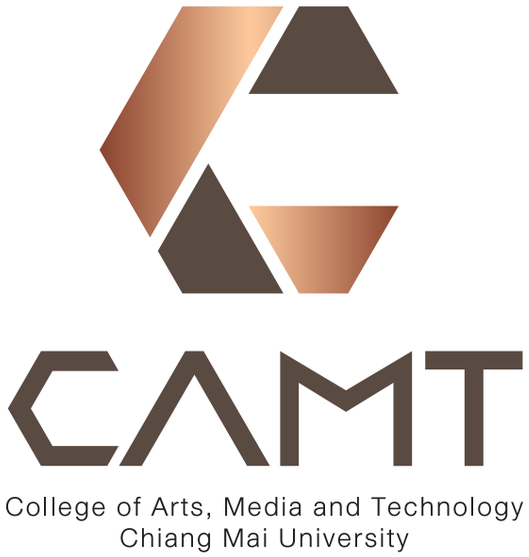
SE 481 Introduction to Information Retrieval

(IR for SE)

Module #3 — IR Models



Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

- Some histories
- Scoring documents
- Term frequency (tf) & Inverse document frequency (idf)
- Vector space model
- Probabilistic model

Some histories

- Boolean model (±1950)
- Document similarity (±1957)
- Vector space model (±1970)
- Probabilistic retrieval (±1976)
- Language models (±1998)
- Google PageRank (±1998)

Boolean retrieval

- Boolean retrieval involves answering yes/no questions,
  - determining whether a document matches a query or not.
- It does **not** assess the degree of relevance of document A compared to document B.
- It becomes less practical when a query returns a large number of matched results (e.g., 1000++).

Boolean retrieval

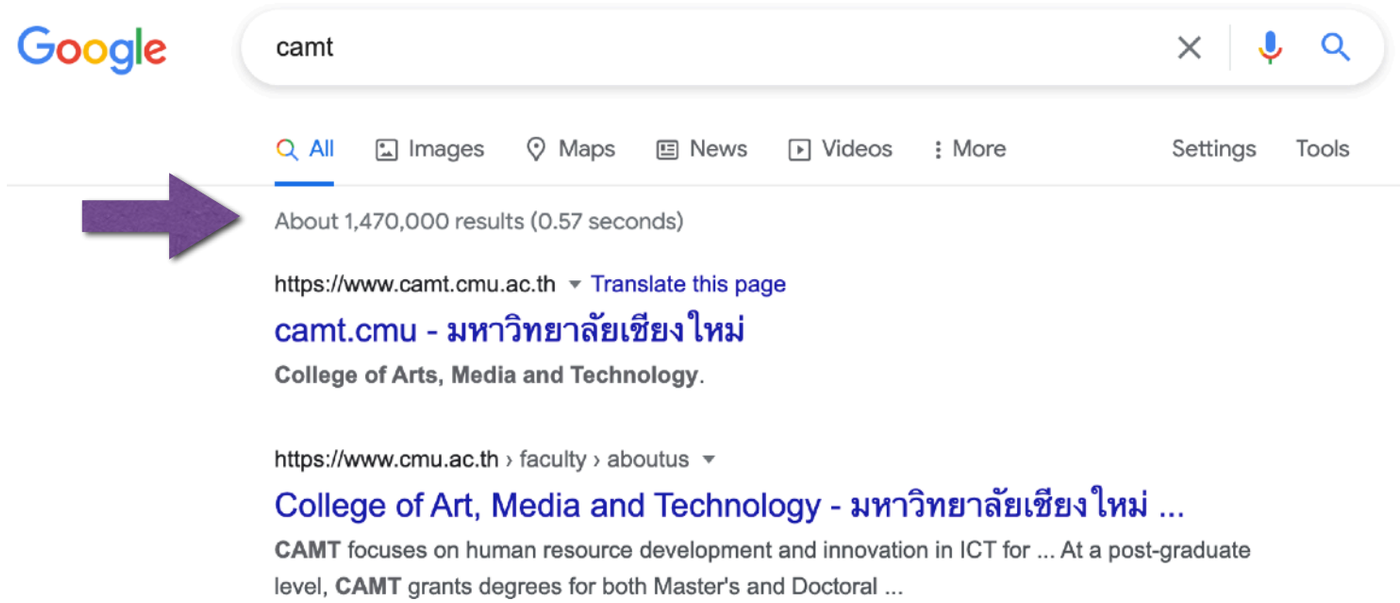
- Boolean retrieval primarily focuses on exact matching
  - more aligned with **data retrieval** rather than **information retrieval**.
- In practice, Boolean retrieval can be challenging:
  - Using AND (intersect) often yields too few results.
  - Using OR (union) often yields too many results.

Boolean retrieval

- Venn diagrams

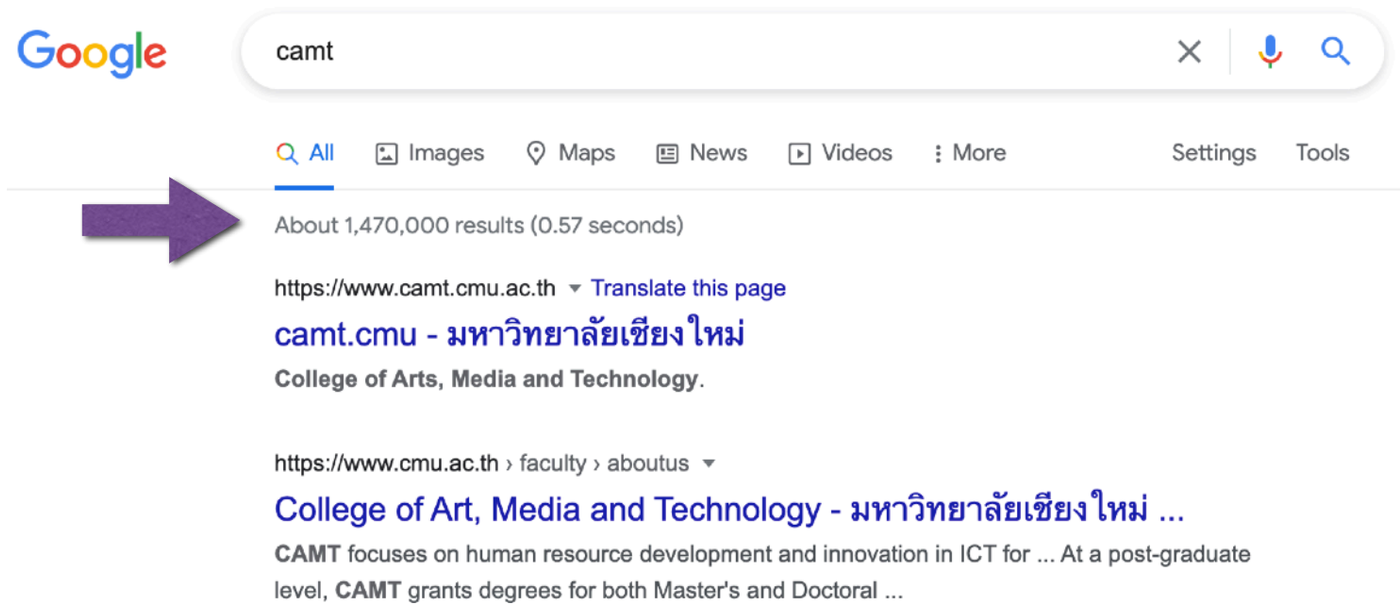
Ranked retrieval models

- A score, typically a real number, is also provided along with the results.
  - This score allows us to rank and order the results based on their relevance.
- Dealing with large result sets is generally less challenging than in Boolean retrieval.
- We can address this by displaying only the top k results, typically around 10, to users.



What do we need?

- Ranking (scoring) algorithm
  - Assign a score – [0, 1] – to each document;
  - This score reflects the quality of the match between the documents and the query.
  - Essentially, it quantifies the degree of alignment between the query and the document, serving as a query-document matching score.



Query-document matching scores

- Keys
  - It is essential to establish a method for assigning a score to a query/document pair.
  - When a query term is absent in the document, the score should be set to 0 for the term.
  - The higher the relevance of the query to the documents, the greater the assigned scores should be.
    - For instance:
      - A higher score implies a more frequent occurrence of the query term in the document.
      - Are there any other possible alternatives to consider for scoring?

