Book Recommendation System

CS 464 - Group 15
Utku Kurtulmuş 21903025
Ferhat Korkmaz 21901940
Ömer Oktay Gültekin 21901413
Dilay Yigit 21602059
Muhammet Oğuzhan Gültekin 21801616




Overview

- Introduction
- Data Preprocessing & Dataset
- Model ~ Collaborative Filtering
- Content Based Filtering
- Evaluation Metric
- Book Recommendation Logic
- Source Code
- Questions, Comments & Recommendations

Introduction

Dataset Description

- Amazon Books Reviews

Chosen Method

- Collaborative Filtering
- Content Based Methods

Predictive Modeling Phase

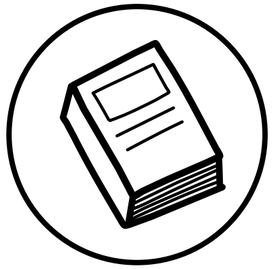
- Embedded Neural Networks

Data Preprocessing

Eliminated fields: price, profileName, review/helpful, review/time, review/summary, review/text, image, etc.

User Rating Data

+



Book Features

As a initial step we decided to use Collobarative Filtering Approach, therefore we have eliminated the features of the book, and looked for only the interactions between user and book itself.

Books_Rating

	Id	Title	Price	User_id	profileName	review/helpfulness	review/score	review/time	review/summary	review/text
2971446	B000G167FA	59970	NaN	9731	Ellen C. Falkenberry "ellenf"	5/5	5.0	-1	Have you ever watched the fairies when the rai...	Although the new cover looks more like a Book ...
2006209	B00005O4HA	52409	NaN	7094	Stan Vernooy	4/5	5.0	840240000	The best mystery novel I have ever read	I have been a mystery reader for decades, and ...
2111966	B000PRURRE	30319	NaN	6939	Michael J. Tresca "Talien"	2/2	5.0	850694400	A golden piece of sci-fi pulp that shines even...	This is one of those books that seems like it'...
1359864	0671536109	62006	NaN	6939	Michael J. Tresca "Talien"	18/19	5.0	850694400	The final world on the chronology of the Star ...	This book is a must have for any serious Star ...
100284	B0001BJEG4	30320	NaN	6939	Michael J. Tresca "Talien"	2/2	5.0	850694400	A golden piece of sci-fi pulp that shines even...	This is one of those books that seems like it'...

- **Books_Rating: Information about 3M book reviews for 212404 unique book.**

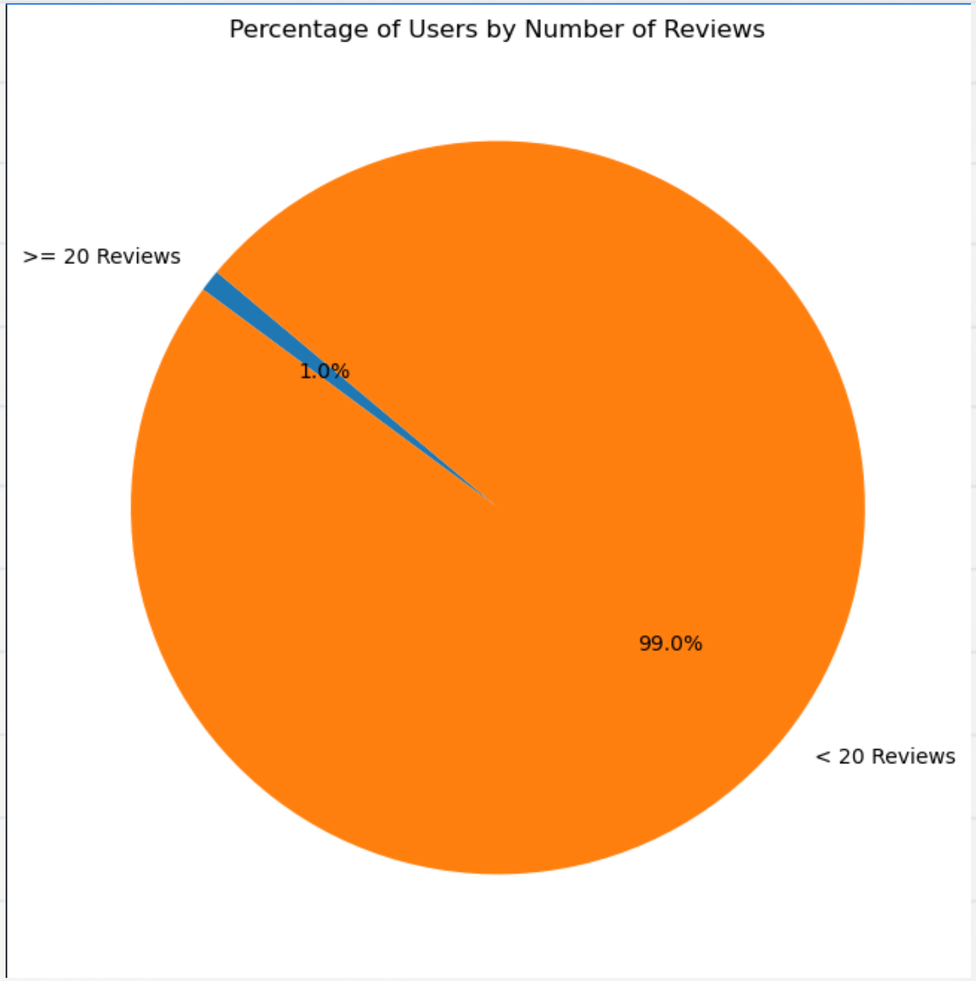
Books_Data

	Title	description	authors	image	previewLink	publisher	publishedDate	infoLink	categories	ratingsCount
0	Its Only Art If Its Well Hung!	NaN	['Julie Strain']	http://books.google.com/books/content?id=DykPA...	http://books.google.nl/books?id=DykPAAAACAAJ&d...	NaN	1996	http://books.google.nl/books?id=DykPAAAACAAJ&d...	['Comics & Graphic Novels']	NaN
1	Dr. Seuss: American Icon	Philip Nel takes a fascinating look into the k...	['Philip Nel']	http://books.google.com/books/content?id=IjvHQ...	http://books.google.nl/books?id=IjvHQsCn_pgC&p...	A&C Black	2005-01-01	http://books.google.nl/books?id=IjvHQsCn_pgC&d...	['Biography & Autobiography']	NaN
2	Wonderful Worship in Smaller Churches	This resource includes twelve principles in un...	['David R. Ray']	http://books.google.com/books/content?id=2tsDA...	http://books.google.nl/books?id=2tsDAAAACAAJ&d...	NaN	2000	http://books.google.nl/books?id=2tsDAAAACAAJ&d...	['Religion']	NaN
3	Whispers of the Wicked Saints	Julia Thomas finds her life spinning out of co...	['Veronica Haddon']	http://books.google.com/books/content?id=aRSIg...	http://books.google.nl/books?id=aRSIgJlq6JwC&d...	iUniverse	2005-02	http://books.google.nl/books?id=aRSIgJlq6JwC&d...	['Fiction']	NaN
4	Nation Dance: Religion, Identity and Cultural ...	NaN	['Edward Long']	NaN	http://books.google.nl/books?id=399SPgAACAAJ&d...	NaN	2003-03-01	http://books.google.nl/books?id=399SPgAACAAJ&d...	NaN	NaN

- **Books_Data: The details of 212404 books such as genres, authors, cover, description and etc.**

Dataset

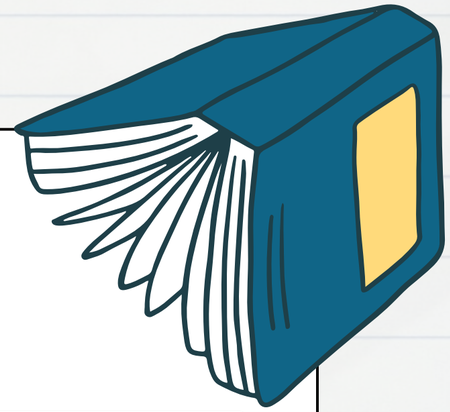
- We have eliminated the user's data who have less than 20 review to get better results.
- As a result there are 7209 users that have reviewed at least 20 books. And 548134 unique review data.
- We have label encoded the user_id and title columns, using the label encoder of scikitlearn.



