به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

پردازش زبان طبیعی

تمرین شماره ۲ - بخش دوم

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

فروردین‌ماه ۱۴۰۳

فهرست مطالب

۱_ پاسخ سوال دوم

ابتدا داده‌ها در گوگل درایو آپلود شد تا در کولب از آن‌ها استفاده کنیم.:

```
[3] import pandas as pd
    from google.colab import drive
    drive.mount('/content/gdrive')

    Mounted at /content/gdrive
```

Part 1: Preproccessing

```
data = pd.read_json('/content/gdrive/MyDrive/Colab Notebooks/sarcasm.json', lines=True)
data.head()
```

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scien...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...

۱-۱_ پیش‌پردازش داده‌ها

در این مرحله برخی پیش‌پردازش‌ها را اعمال کردیم. برای اینکه متوجه بشویم که آیا مفید هستند یانه در مرحله آخر که ارزیابی انجام می‌شود با حذف آن‌ها امتیازات بدست آمده را مقایسه می‌کنیم. پس این پیش‌پردازش‌هایی که اعمال می‌شوند مثل تبدیل به حرف کوچک ممکن است برای تشخیص کنایه اصلا مفید نباشد ولی در این مرحله آن‌ها را اعمال می‌کنیم سپس در مرحله آخر با حذف آن‌ها تاثیر آن‌ها را می‌سنجیم.

تبدیل حروف بزرگ به کوچک

هرچند که نوع نوشتن کلمات در کنایه آمیز بودن آن تاثیر دارد ولی این این مرحله این موضوع را به صورت آزمایشی اعمالم می‌کنیم. همانند سوال قبل این مورد به صورت زیر انجام می‌شود:

```
# Normalization -> lower case
data['headline'] = data['headline'].str.lower()
data.head()
```

	is_sarcastic	headline	article_link
0	1	thirtysomething scientists unveil doomsday clo...	https://www.theonion.com/thirtysomething-scien...
1	0	dem rep. totally nails why congress is falling...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	eat your veggies: 9 deliciously different recipes	https://www.huffingtonpost.com/entry/eat-your-...
3	1	inclement weather prevents liar from getting t...	https://local.theonion.com/inclement-weather-p...
4	1	mother comes pretty close to using word 'strea...	https://www.theonion.com/mother-comes-pretty-c...

حذف StopWords انگلیسی از متن

باتوجه به اینکه تا کنون تشخیص کنایه انجام نداده‌ایم این مورد هم مثل مورد قبلی است و نمیدانم که در این روش حذف آن‌ها چقدر تاثیر گذار است. این مورد نیز در مرحله آخر بررسی خواهدشد:

```
# Removing Stop Words
stop_words = set(stopwords.words('english'))
def remove_stopwords(text):
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]
    text = ' '.join(tokens)
    return text
data['headline'] = data['headline'].apply(lambda text: remove_stopwords(text))
```

نرمال‌سازی با حذف علائم نگارشی

علائم نگارشی احتمالا تاثیری در کنایه نداشته باشند. هرچند که ممکن است برخی جاها علامت سوال کاربرد داشته باشد که یک فکت به عنوان یه پرسش مطرح شود. میزان تاثیر حذف آن‌ها بعدا بررسی می‌شود. به این منظور تابع remove_punctuation تعریف شده که در این تابع علامات نگارشی با کمک string.punctuation با پوچ جایگذاری می‌شوند که بر روی تمامی headlineها اعمال می‌شود :

```
[25] # Normalization -> Remove Punctuation
    import string
    def remove_punctuation(text):
        translator = str.maketrans('', '', string.punctuation)
        return text.translate(translator)

    data['headline'] = data['headline'].apply(lambda text: remove_punctuation(text))
```

اعمال توکنایزر

برای توکنایز کردن متن هر توییت از پکیج NLTK تابع word_tokenize وارد می‌شود. سپس با کمک تابع تعریف شده متن هر توییت با توکن‌های آن جایگزین می‌شود:

```
# Tokenization
from nltk.tokenize import word_tokenize
def tokenize_text(text):
    tokens = word_tokenize(text)
    return tokens
data['headline'] = data['headline'].apply(lambda text: word_tokenize(text))
data.head()
```

	is_sarcastic	headline	article_link
0	1	[thirtysomething, scientists, unveil, doomsday...	https://www.theonion.com/thirtysomething-scien...
1	0	[dem, rep, totally, nails, congress, falling, ...	https://www.huffingtonpost.com/entry/donna-edw...
2	0	[eat, veggies, 9, deliciously, different, reci...	https://www.huffingtonpost.com/entry/eat-your-...
3	1	[inclement, weather, prevents, liar, getting, ...	https://local.theonion.com/inclement-weather-p...
4	1	[mother, comes, pretty, close, using, word, st...	https://www.theonion.com/mother-comes-pretty-c...

