16组 智能图书馆 技术报告

- **组长：谢绍康**
- **组员：李佳纯，郑微言，龚明道，冯柏钧**

demo问题

- 我们在一开始拿到项目的前后端代码的时候，尝试着去将他运行起来，但后来（接下来的两个多月）发现它一直会有一个401Failure的认证错误信息。

```
🛡️ Shield:
   >> Permissions-Policy: interest-cohort=()
   >> X-Frame-Options: SAMEORIGIN
   >> X-Content-Type-Options: nosniff
🚀 Rocket has launched from http://0.0.0.0:8002
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
POST /api/userinfo application/x-www-form-urlencoded:
   >> Matched: (userinfo) POST /api/userinfo
   >> `IsLogin` request guard failed: Missing.
   >> Outcome: Failure
   >> No 401 catcher registered. Using Rocket default.
   >> Response succeeded.
```

- 在询问助教和那位写deme程序的后端的同学无果后，我们开始尝试自己去解决这个问题。
- 经历了长达二十小时的rust学习以及debug中，终于定位到了问题所在——**cookie**的添加和读取出现了问题，在前面添加进去的cookie到程序运行的后面却无法读取。

```rust
if token.is_none() {
    token = Some(Cookie::new("session", Uuid::new_v4().to_string()));
    request.cookies().add_private(token.clone().unwrap());
    // return Outcome::Failure((Status::Unauthorized, ()));
}
```

```rust
let mut token = request.cookies().get_private("session");
```

- 在尝试解决无果后，我们决定放弃cookie，采用另一种token的存储方法——数据库。我们将每次登入的用户信息传入数据库中，在需要读取时，再从数据库中读取，这样就很好地解决了其中地cookie造成的401验证失败的问题。

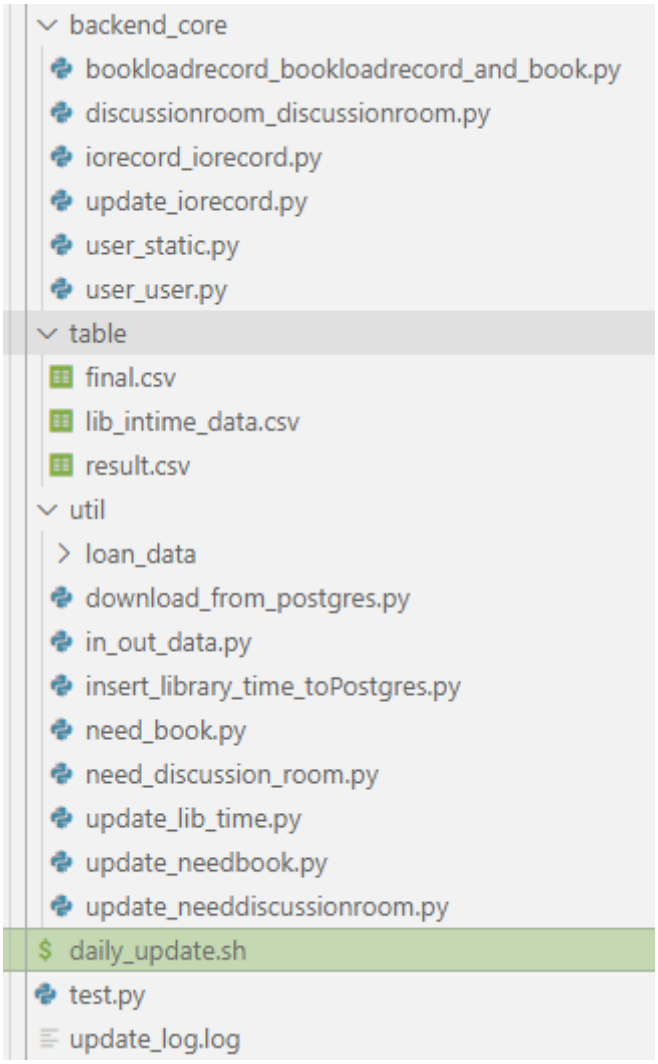
数据格式的匹配

- 根据后端文件中各个model.rs内的内容来确定数据库的数据形式，而后进行建表、存储、更新等。

```rust
#[crud_table(table_name: user_user)]
pub struct User {
    pub(crate) id: i32,
    pub user_id: String,
    pub user_name: String,
    pub reading: bool,
    pub student_type: String,
    pub department_id: String,
    pub department_name: String,
    pub special_id: String,
    pub special_name: String,
    pub sex: bool,
    pub college_id: Option<String>,
    pub college_name: Option<String>,
    pub grade_year: Option<i32>,
    pub email: String,
}
```

