# 实验3：关联技术挖掘

## 实验目标

1. 掌握关联规则运用
2. 熟练运用LLM到函数调用的方法
3. 熟练运用LLM完成函数返回值到自然语言的转换

## 实验任务

1. 根据现有数据创建频繁集，并且在此基础上做关联规则。
2. 将关联规则作为知识库，根据用户提问，搜索关联规则，并且将结果以自然语言的形式答复给用户。我擅长java，请给我一些建议。答复应该包含和java强相关的技术以及java技术对应最多的岗位。
3. 创建技术-岗位的关联性函数
4. 实现用户使用自然语言完成三类问题的问答（输入自然语言，返回自然语言）：（1）我比较擅长xxx，请给我推荐一些关联技术（关联规则）（2）从事xxx岗位需要学什么？(从岗位查询技术)(3) 我会xxx，什么岗位比较适合我？（从技术查岗位）

## 实验步骤

## 计算关联规则

已经预处理好的数据集，除了部分域做了数字化外，还增加了一列技能列（skill\_list)。该列是由岗位信息（positionDetail)这个非结构化文本中抽取的所需技能清单。

调用Apriori算法挖掘频繁项集，推算关联规则。

## 自然语言到函数调用

#### 2.1 基于关联规则信息的大模型对话

找出与给定技能（skill）最相关的其他技术。首先打印出正在处理的技能，然后加载数据集。接着使用 getAsso 函数来找出与指定技能相关联的其他技术。

如果找到相关技术，它会将这些技术转换为一个字符串列表，并用逗号和空格连接起来。

|  |
| --- |
| def get\_asso\_skill(dataset, skill, prompt):  print(f"get\_asso\_skill: {skill}")  biga\_load = load\_apriori("apriori.bin")  s = getAsso(biga\_load, skill.lower(), 0.1)  print(f"{skill}对应的关联技术是：{s}")  if len(s) == 0:  return prompt  else:  s = list(s)  s = ', '.join(s)  return f"针对问题{prompt},我查询到关联性最高的技术是{str(s)}，请将查询结果组织成人类语言描述。可以附加一些学习建议。总字数不超过200字" |

#### 2.2 从技能查询岗位

计算与特定技能（skill）最相关的岗位名称，并返回与该技能最相关的前 count 个岗位。首先将技能名称转换为小写，然后遍历数据集 dataset 中的每一行，提取技能列表，并检查是否有与目标技能匹配的技能。如果匹配，它会统计该技能在每个岗位名称中的出现次数。

|  |
| --- |
| def cacu\_skill\_position\_wordcount(dataset, skill, prompt, count=2):  print(f"get\_asso\_skill: {skill}")  *# 将skill转为小写。* skill = skill.lower()  *# 构建字典，方便后续排序* postion\_dict = {}  *# 遍历dataset* for i in range(len(dataset)):  *# 将dataset中这一行的'skill\_list'的内容取出来，使用逗号进行切割。* skills = dataset['skill\_list'][i].split(',')  *# 遍历切割后的技能列表* for s in skills:  *# 如果当前技能与目标技能相同，则进行统计* if s.strip().lower() == skill:  *# 获取当前行的职位名称* position = dataset['positionName'][i]  *# 如果该职位在字典中，则将其词频加1，否则初始化为1* if position in postion\_dict:  postion\_dict[position] += 1  else:  postion\_dict[position] = 1  *# 将字典按照value排序，倒序排列，取前面count个成员* sorted\_positions = sorted(postion\_dict.items(), key=lambda x: x[1], reverse=True)[:count]  return f"针对问题{prompt},我查询到关联性最高的岗位是{str(sorted\_positions)}，请将查询结果组织成人类语言描述。可以附加一些建议。总字数不超过200字" |

#### 2.3 从岗位查询技能

找出与特定岗位名称最相关的技能。它首先通过比较函数 compare\_str来匹配数据集中的岗位名称，并将匹配的行存储在 postion\_data 列表中。

然后，它将这些匹配行中的技能列表合并到一个列表中。对于列表中的每个技能，函数去除空格并进行词频统计。最后，统计结果按照出现次数降序排序，并返回前 count 个技能及其对应的出现次数。

|  |
| --- |
| def cacu\_postion\_skill\_wordcount(dataset, postionName, prompt, count=4):  print(f"get\_asso\_skill: {postionName}")  postion\_data = []  *# 将postionName和数据集中'positionName'列匹配的行抽取出来，放入postion\_data* for index, row in dataset.iterrows():  if compare\_str(row['positionName'], postionName):  postion\_data.append(row)  *# 将postion\_data的'skill\_list'放入skill\_list* skill\_list = []  for row in postion\_data:  skill\_list.extend(row['skill\_list'].split(','))    *# 将skill\_list中的每个成员都用逗号切割，将所有切割出来的成员做词频统计* skill\_dict = {}  for skill in skill\_list:  skill = skill.strip() *# 去除前后空格* if skill:  if skill in skill\_dict:  skill\_dict[skill] += 1  else:  skill\_dict[skill] = 1  *# 将统计出来的词频字典倒序排列，取最高的count个成员返回* sorted\_skills = sorted(skill\_dict.items(), key=lambda x: x[1], reverse=True)[:count]   return f"针对问题{prompt},我查询到关联性最高的技术是{str(sorted\_skills)}，请将查询结果组织成人类语言描述。可以附加一些建议。总字数不超过200字" |

