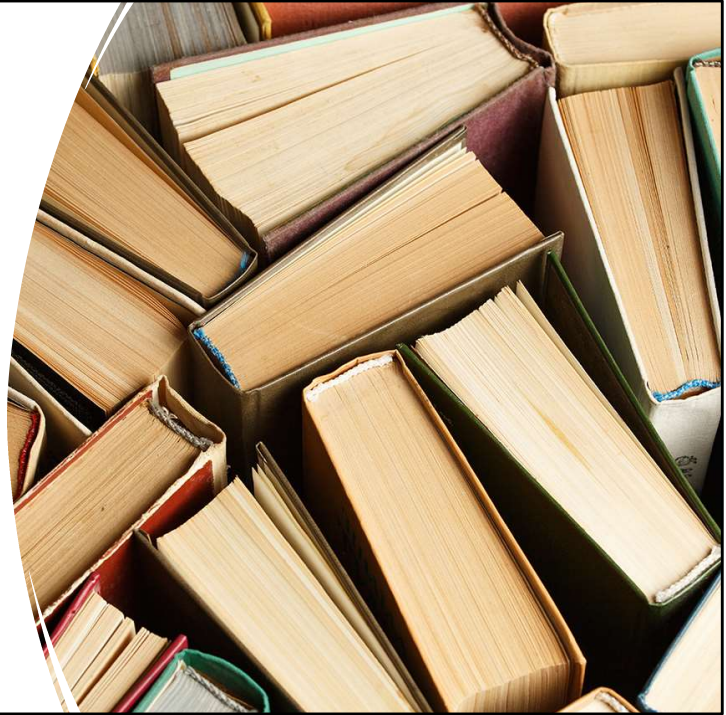


Data-Intensive
Distributed
Computing
CS431/451/631/651

Module 4 – Analysing Text



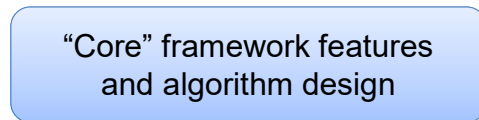
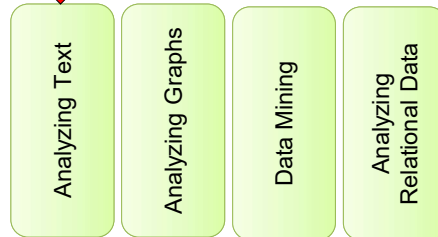
This Module's Agenda

Language Models

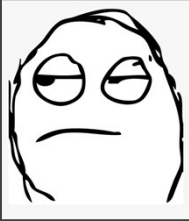
Natural Language Processing

Information Retrieval / Search

Structure of the Course



This all
seems
LAME!



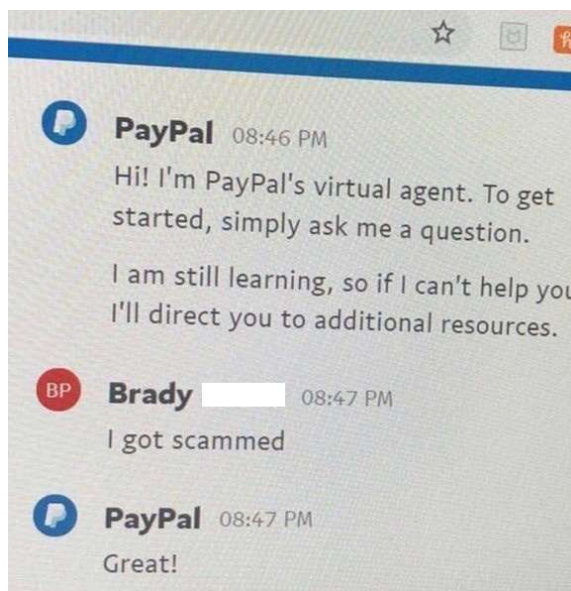
First of all, how dare you

Walk before you run

Language Models are about so
much more than Word Counts and
PMI!

Natural Language Processing

NLP for short



Probabilistic Model

$$P(w_1, w_2, \dots, w_k)$$

– The probability of encountering the sentence $w_1 w_2 \dots w_k$

What good is this?

- Machine Translation

$$P(\text{"High winds expected"}) > P(\text{"Large winds expected"})$$

- Spell Checker that's not fooled by homophones

$$P(\text{"Waterloo is a great city!"}) > P(\text{"Waterloo is a grate city!"})$$

- Speech Recognition

$$P(\text{"I saw a van"}) > P(\text{"Eyes awe of an"})$$

We want to be able to take a sentence with k words, and assign a probability to it. This lets us rank alternatives. Of course this doesn't tell us WHERE those alternatives even came from, but one step at a time!

Probabilistic Model

$$P(w_1, w_2, \dots, w_k)$$

– The probability of encountering the sentence $w_1 w_2 \dots w_k$

How LLMs work*:

1. Given $w_1 w_2 \dots w_{k-1}$ obtain probability distribution for w_k
2. Sample word from distribution
3. Repeat until you generate the special “stop” word.

* Basically

The tricky part is generating the probability distribution, of course! And...there are a lot of different sampling techniques to choose from.

Again, not an AI course so we won't get into too much detail

Probabilistic Model

$$P(w_1, w_2, \dots, w_k) =$$

$$P(w_1) \times P(w_2, \dots, w_k | w_1) =$$

...

$$P(w_1) \times P(w_2 | w_1) \times \dots \times P(w_k | w_1, w_2, \dots, w_{k-1})$$

$$P(\text{"I saw a van"}) = P(\text{"I"}) \times P(\text{"saw"} | \text{"I"}) \times P(\text{"a"} | \text{"I saw"}) \times$$

$$P(\text{"van"} | \text{"I saw a"})$$

Q: Can we actually use this?

Chain rule – $P(A, B) = P(B) \times P(A | B)$ – Like with PMI

Question: Is this reasonable? How long is a typical sentence? 15-20 words in modern writing. 70+ in ye olden times. PMI took us two passes, will this take 20-70 passes?

A: No*

The size of a sentence is unbounded (even if we might assume a reasonable maximum length)

“Intractable” is the best word to use

Let’s say we set the sentence length to max of 20

Let’s say there are 100k commonly used English words

$$100k^{20} = 10^{100}$$

Fun language quirk: The dictionary says tractable means easy, so intractable means not easy. In a bit of coy understatement, “intractable” in Mathematics actually means “impossible”

Both of those assumptions are underestimates!

No* - as I just said, LLMs work this way, but they don’t’ actually base the conditional probability of the next token on ALL of the surrounding context.