Final Version

	Title	User_id	review/score	review/time
1	20778	5128	5.0	1095724800
3	20778	4136	4.0	1090713600
5	20778	3562	4.0	1127174400
6	20778	321	5.0	1100131200
11	93849	5561	5.0	1291766400

Dataset



Training Dataset

Comprises 80% of the data, used for training the model on user ratings for books.

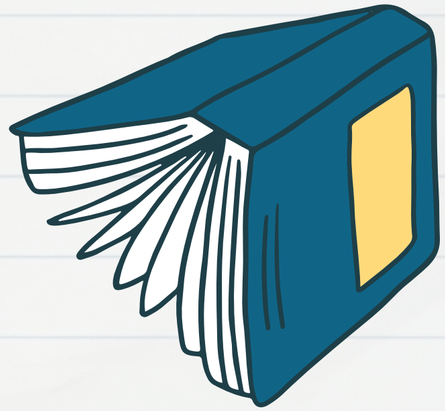
Validation Dataset

Consists of 10% of the data, employed to fine-tune and validate the model during training.

Test Dataset

The remaining 10%, used to assess the model's performance on new, unseen data

We have sorted our array according to the review date, and oldest 80 percent are reserved for the train set and newest 10 percent for the test test. We are using a 80-10-10 split.



Models

Embedded Neural Network

- We are using Neural Networks with embedding as our initial model. We selected TensorFlows Neural Network library to implement our model.
- We are using a collaborative filtering approach therefore, we have 2 embedding layers to embed encoded user_id and title columns.

```
# Define input layers for user and item IDs
user_input = Input(shape=(1,), name='user_input')
item_input = Input(shape=(1,), name='item_input')

# Embedding layers
user_embedding = Embedding(input_dim=num_users, output_dim=embedding_size, input_length=1)(user_input)
item_embedding = Embedding(input_dim=num_items, output_dim=embedding_size, input_length=1)(item_input)
```

- After that we have one dense layer with 128 (default) neurons with the Relu activation function.
- And one final output layer.

```
# Concatenate the flattened embeddings
concatenated_features = Concatenate()([user_flat, item_flat])

# Dense layer for further processing
dense_layer = Dense(dense_size, activation='relu')(concatenated_features)
```

Embedded Neural Network

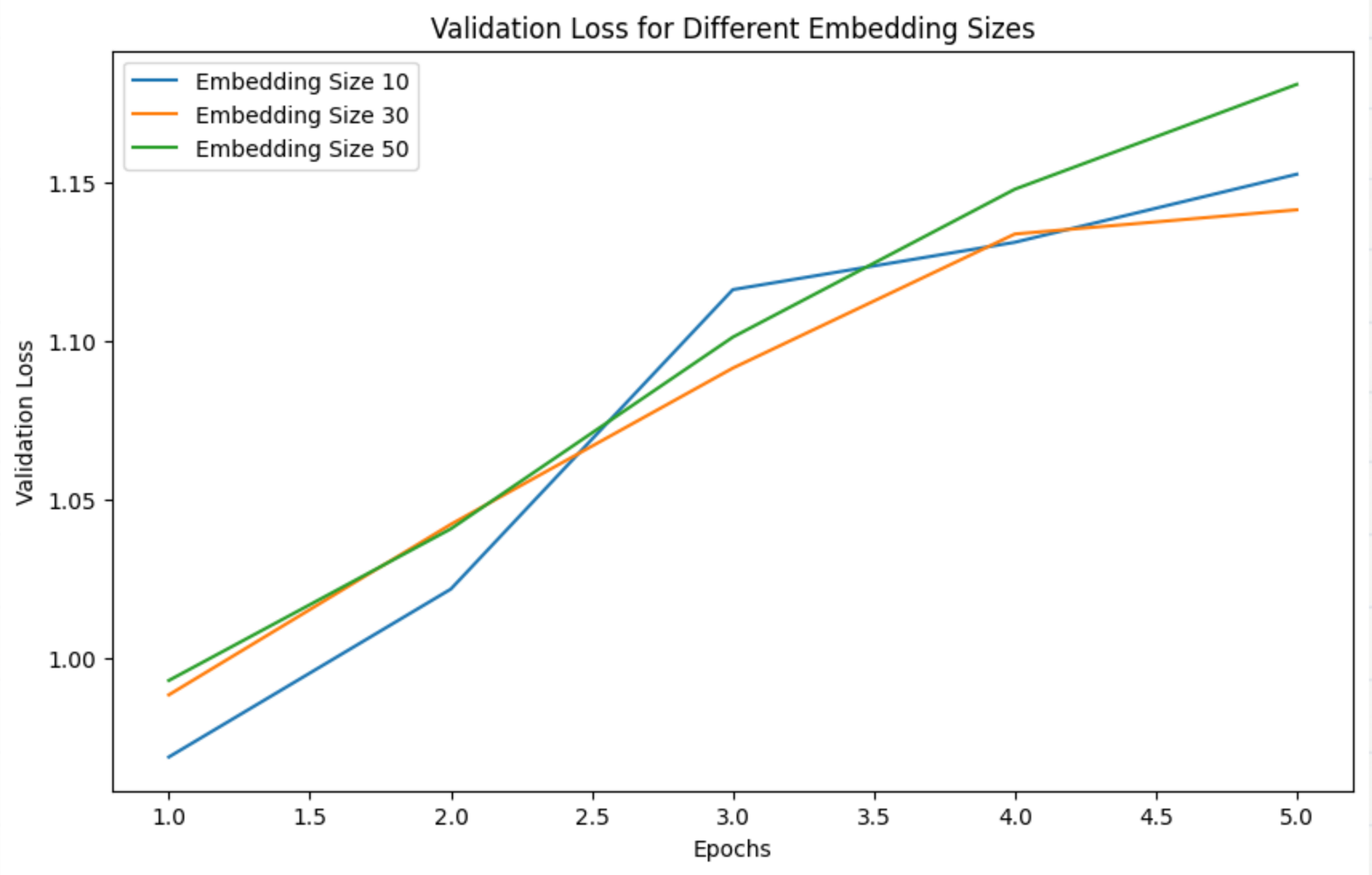
- We are using Adam optimizer and mean squared error to complete our model.

```
# Output layer
output_layer = Dense(1)(dense_layer)

# Create the model
model = Model(inputs=[user_input, item_input], outputs=output_layer)
model.compile(optimizer='adam', loss='mean_squared_error')

return model
```

