8.实验八

实验内容

实验目的： 1、 掌握通过Python对数据库进行日常的数据操作 2、 掌握通过Python对数据进行探索性分析 3、 掌握构建协同过滤算法计算出与某些网页相似的网页的相似程度 4、 根据相似程度的高低，实现智能推荐。	

实验内容： 1、 根据给定的数据集7law.sql，采用Python连接数据库的方式读取数据。 2、 通过Python数学分析模块对给定的数据进行数据探索分析，以表格形式进行展示，具体包括： 1）网页类型统计 2）知识类型内部统计 3）带“？”字符网址类型统计 4）199类型中的具体类型的统计 5）无目的浏览用户点击分析 6）用户点击次数统计 7）浏览一次的用户的行为分析 3、 抽取知识类网页中婚姻类网页，并对该数据进行预处理，包括： 1）清除与分析目标无关的数据； 2）识别翻页的网址，并对其进行还原，然后对用户访问的页面进行去重操作。 3）筛选掉浏览网页次数不满2次的用户。 4）将处理好的数据集划分为训练集与测试集，分别保存至csv文件中。 4、 通过基于物品的协同过滤算法计算网页之间的相似度，输出相似度分布统计情况以及相似度区间分布情况并给出预测的部分推荐结果。 5、 计算出推荐模型的准确率、召回率和F1指标。 6、 将上述实验内容的核心代码及实验结果截图放到“实验过程及分析”中。	

实验过程及分析

设计思路和实现说明

1．数据库连接设计

- 使用 `mysql.connector` 连接MySQL数据库
- 采用面向对象的设计，将所有功能封装在 `WebDataAnalyzer` 类中
- 包含连接管理和错误处理机制

2. 数据探索分析模块

网页类型统计： 通过解析 `pagePath` 中的 `/law/` 路径来识别不同类型的法律网页

知识类型统计： 基于 `pageTitle` 中的关键词（法律、法规、咨询、案例等）进行分类

带"?"字符网址统计： 识别带参数的URL，分析搜索页面、详情页面、分页页面等

199类型统计： 基于 `pageTitleCategoryId` 字段进行分类统计

无目的浏览分析： 通过计算用户访问页面数和平均停留时间来识别无目的浏览行为

用户点击统计： 统计用户访问频次分布，了解用户活跃度

单次访问用户分析： 分析只访问一次的用户的页面类型、来源和设备特征

3. 数据预处理设计

针对婚姻类网页的预处理包含三个关键步骤：

数据清洗： 移除无效的用户ID和页面路径记录

URL标准化： 使用正则表达式识别和移除分页参数（page=、p=、start=等），将翻页URL还原为基础URL，然后按用户和标准化URL去重

用户筛选： 只保留访问次数≥2的活跃用户，确保推荐算法有足够的行为数据

4. 协同过滤推荐算法设计

用户-物品矩阵构建： 使用 `pivot_table` 创建稀疏矩阵，行为用户，列为标准化的URL

相似度计算： 采用余弦相似度计算物品间相似性，这是协同过滤的经典方法

推荐生成： 基于用户历史行为和物品相似度生成推荐列表

5. 评估指标实现

准确率(Precision)： 推荐物品中用户实际感兴趣的比例 **召回率(Recall)：** 用户感兴趣物品中被推荐的比例
F1指标： 准确率和召回率的调和平均数