Query-document matching scores

- Recall a binary representation
  - E.g., Binary term-document incidence matrix

	applic	design	develop	experi	requir	softwar	team	technolog	test	work
0	2	2	3	1	1	1	1	1	3	1
1	1	1	2	1	1	1	1	2	0	1
2	2	0	2	1	1	1	1	0	2	2
3	0	0	2	1	0	1	2	0	1	1
4	1	1	2	1	1	1	1	2	0	1
...	...	...	...	...	...	...	...	...	...	...
995	2	2	5	1	0	1	2	2	0	2
996	2	1	5	2	1	1	2	0	2	2
997	4	1	3	1	3	1	2	2	3	3
998	4	2	4	1	2	1	2	2	3	3
999	2	2	2	0	1	1	2	2	1	2

Query-document matching scores

- Term-document count matrices
  - Consider the number of occurrences of a term in a document:

	applic	design	develop	experi	requir	softwar	team	technolog	test	work
0	2	2	3	1	1	1	1	1	3	1
1	1	1	2	1	1	1	1	2	0	1

- This is also known as a Bag of words model

Converting into a full scikit-learn process

```
def create_stem_cache(cleaned_description):
    tokenized_description = cleaned_description.apply(lambda s: word_tokenize(s))
    concated = np.unique(np.concatenate([s for s in tokenized_description.values]))
    stem_cache = {}
    ps = PorterStemmer()
    for s in concated:
        stem_cache[s] = ps.stem(s)
    return stem_cache
```

```
def create_custom_preprocessor(stop_dict, stem_cache):
    def custom_preprocessor(s):
        ps = PorterStemmer()
        s = re.sub(r'[^A-Za-z]', ' ', s)
        s = re.sub(r'\s+', ' ', s)
        s = word_tokenize(s)
        s = list(OrderedSet(s) - stop_dict)
        s = [word for word in s if len(word)>2]
        s = [stem_cache[w] if w in stem_cache else ps.stem(w) for w in s]
        s = ' '.join(s)
        return s
    return custom_preprocessor
```

Converting into a full scikit-learn process

```
1   def sk_vectorize(texts, cleaned_description, stop_dict, stem_cache):
2       my_custom_preprocessor = create_custom_preprocessor(stop_dict, stem_cache)
3       vectorizer = CountVectorizer(preprocessor=my_custom_preprocessor)
4       vectorizer.fit(cleaned_description)
5       query = vectorizer.transform(texts)
6       print(query)
7       print(vectorizer.inverse_transform(query))
```

```
1   cleaned_description = m1.get_and_clean_data()
2   stem_cache = create_stem_cache(cleaned_description)
3   stop_dict = set(stopwords.words('English'))
4   sk_vectorize(['python is simpler than java'], cleaned_description, stop_dict, stem_cache)
```

(0, 13947) 1

(0, 21383) 1

(0, 24234) 1

['java', 'python', 'simpler']

Bag of words

- Bag of words
  - Vector representation doesn't consider the ordering of words in a document.
  - *python is simpler than java* and *java is simpler than python* are represented by the **same** vector.

Ref: https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/

An Scikit-learn process

```
sk_vectorize(['python is simpler than java', 'java is simpler than python]', cleaned_description,
stop_dict, stem_cache)
```

- Bag of words >> these yield the same vectors.

(0, 13947) 1	(1, 13947) 1
(0, 21383) 1	(1, 21383) 1
(0, 24234) 1	(1, 24234) 1
['java', 'python', 'simpler']	['java', 'python', 'simpler']

Handling sequence

```
1   bigram_vectorizer = CountVectorizer(preprocessor=my_custom_preprocessor, ngram_range=(1, 2))
2   bigram_vectorizer.fit(cleaned_description)
3   print(len(bigram_vectorizer.get_feature_names_out()))
```

Features length: 30513 -> 396338

Handling sequence

```
1   trigram_vectorizer = CountVectorizer(preprocessor=my_custom_preprocessor, ngram_range=(1, 3))
2   trigram_vectorizer.fit(cleaned_description)
3   print(len(trigram_vectorizer.get_feature_names_out()))
```

Features length: 30513 -> 396338 -> 1103601

Term frequency (*tf*)

- *tf* (t, d): The number of times term t appears in document d.
- In information retrieval (IR), frequency refers to the count of occurrences.
- However, relying solely on raw *tf* does not convey the full picture:
  - A mere increase in frequency by a factor of 10 does not necessarily equate to a 10-fold increase in relevance.
  - A practical solution involves the application of **weighting techniques**.

Term frequency (*tf*) with *log* weighting

$$w_{t,d} = log_{10}(1 + tf_{t,d}) \text{ if } tf_{t,d} > 0$$

- A score system for a document-query pair:
  - sum over terms ***t*** in both ***q*** and ***d***
  - 0 when none of the query terms is present in the document.

Logarithm Table

Table of base 10, base 2 and base e (ln) logarithms:

x	$\log_{10} x$	$\log_2 x$	$\log_e x$
1	0.000000	0.000000	0.000000
2	0.301030	1.000000	0.693147
3	0.477121	1.584963	1.098612
4	0.602060	2.000000	1.386294
5	0.698970	2.321928	1.609438
6	0.778151	2.584963	1.791759
7	0.845098	2.807355	1.945910
8	0.903090	3.000000	2.079442
9	0.954243	3.169925	2.197225
10	1.000000	3.321928	2.302585
20	1.301030	4.321928	2.995732
30	1.477121	4.906891	3.401197
40	1.602060	5.321928	3.688879
50	1.698970	5.643856	3.912023
60	1.778151	5.906991	4.094345
70	1.845098	6.129283	4.248495
80	1.903090	6.321928	4.382027
90	1.954243	6.491853	4.499810
100	2.000000	6.643856	4.605170
200	2.301030	7.643856	5.298317
300	2.477121	8.228819	5.703782
400	2.602060	8.643856	5.991465
500	2.698970	8.965784	6.214608
600	2.778151	9.228819	6.396930
700	2.845098	9.451211	6.551080
800	2.903090	9.643856	6.684612
900	2.954243	9.813781	6.802395
1000	3.000000	9.965784	6.907755

Ref: https://www.quora.com/In-log-table-log10-0-but-in-Quora-some-say-log10-1-which-is-correct-and-what-is-the-reason

Observation: Rare terms are more informative

- E.g., stop words have no information
- Thus, we want a higher weight for those rare terms

Collection vs document frequency

- Collection Frequency (*cf*): The count of term t's occurrences across the entire document collection.
- Document Frequency (*df*): **The number of documents** in which term *t* is found.

Word	Collection freq	Document freq
software	3000	2900
oauth	300	2

- Term **oauth** should receive a higher weighting due to its potential to represent documents containing it.

Inverse document frequency (*idf*)

- df_t is the document frequency of ***t***:
  - The number of documents that contain t
- $idf_t = \log_{10}(N/df_t)$
- *idf* example, suppose that N = 1,000,000

Term	df_t	idf_t
oauth	1	6
virtualize	100	4
frontend	1,000	3
error	10,000	2
application	100,000	1
the	1,000,000	0

Effect of *idf* on ranking

- *idf* plays a significant role in the ranking of documents when queries consist of at least two terms.
- In queries such as **iOS application,** *idf* assigns a higher weight to the term **iOS** compared to **application**.

tf-idf weighting

- The *tf-idf* weight of a term is calculated as the product of its *tf* weight and *idf* weight, e.g., as follows:

$$w_{t,d} = log_{10}(1 + tf_{t,d}) \times log_{10}(1 + \frac{N}{df_t})$$

- This weighting scheme is widely regarded as one of the most straightforward but very effective in information retrieval (IR).
- The scores assigned to terms increase **with two key factors:**
  - The number of occurrences within a document.
  - The rarity of the term in the entire document collection.

tf-idf weighting

```
01  X = vectorizer.transform(cleaned_description)
02  N = len(cleaned_description)
03
04  df = np.array((X.todense()>0).sum(0))[0]
05  idf = np.log10(1+(N / df))
06  tf = np.log10(X.todense()+1)
07
08  tf_idf = np.multiply(tf, idf)
09
10  X = sparse.csr_matrix(tf_idf)
11  print(X.toarray())
12
13  X_df = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names_out())
14
15  max_term = X_df.sum().sort_values()[-20:].sort_index().index
16  print(X_df[max_term].to_markdown())
```

$$w_{t,d} = log_{10}(1 + tf_{t,d}) \times log_{10}(1 + \frac{N}{df_t})$$

Score for a document in a given query

$$Score(q,d) = \sum_{t \in q \cap d} (tf \cdot idf)_{t,d}$$

- Binary —> count —> weight matrix

	applic	design	develop	provid	requir	system	team	technolog	test	work
0	0.162810	0.169439	0.183138	0.127158	0.105262	.00000	0.105974	0.113112	0.236987	0.096707
1	0.102722	0.106904	0.145133	0.000000	0.105262	.00000	0.105974	0.179279	0.000000	0.096707
2	0.162810	0.000000	0.145133	0.127158	0.105262	.11439	0.105974	0.000000	0.187808	0.153277
3	0.000000	0.000000	0.145133	0.127158	0.000000	.11439	0.167965	0.000000	0.118493	0.096707
4	0.102722	0.106904	0.145133	0.000000	0.105262	.00000	0.105974	0.179279	0.000000	0.096707

Activity

- Show the X_df dataframe, showing with only the top 20 **bigram** terms with the highest sum of TF-IDF scores.
  - Compare this with the computation time of unigram analysis.

