‘wikiHow’: NLP-Based Text Summarization and Semantic Analysis

Mt. SAC CISB 63 Final Project Fall 2023

By

Paul Sandeen

Introduction

A wiki (pronounced “WIK-ee”) is an Internet website containing information created and maintained by its readership community. The website wikiHow is a knowledge base that contains “how-to” information on a wide variety of topics including Hobbies and Crafts, Pets and Animals, Philosophy and Religion, Relationships, and Health. The goal of wikiHow is to allow readers to acquire new skills by reading articles with step-by-step illustrated instructions on how to perform various tasks related to topic categories.

The wikiHow Dataset is a large collection of articles from the wikiHow knowledge base saved in a Comma-Separated Value (.csv) file format. There are three columns in the CSV file:

- Title: The title of the article on wikiHow.
- Headline: The summary of the steps in the step-by-step description.
- Text: The body of the article containing step-by-step directions.

This project will use Natural Language Processing (NLP) techniques to analyze the articles in the wikiHow Dataset to identify related topics and provide summarization of the articles.

Intended Audience

This project will benefit anyone with an interest in NLP techniques for text summarization and topic identification. Proficiency with the Python programming language is required. Familiarity with the basic concepts of text summarization and topic identification is assumed.

Materiels and Methods

This project uses the Python programming language running in the Anaconda environment.

Associated Python data science libraries: Numpy, pandas, Matplotlib, Seaborn.

Associated Python NLP libraries: scikit-learn, NLTK, Word2Vec, Seq2Seq, LDA, TF-DFI, spaCy

The project was composed as a Jupyter Notebook.

Data Source

The single data file used for the project is:

- wikihowAll.csv

File size: ~605 MB

The data file can be downloaded from:

https://github.com/mahnazkoupaee/WikiHow-Dataset (https://github.com/mahnazkoupaee/WikiHow-Dataset)

Acquire the Necessary Software Packages

In [1]:
```
# Import the Python Data Science libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import string
import ast
```

In [2]:
```
# Import the Python scikit-learn libraries
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.datasets import make_classification
```

In [3]:
```
# Import the Python NLP libraries

from textblob import TextBlob

import tensorflow as tf

import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk

import re

from datetime import date
from datetime import datetime

from wordcloud import WordCloud

import spacy
from spacy import displacy

from gensim import corpora, models
import gensim.downloader as api

from string import punctuation

import transformers
from transformers import pipeline

from tqdm import tqdm
from gensim.models import Word2Vec
import umap

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

from sklearn.decomposition import LatentDirichletAllocation
from sklearn.metrics.pairwise import cosine_similarity

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [4]:
```
# Disbable Jupyter notebook warnings
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
# Download the NLTK databases
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\psand\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[5]: True

Load the Textual Data

In [6]:
```
# Load the articles into a pandas DataFrame
articles_df = pd.read_csv("data/wikihowAll.csv")
```

Exploratory Data Analysis (EDA)

Display the head of the DataFrame

In [7]:
```
# User complaints dictionary
articles_df.head()
```

Out[7]:

	headline	title	text
0	\nKeep related supplies in the same area.,\nMa...	How to Be an Organized Artist1	If you're a photographer, keep all the necess...
1	\nCreate a sketch in the NeoPopRealist manner ...	How to Create a Neopoprealist Art Work	See the image for how this drawing develops s...
2	\nGet a bachelor's degree.,\nEnroll in a studi...	How to Be a Visual Effects Artist1	It is possible to become a VFX artist without...
3	\nStart with some experience or interest in ar...	How to Become an Art Investor	The best art investors do their research on t...
4	\nKeep your reference materials, sketches, art...	How to Be an Organized Artist2	As you start planning for a project or work, ...

Note: The columns 'headline' and 'title' should be swapped to get the 'title' > 'headline' > 'text' order.

In [8]:
```
# Swap the positions of the 'headline' and 'title' columns
move_column = articles_df.pop('title')
articles_df.insert(0, 'title', move_column)
articles_df.head()
```

Out[8]:

	title	headline	text
0	How to Be an Organized Artist1	\nKeep related supplies in the same area.,\nMa...	If you're a photographer, keep all the necess...
1	How to Create a Neopoprealist Art Work	\nCreate a sketch in the NeoPopRealist manner ...	See the image for how this drawing develops s...
2	How to Be a Visual Effects Artist1	\nGet a bachelor's degree.,\nEnroll in a studi...	It is possible to become a VFX artist without...
3	How to Become an Art Investor	\nStart with some experience or interest in ar...	The best art investors do their research on t...
4	How to Be an Organized Artist2	\nKeep your reference materials, sketches, art...	As you start planning for a project or work, ...

Discribe the contents of the DataFrame

In [9]:
```
# Describe the DataFrame
articles_df.describe
```

Out[9]:
```
<bound method NDFrame.describe of                                                  title  \
0                  How to Be an Organized Artist1
1           How to Create a Neopoprealist Art Work
2               How to Be a Visual Effects Artist1
3                    How to Become an Art Investor
4                  How to Be an Organized Artist2
...                                           ...
215360               How to Pick a Stage Name3
215361               How to Pick a Stage Name4
215362                 How to Identify Prints1
215363                 How to Identify Prints2
215364                 How to Identify Prints3

                                                headline  \
0       \nKeep related supplies in the same area.,\nMa...
1       \nCreate a sketch in the NeoPopRealist manner ...
2       \nGet a bachelor's degree.,\nEnroll in a studi...
3       \nStart with some experience or interest in ar...
4       \nKeep your reference materials, sketches, art...
...                                                   ...
215360  \nConsider changing the spelling of your name....
215361  \nTry out your name.,\nDon't legally change yo...
215362  \nUnderstand the process of relief printing.,\...
215363  \nUnderstand the process of intaglio printing....
215364  \nUnderstand the different varieties of lithog...

                                                    text
0        If you're a photographer, keep all the necess...
1        See the image for how this drawing develops s...
2        It is possible to become a VFX artist without...
3        The best art investors do their research on t...
4        As you start planning for a project or work, ...
...                                                   ...
215360   If you have a name that you like, you might f...
215361   Your name might sound great to you when you s...
215362   Relief printing is the oldest and most tradit...
215363   Intaglio is Italian for "incising," and corr...
215364   Lithography is a big term often used to refer...

[215365 rows x 3 columns]>
```