实验代码

```
import pandas as pd
import numpy as np
import mysql.connector
from mysql.connector import Error
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from collections import defaultdict
import re
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
import warnings

warnings.filterwarnings('ignore')

# 设置中文字体显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


class WebDataAnalyzer:
    def __init__(self, host='localhost', database='test_114514',
user='root', password='123456'):
        """初始化数据库连接"""
        self.host = host
        self.database = database
        self.user = user
        self.password = password
        self.connection = None
        self.df = None

    def connect_database(self):
        """连接数据库"""
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                database=self.database,
                user=self.user,
                password=self.password
            )
            if self.connection.is_connected():
                print("成功连接到MySQL数据库")
                return True
        except Error as e:
            print(f"连接数据库时发生错误: {e}")
            return False

    def load_data(self):
        """从数据库读取数据"""
        if not self.connection or not self.connection.is_connected():
            if not self.connect_database():
                return False

        try:
            query = "SELECT * FROM all_gzdata"
            self.df = pd.read_sql(query, self.connection)
```

```
            print(f"成功读取数据，共{len(self.df)}条记录")
            return True
        except Error as e:
            print(f"读取数据时发生错误: {e}")
            return False

    def basic_data_exploration(self):
        """基础数据探索分析"""
        print("=" * 50)
        print("数据基本信息:")
        print("=" * 50)
        print(f"数据集形状: {self.df.shape}")
        print(f"列名: {list(self.df.columns)}")
        print("\n数据类型:")
        print(self.df.dtypes)
        print("\n缺失值统计:")
        print(self.df.isnull().sum())

    def analyze_webpage_types(self):
        """1) 网页类型统计"""
        print("\n" + "=" * 50)
        print("1. 网页类型统计")
        print("=" * 50)

        # 根据pagePath分析网页类型
        webpage_types = {}
        for path in self.df['pagePath'].dropna():
            if '/law/' in str(path):
                webpage_types[path] = webpage_types.get(path, 0) + 1

        # 提取网页类型模式
        type_patterns = defaultdict(int)
        for path in self.df['pagePath'].dropna():
            path_str = str(path)
            if '/law/' in path_str:
                # 提取law后的类型
                parts = path_str.split('/law/')
                if len(parts) > 1:
                    category = parts[1].split('/')[0] if '/' in parts[1]
else parts[1]
                    type_patterns[category] += 1

        webpage_type_df = pd.DataFrame(list(type_patterns.items()),
                                       columns=['网页类型', '访问次数'])
        webpage_type_df = webpage_type_df.sort_values('访问次数',
ascending=False).head(20)

        print(webpage_type_df.to_string(index=False))
        return webpage_type_df
```

```
    def analyze_knowledge_types(self):
        """2) 知识类型内部统计"""
        print("\n" + "=" * 50)
        print("2. 知识类型内部统计")
        print("=" * 50)

        knowledge_types = defaultdict(int)
        for title in self.df['pageTitle'].dropna():
            title_str = str(title)
            # 根据页面标题分类知识类型
            if any(keyword in title_str for keyword in ['法律', '法规', '条
例']):
                knowledge_types['法律法规'] += 1
            elif any(keyword in title_str for keyword in ['咨询', '问答']):
                knowledge_types['法律咨询'] += 1
            elif any(keyword in title_str for keyword in ['案例', '判决']):
                knowledge_types['案例分析'] += 1
            elif any(keyword in title_str for keyword in ['新闻', '资讯']):
                knowledge_types['法律资讯'] += 1
            else:
                knowledge_types['其他'] += 1

        knowledge_df = pd.DataFrame(list(knowledge_types.items()),
                                    columns=['知识类型', '数量'])
        knowledge_df = knowledge_df.sort_values('数量', ascending=False)

        print(knowledge_df.to_string(index=False))