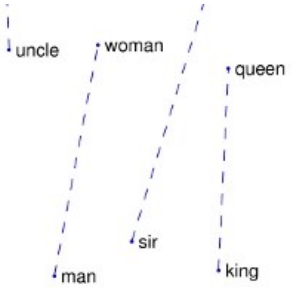
جداسازی داده آزمون

باتوجه به اینکه بیان شده ۲۰ درصد داده‌ها باید داده آزمون باشند، پس ابتدا نمونه‌برداری‌هایی که از هرکلاس داشتیم را ترکیب کرده و ۲۰ درصد آن‌ها را به عنوان داده آزمون و ۸۰ درصد آن‌ها را به عنوان داده آموزش دسته‌بندی می‌کنیم:

```
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

۲-۱_ بارگذاری GLOVE

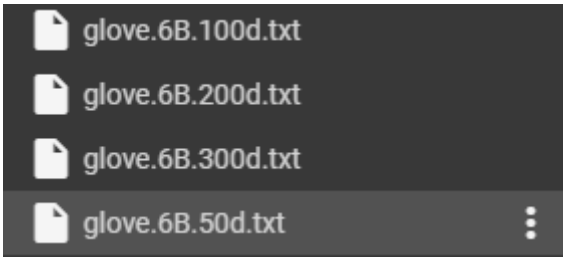
ابتدا لازم است وکتورهای کلماتی که آموزش داده‌شده اند دانلود شده و برای استفاده در گوگل درایو بارگذاری شود. همانطور که در درس خواندیم بردار کلمات به ما کمک می‌کند که ارتباط معنایی بین کلمات را به فرمتی تبدیل کنیم که می‌توان با کمک آن مسائل مهمی از پردازش زبان طبیعی را حل کرد. همچنین ارتباط بین دو کلمه به اندازه فاصله بردار آن‌ها خواهد بود. مثلا ارتباط بین ایران و تهران که پایتخت آن است باید به اندازه ژاپن و توکیو باشد که این مباحث در کلاس درس به طور کلی بررسی شد. این نوع از رویکرد بر خلاف سوال قبلی وکتورهای dense می‌سازد.



ابتدا فایل مربوطه را دانلود کرده و از حالت فشرده خارج می‌کنیم:

```
Download on Google Drive and unzip

!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip

--2024-04-02 17:46:23--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-04-02 17:46:23--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-04-02 17:46:23--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.00MB/s    in 2m 40s

2024-04-02 17:49:03 (5.15 MB/s) - 'glove.6B.zip' saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

این فایل فشرده شامل فایل‌های زیر است که برای ابعاد مختلف ارائه شده است:



مثلا برای ۵۰ بعدی فایل به صورت زیر است :

```
off 0.45341 -0.71044 0.25826 -0.015757 0.14649 0.031418 -1.0812 0.31886 0.055295 -0.12677 -0.45668 0.030
south -0.039674 0.47097 -0.31102 0.38088 -0.59254 -0.38231 -0.68876 0.15252 0.46303 -0.9099 -0.38665 -0.
american -1.0988 0.48544 -0.5004 0.1491 0.41042 0.32605 -1.4592 -0.52025 -0.088054 -0.2384 0.43632 -0.14
minister 0.039931 0.50725 -0.18625 0.32268 0.65427 0.20286 -0.4966 0.13383 -1.7112 -1.0943 0.10988 0.747
police 0.49725 -1.1949 0.37137 -0.081662 0.69114 -0.69982 -0.25723 0.5943 0.059978 -1.499 -0.07122 -1.00
well 0.27691 0.28745 -0.29935 -0.19964 0.12956 0.15555 -0.64522 -0.3409 -0.11833 0.15798 0.13969 0.24872
including 0.48518 0.56214 0.064657 0.44652 -0.29881 0.91376 -0.84552 -0.93462 0.093077 0.05312 -0.25653
team -0.62801 0.12254 -0.3914 0.87937 0.28572 -0.41953 -1.4265 0.80463 -0.27045 -0.82499 1.0277 0.18546
international -0.1666 0.68994 -0.76138 0.55857 -0.28302 -1.0075 -0.35157 -0.46751 0.99938 0.21826 0.8740
week 0.25009 -0.033956 0.094114 0.32336 -0.01654 -0.63211 -1.2778 0.32265 -0.14109 -0.29001 -0.73881 -1.
officials 0.99818 -0.44131 0.39641 0.36621 -0.23042 -0.84895 -0.82396 0.63432 0.078578 -1.0304 -0.11953
still 0.47689 -0.076447 0.36768 -0.3947 0.47169 -0.038062 -0.78328 0.14814 -0.30044 -0.072707 -0.030916
both 0.25707 0.25305 -0.082529 0.12036 0.19351 0.66386 -0.45209 -0.084674 -0.57591 -0.0055412 0.35637 0.
even 0.38336 -0.095871 0.12229 -0.51625 0.3491 0.1705 -0.55374 -0.0017357 -0.47808 0.43859 -0.25184 0.16
high -0.65758 1.2502 0.19082 -0.6742 -0.10125 -0.046932 -0.27648 -0.48253 0.39254 0.3704 -0.24111 -0.642
part 0.70504 0.18255 -0.75188 0.0039397 -0.053423 0.29625 -0.42377 -0.013111 0.16573 -0.43932 0.07506 -0
told 0.37633 0.058652 0.17005 0.46863 0.95799 -0.82027 -0.83683 0.53315 -0.22664 -1.1505 0.11026 0.22662
those 0.65102 0.0025814 0.45799 -0.48064 0.38459 0.56754 -0.44011 -0.44394 -0.53883 0.17027 -0.27204 0.1
end -0.04116 0.22243 -0.11458 -0.33628 0.038872 -0.066803 -0.58309 -0.19971 -0.51087 -0.29037 -0.49177 -
former -0.53526 0.54543 0.11158 0.59775 0.30465 0.19476 -1.2582 0.32169 -0.89948 -0.94368 -0.0071382 0.3
these 1.0074 0.18912 -0.11732 -0.36526 -0.051616 0.54116 -0.15308 -0.52727 -0.48022 -0.0072622 -0.098221
```

