ДОДАТОК А

Програмний код

Лістинг А.1 – Програмний код файлу main.py

```
import pandas as pd
import datetime
import nltk
import spacy

import re
from time import time
from collections import defaultdict

import logging  # Setting up the loggings to monitor gensim
logging.basicConfig(format="%(levelname)s - %(asctime)s:
%(message)s", datefmt= '%H:%M:%S', level=logging.INFO)

data = pd.read_json("arxivData.json")
print(list(data))
data.shape
data.head()
data.isnull().sum()
abstracts_only = pd.read_pickle('cleaned_abstract.pkl')
recommend_df = pd.read_pickle('recommend_df.pkl')
from gensim.models.phrases import Phrases, Phraser
sent = [row.split() for row in abstracts_only['clean'] if row
is not None]
phrases = Phrases(sent, min_count= 20, progress_per=10000) #
min_count=30
bigram = Phraser(phrases)
sentences = bigram[sent]
### get trigrams, by including found bigrams/phrases
phrases2 = Phrases(sent, min_count= 20, progress_per=10000) #
min_count=30
trigram = Phraser(phrases2)

sentences = trigram[sentences]
import multiprocessing

from gensim.models import Word2Vec
w2v_model = Word2Vec.load("word2vec_model.model")
import numpy as np
def get_sentence_vector(words, model):
    vectors = [model.wv[word] for word in words if word in
model.wv]
    return np.mean(vectors, axis=0) if vectors else
np.zeros(model.vector_size)

# Convert each sentence into a numerical vector
```

```
sentence_vectors = [get_sentence_vector(sentence, w2v_model)
for sentence in sentences]

# Compute cosine similarity
from sklearn.metrics.pairwise import cosine_similarity
cosine_sim = cosine_similarity(sentence_vectors,
sentence_vectors)
print (cosine_sim)
indices = pd.Series(recommend_df.index,
index=recommend_df['abstract']).drop_duplicates()

def get_recommendations(abstract, cosine_sim, indices):
    # Get the index of the movie that matches the title
    idx = indices[abstract]
    # Get the pairwsie similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)
    # Get the scores for 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    paper_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    return recommend_df['abstract'].iloc[paper_indices]

base_idx = 1
# get the first paper in the recommendation dataset
paper = recommend_df['abstract'][base_idx]
category = recommend_df['categories'][base_idx]
# Generate recommendations
print(get_recommendations(paper, cosine_sim, indices))
print('')
print('-----------------------------------------------------
---------------------------------')
similar_paper_idx = 6599
sim_paper = recommend_df['abstract'][similar_paper_idx]
sim_category = recommend_df['categories'][similar_paper_idx]
print('The article in question for recommendation has the
following category assigned to it:')
print('')
print(category)
print('')
print('See the full paper abstract below:')
print('')
print(paper)
print('-----------------------------------------------------
---------------------------------')
print('-----------------------------------------------------
---------------------------------')
print('-----------------------------------------------------
---------------------------------')
```

```
print('-------------------------------------------------------
--------------------------------')
print('')
print('The article in question for recommendation has the
following category assigned to it:')
print('')
print(sim_category)
print('')
print('-------------------------------------------------------
--------------------------------')
print('')
print(sim_paper)
# get the first paper in the recommendation dataset
paper = data['summary'][500]

# Generate recommendations
print(get_recommendations(paper, cosine_sim, indices))
corpus = recommend_df['abstract'].head(2000)
start = time.time()
# Initialize an instance of tf-idf Vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Generate the tf-idf vectors for the corpus
tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)
pca = PCA(n_components=0.95)
df_reduced = pca.fit_transform(tfidf_matrix.toarray())
df_reduced
k = 20 # optimal k found in elbow plot
kmeans = KMeans(n_clusters=k, random_state=42)
y_pred = kmeans.fit_predict(df_reduced)
df = pd.concat([
    pd.Series(corpus),
    pd.Series(y_pred, name='cluster'),
    recommend_df['title'].head(2000)
], axis=1)
df
tsne = TSNE(verbose=1, perplexity=100, random_state=42)
X_embedded = tsne.fit_transform(df_reduced)
fig = px.scatter(df, x=X_embedded[:,0], y=X_embedded[:,1],
color=y_pred.astype(str),
                 hover_data=['title',],
                 height= 1000, width=1000,
                title = "t-SNE with Kmeans Labels")
fig.show()
category_cluster_df = pd.concat([
    recommend_df['categories'].head(2000),
    pd.Series(y_pred, name='cluster')],
    axis=1)
category_cluster_df
def get_cluster_recommendations(paper_idx: int=0):
    category = recommend_df['categories'][paper_idx]
    query = str("categories == '" + str(category) + str("'"))
```

```
    best_cluster =
category_cluster_df.query(query).cluster.value_counts().reset_
index().head(1)
    best_cluster.columns = ['cluster', 'num_records']
    query = str("cluster == " + str(best_cluster['cluster']))
    results = df.query('cluster == 15')[['abstract', 'title']]
    return results
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	68 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Грішаєва А.М.			Дослідження та застосування методів NLP для вирішення проблеми холодного старту в рекомендаційних системах	Шифр групи	Код напр./спец.
Перевірив		Рябова Н.В.				СШІм-22-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					