Title generation

April 8, 2024

import pandas as pd import numpy as np import re from nltk.corpus import stopwords from nltk.tokenize import word_tokenize from gensim.models import Word2Vec import nltk from nltk.stem import WordNetLemmatizer

nltk.download('stopwords') nltk.download('punkt') nltk.download('wordnet')

data = pd.read_csv('../input/poetry-foundation-poems/PoetryFoundationData.csv') data = data.dropna()

[2]:
```
data.head()
```

[2]:
```
    Unnamed: 0                                              Title  \
6            6  \r\r\n                    Invisible Fish\r\r\n…
7            7  \r\r\n                    Don't Bother the Ear…
9            9  \r\r\n                    ["Hour in which I co…
16          16  \r\r\n                    scars\r\r\n         …
17          17  \r\r\n                    what remains two\r\r…

                                                  Poem          Poet  \
6   \r\r\nInvisible fish swim this ghost ocean now…     Joy Harjo
7   \r\r\nDon't bother the earth spirit who lives …     Joy Harjo
9   \r\r\nHour in which I consider hydrangea, a sa…  Simone White
16  \r\r\nmy father's body is a map\r\r\na record …   Truong Tran
17  \r\r\nit has long been forgotten this practice…   Truong Tran

                                                 Tags
6   Living,Time & Brevity,Relationships,Family & A…
7   Religion,The Spiritual,Mythology & Folklore,Fa…
9   Living,Parenthood,The Body,The Mind,Nature,Tre…
16                        The Body,Family & Ancestors
17                        Infancy,Parenthood,The Body
```

[3]:
```
data = data.head(10000)
```

0.1 Preprocessing steps

Lowercasing: Converts all text to lowercase, ensuring uniformity.

Punctuation Removal: Eliminates non-alphanumeric characters and whitespace, such as punctuation marks, to focus on the words.

Special Character Removal: Gets rid of specific special characters, like newline characters, for a cleaner text.

Tokenization: Splits the text into individual words or tokens, facilitating further analysis.

Stopword Removal: Filters out common words (stopwords) that usually do not contribute much to the meaning of the text.

Lemmatization: Reduces words to their base or root form, aiding in standardizing different forms of words.

[4]:
```
import nltk
#nltk.download('wordnet')
```

[5]:
```
!unzip /usr/share/nltk_data/corpora/wordnet.zip -d /usr/share/nltk_data/corpora/
```

```
Archive:  /usr/share/nltk_data/corpora/wordnet.zip
   creating: /usr/share/nltk_data/corpora/wordnet/
  inflating: /usr/share/nltk_data/corpora/wordnet/lexnames
  inflating: /usr/share/nltk_data/corpora/wordnet/data.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/adv.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/index.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/cntlist.rev
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adj
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adj
  inflating: /usr/share/nltk_data/corpora/wordnet/LICENSE
  inflating: /usr/share/nltk_data/corpora/wordnet/citation.bib
  inflating: /usr/share/nltk_data/corpora/wordnet/noun.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/verb.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/README
  inflating: /usr/share/nltk_data/corpora/wordnet/index.sense
  inflating: /usr/share/nltk_data/corpora/wordnet/data.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/index.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/adj.exc
```

[6]:
```
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet


def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Punctuation
```

```
    text = re.sub(r'[^\w\s]', '', text)

    # Special characters
    text = re.sub(r'[\r\n]', '', text)

    # Token
    tokens = word_tokenize(text)

    # Stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in␣
 ↪stop_words]

    # Lemmatize
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

    return ' '.join(lemmatized_tokens)

text_to_preprocess = "This is an example sentence! It has some special␣
 ↪characters *&^%$# and stopword."
processed_text = preprocess_text(text_to_preprocess)
print(processed_text)
```

```
example sentence special character stopword
```

[8]:
```
data['Title'] = data['Title'].apply(preprocess_text)
data['Poem'] = data['Poem'].apply(preprocess_text)
```

0.2 Embedding Techniques

1. TF-IDF is used for document representation and importance weighting.
2. Word2Vec is employed for generating word embeddings and capturing semantic relationships between words.
3. CBOW predicts a target word based on its context, useful for tasks that involve understanding the meaning of words in a given context.