برای هر کلمه یک بردار ۵۰ بعدی وجود دارد. حال فایل متنی را ایمپورت کرده و در یک آرایه قرار می‌دهیم. به این منظور از ایده موجود در یکی از پرسش‌های مطرح‌شده در stackoverflow استفاده شد. باتوجه به اینکه هر خط ۱ کلمه است، آن را به فضای دیکشنری در پایتون میبریم که هر کلمه مقداری برابر تمام اعداد بعد خود دارد که همان مقادیر بردار آن است.

```
import numpy as np
path = "/content/glove.6B.50d.txt"
word_vectors = {}
with open(path, 'r', encoding='utf-8') as f:
    for line in f:
        lines = line.split()
        word = lines[0]
        vector = np.asarray(lines[1:], dtype='float32')
        word_vectors[word] = vector
```

```
word_vectors
```

```
{'the': array([ 4.1800e-01,  2.4968e-01, -4.1242e-01,  1.2170e-01,  3.4527e-01,
       -4.4457e-02, -4.9688e-01, -1.7862e-01, -6.6023e-04, -6.5660e-01,
        2.7843e-01, -1.4767e-01, -5.5677e-01,  1.4658e-01, -9.5095e-03,
        1.1658e-02,  1.0204e-01, -1.2792e-01, -8.4430e-01, -1.2181e-01,
       -1.6801e-02, -3.3279e-01, -1.5520e-01, -2.3131e-01, -1.9181e-01,
       -1.8823e+00, -7.6746e-01,  9.9051e-02, -4.2125e-01, -1.9526e-01,
        4.0071e+00, -1.8594e-01, -5.2287e-01, -3.1681e-01,  5.9213e-04,
        7.4449e-03,  1.7778e-01, -1.5897e-01,  1.2041e-02, -5.4223e-02,
       -2.9871e-01, -1.5749e-01, -3.4758e-01, -4.5637e-02, -4.4251e-01,
        1.8785e-01,  2.7849e-03, -1.8411e-01, -1.1514e-01, -7.8581e-01],
      dtype=float32),
```

برای نمونه وکتور کلمه ایران به صورت زیر است:

```
word_vectors["iran"]

array([-0.18997  ,  0.11493  ,  0.85566  , -0.039811 ,  0.10742  ,
       -0.44042  ,  1.2496   ,  0.49928  ,  0.58689  ,  0.8321   ,
        0.027948 , -0.85445  , -0.39854  , -0.18763  , -0.050099 ,
        0.95036  ,  0.59861  ,  0.25454  ,  0.6548   ,  0.87505  ,
        0.82139  , -0.0041283,  0.9193   , -0.033385 ,  0.1914   ,
       -3.0393   ,  0.58703  ,  0.23673  ,  0.031058 ,  0.17775  ,
        2.4503   , -0.35655  , -0.68777  , -0.43984  ,  0.12271  ,
       -0.46345  , -0.29642  ,  0.33648  , -1.6442   ,  0.23183  ,
       -0.019779 ,  0.0057172,  0.94701  , -1.2708   ,  0.53767  ,
        0.80297  , -0.70422  ,  1.7059   , -0.64729  , -0.97299  ],
      dtype=float32)
```

حال باید embedding vector تشکیل شود.

ابتدا تمامی کلماتی که در Vocabulary هستند را در یک ماتریس قرار می‌دهیم و مقدار متناظر با بردار آن‌ها را در آن قرار می‌دهیم. برای نمونه مشاهده می‌شود که کلمه action به خوبی در ماتریس جدید قرار داده شده است:

```
num_docs = len(x_train)
num_words = len(vocabulary)
embedding_matrix = np.zeros((num_words, dim), dtype=np.float32)

[30] for i in range(len(vocabulary)):
       vector = word_vectors.get(vocabulary[i])
       if vector is not None:
         embedding_matrix[i] = vector

vocabulary[900]

'action'

[31] word_vectors["action"]

array([ 2.4196e-01, -7.4387e-01, -5.1436e-01, -8.9699e-02, -2.6825e-01,
        4.5664e-01,  7.2555e-01, -2.1819e-01,  4.2826e-02,  1.8943e-01,
       -3.2057e-01,  5.7118e-02, -5.9674e-01,  2.2751e-01,  1.2945e-01,
       -2.9623e-01,  6.4950e-01, -2.2105e-01, -5.7708e-01, -4.5436e-01,
        2.5636e-01,  9.9275e-02, -1.1760e-01, -1.6338e-01, -2.3526e-01,
       -1.8493e+00, -1.0322e-01, -3.5198e-01,  2.4535e-03,  3.3223e-02,
        3.3513e+00,  1.5836e-01, -1.0583e+00, -7.2565e-01, -5.7345e-01,
       -5.7453e-02,  4.7996e-01, -9.9358e-01, -8.0074e-01, -6.2818e-01,
       -2.2644e-01,  1.4946e-01, -3.2743e-02, -3.3740e-01,  2.0041e-01,
       -2.3766e-03,  1.2109e-02,  5.5461e-01,  1.5185e-01,  7.0218e-01],
      dtype=float32)

[32] embedding_matrix[900]

array([ 2.4196e-01, -7.4387e-01, -5.1436e-01, -8.9699e-02, -2.6825e-01,
        4.5664e-01,  7.2555e-01, -2.1819e-01,  4.2826e-02,  1.8943e-01,
       -3.2057e-01,  5.7118e-02, -5.9674e-01,  2.2751e-01,  1.2945e-01,
       -2.9623e-01,  6.4950e-01, -2.2105e-01, -5.7708e-01, -4.5436e-01,
        2.5636e-01,  9.9275e-02, -1.1760e-01, -1.6338e-01, -2.3526e-01,
       -1.8493e+00, -1.0322e-01, -3.5198e-01,  2.4535e-03,  3.3223e-02,
        3.3513e+00,  1.5836e-01, -1.0583e+00, -7.2565e-01, -5.7345e-01,
       -5.7453e-02,  4.7996e-01, -9.9358e-01, -8.0074e-01, -6.2818e-01,
       -2.2644e-01,  1.4946e-01, -3.2743e-02, -3.3740e-01,  2.0041e-01,
       -2.3766e-03,  1.2109e-02,  5.5461e-01,  1.5185e-01,  7.0218e-01],
      dtype=float32)
```

حال یک ماتریس برای تمامی document ها که همان عناوین هستند، می‌سازیم.

