Consine Similarity

- 두 벡터의 크기와 상관없이, 상호 방향성이 얼마나 유사한지에 기반
  - 두 벡터의 사잇각이을 계산하여 유사도 측정
- 문서(문장)의 크기를 고려하지 않는 빈도수 기반의 단점 보완

```
import warnings
warnings.filterwarnings('ignore')
```

I. TF-IDF Vectorization

1) 문장 3개 지정

```
doc_list = ['if you take the blue pill, the story ends' ,
            'if you take the red pill, you stay in Wonderland',
            'if you take the red pill, I show you how deep the rabbit hole goes']
```

2) Coordinate(좌표) 양식

- '0'이 아닌 데이터포이트의 좌표만 저장

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vec_simple = TfidfVectorizer()
feature_vec_simple = tfidf_vec_simple.fit_transform(doc_list)

print(feature_vec_simple)
```

```
  (0, 2)        0.41556360057939173
  (0, 13)       0.41556360057939173
  (0, 8)        0.24543855687841593
  (0, 0)        0.41556360057939173
  (0, 15)       0.49087711375683185
  (0, 14)       0.24543855687841593
  (0, 17)       0.24543855687841593
  (0, 6)        0.24543855687841593
  (1, 16)       0.39624495215024286
  (1, 7)        0.39624495215024286
  (1, 12)       0.39624495215024286
  (1, 10)       0.3013544995034864
  (1, 8)        0.2340286519091622
  (1, 15)       0.2340286519091622
  (1, 14)       0.2340286519091622
  (1, 17)       0.4680573038183244
```

```
(1, 6)        0.2340286519091622
(2, 3)        0.3098560092999078
(2, 4)        0.3098560092999078
(2, 9)        0.3098560092999078
(2, 1)        0.3098560092999078
(2, 5)        0.3098560092999078
(2, 11)       0.3098560092999078
(2, 10)       0.23565348175165166
(2, 8)        0.1830059506093466
(2, 15)       0.3660119012186932
(2, 14)       0.1830059506093466
(2, 17)       0.3660119012186932
(2, 6)        0.1830059506093466
```

3) (밀집)행렬 변환

```
feature_vec_dense = feature_vec_simple.todense()

print(feature_vec_dense)
```

```
[[0.4155636  0.         0.4155636  0.         0.         0.
  0.24543856 0.         0.24543856 0.         0.         0.
  0.         0.4155636  0.24543856 0.49087711 0.         0.24543856]
 [0.         0.         0.         0.         0.         0.
  0.23402865 0.39624495 0.23402865 0.         0.3013545  0.
  0.39624495 0.         0.23402865 0.23402865 0.39624495 0.4680573 ]
 [0.         0.30985601 0.         0.30985601 0.30985601 0.30985601
  0.18300595 0.         0.18300595 0.30985601 0.23565348 0.30985601
  0.         0.         0.18300595 0.3660119  0.         0.3660119 ]]
```

4) 개별 Feature Vector 추출

```
import numpy as np

vec1 = np.array(feature_vec_dense[0]).reshape(-1,)
vec2 = np.array(feature_vec_dense[1]).reshape(-1,)
vec3 = np.array(feature_vec_dense[2]).reshape(-1,)

vec1, vec2, vec3
```

```
(array([0.4155636 , 0.        , 0.4155636 , 0.        , 0.        ,
        0.        , 0.24543856, 0.        , 0.24543856, 0.        ,
        0.        , 0.        , 0.        , 0.4155636 , 0.24543856,
        0.49087711, 0.        , 0.24543856]),
 array([0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.23402865, 0.39624495, 0.23402865, 0.        ,
        0.3013545 , 0.        , 0.39624495, 0.        , 0.23402865,
        0.23402865, 0.39624495, 0.4680573 ]),
 array([0.        , 0.30985601, 0.        , 0.30985601, 0.30985601,
        0.30985601, 0.18300595, 0.        , 0.18300595, 0.30985601,
        0.23565348, 0.30985601, 0.        , 0.        , 0.18300595,
        0.3660119 , 0.        , 0.3660119 ]))
```

II. cos_similarity( )

- 두 벡터의 내적을 총 벡터 크기로 정규화(L2 Norm)
  - dot_product : 두 벡터의 내적
  - l2_norm : 총 벡터 크기의 합

```
import numpy as np

def cos_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    l2_norm = (np.sqrt(sum(np.square(v1))) * np.sqrt(sum(np.square(v2))))
    similarity = dot_product / l2_norm

    return similarity
```

1) 'vec1', 'vec2' 코사인 유사도

```
similarity_simple = cos_similarity(vec1, vec2)

print('vec1, vec2 코사인 유사도: {0:.5f}'.format(similarity_simple))
```

```
vec1, vec2 코사인 유사도: 0.40208
```

2) 'vec1', 'vec3' 코사인 유사도

```
similarity_simple = cos_similarity(vec1, vec3)

print('vec1, vec3 코사인 유사도: {0:.5f}'.format(similarity_simple))
```

```
vec1, vec3 코사인 유사도: 0.40425
```

3) 'vec2', 'vec3' 코사인 유사도

```
similarity_simple = cos_similarity(vec2, vec3)

print('vec2, vec3 코사인 유사도: {0:.5f}'.format(similarity_simple))
```

```
vec2, vec3 코사인 유사도: 0.45647
```

III. sklearn - cosine_similarity( )

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_similarity(feature_vec_simple[0], feature_vec_simple)
```

```
array([[1.        , 0.40207758, 0.40425045]])
```

```
cosine_similarity(feature_vec_simple[0], feature_vec_simple[1:])
```

```
array([[0.40207758, 0.40425045]])
```

```
cosine_similarity(feature_vec_simple, feature_vec_simple)
```

```
array([[1.        , 0.40207758, 0.40425045],
       [0.40207758, 1.        , 0.45647296],
       [0.40425045, 0.45647296, 1.        ]])
```

IV. Topic Problem

1) 문장 지정

```
sent_list = ['I eat an apple' ,
             'Koo have fruit',
             'I sell an apple']
```

2) 벡터 변환

```
tfidf_vec = TfidfVectorizer()
feature_vec = tfidf_vec.fit_transform(sent_list)
```

3) 문장1 vs. 문장2

```
cosine_similarity(feature_vec[0], feature_vec[1])
```

```
array([[0.]])
```

4) 문자1 vs. 문장3

```
cosine_similarity(feature_vec[0], feature_vec[2])
```

```
array([[0.53634991]])
```

V. Word2Vec

```
# Load Pretrained Word2Vec
import tensorflow_hub as hub

embed = hub.load('https://tfhub.dev/google/Wiki-words-250/2')
```

```
words = ['apple', 'eat', 'fruit', 'have', 'sell']
```

```
embeddings = embed(words)
```

```
import numpy as np

for i in range(len(words)):
    for j in range(i,len(words)):
        print("(",words[i], ",", words[j],")",np.inner(embeddings[i], embeddings[j]))
```

```
( apple , apple ) 1.0
( apple , eat ) 0.48909307
( apple , fruit ) 0.78753763
( apple , have ) 0.13348329
( apple , sell ) 0.106232405
( eat , eat ) 1.0
( eat , fruit ) 0.53294003
( eat , have ) 0.3232242
( eat , sell ) 0.2691978
( fruit , fruit ) 1.0
( fruit , have ) 0.13598028
( fruit , sell ) 0.11212408
( have , have ) 1.0
( have , sell ) 0.21071003
( sell , sell ) 1.0
```

#

#

#

The End

#

#

#