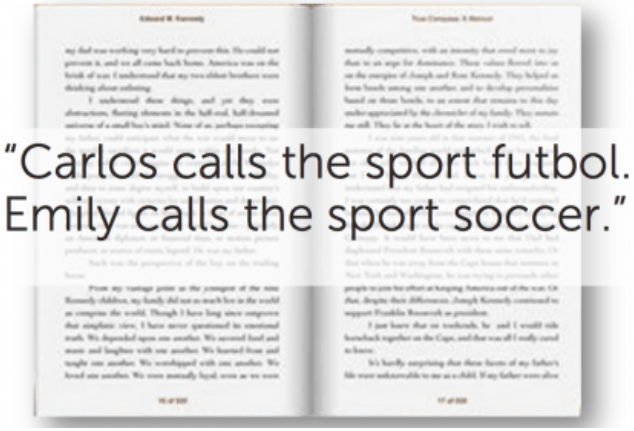
텍스트 유사도

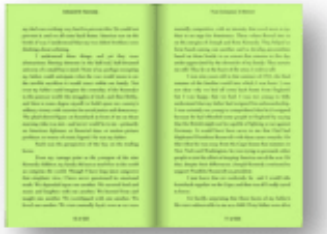

ІНСТИТУТ МІСТА
ГТЦ №OEZ121
УГОДИ №26-52
ГТЦ № OEZ 250
УГОДИ №1-2
ГТЦ № OEZ 260
OEZ 331
OEZ 409 (2017)
ГТЦ №OEZ131
УГОДИ №1-25
Г.МАРШАЛА
2013 Р.
ГТЦ № OEZ 277
УГОДИ №45,
47-70
ГТЦ №OEZ172
УГОДИ №68-90
ГТЦ №OEZ172
УГОДИ №91-108
OEZ 314
OEZ 316
OEZ 285
OEZ 280
IV ФМС-2016
V ФМС-2017
OEZ 338 (ММ)
OEZ443-445.ДА (ММ)
OEZ 342 (ММ)
Соціологічні
Підзамче
МЕРМ
МАТРА

박경규

유사도 측정



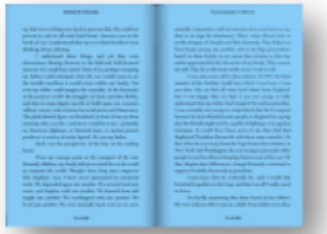
"Carlos calls the sport futbol.
Emily calls the sport soccer."



1 0 0 0 5 3 0 0 1 0 0 0 0

1*3
+
5*2
= 13

3 0 0 0 2 0 0 1 0 1 0 0 0

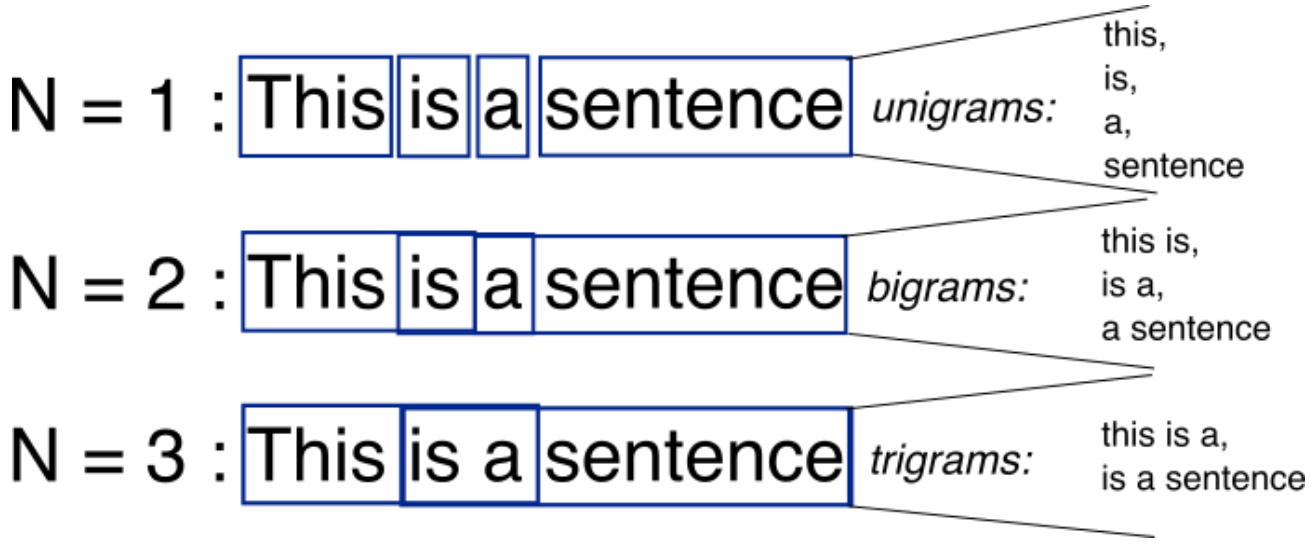


N-gram 모델

언어 모델(Language Model) 에서 사용되는 횟수 기반의 벡터 표현방식(Count-based representation) 입니다.

N-gram

- N=1 : 1-gram(unigram)
- N=2 : 2-gram(bigram)
- N=3: 3-gram(trigram)
- N=4 : 4-gram



$$similarity = \frac{tf(A,B)}{tokens(A)}$$

N-grams 단점

- N-gram은 장기 의존성(Long-term dependency) 이 없습니다. N-gram에서는 일부 단어 시퀀스의 횟수만을 가지고 판단하기 때문에 문장 앞쪽의 문맥을 고려하지 않은 채 토큰을 선택하게 됩니다.

N-gram 모델

text_similarity.ipynb

```
# 어절 단위 n-gram
def word_ngram(bow, num_gram):
    text = tuple(bow)
    ngrams = [text[x:x + num_gram] for x in range(0, len(text))]
    return tuple(ngrams)


# 음절 n-gram 분석
def phoneme_ngram(bow, num_gram):
    sentence = ' '.join(bow)
    text = tuple(sentence)
    slen = len(text)
    ngrams = [text[x:x + num_gram] for x in range(0, slen)]
    return ngrams
```

N-gram 모델

```
# 유사도 계산
def similarity(doc1, doc2):
    cnt = 0
    for token in doc1:
        if token in doc2:
            cnt = cnt + 1

    return cnt/len(doc1)
```

$$similarity = \frac{tf(A,B)}{tokens(A)}$$

코사인 유사도

단어/문자을 벡터로 표현하면, 벡터간 거리나 각도를 이용해 유사성을 파악할 수 있다.

- 출현빈도로 유사도를 계산하면 동일한 단어가 많이 포함 될수록 벡터의 크기가 커집니다.
- 코사인 유사도는 벡터의 크기와 상관없이 결과 값이 안정적입니다.

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

```
def cos_sim(vec1, vec2):
    return dot(vec1, vec2) / (norm(vec1) * norm(vec2))
```

코사인 유사도

```
# TDM 만들기
def make_term_doc_mat(sentence_bow, word_dics):
    freq_mat = {}
    for word in word_dics:
        freq_mat[word] = 0
    for word in word_dics:
        if word in sentence_bow:
            freq_mat[word] += 1
    return freq_mat


# 단어 벡터 만들기
def make_vector(tdm):
    vec = []
    for key in tdm:
        vec.append(tdm[key])
    return vec
```