Smaller Limit: N-Gram



Basic Idea: Probability of next word only depends on the previous $(N - 1)$ words



$$P(w_k | w_1, w_2, \dots, w_{k-1}) \approx P(w_k | w_{k-N+1}, w_{k-N+2}, \dots, w_{k-1})$$



$N = 1$: Unigram Model- $P(w_1, w_2, w_3, \dots) = P(w_1) P(w_2) \dots P(w_k)$

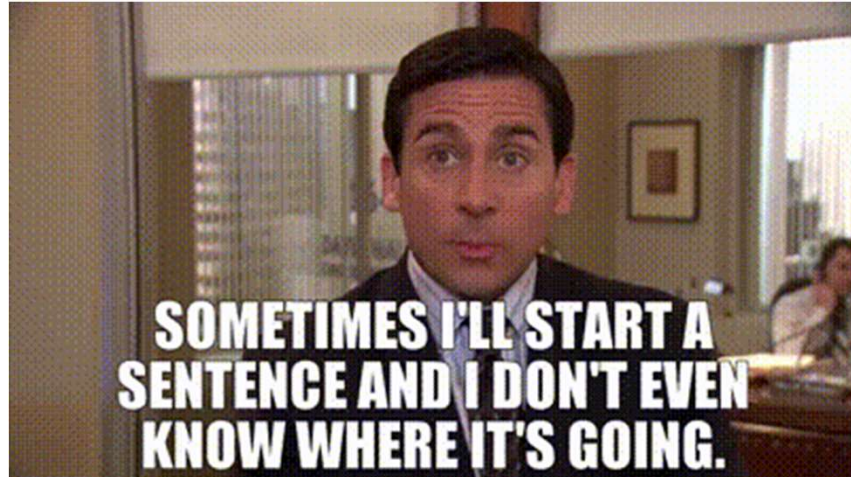


$N = 2$: Bigram Model- $P(w_1, w_2, w_3, \dots) = P(w_1) P(w_2 | w_1) \dots P(w_k | w_{k-1})$

Google uses
N-Grams for
Suggestions
(N is small)



The meme is intentionally deep fried. I'm told you Zoomers like that



People also use N-grams. Not really. Maybe. It's classified.

Do it with Hadoop!

- Unigram: $P(w) = C(w) / N$
- Bigram: $P(w_i, w_j) = C(w_i, w_j) / N$
 $P(w_j | w_i) = P(w_i, w_j) / P(w_i) = C(w_i, w_j) / C(w_i)$

You can probably figure out trigrams yourself.

Seem Familiar?

If it's not, I'm sorry about your mark on A1 ☹

State of the art rarely goes above 5-grams (call them that not penta-grams)

Example: Bigrams

“Training Corpus”

^ I am Sam \$

^ Sam I am \$

^ I do not like green eggs and ham \$

Counts

$(\wedge, I) = 2$

$(\wedge, \text{Sam}) = 1$

$(I, \text{am}) = 2$

...

Probabilities

$P(I | \wedge) = 2/3$

$P(\text{Sam} | \wedge) = 1/3$

$P(\text{am} | I) = 2/3$

$P(\$ | \text{Sam}) = 1/2$

....

Note: We never cross sentence boundaries

The probability $P(a | b) = C(b, a) / C(b, *)$

Recall that $C(b,*)$ means “count of all pairs that start with b”

Example: Bigrams

$$\begin{aligned} &P(\text{I like ham}) \\ &= P(\text{I} \mid \wedge) P(\text{like} \mid \text{I}) P(\text{ham} \mid \text{like}) P(\$ \mid \text{ham}) \\ &= 0 \end{aligned}$$

Thoughts?

Probabilities

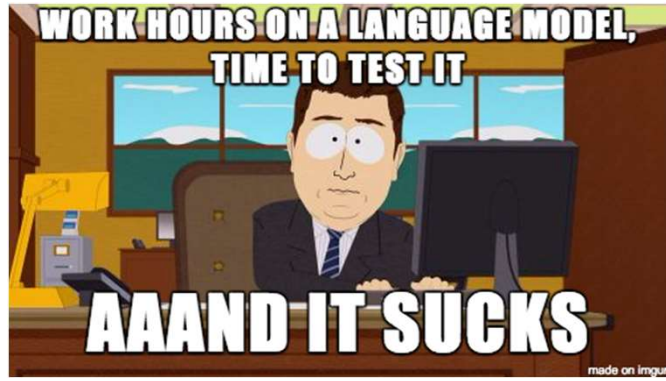
$$P(\text{I} \mid \wedge) = 2/3$$

$$P(\text{Sam} \mid \wedge) = 1/3$$

$$P(\text{am} \mid \text{I}) = 2/3$$

$$P(\$ \mid \text{Sam}) = 1/2$$

....



No More Zeros

$P(s) = 0$ means “sentence s is impossible”.

Not true (probably).

“The pirate was purple and wanted eleven candy ants.”

Your language model didn't think anybody would say that.

Take THAT, computer!

Explanation for the weird sentence. My son Charlie.

1. He likes pirates (and ghosts, and Halloween in general)
2. He likes purple
3. After the rogers “we want to earn Canadians trust back” he said “Candy ants trust”

His latest work is “Thank you for using self chicken”

Playing it Smooth

If a single n-gram in the sentence has never been seen, $P = 0$

Just one unusual word takes a sentence from “likely” to “impossible”

That’s a “discontinuity”

Removing 0s makes the distribution “smooth”

How can we remove 0s?

Robin Hood

“Take from the rich, give to the poor”

(Or, maybe, universal basic income? Every n-gram gets a non-zero probability)



I picked this picture from Office's suggestions because it's hilariously bad. I might be training it to give bad suggestions. Or funny suggestions. Oh, also, my Robin Hood is the furry Disney Robin Hood. Pretty sure he put a scarf on his head at one point in that movie. So I guess it's related after all. Spooky.

Laplace Smoothing

Start each count at 1, not 0.

Time tested and simple

Counts

$(\wedge, l) = 2$

$(\text{ham}, \$) = 1$

$(l, \text{like}) = 0$

$(\text{like}, \text{ham}) = 0$

...

Counts (Smooth)

$(\wedge, l) = 3$

$(\text{ham}, \$) = 2$

$(l, \text{like}) = 1$

$(\text{like}, \text{ham}) = 1$

...

Laplace Smoothing (bigram probabilities)

$$P(A, B) = \frac{C(A, B)}{N} \rightarrow P_L(A, B) = \frac{C(A, B) + 1}{N + V^2}$$

What's V? Vocabulary size. Since every pair of words (A,B) has a +1, we need to add V^2 to N.