Documents as vectors

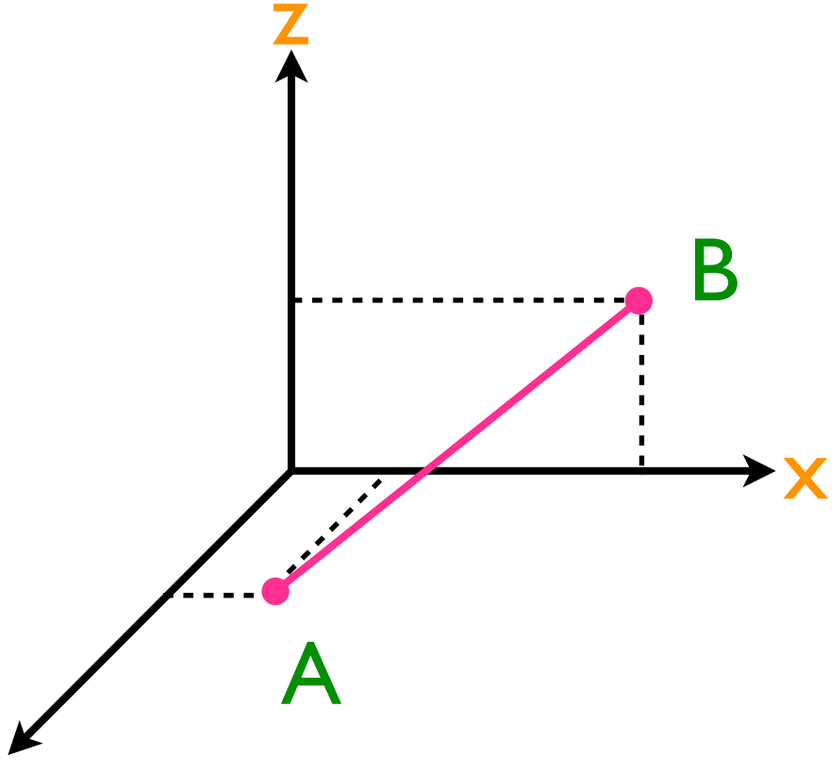
- The concept of term vectors has been previously mentioned
  - |V|-dimensional vector space
  - Terms serve as the axes of this vector space.
- These term vectors are extremely high-dimensional:
  - In real-world search engines, they can have dimensions exceeding a billion.
  - However, it's important to note that these vectors are highly sparse, with the majority of their dimensions being **zero**.

Queries as vectors

- **Key Idea 1:** Represent queries using the same mechanism as documents, which means representing them as vectors within the same space.
- **Key idea 2:** Rank documents based on their closeness or proximity to the query within this vector space
  - In this context, proximity is synonymous with the similarity of vectors, and it's inversely related to the distance between them.

Formalizing vector-space proximity

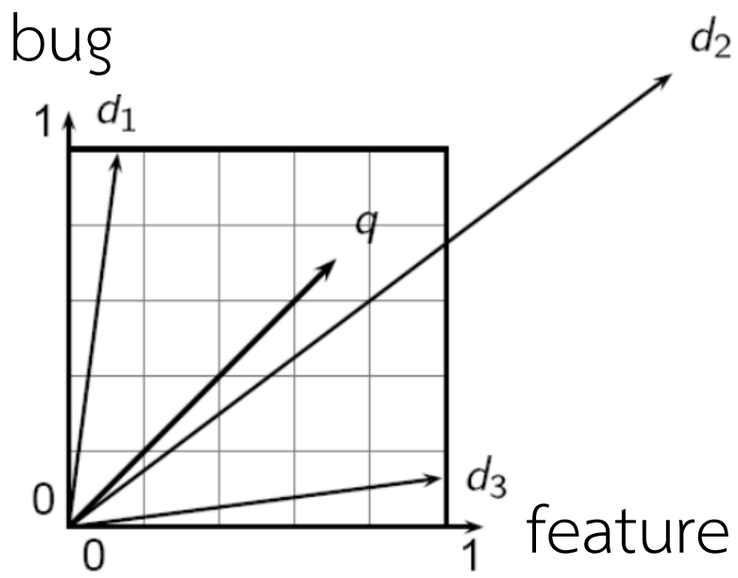
- Discuss about distance function
- The most commonly employed distance function is the Euclidean distance.
  - However, it is not entirely suitable for this context.



$$D(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$$

Why not Euclidean distance?

- Euclidean distance tends to be large when dealing with vectors of varying lengths.
  - This limitation becomes more evident in high-dimensional spaces.
  - E.g., the distance between q and d_2 can be large
    - even if the term distributions in q and d_2 are very similar.
    - when there are significant differences in frequencies.

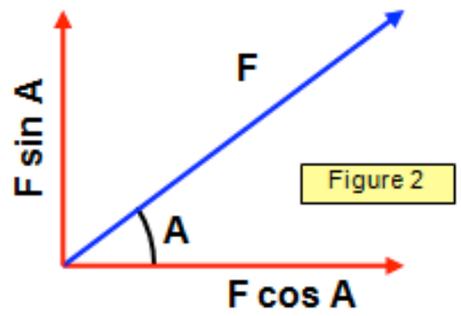


Then, what shall be used?

- The angle between vectors appears to convey more meaningful information than distance.
  - For instance, consider duplicating a document d and appending its content to itself, creating a modified document d'.
    - Both d and d' contain identical content.
    - However, the Euclidean distance between d and d' would be large.
- In contrast, the angle between the two documents is 0, indicating maximum similarity.

Vectors and angle

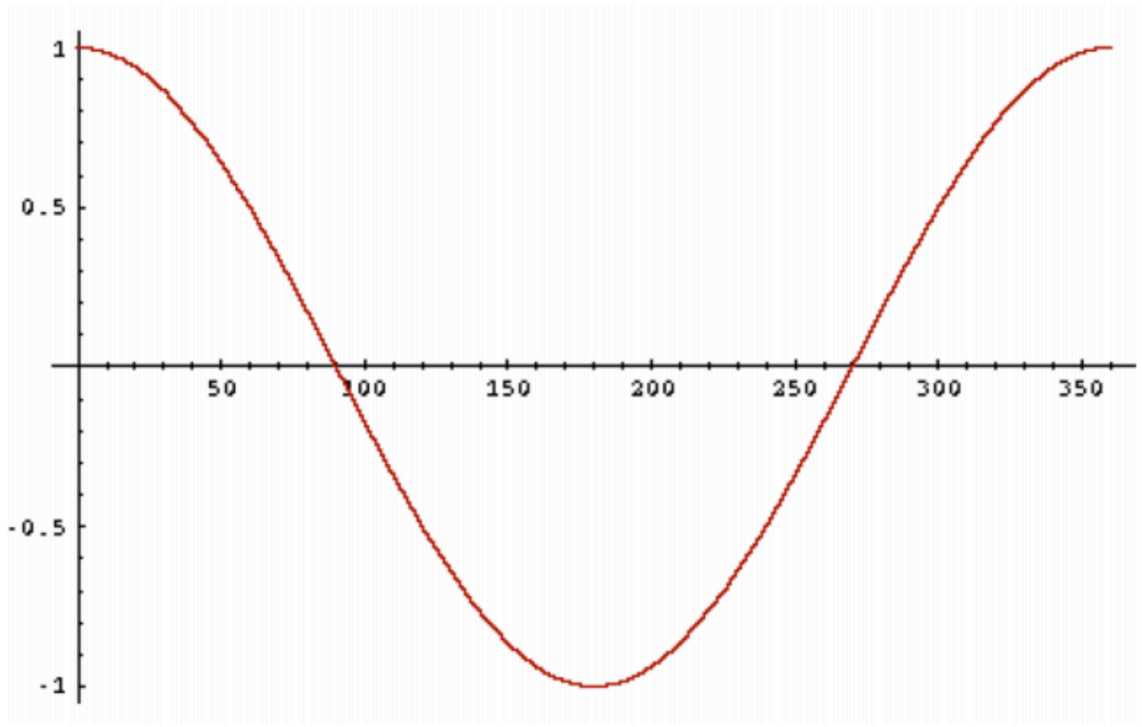
- Cosines
  - Rank documents by arranging them in **descending** order of the angle between the query and document vectors.
  - Alternatively, rank documents by arranging them in ascending order of the cosine similarity value between q and d.
  - The dot product can be meaningfully utilized in these calculations.



Ref: https://www.schoolphysics.co.uk/age16-19/Mechanics/Statics/text/Components_of_vectors/index.html

Vectors and angle

- From angles to cosines

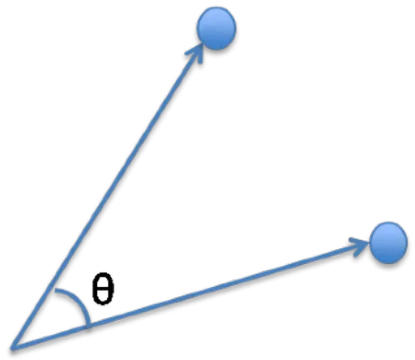


Ref: https://halalhassan.wordpress.com/2013/08/31/cosine-similarity-in-mapreduce-hadoop/

Vectors and angle

- From angles to cosines

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



- Length normalization
  - To length-normalize a vector, each of its components is divided by its length.