TF-IDF

[9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

titles_corpus = data['Title']
poems_corpus = data['Poem']

tfidf_vectorizer_titles = TfidfVectorizer()
tfidf_matrix_titles = tfidf_vectorizer_titles.fit_transform(titles_corpus)

tfidf_vectorizer_poems = TfidfVectorizer()
```

```
tfidf_matrix_poems = tfidf_vectorizer_poems.fit_transform(poems_corpus)
```

[10]:
```
tfidf_vectors_titles = tfidf_matrix_titles.toarray()
tfidf_vectors_poems = tfidf_matrix_poems.toarray()
```

[11]:
```
tfidf_vectors_titles
```

[11]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

[12]:
```
tfidf_vectors_poems
```

[12]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

Word2Vec

[13]:
```
# Word2Vec
from gensim.models import Word2Vec

titles_corpus = [word_tokenize(title) for title in data['Title']]
poems_corpus = [word_tokenize(poem) for poem in data['Poem']]

word2vec_model_titles = Word2Vec(sentences=titles_corpus, vector_size=100,␣
 ↪window=5, sg=1, min_count=1)
word2vec_model_poems = Word2Vec(sentences=poems_corpus, vector_size=100,␣
 ↪window=5, sg=1, min_count=1)

vector_word2vec_title = word2vec_model_titles.wv['word']
vector_word2vec_poem = word2vec_model_poems.wv['word']

vector_word2vec_title

vector_word2vec_poem
```

[13]:
```
array([-0.61564875,  0.36760208, -0.35847807, -0.05511908,  0.06266135,
       -0.4356819 , -0.26802343,  0.7998006 ,  0.25615156, -0.06543805,
```

