Collaborative Filtering Model

By Josh Houlding

The motivation of collaborative filtering comes from the idea that you'll probably like things that people with similar viewing habits also like.

For this activity, in 500-750 words, answer the following:

1. Formulate a prediction question that you want to answer by applying a collaborative filtering recommender algorithm. Examples: How to determine which users or movies are similar to one another; how to determine the rating that a user would give to a movie based on the ratings of similar users; how to measure the accuracy of the ratings.
2. Search and locate a dataset that is relevant to the question(s) you created in the previous step. You may search repositories such as Data.gov, UCI Machine Learning, Kaggle, or Scikit-Learn.
3. Import the necessary libraries in Python for building a recommender system.
4. Load the dataset into a data frame and preprocess it as needed.
5. Split the dataset into training and testing sets.
6. Fit a recommender system model on the training data.
7. Make some predictions and use MAE and RMSE to measure/estimate the accuracy of your predictions. Report and interpret the results.
8. Use the GridSearchCV class to fine-tune the algorithm parameters. Report the best parameter for any accuracy measure. Explain.

Task 1

Formulate a prediction question that you want to answer by applying a collaborative filtering recommender algorithm. Examples: How to determine which users or movies are similar to one another; how to determine the rating that a user would give to a movie based on the ratings of similar users; how to measure the accuracy of the ratings.

Prediction question: Which video games on Steam are similar to one another?

Task 2

Search and locate a dataset that is relevant to the question(s) you created in the previous step. You may search repositories such as Data.gov, UCI Machine Learning, Kaggle, or Scikit-Learn.

Dataset chosen: Steam Store Games (Clean Dataset) https://www.kaggle.com/datasets/nikdavis/steam-store-games

This dataset includes data gathered around May 2019 about over 27,000 video games on Steam, the largest PC digital game platform. It was collected directly from the Steam Store using the third-party SteamSpy API.

Task 3

Import the necessary libraries in Python for building a recommender system.

In [731…
```
# Install libraries using PIP
# !pip install scikit-surprise
```

In [732…
```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
```

Task 4

Load the dataset into a data frame and preprocess it as needed.

In [733…
```
# Load and view data
df = pd.read_csv("steam.csv")
df.head()
```

Out[733]:

	appid	name	release_date	english	developer	publisher	platforms	required_age	categories	genres	steamspy_tags
0	10	Counter-Strike	2000-11-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
1	20	Team Fortress Classic	1999-04-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
2	30	Day of Defeat	2003-05-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Valve Anti-Cheat enabled	Action	FPS;World War II;Multiplayer
3	40	Deathmatch Classic	2001-06-01	1	Valve	Valve	windows;mac;linux	0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer
4	50	Half-Life: Opposing Force	1999-11-01	1	Gearbox Software	Valve	windows;mac;linux	0	Single-player;Multi-player;Valve Anti-Cheat en...	Action	FPS;Action;Sci-fi

In [734...

```
# Show datatypes and shape
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27075 entries, 0 to 27074
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   appid             27075 non-null  int64
 1   name              27075 non-null  object
 2   release_date      27075 non-null  object
 3   english           27075 non-null  int64
 4   developer         27074 non-null  object
 5   publisher         27061 non-null  object
 6   platforms         27075 non-null  object
 7   required_age      27075 non-null  int64
 8   categories        27075 non-null  object
 9   genres            27075 non-null  object
 10  steamspy_tags     27075 non-null  object
 11  achievements      27075 non-null  int64
 12  positive_ratings  27075 non-null  int64
 13  negative_ratings  27075 non-null  int64
 14  average_playtime  27075 non-null  int64
 15  median_playtime   27075 non-null  int64
 16  owners            27075 non-null  object
 17  price             27075 non-null  float64
dtypes: float64(1), int64(8), object(9)
memory usage: 3.7+ MB
```

Inserting column for proportion of positive reviews

This new column will give us insight into how favorably the Steam community views a particular game overall.

In [735…

```
# Insert column for proportion of reviews that are positive
proportion_positive = df["positive_ratings"] / (df["positive_ratings"] + df["negative_ratings"])
df.insert(14, "proportion_positive", proportion_positive)
```

Splitting `release_date` into multiple features

`release_date` can only be useful in modeling if it is split into 3 features: year, month and day.