Display the info for the DataFrame

In [10]:
```
# Display the DataFram info
articles_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215365 entries, 0 to 215364
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   title     215364 non-null  object
 1   headline  214547 non-null  object
 2   text      214294 non-null  object
dtypes: object(3)
memory usage: 4.9+ MB
```

Display the 'title', 'headline', and 'text' of the first article

In [11]:
```
# Display the first article 'title'
articles_df.title[0]
```

Out[11]:
```
'How to Be an Organized Artist1'
```

In [12]:
```
# Display the first article 'headline'
articles_df.headline[0]
```

Out[12]:
```
'\nKeep related supplies in the same area.,\nMake an effort to clean a dedicated workspace after every sessio
n.,\nPlace loose supplies in large, clearly visible containers.,\nUse clotheslines and clips to hang sketches,
photos, and reference material.,\nUse every inch of the room for storage, especially vertical space.,\nUse cha
lkboard paint to make space for drafting ideas right on the walls.,\nPurchase a label maker to make your organ
ization strategy semi-permanent.,\nMake a habit of throwing out old, excess, or useless stuff each month.'
```

In [13]:
```
# Display the first article 'text'
articles_df.text[0]
```

Out[13]:
```
" If you're a photographer, keep all the necessary lens, cords, and batteries in the same quadrant of your hom
e or studio. Paints should be kept with brushes, cleaner, and canvas, print supplies should be by the ink, et
c. Make broader groups and areas for your supplies to make finding them easier, limiting your search to a much
smaller area. Some ideas include:\n\n\nEssential supplies area -- the things you use every day.\nInspiration a
nd reference area.\nDedicated work area .\nInfrequent or secondary supplies area, tucked out of the way.;\n, T
his doesn't mean cleaning the entire studio, it just means keeping the area immediately around the desk, ease
l, pottery wheel, etc. clean each night. Discard trash or unnecessary materials and wipe down dirty surfaces.
Endeavor to leave the workspace in a way that you can sit down the next day and start working immediately, wit
hout having to do any work or tidying.\n\n\nEven if the rest of your studio is a bit disorganized, an organize
d workspace will help you get down to business every time you want to make art.\n\n, As visual people, a lot o
f artist clutter comes from a desire to keep track of supplies visually instead of tucked out of sight. By usi
ng jars, old glasses, vases, and cheap, clear plastic drawers, you can keep things in sight without leaving it
strewn about haphazardly. Some ideas, beyond those just mentioned, include:\n\n\nCanvas shoe racks on the back
of the door\nWine racks with cups in each slot to hold pens/pencils.\nPlastic restaurant squirt bottles for pa
int, pigment, etc., Simply string up the wires across a wall or along the ceiling and use them to hold essenti
al papers that you don't want to cut or ruin with tacks or tape. Cheap and easy, this is also a good way to ha
ndle papers and ideas you touch regularly or need to pin up and down for inspiration., Shelving is an artist's
best friend and is a cheap and easy way to get more room in your studio or art space. Don't be afraid to get u
p high either, especially for infrequently used supplies. The upper reaches of the room are often the most und
er-utilized, but provide vital space for all your tools and materials., Turning one wall into a chalkboard giv
es you a perfect space for ideas, sketches, and planning without requiring extra equipment or space. You can e
ven use it for smaller areas. Paint over jars or storage equipment, allowing you to relabel them with chalk as
your needs change.\n\n, A lot of disorganization comes when you keep moving the location of things, trying to
optimize your space by reorganizing frequently. This usually has the opposite effect, leading to lost items an
d uncertainty when cleaning, but an afternoon with a label maker can solve everything. Instead of spending all
of your mental energy looking for or storing things, you can just follow the labels, freeing your mind to thin
k about art., Once a month, do a purge of your studio. If it isn't essential or part of a project, either thro
w it out or file it away for later. Artists are constantly making new things, experimenting, and making a mes
s. This is a good thing, but only if you set aside time to declutter. It may not be fun at the moment, but it
is a lot more fun than spending 30 minutes digging through junk to find the right paint or an old sketch.\n\n
\nDon't be sentimental here. If you haven't used it in the last six months there is little chance you'll use i
t in the next six months. Toss it.\n\n"
```

Result: There are 215,364 articles in the DataFrame. All the data is text and there is no missing data.

The 'headline' provides a summary of each step that is detailed in the article 'text'. The article 'text' is ideal for summarization and comparison with the article 'headline'.

Apply NLP Techniques

Part 1: Gensim and Topic Modeling

The Gensim library can be used to analyze the ‘text’ column in order perform the following:

- tokenization (create tokens from the individual words)
- create a dictionary from the tokens
- create a Bag of Words (BOW) representation
- use Latent Dirichlet Allocation (LDA) for topic modeling.

In [14]:
```
# Define the stopwords and punctuation
stop_words = set(stopwords.words('english'))
punctuation = set(string.punctuation)

# Define the input string as the text from the first article
input_string = articles_df.at[0, 'text']

# Tokenize the input string into lower case words
words = word_tokenize(input_string.lower())

# Remove stopwords and punctuation
filtered_words = [word for word in words if word not in stop_words and word not in punctuation]
```

Create a Gensim Dictionary

In [15]:
```
# Create a Gensim dictionary from filtered_words
dictionary = corpora.Dictionary([filtered_words])

# Convert the filtered_words into Python list of tokens for doc2bow
filtered_tokens = [filtered_words]
```

Create a Gensim corpus using Bag of Words (BOW)

In [16]:
```
# Create the Gensim corpus using Bag of Words (BOW)
corpus = [dictionary.doc2bow(tokens) for tokens in filtered_tokens]

# Print the tokens and their IDs in the dictionary
print("Tokens and their IDs:")
print(dictionary.token2id)
```

```
Tokens and their IDs:
{"'ll": 0, "'re": 1, "'s": 2, '--': 3, '30': 4, 'across': 5, 'afraid': 6, 'afternoon': 7, 'allowing': 8, 'alon
g': 9, 'also': 10, 'area': 11, 'areas': 12, 'around': 13, 'art': 14, 'art.': 15, 'artist': 16, 'artists': 17,
'aside': 18, 'away': 19, 'back': 20, 'batteries': 21, 'best': 22, 'beyond': 23, 'bit': 24, 'bottles': 25, 'bro
ader': 26, 'brushes': 27, 'business': 28, 'canvas': 29, 'ceiling': 30, 'chalk': 31, 'chalkboard': 32, 'chanc
e': 33, 'change': 34, 'cheap': 35, 'clean': 36, 'cleaner': 37, 'cleaning': 38, 'clear': 39, 'clutter': 40, 'co
mes': 41, 'constantly': 42, 'cords': 43, 'cups': 44, 'cut': 45, 'day': 46, 'declutter': 47, 'dedicated': 48,
'desire': 49, 'desk': 50, 'digging': 51, 'dirty': 52, 'discard': 53, 'disorganization': 54, 'disorganized': 5
5, 'door': 56, 'drawers': 57, 'easel': 58, 'easier': 59, 'easy': 60, 'effect': 61, 'either': 62, 'endeavor': 6
3, 'energy': 64, 'entire': 65, 'equipment': 66, 'especially': 67, 'essential': 68, 'etc': 69, 'etc.': 70, 'eve
n': 71, 'every': 72, 'everything': 73, 'experimenting': 74, 'extra': 75, 'file': 76, 'find': 77, 'finding': 7
8, 'follow': 79, 'freeing': 80, 'frequently': 81, 'friend': 82, 'fun': 83, 'get': 84, 'gives': 85, 'glasses':
86, 'good': 87, 'groups': 88, 'handle': 89, 'haphazardly': 90, 'help': 91, 'high': 92, 'hold': 93, 'home': 94,
'ideas': 95, 'immediately': 96, 'include': 97, 'infrequent': 98, 'infrequently': 99, 'ink': 100, 'inspiratio
n': 101, 'inspiration.': 102, 'instead': 103, 'items': 104, 'jars': 105, 'junk': 106, 'keep': 107, 'keeping':
108, 'kept': 109, 'label': 110, 'labels': 111, 'last': 112, 'later': 113, 'leading': 114, 'leave': 115, 'leavi
ng': 116, 'lens': 117, 'limiting': 118, 'little': 119, 'location': 120, 'looking': 121, 'lost': 122, 'lot': 12
3, 'make': 124, 'maker': 125, 'making': 126, 'materials': 127, 'materials.': 128, 'may': 129, 'mean': 130, 'me
ans': 131, 'mental': 132, 'mentioned': 133, 'mess': 134, 'mind': 135, 'minutes': 136, 'moment': 137, 'month':
138, 'months': 139, 'moving': 140, 'much': 141, "n't": 142, 'necessary': 143, 'need': 144, 'needs': 145, 'ne
w': 146, 'next': 147, 'night': 148, 'often': 149, 'old': 150, 'one': 151, 'opposite': 152, 'optimize': 153, 'o
rganized': 154, 'paint': 155, 'paints': 156, 'papers': 157, 'part': 158, 'pens/pencils': 159, 'people': 160,
'perfect': 161, 'photographer': 162, 'pigment': 163, 'pin': 164, 'planning': 165, 'plastic': 166, 'pottery': 1
67, 'print': 168, 'project': 169, 'provide': 170, 'purge': 171, 'quadrant': 172, 'racks': 173, 'reaches': 174,
'reference': 175, 'regularly': 176, 'relabel': 177, 'reorganizing': 178, 'requiring': 179, 'rest': 180, 'resta
urant': 181, 'right': 182, 'room': 183, 'ruin': 184, 'search': 185, 'secondary': 186, 'sentimental': 187, 'se
t': 188, 'shelving': 189, 'shoe': 190, 'sight': 191, 'simply': 192, 'sit': 193, 'six': 194, 'sketch': 195, 'sk
etches': 196, 'slot': 197, 'smaller': 198, 'solve': 199, 'space': 200, 'spending': 201, 'squirt': 202, 'star
t': 203, 'storage': 204, 'storing': 205, 'strewn': 206, 'string': 207, 'studio': 208, 'supplies': 209, 'surfac
es': 210, 'tacks': 211, 'tape': 212, 'thing': 213, 'things': 214, 'think': 215, 'throw': 216, 'tidying': 217,
'time': 218, 'tools': 219, 'toss': 220, 'touch': 221, 'track': 222, 'trash': 223, 'trying': 224, 'tucked': 22
5, 'turning': 226, 'uncertainty': 227, 'under-utilized': 228, 'unnecessary': 229, 'upper': 230, 'use': 231, 'u
sed': 232, 'using': 233, 'usually': 234, 'vases': 235, 'visual': 236, 'visually': 237, 'vital': 238, 'wall': 2
39, 'want': 240, 'way': 241, 'wheel': 242, 'wine': 243, 'wipe': 244, 'wires': 245, 'without': 246, 'work': 24
7, 'working': 248, 'workspace': 249}
```

Create a Latent Dirichlet Allocation (LDA) Model for Topic Modeling

In [17]:
```
# Create the LDA model for topic modeling
lda_model = models.LdaModel(corpus, num_topics=1, id2word=dictionary, passes=15)
```

In [18]:
```
# Display the LDA topics
for idx, topic in lda_model.print_topics(-1):
    print(f'Topic: {idx} \nWords: {topic}\n')
```

```
Topic: 0
Words: 0.012*"n't" + 0.012*"area" + 0.012*"supplies" + 0.010*"space" + 0.010*"studio" + 0.010*"things" + 0.008
*"keep" + 0.008*"use" + 0.008*"ideas" + 0.008*"way"
```

Compare to the LDA topics to human-generated subject headlines

In [19]:
```
# Print the article headline
articles_df['headline'][0]
```

Out[19]:
```
'\nKeep related supplies in the same area.,\nMake an effort to clean a dedicated workspace after every sessio
n.,\nPlace loose supplies in large, clearly visible containers.,\nUse clotheslines and clips to hang sketches,
photos, and reference material.,\nUse every inch of the room for storage, especially vertical space.,\nUse cha
lkboard paint to make space for drafting ideas right on the walls.,\nPurchase a label maker to make your organ
ization strategy semi-permanent.,\nMake a habit of throwing out old, excess, or useless stuff each month.'
```

Result Interpretation: Many of the words from the LDA model of the text body appear in the human-generated article headline ('supplies', 'area', 'space', 'ideas').