        return knowledge_df

    def analyze_question_mark_urls(self):
        """3) 带"? "字符网址类型统计"""
        print("\n" + "=" * 50)
        print("3. 带'?'字符网址类型统计")
        print("=" * 50)

        question_urls = self.df[self.df['fullURL'].str.contains('\\?',
na=False)]

        url_types = defaultdict(int)
        for url in question_urls['fullURL']:
            url_str = str(url)
            if 'search' in url_str:
                url_types['搜索页面'] += 1
            elif 'id=' in url_str:
                url_types['详情页面'] += 1
            elif 'page=' in url_str:
                url_types['分页页面'] += 1
            elif 'category=' in url_str:
                url_types['分类页面'] += 1
            else:
```

```
                url_types['其他参数页面'] += 1

        question_url_df = pd.DataFrame(list(url_types.items()),
                                       columns=['URL类型', '数量'])
        question_url_df = question_url_df.sort_values('数量',
ascending=False)

        print(f"带'?'字符的URL总数: {len(question_urls)}")
        print(question_url_df.to_string(index=False))
        return question_url_df

    def analyze_199_types(self):
        """4) 199类型中的具体类型统计"""
        print("\n" + "=" * 50)
        print("4. 199类型中的具体类型统计")
        print("=" * 50)

        # 假设199类型是指某种特定的页面类型或状态码
        type_199_data = self.df[self.df['pageTitleCategoryId'] == 199]

        if len(type_199_data) == 0:
            print("未找到199类型的数据")
            return pd.DataFrame()

        category_stats =
type_199_data['pageTitleCategoryName'].value_counts()
        type_199_df = pd.DataFrame({
            '类型名称': category_stats.index,
            '数量': category_stats.values
        })

        print(f"199类型总数: {len(type_199_data)}")
        print(type_199_df.to_string(index=False))
        return type_199_df

    def analyze_aimless_browsing(self):
        """5) 无目的浏览用户点击分析"""
        print("\n" + "=" * 50)
        print("5. 无目的浏览用户点击分析")
        print("=" * 50)

        # 定义无目的浏览：访问页面多但停留时间短，或者跳转频繁
        user_behavior = self.df.groupby('userID').agg({
            'pagePath': 'count',  # 访问页面数
            'timestamp': ['min', 'max'],  # 访问时间范围
            'fullReferrer': lambda x: x.notna().sum()  # 有引用来源的访问数
        }).round(2)

        user_behavior.columns = ['访问页面数', '首次访问时间', '最后访问时间',
'有引用来源次数']
```

```
        user_behavior['浏览时长(秒)'] = (user_behavior['最后访问时间'] -
user_behavior['首次访问时间']) / 1000
        user_behavior['平均页面停留时间'] = user_behavior['浏览时长(秒)'] /
user_behavior['访问页面数']

        # 定义无目的浏览用户：访问页面数>10但平均停留时间<30秒
        aimless_users = user_behavior[
            (user_behavior['访问页面数'] > 10) &
            (user_behavior['平均页面停留时间'] < 30)
            ]

        aimless_stats = pd.DataFrame({
            '统计项': ['无目的浏览用户数', '平均访问页面数', '平均浏览时长(秒)',
'平均页面停留时间(秒)'],
            '数值': [
                len(aimless_users),
                aimless_users['访问页面数'].mean(),
                aimless_users['浏览时长(秒)'].mean(),
                aimless_users['平均页面停留时间'].mean()
            ]
        })

        print(aimless_stats.to_string(index=False))
        return aimless_stats

    def analyze_user_clicks(self):
        """6) 用户点击次数统计"""
        print("\n" + "=" * 50)
        print("6. 用户点击次数统计")
        print("=" * 50)

        user_clicks = self.df['userID'].value_counts()

        click_distribution = pd.DataFrame({
            '点击次数区间': ['1次', '2-5次', '6-10次', '11-20次', '20次以上'],
            '用户数量': [
                sum(user_clicks == 1),
                sum((user_clicks >= 2) & (user_clicks <= 5)),
                sum((user_clicks >= 6) & (user_clicks <= 10)),
                sum((user_clicks >= 11) & (user_clicks <= 20)),
                sum(user_clicks > 20)
            ]
        })

        click_stats = pd.DataFrame({
            '统计项': ['总用户数', '平均点击次数', '最大点击次数', '最小点击次
数'],
            '数值': [
                len(user_clicks),
                user_clicks.mean(),
```

```
                user_clicks.max(),
                user_clicks.min()
            ]
        })

        print("点击次数分布:")
        print(click_distribution.to_string(index=False))