این ماتریس ۳ بعدی است. در این ماتریس هر سطر یک document است. در این ماتریس باید مقدار بردار کلمات تشکیل دهنده آن قرار داده شود پس ستون‌ها یا بعد دوم آن برابر تعداد کلمات استفاده شده در آن است. باتوجه به اینکه عناوین طول متفاوتی دارند، بعد دوم را برابر بیشترین کلمات موجود در داده‌ها قرار می‌دهیم. بعد سوم نیز برای ذخیره بردار هر کلمه استفاده می‌شود که اندازه آن بستگی به ابعاد استفاده شده است مثلا ۳۰۰.

برای نمونه در این نمونه‌برداری از داده‌ها، بیشترین کلمات برابر ۲۷ است:

```
[33] max_length = max(len(doc) for doc in x_train)

max_length
```

برای ماتریس نهایی راهکارهای مختلفی وجود دارد تا بتوانیم به رگرسیون لجستیک بدهیم. یک روش میانگین‌گیری است. یک روش جمع تمامی بردارهای کلمات هر داکیومنت است. ولی ما از همان روش مذکور استفاده کرده و یک ماتریس ۳ بعدی اولیه میسازیم:

```
max_length = max(len(doc) for doc in x_train)

[63] train_document_matrix = np.zeros((num_docs, max_length, dim))
     for i, doc in enumerate(x_train):
         for j, word in enumerate(doc):
             embedding_vector = embedding_matrix[vocab_index[word]]
             if embedding_vector is not None:
                 train_document_matrix[i, j] = embedding_vector

train_document_matrix

array([[[ 4.57690001e-01,  8.51909995e-01,  3.10979992e-01, ...,
         -7.42259979e-01,  7.70529985e-01,  2.63940006e-01],
        [ 1.66750001e-03, -1.63760006e-01, -9.26479995e-02, ...,
         -2.49060001e-02, -6.95720017e-02,  1.13429999e+00],
        [-8.67509991e-02, -1.04390003e-01, -4.84620005e-01, ...,
         -2.46340007e-01, -1.33120000e-01,  8.94259989e-01],
```

همین کار برای داده‌های تست انجام می‌شود و با کمک ماتریس وکتورهای داده‌های آموزشی ماتریس نهایی آن را برای ارزیابی مدل میسازیم.

```
# Test Data Matrix
vocabulary_test = set()
for document in x_test:
  for word in document:
    vocabulary_test.add(word)
vocabulary_test = sorted(list(vocabulary_test))
vocab_index_test = {word: i for i, word in enumerate(vocabulary_test)}

num_docs_test = len(x_test)
num_words_test = len(vocabulary_test)
max_length_test = max(len(doc) for doc in x_test)

test_document_matrix = np.zeros((num_docs_test, max_length_test, dim))
for i, doc in enumerate(x_test):
    for j, word in enumerate(doc):
        embedding_vector = embedding_matrix[vocab_index_test[word]]
        if embedding_vector is not None:
            test_document_matrix[i, j] = embedding_vector
```

حال دو ماتریس بردارهای جانمایی برای داده‌های آموزشی و تست داریم. تاکید می‌شود که در این تمرین، تهیه ماتریس داده‌های آموزشی بر اساس بردارهایی بود که باتوجه به داده‌های آموزشی بدست آورده‌ایم. میتوانستیم مستقیم از Glove استفاده کنیم ولی من هدفم این بود که داده‌های تست کاملا مستقل باشند و کاملا وابسته به داده‌های آموزشی باشند.