Part 2: Named Entity Recognition (NER) to View Entities

The purpose of Named Entity Recognition (NER) is to identify named entities in a body of text, such as:

- Organizations
- Dates
- Locations
- People

This section will compare the NER results visually using spaCy for article text with the section headlines visually to see the similarities between the two.

Prepare spaCy and load the text

In [20]:
```
# Configure the spaCy library for text rendering
nlp = spacy.load('en_core_web_sm')
```

In [21]:
```
# Load the first article 'text' and process it for spaCy
doc_text = nlp(articles_df.at[0, 'text'])

# Render the output text with named entities highlighted
displacy.render(doc_text, style='ent', jupyter=True)
```

If you're a photographer, keep all the necessary lens, cords, and batteries in the same quadrant of your home or studio. Paints should be kept with brushes, cleaner, and canvas, print supplies should be by the ink, etc. Make broader groups and areas for your supplies to make finding them easier, limiting your search to a much smaller area. Some ideas include:

Essential supplies area -- the things you use every day DATE .

Inspiration and reference area.

Dedicated work area .

Infrequent or secondary supplies area, tucked out of the way.;

, This doesn't mean cleaning the entire studio, it just means keeping the area immediately around the desk, easel, pottery wheel, etc. clean each night TIME . Discard PERSON trash or unnecessary materials and wipe down dirty surfaces. Endeavor to leave the workspace in a way that you can sit down the next day DATE and start working immediately, without having to do any work or tidying.

Even if the rest of your studio is a bit disorganized, an organized workspace will help you get down to business every time you want to make art.

, As visual people, a lot of artist clutter comes from a desire to keep track of supplies visually instead of tucked out of sight. By using jars, old glasses, vases, and cheap, clear plastic drawers, you can keep things in sight without leaving it strewn about haphazardly. Some ideas, beyond those just mentioned, include:

Canvas PRODUCT shoe racks on the back of the door

Wine racks with cups in each slot to hold pens/pencils.

Plastic restaurant squirt bottles for paint, pigment, etc., Simply string up the wires across a wall or along the ceiling and use them to hold essential papers that you don't want to cut or ruin with tacks or tape. Cheap and easy, this is also a good way to handle papers and ideas you touch regularly or need to pin up and down for inspiration., Shelving is an artist's best friend and is a cheap and easy way to get more room in your studio or art space. Don't be afraid to get up high either, especially for infrequently used supplies. The upper reaches of the room are often the most under-utilized, but provide vital space for all your tools and materials., Turning one CARDINAL wall into a chalkboard gives you a perfect space for ideas, sketches, and planning without requiring extra equipment or space. You can even use it for smaller areas. Paint over jars or storage equipment, allowing you to relabel them with chalk as your needs change.

, A lot of disorganization comes when you keep moving the location of things, trying to optimize your space by reorganizing frequently. This usually has the opposite effect, leading to lost items and uncertainty when cleaning, but an afternoon TIME with a label maker can solve everything. Instead of spending all of your mental energy looking for or storing things, you can just follow the labels, freeing your mind to think about art., Once a month, do a purge of your studio. If it isn't essential or part of a project, either throw it out or file it away for later. Artists are constantly making new things, experimenting, and making a mess. This is a good thing, but only if you set

aside time to declutter. It may not be fun at the moment, but it is a lot more fun than spending 30 minutes TIME digging through junk to find the right paint or an old sketch.

Don't be sentimental here. If you haven't used it in the last six months DATE there is little chance you'll use it in the next six months DATE . Toss it.

In [22]:
```
# Load the first article 'headline' and process it for spaCy
doc_text = nlp(articles_df.at[0, 'headline'])

# Render the output text with named entities highlighted
displacy.render(doc_text, style='ent', jupyter=True)
```

Keep related supplies in the same area.,

Make an effort to clean a dedicated workspace after every session.,

Place loose supplies in large, clearly visible containers.,

Use clotheslines and clips to hang sketches, photos, and reference material.,

Use every inch of the room for storage, especially vertical space.,

Use chalkboard paint to make space for drafting ideas right on the walls.,

Purchase PERSON a label maker to make your organization strategy semi-permanent.,

Make a habit of throwing out old, excess, or useless stuff each month DATE .

Result Interpretation: NER tagging was not helpful in comparing the contents of the article text with the article summary headlines. The article headlines were too general and did not contain specific information related to the article text to associate meaning between the two.

Part 3: Hugging Face Transformers and Text Summarization

Hugging Face is an Python library for a type of deep learning models called transformers that can be used for tokenization and text summarization.

Hugging Face provides:

- A wide selection of pre-trained models
- The transformers library, a series of APIs to work with text
- Tokenizers to break up large bodies of text into single words
- Pipelines to process text without writing substantial code

Create a text classification pipeline using a pre-trained model

In [23]:
```
# Use the first article 'text' to be summarized
text = articles_df.at[0, 'text']
```

Create the text summarization pipeline using the t5-base (Text-To-Text Transfer Transformer) pre-trained model

In [24]:
```
# Create the summarization pipline
nlp = pipeline(task='summarization', model='t5-base')
```

```
All PyTorch model weights were used when initializing TFT5ForConditionalGeneration.

All the weights of TFT5ForConditionalGeneration were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFT5ForCon
ditionalGeneration for predictions without further training.
```

Perform the text summarization of the input text

In [25]:
```
# Text summarization with a max length of 100 characters and a min length of 30 characters
summarized_text = nlp(text, max_length=100, min_length=30)

# Print the summarized text
summarized_text[0]['summary_text']
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (813 > 512).
Running this sequence through the model will result in indexing errors
```

Out[25]:
```
"if you're a photographer, keep all the necessary supplies in the same quadrant of your home or studio . make
broader groups and areas for your supplies to make finding them easier . an organized workspace will help you
get down to business every time you want to make art ."
```

In [26]:
```
# Print the length of the original text and summary text
print(f"Length of original text: {len(text)}")
print(f"Length of summary text: {len(summarized_text[0]['summary_text'])}")
print(f"Length of human-generated article headline: {len(articles_df.at[0, 'headline'])}")
```

```
Length of original text: 3425
Length of summary text: 273
Length of human-generated article headline: 536
```

Result Interpretation: Note that the original input text length was 3,425 characters, and the summary text was only 329 charactes. The generated summary was actually shorter than the human-generated article headline summaries (536 characters).