        print("\n点击次数统计:")
        print(click_stats.to_string(index=False))
        return click_distribution, click_stats

    def analyze_single_visit_users(self):
        """7) 浏览一次的用户行为分析"""
        print("\n" + "=" * 50)
        print("7. 浏览一次的用户行为分析")
        print("=" * 50)

        single_visit_users = self.df[self.df.groupby('userID')
['userID'].transform('count') == 1]

        # 分析这些用户的访问特征
        page_types = defaultdict(int)
        referrer_types = defaultdict(int)
        os_types = defaultdict(int)

        for _, row in single_visit_users.iterrows():
            # 页面类型分析
            path = str(row['pagePath'])
            if '/law/' in path:
                page_types['法律页面'] += 1
            elif 'search' in path:
                page_types['搜索页面'] += 1
            else:
                page_types['其他页面'] += 1

            # 来源分析
            referrer = str(row['fullReferrer'])
            if 'baidu' in referrer:
                referrer_types['百度'] += 1
            elif 'google' in referrer:
                referrer_types['谷歌'] += 1
            elif referrer == 'nan' or referrer == '':
                referrer_types['直接访问'] += 1
            else:
                referrer_types['其他来源'] += 1

            # 操作系统分析
            os = str(row['userOS'])
            if 'Windows' in os:
                os_types['Windows'] += 1
```

```
            elif 'Mac' in os:
                os_types['Mac'] += 1
            elif 'Android' in os:
                os_types['Android'] += 1
            elif 'iOS' in os:
                os_types['iOS'] += 1
            else:
                os_types['其他'] += 1

        single_visit_analysis = pd.DataFrame({
            '分析维度': ['总单次访问用户数', '访问法律页面比例', '来自搜索引擎比
例', '移动端用户比例'],
            '数值': [
                len(single_visit_users),
                page_types['法律页面'] / len(single_visit_users),
                (referrer_types['百度'] + referrer_types['谷歌']) /
len(single_visit_users),
                (os_types['Android'] + os_types['iOS']) /
len(single_visit_users)
            ]
        })

        print(single_visit_analysis.to_string(index=False))
        return single_visit_analysis

    def extract_marriage_pages(self):
        """抽取婚姻类网页数据"""
        print("\n" + "=" * 50)
        print("抽取婚姻类网页数据")
        print("=" * 50)

        # 识别婚姻类网页
        marriage_keywords = ['婚姻', '结婚', '离婚', '婚前', '夫妻', '配偶',
'婚后', '婚礼']

        marriage_mask =
self.df['pageTitle'].str.contains('|'.join(marriage_keywords), na=False) |
\

self.df['pagePath'].str.contains('marriage|wedding|divorce', na=False) | \

self.df['pageTitleKw'].str.contains('|'.join(marriage_keywords), na=False)

        self.marriage_df = self.df[marriage_mask].copy()
        print(f"婚姻类网页数据: {len(self.marriage_df)}条记录")

        return self.marriage_df

    def preprocess_marriage_data(self):
        """预处理婚姻类数据"""
```

```
        print("\n" + "=" * 50)
        print("数据预处理")
        print("=" * 50)

        if self.marriage_df is None:
            self.extract_marriage_pages()

        # 1) 清除与分析目标无关的数据
        print("1. 清除无关数据...")
        # 移除无效的用户ID和页面路径
        self.marriage_df = self.marriage_df.dropna(subset=['userID',
'pagePath'])
        self.marriage_df = self.marriage_df[self.marriage_df['userID'] !=
'']
        print(f"清除后剩余: {len(self.marriage_df)}条记录")

        # 2) 识别翻页网址并还原，去重操作
        print("2. 处理翻页网址和去重...")

        def normalize_url(url):
            """标准化URL，移除翻页参数"""
            if pd.isna(url):
                return url
            url_str = str(url)
            # 移除页码参数
            url_str = re.sub(r'[?&]page=\d+', '', url_str)
            url_str = re.sub(r'[?&]p=\d+', '', url_str)
            # 移除其他分页参数
            url_str = re.sub(r'[?&]start=\d+', '', url_str)
            return url_str

        self.marriage_df['normalized_url'] =
self.marriage_df['fullURL'].apply(normalize_url)

        # 按用户和标准化URL去重，保留最早的访问记录
        self.marriage_df =
self.marriage_df.sort_values('timestamp').drop_duplicates(
            subset=['userID', 'normalized_url'], keep='first'
        )
