Collaborative Anime Recommendation System

Question

How well can a model be made which recommends animes to other viewers based on the ratings of the user and others?

Process

- Identify distributions
- Filter out animes with few ratings and users with few ratings
- Remove scores with -1 as rating
- Cross validate models (did not have computational power for multiple models, stuck with SVD)
- Use best model to find predictions

Problem areas:

The logic of this filtering is sound, and the result is satisfying, but the ratings displayed are likely less accurate than they appear. Unfortunately, the dataset uses -1 to convey that a user has watched anime, but not rated it, placing the recommender system at an impossible crossroads. These can be dealt with in one of 3 ways:

- Remove all -1 values (results in animes being recommended to users who likely already enjoyed them)
- Replace all -1 values with mean/median (results in too many scores being rated average, and also brings down ratings of popular shows)
- Rate on a scale from -1 to 10 (just blatant misinformation as this logically makes no sense) Consequently, there is no perfect way to recommend animes with this dataset, but the removal of -1 values returned the best results.

RMSE - 1.1206

In [141…

```
#imports

from surprise import SVD, Dataset, Reader, SVDpp, SlopeOne, NMF, NormalPredictor, K
from surprise.model_selection import cross_validate, KFold, train_test_split
import pandas as pd
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import plotly.graph_objects as go
from plotly.offline import iplot
import kaggle

warnings.filterwarnings('ignore')
```

In [113…

```
#load data

kaggle.api.authenticate()
kaggle.api.dataset_download_files('maulipatel18/anime-content-based-recommendation-
anime_df = pd.read_csv('../data/anime.csv')
anime_df.head()
```

Out[113…

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

In [114…

```
rating_df = pd.read_csv('../data/rating.csv')
rating_df.head()
```

Out[114…

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

In [115…

```
#distribution of all rating scores

data = rating_df['rating'].value_counts()

trace = go.Bar(x=data.index, y=data.values)
```

```
layout = dict(
    title = 'Rating Distribution',
    xaxis = dict(title = 'Rating'),
    yaxis = dict(title = 'Count')
)

fig = go.Figure(data=[trace], layout=layout)
iplot(fig)
```

In [116…

```
#distribution of ratings per user

data = rating_df.groupby('user_id')['anime_id'].count().clip(upper=100)
trace = go.Histogram(x=data.values, name='Count Ratings', xbins=dict(start=0, end=1

layout = go.Layout(title = 'Distribution of Ratings per User (Capped at 100)',
                   xaxis = dict(title = 'Ratings Per User'),
                   yaxis = dict(title = 'Count'),
                   bargap = 0.2)

fig = go.Figure(data=[trace], layout=layout)
iplot(fig)
```

In [117…

```
#distribution of ratings per anime

data = rating_df.groupby('anime_id')['rating'].count().clip(upper=100)
trace = go.Histogram(x=data.values, name='Count Ratings', xbins=dict(start=0, end=1

layout = go.Layout(title = 'Distribution of Ratings per Anime (Capped at 100)',
                   xaxis = dict(title = 'Ratings Per Anime'),
                   yaxis = dict(title = 'Count'),
                   bargap = 0.2)

fig = go.Figure(data=[trace], layout=layout)
iplot(fig)
```

COLLABORATIVE RECOMMENDATIONS

These predictions will use the ratings and similar preferences between users to generate recommendations

In [118…

```
#OPTIONAL filter out sparsely rated animes and reviewers with few total reviews

min_anime_ratings = 40
filtered_ratings = rating_df['anime_id'].value_counts() > min_anime_ratings
filtered_ratings = filtered_ratings[filtered_ratings].index.to_list()

min_user_ratings = 30
filtered_users = rating_df['user_id'].value_counts() > min_user_ratings
filtered_users = filtered_users[filtered_users].index.tolist()

df_clean = rating_df[(rating_df['anime_id'].isin(filtered_ratings)) & (rating_df['u
```

```
#create scale for rating, and create a dataset which can be loaded into the surpris
reader = Reader(rating_scale=(1,10))
model_data = Dataset.load_from_df(df_clean[['user_id', 'anime_id', 'rating']], read
```

In [119…

```
#OPTIONAL replace -1 ratings with the mean of all ratings instead, as -1 (viewed bu
#THIS GREATLY IMPROVES THE MODEL FOR SOME MODEL TYPES

# df_clean['rating'] = df_clean['rating'].replace(-1, df_clean['rating'].mean())
# model_data = Dataset.load_from_df(df_clean[['user_id', 'anime_id', 'rating']], re

#ALTERNATIVE remove all ratings of -1
#THIS PROVIDES THE BEST RESULTS, BUT MAY BE RECOMMENDING ANIMES THE USER HAS ALREAD

df_clean = df_clean[df_clean['rating'] != -1]
model_data = Dataset.load_from_df(df_clean[['user_id', 'anime_id', 'rating']], read
```

In [137…

```
#split data into train/test
train, test = train_test_split(model_data, test_size=0.2, random_state=8)

#initialize models
#USING SVD ONLY, AS DATASET IS TOO LARGE FOR OTHER TYPES
#models = [KNNWithZScore(), KNNBaseline(), KNNBasic(), KNNWithMeans(), CoClustering
#test different models, cross validate to find the likely optimal model
models = [SVD(random_state=8)]
benchmarks = []
for model in models:
    result = cross_validate(model, model_data, measures=['RMSE'], cv=5, verbose=Fal

    d = pd.DataFrame.from_dict(result).mean(axis=0)
    d = d.append(pd.Series([str(model).split(' ')[0].split('.')[-1]], index=['Model
    benchmarks.append(d)
    print(str(model))
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD object at 0x000001F358681FF
0>
```

Out[137…

Model	test_rmse	fit_time	test_time
SVD	1.122007	58.541967	12.777247

In [154…

```
model = SVD(random_state=8)
#fit final model
model.fit(train)

#get model preditions
pred = model.test(test)

#get top n recommendations for a given user
def get_top_n_recs(model, user_id, n):

    #find all shows not yet rated
    seen = set(df_clean[df_clean['user_id']==user_id]['anime_id'])
    all_animes = set(df_clean['anime_id'])
    animes_to_pred = list(all_animes - seen)
```

```
    #make predictions for all unrated shows
    preds = [model.predict(user_id, anime_id) for anime_id in animes_to_pred]

    #sort preds by estimated rating
    top_n = sorted(preds, key=lambda x: x.est, reverse=True)[:n]

    return [(pred.iid, pred.est) for pred in top_n]

user_id = 1
recs = get_top_n_recs(model, user_id, 5)

for show_id, estimated_rating in recs:
    print(f"Anime: {anime_df[anime_df['anime_id'] == show_id]['name'].values[0]} |
```

```
Anime: Death Note | Estimated Rating: 9.78
Anime: Shingeki no Kyojin | Estimated Rating: 9.74
Anime: Gintama&#039; | Estimated Rating: 9.74
Anime: Gintama° | Estimated Rating: 9.66
Anime: Fairy Tail | Estimated Rating: 9.65
```

In [145…

```
#final model rmse
accuracy.rmse(predictions=pred)
```

```
RMSE: 1.1206
```

Out[145…

```
1.1205518154421188
```