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Ref: https://halalhassan.wordpress.com/2013/08/31/cosine-similarity-in-mapreduce-hadoop/

Vectors and angle

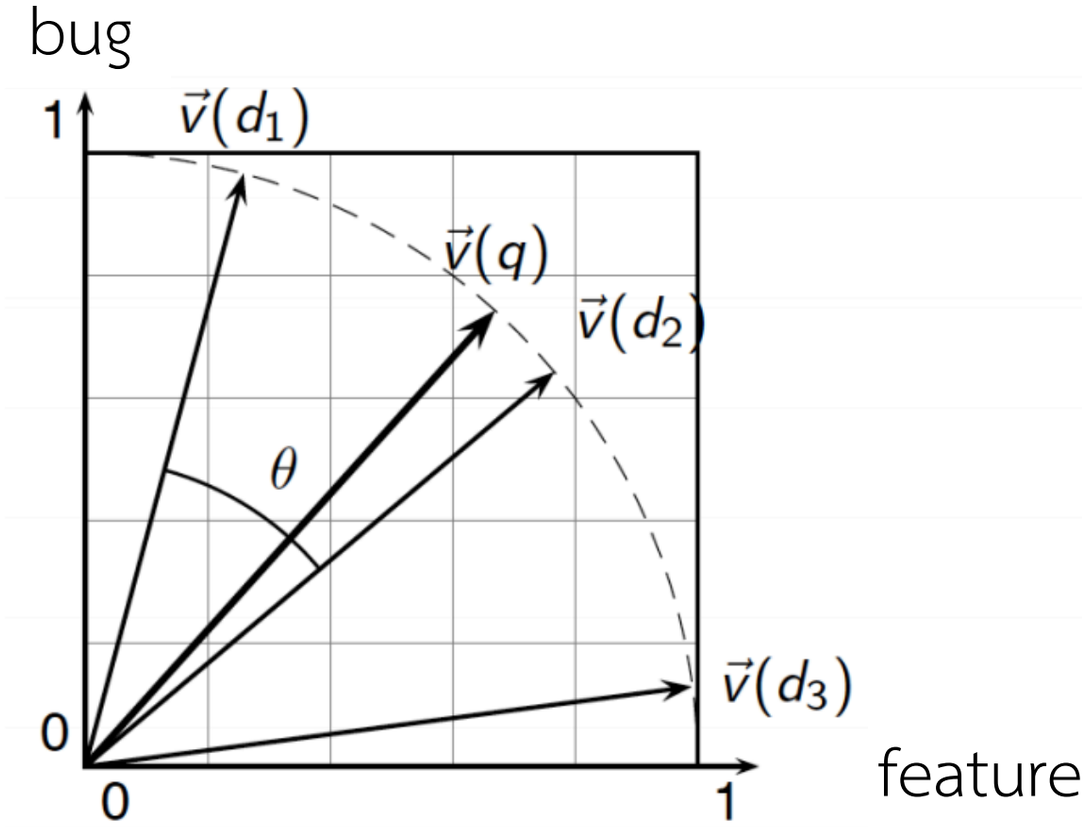
- *cosine*$_{(q,d)}$

$$cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- For length-normalized vectors, cosine similarity is equivalent to the dot product (scalar product).

$$cos(\vec{q},\vec{d}) = \vec{q}\cdot\vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

Cosine similarity illustrated



More examples

- How similar are these three books
  - DevOps Handbook
  - Continuous Delivery
  - Distributed Computing

	business	computer	git	parallel
DevOpsHandbook	100	200	200	50
ContinuousDelivery	90	200	300	0
DistributedComputing	5	200	10	200

tf (counts)

Let’s simplify by ignoring idf in this example

$$log_{10}(1 + tf)$$

	business	computer	git	parallel
DevOpsHandbook	2.004321	2.303196	2.303196	1.707570
ContinuousDelivery	1.959041	2.303196	2.303196	0.000000
DistributedComputing	0.778151	2.303196	1.041393	2.303196

dot(DH,CD) ≈ 14.94

dot(DH,DC) ≈ 13.19

dot(CD,DC) ≈ 9.410

```
1   arr = np.array([[100, 200, 200, 50], [90, 200, 300, 0], [5, 200, 10, 200]])
2
3   data = pd.DataFrame(arr, index=['DevOpsHandbook', 'ContinuousDelivery',
'DistributedComputing'], columns=['business', 'computer', 'git', 'parallel'])
4   data = np.log10(data + 1)
```

```
1   print(data.loc['DevOpsHandbook'].dot(data.loc['ContinuousDelivery']))
2   print(data.loc['DevOpsHandbook'].dot(data.loc['DistributedComputing']))
3   print(data.loc['ContinuousDelivery'].dot(data.loc['DistributedComputing']))
```

Let’s simplify by ignoring idf in this example

Length normalization

	business	computer	git	parallel
DevOpsHandbook	0.478543	0.549901	0.549901	0.407692
ContinuousDelivery	0.501071	0.589096	0.633951	0.000000
DistributedComputing	0.221882	0.656732	0.296942	0.656732

```
1   data.loc['DevOpsHandbook'] /= np.sqrt((data.loc['DevOpsHandbook'] ** 2).sum())
2   data.loc['ContinuousDelivery'] /= np.sqrt((data.loc['ContinuousDelivery'] ** 2).sum())
3   data.loc['DistributedComputing'] /= np.sqrt((data.loc['DistributedComputing'] ** 2).sum())
4   print(data.to_markdown())
```

With normalized length

Dot product

	business	computer	git	parallel
DevOpsHandbook	0.478543	0.549901	0.549901	0.407692
ContinuousDelivery	0.501071	0.589096	0.633951	0.000000
DistributedComputing	0.221882	0.656732	0.296942	0.656732

```
1   print(data.loc['DevOpsHandbook'].dot(data.loc['ContinuousDelivery']))
2   print(data.loc['DevOpsHandbook'].dot(data.loc['DistributedComputing']))
3   print(data.loc['ContinuousDelivery'].dot(data.loc['DistributedComputing']))
```

Scikit-learn's builtin tf-idf

```
1   tf_idf_vectorizer = TfidfVectorizer(preprocessor=my_custom_preprocessor, use_idf=True)
2   tf_idf_vectorizer.fit(cleaned_description)
3   transformed_data = tf_idf_vectorizer.transform(cleaned_description)
4   X_tfidf_df = pd.DataFrame(transformed_data.toarray(),
columns=tf_idf_vectorizer.get_feature_names_out())
5   max_term = X_tfidf_df.sum().sort_values()[-10:].sort_index().index
6   X_tfidf_df[max_term].head(5)
```

	applic	design	develop	provid	requir	system	team	technolog	test	work
0	0.044218	0.046397	0.057155	0.030353	0.028125	0.022774	0.026645	0.000000	0.078244	0.020490
1	0.031692	0.033254	0.054619	0.000000	0.000000	0.032645	0.038194	0.000000	0.000000	0.029371
2	0.047755	0.000000	0.041152	0.032782	0.030375	0.024596	0.000000	0.027089	0.056336	0.044258
3	0.000000	0.000000	0.056202	0.000000	0.041483	0.000000	0.000000	0.036995	0.038469	0.030222
4	0.031692	0.033254	0.054619	0.000000	0.000000	0.032645	0.038194	0.000000	0.000000	0.029371

Small workout

- Compare the differences between **CountVectorizer** and **TfidfVectorizer**
  - Based on the **entire document collection** and find 5 documents most relevant with the **Document** at row #0
  - Use the entire document collection and find 5 documents most relevant with the query **aws devops**

Summary – vector space ranking

- Represent the **query** as a weighted *tf-idf* vector.
- Represent each **document** as a weighted *tf-idf* vector.
- Calculate the **cosine similarity** score between the query vector and each document vector.
- **Rank** documents in descending order based on their similarity scores.
- Return the **top K** (e.g., K = 5) documents to the user.

Beyond *tf-idf*

- Probability ranking
  - Estimate the likelihood of a document being relevant to a query term.
- Language model
  - Combines multiple models to improve retrieval.
  - Incorporates document language models with Document Priors, Relevance Models, Translation Models, and Aspect Models.
- Google PageRank
  - A system for ranking web pages based on authority.
  - Utilizes a link analysis algorithm that assigns a weight to each element of a hyperlinked set of documents.

Probability ranking

- Probability ranking
  - Estimate the likelihood of a document being relevant to a query term.
  - **Question:** In what order do we present documents to the user?
    - Sequence documents by their **best** to **least** relevance.
    - **Ranking Concept:** Documents are ranked based on the probability that they meet the user's **information need**.

Probability ranking — principle

- Ranked Responses
  - Information Retrieval (IR) systems respond to queries by ranking documents.
  - Documents are ordered by decreasing likelihood of relevance to the user's request.

Recall a few probability basics

- For events A and B:

$$p(A, B) = p(A \cap B) = p(A|B)p(B) = p(B|A)p(A)$$

- Bayes's rule:

prior

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} = \frac{p(B|A)p(A)}{\sum_{X=A,\overline{A}} p(B|X)p(X)}$$

posterior

- Odds:

$$O(A) = \frac{p(A)}{p(\overline{A})} = \frac{p(A)}{1 - p(A)}$$

Probability ranking — principle

- Let ***d*** represent a document in the collection
- Let ***R*** be the level of **relevance** of a document with regard to the given query, **where R=1 means relevant and R=0 mean not relevant**

- What to find $p(R=1|d)$ — from $p(A|B) = \frac{p(B|A)p(A)}{p(B)} = \frac{p(B|A)p(A)}{\sum_{X=A,\overline{A}} p(B|X)p(X)}$

$$p(R=1|d) = \frac{p(d|R=1)p(R=1)}{p(d)}$$

$$p(R=0|d) = \frac{p(d|R=0)p(R=0)}{p(d)}$$

$$p(R=0|d) + p(R=1|d) = 1$$

- p(R=1) and p(R=0) — prior probability of retrieving a relevant or non-relevant document at random
- p(x|R=1) and p(x|R=0) — probability that if a relevant (not relevant) document is retrieved.