In [736…

```
# Show current column datatype
print("Datatype of \"release_date\":", df["release_date"].dtype)

# Convert "release_date" to datetime
```

```
df["release_date"] = pd.to_datetime(df["release_date"])
print("Successfully converted datatype of column.")
print("Datatype of \"release_date\":", df["release_date"].dtype)

# Split into year, month and day
year = df["release_date"].dt.year
month = df["release_date"].dt.month
day = df["release_date"].dt.day

# Attach new columns to the dataframe
df = pd.concat([df, year.rename("year"), month.rename("month"), day.rename("day")], axis=1)
print("Concatenated df with new year, month and day columns.")

# Drop original "release_date" column
df.drop(columns={"release_date"}, inplace=True)
print("Dropped original \"release_date\" column.")
```

```
Datatype of "release_date": object
Successfully converted datatype of column.
Datatype of "release_date": datetime64[ns]
Concatenated df with new year, month and day columns.
Dropped original "release_date" column.
```

Normalizing numeric variables not already on [0, 1] scale

In [737…

```
from sklearn.preprocessing import MinMaxScaler

# Define variables to normalize
variables_to_normalize = ["required_age", "achievements", "positive_ratings", "negative_ratings", "proportion_positive"
                          "average_playtime", "median_playtime", "price", "year", "month", "day"]

# Normalize variables
scaler = MinMaxScaler()
df[variables_to_normalize] = scaler.fit_transform(df[variables_to_normalize])
```

View final dataframe for modeling

In [738…

```
# View max columns
pd.set_option("display.max_columns", None)

# Show final dataframe shape
print("Dataframe shape:", df.shape)
```

```
# Show final dataframe
df.head()
```

```
Dataframe shape: (27075, 21)
```

Out[738]:

	appid	name	english	developer	publisher	platforms	required_age	categories	genres	steamspy_tags	achievement
0	10	Counter-Strike	1	Valve	Valve	windows;mac;linux	0.0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer	0.
1	20	Team Fortress Classic	1	Valve	Valve	windows;mac;linux	0.0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer	0.
2	30	Day of Defeat	1	Valve	Valve	windows;mac;linux	0.0	Multi-player;Valve Anti-Cheat enabled	Action	FPS;World War II;Multiplayer	0.
3	40	Deathmatch Classic	1	Valve	Valve	windows;mac;linux	0.0	Multi-player;Online Multi-Player;Local Multi-P...	Action	Action;FPS;Multiplayer	0.
4	50	Half-Life: Opposing Force	1	Gearbox Software	Valve	windows;mac;linux	0.0	Single-player;Multi-player;Valve Anti-Cheat en...	Action	FPS;Action;Sci-fi	0.

Task 5

Split the dataset into training and testing sets.

I had some difficulty figuring out what the label of this model should be, but I eventually settled on using `proportion_positive`. This column is appropriate because user ratings for games will be helpful in determining which are similar to each other.

In [739…

```
# Save indices and IDs of original dataset
original_indices = df.index.tolist()
game_ids = df["appid"].tolist()

# Select features and label
x = df.drop(columns={"appid", "name", "developer", "publisher"})
y = df["average_playtime"]

# Perform train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Task 6

Fit a recommender system model on the training data.

In [740…

```
# Import necessary libraries
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import SVD  # Example collaborative filtering algorithm
from surprise import accuracy
```

In [741…

```
# Create reader
reader = Reader()

# Select features
"""
Features of interest: "developer", "publisher", "categories", "genres", "steamspy_tags", "positive_ratings",
"negative_ratings", "proportion_positive", "average_playtime", "median_playtime", "price", "year"
"""

data = Dataset.load_from_df(df[["appid", "genres", "proportion_positive"]], reader)

# Perform train-test split
train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

# Create and fit Singular Value Decomposition (SVD) model
model = SVD()
model.fit(train_set)

# Make predictions with model
predictions = model.test(test_set)
```

Task 7

Make some predictions and use MAE and RMSE to measure/estimate the accuracy of your predictions. Report and interpret the results.

In [742…

```
# Get first 10 predictions
prediction_subset = predictions[:10]
prediction_subset
```

Out[742]:

```
[Prediction(uid=618640, iid='Action;Indie;Early Access', r_ui=0.7619047619047619, est=1, details={'was_impossible': Fal
se}),
 Prediction(uid=285740, iid='Casual;Indie;Simulation', r_ui=0.939297124600639, est=1, details={'was_impossible': Fals
e}),
 Prediction(uid=477160, iid='Adventure;Indie', r_ui=0.9137492350800348, est=1, details={'was_impossible': False}),
 Prediction(uid=868780, iid='Adventure;Indie;RPG', r_ui=0.9333333333333333, est=1, details={'was_impossible': False}),
 Prediction(uid=235820, iid='Action;Indie', r_ui=0.8363636363636363, est=1, details={'was_impossible': False}),
 Prediction(uid=282590, iid='Indie;Strategy', r_ui=0.8003220611916264, est=1, details={'was_impossible': False}),
 Prediction(uid=244690, iid='Adventure;Indie', r_ui=0.6781115879828327, est=1, details={'was_impossible': False}),
 Prediction(uid=816170, iid='Adventure;Indie', r_ui=1.0, est=1, details={'was_impossible': False}),
 Prediction(uid=517480, iid='Action;Indie', r_ui=0.25, est=1, details={'was_impossible': False}),
 Prediction(uid=988980, iid='Action;Casual;Indie;Strategy', r_ui=1.0, est=1, details={'was_impossible': False})]
```

In [743…