Hugging Face did an excellent job of text summariztion. Note that the generic model t5-base was used. Other models are available, such as BART and GPT.

Part 4: Word2Vec and Semantic Analysis

Goal: Word2Vec will learn representations based on the context on how the words are used. Word2Vec provides the ability to predice a word based on its surrounding words. The prediction works because words of similar meaning are often surrounded by the same words.

Word2Vec offers:

- Word embeddings to represent each word uniquely as a high-dimensional vector
- Vector spaces, where words with similar meanings are located close to each other
- Semantic relationships that can be applied to words that are closely related

Build the Word2Vec model

In [27]:
```
# Capture the article titles
titles = []

# Populate the list of article titles
for title in tqdm(articles_df['title']):
    titles.append(title)
```

```
100%|██████████| 215365/215365 [00:00<00:00, 85877
2.08it/s]
```

In [28]:
```
# Tokenize titles into words
tokenized_titles = []
for title in titles:
    try:
        tokens = word_tokenize(title.lower())
    except:
        tokens = ' '
    tokenized_titles.append(tokens)
```

In [29]:
```
# Create the Word2Vec model
model = Word2Vec(window=10,          # The max distance between the current word and predicted word
                 sg=1,               # The Skip-gram model used
                 hs=0,               # "Hierarchical Softmax". When hs=0, negative sampling is used instead of
                 negative=10,        # Number of negative samples for each target word during training.
                 alpha=0.03,         # Starting value used to control the learning rate
                 min_alpha=0.0007,   # Final value used to control the learning rate
                 seed=4)             # Defines the random seed for reproducibility.
```

In [30]:
```
# Build the Word2Vec model vocabulary
model.build_vocab(tokenized_titles,      # Contains a list of sentences or tokenized text from your corpus.
                  progress_per=200)            # How often progress information should be displayed while buila
```

In [31]:
```
# Train the model
model.train(tokenized_titles, total_examples=model.corpus_count, epochs=10, report_delay=1)
```

Out[31]:
```
(8583470, 15324760)
```

Investigate the Word2Vec model

In [32]:
```
# Print the model parameters
print(model)
```

```
Word2Vec<vocab=17116, vector_size=100, alpha=0.03>
```

In [33]:
```
# Display the information about the word vectors
X = model.wv[model.wv.key_to_index]
print(f"Shape of X: {len(X)} words x {len(X[0])} dimensions")
```

```
Shape of X: 17116 words x 100 dimensions
```

Result: The vocabulary of the article titles has 17,116 words in 100 dimensions.

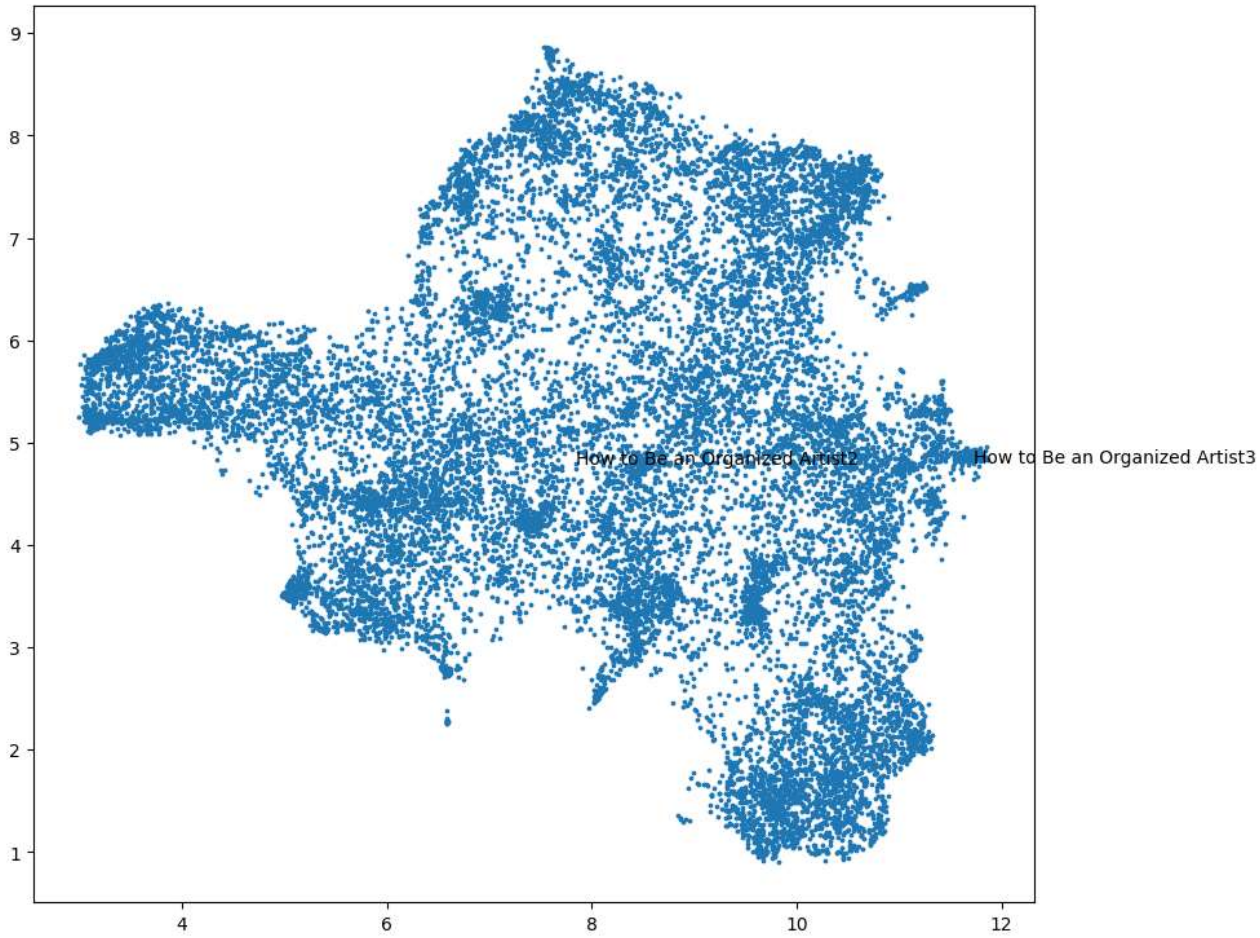
Visualize the Word2Vec Embeddings

In [34]:
```
# Calculate cluster_embedding
cluster_embedding = umap.UMAP(n_neighbors=30, min_dist=0.0, n_components=2, random_state=42).fit_transform(X)

# Create the scatter plot of the cluster embedding
plt.figure(figsize=(10, 9))
plt.scatter(cluster_embedding[:, 0], cluster_embedding[:, 1], s=3, cmap='Spectral')

# Add a label to the plot
plt.annotate(articles_df['title'][4], (cluster_embedding[4, 0], cluster_embedding[4, 1]))
plt.annotate(articles_df['title'][5], (cluster_embedding[5, 0], cluster_embedding[5, 1]))
```

