```
from google.colab import drive
drive.mount('/content/gdrive')

SOURCE_DIR = 'gdrive/MyDrive/ACL/'

    Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=
```

```
import torch
import re
import numpy as np


cos = torch.nn.CosineSimilarity(dim=0, eps=1e-6)


def find_k_nearest_neighbors(word, embedding_dict, k):
  words_cosine_similarity = dict()
  for token in embedding_dict.keys():
    words_cosine_similarity[token] = cos(embedding_dict[word], embedding_dict[token]).item()
  words_cosine_similarity = dict(sorted(words_cosine_similarity.items(), key=lambda item: item[1]))
  return list(words_cosine_similarity.keys())[-k:][::-1]

def delete_hashtag_usernames(text):
  try:
    result = []
    for word in text.split():
      if word[0] not in ['@', '#']:
        result.append(word)
    return ' '.join(result)
  except:
    return ''

def delete_url(text):
  text = re.sub(r'http\S+', '', text)
  return text


word = 'زندگی'
k = 10
```

0. Data preprocessing

```
!pip install json-lines

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting json-lines
      Downloading json_lines-0.5.0-py2.py3-none-any.whl (6.8 kB)
    Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from json-lines) (1.15.0)
    Installing collected packages: json-lines
    Successfully installed json-lines-0.5.0
```

```
import json_lines
```

```
# 1. extract all tweets from files and save them in memory base on each year.
import pandas as pd
df = pd.read_csv('mahsa_amini_data.csv', usecols= ['Text'])
#print(df['Text'][0])
# 2. remove urls, hashtags and usernames.
data = [delete_hashtag_usernames(delete_url(text)) for text in df['Text']]
print(data)

    ['بنشین تا شود نقش فال ما نقش هم\u200cمال کشی کرده.', 'برای ایران، برای مهسا.', 'مرگ بر دیکتاتور', 'نذاریم خونشون پایمال شه
```

1. One hot encoding

```
# 1. find one hot encoding of each word for each year
distinct_words = list(set([word for snt in data for word in snt.split(' ')]))
```

```
n = len(distinct_words)
one_hot = dict()
for i, word in enumerate(distinct_words):
  array = np.zeros(n)
  array[i] = 1
  one_hot.update({word : torch.tensor(array)})
# 2. find 10 nearest words from "زندگی"
print(find_k_nearest_neighbors('زندگی', one_hot, 10))
```

```
    ['زندگی.', 'ایران:ا', 'پولش', 'بودبراسردشدن', 'دوشانِ', 'خوبیه:)', 'حالم', 'بدبختای', 'جداگانه', 'رنگ']
```

pros: it helps in representing categorical data By representing categorical data as binary vectors, one-hot encoding helps models better capture relationships between features and increases their predictive power. cons: One-hot encoding can increase the number of features, which can lead to challenges. one-hot encoding can result in sparse data and most of the encoded features are zeros

2. TF-IDF

```
from os import ST_NOATIME
import math

# 1. find the TF-IDF of all tweets.
tf_all = list()
#data=['he is good', 'he is he']
idf = dict()
for snt in data:
  for word in snt.split(' '):
    idf.update({word: 0})
for snt in data:
  tf = dict()
  for word in snt.split(' '):
    tf.update({word: 0})
  for word in snt.split(' '):
    tf.update({word: int(tf[word]) + 1})
    idf.update({word: int(idf[word] + 1)})
  n = len(snt.split(' '))
  for word in tf:
    tf.update({word: float(tf[word]) / n})
  tf_all.append(tf)
unique = len(idf)
list_word = dict()
for i, word in enumerate(idf):
  idf.update({word: math.log(unique/idf[word])})
  list_word.update({word: i})

array = np.zeros((len(data), unique))
for i,snt in enumerate(data):
  for word in snt.split(' '):
    array[i, list_word[word]] = tf_all[i][word] * idf[word]

dict_tf_idf = dict()
for i, snt in enumerate(data):
  dict_tf_idf.update({snt: torch.tensor(array[i])})

# 2. choose one tweets randomly.
# 3. find 10 nearest tweets from chosen tweet.
find_k_nearest_neighbors(data[1], dict_tf_idf, 10)
```

```
    ['این گوزو رو کی گردن میگیره؟؟ دچار زوال عقل شده از بس پای منبر دستمال کشی کرده.',
     'برای پدرم که به خاطر کرونا داشت دچار زوال عقل میشد و شده بود سی کیلو اما طاقت آورد...',
     'دچار آزادی شده ایم!!! دچار یعنی عاشق!!!!',
     'برای دستای بی دستمال',
     'سرکارخانم پرستو گوزو در حاله ماله کشی ریدم دهن آدم دروغگو',
     'دیگه هرچی بالای منبر بگین کیرمونم نیست🖕',
     'زنی خفته در خاک ایران را بیدار کرده.',
     'از بس ذهن کثافتی دارن',
     'به نیت رهایی منبر رسول الله و روضه سیدالشهدا',
     'ازادی از رگ گردن نزدیک تر است']
```