You can imagine how to apply this to trigrams, 4-grams, etc.

Common question: "Shouldn't it be $V(V-1)$?" Nope. For A_1+A_2 cooccurrence we did not count the pair (X, X), but for bigrams we do, e.g. you might have the classic sentence "John had "had", while Carol had "had had". "Had had" had had a bigger impact than "had" had had.

Other Smoothing Techniques

- Good-Turing – Used by Good and Turing as part of cracking Enigma
- Katz backoff – “backoff” means “if n-gram says 0, try (n-1)-gram”
- Jelinek Mercer – Interpolate between n-grams and unigrams
 - $P_{JM}(A,B) = \lambda P(A,B) + (1 - \lambda)P(A)P(B)$
- Dirichlet Smoothing, Witten-Bell – Ways to pick λ
- Kneser-Ney – Current Best Practice
 - Google (used to?) use this for Google Translate, implemented on their MapReduce framework

Hidden Markov Model (Hmm)

Used a lot in
Bioinformatics

Also used a lot in NLP

Popular with Witchers



This slide is here for 3 reasons

1. The Hmm meme
2. I used them a fair bit in Bioinformatics
3. The fact that it's somewhat relevant (weak third place)

HMM and NLP

Phoneme Recognition

- Turn an audio stream into a word stream

Part-of-Speech (PoS) tagging

- Doesn't REPLACE N-grams, but helps add context to words
- Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo

Buffalo is an animal, a city, and a verb that means, essentially, to intimidate. That means the above nonsense is technically grammatical and means:
"Buffalo from Buffalo that are intimidated by other buffalo from Buffalo intimidate a third group of buffalo from Buffalo."

Hadoop HMM

- We don't use HMM on any assignments
 - Grads, feel free for your projects!
- The textbook discusses how to adapt HMM training for MapReduce
 - Surprise! Highly parallelizable
- It's iterative, which means Spark is a good alternative!



Transformers
(More than
Meets the
Eye)

Q: Hey, I saw that ChatGPT can do 8K, 16K, even 64K context...how is it not all zeros???

A1: It's a Transformer Neural Network, not a table of n-gram counters. Words will have latent probability – no smoothing needed

A2: "Self-Attention" – In a context of 4000 words, only some words are important.

The Yi model can do 200K context! It's pretty good for a 34 billion parameter model.



Topic Shift: Searching!
("Information Retrieval")

The Central Problem in Search



Concepts



Query Terms

"tragic love story"



Concepts



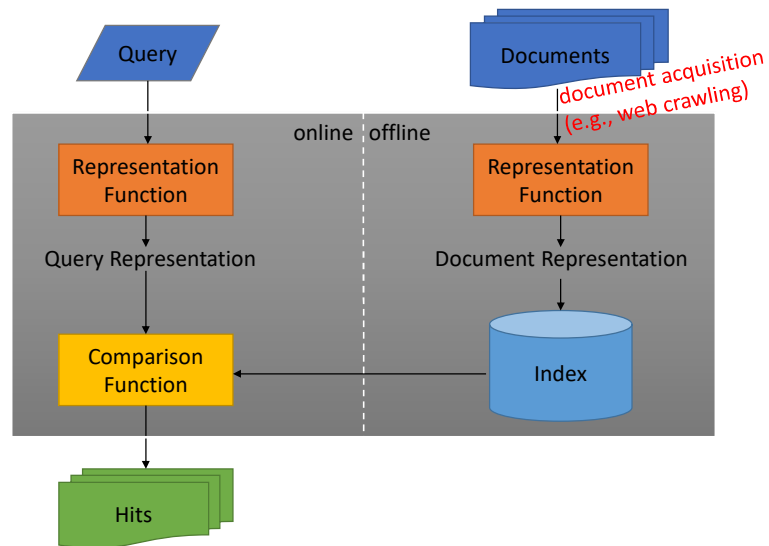
Document Terms

"fateful star-crossed romance"



These two things should match! They don't look similar though, do they? No words in common. Language is hard!

Abstract IR Architecture



Representation Matters

Computers can't "understand" (or can they?)

We need to tell them what "relevant" means.

Simple form: "Bag of Words"

Assumptions: terms are independent, relevance is irrelevant, the concept of a "word" is well defined

All of those assumptions are obviously wrong. However, so what? "First let's assume a spherical cow" etc.

What's a word?

天主教教宗若望保祿二世因感冒再度住進醫院。這是他今年第二度因同樣的病因住院。

الناطق باسم -وقال مارك ريجيف
إن شارون قبل -الخارجية الإسرائيلية
الدعوة وسيقوم للمرة الأولى بزيارة
تونس، التي كانت لفترة طويلة المقر

1982.الرسمي لمنظمة التحرير الفلسطينية بعد خروجها من لبنان عام

Выступая в Мещанском суде Москвы экс-глава ЮКОСа заявил
не совершал ничего противозаконного, в чем обвиняет его
генпрокуратура России.

व्युत्पत्त्या मेश्चान्स्कोम सुदो मस्कोवो एक्स-ग्लवा युकोसा जयविल
ने सोवर्शल निचो प्रोतिवोजकननो, व चोम ओबविनायत एगो
गेनप्रोकुरातुरा रूसी.

भारत सरकार ने आर्थिक सर्वेक्षण में वित्तीय वर्ष 2005-06 में सात
फ्रीसदी विकास दर हासिल करने का आकलन किया है और कर सुधार
पर ज़ोर दिया है

日米連合で台頭中国に対処...アーミテージ前副長官提言

조재영 기자= 서울시는 25일 이명박 시장이 "행정중심복합도시"
건설안에 대해 "군대라도 동원해 막고싶은 심정"이라고 말했다는
일부 언론의 보도를 부인했다.

These are all news blurbs. Top to bottom – Chinese, Arabic, Russian, Hindi, Japanese, Korean

Oh, there's also the inscription from The One Ring. The script is elvish, but the words are in the black tongue of Mordor, which I shall not utter here.

Stick to English

What words does the document contain?

- Tokenizer (remove punctuation)
- Case Folding (treat things as lower case, put Unicode into canonical form)
 - Bush vs bush



Unicode issue: é is a single character. Or it's e followed by the “acute” combining diacritics. Unicode defines a canonical form, the “standard” way to represent a string that may have many equivalent forms.

The thing I did with coöp vs co-op vs coop is also relevant! How'd'ya like that setup? The long game.
Capitalization is not important, except when it is.