```
       -0.09168253,  0.07666601, -0.0595214 ,  0.05679855, -0.02169015,
       -0.8373362 , -0.05697606, -0.2257143 ,  0.13990766, -0.42602637,
        0.24868484,  0.30449212,  0.33479708, -0.69590473,  0.08113886,
        0.00436677, -0.06490733,  0.24983953, -0.17020184,  0.23872787,
        0.9120111 ,  0.1017668 , -0.35379466, -0.2619883 , -0.24904262,
        0.72181445, -0.3605565 , -0.12455251, -0.36837575,  0.14135344,
        0.13301773,  0.11048333, -0.30372357, -0.35876217,  0.20555161,
       -0.33469075, -0.54003876, -0.26559895,  0.59868264, -0.03449059,
        0.3437828 , -0.08785414, -0.14583483,  0.02266433, -0.26996124,
        0.38437626,  0.4752406 , -0.01350064, -0.3693143 ,  0.01363356,
       -0.10463747, -0.12735671,  0.21814272, -0.21503635, -1.280752  ,
        1.0079564 , -0.03647245,  0.41894355, -0.55160826,  0.36642304,
       -0.31180286,  0.90975875,  0.7864814 ,  0.02240902,  0.31931767,
       -0.24785183, -0.05761935, -0.22031166,  0.1561128 ,  0.48164964,
       -0.84711397, -0.3042746 , -0.30614355,  0.71568793,  0.23676948,
        0.18341582,  0.3579364 ,  0.13195878,  0.6171721 ,  0.302573  ,
        0.67404234, -0.42081454,  0.41459596,  0.05431885,  0.3897594 ,
        0.89945024, -0.10775471, -0.12452887, -0.5369774 , -0.30139953],
      dtype=float32)
```

[14]:
```
vector_word2vec_title
```

[14]:
```
array([-0.00289828,  0.00110666,  0.00292816, -0.00949872,  0.00503818,
        0.0011494 ,  0.00049574, -0.00119059, -0.0092072 ,  0.00674964,
       -0.00934503, -0.01061274,  0.00420963, -0.00843926, -0.0032115 ,
       -0.00325861, -0.00197454, -0.00901859,  0.0034546 , -0.00174992,
        0.00506706, -0.00065615,  0.00793798, -0.0104944 ,  0.00496034,
       -0.00583619,  0.00030671, -0.00885495, -0.00416814, -0.00609893,
       -0.00199617,  0.00990127,  0.00308354,  0.00130481, -0.00628283,
        0.00779061, -0.00781189, -0.00356099, -0.00745947, -0.00970036,
        0.00292479, -0.00452079,  0.00197807,  0.00467002, -0.00505079,
       -0.00187027, -0.00375819, -0.01027027, -0.009268  ,  0.00226196,
        0.00503264, -0.00139226, -0.00575167,  0.0037601 ,  0.00096041,
       -0.00853467,  0.00783382,  0.00836367, -0.00843945, -0.00081858,
        0.00661208, -0.00196113, -0.00076064, -0.00459656,  0.00498381,
        0.00842904,  0.00219084, -0.00439652, -0.00322783, -0.00250494,
       -0.01117726,  0.00882014,  0.00043858, -0.00104352,  0.00791155,
       -0.00817426,  0.00630665, -0.00818653, -0.00684191,  0.00549073,
       -0.00463506, -0.00334209, -0.00745878, -0.00167935, -0.00310585,
       -0.00396145,  0.00940909,  0.01146358,  0.00168201,  0.00123423,
       -0.00420393,  0.00864689, -0.00483842,  0.0065558 ,  0.01178976,
        0.00740549, -0.00012565,  0.00445795,  0.00754824, -0.00397223],
      dtype=float32)
```

[15]:
```
vector_word2vec_poem
```

[15]:
```
array([-0.61564875,  0.36760208, -0.35847807, -0.05511908,  0.06266135,
       -0.4356819 , -0.26802343,  0.7998006 ,  0.25615156, -0.06543805,
       -0.09168253,  0.07666601, -0.0595214 ,  0.05679855, -0.02169015,
       -0.8373362 , -0.05697606, -0.2257143 ,  0.13990766, -0.42602637,
        0.24868484,  0.30449212,  0.33479708, -0.69590473,  0.08113886,
        0.00436677, -0.06490733,  0.24983953, -0.17020184,  0.23872787,
        0.9120111 ,  0.1017668 , -0.35379466, -0.2619883 , -0.24904262,
        0.72181445, -0.3605565 , -0.12455251, -0.36837575,  0.14135344,
        0.13301773,  0.11048333, -0.30372357, -0.35876217,  0.20555161,
       -0.33469075, -0.54003876, -0.26559895,  0.59868264, -0.03449059,
        0.3437828 , -0.08785414, -0.14583483,  0.02266433, -0.26996124,
        0.38437626,  0.4752406 , -0.01350064, -0.3693143 ,  0.01363356,
       -0.10463747, -0.12735671,  0.21814272, -0.21503635, -1.280752  ,
        1.0079564 , -0.03647245,  0.41894355, -0.55160826,  0.36642304,
       -0.31180286,  0.90975875,  0.7864814 ,  0.02240902,  0.31931767,
       -0.24785183, -0.05761935, -0.22031166,  0.1561128 ,  0.48164964,
       -0.84711397, -0.3042746 , -0.30614355,  0.71568793,  0.23676948,
        0.18341582,  0.3579364 ,  0.13195878,  0.6171721 ,  0.302573  ,
        0.67404234, -0.42081454,  0.41459596,  0.05431885,  0.3897594 ,
        0.89945024, -0.10775471, -0.12452887, -0.5369774 , -0.30139953],
      dtype=float32)
```

CBOW

[16]:
```
cbow_model_titles = Word2Vec(sentences=titles_corpus, vector_size=100,␣
 ↪window=5, sg=0, min_count=1)

cbow_model_poems = Word2Vec(sentences=poems_corpus, vector_size=100, window=5,␣
 ↪sg=0, min_count=1)