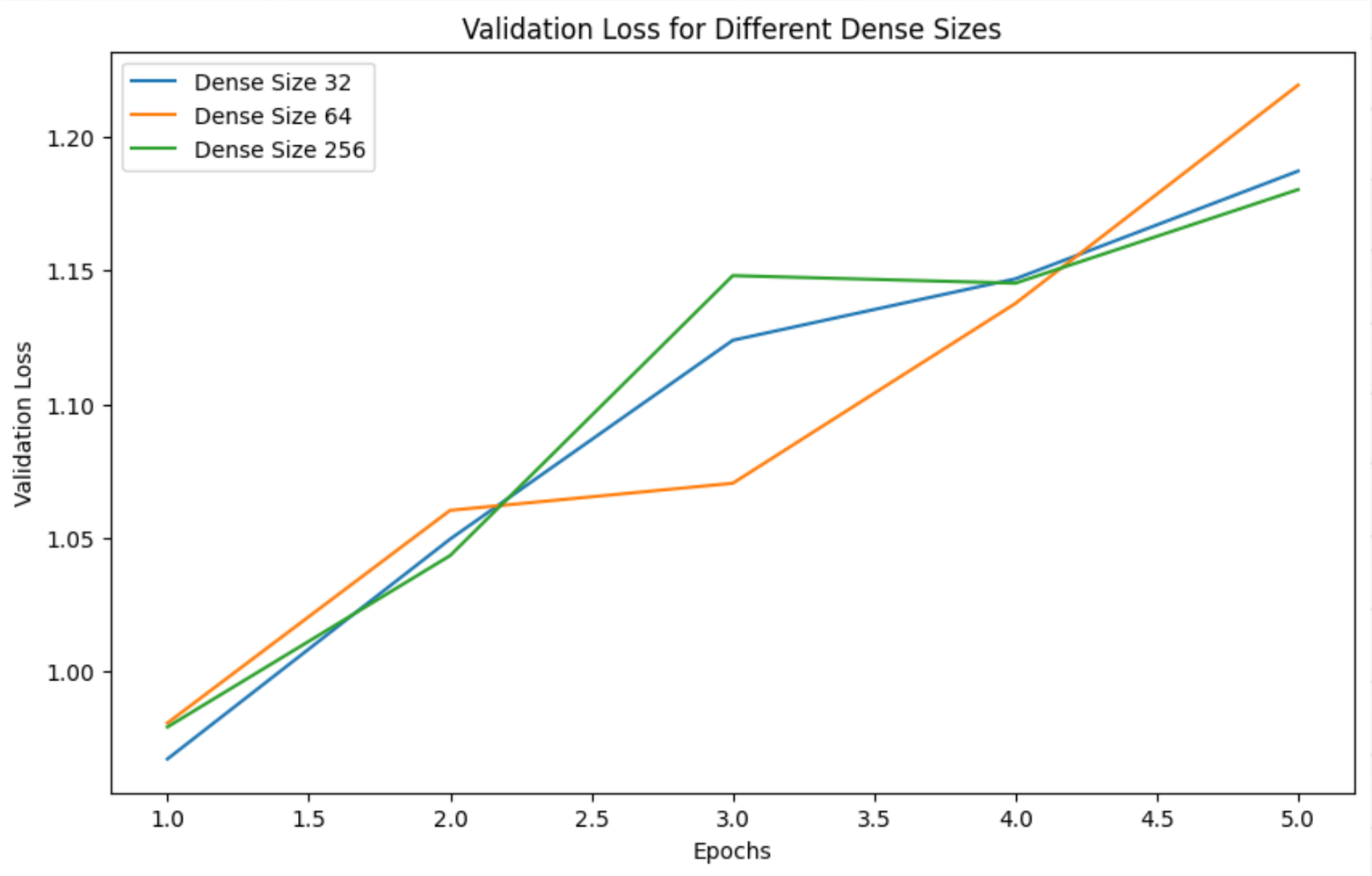
- We are using our validation set along with the train set to increase the accuracy of our model

```
num_epochs = 5
default_batch_size = 64
model = create_collaborative_filtering_model()
model.fit(
    [
        train_set['User_id'],
        train_set['Title'],
    ],
    train_set['review/score'],
    epochs=num_epochs,
    batch_size = default_batch_size,
    validation_data=(
        [
            validation_set['User_id'],
            validation_set['Title'],
        ],
        validation_set['review/score']
    )
)
```