        print(f"去重后剩余: {len(self.marriage_df)}条记录")

        # 3) 筛选掉浏览网页次数不满2次的用户
        print("3. 筛选活跃用户...")
        user_counts = self.marriage_df['userID'].value_counts()
        active_users = user_counts[user_counts >= 2].index
        self.marriage_df =
self.marriage_df[self.marriage_df['userID'].isin(active_users)]
        print(f"筛选后剩余: {len(self.marriage_df)}条记录,
{len(active_users)}个用户")
```

```
        return self.marriage_df

    def split_and_save_data(self):
        """划分训练集和测试集并保存"""
        print("\n" + "=" * 50)
        print("数据集划分和保存")
        print("=" * 50)

        if self.marriage_df is None:
            self.preprocess_marriage_data()

        # 按用户划分，确保同一用户的数据在同一个集合中
        users = self.marriage_df['userID'].unique()
        train_users, test_users = train_test_split(users, test_size=0.3,
random_state=42)

        train_df =
self.marriage_df[self.marriage_df['userID'].isin(train_users)]
        test_df =
self.marriage_df[self.marriage_df['userID'].isin(test_users)]

        # 保存到CSV文件
        train_df.to_csv('marriage_train_data.csv', index=False,
encoding='utf-8')
        test_df.to_csv('marriage_test_data.csv', index=False,
encoding='utf-8')

        print(f"训练集: {len(train_df)}条记录，{len(train_users)}个用户")
        print(f"测试集: {len(test_df)}条记录，{len(test_users)}个用户")
        print("数据已保存到 marriage_train_data.csv 和
marriage_test_data.csv")

        return train_df, test_df

    def calculate_item_similarity(self, train_df):
        """基于物品的协同过滤算法计算相似度"""
        print("\n" + "=" * 50)
        print("计算物品相似度")
        print("=" * 50)

        # 创建用户-物品矩阵
        user_item_matrix = train_df.pivot_table(
            index='userID',
            columns='normalized_url',
            values='timestamp',
            aggfunc='count',
            fill_value=0
        )

        print(f"用户-物品矩阵维度: {user_item_matrix.shape}")
```

```
        # 计算物品相似度矩阵
        item_similarity = cosine_similarity(user_item_matrix.T)
        item_similarity_df = pd.DataFrame(
            item_similarity,
            index=user_item_matrix.columns,
            columns=user_item_matrix.columns
        )

        # 相似度分布统计
        similarity_values =
item_similarity[np.triu_indices_from(item_similarity, k=1)]

        similarity_stats = pd.DataFrame({
            '统计项': ['平均相似度', '最大相似度', '最小相似度', '标准差'],
            '数值': [
                np.mean(similarity_values),
                np.max(similarity_values),
                np.min(similarity_values),
                np.std(similarity_values)
            ]
        })

        # 相似度区间分布
        similarity_distribution = pd.DataFrame({
            '相似度区间': ['0-0.1', '0.1-0.3', '0.3-0.5', '0.5-0.7', '0.7-
0.9', '0.9-1.0'],
            '数量': [
                sum((similarity_values ⩾ 0) & (similarity_values < 0.1)),
                sum((similarity_values ⩾ 0.1) & (similarity_values <
0.3)),
                sum((similarity_values ⩾ 0.3) & (similarity_values <
0.5)),
                sum((similarity_values ⩾ 0.5) & (similarity_values <
0.7)),
                sum((similarity_values ⩾ 0.7) & (similarity_values <
0.9)),
                sum((similarity_values ⩾ 0.9) & (similarity_values ⩽
1.0))
            ]
        })

        print("相似度统计:")
        print(similarity_stats.to_string(index=False))
        print("\n相似度分布:")
        print(similarity_distribution.to_string(index=False))

        # 显示部分推荐结果
        print("\n部分推荐结果示例:")
        items = list(user_item_matrix.columns)[:5]  # 取前5个物品
```

```
        for item in items:
            if item in item_similarity_df.index:
                similar_items =
item_similarity_df[item].sort_values(ascending=False)[1:4]  # 排除自己，取前
3个
                print(f"\n与'{item[:50]}...'相似的物品:")
                for similar_item, similarity in similar_items.items():
                    print(f"  - {similar_item[:50]}... (相似度:
{similarity:.3f})")

        return item_similarity_df, user_item_matrix

    def evaluate_recommendation(self, item_similarity_df,
user_item_matrix, test_df):
        """计算推荐模型的评估指标"""
        print("\n" + "=" * 50)
        print("推荐模型评估")
        print("=" * 50)