vector_cbow_title = cbow_model_titles.wv['word']
vector_cbow_poem = cbow_model_poems.wv['word']
```

[17]:
```
vector_cbow_title
```

[17]:
```
array([-3.2260781e-03,  6.4040173e-04,  2.5771838e-03, -9.1775665e-03,
        4.8798267e-03,  2.3725959e-03,  1.4686928e-04, -2.6804151e-03,
       -8.0423811e-03,  7.4372762e-03, -9.0459473e-03, -8.8737151e-03,
        4.0909424e-03, -8.3050225e-03, -3.2756331e-03, -2.4975738e-03,
       -1.7099706e-03, -7.4503943e-03,  4.1964673e-03,  8.1222861e-05,
        4.0565673e-03, -1.1789326e-03,  7.4997158e-03, -9.5121777e-03,
        5.5554770e-03, -5.0952896e-03,  1.5099003e-03, -7.8000785e-03,
       -2.8056395e-03, -6.7042192e-03, -2.8085026e-03,  9.8439623e-03,
        2.7494077e-03,  1.6241419e-03, -6.1696600e-03,  6.0504982e-03,
       -8.1784856e-03, -2.6467207e-03, -6.8712123e-03, -7.2804946e-03,
        2.3560089e-03, -4.4782120e-03,  2.0219772e-03,  3.9527803e-03,
       -6.1337915e-03, -1.7406832e-03, -3.2879103e-03, -9.9604158e-03,
```

```
       -9.6855098e-03,  1.5640835e-03,  4.4717556e-03, -2.5857138e-04,
       -5.4426943e-03,  3.5450535e-03,  1.2452372e-03, -9.3290135e-03,
        7.5597726e-03,  7.7853827e-03, -8.6018359e-03, -6.2566495e-04,
        6.2846746e-03, -2.7892727e-03, -4.4844599e-04, -4.4585974e-03,
        5.7740887e-03,  7.4153477e-03,  2.3007982e-03, -4.4523664e-03,
       -2.5492101e-03, -3.2355110e-03, -1.0344214e-02,  7.8495294e-03,
        9.6624317e-05, -8.9948630e-04,  7.3518544e-03, -8.9163706e-03,
        6.1201798e-03, -7.6887896e-03, -5.9161172e-03,  5.3185239e-03,
       -4.0065902e-03, -3.7624547e-03, -6.8612294e-03, -3.2014414e-03,
       -2.5725081e-03, -4.0900847e-03,  9.7859092e-03,  1.0166497e-02,
        3.5907346e-05, -2.1235453e-04, -5.2671051e-03,  8.1508830e-03,
       -4.9037100e-03,  6.0022692e-03,  1.0120041e-02,  6.4232438e-03,
       -7.4887520e-04,  5.1576351e-03,  6.7719100e-03, -3.6560656e-03],
      dtype=float32)
```

[18]:
```
vector_cbow_poem
```

[18]:
```
array([-0.9794445 ,  1.3536996 , -0.14938392, -0.534428  ,  0.84249955,
       -2.2010198 ,  0.15351824,  3.4402556 , -0.8292523 , -1.4037759 ,
       -0.69861156, -1.4629518 ,  0.15741575,  1.1606286 ,  0.01758538,
       -1.8453075 ,  0.17237951, -1.4490883 ,  0.24232948, -2.230654  ,
        1.2622486 ,  0.6226575 ,  1.249174  ,  0.31618991,  0.6206465 ,
        0.278654  , -1.0792649 , -0.37523973, -1.7318637 ,  0.24850094,
        2.6933055 ,  0.735218  , -0.28469485, -1.3954331 , -0.29144117,
        1.616018  , -0.27086112, -1.2002989 , -0.49947304, -2.1122627 ,
       -0.07801346, -0.9815619 , -0.5807417 , -0.5284042 ,  1.2762492 ,
       -1.0242841 , -2.2197843 ,  0.13200691,  0.5349314 ,  0.6007352 ,
        0.6579376 , -1.6454868 , -1.0770009 , -0.24198686, -0.9358163 ,
        0.11665091,  1.2407091 , -0.31810552, -1.704197  ,  0.03843501,
        0.5997982 , -0.1038193 ,  0.47340804, -0.28007522, -2.6818333 ,
        1.1520594 ,  0.78172964,  0.65741664, -2.6657627 ,  2.070497  ,
       -0.91031784,  0.963092  ,  1.7611814 ,  0.36986443,  1.4346917 ,
        0.1807785 , -0.24511957, -0.34584618, -1.1728116 ,  1.3355895 ,
       -1.0284793 , -0.32566807, -1.3161582 ,  1.7930986 , -0.3337716 ,
       -0.5982069 ,  0.62947583,  1.3801053 ,  1.2389158 ,  0.9991177 ,
        1.3457344 , -0.03453258,  1.7379031 , -0.07166918,  1.9420028 ,
        2.1211433 ,  1.2379456 , -0.9602313 ,  0.55663747, -0.69457716],
      dtype=float32)
```

Visual Representation using word cloud

[19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 6 to 10801
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
```

```
 0   Unnamed: 0  10000 non-null  int64
 1   Title       10000 non-null  object
 2   Poem        10000 non-null  object
 3   Poet        10000 non-null  object
 4   Tags        10000 non-null  object
dtypes: int64(1), object(4)
memory usage: 468.8+ KB
```

[20]:
```
subset_data = data


subset_data['Title'] = subset_data['Title'].apply(preprocess_text)
subset_data['Poem'] = subset_data['Poem'].apply(preprocess_text)
```

[21]:
```
import torch
import numpy as np
from transformers import BertTokenizer, BertModel
from tqdm import tqdm
import pandas as pd

class BertSequenceVectorizer:
    def __init__(self):
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'

        self.model_name = 'bert-base-uncased'

        self.tokenizer = BertTokenizer.from_pretrained(self.model_name)

        self.bert_model = BertModel.from_pretrained(self.model_name)
        self.bert_model = self.bert_model.to(self.device)

        self.max_len = 128

    def vectorize(self, sentence: str) -> np.array:
        inp = self.tokenizer.encode(sentence, add_special_tokens=True)
        len_inp = len(inp)

        if len_inp >= self.max_len:
            inputs = inp[:self.max_len]
            masks = [1] * self.max_len
        else:
            inputs = inp + [0] * (self.max_len - len_inp)
            masks = [1] * len_inp + [0] * (self.max_len - len_inp)

        inputs_tensor = torch.tensor([inputs], dtype=torch.long).to(self.device)
        masks_tensor = torch.tensor([masks], dtype=torch.long).to(self.device)
```

```
        with torch.no_grad():
            bert_out = self.bert_model(inputs_tensor, masks_tensor)
            seq_out, pooled_out = bert_out.last_hidden_state, bert_out.
 ↪pooler_output

        if torch.cuda.is_available():
            return seq_out[0][0].cpu().detach().numpy()
        else:
            return seq_out[0][0].detach().numpy()

BSV = BertSequenceVectorizer()

data['Poem_bert'] = tqdm(data['Poem'].apply(lambda x: BSV.vectorize(x)))

display(data[0:2])
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Token indices sequence length is longer than the specified maximum sequence
length for this model (620 > 512). Running this sequence through the model will
result in indexing errors
100%|     | 10000/10000 [00:00<00:00, 1839687.71it/s]
```

```
   Unnamed: 0                     Title  \
6           6            invisible fish
7           7  dont bother earth spirit

                                                Poem       Poet  \
6  invisible fish swim ghost ocean described wave…  Joy Harjo
7  dont bother earth spirit life working story ol…  Joy Harjo

                                                Tags  \
6  Living,Time & Brevity,Relationships,Family & A…
7  Religion,The Spiritual,Mythology & Folklore,Fa…

                                           Poem_bert
6  [-0.070620015, -0.012272737, 0.1761645, -0.166…
7  [-0.108606495, -0.01237058, 0.26703766, -0.105…
```

[22]:
```
import torch
import numpy as np
from transformers import BertTokenizer, BertModel
```

```
from tqdm import tqdm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

data['Poem_bert'] = data['Poem'].apply(lambda x: BSV.vectorize(x))

X_train, X_test, y_train, y_test = train_test_split(data['Poem_bert'],␣
 ↪data['Title'], test_size=0.2, random_state=42)
```

[23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)


logistic_model = LogisticRegression(max_iter=1000)  # Increase max_iter
logistic_model.fit(list(X_train), y_train_encoded)

new_poem = "Brian If you knew who I was now When I knew who you were then Would␣
 ↪you forgive me before We ever became them? "
vectorized_poem = BSV.vectorize(new_poem)

predicted_title_encoded = logistic_model.predict([vectorized_poem] )[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: poem
```

[30]:
```
new_poem = "think I've seen this film before And I didn't like the ending␣
 ↪You're not my homeland anymore So what am I defending now? You were my town␣
 ↪Now I'm in exile, seein' you out I think I've seen this film before"
vectorized_poem = BSV.vectorize(new_poem)

predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: selfportrait
```

[29]:
```
new_poem = "Matches burn after the other Leaves turn and stick to each other␣
 ↪Wages earned and lessons learned But I, I'm right where you left me. Wind in␣
 ↪my hair"
vectorized_poem = BSV.vectorize(new_poem)
```

