Building User-Based Recommendation Model for Amazon .

November 9, 2022

DESCRIPTION

The dataset provided contains movie reviews given by Amazon customers. Reviews were given between May 1996 and July 2014.

Data Dictionary

UserID – 4848 customers who provided a rating for each movie

Movie 1 to Movie 206 – 206 movies for which ratings are provided by 4848 distinct users

Data Considerations

All the users have not watched all the movies and therefore, all movies are not rated. These missing values are represented by NA.

Ratings are on a scale of -1 to 10 where -1 is the least rating and 10 is the best.

Analysis Task

Exploratory Data Analysis:

Which movies have maximum views/ratings?

What is the average rating for each movie? Define the top 5 movies with the maximum ratings.

Define the top 5 movies with the least audience.

Recommendation Model: Some of the movies hadn't been watched and therefore, are not rated by the users. Netflix would like to take this as an opportunity and build a machine learning recommendation algorithm which provides the ratings for each of the users.

Divide the data into training and test data

Build a recommendation model on training data

Make predictions on the test data

[ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

[ ]:

```
ratings=pd.read_csv('Amazon - Movies and Tv Ratings.csv', index_col=0)
ratings.head()
```

[ ]:

```
ratings.tail()
```

[ ]:
```
ratings.describe()
```

[ ]:
```
ratings.dtypes
```

[ ]:
```
ratings.isna().sum()
```

[ ]:
```
ratings.fillna(0)
```

1 Exploratory Data Analysis:

[ ]:
```
rating_stack=ratings.stack().reset_index()
rating_stack
```

[ ]:
```
rating_stack.columns = ['User_ID','Movie','Rating']
```

[ ]:
```
n_ratings = len(rating_stack)
```

[ ]:
```
n_movies = len(rating_stack['Movie'].unique())
n_users = len(rating_stack['User_ID'].unique())
```

[ ]:
```
print(f"Number of ratings: {n_ratings}")
print(f"Number of unique movieId's: {n_movies}")
print(f"Number of unique users: {n_users}")
print(f"Average ratings per user: {round(n_ratings/n_users, 2)}")
print(f"Average ratings per movie: {round(n_ratings/n_movies, 2)}")
```

[ ]:
```
rating_stack.head()
```

[ ]:
```
ratings_mean=rating_stack.groupby('Movie')['Rating'].describe()['mean']
ratings_mean
```

[ ]:
```
ratings_count = rating_stack.groupby('Movie')['Rating' ].describe()['count']
ratings_count
```

[ ]:
```
ratings_concat = pd.concat([ratings_count, ratings_mean], axis = 1)
ratings_concat
```

[ ]:
```
ratings_concat['mean'].plot(bins=25, kind='hist', color = 'indianred')
plt.legend()
plt.show()
```

[ ]:
```
ratings_concat['count'].plot(bins=25, kind='hist', color = 'indianred')
plt.legend()
plt.show()
```

1.0.1 Which movies have maximum views/ratings?

[ ]:
```
top=ratings_concat.sort_values('mean', ascending=False).head(5)
top
```

1.0.2 What is the average rating for each movie? Define the top 5 movies with the maximum ratings.

[ ]:
```
top.plot()
plt.legend()
plt.show()
```

1.0.3 Define the top 5 movies with the least audience.

[ ]:
```
least=ratings_concat.sort_values('mean', ascending = True).head(5)
```

[ ]:
```
least
```

[ ]:
```
least.plot()
plt.legend()
plt.show()
```

1.0.4 Divide the data into training and test data

[ ]:
```
from collections import Counter
```

[ ]:
```
rating_stack.columns
```

[ ]:
```
df1=Counter(rating_stack['Movie'])
movie=pd.DataFrame.from_dict(df1,orient='index')
movie
```

[ ]:
```
x=movie
x=x.head()
x
```

[ ]:
```
df2=Counter(rating_stack['Rating'])
ratings=pd.DataFrame.from_dict(df2,orient='index')
ratings
```

[ ]:
```
y=ratings
y
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

[ ]:
```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.
  ↪20,random_state=0)
```

```
[ ]: logreg=LogisticRegression()
```

```
[ ]: x_train.shape
```

```
[ ]: x_test.shape
```

```
[ ]: y_test.shape
```

```
[ ]: y_train.shape
```

2 Make predictions on the test data

```
[ ]: logreg.fit(x_train, y_train)
```

```
[ ]: y_pred=logreg.predict(x_test)
     y_pred
```

```
[ ]: y_pred.shape
```

3 Build a recommendation model on training data

```
[ ]: from sklearn import metrics
```

```
[ ]: cmatrix=metrics.confusion_matrix(y_test, y_pred)
```

```
[ ]: cmatrix=metrics.ConfusionMatrixDisplay(confusion_matrix=cmatrix,
                                            display_labels=[False, True])
```

```
[ ]: cmatrix.plot()
     plt.show()
```

```
[ ]: from sklearn.metrics import accuracy_score
```

```
[ ]: accuracy_score(y_test, y_pred)
```

```
[ ]: from sklearn.linear_model import LinearRegression
```

```
[ ]: linreg=LinearRegression()
```

```
[ ]: linreg.fit(x_train, y_train)
```

```
[ ]: y_pred1=linreg.predict(x_test)
     y_pred1=pd.DataFrame(y_pred1, columns=['Predicted'])
     y_pred1
```

```
[ ]: y_pred1.shape
[ ]: from sklearn.metrics import mean_absolute_error,mean_squared_error, r2_score
```

```
[ ]: mae=mean_absolute_error(y_test, y_pred1)
     mae
```

```
[ ]: mse=mean_squared_error(y_test, y_pred1)
     mse
```

```
[ ]: r2=r2_score(y_test, y_pred1)
     r2
```

```
[ ]: from math import sqrt as sqrt
```

```
[ ]: smse=sqrt(mean_squared_error(y_test, y_pred1))
     smse
```