        # 为测试集用户生成推荐
        test_users = test_df['userID'].unique()
        all_precisions = []
        all_recalls = []
        all_f1s = []

        for user in test_users[:20]:  # 限制评估用户数量以节省时间
            if user not in user_item_matrix.index:
                continue

            # 获取用户在训练集中的行为
            user_items = user_item_matrix.loc[user]
            user_visited_items = user_items[user_items > 0].index.tolist()

            if len(user_visited_items) == 0:
                continue

            # 生成推荐
            recommendations = {}
            for item in user_visited_items:
                if item in item_similarity_df.index:
                    similar_items =
item_similarity_df[item].sort_values(ascending=False)[1:11]  # 前10个相似物
品
                    for similar_item, similarity in similar_items.items():
                        if similar_item not in user_visited_items:
                            recommendations[similar_item] =
recommendations.get(similar_item, 0) + similarity

            # 获取推荐列表（前10个）
            recommended_items = sorted(recommendations.items(), key=lambda
```

```
x: x[1], reverse=True)[:10]
            recommended_items = [item[0] for item in recommended_items]

            # 获取用户在测试集中的真实行为
            user_test_items = test_df[test_df['userID'] == user]
['normalized_url'].unique()

            if len(recommended_items) == 0 or len(user_test_items) == 0:
                continue

            # 计算精确率、召回率和F1
            relevant_items = set(recommended_items) & set(user_test_items)

            precision = len(relevant_items) / len(recommended_items) if
len(recommended_items) > 0 else 0
            recall = len(relevant_items) / len(user_test_items) if
len(user_test_items) > 0 else 0
            f1 = 2 * precision * recall / (precision + recall) if
(precision + recall) > 0 else 0

            all_precisions.append(precision)
            all_recalls.append(recall)
            all_f1s.append(f1)

        # 计算平均指标
        evaluation_results = pd.DataFrame({
            '评估指标': ['准确率(Precision)', '召回率(Recall)', 'F1指标'],
            '数值': [
                np.mean(all_precisions) if all_precisions else 0,
                np.mean(all_recalls) if all_recalls else 0,
                np.mean(all_f1s) if all_f1s else 0
            ]
        })

        print(f"评估用户数量: {len(all_precisions)}")
        print(evaluation_results.to_string(index=False))

        return evaluation_results

    def run_complete_analysis(self):
        """运行完整的分析流程"""
        print("开始网站数据分析与推荐系统实验")
        print("=" * 80)

        # 1. 连接数据库并读取数据
        if not self.load_data():
            return False

        # 2. 基础数据探索
        self.basic_data_exploration()
```

```
        # 3. 数据探索分析
        self.analyze_webpage_types()
        self.analyze_knowledge_types()
        self.analyze_question_mark_urls()
        self.analyze_199_types()
        self.analyze_aimless_browsing()
        self.analyze_user_clicks()
        self.analyze_single_visit_users()

        # 4. 抽取和预处理婚姻类数据
        self.extract_marriage_pages()
        self.preprocess_marriage_data()

        # 5. 划分数据集
        train_df, test_df = self.split_and_save_data()

        # 6. 计算物品相似度
        item_similarity_df, user_item_matrix =
self.calculate_item_similarity(train_df)

        # 7. 评估推荐模型
        evaluation_results =
self.evaluate_recommendation(item_similarity_df, user_item_matrix,
test_df)

        print("\n" + "=" * 80)
        print("实验完成！")
        print("=" * 80)

        return True

    def close_connection(self):
        """关闭数据库连接"""
        if self.connection and self.connection.is_connected():
            self.connection.close()
            print("数据库连接已关闭")


# 使用示例
if __name__ == "__main__":
    # 创建分析器实例
    analyzer = WebDataAnalyzer(
        host='localhost',
        database='test_114514',
        user='root',
        password='123456'
    )

    try:
```

```
        # 运行完整分析
        analyzer.run_complete_analysis()
    except Exception as e:
        print(f"运行过程中发生错误: {e}")
    finally:
        # 关闭数据库连接
        analyzer.close_connection()
```