Probabilistic retrieval strategy

- Relevance Contribution
  - Assess the contribution of each term to the document's relevance.
- Combining Relevance Signals
  - Calculate the document's overall probability of relevance by combining term contributions.
- Document Ordering
  - Arrange documents by their relevance probability in descending order.
- Error Minimization
  - Aim to minimize the error in relevance judgment, reducing the risk of irrelevant retrieval.

Traditional — Binary independence model

- **Binary = Boolean**: documents are represented as binary incidence vectors of terms … the simple representation
- $\vec{x} = (x_1, ..., x_n)$
- $x_i = 1$ **if and only if** term *i* is present in document *x*
- Queries: binary term incidence vectors

Traditional — Binary independence model

- Given query ***q***,
  - Fore each document ***d***, compute p(*R*/*q*, ***x***) where ***x*** is binary term incidence vector representing ***d***
  - Applying the combined **odds and Bayes' rule**:

$$O(R|q,\vec{x}) = \frac{p(R=1|q,\vec{x})}{p(R=0|q,\vec{x})} = \frac{\frac{p(R=1|q)p(\vec{x}|R=1,q)}{p(\vec{x}|q)}}{\frac{p(R=0|q)p(\vec{x}|R=0,q)}{p(\vec{x}|q)}}$$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = \frac{p(R=1|q,\vec{x})}{p(R=0|q,\vec{x})} = \frac{p(\vec{x}|R=1,q)}{p(\vec{x}|R=0,q)}$$

Need an estimation

- Assume that all terms are **independent**

$$\frac{p(\vec{x}|R=1,q)}{p(\vec{x}|R=0,q)} = \prod_{i=1}^{n} \frac{p(x_i|R=1,q)}{p(x_i|R=0,q)}$$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{i=1}^{n} \frac{p(x_i|R=1,q)}{p(x_i|R=0,q)}$$

Traditional — Binary independence model

- It can be further simplified since x_i can be only 0 or 1.

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{x_i=1} \frac{p(x_i=1|R=1,q)}{p(x_i=1|R=0,q)} \cdot \prod_{x_i=0} \frac{p(x_i=0|R=1,q)}{p(x_i=0|R=0,q)}$$

- Let $p_i = p(x_i=1|R=1,q); r_i = p(x_i=1|R=0,q);$

- Assume, for all terms not occurring in the query ($q_i = 0$) $p_i = r_i$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

	Document	Relevant (R=1)	Not relevant (R=0)
Term present	$x_i = 1$	p_i	r_i
Term absent	$x_i = 0$	$(1 - p_i)$	$(1 - r_i)$

Traditional — Binary independence model

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{x_i=q_i=1} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

All matching terms

Non-matching query terms

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \left(\frac{1-r_i}{1-p_i} \cdot \frac{1-p_i}{1-r_i}\right) \prod_{\substack{x_i=0 \\ q_i=1}} \frac{1-p_i}{1-r_i}$$

$$O(R|q,\vec{x}) = O(R|q) \cdot \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \cdot \prod_{q_i=1} \frac{1-p_i}{1-r_i}$$

All matching terms

All query terms

Traditional — Binary independence model

Constant for each query

$$O(R|q,\vec{x}) = O(R|q) \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \prod_{q_i=1} \frac{1-p_i}{1-r_i}$$

Only quantity to be estimated for rankings

- Residual status value

$$RSV = log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \approx \sum_{x_i=q_i=1} log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Traditional — Binary independence model

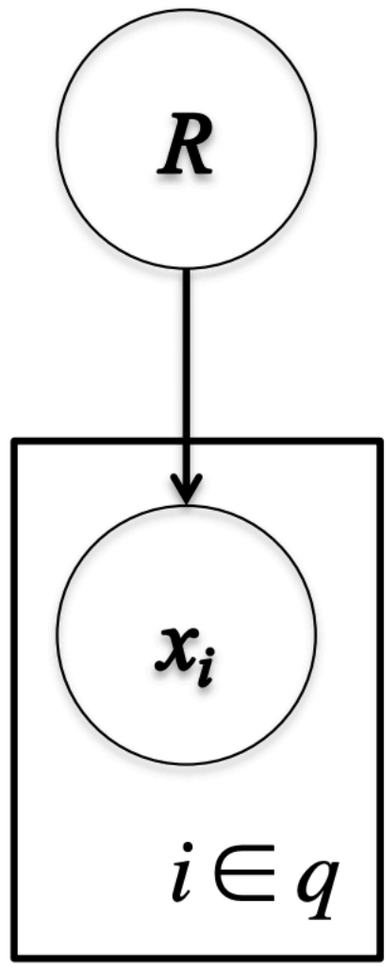
- Stripping all the constants, all we need to do is to compute the Retrieval Status Value (RSV)

$$RSV = log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \approx \sum_{x_i=q_i=1} log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$RSV = \sum_{x_i=q_i=1} c_i; c_i = log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

- The c_i are log odds ratios
- c_i functions as the term weights in this model

Graphical model for BIM



Binary variables

$$x_i = (tf_i \neq 0)$$

Binary independence model

- Estimating RSV coefficients in theory

$$c_i = log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

- For each term i look at the below table of document counts

Documents	Relevant	Non-relevant	Total
$x_i = 1$	s	n - s	n
$x_i = 0$	S - s	N - n - S + s	N - n
Total	S	N - S	N

- Estimates: $p_i \approx \frac{s}{S}$ $r_i \approx \frac{n-s}{N-S}$

$$c_i \approx K(N,n,S,s) = log \frac{s/(S-s)}{(n-s)/(N-n-S+s)}$$

Estimation

- If non-relevant documents are approximated by the whole collection, then r_i (the probability of occurrence in **non-relevant documents** for query) is n/N and

$$log\frac{1-r_i}{r_i} = log\frac{N-n-S+s}{n-s} \approx log\frac{N-n}{n} \approx log\frac{N}{n} = IDF$$

Estimation — on the other hand

- p_i (the probability of occurrence in **relevant documents** for query) cannot be approximated easily
- Various approaches to estimate p_i
  - Applying relevance weighting
    - constant, e.g., $p_i = 0.5$
    - proportional to the probability of occurrence in collection

Okapi BM25

- **BM** signifies Best match; **25** refers to the version number, from iterative testing.
- Contemporary Standard
  - Regarded as the conventional standard for probabilistic IR models today.
- Core Principle
  - Balances term frequency and document length without overcomplicating the model with excessive parameters.

Generative model for documents

- Words are drawn independently from the vocabulary using a multinomial distribution

Ref: https://dataaspirant.com/bag-of-words-bow/

Okapi BM25

- Distribution of term frequencies (*tf*) follows a binomial distribution – approximated by a Poisson
  - Binomial distribution $P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$
  - Poisson distribution $P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$
- Recommended ref — http://makemeanalyst.com/normal-distribution-binomial-distribution-poisson-distribution/ and https://towardsdatascience.com/poisson-process-and-poisson-distribution-in-real-life-modeling-peak-times-at-an-ice-cream-shop-b61b74fb812

Binomial

- n — repeated experiments
- x — success trial form n trials
- p — probability of success

$$P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$$

- Example — calculate the following probabilities that a student randomly answers 3 exam questions without reading the questions. All of which is a 4-multiple choice.
  - All correct
  - 2 correct
  - 1 correct
  - All fail

Binomial

- All correct — p(TTT) = 0.25^3 = 0.02
- Two correct — p(TTF) + p(TFT) + p(FTT) = (0.25 * 0.25 * 0.75) * 3 = 0.14
- One correct — p(FFT) + p(FTF) + p(TFF) = (0.75 * 0.75 * 0.25) * 3 = 0.42
- All fail — p(FFF) = 0.75^3 = 0.42

$$P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!x!} p^x q^{n-x}$$

Poisson model

- Can estimate a binomial distribution when event occurrences are rare.
- Calculates the probability of **x** events in a set period, given a known average rate **λ** (where **λ** = cf/T), independent of previous events.
- **Practical applications**
  - Useful for estimating occurrences, such as the number of customers in a given time frame by observing a shorter interval.

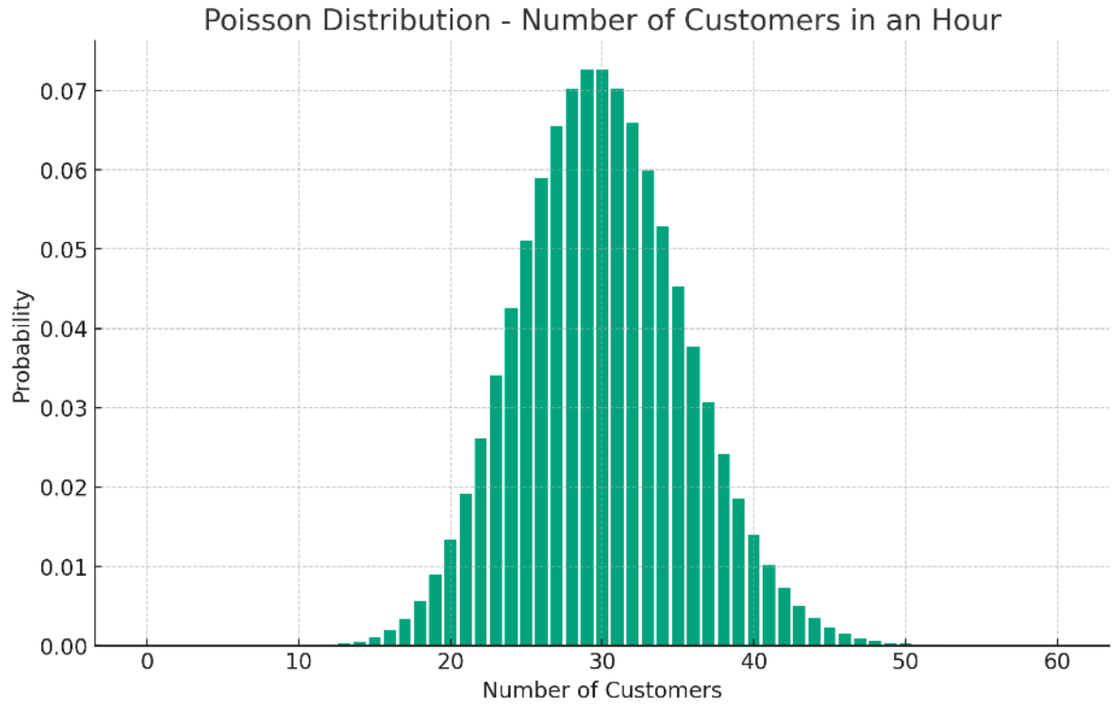
$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Poisson model — application example

- Let's assume that on average, your observed that your tester team finds 5 bugs every 10 minutes.
- We can use this rate (λ) to estimate the number of bug in an hour (which is six 10-minute intervals). = 30 bugs per hour.

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

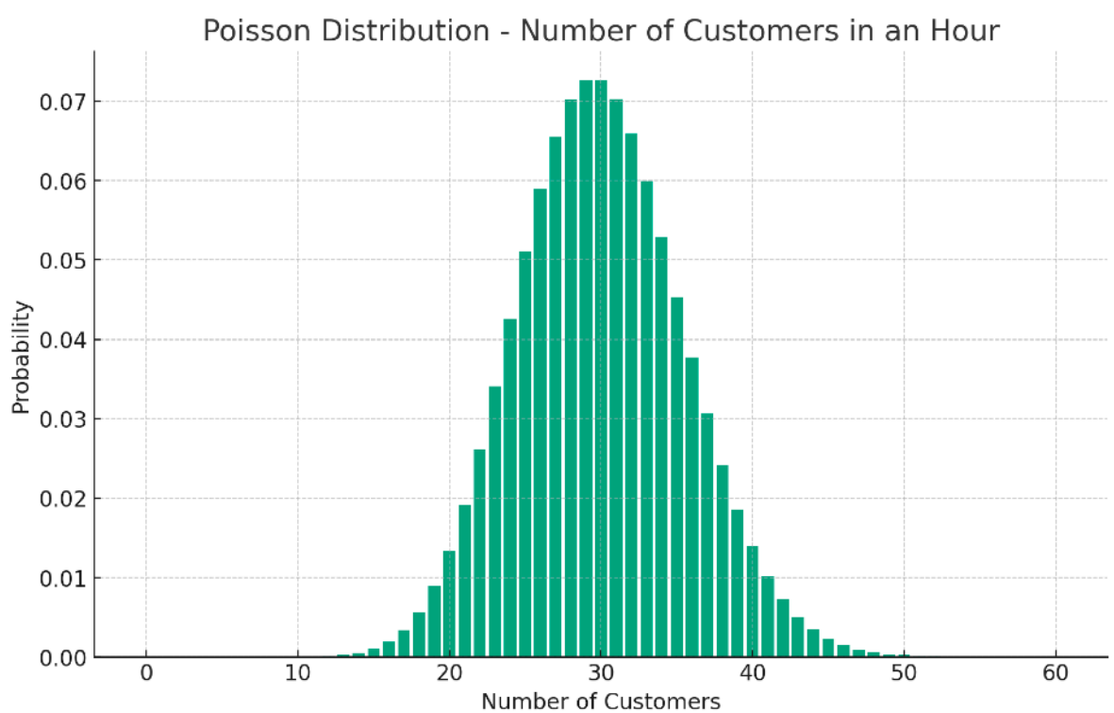
- ~7.2% chance to see an exact number of 30 bugs per hour.



Poisson model — application example

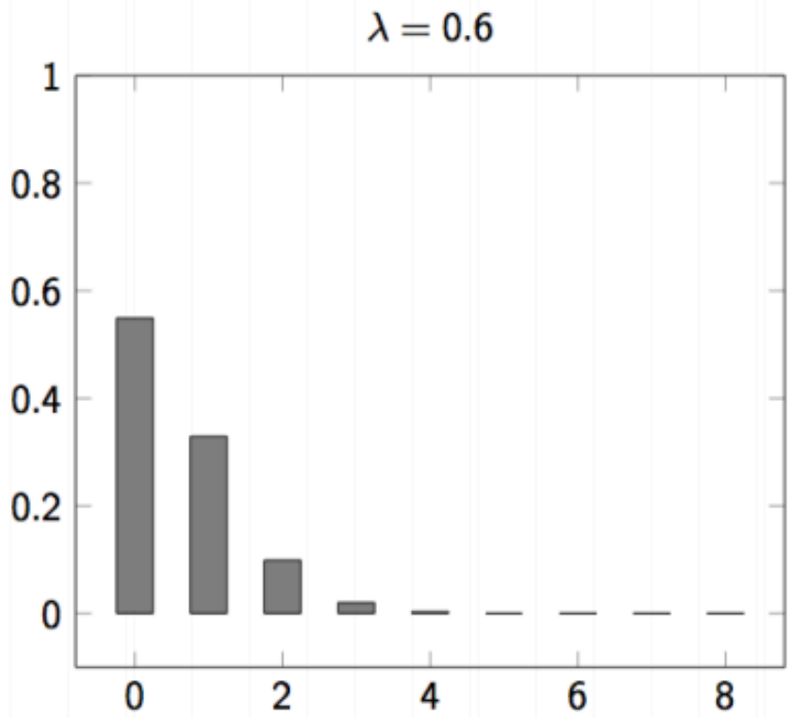
- E.g., by using the cumulative distribution function (CDF) of the Poisson distribution, we can determine that there is a **45.16%** chance that the team will find more than **30** bugs in an hour, based on the average rate observed;
- However, the probability of there being more than **40** bugs in an hour is just around **3.23%.**

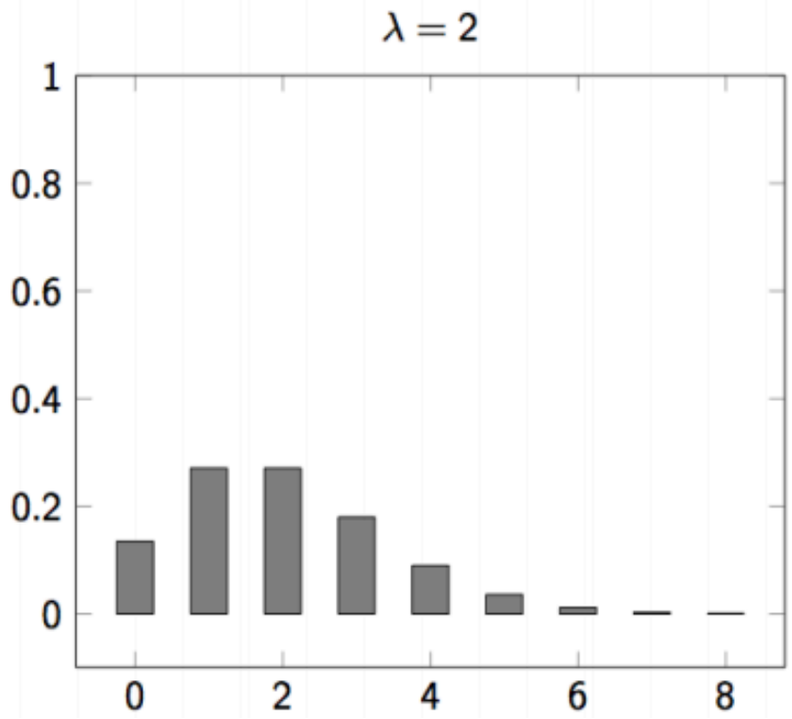
$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$



Poisson-distributed term frequencies

- Assume that term frequencies within a document (tf_i) adhere to a Poisson distribution.
  - **Fixed interval** implies fixed document length





Poisson model — Application example

		Documents containing *k* occurrences of word (λ = 53/650)												
Freq	**Word**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
53	expected	599	49	2										
52	*based*	600	48	2										
53	*conditions*	604	39	7										
55	*cathexis*	619	22	3	2	1	2	0	1					
51	*comic*	642	3	0	1	0	0	0	0	0	0	1	1	2