pros: TF-IDF measures the importance of a word based on its frequency and rarity, which can help capture the semantic meaning of words. so it can be used in search engines to rank documents based on their relevancy. cons: TF-IDF does not capture the order or context of words in a document.

3. Word2Vec

```
# 1. train a word2vec model base on all tweets for each year.
from gensim.models import Word2Vec

sentences = list()
for snt in data:
  temp_list = list([word for word in snt.split(' ')])
  sentences.append(temp_list)

model = Word2Vec(sentences)
# 2. find 10 nearest words from "زندگی"
model.wv.most_similar('زندگی')
```

```
[('،زندگی', 0.9909707307815552),
 ('زن', 0.9786562919616699),
 ('آزادی', 0.978026270866394),
 ('،', 0.9778410196304321),
 ('زندگی،', 0.9747486710548401),
 ('امید', 0.9689344167709351),
 ('ازادی', 0.9639194011688232),
 ('زن،', 0.9611381888389587),
 ('میهن', 0.957511305809021),
 ('ابادی', 0.9512325525283813)]
```

pros: Word2Vec is designed to capture the semantic relationships between words, such as synonyms, antonyms, and analogies.Word2Vec generates vector representations of words, which can be used as features models for text classification.Word2Vec can help reduce the sparsity of text data. cons: Training a Word2Vec model can be computationally expensive.it requires a large data in order to have meaningful relationships between words. and it may not work well on multilingual processings

4. Contextualized embedding

```
model_checkpoint = "HooshvareLab/bert-base-parsbert-uncased"
```

```
!pip install transformers[sentencepiece]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers[sentencepiece]
  Downloading transformers-4.26.1-py3-none-any.whl (6.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 41.4 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece])
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]) (3.9.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]) (1.2
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]) (4.64
Collecting huggingface-hub<1.0,>=0.11.0
  Downloading huggingface_hub-0.12.1-py3-none-any.whl (190 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.3/190.3 KB 20.0 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]) (2.25.1
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 58.7 MB/s eta 0:00:00
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece]) (6.0
Collecting sentencepiece!=0.1.92,>=0.1.91
  Downloading sentencepiece-0.1.97-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 62.4 MB/s eta 0:00:00
Requirement already satisfied: protobuf<=3.20.2 in /usr/local/lib/python3.8/dist-packages (from transformers[sentencepiece])
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->transformers[
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers[ser
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers[sentencep
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers[sent
Installing collected packages: tokenizers, sentencepiece, huggingface-hub, transformers
Successfully installed huggingface-hub-0.12.1 sentencepiece-0.1.97 tokenizers-0.13.2 transformers-4.26.1
```

```
from transformers import BertModel,BertTokenizer

model = BertModel.from_pretrained(model_checkpoint, output_hidden_states = True)
tokenizer = BertTokenizer.from_pretrained(model_checkpoint)
```

```
# 1. fine tune a bert model base on all tweets for each year.

# 2. find 10 nearest words from "زندگی"
```

Colab paid products - Cancel contracts here

✓ 0s completed at 1:37 PM