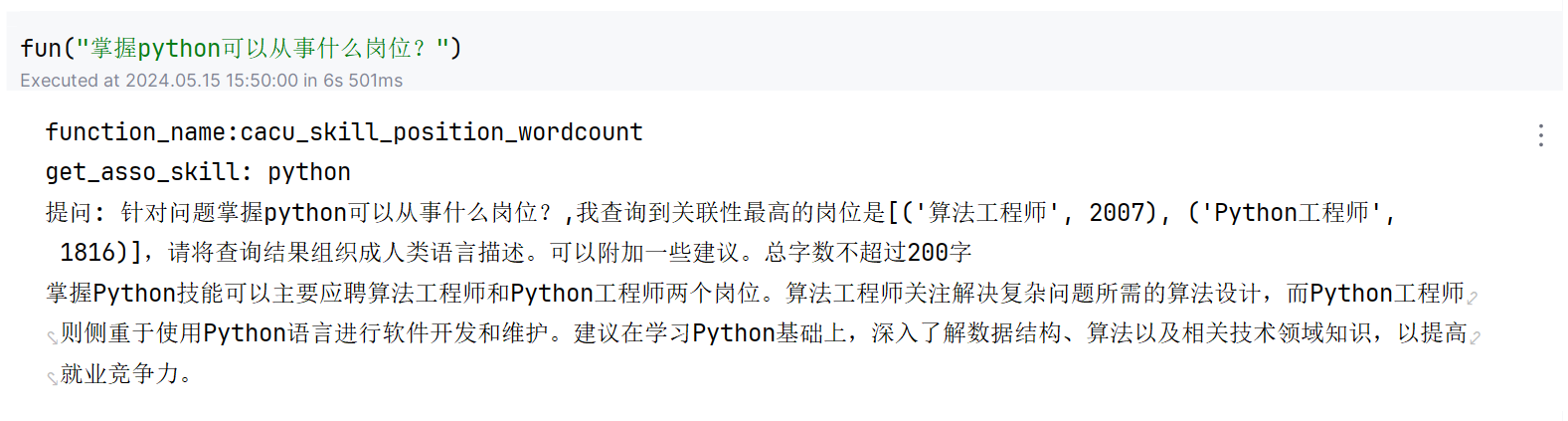
## 测试结果

这里至少测试三个用例，覆盖三个函数调用。

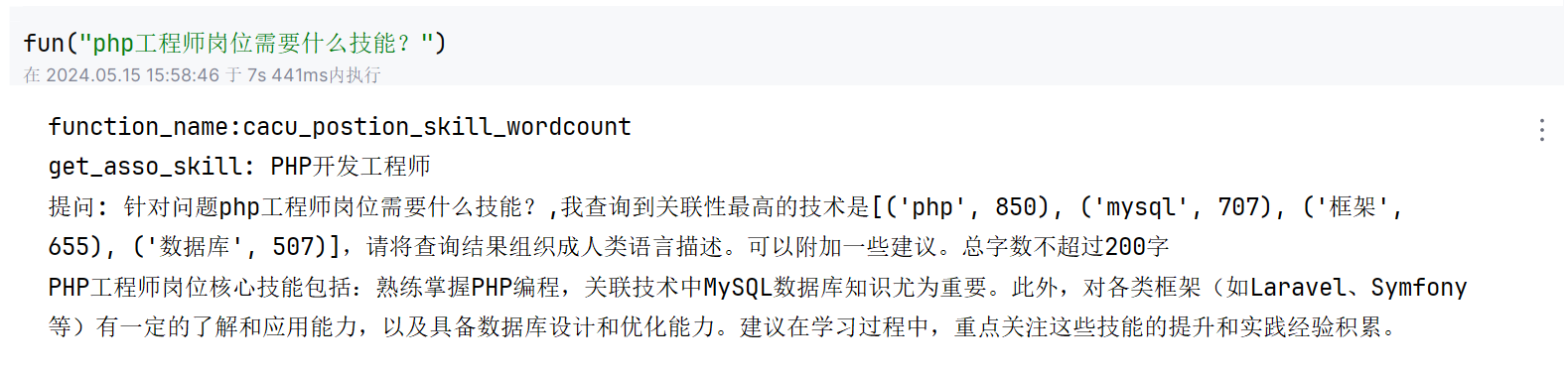
1. get\_asso\_skill调用测试：



1. cacu\_skill\_position\_wordcount调用测试：



1. cacu\_postion\_skill\_wordcount调用测试：



## 实验总结

本次实验过程遇到了哪些问题，如何解决的，有什么心得和收获。