Eliteness

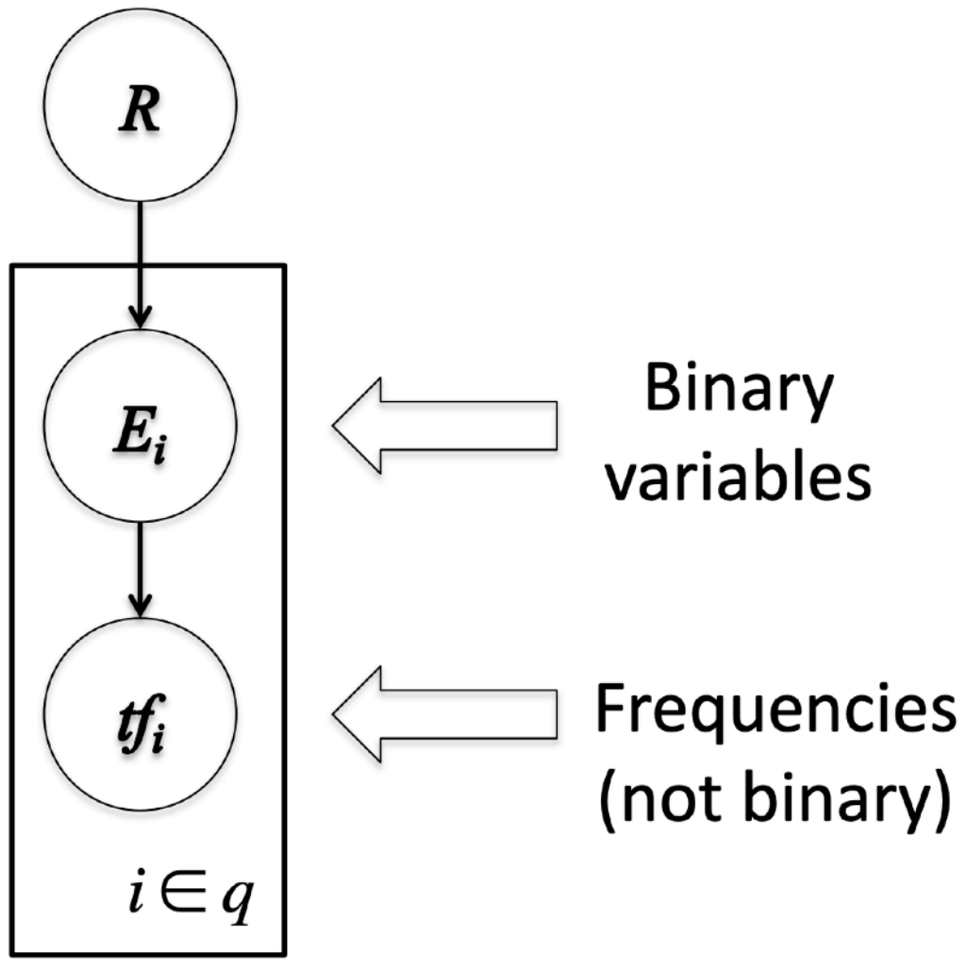
- Term frequencies are indicative of a term's eliteness within a document.
- Understanding Eliteness
  - Eliteness is a hidden variable for each document-term pair, represented as E_i for term *i*
  - It signifies the **aboutness** of a term
    - a document is considered 'elite' for a term if it is significantly about that term's concept.

Elite terms

In software engineering, a **design pattern** is a general repeatable **solution** to a commonly occurring **problem** in **software design**. A **design pattern** isn't a finished **design** that can be transformed directly into **code**. It is a **description** or **template** for how to **solve** a **problem** that can be used in many different situations.

Ref: https://sourcemaking.com/design_patterns

Graphical model with eliteness



Utilizing RSV

$$RSV^{elite} = \sum_{i \in q, tf_i > 0} c_i^{elite}(tf_i)$$

- where $c_i^{elite}(tf_i) = log \frac{p(TF_i = tf_i|R = 1)p(TF_i = 0|R = 0)}{p(TF_i = 0|R = 1)p(TF_i = tf_i|R = 0)}$

- Then, utilizing eliteness we will have:

$$p(TF_i = tf_i|R) = p(TF_i = tf_i|E_i = elite)p(E_i = elite|R)$$
$$+p(TF_i = tf_i|E_i = \overline{elite})(1 - p(E_i = elite|R)$$

2-Poisson model

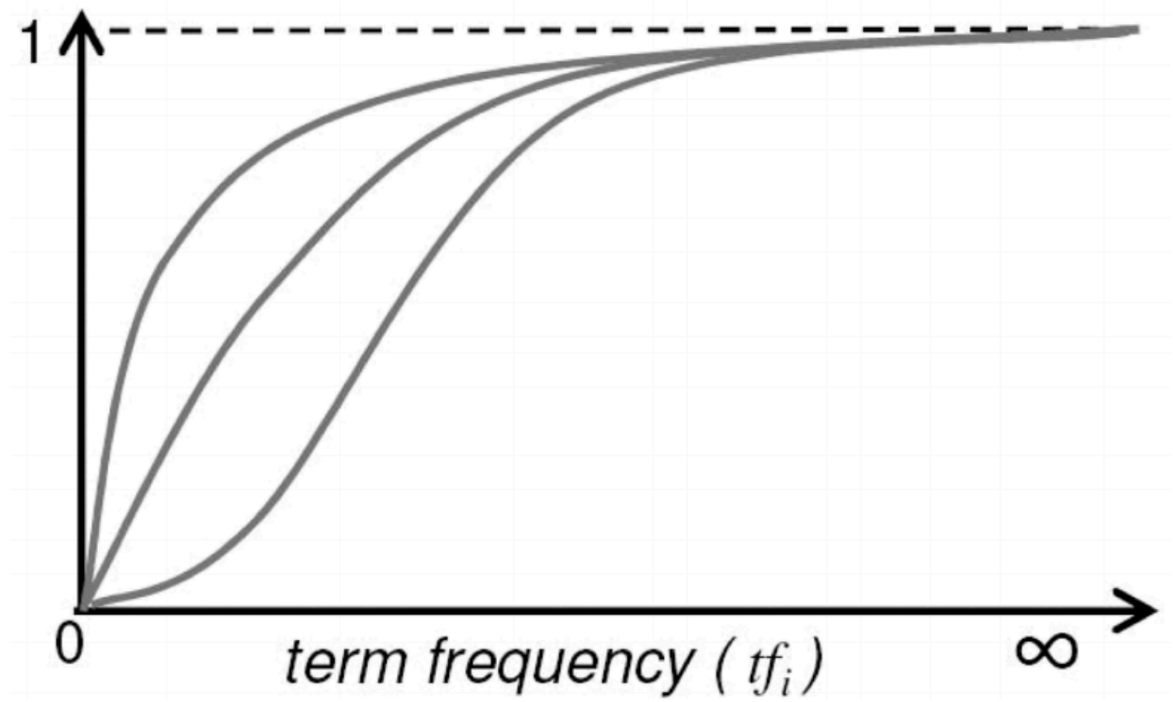
- The distribution of the elite terms and non-elite terms are not the same.

$$p(TF_i = k_i|R) = \pi \frac{\lambda^k}{k!} e^{-\lambda} + (1-\pi) \frac{\mu^k}{k!} e^{-\mu}$$

- π is the probability that the document is elite for term
- We don't know π, λ, μ

Observing the behavior

$$c_i^{elite}(tf_i)$$

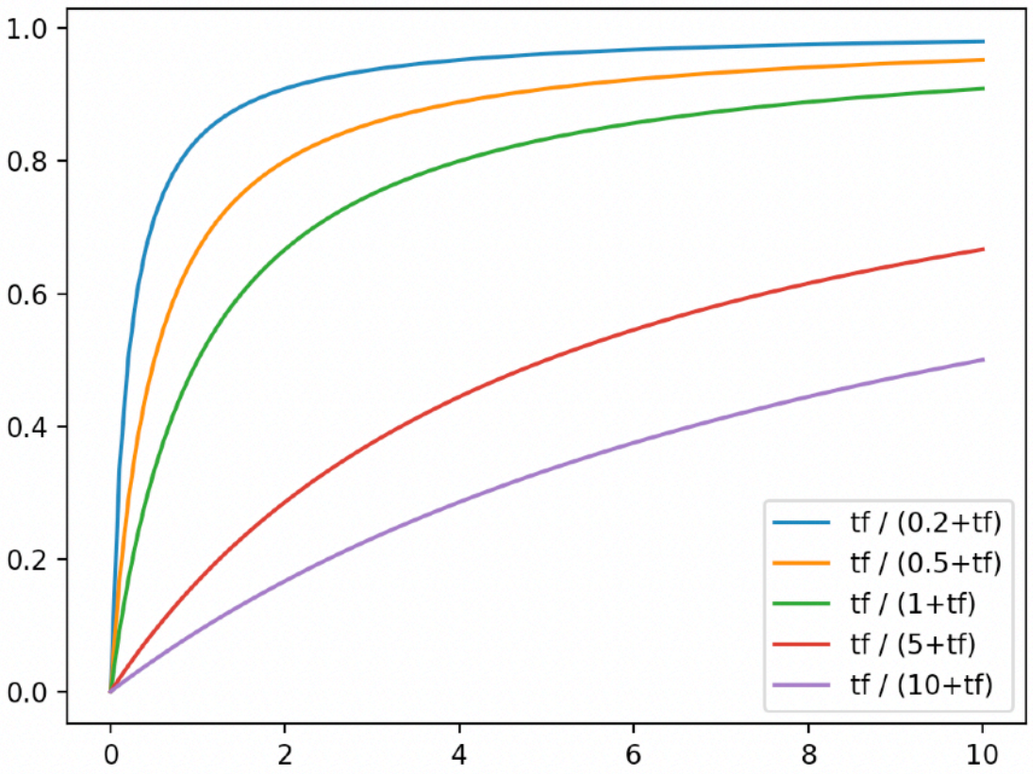


2-Poisson model

- Precisely simulating the relation graph for eliteness is complex.
- Employ an equation with similar qualitative properties to approximate the graph.
- This approach captures the essence of the eliteness concept without detailed simulation.

$$\frac{tf}{k_1 + tf}$$

Saturation function




```
1   import matplotlib.pyplot as plt
2   tf = np.linspace(0, 10, 100)
3   for k in [0.2, 0.5, 1, 5, 10]:
4       plt.plot(tf, tf / (k+tf), label="tf / ("+str(k)+"tf)"
5   plt.legend()
6   plt.show()
```

- A larger k_1 means tf_i continues to have a significant impact on the document's score.
- A smaller k_1 the contribution of tf_i to the score diminishes more rapidly.

