Indexing:

- Inverted Index: For each term t, we must store a list of all documents that contain t.
  - Identify each doc by docID

Term		docs that contain term
Brutus	→	1, 3, 6, 10, 14
Jason	→	1, 3, 7, 9, 15, 16

Called inverting because you map ~~term~~ terms to documents

will be dictionary data type

postings, should be sorted by docID. Also should be of variable length Array. so like linked list.

- Inverted index steps
  1) Collect documents to be indexed
  2) Tokenize documents.
     - go through each document and assign each term with the doc it's in.

doc 1	doc 2
Jason goes to store	Kelsey goes here

tokens

Jason - 1	Kelsey - 2
goes - 1	goes - 2
to - 1	here - 2
store - 1	

  3) modify ~~tokens~~ tokens to make things like "friends" to "friend"
  4) sort Alphabetically by term.
  5) index the term listings.
     - multiple term entries in a single document are merged
     - Split into dictionary and postings as shown at top of page.
     - Doc freq. information is added.

INDEXING :

- Positional indexes :

  - In the postings of inverted index, store positions in which the tokens appear.

    <term, # of docs containing term,
    doc#, position1, position 2, etc ,
    doc# , position1, position 2, etc,
    >

- Zipf's law : frequency of any word is inversly proportional to its Rank in the frequency table.

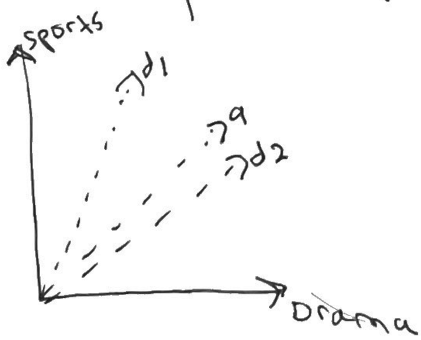
  ~~A~~ Basically words like I, you, an appear a lot in documents, but words like "biology" appear less so they are Ranked higher.

RANKING:

- Vector space model:

Boolean retrival	vs Ranked Retrival
- good for expert users with persice understanding of collection.	- most users incapable of writing boolean queries.
- easily comb through lot's of documents	- most users dont want to wade through loads of results.
	- boolean retrival produces too few or too many results

- Represent document and query by concept vectors.
  - Each vector is one-dimension.
- measure similarity between query vector and document vector.



- x, y are concept vectors, which are independent of the documents.

- vectors have a weight of each concept and query has weight to define importance of concept.
- Two ways to define weights
  - Tf : Count of a term t in doc d.
    weigh a term more if it occurs more in a document.
  - IDF: Assign a higher weight to rare terms.
  - Tf-idf vector: Cross product of TF and IDF
- Compute Cosine similarity on tf-Idf vectors.

RANKING:

BM25 - Goal is to be sensitive to term frequency and document length while not adding too many parameters

- Normalize ~~document length~~ tf using ~~Norma~~ document length

- 2 parameters
  - term frequency scaling for document
  - ~~document length normalization~~
  - term freq. scaling for query

- BM25 is better than vsm because of the document length normalization.

CLASSIFICATION:

3 ways to classify

- manually → consistent when problem size and team is small. cannot be scaled to a large dataset
- Hand-coded rule-based classifiers
- supervised learning
  ↳ classify document d to a class c.
  Classes are predefined.
  a training set of D documents is used with label in C.
  Naive bayes, k-nearest neighbors
  All need a hand-classified training data set.

- Evaluation

  Classification Accuracy - r/n   n = # test docs
  r = # of test docs correctly classified.

  recall,
  F1,

CLASSIFICATION :

Rocchio - Create centroids which is an average vector of all documents that belong to a class.

~~Compare~~ ~~?~~ Attempt to classify a document to the most related centroid.

worse than Naive Bayes

KNN - define K the number of documents to consider when classifying a document.

No training necisary.

Scales well to large # of classes.

more accurate than Rocchio, Naive Bayes

Variance vs Bias -

bias - simplifying assumptions made by the model

Variance - amount that the estimate of the target function will change given different training data.

KNN high varriance, low bias

Rocchio/NB low varriance, high bias

Naive Bayes - Assign Probabilities to each class that the document in question belongs to that class.

good dependable baseline for text classification

$$\text{Probability for a class} = \frac{\left(\text{\# times term appears in class's docs}\right)}{\text{total \# words in class's docs}}$$

sometimes apply laplace smoothing "add 1"