Out[34]:
```
Text(11.722657, 4.798214, 'How to Be an Organized Artist3')
```



In [35]:
```
# Display the fourth and fifth article 'title'
print(articles_df['title'][4])
print(articles_df['title'][5])
```

```
How to Be an Organized Artist2
How to Be an Organized Artist3
```

Plot Interpretation: The plot represents a t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization of the word vectors reduced to two-dimensional space, showing word clusters based on the word vectors created by Word2Vec.

Note: Each word is represented by a point on the graph. Words close together have similar meanings in the 100-dimension space.

The fourth article title “How to Be an Organized Artist2” and the fifth article title “How to Be an Organized Artist3” are close to each other, separated by the difference between ‘Artist2’ and ‘Artist3’.

Part 5: Non-Negative Matrix Factorization (NMF) for Topic Modeling

NMF is an NLP technique for topic modeling and feature extraction. NMF offers:

- Unsupervised learning to extract topics from large text documents
- Feature extraction and text classification by identifying features that can be removed

- Text summarization by identifying the underlying sturcture of a document by finding key terms

Create the TF-IDF matrix

In [36]:
```
# Replace 'nan' values in the DataFrame with an empty space
df = pd.DataFrame(articles_df['title'].fillna(''))
df.head(5)

# Create the TF-IDF matrix vectorizer
vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['title'])
```

Use NMF to create the topics and build the model

In [37]:
```
# Create the number of topics
num_topics = 15

# Create the model
nmf_model = NMF(n_components=num_topics, init='random', random_state=42)

# Build the matrix
nfm_matrix = nmf_model.fit_transform(tfidf_matrix)
```

In [38]:
```
# Display the topics and the top words in each topic
feature_names = vectorizer.get_feature_names_out()

# This loop iterates over each topic extracted by NMF.
for topic_idx, topic in enumerate(nmf_model.components_):
    top_words_indices = topic.argsort()[-10:][::-1]
    top_words = [feature_names[i] for i in top_words_indices]
    # The join method is used to concatenate the words into a single string, separated by commas.
    print(f"Topic #{topic_idx + 1}: {', '.join(top_words)}")
```

```
Topic #1: draw, cartoon, 3d, anime, manga, using, face, dog, cat, little
Topic #2: use, google, android, windows, leftover, twitter, microsoft, facebook, iphone, oil
Topic #3: write, letter, good, essay, book, paper, story, plan, resume, business
Topic #4: rid, acne, skin, dog, treat, cold, dry, home, bad, using
Topic #5: create, using, account, windows, google, home, microsoft, good, website, facebook
Topic #6: stop, dog, cat, help, child, using, eating, feeling, iphone, getting
Topic #7: cook, prepare, chicken, rice, pork, beef, frozen, red, baby, food
Topic #8: deal, treat, school, friend, people, child, help, avoid, parents, dog
Topic #9: like, look, act, good, dress, girl, school, girls, tell, know
Topic #10: make, chocolate, money, cake, chicken, paper, homemade, runescape, cream, vegan
Topic #11: choose, right, dog, good, cat, wedding, school, healthy, buy, home
Topic #12: play, game, games, guitar, minecraft, card, video, super, google, online
Topic #13: clean, leather, white, car, kitchen, dog, room, wood, water, cat
Topic #14: care, hair, dog, skin, child, baby, cat, pet, new, grow
Topic #15: remove, windows, change, hair, install, stains, paint, using, computer, car
```

Result Interpretation: The Topics produced by NMF produce a list of words that are associated with that topic. For example, in Topic #1, ‘draw’, ‘cartoon’, ‘3d’, ‘anime’, ‘manga’ are strongly related. Other words, like ‘face’, ‘dog’, ‘cat’ seem to have little relation, but may be articles about drawing those items.

Part 6: A Recommendation System Using Latent Dirichlet Allocation (LDA)

Description: Latent Dirichlet Allocation (LDA) is used with cosine similarity to recommend articles without labels using unsupervised learning. LDA provides:

- Topic discovery by analyzing the underlying structure of text documents without having topic labels
- Bag-Of-Words (BOW) text representation, ignoring word order to focus on word frequency
- A probabilistic approach based on a probability distribution of the words in the topics

Create a DataFrame to store the article titles

In [39]:
```
# Create a pandas DataFrame to hold the article titles
df = pd.DataFrame(articles_df['title'].fillna(''))
df.head(5)
```

Out[39]:

	title
0	How to Be an Organized Artist1
1	How to Create a Neopoprealist Art Work
2	How to Be a Visual Effects Artist1
3	How to Become an Art Investor
4	How to Be an Organized Artist2

Create the vectorizer and fit the model

In [40]:
```
# Create the vectorizer and TFIDF matrix
vectorizer = TfidfVectorizer(max_features=1000, stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['title'])
```

Create the topics

In [41]:
```
# Use LDA for topic modeling
num_topics = 15
lda = LatentDirichletAllocation(n_components=num_topics, random_state=42)
lda.fit(tfidf_matrix)
```

Out[41]:
```
LatentDirichletAllocation(n_components=15, random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [42]:
```
# Create the topic distribution
topic_distributions = lda.transform(tfidf_matrix)
```

In [43]:
```
# Choose the first article as a sample
article_number = 0
article_topic = topic_distributions[article_number]
```

Use cosine similarity to find articles with similar topics

In [44]:
```
# Use the cosine similarity between articles based on topic distributions
cosine_similarities = cosine_similarity(topic_distributions, [article_topic])
```

In [45]:
```
# Use cosine similarity to recommend similar articles
num_recommendations = 5
similar_articles = list(enumerate(cosine_similarities))
similar_articles = sorted(similar_articles, key=lambda x: x[1], reverse=True)
```

In [46]:
```
# Print the similar articles
print(f"Recommend articles similar to: {df['title'][0]}")
print("Recommended articles:")
for idx, similarity in similar_articles[1:num_recommendations+1]:
    print(f"Similarity: {similarity[0]:.4f} | Article: {df['title'][idx]}")
```