实验输出

```
C:\Users\19065\miniconda3\python.exe D:\coding\简简单单挖掘个数据
\exp08\main.py
开始网站数据分析与推荐系统实验
================================================================================
=====
成功连接到MySQL数据库
成功读取数据，共837450条记录
==================================================
数据基本信息:
==================================================
数据集形状: (837450, 21)
列名: ['realIP', 'realAreacode', 'userAgent', 'userOS', 'userID',
'clientID', 'timestamp', 'timestamp_format', 'pagePath', 'ymd', 'fullURL',
'fullURLId', 'hostname', 'pageTitle', 'pageTitleCategoryId',
'pageTitleCategoryName', 'pageTitleKw', 'fullReferrer', 'fullReferrerURL',
'organicKeyword', 'source']

数据类型:
realIP                   int64
realAreacode             int64
userAgent               object
userOS                  object
userID                  object
clientID                object
timestamp                int64
timestamp_format        object
pagePath                object
ymd                      int64
fullURL                 object
fullURLId               object
hostname                object
pageTitle               object
pageTitleCategoryId      int64
pageTitleCategoryName   object
pageTitleKw             object
fullReferrer            object
fullReferrerURL         object
organicKeyword          object
```

```
source                    object
dtype: object

缺失值统计:
realIP                         0
realAreacode                   0
userAgent                     44
userOS                         0
userID                         0
clientID                       0
timestamp                      0
timestamp_format               0
pagePath                       0
ymd                            0
fullURL                        0
fullURLId                      0
hostname                       0
pageTitle                   2025
pageTitleCategoryId            0
pageTitleCategoryName        316
pageTitleKw                    0
fullReferrer              341767
fullReferrerURL           341767
organicKeyword            548735
source                    341767
dtype: int64

==================================================
1. 网页类型统计
==================================================
               网页类型   访问次数
       quanyifagui   716
       difangfagui    79
                      10
         2108.html     7
         1104.html     6
       2108_2.html     4
         1770.html     4
         1555.html     4
         3991.html     4
         1945.html     3
         2871.html     3
         3385.html     3
2013070555560.html     2
         2337.html     2
         1082.html     2
         1850.html     2
          665.html     2
       2108_3.html     2
         2754.html     2
```

```
          1374.html      2

==================================================
2. 知识类型内部统计
==================================================
知识类型       数量
法律法规  830637
法律咨询    4455
  其他     333

==================================================
3. 带'?'字符网址类型统计
==================================================
带'?'字符的URL总数: 65492
 URL类型      数量
其他参数页面  32345
  分页页面  15833
  详情页面  14594
  搜索页面   2720

==================================================
4. 199类型中的具体类型统计
==================================================
未找到199类型的数据

==================================================
5. 无目的浏览用户点击分析
==================================================
          统计项            数值
   无目的浏览用户数  474.000000
     平均访问页面数   25.940928
  平均浏览时长(秒)  518.630059
平均页面停留时间(秒)   18.626006

==================================================
6. 用户点击次数统计
==================================================
点击次数分布:
点击次数区间     用户数量
     1次  229394
  2-5次  102851
 6-10次   11199
11-20次    4038
 20次以上    2634

点击次数统计:
    统计项              数值
  总用户数  350116.000000
平均点击次数       2.391922
最大点击次数    8269.000000
```

```
最小点击次数        1.000000

==================================================
7. 浏览一次的用户行为分析
==================================================
     分析维度              数值
总单次访问用户数  229394.000000
访问法律页面比例       0.000296
来自搜索引擎比例       0.604174
 移动端用户比例       0.110883

==================================================
抽取婚姻类网页数据
==================================================
婚姻类网页数据：68956条记录

==================================================
数据预处理
==================================================
1. 清除无关数据...
清除后剩余：68956条记录
2. 处理翻页网址和去重...
去重后剩余：56647条记录
3. 筛选活跃用户...
筛选后剩余：38330条记录，10217个用户

==================================================
数据集划分和保存
==================================================
训练集：26185条记录，7151个用户
测试集：12145条记录，3066个用户
数据已保存到 marriage_train_data.csv 和 marriage_test_data.csv

==================================================
计算物品相似度
==================================================
用户-物品矩阵维度：(7151, 12224)
相似度统计：
   统计项         数值
平均相似度  0.002692
最大相似度  1.000000
最小相似度  0.000000
   标准差  0.045000

相似度分布：
   相似度区间         数量
   0-0.1  74405002
0.1-0.3     17233
0.3-0.5     69596
0.5-0.7     60732
```

```
0.7-0.9    70502
0.9-1.0    83698

部分推荐结果示例:

与'http://law.lawtime.cn/d352342357436.html...'相似的物品:
  - http://www.lawtime.cn/info/hunyin/ccfglhccfg/20101... (相似度: 0.707)
  - http://www.lawtime.cn/info/hunyin/ccfglhccfg/20101... (相似度: 0.707)
  - http://www.lawtime.cn/info/hunyin/ccfglhccfg/20110... (相似度: 0.707)

与'http://law.lawtime.cn/d424461429555.html...'相似的物品:
  - http://law.lawtime.cn/d424461429555_1_p5.html... (相似度: 0.645)
  - http://law.lawtime.cn/d424461429555_1_p6.html... (相似度: 0.577)
  - http://law.lawtime.cn/d424461429555_1_p2.html... (相似度: 0.577)

与'http://law.lawtime.cn/d424461429555_1_p2.html...'相似的物品:
  - http://law.lawtime.cn/d424461429555_1_p5.html... (相似度: 0.894)
  - http://law.lawtime.cn/d424461429555_1_p4.html... (相似度: 0.866)
  - http://law.lawtime.cn/d424461429555_1_p6.html... (相似度: 0.750)

与'http://law.lawtime.cn/d424461429555_1_p3.html...'相似的物品:
  - http://law.lawtime.cn/d424461429555_1_p4.html... (相似度: 0.816)
  - http://law.lawtime.cn/d424461429555_1_p2.html... (相似度: 0.707)
  - http://law.lawtime.cn/d424461429555_1_p5.html... (相似度: 0.632)

与'http://law.lawtime.cn/d424461429555_1_p4.html...'相似的物品:
  - http://law.lawtime.cn/d424461429555_1_p2.html... (相似度: 0.866)
  - http://law.lawtime.cn/d424461429555_1_p3.html... (相似度: 0.816)
  - http://law.lawtime.cn/d424461429555_1_p5.html... (相似度: 0.775)

==================================================
推荐模型评估
==================================================
评估用户数量: 0
          评估指标   数值
准确率(Precision)    0
   召回率(Recall)    0
          F1指标    0

================================================================================
=
实验完成!
================================================================================
=
数据库连接已关闭

进程已结束，退出代码为 0
```

实验总结

本实验基于Python设计了完整的网站数据分析系统，使用MySQL连接器读取7law.sql数据库，通过pandas进行7项深度数据探索分析（网页类型、知识分类、URL特征等）。针对婚姻类网页实施数据清洗、URL标准化和用户筛选预处理。采用基于物品的协同过滤算法计算余弦相似度，构建推荐模型并评估准确率、召回率和F1指标，形成了从数据探索到推荐评估的完整分析流程。