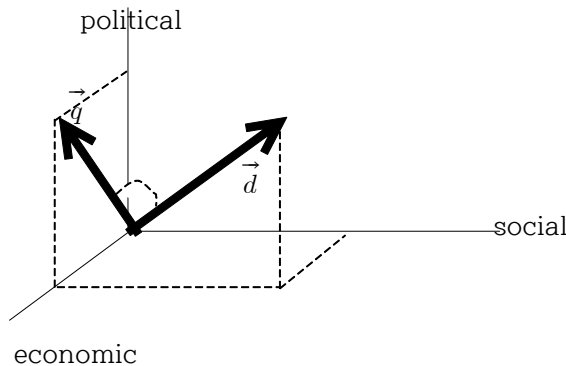


vector space model

Gerard Salton과 그의 동료들이 제안한 모델이며 Luhn의 유사성 기준을 근거로 하였다. 이들은 색인 표현과 쿼리를 각 용어에 독립된 차원을 할당하는 고 차원의 Euclidean space에 있는 벡터로 생각했다. 대체로 유사도는 두 개의 벡터 \vec{d} 와 \vec{q} 를 분리시킨 코사인 각도이다. 만일 벡터가 다차원 공간에서 직교적이라면 코사인 각은 0이고, 만일 각도가 0도라면 코사인 각은 1이다. 이런 코사인 공식은 다음과 같다:

$$score(\vec{d}, \vec{q}) = \frac{\sum_{k=1}^m d_k \cdot q_k}{\sum_{k=1}^m (d_k)^2 \cdot \sum_{k=1}^m (q_k)^2} \quad (\text{등식 2})$$

다차원 공간에서 벡터들 간의 각도 metaphor(은유)는 이 모델의 의미를 비전문가에게 설명하는 것을 쉽게 만든다. 3차원까지, 누구나 다큐와 쿼리 벡터를 쉽게 시각화할 수 있다. 아래의 도 4는 좌표에서 3가지 용어 social, economic, political에 의해 측정된(span) 다큐 벡터와 쿼리 벡터를 시각화한 예이다. 새로운 정보검색 문제에 이 모델을 기하학적으로 해석하도록 적용하는 것은 비교적 쉽다. 벡터 스페이스 모델은 automatic text categorisation 그리고 document clustering에서 주로 사용되고 있다.



벡터들 간의 코사인을 측정하는 것은 unit length에 벡터들을 normalising하여 vector inner product를 구하는 것과 같다. 만일 색인 표현과 쿼리가 올바르게 normalised된다면, 등식 1의 vector product measure는 강력한 이론적 동기를 갖게된다. 그 때 공식은 다음과 같다:

$$score(\vec{d}, \vec{q}) = \sum_{k=1}^m n(d_k) \cdot n(q_k)$$

$$\text{조건(where)} \quad n(v_k) = \frac{v_k}{\sqrt{\sum_{k=1}^m (v_k)^2}} \quad (\text{등식 3})$$

▪ Relevance feedback

몇 가지 성공적인 검색 알고리즘은 만일 벡터 길이가 정규화 되었다면 벡터 스페이스 모델

을 기본으로 삼고 있다. 다음의 예는 Joseph Rocchio의 적합성 피드백 알고리즘이다. Rocchio는 다음과 같은 적합성 피드백 알고리즘과 관련해서, 이것의 조건으로 \vec{q}_{old} 가 본래의 쿼리이고, $\vec{q}_{\neq w}$ 는 수정된 쿼리이며, $\vec{d}_{rel}^{(i)}$ ($1 \leq i \leq r$)은 이용자가 적합하다고 선택한 r 다큐들 중의 하나이고, $\vec{d}_{nonrel}^{(i)}$ ($1 \leq i \leq r$)은 이용자가 부적합하다고 선택한 r 다큐들 중의 하나라고 제안하였다.

$$\vec{q}_{\neq w} = \vec{q}_{old} + \frac{1}{r} \sum_{i=1}^r \vec{d}_{rel}^{(i)} - \frac{1}{n} \sum_{i=1}^n \vec{d}_{nonrel}^{(i)} \quad (\text{등식 4})$$

다큐와 쿼리들의 정규화된 벡터들은 원점(origin)에서부터 unit length로 초구(hypersphere) 상에서 포인트처럼 보여질 수 있다. 위의 등식 4에서, 최초의 합계(sum)은 초구에서 알려져 있는 적합한 다큐들의 포인트들 중심(centroid)을 계산한다. 이 centroid에서 알려진 적합 다큐의 각도는 최소화 된다. 두 번째 합계는 알려져 있는 부적합한 다큐들의 포인트들 중심(centroid)을 계산한다. 알려진 적합 다큐의 centroid 쪽으로 쿼리를 가깝게 이동시키고, 알려진 부적합 다큐에서 쿼리를 멀게 이동시키는 것이 검색 성능을 확실하게 개선시킨다.