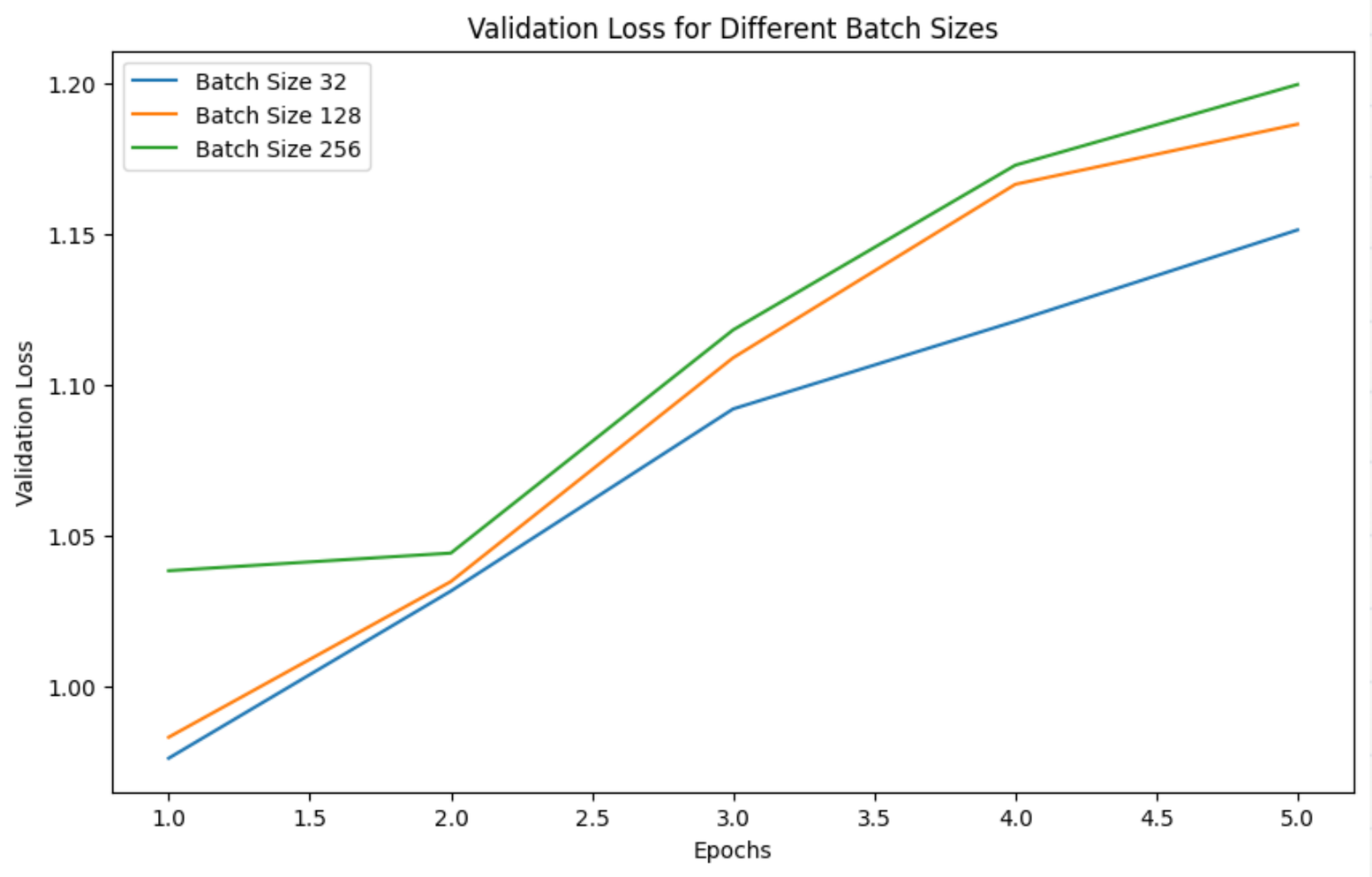
Parameter Tuning



Parameter Tuning

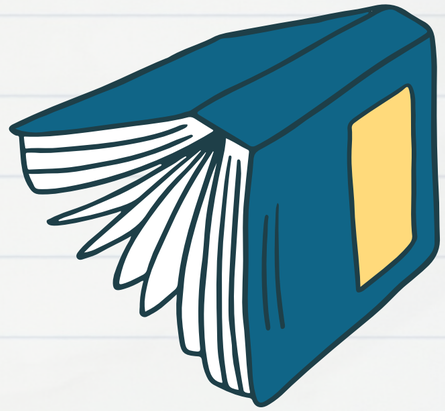


Parameter Tuning



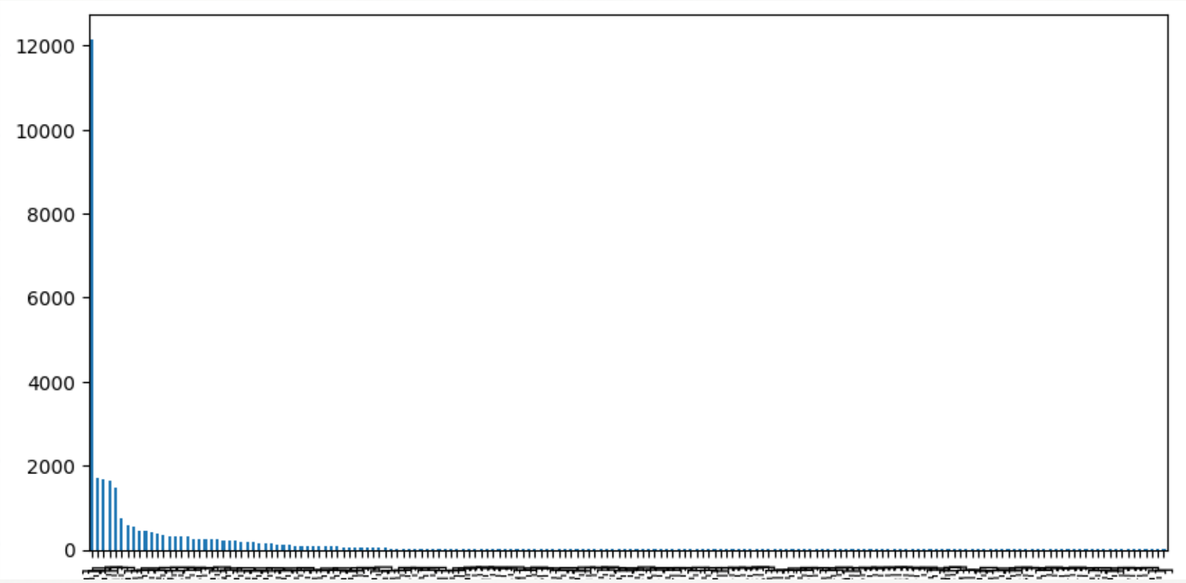
Best Parameters

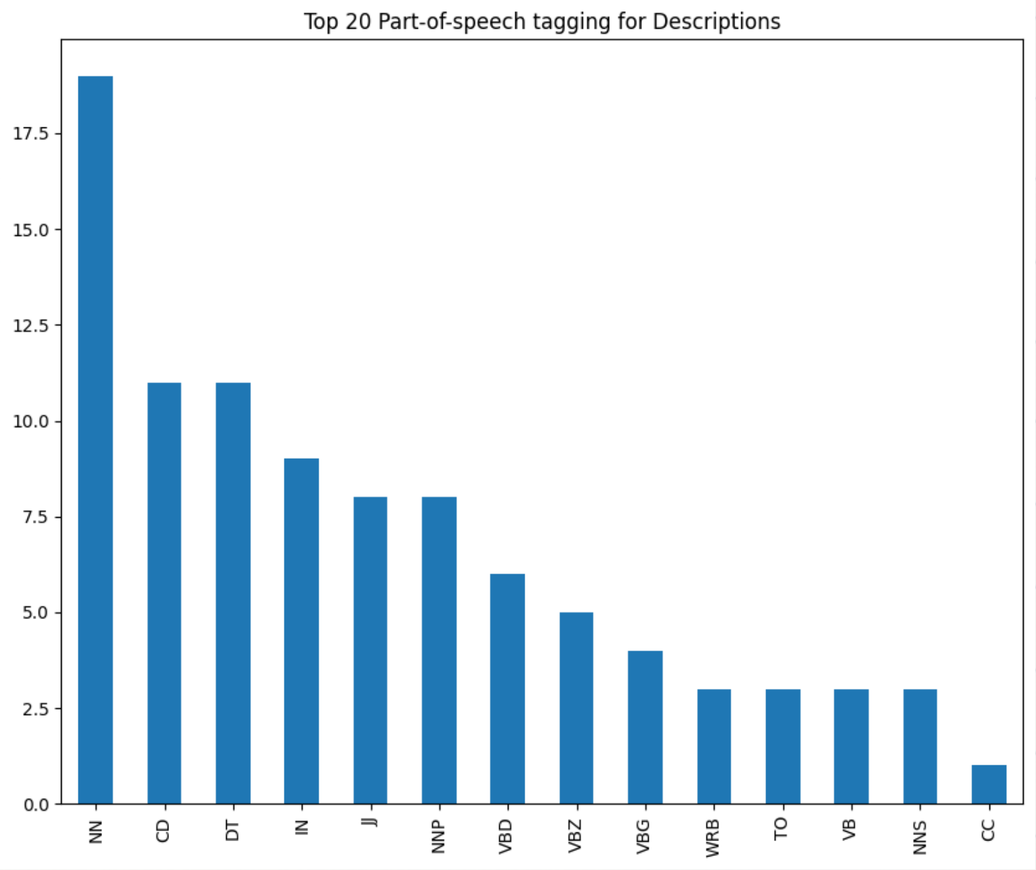
```
#best model
best_batch_size = 32
best_embedding_size = 30
best_dense_size = 256
model = create_collaborative_filtering_model(best_embedding_size, best_dense_size)
model.fit(
      [
          train_set['User_id'],
          train_set['Title'],
      ],
      train_set['review/score'],
      epochs=num_epochs,
      batch_size = best_batch_size,
      validation_data=(
          [
              validation_set['User_id'],
              validation_set['Title'],
          ],
          validation_set['review/score']
      )
  )
```



