3_Interactive_session

December 10, 2023

0.0.1 Backend

[ ]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re
from surprise import dump
from surprise import Dataset, Reader, SVD, KNNBasic
from surprise.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from IPython.display import display
import ipywidgets as widgets
from surprise.model_selection import GridSearchCV
import tqdm

from proj_util import *

with open('./Model/search_TfidfVectorizer.pkl', 'rb') as f:
    vectorizer = pickle.load(f)
    assert(isinstance(vectorizer, TfidfVectorizer))
with open('./Model/search_TfidfVectorizer_en.pkl', 'rb') as f:
    vectorizer_en = pickle.load(f)
    assert(isinstance(vectorizer_en, TfidfVectorizer))

with open('./Data/search_tfidf.pkl', 'rb') as f:
    tfidf = pickle.load(f)
with open('./Data/search_tfidf_en.pkl', 'rb') as f:
    tfidf_en = pickle.load(f)

anime_with_synopsis = pd.read_feather('./Data/anime_with_synopsis.feather')
anime = pd.read_feather('./Data/anime.feather')
anime_for_search = pd.read_feather('./Data/anime_for_search.feather')

svd_big = dump.load('./Model/svd.pkl')[0]
```

```
focus_rating_2 = pd.read_feather('./Data/For_SVD/focus_rating_2.feather')
user_ids = focus_rating_2['user_id'].unique()

def search(keyword: str):
    return search_util(keyword, vectorizer, vectorizer_en, tfidf, tfidf_en,␣
 ↪anime_for_search)

svd_big = dump.load('./Model/svd_big.pkl')[0]
```

[ ]:

```
anime_ids = pd.Series(anime_for_search['MAL_ID'].unique())
anime_ids.index = anime_ids.values
```

Since SVD algorithm does not support "new" user per-se. We well calcualte the similarity from existing users

0.0.2 Search

[ ]:

```
search_widget = widgets.Text(
    value = '', description = 'Title'
)

search_btn = widgets.Button(
    description='Search',
    disabled=False,
    button_style='info',
    tooltip='Search',
    icon='search'
)

search_output = widgets.Output()

def search_event(sender):
    search_output.clear_output()
    x = search_widget.value
    if len(x) < 3:
            return
    with search_output:
        display(search(x))

search_btn.on_click(search_event)

display(widgets.VBox([widgets.HBox([search_widget, search_btn]),␣
 ↪search_output]))
```

```
VBox(children=(HBox(children=(Text(value='', description='Title'),␣
 ↪Button(button_style='info', description='Se…
[ ]: recommend_widget = widgets.BoundedIntText(value = None, min = 0, max =␣
      ↪max(user_ids), description = '')

     recooemnd_btn = widgets.Button(
         description='Recommend',
         disabled=False,
         button_style='info',
         tooltip='Search',
         icon='search'
     )

     current_ratings = widgets.Output()

     recommend_output = widgets.Output()

     def recommend_titles(_):
         q = recommend_widget.value
         if q in user_ids:
             u_id = q
         else:
             u_id = user_ids[recommend_widget.value]
         u_titles = focus_rating_2[focus_rating_2['user_id'] == u_id]
         current = u_titles \
             .join(anime_for_search.set_index('MAL_ID'), on='anime_id')\
             .sort_values(by=['rating', 'Avg. Score'], ascending=False)
         current = current.head(10).iloc[:, 3:]
         current.index = current['anime_id']
         current = current[['rating', 'Name', 'English name', 'Genres']]

         # recommend
         remaining_anime = anime_ids[~anime_ids.
      ↪isin(focus_rating_2[focus_rating_2['user_id'] == u_id]['anime_id'])]
         predicted_anime = pd.DataFrame(remaining_anime.apply(lambda x: svd_big.
      ↪predict(u_id, x)[3]).sort_values(ascending=False).head(100),
                                        columns=['Pred. rating']).
      ↪join(anime_for_search.set_index('MAL_ID'))\
                               .sort_values(by=['Pred. rating', 'Avg. Score'],␣
      ↪ascending=False)\
                               .head(10)

         predicted_anime.index.name = 'anime_id'

         current_ratings.clear_output()
         recommend_output.clear_output()

         with current_ratings:
             display(current)
```

```
    with recommend_output:
        display(predicted_anime[['Pred. rating', 'Name', 'English name',␣
 ↪'Genres']])

recoomend_btn.on_click(recommend_titles)

display(widgets.VBox([widgets.HBox([recommend_widget, recoomend_btn]),
                      widgets.Label('History'),
                      current_ratings,
                      widgets.Label('Recommend Output'),
                      recommend_output]))

# eg id 99580
```

```
VBox(children=(HBox(children=(BoundedIntText(value=0, max=353400),␣
 ↪Button(button_style='info', description='Re…
```

[ ]:

```
def plot_this(df: pd.DataFrame):
    plt.figure(figsize=(20, 10))
    # Use the barplot function from Seaborn
    ax = sns.barplot(x='Name', y='Pred. rating', data=df, errorbar=None,␣
 ↪width=0.5,
                    palette="crest")
    # Annotate each bar with the genre name
    for i, p in enumerate(ax.patches):
        ax.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
                    ha='center', va='center', xytext=(0, 10),␣
 ↪textcoords='offset points', fontsize=8, color='black')
        # Adding genre names inside the bars
        ax.text(p.get_x() + p.get_width() / 2., p.get_height() / 2., df['Pred.␣
 ↪rating'].iloc[i],
                ha='center', va='center',rotation=90, fontsize=12,␣
 ↪color='white', weight='bold')
    # Customize the plot
    plt.title('Top recommendation', fontsize=20)
    plt.xlabel('Genre', fontsize=18)
    plt.ylabel('Average User Rating', fontsize=18)
    ax.set_xticks([])
    ax.margins(x=0.1)

    return ax
```

[ ]:

```
```