```
predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: garden love
```

[31]:
```
new_poem ="It is thy desire in us that desireth. It is thy urge in us that␣
 ↪would turn our nights, which are thine, into days which are thine also. "
vectorized_poem = BSV.vectorize(new_poem)

predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: prayer
```

[32]:
```
new_poem = "Ah Sun-flower! weary of time, Who countest the steps of the Sun:␣
 ↪Seeking after that sweet golden clime Where the travellers journey is done."
vectorized_poem = BSV.vectorize(new_poem)

predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: wedding hymn
```

[36]:
```
new_poem = "The sea is calm tonight. The tide is full, the moon lies fair Upon␣
 ↪the straits; on the French coast the light Gleams and is gone; the cliffs of␣
 ↪England stand, Glimmering and vast, out in the tranquil bay."
vectorized_poem = BSV.vectorize(new_poem)

predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
predicted_title = label_encoder.inverse_transform([predicted_title_encoded])[0]
print("Predicted Title:", predicted_title)
```

```
Predicted Title: vulnerary
```

[38]:
```
import os


output_dir = "./saved_bert_model"


if not os.path.exists(output_dir):
    os.makedirs(output_dir)

print("Directory created successfully:", output_dir)
```

```
Directory created successfully: ./saved_bert_model
```

[39]:
```python
from transformers import BertTokenizer, BertModel


output_dir = "./saved_bert_model"


if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Save the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenizer.save_pretrained(output_dir)


model = BertModel.from_pretrained('bert-base-uncased')
model.save_pretrained(output_dir)

print("Model and tokenizer saved successfully to:", output_dir)
```

```
Model and tokenizer saved successfully to: ./saved_bert_model
```

[41]:
```python
from IPython.display import FileLink

FileLink(r'saved_bert_model.zip')
```

[41]:
```
/kaggle/working/saved_bert_model.zip
```

poem-generation-using-bert

April 8, 2024

[1]:
```
import pandas as pd
import numpy as np
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
import nltk
from nltk.stem import WordNetLemmatizer


nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')


data = pd.read_csv('../input/poetry-foundation-poems/PoetryFoundationData.csv')
data = data.dropna()
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /usr/share/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[2]:
```
data.head()
```

[2]:

	Unnamed: 0		Title	\
6	6	\r\r\n	Invisible Fish\r\r\n…	
7	7	\r\r\n	Don’t Bother the Ear…	
9	9	\r\r\n	["Hour in which I co…	
16	16	\r\r\n	scars\r\r\n	…
17	17	\r\r\n	what remains two\r\r…	

	Poem	Poet	\
6	\r\r\nInvisible fish swim this ghost ocean now…	Joy Harjo	
7	\r\r\nDon’t bother the earth spirit who lives …	Joy Harjo	
9	\r\r\nHour in which I consider hydrangea, a sa…	Simone White	

```
16  \r\r\nmy father's body is a map\r\r\na record …   Truong Tran
17  \r\r\nit has long been forgotten this practice…   Truong Tran

                                                    Tags
6   Living,Time & Brevity,Relationships,Family & A…
7   Religion,The Spiritual,Mythology & Folklore,Fa…
9   Living,Parenthood,The Body,The Mind,Nature,Tre…
16                        The Body,Family & Ancestors
17                        Infancy,Parenthood,The Body
```

[3]:
```
data = data.head(10000)
```

0.1 Preprocessing steps

Lowercasing: Converts all text to lowercase, ensuring uniformity.

Punctuation Removal: Eliminates non-alphanumeric characters and whitespace, such as punctuation marks, to focus on the words.

Special Character Removal: Gets rid of specific special characters, like newline characters, for a cleaner text.

Tokenization: Splits the text into individual words or tokens, facilitating further analysis.

Stopword Removal: Filters out common words (stopwords) that usually do not contribute much to the meaning of the text.

Lemmatization: Reduces words to their base or root form, aiding in standardizing different forms of words.

[4]:
```
import nltk
#nltk.download('wordnet')
```

[5]:
```
!unzip /usr/share/nltk_data/corpora/wordnet.zip -d /usr/share/nltk_data/corpora/
```

```
Archive:  /usr/share/nltk_data/corpora/wordnet.zip
   creating: /usr/share/nltk_data/corpora/wordnet/
  inflating: /usr/share/nltk_data/corpora/wordnet/lexnames
  inflating: /usr/share/nltk_data/corpora/wordnet/data.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/adv.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/index.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/cntlist.rev
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adj
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adj
  inflating: /usr/share/nltk_data/corpora/wordnet/LICENSE
  inflating: /usr/share/nltk_data/corpora/wordnet/citation.bib
  inflating: /usr/share/nltk_data/corpora/wordnet/noun.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/verb.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/README
```

```
  inflating: /usr/share/nltk_data/corpora/wordnet/index.sense
  inflating: /usr/share/nltk_data/corpora/wordnet/data.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/index.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/adj.exc
```

[6]:
```
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet


def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Special characters
    text = re.sub(r'[\r\n]', '', text)

    # Token
    tokens = word_tokenize(text)

    # Stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in␣
 ↪stop_words]

    # Lemmatize
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

    return ' '.join(lemmatized_tokens)

text_to_preprocess = "This is an example sentence! It has some special␣
 ↪characters *&^%$# and stopword."
processed_text = preprocess_text(text_to_preprocess)
print(processed_text)
```

```
example sentence special character stopword
```

[7]:
```
'''import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

```
from nltk.stem import PorterStemmer

def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Special characters
    text = re.sub(r'[\r\n]', '', text)

    # Tokenization
    tokens = word_tokenize(text)

    # Stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in␣
 ↪stop_words]

    # Stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]

    return ' '.join(stemmed_tokens)

text_to_preprocess = "This is an example sentence! It has some special␣
 ↪characters *&^%$# and stopword."
processed_text = preprocess_text(text_to_preprocess)
print(processed_text)'''
```

[7]:
```
'import re\nfrom nltk.tokenize import word_tokenize\nfrom nltk.corpus import
stopwords\nfrom nltk.stem import PorterStemmer\n\ndef preprocess_text(text):\n
# Lowercase\n    text = text.lower()\n\n    # Punctuation\n    text =
re.sub(r\'[^\\w\\s]\', \'\', text)\n\n    # Special characters\n    text =
re.sub(r\'[\r\n]\', \'\', text)\n\n    # Tokenization\n    tokens =
word_tokenize(text)\n\n    # Stopwords\n    stop_words =
set(stopwords.words(\'english\'))\n    filtered_tokens = [word for word in
tokens if word.lower() not in stop_words]\n\n    # Stemming\n    stemmer =
PorterStemmer()\n    stemmed_tokens = [stemmer.stem(word) for word in
filtered_tokens]\n\n    return \' \'.join(stemmed_tokens)\n\ntext_to_preprocess
= "This is an example sentence! It has some special characters *&^%$# and
stopword."\nprocessed_text =
preprocess_text(text_to_preprocess)\nprint(processed_text)'
```

[8]:
```
data['Title'] = data['Title'].apply(preprocess_text)
data['Poem'] = data['Poem'].apply(preprocess_text)
```

0.2 Embedding Techniques

1. TF-IDF is used for document representation and importance weighting.
2. Word2Vec is employed for generating word embeddings and capturing semantic relationships between words.
3. CBOW predicts a target word based on its context, useful for tasks that involve understanding the meaning of words in a given context.

TF-IDF

[9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

titles_corpus = data['Title']
poems_corpus = data['Poem']

tfidf_vectorizer_titles = TfidfVectorizer()
tfidf_matrix_titles = tfidf_vectorizer_titles.fit_transform(titles_corpus)

tfidf_vectorizer_poems = TfidfVectorizer()
tfidf_matrix_poems = tfidf_vectorizer_poems.fit_transform(poems_corpus)
```