We will continue to depend on Jimmy's tokenizer and not worry about confusing Jack Black with a darkly coloured device for lifting cars.

“What’s in the bag?”

AKA
Foreshadowing ML

A word is an integer? What about a vector of floats?

A representation is often called an *embedding*

You take a high dimensional object e.g. a text document, and embed it in a lower-dimensional plane

Distance between embeddings: cosine

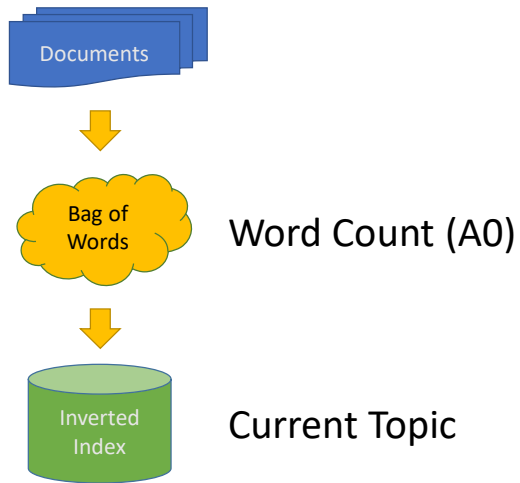
Distance between words: “semantic similarity”

Goal for embeddings: $D(e_1, e_2) \sim D(w_1, w_2)$

E.g. $D(\text{“love”, “romance”})$ is low, so their embeddings *should* have a similarly low distance.

Guess what my friends? PMI is a great way to estimate the “semantic similarity” of words. PMI gives similarity measures similar to cosine similarity! $\text{PMI} = 0 \Rightarrow$ terms are uncorrelated aka orthogonal. $\text{Cos} = 0 \Rightarrow$ terms are uncorrelated aka orthogonal

Bag of Words



We're ignoring syntax, semantics, knowledge of language, meanings of words, etc – however, BOW is often used with vector embeddings (e.g. word2vec)

Doc 1 Doc 2 Doc 3 Doc 4
 one fish, two fish red fish, blue fish cat in the hat green eggs and ham

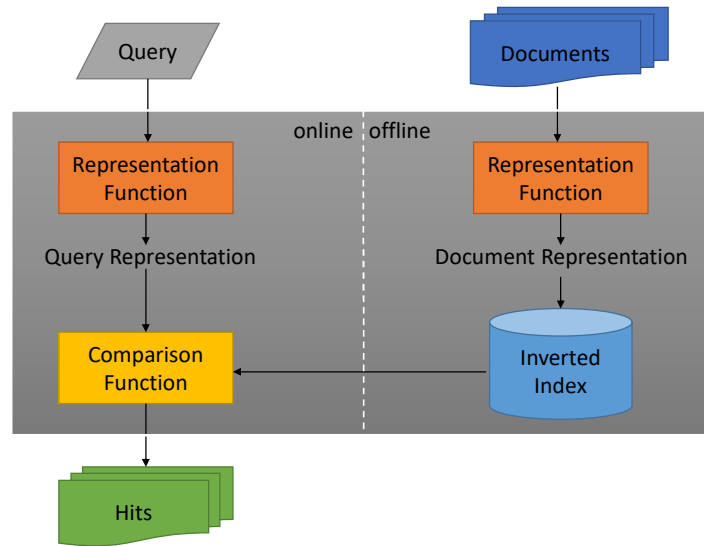
	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			

What goes in each cell?

boolean
 count
 positions

Cs451 – Terminology – Inverted Index. Maps context to documents. Forward Index. Maps documents to context. (Seems strange to me, so a book's index is "inverted"?)

Abstract IR Architecture





Scaling Assumptions

- Queries are small
- Postings are not
 - There are a LOT of documents (100M? 1B? 10B?)
 - $1\text{B docs} * 1\text{ bit} = 120\text{MB} / \text{unique word}$
- How many unique words?

Vocabulary Size: Heaps' Law

$$M = kT^b$$

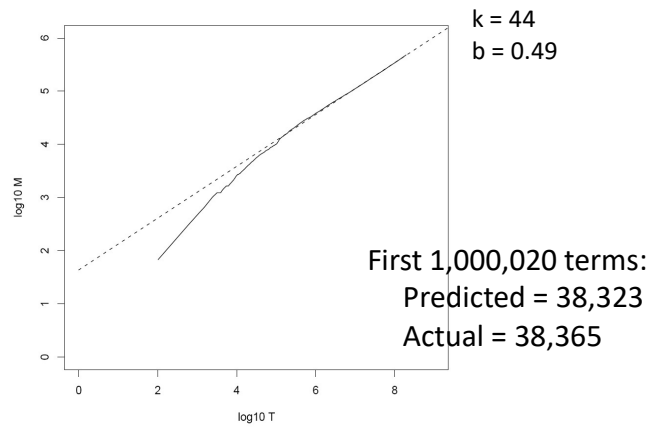
M is vocabulary size
 T is collection size (number of documents)
 k and b are constants

Typically, k is between 30 and 100, b is between 0.4 and 0.6

Heaps' Law: linear in log-log space

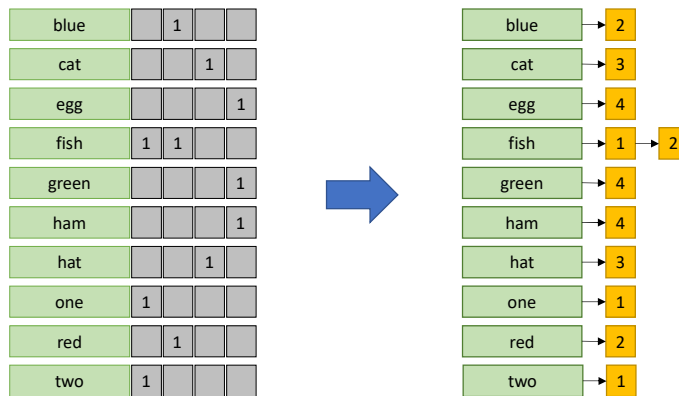
Surprise: Vocabulary size grows unbounded!

Heaps' Law for RCV1



Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

Saving Space, Postings List



This saves a lot of space because most terms do not appear in most documents (so most rows are mostly 0s). Most? Not all?

Postings Size: Zipf's Law

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^N (1/n^s)}$$

N number of elements
 k rank
 s characteristic exponent

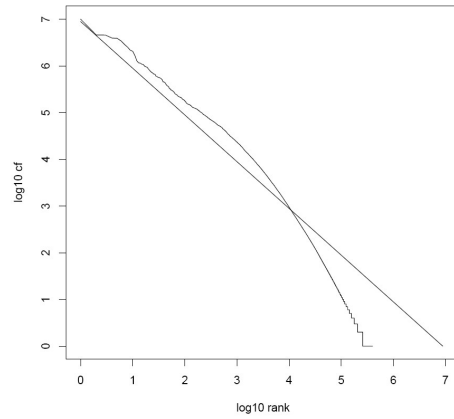
Zipf's Law: (also) linear in log-log space

In other words:

A few elements occur very frequently
Many elements occur very infrequently

https://www.youtube.com/watch?v=fCn8zs912OE&t=253s&ab_channel=Vsauce

Zipf's Law for RCV1

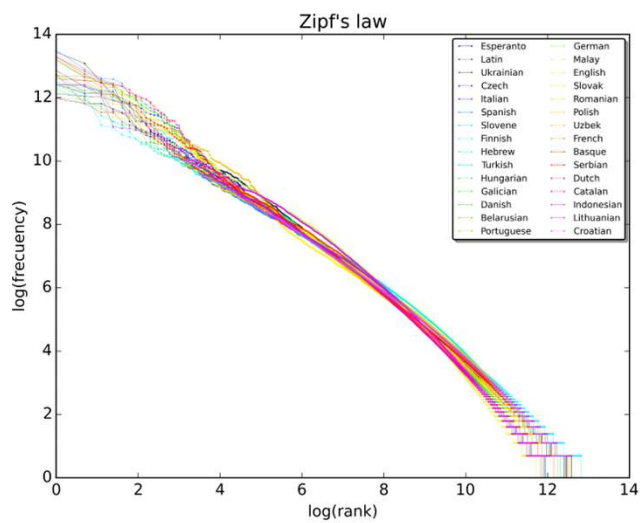


Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

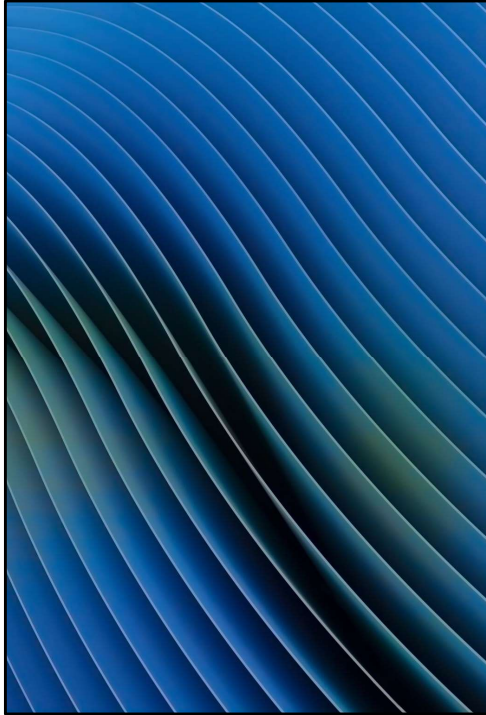
Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)

Close enough

Zipf's Law for Wikipedia



Rank versus frequency for the first 10m words in 30 Wikipedias (dumps from October 2015)



MapReduce to the Rescue

Map –

- input (docid: doctext)
- output (term: (docid, freq))
- (can add metadata, e.g. pos)

Reduce –

- input (term : Iterator[(docid, freq)])
- output (term : Postings List)

See any scaling issues?

Pseudo-Code, Mapper

```
def map(docid: Long, doctext: String):  
    counts = counter()  
    for term in tokenize(doctext):  
        counts.add(term)  
    for term, freq in counts:  
        emit(term, (docid, freq))
```

We can assume each document has only a few million unique terms, so the counter will easily fit in a mapper's memory

Pseudo-code, Reducer

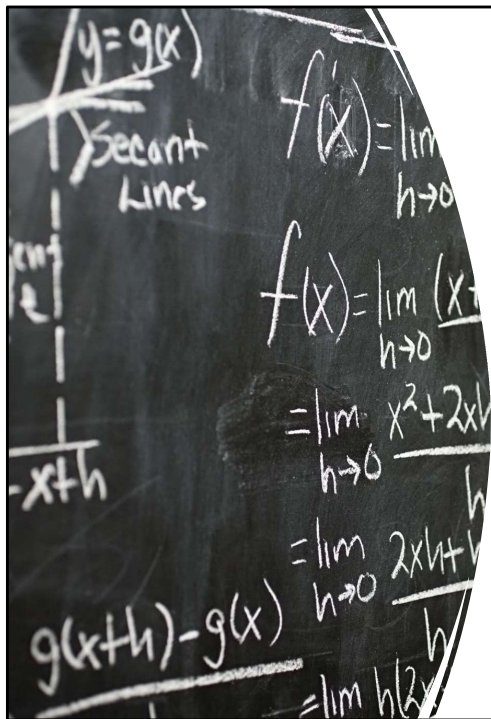
```
def reduce(term: String, postings: Iterator[(Long, Int)]):  
    p = list()  
    for docid, freq in postings:  
        p.append((docid, freq)) Problem? How big is this list going to be???  
    p.sort() Problem?  
    emit(term, p)
```

How big does p get? Zipf's law says "usually small, sometimes not small". Sorting is $O(n \log n)$. That's not ideal if n is large. Isn't Hadoop good at sorting? (It is.) Besides which, Hadoop already is sorting (by keys) so really we're sorting twice (even if the second sorts are on a much smaller n, it's still not nothing).

If you did the readings you remember...

- “Secondary Sorting Pattern”
 - (Another “fancy term for simple concept”)
- $(A : (B, C)) \Rightarrow ((A, B) : C)$
 - Remember to make the partitioner send (A, x) to the same partition for all x
- Now it’s already sorted by document ID
 - Cool, but how does that save us space?

Why does this make it faster? Well, the mappers are already sorting by key, so now instead of two sort passes through the data, we’re only doing one! Additionally, if you have more mappers than reducers (common) you have more machines doing the sort in parallel.



Delta Compression

Zipf's Law works for US now

If a term is rare: There are not many postings

If a term is common: The average delta ($\text{docid}_{i+1} - \text{docid}_i$) is small

Delta Encoding (AKA Gap Encoding)

If a sequence is ascending, you can instead write down only the “delta” (difference) between elements:

Sequence: 1, 6, 11, 15, 22, 42, 49, 77

Gaps : 1, 5, 5, 4, 7, 20, 7, 28

Does that save anything though?

Not if your output
is Int.