코사인 유사도

```
# 단어 묶음 리스트를 하나로 합침
bow = bow1 + bow2 + bow3

# 단어 묶음에서 중복제거해 단어 사전 구축
word_dics = []
for token in bow:
    if token not in word_dics:
        word_dics.append(token)

# 문장 별 단어 문서 행렬 계산
freq_list1 = make_term_doc_mat(bow1, word_dics)
freq_list2 = make_term_doc_mat(bow2, word_dics)
freq_list3 = make_term_doc_mat(bow3, word_dics)

# 코사인 유사도 계산
doc1 = np.array(make_vector(freq_list1))
doc2 = np.array(make_vector(freq_list2))
doc3 = np.array(make_vector(freq_list3))
r1 = cos_sim(doc1, doc2)
r2 = cos_sim(doc3, doc1)
print(r1, r2)
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

sentence = ( "휴일 인 오늘 도 서쪽 을 중심 으로 폭염 이 이어졌는데요, 내일 은 반가운 비 소
식 이 있습니다.", "폭염 을 피해서 휴일 에 놀러왔다가 갑작스런 비 로 인해 망연자실 하고 있습니다.")


tfidf_vectorizer = TfidfVectorizer()

# 문장 벡터화
tfidf_matrix = tfidf_vectorizer.fit_transform(sentence)

# 각 단어
text = tfidf_vectorizer.get_feature_names()

# 각 단어의 벡터 값
idf = tfidf_vectorizer.idf_

print(dict(zip(text, idf)))
```

{'갑작스런': 1.4054651081081644, '내일': 1.4054651081081644, '놀러왔다가': 1.4054651081081644, '망연자실': 1.4054651081081644, '반가운': 1.4054651081081644, '서쪽': 1.4054651081081644, '소식': 1.4054651081081644, '오늘': 1.4054651081081644, '으로': 1.4054651081081644, '이어졌는데요': 1.4054651081081644, '인해': 1.4054651081081644, '있습니다': 1.0, '중심': 1.4054651081081644, '폭염': 1.0, '피해서': 1.4054651081081644, '하고': 1.4054651081081644, '휴일': 1.0}

https://soyoung-new-challenge.tistory.com/34

자카드 유사도(Jaccard Similarity)

두 문장을 각각 단어의 집합으로 만든 뒤 두 집합을 통해 유사도를 측정하는 방식 중 하나

- 유사도를 측정하는 방법은 두 집합의 교집합인 공통된 단어의 개수를 두 집합의 합집합, 전체 단어의 갯수로 나눈다.
- 결과값은 공통의 원소의 개수에 따라 0과 1사이의 값이 나올 것이고, 1에 가까울수록 유사도가 높다

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

https://soyoung-new-challenge.tistory.com/34

코사인 유사도

코사인 유사도는 사이킷런에서 유사도 측정을 위한 함수를 제공합니다.

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])
```

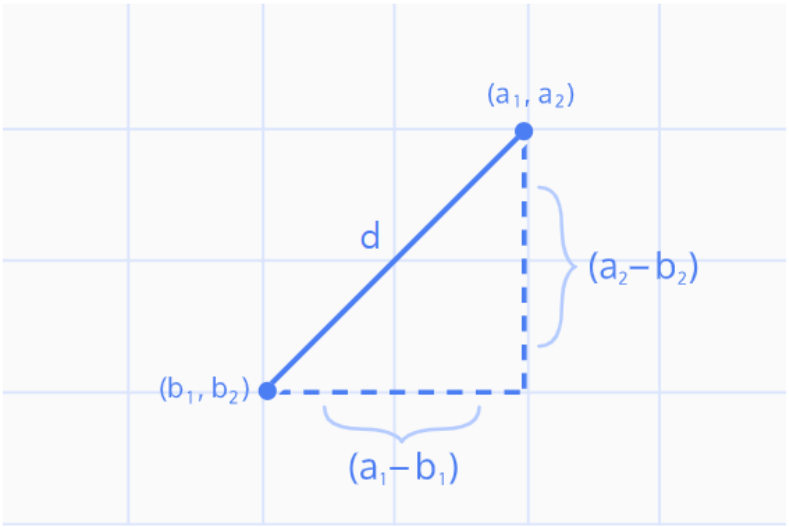
첫 번째 문장 두 번째 문장

https://soyoung-new-challenge.tistory.com/34

유클리디언 유사도 (L2 Distance)

가장 기본적인 거리를 측정하는 유사도 공식입니다.
유클리디언 거리 = L2 거리 : N차원 공간에서 두 점 사이의 최단 거리를 구하는 접근법

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$



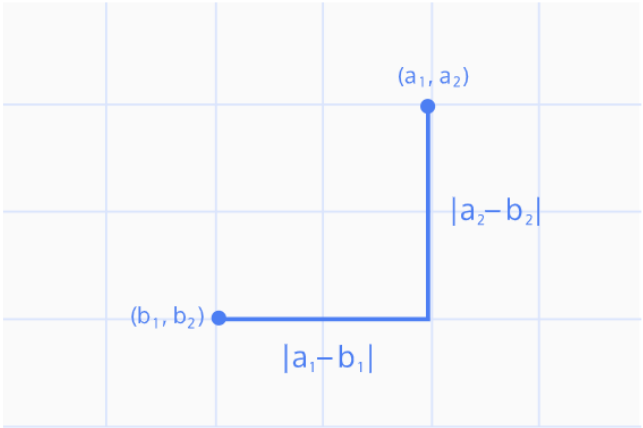

```
from sklearn.metrics.pairwise import euclidean_distances

euclidean_distances(tfidf_matrix[0:1], tfidf_matrix[1:2])
```

https://soyoung-new-challenge.tistory.com/34

맨하탄 유사도 (L1 Distance)

맨하탄 거리를 통해 유사도를 측정하는 방법으로 , 맨하탄 거리 = L1 거리 입니다.

$$\sum_{i=1}^{k} |x_i - y_i|$$



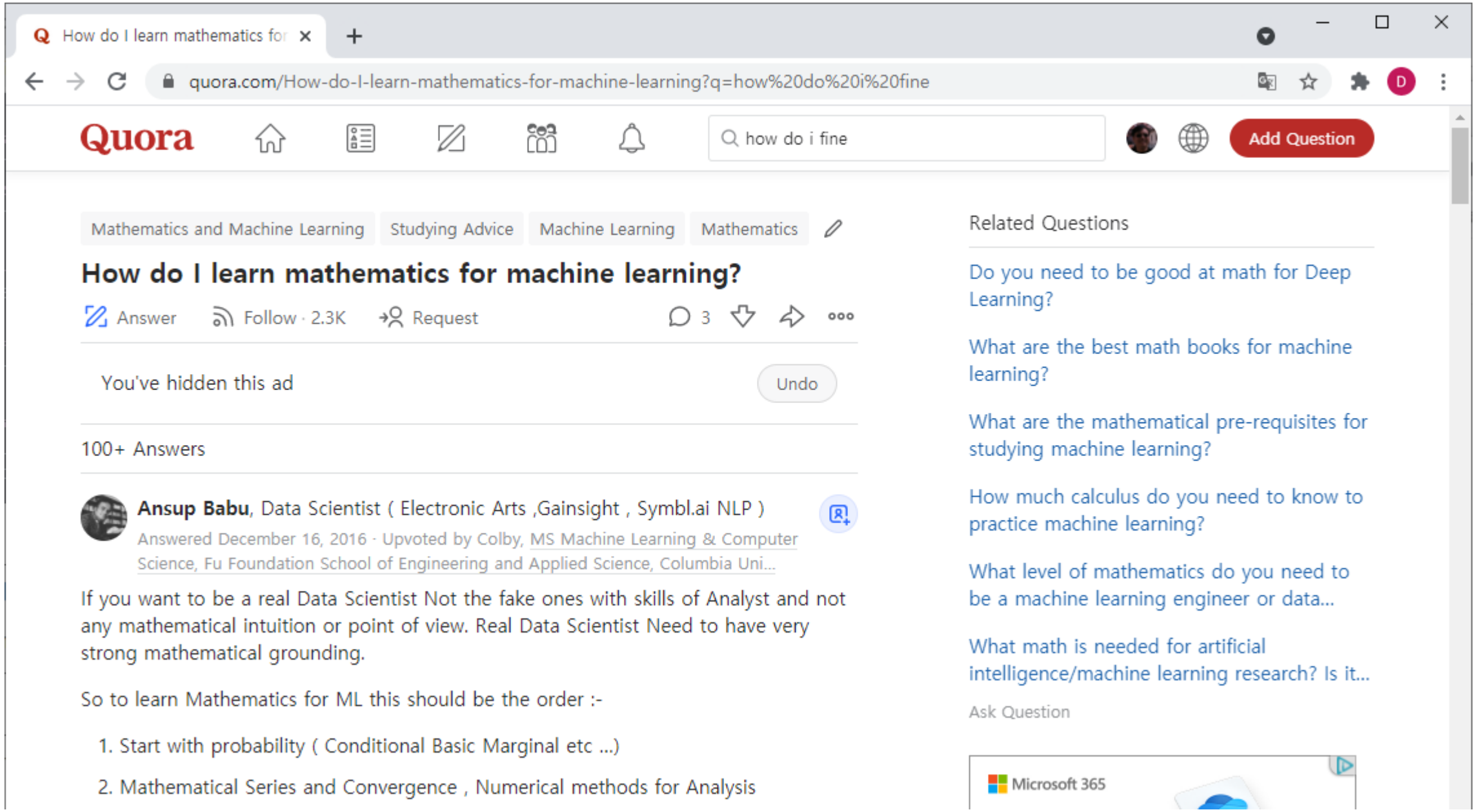

```
import numpy as np
from sklearn.metrics.pairwise import manhattan_distances

def l1_normalize(v):
    norm = np.sum(v)
    return v / norm

# L1 정규화
tfidf_norm_l1 = l1_normalize(tfidf_matrix)
manhattan_distances(tfidf_norm_l1[0:1], tfidf_norm_l1[1:2])
```

https://soyoung-new-challenge.tistory.com/34

Similarity Distance Measure

https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/

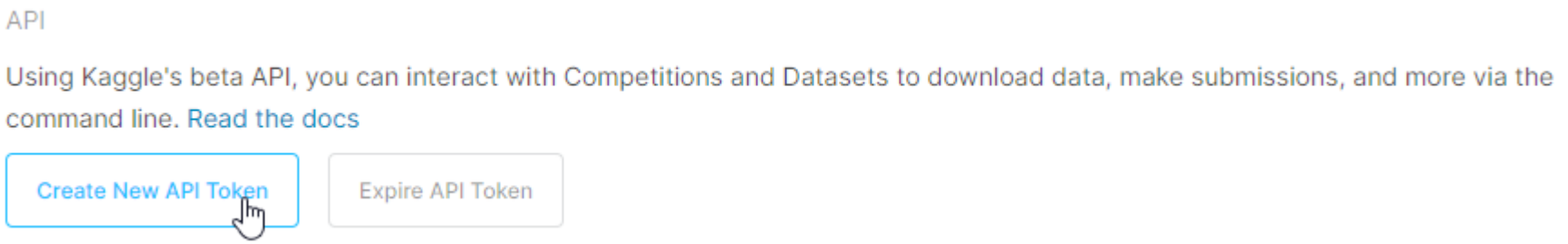
Quora Question Pairs 유사도 측정



Quora Question Pairs 유사도 측정

■ kagge 가입 및 kgggle.json 다운로드

- kaggle 가입 : https://www.kaggle.com/
- kgggle.json 다운로드 : https://www.kaggle.com/<username>/account



■ 데이터 필드

- 데이터 : Quora Question Pairs, https://www.kaggle.com/c/quora-question-pairs/
- 데이터 필드 : id, qid1, qid2, question1, question2, is_duplicate(1-두개의 질문이 중복, 0 - 두개의 질물이 중복 아님)

Quora Question Pairs 유사도 측정

quora_preprocessing.ipynb

quora_similarity_xgboost.ipynb

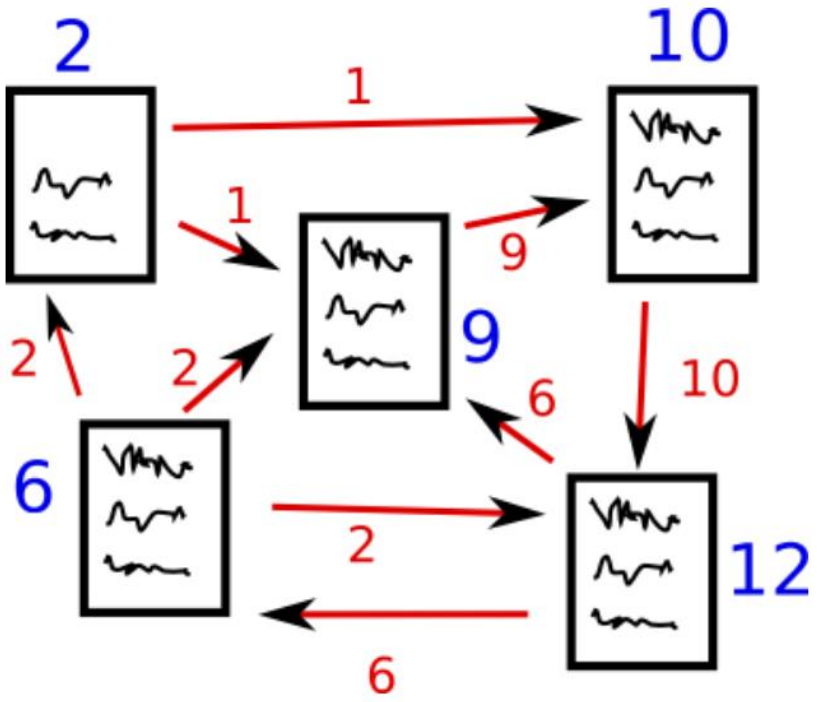
quora_similarity_cnn.ipynb

참고 : 텐서플로 2와 머신러닝으로 시작하는 자연어 처리

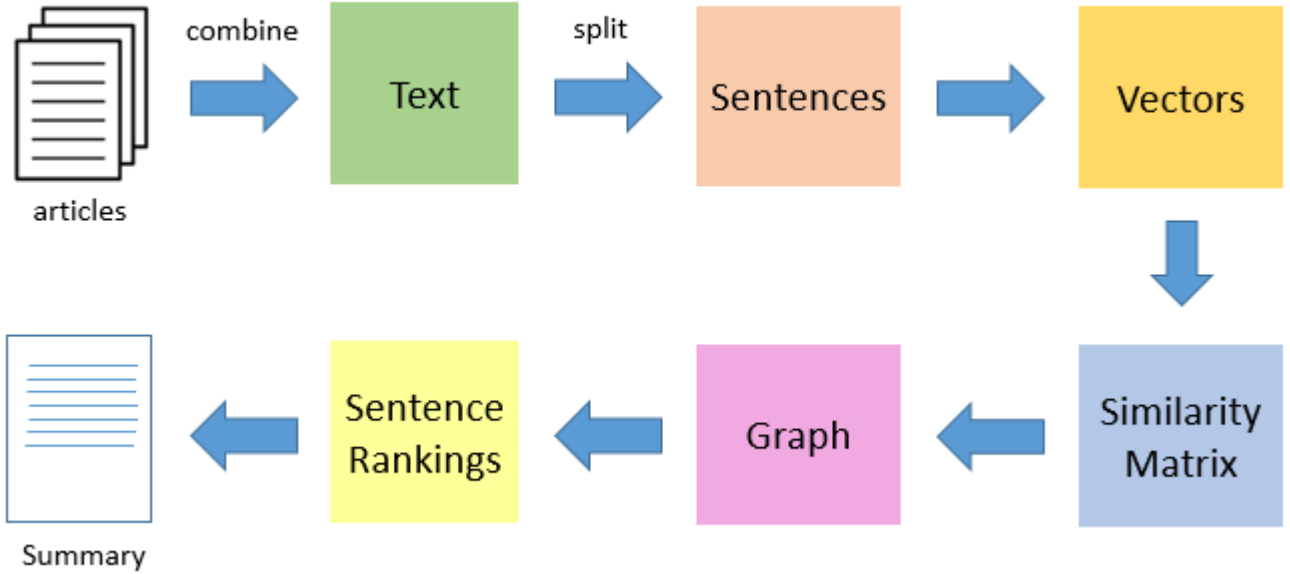
Text Summarization

https://github.com/prateekjoshi565/textrank_text_summarization

PageRank Algorithm



TextRank



https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/

Thank you

kgpark88@gmail.com