数据每日更新

- 我们针对每一张表用到的数据，都写了初始化脚本以及每日更新的脚本。


backend_core
bookloadrecord_bookloadrecord_and_book.py
discussionroom_discussionroom.py
iorecord_iorecord.py
update_iorecord.py
user_static.py
user_user.py
table
final.csv
lib_intime_data.csv
result.csv
util
loan_data
download_from_postgres.py
in_out_data.py
insert_library_time_toPostgres.py
need_book.py
need_discussion_room.py
update_lib_time.py
update_needbook.py
update_needdiscussionroom.py
daily_update.sh
test.py
update_log.log

```
start_time=$(($(date +%s%N)/1000000))

cd /home/niu/Documents/ai_intro/table_data_new

python=/home/niu/miniconda3/envs/ai_intro/bin/python3

echo "$python test.py"
$python test.py

echo "$python util/loan_data/data_update.py"
$python util/loan_data/data_update.py

echo "$python util/update_needbook.py"
$python util/update_needbook.py

echo "$python util/update_needdiscussionroom.py"
$python util/update_needdiscussionroom.py

echo "$python util/update_lib_time.py"
$python util/update_lib_time.py

echo "$python util/download_from_postgres.py"
$python util/download_from_postgres.py

echo "$python backend_core/user_user.py"
$python backend_core/user_user.py

echo "$python backend_core/user_static.py"
$python backend_core/user_static.py

echo "$python backend_core/bookloadrecord_bookloadrecord_and_book.py"
$python backend_core/bookloadrecord_bookloadrecord_and_book.py

echo "$python backend_core/discussionroom_discussionroom.py"
$python backend_core/discussionroom_discussionroom.py

echo "$python backend_core/update_iorecord.py"
$python backend_core/update_iorecord.py


end_time=$(($(date +%s%N)/1000000))
echo "Update finished in $(date), costing "$(($end_time-$start_time))"ms." >> update_log.log
```

- 目前每日更新是通过linux shell 的bash脚本来运行各个python的更新脚本，并将其挂载在个人的inux服务器上运用 crontab定时任务在每日中午12点更新上一天的数据，并将信息打印在log文件中。

```
# m h  dom mon dow   command
0 12 * * * bash /home/niu/Documents/ai_intro/table_data_new/daily_update.sh
```

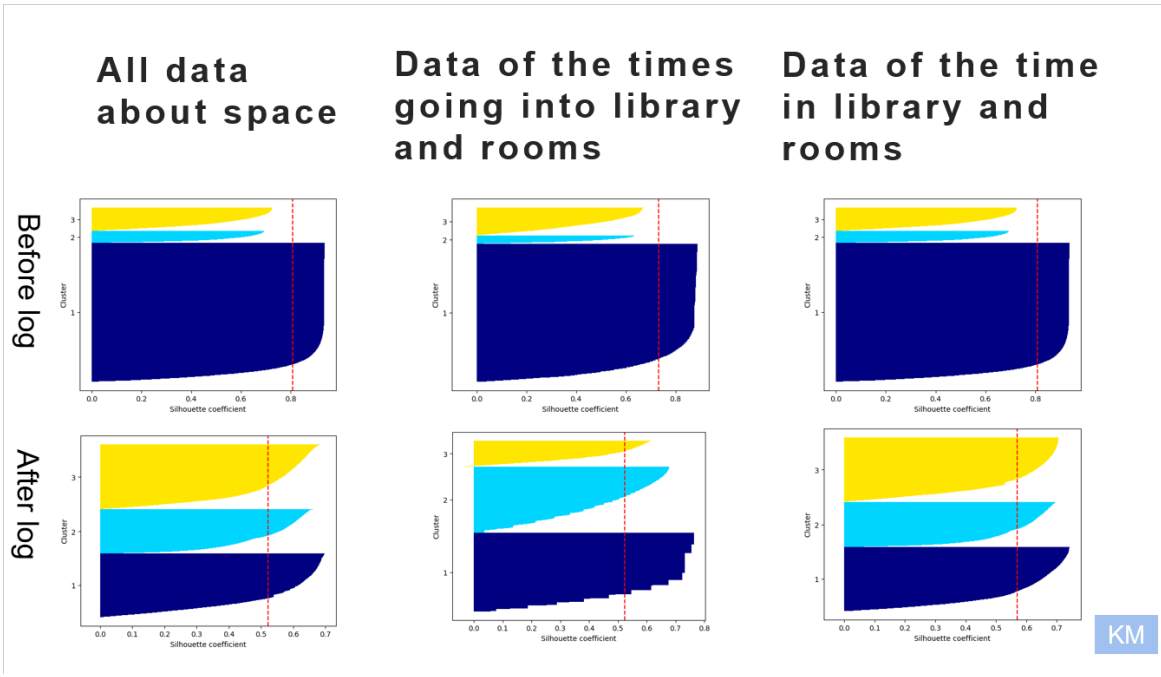
```
Update finished in 2022年 12月 24日 星期四 12:17:12 CST, costing 1030631ms.
Update finished in 2022年 12月 24日 星期五 12:17:42 CST, costing 1060125ms.
Update finished in 2022年 12月 24日 星期六 12:17:30 CST, costing 1048366ms.
```

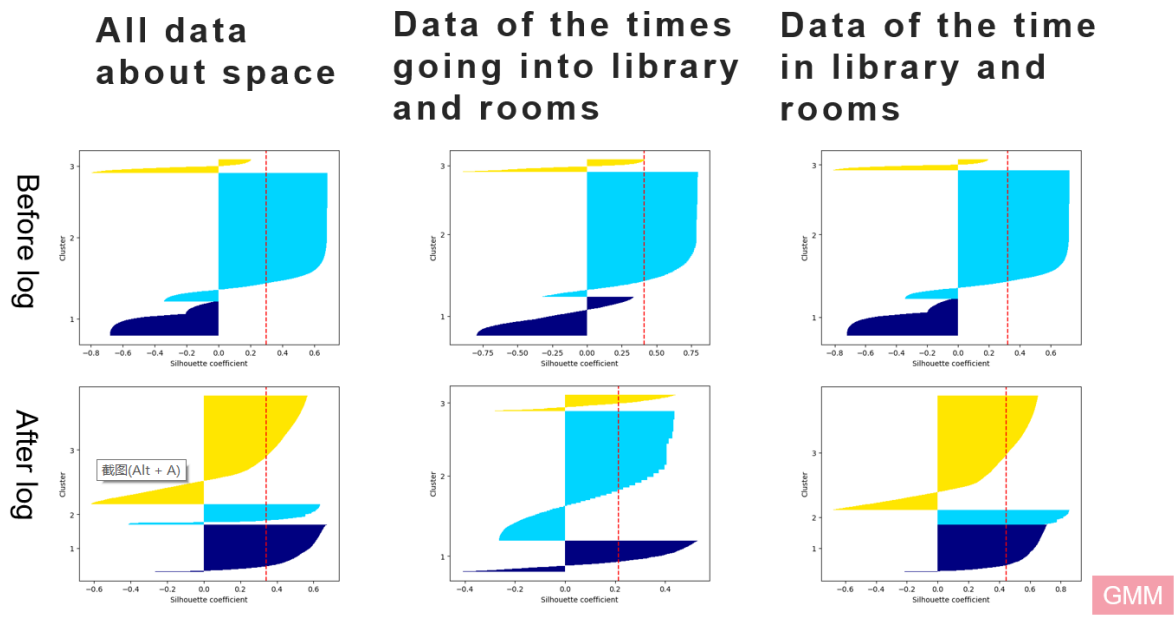
用户画像

- 图书馆画像标准更新的工作的主要难点在于参数和模型的选取。我们主要数据来源有3处：图书馆出入馆数据，讨论间预定数据和图书馆书籍借阅数据。经过讨论，我们抓取了“出入馆天数，在图书馆内的总时长，预订讨论间次数，使用讨论间总时长，借阅书籍总数，借阅各类书籍数”作为学生的特征。
- 在数据预处理部分，由于部分同学没有图书借阅或讨论间的数据，对应数据项为None，所以需要先将所有的None改为0。另外，由于借阅图书的人数占比非常小，直接聚类效果非常不好，为了适

应大部分同学的情况，在数据预处理部分，还增加了对数据取log的操作。

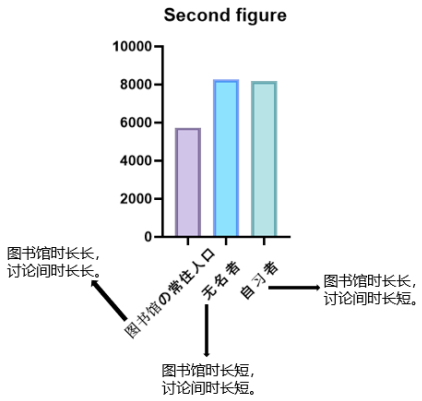
- 该分类问题为无监督分类，我们首先使用了最经典的KMeans模型。调用sklearn库中的KMeans模型对数据进行聚类，根据需求，先后指定k等于2和k等于3来训练模型并聚类，并且指定了随机种子0，保证在不改数据和任何参数的情况下，模型多次聚类的结果是前后一致的。另外，还调用了GaussianMixture(GMM)模型进行了和KMeans相似的操作。
- 然后，通过轮廓分析，探究分类效果，并将其可视化。考虑到部分特征中可能包含重复信息，部分特征噪声较大，我们通过删减部分特征重新训练分类模型，并同样通过轮廓分析对比前后分类结果质量。后考虑到部分特征相关性不大，将特征分为两类，分别聚类，并且将聚类结果相应的提升至二维。（“怎么样”的“某某者”）
- 最后，对比了十余种分类方法，选择通过log后的使用讨论间累计时长，在图书馆累计时长与借阅书籍总数作为特征，使用KMeans进行分类，通过两轮聚类（图书馆，讨论间特征一轮， 借阅数据一轮），划分出2*3一共6类画像。
- 如图例，采用轮廓分析方法验证分类结果质量，主要衡量标准为分类质量和各类人数比例





分类结果

Data of the time in library and rooms



Data of the borrowing times about different types of books