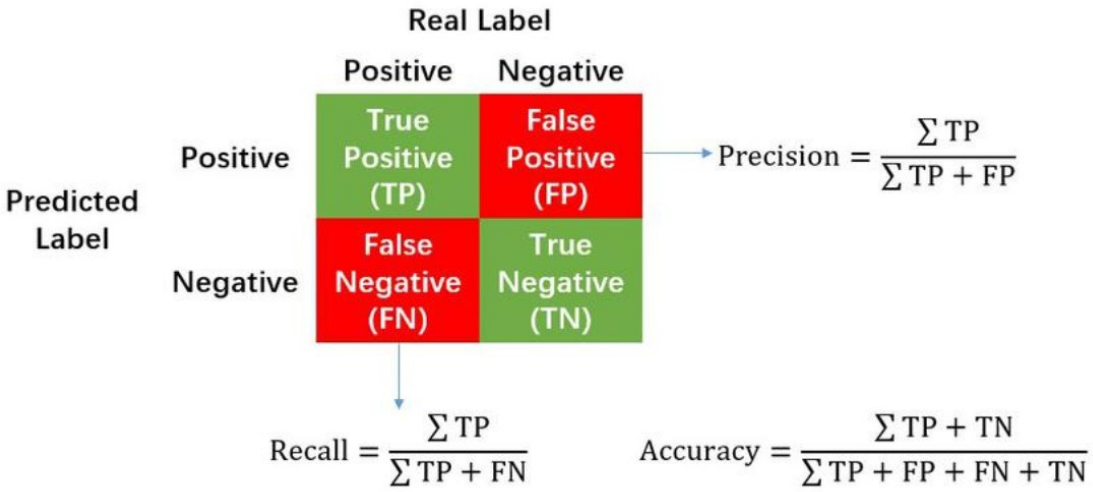
۳-۱_ مدل رگرسیون لجستیک

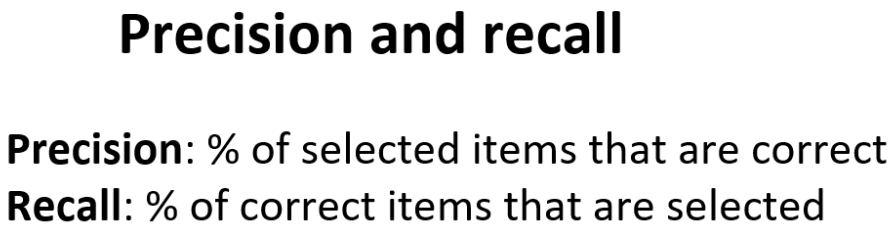
برای حل این مسئله از روش رگرسیون لجستیک برای دسته‌بندی استفاده می‌کنیم. نکته‌ای که باید رعایت شود این است که باید ماتریس ۳ بعدی به ۲ بعدی تبدیل شود تا بتوانیم از این روش استفاده کنیم. برای این کار ۲ راه داریم. یک اینکه کلا دو بعدی بعدی را ۱ بعد در نظر بگیریم. یا اینکه از بردارها میانگین بگیریم. یا اینکه تمامی آن‌ها را جمع کنیم.

Part3: Logestic Regression

```
# 3D Matrix to 2D
train_data_flattened = np.sum(train_document_matrix, axis=1)
test_data_flattened = np.sum(test_document_matrix, axis=1)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, precision_score, recall_score
model = LogisticRegression(max_iter=1500)

model.fit(train_data_flattened, y_train)

y_pred = model.predict(test_data_flattened)
```





Precision همان نسبت تمام موارد مثبتی که درست تشخیص داده شده به موارد مثبت است که درست تشخیص داده شده و یا به اشتباه مثبت تشخیص داده شده‌اند یعنی مدل چقدر موارد مثبتی که تشخیص داده درست هستند..Recall یا فراخوانی برابر است با نسبت تعداد موارد مثبتی که درست تشخیص شده به مواردی است که مثبت بوده و درست تشخیص داده و یا منفی بوده و به اشتباه مثبت تشخیص داده

است یعنی مدل چقدر از موارد مثبت را توانسته تشخیص دهد. F1-Score یک معیار ارزیابی ترکیبی است که باتوجه به ضریب آن مقدار تاثیر هرکدام تعیین می‌شود.

روش جمع بردارها نتایج زیر را خروجی می‌دهد:

```
F1-score: 0.5361
Precision: 0.5232
Recall: 0.5497
```

صحت مدل حدود ۵۳ درصد است. ولی میزان فراخوانی مناسب نیست. یعنی حدود ۵۰ درصد داده‌هایی که کنایه هستند شناسایی نشده‌اند. ولی خب حدود ۵۰ درصد آن‌هایی که شناسایی شده اند درست هستند.

روش میانگین گیری نتایج زیر را خروجی می‌دهد که در شناسایی کنایه بسیار ضعیف است:

```
F1-score: 0.0095
Precision: 0.8667
Recall: 0.0048
```

اگر در مرحله پیش‌پردازش کلمات stopword را حذف نکنیم ، جمع نتایج زیر خروجی می‌دهد:

```
F1-score: 0.5917
Precision: 0.4842
Recall: 0.7604
```

مشاهده می‌شود که صحت و فراخوانی افزایش پیدا کرد. پس بهتر است stopwords را برای تشخیص کنایه در این دیتاست حذف نکنیم.

حال برای بررسی بیشتر، اعداد را در مرحله پیش‌پردازش حذف می‌کنیم.

```
F1-score: 0.5700
Precision: 0.4962
Recall: 0.6695
```

پس اعداد هم در نتایج تاثیر دارند. پس تمامی پیش‌پردازش‌های قبلی مناسب هستند و نیازی نیست اعداد هم حذف شوند. با حذف اعداد کنایه‌های کمتری شناسایی می‌شوند. زیرا که فراخوانی کاهش یافته است.

تمامی موارد بالا بررسی شده در حالت ۵۰ بعدی بودند. حال ابعاد هر کلمه را به ۳۰۰ تغییر داده و دوباره بررسی می‌کنیم. نتایج زیر برای حالتی است که StopWords حذف شده و از روش جمع برای تبدیل آرایه ۳ بعدی به ۲ بعدی استفاده شده:

```
F1-score: 0.5457
Precision: 0.5034
Recall: 0.5958
```

مشاهده می‌شود که با افزایش تعداد ابعاد بردار کلمات، فراخوانی افزایش یافته است.

حال stopwords را حذف نمیکنیم:

```
F1-score: 0.5789
Precision: 0.4717
Recall: 0.7490
```

فراخوانی به دقت قابل قبولی رسیده است.

تا اینجا ما در داده‌های تست از Glove به صورت مستقیم استفاده نکردیم و فقط بردار کلماتی استفاده شد که در داده‌های آموزشی بودند. حال اگر برای داده‌های تست هم مستقیما از Glove استفاده کنیم نتیجه به صورت زیر است:

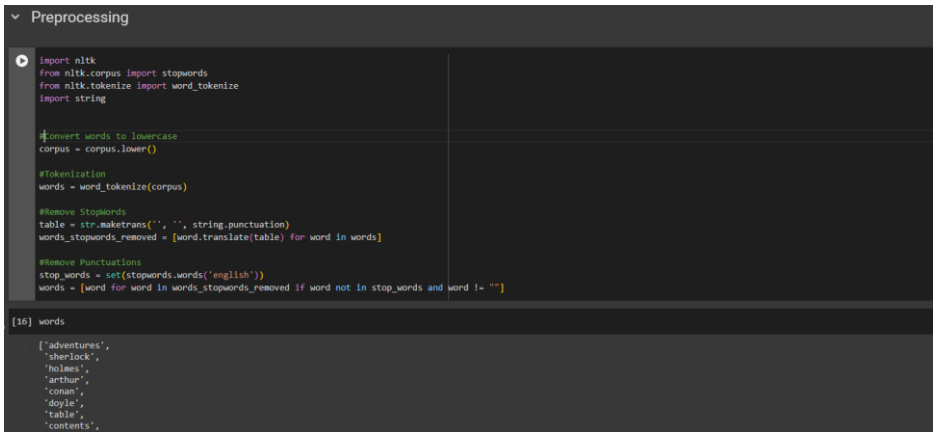
```
F1-score: 0.7506
Precision: 0.7602
Recall: 0.7413
```

مشاهده می‌شود که در این حالت هم فراخوانی هم صحت مقدار بسیار بالاتری نسبت به قبلی ها دارند. پس مشکل اینکه Precision کم بود این است که در روش قبلی، برای تشکیل بردار کلمات داده‌های تست فقط از کلماتی استفاده کردیم که در داده‌های آموزشی از Glove گرفته شد. ولی در آخرین مرحله هم برای داده‌های آموزشی و هم برای داده‌های تست از Glove به صورت مستقیم استفاده کردیم.

۲_پاسخ سوال سوم

در مقاله‌ای که ارائه شده ابتدا به توضیحات کلی روش Skip Gram پرداخته و سپس با معرفی تکنیک NegativeSampling سرعت را افزایش داده است.

ابتدا پیش‌پردازش‌های لازم را انجام می‌دهیم:



همانند مسائل قبلی این تمرین، هم تمامی کلمات به حروف کوچک تبدیل شد و هم Stopwords حذف شد و در انتها هم علائم نگارشی حذف شدند.

۱-۲_ پاسخ بخش اول

باتوجه به اینکه هدف امبدینگ است و ما باید هر کلمه را به یک بردار تبدیل کنیم ابتدا باید روش مشخص شود. در این سوال باید از روش Skip-Gram این کار را انجام دهیم. این روش مانند یک Classifier است با این تفاوت که ما به دنبال بردار ویژگی هایی هستیم که کار دسته‌بندی را انجام می‌دهند. در Skip-Gram کلمه گرفته می‌شود و به اندازه پنجره کلمه می‌دهد. مثلا پنجره ۲ یعنی ۲ کلمه قبلی و ۲ کلمه بعدی را می‌دهد . کلاس + برابر تمام ۲ کلمه قبلی و بعدی در Corpus هستند و کلاس – کلمات غیر همسایه هستند که تعدادشان زیاد است. معمولا k برابر مثبت، کلمات منفی به صورتی تصادفی انتخاب می‌شود. که معمولا k برابر ۲ است. ما کاری به کلاس مثبت و منفی نداریم و وزن ها را می‌خواهیم. بردارهایی که ضرب داخلی آن‌ها بزرگ شود با هم می‌آید. (کاری می‌کنیم که اینطور شود). به ازای هر کلمه ۲ بردار target و context داریم. که اندازه برای مثال طبق مطالب درس بردار target $|V| \times 300$ و بردار context $300 \times |V|$ است. کاری می‌کنیم که ضرب داخلی مثبت‌ها زیاد و منفی ها کم شود.در نهایت برای Word Embedding می‌توان از بردار های target و یا context یا جمع آن‌ها استفاده کرد.

در این سوال گفته شده که دو ماتریس Embedding و Context ساخته شود که$100 \times |V|$ هستند. embedding که ماتریسی برای کلمات Target است و Context نیز برای همان Context است.

Part1 - Embedding and Context Matrix

```
vocab_size = len(vocabulary)
embedding_size = 100

W_embedding = np.random.randn(vocab_size, embedding_size)
W_context = np.random.randn(vocab_size, embedding_size)
```

در ابتدا اعدادی که بردارها تخصیص داده‌شده اعداد تصادفی هستند. ضرب داخلی بردار هر کلمه از embedding در context نشان دهنده Similariry آن کلمه است.

قبل از انتخاب نمونه‌های مثبت و انتخاب نمونه‌های منفی برای آن‌ها ابتدا باتوجه به مقاله ارائه شده یک تابع برای انتخاب تصادفی نمونه‌های منفی می‌نویسیم. باتوجه به توضیحات ارائه‌شده در این مقاله، در تکنیک Negative Sampling به صورت تصادفی با توضیح یکنواخت انجام نمی‌شود. بلکه بر اساس unigram که بر اساس یک آلفا Smooth شده انتخاب می‌شوند. پس احتمال انتخاب نمونه‌های منفی به صورت زیر است:

Choosing noise words

Could pick w according to their unigram frequency P(w)

More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

α= ¾ works well because it gives rare noise words slightly higher probability

```
# Negative Sampling
from collections import Counter
word_frequencies = Counter(words)
total_word_count = sum(word_frequencies.values())
```

```
word_frequencies
```

```
Counter({'adventures': 6,
         'sherlock': 97,
         'holmes': 462,
         'arthur': 21,
```

پس ابتدا unigram بدست آمده را برای تشکیل توزیع احتمال انتخاب نمونه‌های منفی بدست آورده و سپس تابعی برای این کار پیاده‌سازی می‌کنیم. ابتدا فراوانی‌ها تقسیم بر کل کلمات شده تا احتمال هر کلمه محاسبه شده و سپس در فرمول بالا قرار داده می‌شود. آلفای برابر با ۴/۳ باعث می‌شود که کلماتی که احتمال خیلی کم دارند، شانس انتخاب شدن بیشتری داشته باشند.

```
# Negative Sampling Distribution
from collections import Counter
word_frequencies = Counter(words)
total_word_count = sum(word_frequencies.values())
word_frequencies = {word: count/total_word_count for word, count in word_frequencies.items()}

negative_sampling_distribution = np.array(list(word_frequencies.values()))**0.75
negative_sampling_distribution /= negative_sampling_distribution.sum()
```

سپس تابعی برای انتخاب تصادفی نوشته شده که در آن شرط اینکه کلمه تصادفی انتخاب شده، خود کلمه نباشد نیز رعایت شده است.

سپس یک ماتریس تشکیل داده که برای هر جفت کلمه نوع مثبت یا منفی آن نیز مشخص شده است. این ایده از سایت زیر استفاده شد ولی پیاده‌سازی آن را از جایی کپی نشده است.:

https://jalammar.github.io/illustrated-word2vec

در این ماتریس train، برای هر جفت کلمه اگر که مثبت باشد ستون target برابر ۱ و اگر منفی باشد ستون target برابر صفر است. طبق سوال برای هر مثبت ۴ نمونه منفی اضافه می‌شود:

```
41] #Generate Positive and Negative Samples
    training_data = []
    for target, context in positive_pairs:
        negative_samples = generate_negative_samples(target_index=target, num_negative_samples=4)

        training_sample = (target, context, 1)  #target 1 for positive
        training_data.append(training_sample)

        for neg_sample in negative_samples:
            training_data.append((target, neg_sample, 0))  #target 0 for negative
```

```
training_data
```

```
[(5111, 2030, 1),
 (5111, 367, 0),
 (5111, 3150, 0),
 (5111, 3262, 0),
 (5111, 4445, 0),
 (5111, 3960, 1),
 (5111, 4411, 0),
 (5111, 1778, 0),
 (5111, 2258, 0),
 (5111, 307, 0),
```

حال باید مدل را آموزش دهیم تا خروجی تابع سیگموید ضرب داخلی برای نمونه‌های مثبت به ۱ نزدیک شده و نمونه‌های منفی به ۰ نزدیک شود.

برای آموزش از یک کد آماده کمک گرفته شد که در یک حلقه به تعداد Epoch ها مدل آموزش می‌بیند. در این حلقه از دو ماتریس embedding و context برای وزن‌ها استفاده شده است. در هر تکرار در حلقه داخلی یک نمونه انتخاب شده و بر اساس اینکه لیبل آن ۰ یا ۱ است سعی می‌شود وزن‌ها به شکلی تغییر کنند که کلاس‌بندی به خوبی انجام شود.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

learning_rate = 0.01
epochs = 1

for epoch in range(epochs):
    total_loss = 0
    for target, context, label in training_data:
        v_target = W_embedding[target]
        v_context = W_context[context]
        score = np.dot(v_target, v_context)
        predicted_label = sigmoid(score)

        predicted_label = np.clip(predicted_label, 1e-15, 1 - 1e-15)

        g = predicted_label - label

        v_target_update = g * v_context
        v_context_update = g * v_target

        W_embedding[target] -= learning_rate * v_target_update
        W_context[context] -= learning_rate * v_context_update

        loss = -((label * np.log(predicted_label)) + ((1 - label) * np.log(1 - predicted_label)))
        total_loss += loss

    print(f"Epoch {epoch + 1}/{epochs}, Loss: {total_loss/len(training_data)}")

word_vectors = W_embedding + W_context
```

۲-۲_ پاسخ بخش دوم

برای سنجش میزان شباهت تو کلمه یک تابع نوشته شد که فاصله کسینوسی بین دو بردار را محاسبه می‌کند. باتوجه به مدل skip-gram انتظار داریم فاصله ای که king از man دارد به اندازه فاصله queen از woman باشد. به همین دلیل وقتی که فاصله بین این دو را با woman جمع کنیم انتظار داریم بردار بدست آمده نزدیک به بردار queen باشد.

```
def similarity(vec_a, vec_b):
    cosine_sim = np.dot(vec_a, vec_b) / (np.linalg.norm(vec_a) * np.linalg.norm(vec_b))
    return cosine_sim
```

دفعات زیادی این مورد اجرا شد. نتیجه به مقدار بالایی نرسید. بهترین خروجی برابر با ۰.۱۶ بود:

```
0.16308110599029624
```

پیاده‌سازی های مختلفی انجام شد و با تغییر اندازه پنجره و یا بیشترشدن ابعاد نیز مشکل حل نشد و عدد مناسبی بدست نیامد.

در مرحله بعد کلمات دیگر را تست کردیم تا متوجه شویم آیا مدل به خوبی کار می‌کند یا خیر. برای نمونه فاصله دو کلمه sherlock و holmes در ابتدا حدود ۰ بود و با چندبار تکرار مرحله train به دقت بالاتری رسید. ولی مقدار مورد نظر سوال تغییر خاصی نکرد. هرچند که مدل به شکل‌های مختلفی پیاده‌سازی شد.

```
word1 = word_vectors[word_to_index.get('sherlock')]
word2 = word_vectors[word_to_index.get('holmes')]

similarity(word1, word2)
```

```
0.38149930782704733
```

یکی از دلایل می‌تواند این باشد که king و queen زیاد با هم نیامده‌اند و یا در جایگاه‌های مشابه هم استفاده نشده است.

یک مورد دیگر هم تست شد:

```
def similarity(vec_a, vec_b):
    cosine_sim = np.dot(vec_a, vec_b) / (np.linalg.norm(vec_a) * np.linalg.norm(vec_b))
    return cosine_sim

king_vector = word_vectors[word_to_index.get('brother')]
man_vector = word_vectors[word_to_index.get('man')]
woman_vector = word_vectors[word_to_index.get('woman')]

word1 = word_vectors[word_to_index.get('sister')]

word2 = king_vector - man_vector + woman_vector

similarity(word1, word2)
```

```
0.06249311073822624
```

در این مورد Brother - man + woman با sister مقایسه شد. با فقط ۱ بار اجرای ترین از ۰ به حدود ۰.۲۰ رسید که بسیار بهتر است. دلیل این مورد احتمالا این است که این مورد خیلی بیشتر از king و queen در corpus تکرار شده است. با یک بار اجرای دوباره train به مقادیر زیر رسید:

```
def similarity(vec_a, vec_b):
    cosine_sim = np.dot(vec_a, vec_b) / (np.linalg.norm(vec_a) * np.linalg.norm(vec_b))
    return cosine_sim

king_vector = word_vectors[word_to_index.get('brother')]
man_vector = word_vectors[word_to_index.get('man')]
woman_vector = word_vectors[word_to_index.get('woman')]

word1 = word_vectors[word_to_index.get('sister')]

word2 = king_vector - man_vector + woman_vector

similarity(word1, word2)
```

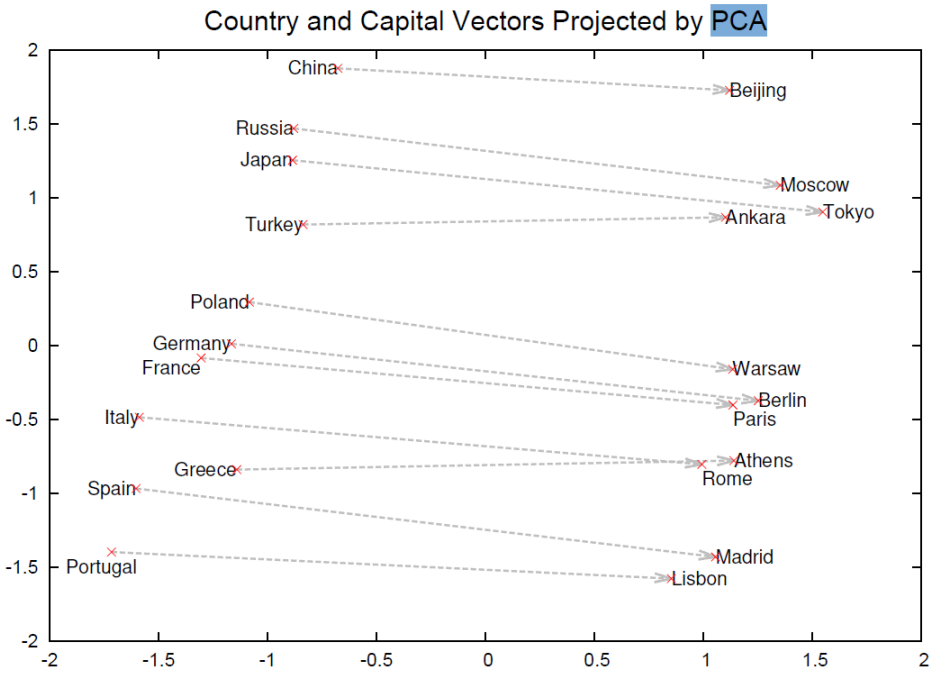
```
0.20672679823683313
```

پس مدل تا حدودی کار می‌کند ولی برای نمونه‌ای که در سوال مطرح شده پاسخ خوبی ارائه نمی‌دهد که دلیل آن ذکر شد.

۳-۲_ پاسخ قسمت سوم

در این قسمت از مسئله خواسته شده که کاهش ابعاد داشته باشیم و ابعاد از ۱۰۰ به ۲ کاهش پیدا کند. برای این کار از روش PCA استفاده می‌شود.

در مقاله به طور دقیق به روش خاصی ارائه شده و به طور کلی گفته شده که ابعاد از ۱۰۰۰ به ۲ کاهش پیدا کرده و نتیجه آن به صورت زیر بوده است.



مشاهده می‌شود که در تصویر بالا، بدون آنکه به مدل یاد داده شود که پایتخت چیست کاملا مشخص شده که پایتخت ها یک سمت و کشورها سمت دیگر هستند و بردارها موازی هستند.حال این مورد نیز برای دو بردار brother-sister و uncle-aunt بررسی می‌کنیم. باتوجه به اینکه این مورد نیز مانند مورد بالاست و هردو رابطه‌هایی هستند که اولی از جنس مذکر و دومی مونث است انتظار داریم بردار آن‌ها موازی باشد. در چند اجرا بردارها موازی نبودند زیرا که PCA همیشه یک طور بردارها را کاهش نمیدهد. ولی نتیجه‌ای که مطلوب بود به صورت زیر است: