multiTag * yearTag
                ))
     # append the maxinum and minimum salary in newDF
    newDataFrame.insert(loc=len(newDataFrame.columns), column="salary_max
        ", value=maximum)
    newDataFrame.insert(loc=len(newDataFrame.columns), column="salary_min
        ", value=minimum)
```

数据预处理-公司福利

```
    # process welfare types
    welfare = []   # name of all welfares
    welfareNum = []  # number of all welfares
```

```
    for i in range(len(dataFrame['welfare'])):
        if not isinstance(dataFrame['welfare'][i],str): # some no-str welfare should be processed
            continue
        tmp = dataFrame['welfare'][i].split(",")
        for item in tmp:
            if item not in welfare: # not in list, append
                welfare.append(item)
                welfareNum.append(1)
            else:
                welfareNum[welfare.index(item)] += 1 # already in list, number++

    finalWelfareList = []  # the final remaining welfare list
    for i in range(len(welfare)):
        if welfareNum[i] > 100:  # remain only welfare that shows up greater than 100
            finalWelfareList.append(welfare[i])

    welfareCollist = [] # the cols of welfares(one-hot)
    for i in range(len(finalWelfareList)):
        welfareCollist.append(len(dataFrame['welfare']) * [0]) # all zero at first
    for i in range(len(dataFrame['welfare'])):
        if not isinstance(dataFrame['welfare'][i],str):
            # some no-str welfare should be processed
            continue
        tmp = dataFrame['welfare'][i].split(",")
        for item in tmp:
            if item in finalWelfareList:
                welfareCollist[finalWelfareList.index(item)][i] = 1 # a welfare exists, add 1
    # add in newDataFrame
    for i in range(len(welfareCollist)):
        newDataFrame.insert(loc=len(newDataFrame.columns), column="welfare_" + finalWelfareList[i],
                            value=welfareCollist[i])
```

数据预处理-公司服务

```
   # process company service
    companyService = [] # all tags of company services
    for i in range(len(dataFrame['company_service_1'])):
        if dataFrame["company_service_1"][i] not in companyService and \
        dataFrame["company_service_1"][i] != "null" and \
        isinstance(dataFrame['company_service_1'][i], str):  # reject the effect of null and nan
            companyService.append(dataFrame['company_service_1'][i])
    for i in range(len(dataFrame['company_service_2'])):
```

```
        if dataFrame["company_service_2"][i] not in companyService and \
        dataFrame["company_service_2"][i] != "null" and \
        isinstance(dataFrame['company_service_2'][i], str):  # reject the
             effect of null and nan
            companyService.append(dataFrame['company_service_2'][i])
    print(companyService)
    print(len(companyService))
    companyServiceCollist = []  # the cols of languages(one-hot)
    for i in range(len(companyService)):
        companyServiceCollist.append(len(dataFrame['company_service_1'])
            * [0])  # all zero at first
    for i in range(len(dataFrame['company_service_1'])):
        # match services
        tmp = dataFrame['company_service_1'][i]
        if not isinstance(tmp, str):  # reject nan
            continue
        companyServiceCollist[companyService.index(tmp)][i] = 1
        tmp = dataFrame['company_service_2'][i]
        if not isinstance(tmp, str):   # reject nan
            continue
        companyServiceCollist[companyService.index(tmp)][i] = 1
    # add into newDataFrame
    for i in range(len(companyService)):
        newDataFrame.insert(loc=len(newDataFrame.columns), column="
            company_service_" + companyService[i],
                            value=companyServiceCollist[i])
```

数据预处理-职位类别

```

    # process job_work_type
    filterKeyWords = ['设计','经理','总监','技术','实习','开发","编辑","
        产品','工程','电子','商务',
                      '销售','硬件','软件","助理","文员","主管","算法","
                          其他","推广","美工","网络","网站",
                      "系统","项目","医","测试","翻译","运营","首席","前
                          端","SEO"]
    jobWorkTypeColist = []
    for i in range(len(filterKeyWords)):
        jobWorkTypeColist.append(len(dataFrame['job_work_type_1']) * [0])
              # all zero at first
    for i in range(len(dataFrame['job_work_type_1'])):
        for j in range(len(filterKeyWords)):
            if re.compile(filterKeyWords[j]).search(dataFrame['
                job_work_type_1'][i]):
                jobWorkTypeColist[j][i] = 1
            if isinstance(dataFrame['job_work_type_2'][i],str) and \
                    re.compile(filterKeyWords[j]).search(dataFrame['
                        job_work_type_2'][i]):
```