```
# Pull out ids of predictions
prediction_ids = [pred.uid for pred in prediction_subset]

# Find games included in first 10 predictions
games = df[df["appid"].isin(prediction_ids)]
display_columns = games[["appid", "name", "genres"]]
display_columns
```

Out[743]:

	appid	name	genres
1717	235820	Element4l	Action;Indie
1848	244690	Face Noir	Adventure;Indie
2696	282590	Star Ruler 2	Indie;Strategy
2786	285740	Kitty Powers' Matchmaker	Casual;Indie;Simulation
9259	477160	Human: Fall Flat	Adventure;Indie
10623	517480	VHSoverdose	Action;Indie
14154	618640	The Crowded Party Game Collection	Action;Indie;Early Access
20896	816170	Spectrubes Infinity	Adventure;Indie
22629	868780	I, Cyborg	Adventure;Indie;RPG
25842	988980	GLAD VALAKAS TOWER DEFENCE	Action;Casual;Indie;Strategy

Looking at these predictions, we can see that all of them have the "indie" genre, and many have "action", "adventure", or "casual". Thus, it would appear these games are fairly similar in genre.

Let's check the MAE and RMSE for this feature combination to see how robust it is:

In [744…

```
# View MAE and RMSE
mae = accuracy.mae(predictions)
rmse = accuracy.rmse(predictions)
```

```
MAE:  0.2866
RMSE: 0.3703
```

The target variable `proportion_positive` is on a scale of $[0, 1]$, so these metrics are rather high. It would be of interest to test at least one other subset of features to see if a lower MAE and RMSE can be achieved.

Trying other feature subsets

In [749…

```
from itertools import product

# Gather feature subsets for testing
other_interesting_features = ["developer", "publisher", "categories", "steamspy_tags", "positive_ratings",
"negative_ratings", "average_playtime", "median_playtime", "price", "year"]
feature_subsets = [["appid", feature, "proportion_positive"] for feature in other_interesting_features]

# Test feature subsets to find optimal RMSE
for subset in feature_subsets:
    data = Dataset.load_from_df(df[subset], reader)
    train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)
    model = SVD()
    model.fit(train_set)
    predictions = model.test(test_set)
    print("RMSE for feature subset", subset, ":", round(accuracy.rmse(predictions, verbose=False), 5))
```

```
RMSE for feature subset ['appid', 'developer', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'publisher', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'categories', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'steamspy_tags', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'positive_ratings', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'negative_ratings', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'average_playtime', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'median_playtime', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'price', 'proportion_positive'] : 0.37031
RMSE for feature subset ['appid', 'year', 'proportion_positive'] : 0.37031
```

It would appear that we get the same RMSE for every other feature we try with the app ID and proportion of positive reviews. I also tried `median_playtime` and `price` as targets for the model, but both of these had RMSE values exceeding 0.98, so it appears I was correct in choosing `proportion_positive` as the target initially.

Task 8

Use the GridSearchCV class to fine-tune the algorithm parameters. Report the best parameter for any accuracy measure. Explain.

In [746…

```
from surprise.model_selection import GridSearchCV
from surprise.dataset import DatasetAutoFolds
from surprise import Dataset
```

```python
from surprise.model_selection import KFold

# Define parameter grid
param_grid = {"n_factors": [50, 100, 200], "reg_all": [0.01, 0.02, 0.05]}

# Initialize GridSearchCV
grid_search = GridSearchCV(SVD, param_grid, measures=["rmse", "mae"], cv=3)

# Fit grid search object to data
grid_search.fit(data)

# Print best RMSE score and parameters
print("Best RMSE score:", grid_search.best_score["rmse"])
print("Best parameters for RMSE:", grid_search.best_params["rmse"])
print("Best MAE score:", grid_search.best_score["mae"])
print("Best parameters for MAE:", grid_search.best_params["mae"])
```

```
Best RMSE score: 0.9993100962168989
Best parameters for RMSE: {'n_factors': 50, 'reg_all': 0.01}
Best MAE score: 0.9992338044351604
Best parameters for MAE: {'n_factors': 50, 'reg_all': 0.01}
```

The best parameters for the grid search are `n_factors` $= 50$ and `reg_all` $= 0.01$. This means that 50 latent factors and a regularization strength of 0.01 maximize model accuracy while striking a balance between complexity and generalizability.

References

ChatGPT. (n.d.). https://chat.openai.com/

Collaborative filtering. (n.d.). Google for Developers. https://developers.google.com/machine-learning/recommendation/collaborative/basics