```
Recommend articles similar to: How to Be an Organized Artist1
Recommended articles:
Similarity: 1.0000 | Article: How to Be a Visual Effects Artist1
Similarity: 1.0000 | Article: How to Be an Organized Artist2
Similarity: 1.0000 | Article: How to Be an Organized Artist3
Similarity: 1.0000 | Article: How to Be a Visual Effects Artist2
Similarity: 1.0000 | Article: How to Be a Visual Effects Artist3
```

Result Interpretation: The desired article title begins with 'How to be a …' and ends with the word 'Artist'. Each of the recommended articles have a Similarity score of 1.0 (highly similar), also start with the phrase "How to Be a/an" and end with the word "Artist"; the number after the word "Artist" indicates the fact that the article is a duplicate, which was also recommended.

Part 7: Text Summarization Using Seq2Seq

A Sequence-to-Sequence (Seq2Seq) can accept a large article text and generate an article summary as output. It takes 'label'/'text' pairs as input that is used to reconstruct the original text. The features of Seq2Seq include:

- An Encoder/Decoder neural network architecture where the encoder takes the input text to produce a context vector, and the the dcoder takes the context vector and reconstructs the original input sequence.
- The input can be variable-length text that is padded to make the text lengths equal
- The Attention Mechanism is used to find a specific part of the input text when processing long text sequences

Build a DataFrame from the 'title' and 'text' of the articles

In [47]:
```
# Create a pandas DataFrame with the article text
df = articles_df[['title', 'text']].iloc[:10].copy()
df.head(10)
```

Out[47]:

	title	text
0	How to Be an Organized Artist1	If you're a photographer, keep all the necess...
1	How to Create a Neopoprealist Art Work	See the image for how this drawing develops s...
2	How to Be a Visual Effects Artist1	It is possible to become a VFX artist without...
3	How to Become an Art Investor	The best art investors do their research on t...
4	How to Be an Organized Artist2	As you start planning for a project or work, ...
5	How to Be an Organized Artist3	When you finish a project, whether it sells o...
6	How to Be a Visual Effects Artist2	This should be a short video showcasing the b...
7	How to Be a Visual Effects Artist3	Networking is a great way to find new opportu...
8	How to Be Good at Improvisation	Some entire movies are improvised, some plays...
9	How to Always Catch Pop Culture References1	Use your friends' conversations to figure out...

Build the tokenizer to process the text

In [48]:
```
# Create tokenizer for 'text'
max_len_text = 1500
tokenizer_text = Tokenizer()
tokenizer_text.fit_on_texts(df['text'])
seq_text = tokenizer_text.texts_to_sequences(df['text'])
padded_text = pad_sequences(seq_text, maxlen=max_len_text, padding='post')
```

In [49]:
```
# Create tokenizer for 'title'
max_len_title = 10
tokenizer_title = Tokenizer()
tokenizer_title.fit_on_texts(df['title'])
seq_title = tokenizer_title.texts_to_sequences(df['title'])
padded_title = pad_sequences(seq_title, maxlen=max_len_title, padding='post')
```

Build the Seq2Seq vcabulary, encoder and decoder

In [50]:
```
# Create the seq2seq model
vocab_size_text = len(tokenizer_text.word_index) + 1
vocab_size_title = len(tokenizer_title.word_index) + 1
latent_dim = 300
```

In [51]:
```
# Create the Seq2Seq Encoder
encoder_inputs = tf.keras.layers.Input(shape=(max_len_text,))
enc_emb = tf.keras.layers.Embedding(vocab_size_text, latent_dim, trainable=True)(encoder_inputs)
encoder_lstm = tf.keras.layers.LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]
```

In [52]:
```
# Create the Seq2Seq Decoder
decoder_inputs = tf.keras.layers.Input(shape=(None,))
dec_emb_layer = tf.keras.layers.Embedding(vocab_size_title, latent_dim, trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = tf.keras.layers.LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = tf.keras.layers.Dense(vocab_size_title, activation='softmax')
output = decoder_dense(decoder_outputs)
```

Build, compile, and train the Seq2Seq model

In [53]:
```
# Create the model
model = tf.keras.models.Model([encoder_inputs, decoder_inputs], output)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
```

In [54]:
```
# Fit (train) the model
model.fit([padded_text, padded_title[:, :-1]], padded_title[:, 1:], epochs=50, batch_size=64)
```

```
Epoch 1/50
1/1 [==============================] - 17s 17s/step - loss: 3.2539
Epoch 2/50
1/1 [==============================] - 9s 9s/step - loss: 3.1782
Epoch 3/50
1/1 [==============================] - 9s 9s/step - loss: 3.0681
Epoch 4/50
1/1 [==============================] - 9s 9s/step - loss: 2.8206
Epoch 5/50
1/1 [==============================] - 9s 9s/step - loss: 2.2289
Epoch 6/50
1/1 [==============================] - 9s 9s/step - loss: 2.4628
Epoch 7/50
1/1 [==============================] - 9s 9s/step - loss: 2.1811
Epoch 8/50
1/1 [==============================] - 9s 9s/step - loss: 1.9917
Epoch 9/50
1/1 [==============================] - 9s 9s/step - loss: 1.9303
Epoch 10/50
1/1 [==============================] - 9s 9s/step - loss: 1.8613
Epoch 11/50
1/1 [==============================] - 9s 9s/step - loss: 1.7930
Epoch 12/50
1/1 [==============================] - 9s 9s/step - loss: 1.7606
Epoch 13/50
1/1 [==============================] - 9s 9s/step - loss: 1.7438
Epoch 14/50
1/1 [==============================] - 9s 9s/step - loss: 1.7073
Epoch 15/50
1/1 [==============================] - 9s 9s/step - loss: 1.6451
Epoch 16/50
1/1 [==============================] - 9s 9s/step - loss: 1.5792
Epoch 17/50
1/1 [==============================] - 9s 9s/step - loss: 1.5354
Epoch 18/50
1/1 [==============================] - 9s 9s/step - loss: 1.5110
Epoch 19/50
1/1 [==============================] - 9s 9s/step - loss: 1.4768
Epoch 20/50
1/1 [==============================] - 9s 9s/step - loss: 1.4287
Epoch 21/50
1/1 [==============================] - 8s 8s/step - loss: 1.3894
Epoch 22/50
1/1 [==============================] - 9s 9s/step - loss: 1.3659
Epoch 23/50
1/1 [==============================] - 9s 9s/step - loss: 1.3410
Epoch 24/50
1/1 [==============================] - 9s 9s/step - loss: 1.3021
Epoch 25/50
1/1 [==============================] - 9s 9s/step - loss: 1.2598
Epoch 26/50
1/1 [==============================] - 9s 9s/step - loss: 1.2293
Epoch 27/50
1/1 [==============================] - 9s 9s/step - loss: 1.2040
Epoch 28/50
1/1 [==============================] - 8s 8s/step - loss: 1.1692
Epoch 29/50
1/1 [==============================] - 9s 9s/step - loss: 1.1337
Epoch 30/50
1/1 [==============================] - 9s 9s/step - loss: 1.1089
Epoch 31/50
1/1 [==============================] - 8s 8s/step - loss: 1.0841
Epoch 32/50
1/1 [==============================] - 9s 9s/step - loss: 1.0502
Epoch 33/50
1/1 [==============================] - 9s 9s/step - loss: 1.0188
Epoch 34/50
1/1 [==============================] - 9s 9s/step - loss: 0.9956
Epoch 35/50
1/1 [==============================] - 9s 9s/step - loss: 0.9663
Epoch 36/50
1/1 [==============================] - 9s 9s/step - loss: 0.9352
Epoch 37/50
1/1 [==============================] - 8s 8s/step - loss: 0.9130
Epoch 38/50
1/1 [==============================] - 9s 9s/step - loss: 0.8870
```

```
Epoch 39/50
1/1 [==============================] - 9s 9s/step - loss: 0.8580
Epoch 40/50
1/1 [==============================] - 9s 9s/step - loss: 0.8360
Epoch 41/50
1/1 [==============================] - 9s 9s/step - loss: 0.8102
Epoch 42/50
1/1 [==============================] - 9s 9s/step - loss: 0.7845
Epoch 43/50
1/1 [==============================] - 9s 9s/step - loss: 0.7636
Epoch 44/50
1/1 [==============================] - 9s 9s/step - loss: 0.7378
Epoch 45/50
1/1 [==============================] - 9s 9s/step - loss: 0.7148
Epoch 46/50
1/1 [==============================] - 8s 8s/step - loss: 0.6929
Epoch 47/50
1/1 [==============================] - 9s 9s/step - loss: 0.6682
Epoch 48/50
1/1 [==============================] - 9s 9s/step - loss: 0.6478
Epoch 49/50
1/1 [==============================] - 9s 9s/step - loss: 0.6243
Epoch 50/50
1/1 [==============================] - 9s 9s/step - loss: 0.6031
```

Out[54]:
```
<keras.src.callbacks.History at 0x13a13c6b790>
```

Generate an article summary by predicting the article title base on the article text.

In [55]:
```
# Specify the text of the article
sample_text = articles_df['text'][0]

# Tokenize the input text and pad any empty space with 0's
seq = tokenizer_text.texts_to_sequences([sample_text])
padded_seq = pad_sequences(seq, maxlen=max_len_text, padding='post')
initial_input = np.zeros((1, max_len_title))

# Use the padded sequence to predict the corresponding article title
for i in range(1, max_len_title):
    predictions = model.predict([padded_seq, initial_input])
    sampled_token_index = np.argmax(predictions[0, i-1, :])
    initial_input[0, i] = sampled_token_index

# Remove padding and convert the generated sequence back to text
generated_title_ids = initial_input[0][1:]
generated_title_text = tokenizer_title.sequences_to_texts([generated_title_ids])[0]

print("Predicted Article Title:")
print(generated_title_text)
```

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 0s 375ms/step
1/1 [==============================] - 0s 402ms/step
1/1 [==============================] - 0s 375ms/step
1/1 [==============================] - 0s 372ms/step
1/1 [==============================] - 0s 375ms/step
1/1 [==============================] - 0s 368ms/step
1/1 [==============================] - 0s 363ms/step
1/1 [==============================] - 0s 376ms/step
Predicted Article Title:
to be an organized artist3
```

In [56]:
```
print('Actual article title:')
articles_df['title'][0]
```

```
Actual article title:
```

Out[56]:
```
'How to Be an Organized Artist1'
```

Result Interpretation: The predicted article title mentions ‘organized artist3’, while the actual article title corresponding to the article text is ‘Organized Artist1’. The text for the titles may be too similar for Seq2Seq to differentiate them.

Summary and conclusion

This project employed several NLP techniques to provide textual analsys. These include:

- NLP libraries and pretrained models (spaCy, NLTK, Transformers, Gensim)
- NLP topics (NER, Seq2Seq, Attention, Word2Vec, Tokenization, LDA, NMF, BOW)
- NLP Applications (Recommendation System, Text Summarization)

Transformers with the Hugging Face library provided excellent results for text summarization with minimal training. Hugging Face is quite advanced and has a robust developer ecosystem building products with it. When doing a project, it becomes apparent quickly which technologies are actually being used by developers based on the number of websites, on-line tutorials, and help that is being provided.

Seq2Seq provided a good way to take paired article titles and article text and generate text that provided an article summary.Seq2Seq, using the encoder/decoder deep neural networks has a great future for generative technologies. Seq2Seq did not have many on-line resources (compared to Hugging Face) but it is seeing use in research and will likely develop over time.

Word2Vec provided excellent results for semantic analysis to be able to visualize how 'close' two articles were in meaning with minimal training.

Publication

YouTube Video Link: https://youtu.be/ZczuOc-wCTY (https://youtu.be/ZczuOc-wCTY)

GitHub Repo Link: https://github.com/PaulSandeen-mtsac/Mt.SAC_CISB63_Final_Project (https://github.com/PaulSandeen-mtsac/Mt.SAC_CISB63_Final_Project)

In [ ]: