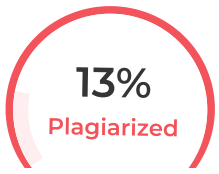
Plagiarism Detector .net

Dec 05, 2023

Plagiarism Scan Report



Characters:4427	Words:454
Sentences:8	Speak Time: 4 Min

Excluded URL	None

Content Checked for Plagiarism

import pandas as pd import numpy as np import matplotlib.pyplot as plt # Reading ratings file # Ignore the timestamp column ratings = pd.read_csv('ratings.csv', sep='\t', encoding='latin-1', usecols=['user_id', 'movie_id', 'rating']) # Reading users file users = pd.read_csv('users.csv', sep='\t', encoding='latin-1', usecols=['user_id', 'gender', 'zipcode', 'age_desc', 'occ_desc']) # Reading movies file movies = pd.read_csv('movies.csv', sep='\t', encoding='latin-1', usecols=['movie_id', 'title','genres']) # Check the top 5 rows print(users.head()) # Check the file info print(users.info()) # Check the top 5 rows print(movies.head()) # Check the file info print(movies.info()) # Data Exploration %matplotlib inline import wordcloud from wordcloud import WordCloud, STOPWORDS # Create a wordcloud of the movie titles movies['title'] = movies['title'].fillna("").astype('str') title_corpus = ' '.join(movies['title']) title_wordcloud = WordCloud(stopwords=STOPWORDS, background_color='black', height=2000, width=4000).generate(title_corpus) # Plot the wordcloud plt.figure(figsize=(16,8)) plt.imshow(title_wordcloud) plt.axis('off') plt.show() # Get summary statistics of rating ratings['rating'].describe() # Import seaborn library import seaborn as sns sns.set_style('whitegrid') sns.set(font_scale=1.5) %matplotlib inline # Display distribution of rating sns.distplot(ratings['rating'].fillna(ratings['rating'].median())) # Join all 3 files into one dataframe dataset = pd.merge(pd.merge(movies, ratings),users) # Display 20 movies with highest ratings dataset[['title','genres','rating']].sort_values('rating', ascending=False).head(20) Loading... 12/5/23, 1:45 PM DL PROJECT - Colaboratory https://colab.research.google.com/drive/1jjW8fDKwtPzY7UnGoYsAguVArikaYcDu#scrollTo=TojAc5MX5bc7&printMode=t 2/3 # Make a census of the genre keywords genre_labels = set() for s in movies['genres'].str.split('|').values: genre_labels = genre_labels.union(set(s)) # Function that counts the number of times each of the genre keywords appear def count_word(dataset, ref_col, census): keyword_count = dict() for s in census: keyword_count[s] = 0 for census_keywords in dataset[ref_col].str.split('|'): if type(census_keywords) == float and pd.isnull(census_keywords): continue for s in [s for s in census_keywords if s in census]: if pd.notnull(s): keyword_count[s] += 1 #_____ # convert the dictionary in a list to sort the keywords by frequency keyword_occurences = [] for k,v in keyword_count.items(): keyword_occurences.append([k,v]) keyword_occurences.sort(key = lambda x:x[1], reverse = True) return keyword_occurences, keyword_count # Calling

this function gives access to a list of genre keywords which are sorted by decreasing frequency keyword_occurences, dum = count_word(movies, 'genres', genre_labels) keyword_occurences[:5] #ContentBased # Break up the big genre string into a string array movies['genres'] = movies['genres'].str.split('|') # Convert genres to string value movies['genres'] = movies['genres'].fillna("").astype('str') from sklearn.feature_extraction.text import TfidfVectorizer tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0.0, stop_words='english') tfidf_matrix = tf.fit_transform(movies['genres']) tfidf_matrix.shape from sklearn.metrics.pairwise import linear_kernel cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix) cosine_sim[:4, :4] # Create two user-item matrices, one for training and another for testing train_data_matrix = train_data[['user_id', 'movie_id', 'rating']].values test_data_matrix = test_data[['user_id', 'movie_id', 'rating']].values # Check their shape print(train_data_matrix.shape) print(test_data_matrix.shape) # Build a 1-dimensional array with movie titles titles = movies['title'] indices = pd.Series(movies.index, index=movies['title']) # Function that get movie recommendations based on the cosine similarity score of movie genres def genre_recommendations(title): idx = indices[title] sim_scores = list(enumerate(cosine_sim[idx])) sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True) sim_scores = sim_scores[1:21] movie_indices = [i[0] for i in sim_scores] return titles.iloc[movie_indices] genre_recommendations('Good Will Hunting (1997)').head(20) genre_recommendations('Toy Story (1995)').head(20) genre_recommendations('Saving Private Ryan (1998)').head(20)

Sources

13% Plagiarized

Content_Based_and_Collaborati...

https://github.com/khanhnamle1994/movielens/blob/master/Content_Based_and_Collaborative_Filtering_Models.ipynb