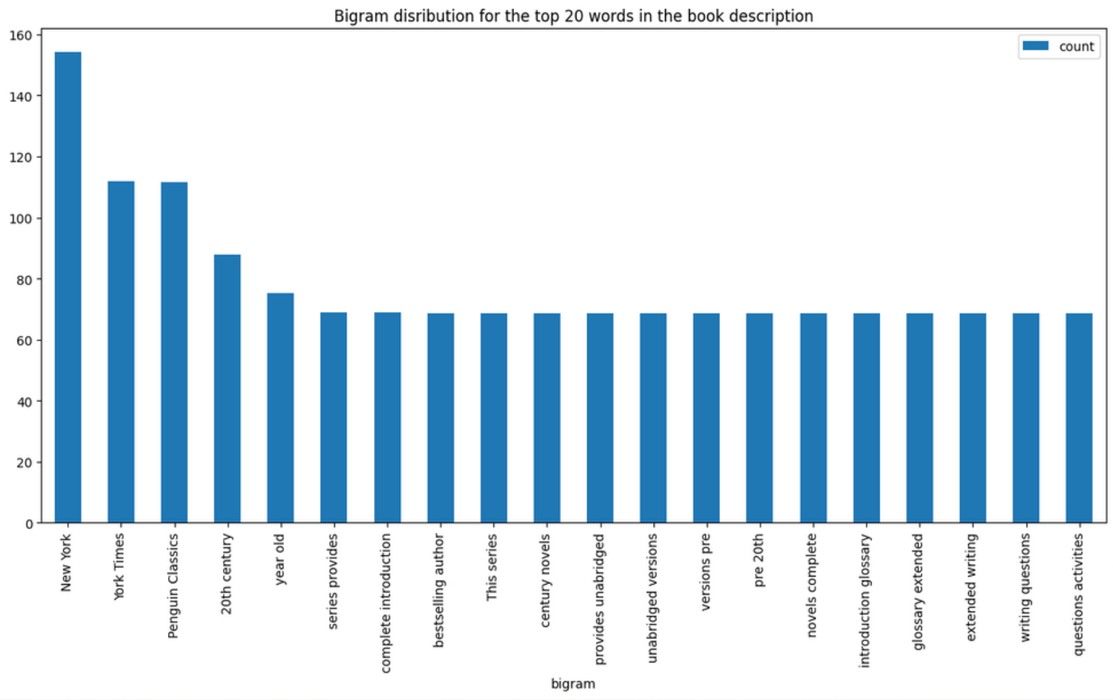
Content-Based Filtering & NLP

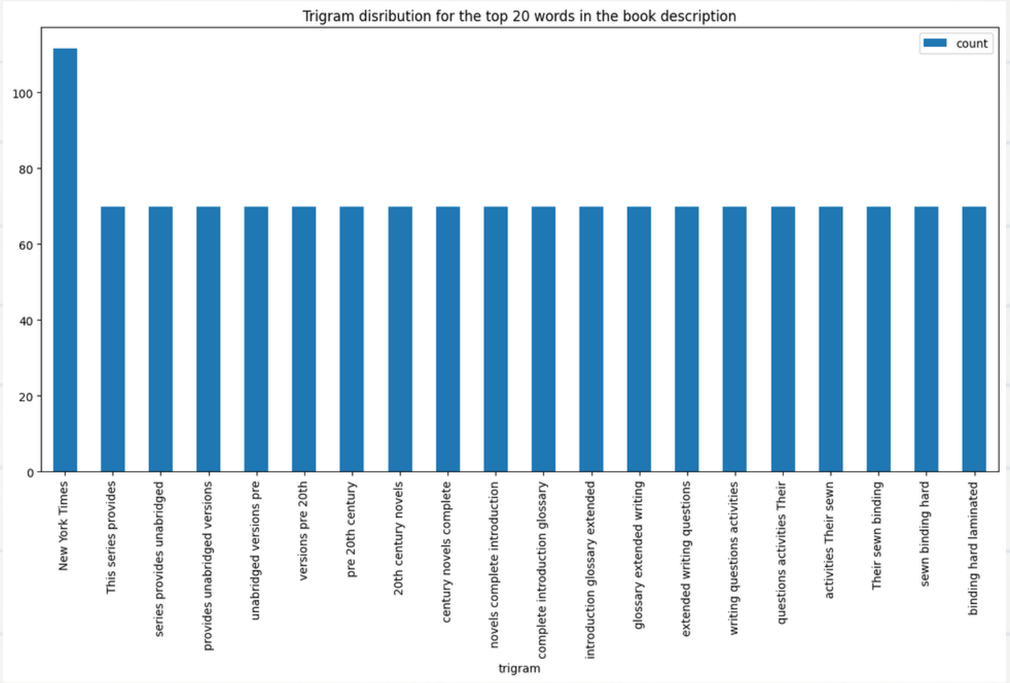
Analyzing the Data





Analyzing the Data

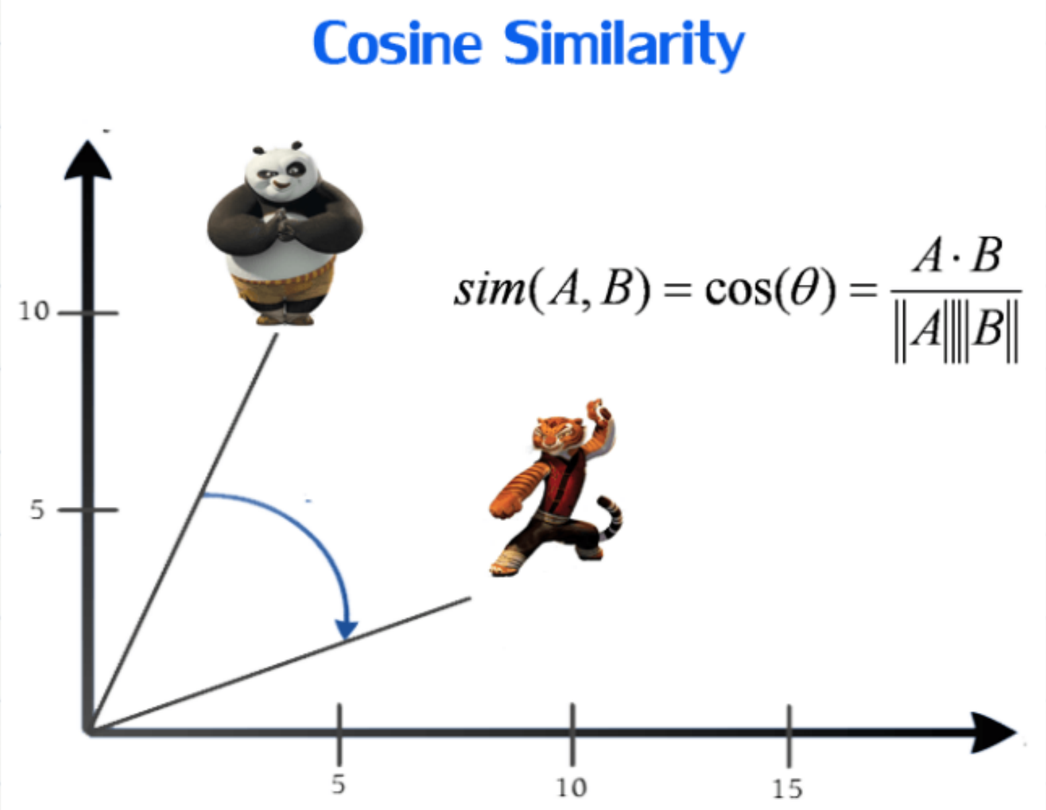




Data Cleaning

```
import nltk
from nltk.corpus import stopwords
import re

nltk.download('stopwords')
from nltk.tokenize import RegexpTokenizer

# Function for removing NonAscii characters
def _removeNonAscii(s):
    return "".join(i for i in s if  ord(i)<128)
# Function for converting into lower case
def make_lower_case(text):
    return text.lower()
# Function for removing stop words
def remove_stop_words(text):
    text = text.split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)
    return text
# Function for removing punctuation
def remove_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    text = tokenizer.tokenize(text)
    text = " ".join(text)
    return text
#Function for removing the html tags
def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

# Applying all the functions in description and storing as a cleaned_desc
merged_dataset['cleaned_desc'] = merged_dataset['description'].apply(_removeNonAscii)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func = make_lower_case)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func = remove_stop_words)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func=remove_punctuation)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func=remove_html)
```

First Model



$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

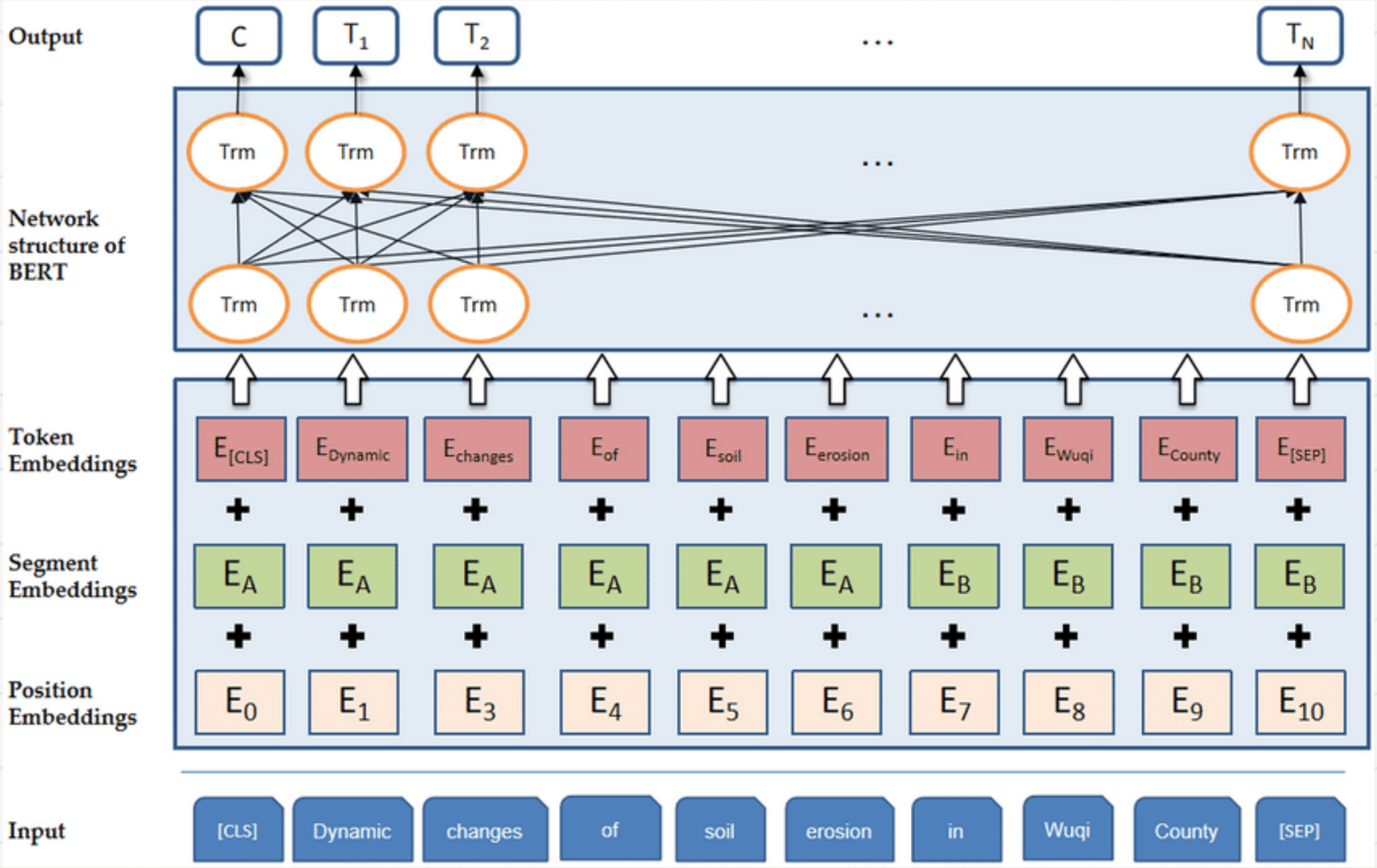
$$IDF(t) = log\frac{N}{1+df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

Applying TF-IDF and Cosine Similarity Method

```python
def recommendByBoth(title):
    # Create a new DataFrame to avoid modifying the original dataset
    data = merged_dataset.copy()

    # Drop duplicates to keep only unique titles
    data = data.drop_duplicates(subset='Title').reset_index(drop=True)

    # Combine title and description into a single feature
    data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

    # Convert the index into series
    indices = pd.Series(data.index, index=data['Title'])

    # Converting the combined feature into vectors using TF-IDF
    tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
    tfidf_matrix = tf.fit_transform(data['combined_features'])

    # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Get the index corresponding to the title
    if title not in indices:
        print(f"Title '{title}' not found.")
        return []

    idx = indices[title]

    # Get the pairwise similarity scores
    sig = list(enumerate(sg[idx]))

    # Exclude the book itself from the similarity scores and sort the books
    sig = sorted(sig, key=lambda x: x[1], reverse=True)
    sig = sig[1:11]  # Get the top 10 most similar books

    # Book indices
    book_indices = [i[0] for i in sig]

    # Top 10 book recommendations
    rec = data[['Title', 'infoLink']].iloc[book_indices]

    # Print the recommended book titles
    for i in rec['Title']:
        print(i)
```

Bert Model


Output
C
T1
T2
...
TN
Network structure of BERT
Trm
Trm
Trm
...
Trm
Trm
Trm
Trm
...
Trm
Token Embeddings
E[CLS]
EDynamic
Echanges
Eof
Esoil
Eerosion
Ein
EWuqi
ECounty
E[SEP]
Segment Embeddings
EA
EA
EA
EA
EA
EA
EB
EB
EB
EB
Position Embeddings
E0
E1
E3
E4
E5
E6
E7
E8
E9
E10
Input
[CLS]
Dynamic
changes
of
soil
erosion
in
Wuqi
County
[SEP]

Applying Bert model

```python
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embeddings(text):
    # Encode text to BERT's format
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512)
    # Get embeddings
    outputs = bert_model(**inputs)
    # Use mean pooling to get a single vector representation
    embeddings = outputs.last_hidden_state.mean(dim=1)
    return embeddings.detach().numpy()
```

```
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 722B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.78MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 6.04MB/s]
config.json: 100% 570/570 [00:00<00:00, 22.5kB/s]
model.safetensors: 100% 440M/440M [00:03<00:00, 136MB/s]
```

```python
[31] def recommend_bert(title, top_n=10):
        # Create a new DataFrame to avoid modifying the original dataset
        data = merged_dataset.copy()

        # Drop duplicates to keep only unique titles
        data = data.drop_duplicates(subset='Title').reset_index(drop=True)

        # Check if the title is in the dataset
        if title not in data['Title'].values:
            print(f"Title '{title}' not found.")
            return []

        # Get the category of the input title
        input_category = data[data['Title'] == title]['categories'].iloc[0]

        # Filter data to include only books in the same category
        category_data = data[data['categories'] == input_category].copy()


        # Print the category and the number of books in this category
        print(f"Category: {input_category}")
        print(f"Number of books in this category: {len(category_data)}")

        category_data['combined_features'] = category_data['Title'] + " " + category_data['cleaned_desc']

        # Get BERT embeddings for the input title
        target_embedding = get_bert_embeddings(category_data[category_data['Title'] == title]['combined_features'].iloc[0])

        # Calculate similarities
        similarities = {}
        for _, row in category_data.iterrows():
            if row['Title'] != title:  # Skip the input title
                book_embedding = get_bert_embeddings(row['combined_features'])
                sim_score = cosine_similarity(target_embedding.reshape(1, -1), book_embedding.reshape(1, -1))[0][0]
                similarities[row['Title']] = sim_score

        # Sort and return top N similar books
        sorted_books = sorted(similarities.items(), key=lambda x: x[1], reverse=True)[:top_n]
        return sorted_books
```

Model Results

```
[ ]  recommendByBoth("Harry Potter and Philosophy: If Aristotle Ran Hogwarts")
```

```
The Irresistible Rise of Harry Potter
Horrible Harry and the Drop of Doom
Horrible Harry's Secret
Faith Journey Through Fantasy Lands
Heads up for Harry
Whatever Happened to Good and Evil?
Philosophy for Dummies
The Lord of the Rings and Philosophy: One Book to Rule Them All (Popular Culture and Philosophy)
The Myth of Magic
The Cambridge Companion to Aquinas (Cambridge Companions to Philosophy)
```

```
recommended_books = recommend_bert("Harry Potter and Philosophy: If Aristotle Ran Hogwarts")
for book in recommended_books:
    print(book)
```

```
Category: ['Fiction']
Number of books in this category: 1377
('The Dreams in the Witch House: And Other Weird Stories (Penguin Classics)', 0.8703593)
('A Simple Story (Penguin Classics)', 0.8650646)
('Edgar Huntly: Or, Memoirs of a Sleep-Walker', 0.8593837)
('Right Behind: A Parody of Last Days Goofiness', 0.8541255)
('Hitting the Skids in Pixeltown: The Phobos Science Fiction Anthology (Volume 2)', 0.853407)
('Forty Stories (Penguin Classics)', 0.85321176)
('Riddle-Master', 0.85053307)
('The Lifted Veil', 0.8501919)
('Lonigan', 0.84961975)
('Siddhartha: An Indian Tale (Penguin Twentieth-Century Classics)', 0.84754115)
```

Content Based Recommendation

- Finding all similarity scores for all books the user read.
- Score justification
- Normalization
- Diversity factor
- Top 10 books

```
20
Is Jesus the Only Savior?
Biblical Preaching: The Development and Delivery of Expository Messages
Now, That's a Good Question!
23 Minutes In Hell: One Man's Story About What He Saw, Heard, and Felt in that Place of Torment
The Jesus Quest: The Third Search for the Jew of Nazareth
New Testament History: A Narrative Account
Daily Life in the United States, 1920-1940: How Americans Lived Through the Roaring Twenties and the Great Depression
Jerusalem Countdown: A Warning to the World
The Race Set Before Us: A Biblical Theology of Perseverance and Assurance
The New Eating Right for a Bad Gut : The Complete Nutritional Guide to Ileitis, Colitis, Crohn's Disease, and Inflammatory Bowel Disease
Greatness of the Kingdom
Slaves, Women & Homosexuals: Exploring the Hermeneutics of Cultural Analysis
The New Testament in Its Literary Environment (Library of Early Christianity)
Word Biblical Commentary Vol. 35a, Luke 1:1-9:20
The Gospel according to Matthew (Pillar New Testament Commentary)
The Theology of the Book of Revelation (New Testament Theology)
The Scandal of the Evangelical Conscience, Why Are Christians Living Just Like the Rest of the World?
Jesus Remembered (Christianity in the Making, Vol. 1)
The Message of the Sermon on the Mount (Bible Speaks Today)
The Book of Genesis: Chapters 18-50 (New International Commentary on the Old Testament)
```

```
Top 10 Sorted Recommendations after Diversity Factor:
Word Biblical Commentary Vol. 33a, Matthew 1-13 (hagner), 483pp (Score: 1.0000)
A Biblical Theology of the New Testament (Score: 0.8682)
Theological Dictionary of the New Testament (Score: 0.7632)
Hebrews: New Testament Commentary (MacArthur New Testament Commentary Series) (Score: 0.7546)
Revelation (Baker Exegetical Commentary on the New Testament) (Score: 0.7441)
Matthew 1-28 MacArthur New Testament Commentary Four Volume Set (shrinkwrapped) (Macarthur New Testament Commentary Serie) (Score: 0.7430)
The New Interpreter's Bible Index (Score: 0.7377)
The Letter of James (Pillar New Testament Commentary) (Score: 0.7149)
The Book of Revelation (New International Commentary on the New Testament) (Score: 0.7146)
The Message of the New Testament: Promises Kept (Score: 0.7038)
```