Thing is, there are
more datatypes
out there!



VInt

Variable-Width Integer type (There's also VLong)

Uses 1-5 bytes to represent an Int (same range as 4 byte fixed width)

How?

Need a way to indicate the length, that's all!

Technically VInt and VLong are exactly the same, writeVInt just passes the int along to writeVLong

VInt Details

If x in $[-112,127]$ – write using 1 byte – that leaves 16 options for other cases

Else: “Magic” byte **1000**SLLL followed by L Bytes for x :

- x is non-negative
- $-x - 1$ if x is negative

1000 – (common prefix, indicates this is a special byte)

S – Sign: 0 = negative, 1 = positive.

L – Length, 3 bit two’s complement: 1 => 111, 2 => 110, 3 => 101, etc.

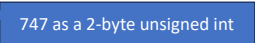
Note that this leaves room for lengths of up to 8 bytes for the magnitude – writeVInt is just a wrapper for writeVLong

You don’t need to know the details for the exam, it’s just fun knowledge to have

VInt Examples

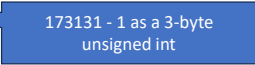
747 : requires 2 bytes and is positive:

(S = 1, LLL = 110)

1000 1110 0000 0010 1110 1011  747 as a 2-byte unsigned int

-173131: requires 3 bytes and is negative:

(S = 0, LLL = 101)

1000 0101 0000 0010 1010 0100 0100 1010  173131 - 1 as a 3-byte unsigned int

You definitely will not be asked to do this yourself.

VInt Compression

Explain!

- No
- Array is storing objects (HEAVY)
- Bytes is just a bunch of bytes



ArrayWritable[VInt]

BytesWritable

Normalize using the LaForge version of the Drake meme. Levar Burton and LaForge are heros and don't you forget it.

More detailed explanation: Vint saves space when you have many packed together. ArrayWritable doesn't pack them together. You need raw bytes, and to use `WritableUtils.writeVInt`



Detour! Other Bit-Bashing Methods

- CS451: You don't need to use these, VInt is fine
- CS431: You don't do an indexing assignment at all
 - (Sorry, it's kinda fun)



A few reasons for this difference in courses

1. Spark doesn't sort by key when reducing, so the secondary sort pattern can't be applied.
2. Spark makes Vint etc a bit trickier to use
3. Bespin has a starting point in Hadoop MapReduce, but not in Spark. So it would be a lot more work for 431 students.

VLQ (Variable Length Quantity)

- This confused me because it's used everywhere and often called VInt or VarInt
- How it works: Slice number into septets. Use high bit to indicate "continues"

Examples:

```
767 => 0010 1111 1111 [binary]
        0000101 1111111 [7-tuples]
        10000 101 0111 1111 [VarInt, 2 bytes]
74  => 0100 1010 [VarInt, 1 byte]
```

In a previous version of the slides I presented this as Vint – As it says above, this is usually called Vint! I just assumed that's what Hadoop used (this version only works for unsigned ints, but there are variations that let the leading byte also indicate sign.

Simple-9

How many ways can you divide up 28 bits?

28 1-bit numbers

14 2-bit numbers

9 3-bit numbers [1 bit wasted]

7 4-bit numbers

5 5-bit numbers [3 bits wasted]

4 7-bit numbers

3 9-bit numbers [1 bit wasted]

2 14-bit numbers

1 28-bit number

- Why 28? 4 bit “selector”, 16 options. Only use 9 though.

- Extend to 64-bit

- 14 ways to divide 60

- Simple

- Works fairly well for gaps

We have to go deeper

Simple-9 (and Simple-14) work at the WORD level

Different ways to store a variable number of values in a single word

VInt (or VLong) works at the BYTE level

Store a fixed range of values in a variable number of bytes

What about BIT-level?

Store a fixed range of values in a variable number of bits

Elias γ Code

Assumptions

- natural numbers with no upper bound
 - like counts, for example
- small numbers are more common than large numbers
 - gaps for common terms, for example
 - term frequency within docs, too?

γ is gamma, fyi

Elias γ Code

Encoding x:

1. Let $N = \lfloor \log_2 x \rfloor$
2. Write N 0s
3. Write x as an $N+1$ -bit number
This number starts with 1. Trust me

Decoding x:

1. Read 0s until a 1, call this N
2. Interpret next $N+1$ bits as a binary number
Including the 1
3. There is no step 3

γ is gamma, fyi

Does γ work well?

Does well for term frequencies (how many times the term appears in the document)

Does OK for gaps, too



Underlying Assumption

The Elias code assumes the values are distributed by a power law

Most numbers should be small, or it doesn't save any space



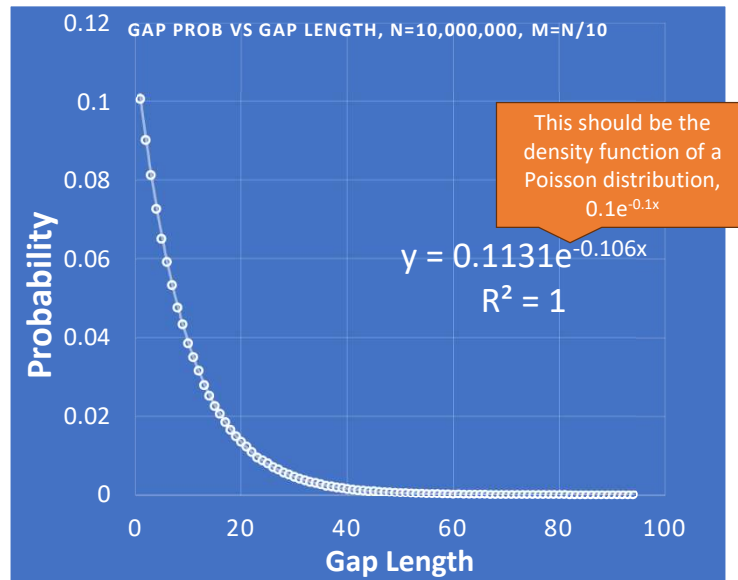
Gap Distribution

What do you think the distribution of gaps looks like if you have N document IDs and a term that matches M documents?

It...has a lot of binomials in it. It's scary.

Unless you remember Stats, in which case it's clearly a Poisson distribution with $\gamma = M/N$

Simulated
gaps for 10
million
documents



As we all know, the probability density for “waiting time” in a Poisson distribution with parameter γ is $\gamma e^{-\gamma t}$

Here $\gamma = 0.1$ and we can clearly see that the simulated distribution matches quite closely.

TF DF Distribution

The math here is actually the same except that the overall DF can exceed N , and sampling is done with replacement.

TL;DR: Still a Power Law



It's a Polish surname
(Gołęb)
transliterated to English

Golomb Code

For encoding positive integer x :

- Quotient and remainder when divided by M

$$q = (x - 1) / M$$

$$r = x - qM - 1$$

q gets encoded in unary (q 0s, then 1)

r gets encoded in truncated binary

Number of documents containing term

$$\text{Let } z = \frac{D_t}{D}$$

Number of documents (total)

Approximate as $\left\lceil \frac{0.69}{z} \right\rceil \rightarrow M = \left\lceil \frac{\log(2-z)}{-\log(1-z)} \right\rceil$ -- good for gap encoding

Pronunciation key: ł is pronounced mostly as an l, but in some parts of eastern Poland (and polish speaking Ukraine) it's more like a w sound.
ą is pronounced somewhere around w "om" or "am" sound.

However, none of that really matters as Golomb himself said it as "Go-lam" so it doesn't really make any sense to dig deep into exactly what part of Poland we're talking about

That's right, each term gets its own custom encoding scheme! Neat.

Note that this is for situations where we only care about "contains" or "doesn't contain", not the number of times a term appears in a document

Golomb Encoding

Unary

n is represented as n 0s, then a 1

5 => 000001

0 => 1

(Or you can switch 0s and 1s)

Truncated Binary

For numbers $\{0, 1, \dots, n-1\}$:

$k = \lfloor \log_2 n \rfloor$

$u = 2^{k+1} - n$

First u codewords:

First u codes with length k

Last n-u codewords:

LAST n - u codes with length k+1

Truncated Binary

$N = 15$ i.e. set = $\{0, 1, \dots, 14\}$

$k = \text{floor}(\log_2 15) = 3$

$u = 2^4 - 15 = 1$

0 => 000

1 => 0010

2 => 0011

3 => 0100

...

14 => 1111

Golomb Code Examples

$$M = 12 \text{ (k=3, u = 4)}$$

$$x = 52$$

$$q = (52 - 1) / 12 = 4 = 00001_u$$

$$r = 52 - 12 * 4 - 1 = 3 = 011_t$$

$$\text{encoded}(x) \Rightarrow 00001011_b$$

$$M = 32 \text{ (k=5, u = 32)}$$

$$x = 52$$

$$q = (52 - 1) / 32 = 1 = 01_u$$

$$r = 52 - 32 * 1 - 1 = 19 = 10011_t$$

$$\text{encoded}(x) \Rightarrow 0110011_b$$

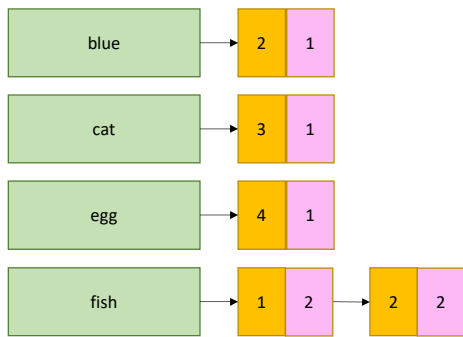
When M is a power of 2, these are also called “Rice codes” – Since $u = m$, there’s no “truncated” binary, it’s just k -bit binary.

Golomb Code in MapReduce

We can't calculate M without knowing df , but we only know that at the end!

(We solved that already with a special key that sorts first)

What if we also want frequencies?



((blue, *), 1)
 ((blue, 2), 1)
 ((cat, *), 1)
 ((cat, 3), 1)
 ((egg, *), 1)
 ((egg, 4), 1)
 ((fish, *), 4) or 2???
 ((fish, 1), 2)
 ((fish, 2), 2)

Orange = docid

Pink = freq (number of times term appears in that document)

(Term, *) = total freq? Or number of documents containing term? If we want to use golomb codes, we need both

ID and Frequency

It's best to treat these as independent

You can encode each with a different approach!

(Don't mix something ALIGNED like VInt with something UNALIGNED like Golomb)

Comparison

Indexing 181MB of Wikipedia sentences, including term frequency

It's pretty normal for the index to be larger than the documents being indexed!

Method	Size
Uncompressed (Int)	182MB
VInt	78MB
Gamma (both)	44MB
Golomb (gap) + Gamma (freq)	41.4MB
Golomb (both, different M)	41.2MB

Why is it bigger? Well each term is using 8 bytes per document it appears in, while common words tend to be short! (And the average is about 5 letters). – removing stop words will make it smaller, but means that a user cannot search for stop words even if they really want to! That might be OK but just be aware.

Going from Gamma to Golomb for the frequencies made it slightly smaller, but barely any. Also need to collect both document frequency for terms, and total frequency (how many times the word appears across all documents) – more messages for very little gain




One Last Thing

There's also the "Exponential Golomb Code"

Same thing, except you use an Elias Gamma code to write the quotient instead of using unary.

It's SLIGHTLY smaller than regular Golomb



MapReduce it?

The indexing problem *Perfect for MapReduce!*

Scalability is critical

Must be relatively fast, but need not be real time

Fundamentally a batch operation

Incremental updates may or may not be important

For the web, crawling is a challenge in itself

The retrieval problem

Must have sub-second response time

For the web, only need relatively few results

Uh... not so good...

That's not to say we don't want to distribute it! In fact we definitely do! There's no way for a single machine to handle thousands of concurrent users who all expect sub-second response times!

Retrieval

Let's assume everything fits in memory on one machine

Boolean Retrieval

Query is formed using Boolean operators (AND OR NOT)

(either matches or doesn't)

Hits: set of documents for which the query is true

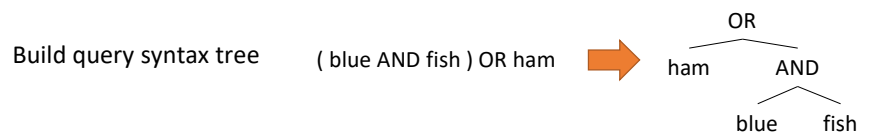


Remember: A set might be sorted by docid, but certainly isn't sorted by relevance to the query. It either matches the query, or doesn't.

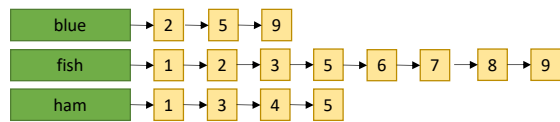
The next few slides present two different algorithms for computing hits based on the posting lists for each term in the query.

Boolean Retrieval

To execute a Boolean query:



For each clause, look up postings



Traverse postings and apply Boolean operator

Term-At-A-Time

For each term, generate sets of documents

- AND = intersection
- OR = union
- NOT = negate

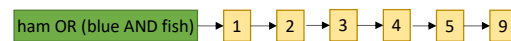
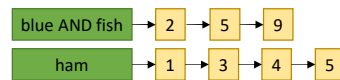
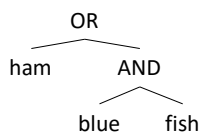
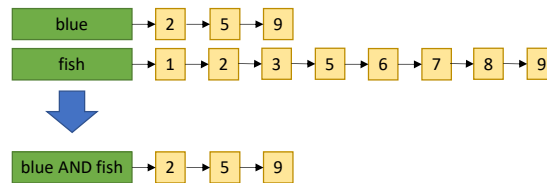
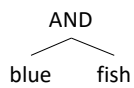
Analysis?

Not seems bad. Really bad. Not (rare term) => just about the whole internet. We SURE about assuming this fits into memory???

It's actually not bad, though. You can have a "negated" flag on a set, and then just store its negation.

Also we only need 2 sets in memory at once. Easy to hold that many in memory (hopefully)

Term-At-A-Time



Since the postings are sorted by docID, AND / OR are modifications of the merge functions from mergesort.

AND – only include a doc if the “next” doc from both postings is equal. OR – if both “next” are equal, only include it once, otherwise same as normal merge

What about not? It’s best to have a flag for the set: “Inverted”. If an inverted set contains 5, it means the true set does NOT contain 5. (Now you have to modify merge though...)



Document- At-A-Time

For each document, see if it passes the query

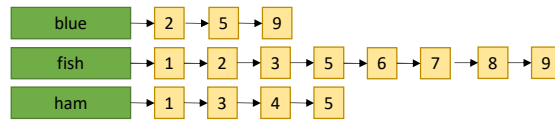
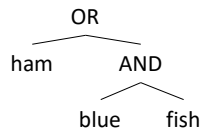
Since documents are in sorted order, modified merge operation will work

Analysis?



Need to have the posting lists for each term in memory simultaneously! (but you can stream them from the FileSystem I'm sure...)
Not makes things a bit awkward again.

Document-At-A-Time



Repeat: for the smallest “next doc” – does it match?

- 1 – has ham, include and advance “fish” and “ham” lists
- 2 – no ham, but does have blue and fish. Include.
- 3, 4, 5 – ham. Include.
- 6, 7, 8 – no ham, fish but not blue. Exclude
- 9 – fish and blue. Include

One More Thing...

Our index file is partitioned! We probably want to distribute lookup, even if it's not a MapReduce task we use.

Options:

- Leave the index partitioned by term
 - Each index file has the COMPLETE collection for a SUBSET of terms
- Repartition by document ID
 - Each index file has a SUBSET of the collection, but the index for ALL terms

Which one makes the most sense?

Note that the textbook discussion of this is referring to Ranked Retrieval where there's no "or" or "not". So with that restriction index partitioning isn't TOO bad – but we'll talk about that in a bit here.

For Boolean Retrieval index partitioning is very tricky to handle – basically for the "term at a time" algorithm shown, many (most?) operations will span partitions and will require a shuffle (or shuffle-like operation, since we're not using MapRed/Spark)

Document indexing is much nicer! Each partition has all terms so it can produce "these are all hits from my partition" – the "reduce" action is then simple concatenation. No shuffle (just collecting the results back to the user's machine)

Ummm, unsorted?

A set of hits is fine...but we probably want them sorted by relevance?

That's a different problem: Ranked Retrieval!

Requires: relevance function $R(q, d)$

Note: A query "X Y Z W" might yield a high Relevance value even if a document only contains terms X Y W.

Ranked Retrieval

Simplify the query. It's now only a list of terms (AND, but not strict – can be missing some terms)

Need a way to weight a hit



Ranked Retrieval

- Can we just do Boolean Retrieval and then sort by relevance?
 - No
- ☹️ Why?
 - See last slide – searching for $q = "x\ y\ z\ w"$ is NOT the same as $"x\ \text{AND}\ y\ \text{AND}\ z\ \text{AND}\ w"$



One way to be relevant

- Terms that occur many times in one document should have high weights (for that document)
- Terms that occur in many times in the entire collection should have low weights

We need:

- term frequency (times a term is used in a document)
- document frequency (number of documents containing the given term)

TF-IDF

(Term Frequency and Inverse
Document Frequency)

$$w_{ij} = tf_{ij} \log \frac{N}{df_i}$$

- w_{ij} – weight (relevance) of term i in document j
- tf_{ij} – number of occurrences of term i in document j
- df_i – number of documents containing term i
- N – total number of documents

The relevance of a document is the sum of the w_i values

Document-At-A-Time

For each document:

1. Score each query term, and add them all up
2. Accumulate best k hits
 1. A min-heap is good for this

PRO: time is $O(n \log k)$, memory is $O(k)$ – k probably a constant

CON: can't terminate early, must look at whole document collection

Another pro – easily distributed

Term-At-A-Time

1. Collect hits and ranking for rarest term into accumulator
2. For each other term in the query:
 1. If a document does NOT have that term, remove from accumulator
 2. Otherwise, add next term's ranking to overall ranking

PRO: Can have early termination heuristics, will not normally need to traverse all documents

CON: uses a lot of memory

Which to use?

Good question. There are tradeoffs. No one correct answer.

Depends a lot on the query, too.



usually what's done is the documents are partitioned by document ID with a "quality" assessment. E.g. partition 0 is the BEST pages, partition 1 is lower quality but still good, etc...

Then you can do document-at-a-time on the BEST partition and if that gets you enough hits with high relevancy then you can stop.

So I guess they BOTH have early termination heuristics. Oops.



What about synonyms?

1. Use word2vec on your document set and create a table of synonyms:
 - if a user searches “love” grab “love” and “romance” postings
2. Use doc2vec to create document embeddings, then...use a Vector Database

[Not on exam, just a brief mention of “what’s next”]



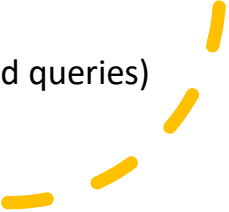
Vector what now?

Vector Database – Vespa, Pinecone,
ChromaDB, PostgreSQL with plugins

Uses HNSW (like a multidimensional skiplist!)
index

Given a query vector, finds the nearest
neighbours

“Just” need to turn documents (and queries)
into vectors!



[Not on exam, just handy to know]