단어의 표현 (Word Representation)

기계는 문자를 그대로 인식할 수 없기때문에 숫자로 변환

1 원-핫 인코딩 (One-Hot Encoding)

1.1 직접 구현해보기

"원숭이, 바나나, 사과" 로 원-핫 인코딩을 한다면

```
# #인코딩 대상 단어들을 담은 리스트 word_ls =
['원숭이','바나나','사과','사과']
```

```
from collections import defaultdict
import numpy as np

def one_hot_encode(word_ls):
    # 고유 단어와 인덱스를 매칭시켜주는 사전 생성 word2id_dic =
    defaultdict(lambda:len(word2id_dic))

    # {단어 : 인덱스} 사전 구축 for
    word in word_ls:
        word2id_dic[word]

    n_unique_words = len(word2id_dic) # 고유한 단어의 갯수 one_hot_vectors = np.zeros((len(word_ls),
    n_unique_words)) # 원핫-벡터를 만들기 위해 비어있는 벡

    for i,word in enumerate(word_ls):
        index = word2id_dic[word] # 해당 단어의 고유 인덱스 one_hot_vectors[i,
        index] = 1 # 해당 단어의 고유 인덱스에만 1 을 더해줌
```

```
    return one_hot_vectors


one_hot_vectors ○= one_hot_encode(word_ls)
one_hot_vectors
```

"코끼리"라는 단어가

추가된다면?

```
word_ls w= ['원숭이','바나나','사과','코끼리']

one_hot_vectors ○= one_hot_encode(word_ls)
one_hot_vectors
```

1.3 sklearn 활용

함수명	설명
t(X[, y])	Fit OneHotEncoder to X.
t_transform(X[, y])	Fit OneHotEncoder to X, then transform X.
inverse_transform(X)	Convert the back data to the original representation.
transform(X)	Transform X using one-hot encoding.

```
# #sklearn 을 활용한 one-hot encoding from numpy
import array from numpy import argmax from
sklearn.preprocessing import LabelEncoder from
sklearn.preprocessing import OneHotEncoder

# 예제 데이터 배열 values =
array(word_ls)
print(values)

# 문자열에 숫자를 붙임 label_enc = LabelEncoder()
```

```
int_enc = label_enc.fit_transform(values)
print(int_enc)

# binary encode onehot_enc = OneHotEncoder(sparse=False)
int_enc = int_enc.reshape(len(int_enc), 1) # n:1 matrix 로 변환
print(int_enc) onehot_enc = onehot_enc.fit_transform(int_enc)
print(onehot_enc)

# one-hot encoding 의 첫번째 배열을 값을 역으로 산출 inverted =
label_enc.inverse_transform([argmax(onehot_enc[0, :])])
print(inverted)


onehot_enc[0, o:] argmax(onehot_enc[0, a:])

label_enc.inverse_transform([argmax(onehot_enc[0, :])])
```

2 밀집 벡터 (Dense Vector)

```
word_embedding_dic w= {
         '사과' : [1.0, 0.5],
        '바나나' : [0.9, 1.2],
   '원숭이' : [0.5, 1.5] }
```

2-1

유사도 계산

2.1.1 유클리디언 거리(Euclidean distance)

두 벡터사이의 직선 거리. 피타고라스 정리를 생각하면 이해하기 쉬움

https://en.wikipedia.org/wiki/Euclidean_distance

```
import numpy as np def euclidean_dist(x,y):
x = np.array(x) y = np.array(y) return np.sqrt(np.sum(x-y)**2)
# 사과와 바나나의 유클리디안 유사도 euclidean_dist(word_embedding_dic['사과'],
word_embedding_dic['바나나'])
```

https://en.wikipedia.org/wiki/Cosine_similarity

2.1.2 코사인 유사도(Cosine Similarity)

두 벡터간의 유사도를 측정하는 방법 중 하나 두 벡터 사이의 코사인을 측정 0 도 = 1, 90 도 = 0, 180 도 = -1

==> 1 에 가까울수록 유사도가 높음

```
def dcosine_similarity(x, y):
# x 와 y, 두 벡터의 코사인 유사도를 계산하는 함수 nominator = np.dot(x, y) # 분자 denominator =
np.linalg.norm(x)*np.linalg.norm(y) # 분모 return nominator/denominator
a a
= np.array([1, 2]) b = np.array([3, 4]) np.dot(a, b)
```


numpy 의 linalg 서브 패키지의 norm 명령으로 벡터의 길이를 계산할 수 있다. 위에서 예로 든 2 차원 벡터 a=[1,2] 의 길이는 √5≈2.236 이다.

더블클릭 또는 Enter 키를 눌러 수정

```
a a= np.array([1, 2])
np.linalg.norm(a)
```

```
# # 사과와 바나나의 코사인 유사도 print(cosine_similarity(word_embedding_dic['사과'],
word_embedding_dic['바나나'])) print(euclidean_dist(word_embedding_dic['사과'],
word_embedding_dic['바나나']))
```

```
# 사과와 원숭이의 코사인 유사도 print(cosine_similarity(word_embedding_dic['사과'],
word_embedding_dic['원숭이'])) print(euclidean_dist(word_embedding_dic['사과'],
word_embedding_dic['원숭이']))
```

```
# #바나나와 원숭이의 코사인 유사도 print(cosine_similarity(word_embedding_dic['바나나'],
word_embedding_dic['원숭이'])) print(euclidean_dist(word_embedding_dic['바나나'],
word_embedding_dic['원숭이']))
```

2.1.3 자카드 유사도(Jaccard index)

https://en.wikipedia.org/wiki/Jaccard_index

```
s1 = '대부분 원숭이는 바나나를 좋아합니다.' s2 =
'코주부 원숭이는 바나나를 싫어합니다.'

# 토큰화를 수행합니다.
token_s1 = s1.split()
token_s2 = s2.split()

union = set(token_s1).union(set(token_s2))
print(union)

intersection = set(token_s1).intersection(set(token_s2))
print(intersection)

print(len(intersection)/len(union)) # 2 를 6 로 나눔.
```

2.1.4 레빈슈타인 거리

더블클릭 또는 Enter 키를 눌러 수정

idf(inverse document frequency smooth) =log(문서갯수/(1+토큰빈도))

```
d1 d= "The cat sat on my face I hate a cat"
d2 = "The dog sat on my bed I love a dog"
```

3 TF-IDF 를 활용한 단어 벡터

```
from math import log10
```

3.1 직접 구현하기

```
# document 내 토큰이 등장한 빈도수 계산 def
f(t, d):
    return d.count(t)
```

weighting schema weight 설명

```
# tf 계산 def
tf(t, d):
```

tf(double normalization 0.5) =0.5 + 0.5(토큰빈도/문서내최빈토큰)

```
    return 0.5 + 0.5*f(t,d)/max([f(w,d) for w in d])
```

```
# idf 계산 def
idf(t, D):
    numerator = len(D) denominator = 1 + len([ True for d
    in D if t in d]) return log10(numerator/denominator) +
    1


# tf-idf 계산 def
tfidf(t, d, D):
    #print(D) print(t)
    #print(d) print(tf(t,d))
    print(idf(t, D))
    print(tf(t,d)*idf(t, D))
    print("===") return
    tf(t,d)*idf(t, D)


# 공백을 기준으로 토큰과 def
tokenizer(d):
    return d.split()


        # tfidf 계산
```

https://colab.research.google.com/drive/1Ee0CfnzULsnWhiF4sfWNXpsxoA0PmBpu#printMode=true d f
tfidfS (D)


```
    def tfidfScorer(D):
        tokenized_D = [tokenizer(d) for d in D]
        result = [] for d in tokenized_D:
            result.append([(t, tfidf(t, d, tokenized_D)) for t in d])
        return result


    corpus = [d1, d2]


    for i, doc in enumerate(tfidfScorer(corpus)):
        print('====== document[%d] ======' % i)
        print(doc)
```

3.2 sklearn 활용


```
from sklearn.feature_extraction.text import CountVector


corpus = [

'you know I want your love', 'I like you', 'what should

"The cat sat on my face I hate a cat" d2 = "The dog s
love a dog" corpus = [d1, d2] count_vect = CountVectori
count_vect.fit_transform(corpus) print(countv.toarray()
단어의 빈도 수를 기록한다. print(count_vect.vocabulary_) # 각 단어
부여되었는지를 보여준다.


from sklearn.feature_extraction.text import TfidfVector
corpus = [
'you know I want your love', 'I like you',
'what should I do ', ] d1 = "The cat sat on

my face I hate a cat" d2 = "The dog sat on
my bed I love a dog" corpus = [d1, d2]
tfidf_vect = TfidfVectorizer().fit(corpus)
tfidfv = tfidf_vect.transform(corpus)
print(tfidfv.toarray())
print(tfidf_vect.vocabulary_)
```

3.3 gensim 활용

```
import gensim.downloader as api from
gensim.models import TfidfModel from
gensim import corpora


corpus = [
            'you know I want your love',
```

https://colab.research.google.com/drive/1Ee0CfnzULsnWhiF4sfWNXpsxoA0PmBpu#printMode=true ' i


```
'I like you', 'what should I do ', ] doc_ls = [doc.split() for doc in corpus] id2word =
corpora.Dictionary(doc_ls) # fit dictionary corpus = [id2word.doc2bow(doc) for doc in doc_ls] # con
corpus to BoW format


tfidf = TfidfModel(corpus) # fit model vector = tfidf[corpus[0]] # apply model to the first
corpus document
```

```
tfidf[corpus][0] tid2word.keys()
```

```
TfidfModel? T
```

4 LSA(Latent Semantic Analysis)를 활용한 단어 벡터

4.1 sklearn 활용

```
doc_ls d= ['바나나 사과 포도 포도',
          '사과 포도', '포도 바나나', '짜장면 짬뽕 탕수육', '볶음밥
          탕수육', '짜장면 짬뽕', '라면 스시', '스시', '가츠동 스시
          소바', '된장찌개 김치찌개 김치', '김치 된장', '비빔밥 김치' ]
```

```
from sklearn.feature_extraction.text import CountVectorizer from
sklearn.decomposition import TruncatedSVD

count_vect = CountVectorizer() countv = count_vect.fit_transform(doc_ls) svd =
TruncatedSVD(n_components=2, algorithm='randomized', n_iter=100) svd.fit(countv)

features = count_vect.get_feature_names() # 단어 집합. 1,000 개의 단어가 저장됨. for i in
range(len(features)) :
  print("{} : {}".format(features[i], svd.components_[:,i]))
```

4.2 gensim 활용


```
doc_ls d= ['바나나 사과 포도 포도',
  '사과 포도', '포도 바나나', '짜장면 짬뽕 탕수육', '볶음밥 탕수육', '짜장면 짬뽕',
   '라면 스시', '스시', '가츠동 스시 소바', '된장찌개 김치찌개 김치', '김치 된장',
        '비빔밥 김치' ] doc_ls = [d.split() for d in doc_ls]
```

```
from gensim import corpora from gensim.models import LsiModel

id2word = corpora.Dictionary(doc_ls) #사전 구축 corpus = [id2word.doc2bow(text) for text in
doc_ls] # 코퍼스 생성 lsi = LsiModel(corpus, id2word=id2word, num_topics=2) #LSA 모델

for i in id2word.keys() :
  print("{} : {}".format(id2word[i], lsi.projection.u[i]))
```