▪ Term weighting and other caveats

▶ 벡터 스페이스 모델의 주요 단점:

1) 벡터 구성요소의 값이 무엇이어서 하는지를 정의할 방법이 없다는 것이다. 벡터 구성 요소에 올바른 값을 할당하는 문제를 용어가중치기법(term weighting)이라 한다. Salton과 Salton & Yang의 초기 실험에서 용어가중치기법이란 결코 사소한 문제가 아니라는 것을 보여주었다. 이들은 용어의 빈도수 tf (다큐 속에 있는 한 용어의 발생 수)와 역문헌빈도수 idf (다큐 빈도수 df (그 용어를 포함하고 있는 다큐의 수)와 관련된 역의 수)를 결합시킨 소위 $tf.idf$ 가중치를 제안하였다. 오늘날 많은 가중치 알고리즘들은 $tf.idf$ 가중치 알고리즘의 친척들이다. Salton의 최초의 $tf.idf$ 가중치는 비교적 성능이 빈약하며, 어떤 경우에는 간단한 idf 가중치보다도 더 떨어진다. 이들은 다음과 같이 정의하였다:

$$d_k = q_k = tf(k, d) \cdot \log \frac{N}{df(k)} \quad (\text{공식 5})$$

where $tf(k, d)$ 는 다큐 d 에 있는 용어 k 의 발생 수이고, $df(k)$ 는 k 를 포함하고 있는 다큐의 수이며, N 은 집단 속에 있는 다큐의 총 수이다.

2) 또 다른 문제는 이것의 실행에 있다. 코사인 척도의 계산은 모든 벡터 구성요소의 값을 필요로 하지만, 도치파일에서는 이것들을 이용할 수 없다. 실재로는 정규화된 값과 vector product 알고리즘이 사용되어야만 한다. 정규화된 가중치는 도치파일에 저장되어 있어야만 하거나, 또는 정규화 값들이 독립적으로 저장되어야만 한다. 두 가지 모두 색인을 갱신하는 경우에 문제를 발생시킨다: 한 건의 새로운 다큐를 추가시키는 것은 그 다큐에서 발생하는 용어들의 다큐 빈도수를 변화시키며, 한 개이상의 이같은 용어를 포함하고 있는 모든 다큐의 벡

터 길이를 변화시킨다.

< 예제 >

간단하게, 다음과 같은 3개의 documents로 이루어진 소규모 문서군(collection) C가 있고, 그 속에 다음과 같은 terms를 포함하고 있다고 가정해 보자:

d1: "new york times"
d2: "new york post"
d3: "los angeles times"

어떤 용어("new", "york", "times")는 두 개의 문서에 포함되어 있고, 또 어떤 것("post", "los", "angels")은 단지 한 개의 문서에만 존재한다. 그리고 문서의 총 수는 3 이다.

이 용어들의 역문헌빈도(idf: Inverse Document Frequency)는 다음과 같다:

angles	$\log_2(3/1)=1.584$
los	$\log_2(3/1)=1.584$
new	$\log_2(3/2)=0.584$
post	$\log_2(3/1)=1.584$
times	$\log_2(3/2)=0.584$
york	$\log_2(3/2)=0.584$

- IDF의 값에서는 상용log를 사용할 수 있다: 그러나 우리의 예제에서는 \log_2 를 사용하였다.

이제 다큐먼트 집단인 C에 있는 모든 용어의 tf(Term Frequency) score(용어빈도수)를 계산해 보면 아래와 같다: vectors에 있는 terms는 알파벳순으로 정렬되어있다고 가정한다.

	angeles	los	new	post	times	york
d1	0	0	1	0	1	1
d2	0	0	1	1	0	1
d3	1	1	0	0	1	0

이제, 각 용어의 idf 값을 tf score에 곱하면, 아래와 같은 documents-by-terms matrix를 얻게 된다: (모든 용어는 C에 있는 각 문서에 단지 한번만 나타나며, 최대 값은 1 이다).