Evaluation Metric

- We are using our test set to evaluate the performance of the final model.

```
# Evaluate the model on the test set
test_loss = model.evaluate(
    [
        test_set['User_id'],
        test_set['Title'],
    ],
    test_set['review/score']
)
print(f'Test Loss: {test_loss}')

# Make predictions on the test set
test_predictions = model.predict(
    [
        test_set['User_id'],
        test_set['Title'],
    ]
)
```

Evaluation Metric

- After we predicted the review results, we have converted our prediction as 1 (the user will like the book) if the prediction score greater than 3, else 0. We did the same thing for the actual test set scores.
- Finally, we have compared our results and got 88.9 % accuracy with our best model.

```
# Define a threshold for classification
threshold = 3

# Convert the continuous predictions to binary (1 if predicted score >= threshold, 0 otherwise)
binary_predictions = (test_predictions >= threshold).astype(int)

# Evaluate accuracy based on the threshold
accuracy_binary = accuracy_score((test_set['review/score'] >= threshold).astype(int), binary_predictions)
print(f'Accuracy (Threshold={threshold}): {accuracy_binary}')
```

```
Accuracy (Threshold=3): 0.8897544422957638
```

Recommendation Methodology (Using Random Books)

- Given a user_id, 10 random books are chosen from the dataset.
- The model predicts the ratings for these books for the user.
- Books with predicted ratings higher than 3.0 are filtered to recommend to the user.

Example Case (user_id is chosen randomly)

```
1/1 [==============================] - 0s 15ms/step
Recommended Books for User ID (encoded): 1396
                                          Book_Title  Predicted_Score
4                              Robots Have No Tails         4.955386
9                             Case for Christianity         4.809158
3  Zerstorergruppe: A History of V./(Z)LG 1 - I./...         4.559104
6                                   Tenbow (Signet)         4.498399
2                                     Edgard Varese         4.412599
7  Taming The Boss (Marrying The Boss) (Romance, ...         4.357439
0  Three Elegies of Ch'u: An Introduction to the ...         4.121614
1  Mathematical Quickies: 270 Stimulating Problem...         3.831302
5    The voice out of the whirlwind: The book of Job         3.557250
8  Out of Time: Designs for the Twentieth-Century...         3.345959
```

Recommendation Methodology (Content Based Filtering Integration)

- Given a user_id, 10 books are chosen from the dataset based on most similar books user read.
- The model predicts the ratings for these books for the user.
- Books with predicted ratings higher than 3.0 are filtered to recommend to the user.

Example Case (user_id is chosen randomly)

```
[7558]
10
1/1 [==============================] - 0s 20ms/step
Recommended Books for User ID (encoded): [7558]
                                          Book_Title  Predicted_Score
5  Matthew 1-28 MacArthur New Testament Commentar...         4.799201
4  Revelation (Baker Exegetical Commentary on the...         4.776924
1           A Biblical Theology of the New Testament         4.597389
0  Word Biblical Commentary Vol. 33a, Matthew 1-1...         4.593740
9    The Message of the New Testament: Promises Kept         4.565482
2        Theological Dictionary of the New Testament         4.565370
8  The Book of Revelation (New International Comm...         4.562567
3  Hebrews: New Testament Commentary (MacArthur N...         4.504512
6                  The New Interpreter's Bible Index         4.492737
7  The Letter of James (Pillar New Testament Comm...         3.918386
```

Source Code

Importing the dataset

```
#To mount google drive, I put my dataset into my drive
from google.colab import drive
drive.mount('/content/drive/')
```
[2]

```
Mounted at /content/drive/
```

```
#To change directory in google drive
import os
os.chdir('/content/drive/MyDrive/CS464-Project/dataset')
```
[3]

```
books_dataset = pd.read_csv('books_data.csv')
ratings_dataset = pd.read_csv('Books_rating.csv')
```
[4]

Data Pre-Processing

```
# Filter the dataset to include only users with at least 20 reviews
filtered_ratings_dataset = ratings_dataset.groupby('User_id').filter(lambda x: len(x) >= 20)
```
[5] Python

```
from sklearn.preprocessing import LabelEncoder

# Encoding for 'User_id', 'Title', and 'categories'
user_encoder = LabelEncoder()
title_encoder = LabelEncoder()

filtered_ratings_dataset['User_id'] = user_encoder.fit_transform(filtered_ratings_dataset['User_id'])
filtered_ratings_dataset['Title'] = title_encoder.fit_transform(filtered_ratings_dataset['Title'])
```
[6] Python

```
filtered_ratings_dataset.shape
```
[12] Python

```
(548134, 10)
```

Split Data Into Training-Validation-Test Set

```
# Sort the dataset based on 'review/time'
filtered_ratings_dataset.sort_values('review/time', inplace=True)

# Calculate the indices for splitting
total_rows = len(filtered_ratings_dataset)
train_size = int(0.8 * total_rows)
test_size = int(0.1 * total_rows)

# Split the dataset
train_set = filtered_ratings_dataset.iloc[:train_size]
validation_set = filtered_ratings_dataset.iloc[train_size:(train_size + test_size)]
test_set = filtered_ratings_dataset.iloc[(train_size + test_size):]
```
[7] Python

Source Code

Train Model

```
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense, Input, Concatenate, Flatten
from tensorflow.keras.models import Model

# Get the number of unique users, items, and categories
num_users = filtered_ratings_dataset['User_id'].nunique()
num_items = filtered_ratings_dataset['Title'].nunique()

# Define the embedding size
embedding_size = 20

# Define input layers for user, item, and content features
user_input = Input(shape=(1,), name='user_input')
item_input = Input(shape=(1,), name='item_input')

# Embedding layers
user_embedding = Embedding(input_dim=num_users, output_dim=embedding_size, input_length=1)(user_input)
item_embedding = Embedding(input_dim=num_items, output_dim=embedding_size, input_length=1)(item_input)

# Flatten embeddings
user_flat = Flatten()(user_embedding)
item_flat = Flatten()(item_embedding)

# Concatenate the flattened embeddings
concatenated_features = Concatenate()([user_flat, item_flat])

# Dense layer for further processing
dense_layer = Dense(128, activation='relu')(concatenated_features)

# Output layer
output_layer = Dense(1)(dense_layer)

# Create the model
model = Model(inputs=[user_input, item_input], outputs=output_layer)
model.compile(optimizer='adam', loss='mean_squared_error')
```

[8]

```
# Define the number of epochs
num_epochs = 10

model.fit(
    [
        train_set['User_id'],
        train_set['Title'],
    ],
    train_set['review/score'],
    epochs=num_epochs,
    validation_data=(
        [
            validation_set['User_id'],
            validation_set['Title'],
        ],
        validation_set['review/score']
    )
)
```

[9]

Source Code

Test Results

```
# Evaluate the model on the test set
test_loss = model.evaluate(
    [
        test_set['User_id'],
        test_set['Title'],
    ],
    test_set['review/score']
)
print(f'Test Loss: {test_loss}')

# Make predictions on the test set
test_predictions = model.predict(
    [
        test_set['User_id'],
        test_set['Title'],
    ]
)

# Create a DataFrame with user, item, and predicted score
predictions_df = pd.DataFrame({
    'User_id': test_set['User_id'],
    'Title': test_set['Title'],
    'Actual_Score': test_set['review/score'],
    'Predicted_Score': test_predictions.flatten(),
})
```

[24]

```
1713/1713 [==============================] - 4s 2ms/step - loss: 1.2794
Test Loss: 1.2794238328933716
1713/1713 [==============================] - 3s 2ms/step
```

```
# Define a threshold for classification
threshold = 3

# Convert the continuous predictions to binary (1 if predicted score >= threshold, 0 otherwise)
binary_predictions = (test_predictions >= threshold).astype(int)

# Evaluate accuracy based on the threshold
accuracy_binary = accuracy_score((test_set['review/score'] >= threshold).astype(int), binary_predictions)
print(f'Accuracy (Threshold={threshold}): {accuracy_binary}')
```

[25]

```
Accuracy (Threshold=3): 0.8845915277118984
```

Recommond Books to a User

```
# Random chosen user_id
user_id_encoded = np.random.choice(train_set['User_id'].unique())

# Random chosen 10 books
random_book_titles = np.random.choice(train_set['Title'].unique(), size=10, replace=False)

# Formatting user ids and books
user_ids_formatted = np.array([user_id_encoded] * 10, dtype='int32')
book_titles_formatted = np.array(random_book_titles, dtype='int32')

# Run the model on the data
predicted_scores = model.predict([user_ids_formatted, book_titles_formatted])

# Format the result
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(random_book_titles),
    'Predicted_Score': predicted_scores.flatten()
})

# Filter the result
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

# Print the Result
print(f"Recommended Books for User ID (encoded): {user_id_encoded}")
print(recommended_books)
```

[27]

Source Code

```python
#Rastgele bir kodlanmış kullanıcı ID'si seçme
users_with_min_books = merged_dataset.groupby('User_id').filter(lambda x: len(x) >= 20)
user_id_encoded = np.random.choice(users_with_min_books['User_id'].unique())
user_read_books = users_with_min_books[users_with_min_books['User_id'] == user_id_encoded]['Title'].unique()

# Create user_ratings dictionary
user_ratings_data = users_with_min_books[users_with_min_books['User_id'] == user_id_encoded][['Title', 'review/score']]
user_ratings = dict(zip(user_ratings_data['Title'], user_ratings_data['review/score']))

# Dictionary to store accumulated similarity scores
accumulated_sim_scores = {}

print(len(user_read_books))
for user_book in user_read_books:
    print(user_book)
    # Get similarity scores of this book against all books
    sim_scores = get_similarity_scores(user_book)

    # Get the user's rating for this book
    user_rating = user_ratings.get(user_book, 0)

    # Accumulate scores, weighted by the user's rating
    for book_title, score in sim_scores.items():
        if book_title not in user_read_books:  # Exclude books the user has already read
            weighted_score = score * user_rating
            if book_title in accumulated_sim_scores:
                accumulated_sim_scores[book_title] += weighted_score
            else:
                accumulated_sim_scores[book_title] = weighted_score

print(accumulated_sim_scores)
# Normalize the scores
max_score = max(accumulated_sim_scores.values())
normalized_scores = {k: v / max_score for k, v in accumulated_sim_scores.items()}

# Sort books based on normalized scores
sorted_recommendations = sorted(normalized_scores.items(), key=lambda x: x[1], reverse=True)

# Select top 10 recommendations
top_recommendations = sorted_recommendations[:10]

final_recommendations = []
diversity_factor = 0.5  # Adjust as needed

for book, score in sorted_recommendations:
    too_similar = any(get_similarity_score(book, rec) > diversity_factor for rec in final_recommendations)

    if not too_similar:
        final_recommendations.append(book)
        if len(final_recommendations) == 10:
            break

# Pair the final recommendations with their scores for sorting
final_recommendations_with_scores = [(book, normalized_scores[book]) for book in final_recommendations]

# Sort the final recommendations based on scores
sorted_final_recommendations = sorted(final_recommendations_with_scores, key=lambda x: x[1], reverse=True)

# Print the sorted final recommendations
print("Top 10 Sorted Recommendations after Diversity Factor:")
for rec in sorted_final_recommendations:
    print(f"{rec[0]} (Score: {rec[1]:.4f})")
```

```python
# Content + Collaborative
# Extract book titles from sorted_final_recommendations

user_id_encoded = np.array([user_id_encoded])
user_id_encoded = user_encoder.transform(user_id_encoded)
print(user_id_encoded)

recommended_book_titles = [rec[0] for rec in sorted_final_recommendations]

# Ensure we have 10 titles (or all of them if less than 10)
book_titles_to_predict = recommended_book_titles[:10]

# Convert book titles to their encoded format
book_titles_formatted = title_encoder.transform(recommended_book_titles[:10])
print(len(book_titles_formatted))

# Prepare user IDs in the correct format
user_ids_formatted = np.array([user_id_encoded] * 10, dtype='int32')

# Make predictions using the collaborative filtering model
predicted_scores = model.predict([user_ids_formatted, book_titles_formatted])

# Create a DataFrame with the recommended book titles and predicted scores
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(book_titles_formatted),
    'Predicted_Score': predicted_scores.flatten()
})

#Puanı 3.0'dan yüksek olan kitapları filtreleme ve azalan sırada sıralama
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

#Önerilen kitapları ve tahmin edilen puanları kullanıcıya sunma
print(f"Recommended Books for User ID (encoded): {user_id_encoded}")
print(recommended_books)
```

Source Code

```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.metrics.pairwise import cosine_similarity
     import pandas as pd

     # Create a new DataFrame to avoid modifying the original dataset
     data = merged_dataset.copy()

     # Drop duplicates to keep only unique titles
     data = data.drop_duplicates(subset='Title').reset_index(drop=True)

     # Combine title and description into a single feature
     data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

     indices = pd.Series(data.index, index=data['Title'])

     # Converting the combined feature into vectors using TF-IDF
     tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
     tfidf_matrix = tf.fit_transform(data['combined_features'])

     # Create a Series mapping book titles to their index
     book_index = pd.Series(range(len(data)), index=data['Title'])
```

```
def get_similarity_scores(title):
    # Check if the title is in the dataset
    if title not in indices:
        print(f"Title '{title}' not found.")
        return {}

    idx = indices[title]

    # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Get the pairwise similarity scores
    sig = list(enumerate(sg[idx]))

    # Exclude the book itself from the similarity scores
    sig = sig[1:]  # Exclude the book itself

    # Create a dictionary of title and similarity score
    similarity_scores = {data.iloc[i]['Title']: score for i, score in sig if i != idx}

    return similarity_scores

def get_similarity_score(book1, book2):
    idx1, idx2 = indices[book1], indices[book2]
    vector1, vector2 = tfidf_matrix[idx1], tfidf_matrix[idx2]

    # Calculate and return the cosine similarity score
    return cosine_similarity(vector1, vector2)[0][0]
```

```
def recommendByBoth(title):
    # Create a new DataFrame to avoid modifying the original dataset
    data = merged_dataset.copy()

    # Drop duplicates to keep only unique titles
    data = data.drop_duplicates(subset='Title').reset_index(drop=True)

    # Combine title and description into a single feature
    data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

    # Convert the index into series
    indices = pd.Series(data.index, index=data['Title'])

    # Converting the combined feature into vectors using TF-IDF
    tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
    tfidf_matrix = tf.fit_transform(data['combined_features'])

    # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Get the index corresponding to the title
    if title not in indices:
        print(f"Title '{title}' not found.")
        return []

    idx = indices[title]

    # Get the pairwise similarity scores
    sig = list(enumerate(sg[idx]))

    # Exclude the book itself from the similarity scores and sort the books
    sig = sorted(sig, key=lambda x: x[1], reverse=True)
    sig = sig[1:11]  # Get the top 10 most similar books

    # Book indices
    book_indices = [i[0] for i in sig]

    # Top 10 book recommendations
    rec = data[['Title', 'infoLink']].iloc[book_indices]

    # Print the recommended book titles
    for i in rec['Title']:
        print(i)
```

Thank you for Listening

Any questions, comments, or recommondations?