```
                jobWorkTypeColist[j][i] = 1
    # add into newDataFrame
    for i in range(len(filterKeyWords)):
        newDataFrame.insert(loc=len(newDataFrame.columns), column="
            job_work_type_" + filterKeyWords[i],
                                value=jobWorkTypeColist[i])
```

职位/薪资预测

```

def provider(filepath="../data/data.csv",
             is_regression=False,
             salary_pred="high",
             all_data=False):
    # read data from the csv file
    df =  pd.read_csv(filepath, header=0, encoding="gb2312")
    # print df.head(5)

    # preprocess the data [city, language, company, job, welfare]
    df_city = df.apply(lambda s: np.argmax(list(s[6:27])), axis=1)
    df_lan = df.apply(lambda s: np.argmax(list(s[174: 181])), axis=1)
    df_com = df.apply(lambda s: np.argmax(list(s[94: 104])), axis=1)
    df_job = df.apply(lambda s: np.argmax(list(s[241: 273])), axis=1)
    df_welfare = df.apply(lambda s: np.sum(list(s[104: 174])), axis=1)

    # merge the properties
    if is_regression:
        data = df.iloc[:,:4]
        if salary_pred == "high":
            labels = df.iloc[:,4]
        elif salary_pred == "low":
            labels = df.iloc[:,5]
        else:
            labels = df.iloc[:,4:6]
    else:
        data = df.iloc[:,:6]
    # print data.head(5)
    # print labels.head(5)

    data['city'] = df_city.values
    data['language'] = df_lan.values
    data['company'] = df_com.values
    if is_regression:
        data['job'] = df_job.values
    else:
        labels = df_job.values
    data['welfare'] = df_welfare.values
    print data.head(2)

```

```
41      # split the data
42      X_train, X_test, Y_train, Y_test = train_test_split(data.values,
            labels.values, test_size=0.2, random_state=42)
43      print X_train.shape, X_test.shape
44      print Y_train.shape, Y_test.shape
45      if all_data:
46          return data.values, labels.values
47      return X_train, X_test, Y_train, Y_test
48
49  def train():
50      X_train, X_test, Y_train, Y_test = salary_provider()
51      ada = AdaBoostRegressor()
52      rf = RandomForestRegressor()
53      bagging = BaggingRegressor()
54      grad = GradientBoostingRegressor()
55      svr = SVR()
56      bayes_ridge = BayesianRidge()
57      elastic_net = ElasticNet()
58      mlp = MLPRegressor(hidden_layer_sizes=(64, 128, 64), max_iter=1000)
59
60      regressors = [ada, rf, bagging, grad, svr, bayes_ridge, elastic_net,
            mlp]
61      regressor_names = ["AdaBoost", "Random␣Forest", "Bagging",
62                         "Gradient␣Boost", "SVR", "Bayesian␣Ridge",
63                         "Elastic␣Net", "MLPRegressor"]
64
65      #regressors = [mlp]
66      #regressor_names = ["MLP"]
67
68      for regressor, regressor_name in zip(regressors, regressor_names):
69          intime = time.time()
70          regressor.fit(X_train, Y_train)
71          Y_pred = regressor.predict(X_test)
72          print X_train.shape, Y_train.shape
73          print "-----------------------------------"
74          print time.time() - intime
75          print "For␣Regressor␣:␣", regressor_name
76          print "Mean␣Absolute␣Error␣:␣", metrics.mean_absolute_error(
                Y_test, Y_pred)
77          # print "Median Absolute Error : ",metrics.median_absolute_error(
                Y_test, Y_pred)
78          # print "Mean Squared Error : ",metrics.mean_squared_error(Y_test
                , Y_pred)
79          print "R2␣Score␣:␣", metrics.r2_score(Y_test, Y_pred)
80          print "--------------------------------\n"
```

References

[1] H. Abdi. The kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, pages 508–510, 2007.

[2] D. Basak, S. Pal, and D. C. Patranabis. Support vector regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

[3] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[4] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[5] T. Brott, H. P. Adams, C. P. Olinger, J. R. Marler, W. G. Barsan, J. Biller, J. Spilker, R. Holleran, R. Eberle, and V. Hertzberg. Measurements of acute cerebral infarction: a clinical examination scale. *Stroke*, 20(7):864–870, 1989.

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

[7] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[8] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[9] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[10] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[11] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[12] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Machine learning*, 42(3):287–320, 2001.

[13] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[14] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[15] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.