- Boolean vector: 문서에 단어 ti가 존재하면 1, 존재하지 않으면 0으로 표현하는 방법

따라서 td-idf의 값은 td의 값과 idf의 값을 곱하여 구하며, 그 결과는 다음과 같다:

	angeles	los	new	post	times	york
d1	0	0	0.584	0	0.584	0.584
d2	0	0	0.584	1.584	0	0.584
d3	1.584	1.584	0	0	0.584	0

잠깐!

TF-IDF(Term Frequency - Inverse Document Frequency)는 정보 검색과 텍스트 마이닝에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. 문서의 핵심어를 추출하거나, 검색 엔진에서 검색 결과의 순위를 결정하거나, 문서들 사이의 비슷한 정도를 구하는 등의 용도로 사용할 수 있다.

TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값으로, 이 값이 높을수록 문서에서 중요하다고 생각할 수 있다. 하지만 단어 자체가 문서군 내에서 자주 사용되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을 의미한다. 이것을 DF(문서 빈도, document frequency)라고 하며, 이 값의 역수를 IDF(역문서 빈도, inverse document frequency)라고 한다. TF-IDF는 TF와 IDF를 곱한 값이다.

IDF 값은 문서군의 성격에 따라 결정된다. 예를 들어 '원자'라는 낱말은 일반적인 문서들 사이에서는 잘 나오지 않기 때문에 IDF 값이 높아지고 문서의 핵심어가 될 수 있지만, 원자에 대한 문서를 모아놓은 문서군의 경우 이 낱말은 상투어가 되어 각 문서들을 세분화하여 구분할 수 있는 다른 낱말들이 높은 가중치를 얻게 된다.]

“new new times”이란 쿼리 q가 주어졌을 때, 우리는 쿼리용의 tf-idf vector를 계산한 다음에, 다큐먼트 집단인 C에 있는 각 다큐의 점수를 cosine similarity measure를 사용하여 이 쿼리와 비교하여 계산한다.

쿼리용어의 tf-idf 값을 계산할 때, 우리는 그 빈도를 최대 빈도수(2)로 나눈 다음에, idf 값을 곱하면 다음과 같다:

“new”는 C에서 2번 출현하였고 q에서도 2번 출현하였다. 그러나 “times”는 C에서는 2번 출현하였으나 q에서는 1번만 출현하였다.

	angeles	los	new	post	times	york
q	0	0	$(2/2)*0.584=0.584$	0	$(1/2)*0.584=0.292$	0

각 문서와 쿼리의 길이를 계산하면 다음과 같다:

(1)루트계산기: <<http://mwultong.blogspot.com/2007/12/root-calculator.html>>

(2)제곱값계산기: <<http://mwultong.blogspot.com/2008/01/pow-calc.html>>

$$\blacksquare d1\text{의 vector length} = \sqrt{0.584^2+0.584^2+0.584^2}= 1.011$$

$$\blacksquare d2\text{의 vector length} = \sqrt{0.584^2+1.584^2+0.584^2}= 1.786$$

$$\blacksquare d3\text{의 vector length} = \sqrt{1.584^2+1.584^2+0.584^2}= 2.316$$

$$\blacksquare q\text{의 vector length} = \sqrt{0.584^2+0.292^2}= 0.652$$

따라서 유사도 값(similarity values)은 다음과 같다:

$$\begin{aligned}\blacksquare \cos\text{Sim}(d1,q) \\ &= (d1:\text{angles의 td-idf값} * q:\text{angles의 td-idf값} + \dots + d1:\text{york의 td-idf값} * q:\text{york의 td-idf값}) / (d1\text{의 vector length} * q\text{의 vector length}) \\ &= (0*0+0*0+0.584*0.584+0*0+0.584*0.292+0.584*0) / (1.011*0.652) \\ &= 0.776\end{aligned}$$

$$\begin{aligned}\blacksquare \cos\text{Sim}(d2,q) &= (0*0+0*0+0.584*0.584+1.584*0+0*0.292+0.584*0) / (1.786*0.652) \\ &= 0.292\end{aligned}$$

$$\begin{aligned}\blacksquare \cos\text{Sim}(d3,q) &= (1.584*0+1.584*0+0*0.584+0*0+0.584*0.292+0*0) / (2.316*0.652) \\ &= 0.112\end{aligned}$$

위의 유사도 값에 따라, 쿼리 결과로 나타나는 최종적인 문서의 서열은 다음과 같다:

- Rank 1:d1,
- Rank 2:d2,
- Rank 3:d3.

FIN