Creating BM25

- The first version: just simply using the saturation function

$$c_i^{BM25v1}(tf_i) = c_i \frac{tf_i}{k_1 + tf_i}$$

- The second version: simplification to IDF

$$c_i^{BM25v2}(tf_i) = log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1 + tf_i}$$

  - (k_1+1) does not change raking, but to make the term score = 1 when tf_i = 1.
  - One significant merit is that the term score are bounded.

Document length normalization

- Typically, larger documents are likely to have larger tf_i score
- Document length:

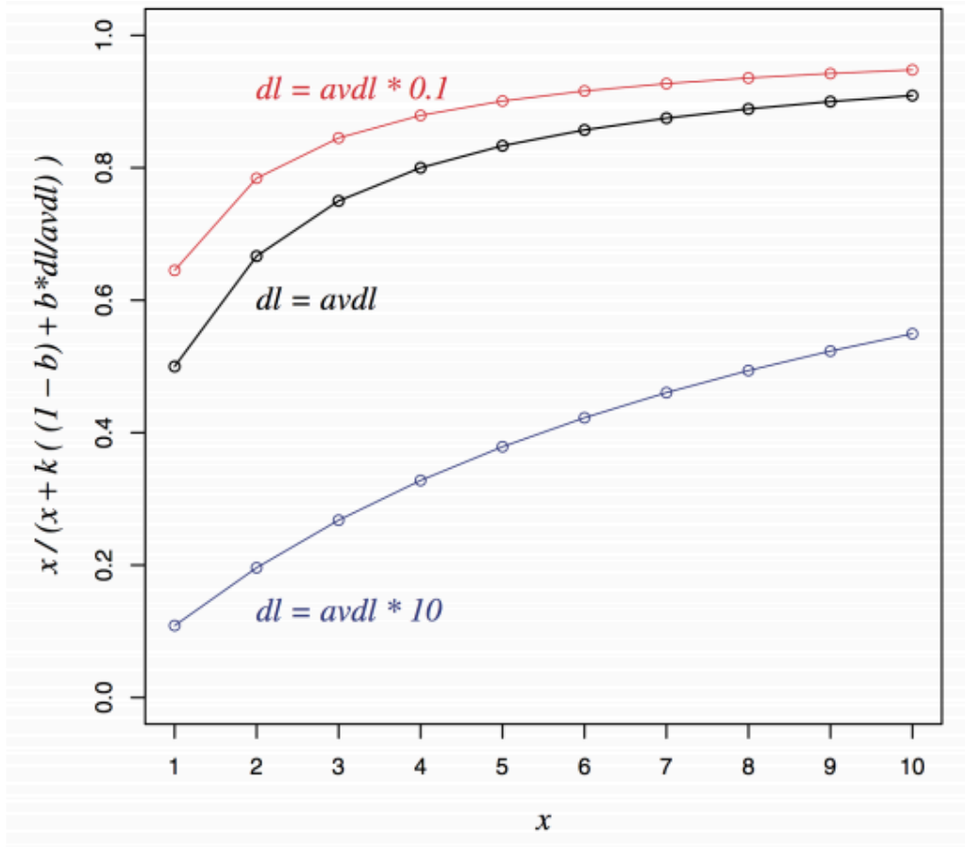
$$dl = \sum_{i \in V} tf_i$$

- *avdl*: Average document length over collection
- Length normalization component:

$$B = \left((1-b) + b \frac{dl}{avdl} \right), 0 \leq b \leq 1$$

  - b = 1: full document length normalization
  - B = 0: no document length normalization

Document length normalization



Okapi BM25

- Normalize *tf* using document length

$$tf_i' = \frac{tf_i}{B}$$

$$c_i^{BM25}(tf_i) = log\frac{N}{df_i} \times \frac{(k_1+1)tf_i'}{k_1 + tf_i'}$$

$$= log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1\left((1-b) + b\frac{dl}{avdl}\right) + tf_i}$$

- BM25 ranking function

$$RSV^{BM25} = \sum_{i \in q} c_i^{BM25}(tf_i) \quad = \sum_{i \in q} log\frac{N}{df_i} \cdot \frac{(k_1+1)tf_i}{k_1\left((1-b) + b\frac{dl}{avdl}\right) + tf_i}$$

Okapi BM25

$$RSV^{BM25} = \sum_{i \in q} log \frac{N}{df_i} \cdot \frac{(k_1 + 1) tf_i}{k_1 \left((1 - b) + b \frac{dl}{avdl} \right) + tf_i}$$

- **k_1** controls term frequency scaling
  - k_1 = 0 is binary model; k_1 large is raw term frequency
- **b** controls document length normalization
  - b = 0 is no length normalization; b=1 is relative frequency
- Typically, **k_1** is set around 1.2 - 2 and **b** is around 0.75 (from brute force?)

BM25

```python
class BM25(object):
    def __init__(self, vectorizer, b=0.75, k1=1.6):
        self.vectorizer = vectorizer
        self.b = b
        self.k1 = k1

    def fit(self, X):
        """ Fit IDF to documents X """
        self.vectorizer.fit(X)
        self.y = super(TfidfVectorizer, self.vectorizer).transform(X)
        self.avdl = self.y.sum(1).mean()

    def transform(self, q):
        """ Calculate BM25 between query q and documents X """
        b, k1, avdl = self.b, self.k1, self.avdl

        # apply CountVectorizer
        len_y = self.y.sum(1).A1
        q, = super(TfidfVectorizer, self.vectorizer).transform([q])
        assert sparse.isspmatrix_csr(q)

        # convert to csc for better column slicing
        y = self.y.tocsc()[:, q.indices]
        denom = y + (k1 * (1 - b + b * len_y / avdl))[:, None]
        idf = self.vectorizer._tfidf.idf_[None, q.indices] - 1.
        numer = y.multiply(np.broadcast_to(idf, y.shape)) * (k1 + 1)
        return (numer / denom).sum(1).A1
```

Modified from

https://gist.github.com/koreyou/f3a8a0470d32aa56b32f198f49a9f2b8

BM25

```
cleaned_description = m1.get_and_clean_data()
bm25 = BM25(tf_idf_vectorizer)
bm25.fit(cleaned_description)
```

```
score = bm25.transform('aws devops')
rank = np.argsort(score)[::-1]
print(cleaned_description.iloc[rank[:5]].to_markdown())
```

```
score = bm25.transform('aws github')
rank = np.argsort(score)[::-1]
print(cleaned_description.iloc[rank[:5]].to_markdown())
```

More GitHub terms appear

```
score = bm25.transform('aws github')
rank = np.argsort(score)[::-1]
print(cleaned_description.iloc[rank[:5]].to_markdown())
```

```
-------------------------------------------------------------------------------------------------------------------
production with docker in aws  knowledge of visualizing data  familiar with github     what you didn t know abo
r scalaapiseligibility for clearancenice to have • hadoop • spark • kafka • elastic search • sql • git  github •
gy  strong team player excellent communication skills and the ability to work independently and in a teambased e
ple code on github  an autonomous selfdriven and positive attitude  strong skills in javascript es6 andor typesc
m architecture solve complex business problems and extend business functionality  qualifications  qualifications
```

Summary

- Some histories
- Scoring documents
- Term frequency (tf)
- Vector space model
- Probabilistic model

Assignment

Create a **simple command-line-interface application** that search the job description data

- Use a Python library like **argparse** to create a **command-line interface application** that accepts a search query.
- Preprocess the text data to create 2-grams (bigrams) for each document.
- Index these bigrams for efficient searching.
- Implement *tf-idf* and *bm25* scoring functions.
- For each query, calculate the scores for all documents and sort them, displaying the top 5 results for each scoring mechanism. **(2 points)**
- Provide an in-depth analysis of how *bm25* scoring may provide better results in certain contexts due to its sophisticated scoring mechanism. **(2 point)**
- There is a python library named **rank_bm25** that provides incorrect bm25 results, identify the issues and modify the library code or provide a wrapper that corrects the results **(1 point)**

Time for questions