CHAPTER #6: VECTOR SPACE MODEL

Dated: _____

★ **Metadata** => Digital docs generally encode in machine-recognizable form, certain metadata like authors, title and date of publication.

★ **field** => The metadata values are called fields.

★ **parametric index** => The system holds the inverted index as well as a parametric index which exists for all fields. These search parameters retrieve information and then are merged with inverted index for final results. e.g: date of creation parameter can be used to get docs after a certain date. The dict comes from a fixed vocab.

★ **zone** => A zone is an arbitrary unbounded amount of text like abstracts or references in research papers. We may built a separate inverted index for each zone to support queries. The dict must structure whatever vocab stems from the zone.

★ author: _____
title: _____
language: _____
} parametric search example

william.abstract → [11] → [121] → [1441]
william.title → [2] → [4] → [8]
} zone index
} zones as ext of dict

william → [2.author, 2.title] → [3.author]
} zone is encoded in postings.

★ ~~Weighted zone scoring~~

★ **Ranked Retrieval** =>

↳ return an ordering over the top documents in the collection.
↳ the users query is in human language.

★ Score-ranking =>

↳ for ranked retrieval, assign a score [0,1] - to each document.

↳ this score measures how well doc & query match.

↳ if score = 0, query is not in document.

↳ the higher the score, the more frequent ~~**~~ term occurs in doc.

↳ the most basic is the jaccard co-efficient.

★ Weighted Zone Scoring => Different zones may have a different importance in evaluating how a document matches a query.

Given a boolean query q and a document d, weighted zone ~~**~~ scoring assigns to the pair (q,d) a score in the interval [0,1] by computing a linear combination of zone scores.

Consider a set of documents each of which has l zones. Let $g_1, \ldots, g_l \in [0,1]$ such that $\sum_{i=1}^{l} g_i = 1$.

zones of a doc.

For $1 \le i \le l$, let s_i be the boolean score (→ only 0 or 1) denoting a match (or mismatch) b/w q and ith zone, then the weighted zone scoring becomes:

$$\sum_{i=1}^{l} g_i s_i$$

It is also called ranked boolean retrieval.

↳ Example: Consider the query shakespeare in a collection in which each document has zones: author, title, body. The Boolean score will be 1 if query exists in zone else 0. Let $g_1 = 0.2$ by author weight, $g_2 = 0.3$ be title weight, $g_3 = 0.5$ be body weight. This suggest that author zone is least important and body zone is most important.

Thus if, term was not to appear in author, the document score will be 0.8.

* term frequency (tf) => number of occurrences of a term in the document.

* Bag of words => In this model, a text is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

ex: John likes to watch movies. Mary likes movies too.

BoW = {John, like, watch, movie, Mary, like, movie}.

To compute scores, we find BoW of the query & find the jaccard co-efficient.

Dated: _____

↳ Issues =>
  ↳ Doc. size affects overall similarity.
  ↳ frequency of common terms are ignored.
  ↳ similarity is based on common terms, which are treated independent of each other.

* Document Frequency (df) => number of docs in the collection that contain the term t.

* Inverse Doc. Freq. (idf) =>
Let N be the number of documents in a collection, the idf for a term t is:

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

The idf of a rare term is high, and of a frequent term is low.

* tf-idf weighting => we combine the two to produce a composite weight for each term in a document.
The tf-idf weighting scheme assigns to a term t a weight in the document d as:

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

  ↳ highest when t occurs many times within a small number of documents. (lending high discriminating power)
  ↳ lower when term occurs fewer times in a document, or occurs in many documents. (less pronounced relevance signal)
  ↳ lowest when the term occurs in virtually all documents.

* Cumulative frequency => how many times a term appears in a collection (cf).

★ Document Vectors =>

At this point, viewing each document has a vector with one component corresponding to each term in the dictionary, together with the weight. If they do not occur in the doc, the weight is zero.

↳ Overlap score measure => the score of a document d is the sum over all query terms, of the number of times each query terms occur in d. We can refine this with the use of tf-idf.

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

★ Vector Space Model => The representation of a set of documents as vectors in a common vector space and it is fundamental to a host of info retrieval operations ranging from scoring docs on a query, document classification & document clustering.

↳ Idea =>

↳ The basic idea of VSM is to have a high-dimensional hyperspace where each term is a dimension and document & query can be represented in this space.

↳ It is a simple yet effective method of designing ranking functions for IR.

↳ Assumptions =>

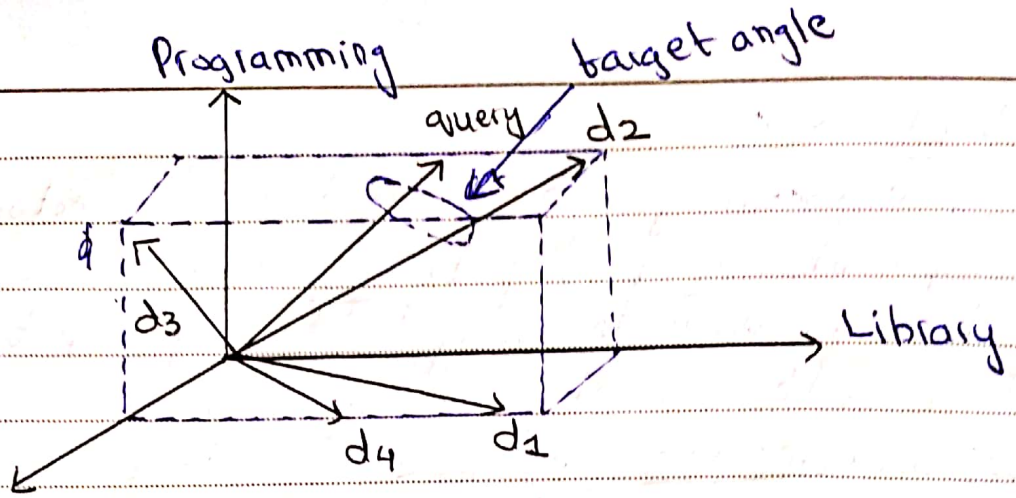
↳ All terms are features and they are all independent.

↳ the vocabulary (V) is limited.

↳ the vocabulary formed a space of |V| dimensions.

↳ Relevance can be computed as distance or angle b/w vectors of query and document.

Dated:



↳ ~~Word Procedure~~ Measurements =>

Let $\vec{V}(d)$ denote the vector derived from document d. How do we quantify the similarity b/w two documents in the vector space.

The first ~~step~~ attempt might be to consider the magnitude of vector difference b/w two vectors. This suffers from a major drawback if the documents have very similar content but one of the documents is too long. Thus the relative distribution of terms may be identical but absolute term frequencies of one may be far larger.

To compensate for this effect, the standard way of quantifying the similarity is to compute the

Cosine similarity.

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

where the numerator is the dot product while the denominator is the Euclidean length. The dot product of two vectors x and y can be calculated as $\sum_{i=1}^{M} x_i y_i$ where M is the number of components and the euclidean length is $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

The effect of denominators is to length normalize the vectors to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$.

$$\cos\theta = \frac{q \cdot d}{|q||d|}$$

* The component magnitudes are tf instead of exist or not exists

Dated:

So, the eqn becomes

$$\text{sim} \leftarrow \text{Sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2)$$

The term documents can be represented with a term document matrix of M X N dimensions where M is the total terms and N is the documents in the collection.

↳ Queries as vectors=>

We are representing queries as vectors as well. The key idea is to assign each document d a score equal to dot product => $\vec{v}(q_1) \cdot \vec{v}(d)$ or

by viewing query as a bag of words, we are able to treat it as a very short document. Hence, we can use cosine similarity to find that document score.

$$\text{Score}(q_1, d) = \frac{\vec{V}(q_1) \cdot \vec{V}(d)}{|\vec{V}(q_1)||\vec{V}(d)|}$$

Computing the cosine similarities b/w query and each document, sorting the resulting scores and selecting the top K documents is really expensive.

~~↳ Instantiation of VSM=>~~

~~↳ VSM is a framework~~

↳ Computing Vector Scores=>

COSINE SCORE (q_1):

```
1   float Scores[N] ← 0
2   Initialize length[N]
3   for each query term t
4       calculate w_{t,q} and fetch posting list for t
5           for each pair (d, tf_{t,d}) in postings list
6               add wf_{t,d} to Scores[d]
7   Read the array length[d]
8   for each d
9       divides scores[d] by length[d]
10  return top K scores.
```

Dated: _____

we are seeking the K documents with the highest vector space scores on the given query.

The array lengths[N] holds the lengths(normalization factors) of each of the N documents. The array Scores holds the document scores. The outermost loop in Step 3 repeats the updating of scores iterating over each query term. In Step 5, we calculate the weight in query vector for term t. Step 6-8 updates the score of each document by adding the contribution from t. This process of adding in contributions one query term at a time is known as term-at-a-time scoring or accumulation, and the N elements of scores array are known as accumulators.

For this purpose, it seems necessary to store tf weight of term with each posting entry. This is wasteful because it requires storing an extra float.

Two ideas can resolve this problem.

First, if we are using ~~inter~~ idf, we need not precompute idf, it suffices to store N/df_t at the head of the postings for t. Second, we store the term frequency for each posting entry. Finally, a priority queue with a ~~max~~ heap taking 2N comparisons to construct and O(logn) extraction can be used ~~fo~~ to get the top K scores.

document-at-a-time => computing scores of one doc at a time.

↳ advantages =>

- ↳ simple model based on linear algebra.
- ↳ Term weights are not binary.
- ↳ allows computing a continuous degree of similarity.
- ↳ allows ranking documents.
- ↳ allows partial matching.

Dated: _____

↳ disadvantages⇒
- ↳ the orders in which terms appear is lost.
- ↳ theoretically assumes that terms are independent.
- ↳ search keywords must precisely match doc terms, word substrings might give false tves.
- ↳ semantic sensitivity; documents with similar context but different term vocab won't be associated resulting in a false -ve.

* Sublinear tf scaling⇒ {reduce impact of high frequency counts}

A common modification can be made to normal tf since all occurences may or may not signify the same importance so the new weight can be calculated as:

$$wf_{t,d} = \begin{cases} 1 + \log tf_{t,d} & \text{if } tf > 0 \\ 0 & \text{otherwise} \end{cases}$$

tf-idf can also be changed as:

$$wf\text{-}idf_{t,d} = wf_{t,d} \times idf_t$$

* Maximum tf normalization⇒

tf-idf scores sometimes give skewed results based on frequency so this issue can be addressed by dividing the raw frequency count of each term by the highest frequency ~~t~~count of any term in the document. This ensures that the highest score can be 1 and all others are proportionally adjusted.

$$n\,tf_{t,d} = a + (1-a)\frac{tf_{t,d}}{tf_{max(d)}}$$

Here, a is called a smoothing term that has a value b/w 0 and 1 usually 0.4 is used for damping the contribution of the second term which is scaling down tf by the largest tf value in d.

↳ unstable, changes in stopwords can cause drastic changes.

↳ a doc may have an unsually frequent outlier.