Result after applying query to inverted index

["augmenting", "returns" "noticed"]

→ [137228 , 327419 , 929777]

[7, 20, 3] [8, 30, 0] [2, 5 3]

1. Language model

without smoothing:

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{mle}(t|M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d}$$

with smoothing: (linear interpolation language model)

$$P(d|q) \propto P(d) \prod_{t \in q} ((1-\lambda)P(t|M_c) + \lambda P(t|M_d))$$

variations:

- how to calculate P(d)
  - → ignore?
  - → length of document?
  - → other?

$$P(t|Md) = \prod_{t \in q} \frac{TF_{t,d}}{Ld}$$

term freq

length of doc

other option: "Bayesian Smoothing":

$$\hat{P}(t|d) = \frac{tf_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha}$$

Worked example. Suppose the document collection contains two documents:
d_1

: Xyzzy reports a profit but revenue is down
d_2

: Quorus narrows quarter loss but revenue decreases further
The model will be MLE unigram models from the documents and collection, mixed with
$\lambda = 1/2$

.
Suppose the query is *revenue down*. Then:

$$P(q|d_1) = [(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2] \quad (105)$$
$$= 1/8 \times 3/32 = 3/256 \quad (106)$$
$$P(q|d_2) = [(1/8 + 2/16)/2] \times [(0/8 + 1/16)/2] \quad (107)$$
$$= 1/8 \times 1/32 = 1/256 \quad (108)$$

So, the ranking is
$d_1 > d_2$

. **End worked example.**

From <http://nlp.stanford.edu/IR-book/html/htmledition/estimating-the-query-generation-probability-1.html>

2. Tf-weighting

$$idf_t = \log \frac{N}{df_t}$$

number of documents that contain t

alternative: use of (count in collection)

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$$

2.1 Sum of tf-idfs

$$Score(q,d) = \sum_{t \in q} tf\text{-}idf_{t,q}$$

2.2 Vector space modeling

seems more flexible.

$$Score(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{\rightarrow \cdot \rightarrow}$$

where $\vec{V}(d)$ is a vector with tf-idf weights

$$\text{score}(q,d) = \frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)|\cdot|\vec{V}(d)|}$$ where $\vec{V}(d)$ is a vector with tf-idf weights for each word in dictionary.

if query = "jealous gossip" $\vec{V}(q) =$ (0, 0, 0, 2, 0, 0, 3, 0) ← index of gossip; ← index of jealous

possible variants for vector weights (applicable to both query & documents)

Term frequency		Document frequency		Normalization	
n (natural)	$\text{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\text{tf}_{t,d})$	t (idf)	$\log\frac{N}{\text{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\text{tf}_{t,d}}{\max_t(\text{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-\text{df}_t}{\text{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \text{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/\textit{CharLength}^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\text{tf}_{t,d})}{1+\log(\text{ave}_{t\in d}(\text{tf}_{t,d}))}$				

⇒ a very standard weighting scheme is *lnc.ltc*, where the document vector has log-weighted term frequency, no idf (for both effectiveness and efficiency reasons), and cosine normalization, while the query vector uses log-weighted term frequency, idf weighting, and cosine normalization.

From <http://nlp.stanford.edu/IR-book/html/htmledition/document-and-query-weighting-schemes-1.html>

► **Figure 6.7** SMART notation for tf-idf variants. Here *CharLength* is the number of characters in the document.

From <http://nlp.stanford.edu/IR-book/html/htmledition/document-and-query-weighting-schemes-1.html>

quick calculations of all scores:

```
COSINESCORE(q)
 1  float Scores[N] = 0
 2  Initialize Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5     for each pair(d, tf_{t,d}) in postings list
 6     do Scores[d] += wf_{t,d} × w_{t,q}
 7  Read the array Length[d]
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

From <http://nlp.stanford.edu/IR-book/html/htmledition/computing-vector-scores-1.html>

Things to consider:

- zone scoring ? (for title) http://nlp.stanford.edu/IR-book/html/htmledition/learning-weights-1.html
  ⇒ seems to be more applicable to boolean judgment, though.