[10]:
```
tfidf_vectors_titles = tfidf_matrix_titles.toarray()
tfidf_vectors_poems = tfidf_matrix_poems.toarray()
```

[11]:
```
tfidf_vectors_titles
```

[11]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

[12]:
```
tfidf_vectors_poems
```

[12]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

Word2Vec

CBOW

Visual Representation using word cloud

[13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 6 to 10801
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  10000 non-null  int64 
 1   Title       10000 non-null  object
 2   Poem        10000 non-null  object
 3   Poet        10000 non-null  object
 4   Tags        10000 non-null  object
dtypes: int64(1), object(4)
memory usage: 468.8+ KB
```

[14]:
```
subset_data = data


subset_data['Title'] = subset_data['Title'].apply(preprocess_text)
subset_data['Poem'] = subset_data['Poem'].apply(preprocess_text)
```

[15]:
```
import torch
import numpy as np
from transformers import BertTokenizer, BertModel
from tqdm import tqdm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

class BertSequenceVectorizer:
    def __init__(self):
        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
        self.model_name = 'bert-base-uncased'
        self.tokenizer = BertTokenizer.from_pretrained(self.model_name)
        self.bert_model = BertModel.from_pretrained(self.model_name).to(self.
 ↪device)
        self.max_len = 128

    def vectorize(self, sentence: str) -> np.array:
        inp = self.tokenizer.encode(sentence, add_special_tokens=True)
        len_inp = len(inp)

        if len_inp >= self.max_len:
            inputs = inp[:self.max_len]
            masks = [1] * self.max_len
        else:
```

```
            inputs = inp + [0] * (self.max_len - len_inp)
            masks = [1] * len_inp + [0] * (self.max_len - len_inp)

        inputs_tensor = torch.tensor([inputs], dtype=torch.long).to(self.device)
        masks_tensor = torch.tensor([masks], dtype=torch.long).to(self.device)

        with torch.no_grad():
            bert_out = self.bert_model(inputs_tensor, masks_tensor)
            seq_out, pooled_out = bert_out.last_hidden_state, bert_out.
 ↪pooler_output

        if torch.cuda.is_available():
            return seq_out[0][0].cpu().detach().numpy()
        else:
            return seq_out[0][0].detach().numpy()

BSV = BertSequenceVectorizer()

# Assuming 'data' is your DataFrame containing poems and titles
data['Poem_bert'] = tqdm(data['Poem'].apply(lambda x: BSV.vectorize(x)))

X_train, X_test, y_train, y_test = train_test_split(data['Poem_bert'],␣
 ↪data['Title'], test_size=0.2, random_state=42)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)

logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(list(X_train), y_train_encoded)

def generate_poem(title):
    vectorized_poem = BSV.vectorize(title)
    predicted_title_encoded = logistic_model.predict([vectorized_poem])[0]
    predicted_title = label_encoder.
 ↪inverse_transform([predicted_title_encoded])[0]
    return predicted_title

# Example usage:
given_title = "Brian If you knew who I was now When I knew who you were then␣
 ↪Would you forgive me before We ever became them?"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Token indices sequence length is longer than the specified maximum sequence
length for this model (620 > 512). Running this sequence through the model will
result in indexing errors
100%|     | 10000/10000 [00:00<00:00, 1361610.18it/s]

Predicted Poem based on the given title:
poem
```

[16]:
```
given_title = "Oranges"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
grain field
```

[17]:
```
given_title = "Haunted"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
white hunter
```

[18]:
```
given_title = "Forever and Always"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
lord
```

[19]:
```
given_title = "Ocean"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
useless useless
```

[20]:
```
given_title = "Summer time"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
snow melting
```

[21]:
```
given_title = "Road not taken"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
hotel
```

[22]:
```
given_title = "Cats"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
still life 1
```

[23]:
```
given_title = "Earth"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
earth shake
```

[24]:
```
given_title = "Life"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
amidwives two portrait
```

[25]:
```
given_title = "Mirror"
predicted_poem = generate_poem(given_title)
print("Predicted Poem based on the given title:")
print(predicted_poem)
```

```
Predicted Poem based on the given title:
still life 1
```