Lecture 1: Intro

- **Information Retrieval (IR)**
  - Finding materials (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
  - It involves the use of algorithms to retrieve information from databases to answer questions.
  - Major example and concern are **search engines**.
- **Scales of IR systems**
  - **Personal IR system** (e.g., retrieving, filtering, or monitoring files from PC).
  - **Enterprise search** (e.g., internal searching tools in organizations for domain-specific IR).
  - **Web search:** system for retrieving information from a large amount of heterogeneous data.
- **History**
  - **Late 1950s to 1960s:** foundation of the field.
    - Luhn's work on automatic indexing (KWIC)
    - Cleverdon's Cranfield evaluation methodology and index experiments
    - Salton's early work on SMART system and experiments
  - **1970s-1980s:** a large number of retrieval models
    - Vector space model, probabilistic models
  - **1990s:** further development of retrieval models and new tasks
    - Language models, TREC evaluation, Web search
  - **2000s-present:** more applications, especially Web search and
    - Interactions with other fields, learning to rank, scalability (e.g., MapReduce), real-time search.
    - **Today's IR systems** are fast, flexible (operate on different languages, tolerate typos), provide ranked retrieval based on **relevance** (the user perception of the result as containing information of value with respect to their personal information need).
- **IR major areas of concern**
  - **Query representation**:
    - How to process queries and overcome lexical and semantic gaps?
  - **Document representation**:
    - What data structures to use for storing information that allow efficient access?
  - **Retrieval model**
    - Which algorithms to use to find the most relevant documents for a given information need?
  - **Speed and space**
    - Quality assessment of an IR systems in terms of efficiency and required storage?
- **Covered topics:**
  - Search, recommendations, question answering, text mining, online ads, audio/images/video parsing.
  - **Search engine indexing:** the collection, parsing, and storing of data to facilitate fast IR.
  - **Scam site**
    - Tries to trick search engine algorithms to get ranked and displayed as relevant when they are not, can be filtered using several factors (e.g., user reports or ML models)
  - **Side-by-side quality rating:**
    - Humans evaluating whether search engine results have improved or not by comparing two results of the same query, side-by-side for different versions of the search engine.
    - They give feedback about the relevance and reliability of information.
    - The feedback is used later to improve the quality of the search engine.

Lecture 2-3: Web Basics

- **Recall from Networks.**
  - Internet, TCP/IP protocol suite.
  - FTP, POP, IMAP, HTTP(S), Cookies, DNS.
  - Web Security (Identification, Authentication, Authorization), digital certificates.
- **Hypertext languages, Schema languages and related Web terms.**
  - SGML, HTML (5), XML, XSD, DTD, RSS
  - WWW, URI/URL/URN.
- **Browsers and Browser Engines, SAX, and DOM.**
  - Browser fingerprint (fingerprint JS)
- **Crawling:**
  - robots.txt, sitemap.xml, APIs, OAuth.

Lecture 4: Quality Assessment

- **Quality assessment:**
  - Usage of mathematical metrics to evaluate the quality of search engine.
  - Requires having a mathematical definition of relevance.
  - Depend on the targets of the service/company.
- **Offline Metrics**
  - Accuracy, Precision, Recall, F1.
  - Mean Average Precision (MAP)
  - Mean Reciprocal Rank (MRR)
  - (Normalized) Discounted Cumulative Gain ((N)DCG)
  - pFound
- **Online Metrics:**
  - A/B Testing.
- **Other quality aspects**
  - Fresh, top-ranked results vs. old, high-quality ones.
  - Diversity and novelty of recommendations.
  - Legal and ethical safety.

Lecture 5: Indexing

- **Recall from Theoretical Computer Science (Lectures 10-11)**:
  - **Formal Grammar:**
    - Generates/validates strings of a language through a rewriting (reduction) process.
    - Grammar is a 4-tuple: $< \boldsymbol{\Sigma}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{S} >$
      - Σ: language terminals (non-reducible), N: non-terminals (reducible)
      - P: production rules, S: staring symbol.
    - Formalizing language grammar allows recognition and generation by computers.
  - **Chomsky Hierarchy:** describes a hierarchy of languages.
    - **Recursively enumerable (general, unrestricted):** $\alpha \rightarrow \beta, \alpha \neq \varepsilon$
      - All production rules are allowed.
    - **Context-sensitive:** $\alpha A \beta \rightarrow \alpha \gamma \beta, \gamma \neq \varepsilon$
      - Rewriting a non-terminal will vary depending on its context.
    - **Context-free:** $A \rightarrow \gamma$
      - Rewriting a non-terminal doesn't depend on its surroundings/context
    - **Regular:** $A \rightarrow \alpha B$ XOR $A \rightarrow B\alpha$
      - Allows rewriting a nonterminal as a terminal followed/preceded by at most one nonterminal, but not both.

    A, B: non-terminals.

    α, β, γ: any sequence of terminals and/or non-terminals.

    ε: the empty string.
  - **Example of a syntax tree for an English sentence with the corresponding production rules written in the Extended Backus-Naur Form (EBNF)**

    Determiner = the | a

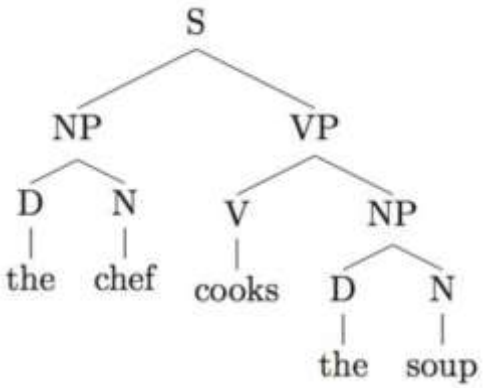
    Noun = chef | soup

    Verb = cooks

    Noun**P**hrase = **D**eterminer **N**oun | …

    Verb**P**hrase = **V**erb **N**oun | **V**erb **N**oun**P**hrase

    Sentence = **N**oun**P**hare **V**erb**P**hrase

  - **In this example:**
    - NounPhrase can be object (item) or subject, can for example allow adjectives (e.g., fast rabbit).
    - VerbPhrase is like an action or intent that – when identified by a system – can trigger some functionality (e.g., Alexa, **play** music).

- **Language as a mathematical object**
  - **Pragmatics:** why and when would you use a certain language?
  - **Syntax (grammar):**
    - Enforces rules on letter and words order by describing how to form valid strings belonging to the language.
    - These rules can change the meaning of the sentence (semantics).
    - Reduces the complexity of language and diversity of valid sentences by enforcing such rules.

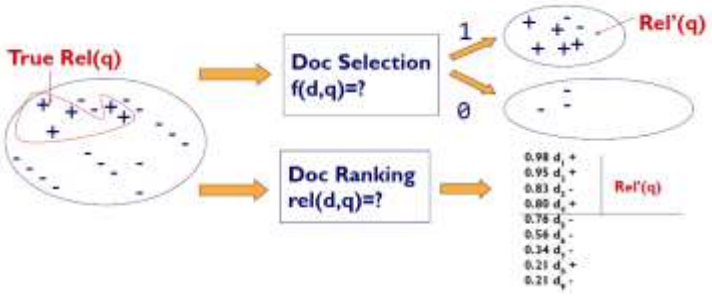
- **Tokenization (scanning):**
  - Extracting tokens (lexemes) from language text that are used as non-terminals in grammar, tokens types are typically introduced to group similar tokens together and to identify the role of a certain token in a certain position within the text.
- **Compression techniques:**
  - Methods used to reduce the size of the language (the number of different tokens) and hence reduce complexity and make processing easier.
  - Using them will indeed result in loss of meaning and sometimes intent, but we still sacrifice that in exchange of reducing complexity.

Method	Description	Example
Case folding	Transforming all words to lowercase.	London → london.
Stemming	Removing prefixes and suffixes to get the important part of the word that carries meaning only	compression/uncompressed → compress
Lemmatization	Converting the inflected forms of a word into a single token (lemma, dictionary-form) representing the important part.	better → good
Ignoring stop words	Removing frequent words that doesn't carry a meaning itself	to, the, it, be → ε
Thesaurus	Using a dictionary of synonyms to replace.	rapid → fast

- **Semantics:**
  - Meaning of phrases and context.
  - Lies in words and how they are organized.
  - Structural vs. Cognitive way of understanding languages
    - **Structural:** languages have structure, that's why we can understand them.
    - **Cognitive:** human brain is flexible and can understand a language of any form by experience.

- **Boolean Information Retrieval (BIR)**
  - The earliest IR model that used simple matching from user query (set of words) against the set of all documents. It uses Boolean operators to include/exclude documents.
  - **Example query:** ("obama" AND "healthcare" AND NOT "news")
  - **Implementing recommendation system using BIR:**
    - **Pre-select:**
      - Gets the approximate set of candidate documents for a given query (can be done in a logarithmic time if documents are stored in an index data structure).
      - Set operations are applied (e.g., AND will result in intersection operation being done on the resulting documents).
    - **Filtering:**
      - From the resulting set, apply an exact-matching method to filter out the irrelevant documents.
      - In the pre-select stage, we lost some information by just collecting all documents containing the query words, in this stage we restore the lost information by filtering again based on word order, form, etc.
    - **Ranking:**
      - Assign relevance scores and sort based on them.

- **Problems with BIR**
  - **Over-constrained query:** little to no results.
  - **Under-constrained query:** almost all documents are query results.
  - **Not very human-friendly.**
  - **BIR aims for document selection.**
    Document ranking is often preferred.
    - BIR aims to get all relevant documents, but no one will check all the results, users quit as their need is satisfied.
    - Ranking documents allows the user to control search boundary.



- **Direct/Forward Index vs. Inverted index (in the context of search engines)**
  - **Index:** given a document, returns the list of words appearing in it.
    - **Example:** Google crawls the web, figuring out which words appear in each page.
  - **Inverted index:** given a word, retrieves the list of documents in which it appears.
    - **Example:** user query (after being pre-processed) is used as a key for the inverted index to retrieve the documents satisfying information need.
  - **Building an inverted index:**
    - **Build a lexicon**: set of all reduced words (that are not extremely common or extremely rare among documents).
      - Some forms have 2+ words (n-grams) that shouldn't be separated to not lose the lexical meaning (e.g., New York should be scanned as a single token).
      - Words that occur in almost all texts are likely to be stop words and shouldn't be indexed (they will increase the size of index but won't help with discrimination).
      - Words that are extremely rare might just be typos or non-existing terms.
    - **Build a posting list** for each word in the lexicon
      - **Posting list:** set of pointers to documents containing a certain word.
    - **(Optional) store the index** as a sparse matrix in persistent storage
      - Storing indexes in hard-drives is inefficient for modern applications as accessing and using them will be slow.
      - Storing large indexes can be done with prefix trees (tries) or horizontal fragmentation (sharding) of data based on hash or lexicographical order.
      - Building an index requires a bigger amount of memory than actually storing it.
      - Indexes are usually static (once built they rarely get updated).
  - **Ranking over inverted index**
    - **Term Frequency (TF):** the number of times a certain term (token) appears in a document.
      - **Boolean Frequency:** $\mathrm{TF}(t, d) = 1$ if the term appears in the document and 0 otherwise.
      - **Adjusted TF:** $\mathrm{TF}(t, d)$ divided by the number of words in the document.
      - **Logarithmically scaled TF:** $\log(1 + \mathrm{TF}(t, d))$
      - **Augmented Frequency:** $0.5 + 0.5 * \frac{\mathrm{TF}(t, d)}{\max\{\mathrm{TF}(t', d) : t' \in d\}}$
    - **Inverse Document Frequency (IDF):** how discriminative is the term among all documents.
      - $\mathrm{IDF}(t, D) = \ln\left(\frac{|D|}{|d \in D : t \in d|}\right)$
    - **TF-IDF:** multiplication of the above metrics, reflects the importance of a term in a document.

Lecture 6: Wildcards, Spellchecking, and Query Correction

- **Wildcards**
  - Placeholder for zero or more characters that are typically used in search engines when the user is uncertain of the spelling of some query term, or is aware of multiple spellings and want to include all.
  - **Most common wildcards**
    - Asterisk (*) matches zero or more characters.
    - Question mark (?) matches zero or one character.
  - **SQL "LIKE" syntax**
    - (%) matches 0 or more characters.
    - (_) matches one character.
  - **Regular Expressions**
    - The most expressive and general method for matching languages.
    - PostgreSQL uses (~) for matching against regexes (it also supports original LIKE syntax).
  - **Matching results can be done efficiently depending on the data structure storing the text.**
    - Trailing wildcards can be handled using search trees (e.g., trie)
    - Leading wildcards can be handled using reverse search trees.
    - **Wildcards appearing in the middle** are harder to match against
      - SQL "LIKE" statement only uses the string before the first wildcard to lower search space, this proceeds with linear search.
      - Postgres uses B-tree index for if the pattern is constant and the first letter is not a wildcard.
- **Implementing wildcard queries**
  - **Permuterm index (store all rotations)**
    - Builds the lexicon from an extended version of the data to support a wider range of queries.
    - The $ marks the end of the string and is used to support the usage of prefix and infix wildcards by rotating the query such that the wildcard goes to the end, as explained on the right:
    - **Example:**
      - "hello" is stored as (hello$, ello$h, llohe, lohel, o$hell, $hello)
      - "world" is stored as (world$, orld$w, rldwo, ldwor, d$worl, $world)
      - **Lookup(*o) = Lookup(o$*) =** o$hell = hello
      - **Lookup(*l*) = Lookup(l*) =** lohel, ldwor = hello, world
    - **Problems:**
      - Increasing vocabulary size → more memory required.
      - Not handling multiple wildcards.

X → X$
X* → $X*
X → X$
X → X*
X*Y → Y$X*
X*Y*Z → X*Z to narrow the search space, then do a full scan.

  - **N-gram index (store all substrings of length *n*)**
    - Similar to permuterm index, builds the lexicon from an extended version of the data to support wildcard queries. Uses $ to mark the beginning and the end of a word then convert wildcards into Boolean queries.
    - **Example**
      - 3-grams of "hello" are (he, hel, ell, llo, lo)
      - Lookup(h*o) = Lookup($h) & Lookup(o$)

- **Multigram index:** an n-gram index that filters out unnecessary grams.
  - **Selectivity** of an n-gram = $\frac{\text{number of docs containing that gram}}{\text{total number of docs}}$
  - **Filtering Factor (FF)** of an n-gram gives an impression about the importance of a certain n-gram and is equal to 1 – the selectivity of that n-gram.
    - **Example:** the 3-gram **"the"** has a very high selectivity (almost all documents contain it), thus a very low filtering factor, and is thus, not useful for filtering.
  - By building the inverted index only on the grams with a filtering factor bigger than some threshold, we make the system faster.
  - **Problems:**
    - Matching becomes less accurate as the ignored grams might not have 0 FF.
    - The index is hard to maintain online (selectivity should be recalculated as new documents are added).
- **RegEx support:** several ideas were proposed to support regular expressions in IR systems:
  - Fast Regular Expression Engine (FREE)
    - **General idea:** convert the regex into a Boolean query and use an existing n-gram index**.**
    - **Example:** /[ab]cde/ → (acd OR bcd) AND cde
  - Google Code Search (GCS)
    - Get 5 properties of the query: emptyable, exact, prefix, suffix, match.
    - Recursively union them (with possible simplification)
    - Use inverted index of trigrams for query evaluation
  - Automaton Transformation
    - Represent the regex as FSA.
    - Simplify FSA graph (some edges won't be supported).
    - Traverse the graph and convert to binary query.

- **Spell checking and correction**
  - Usage of **proximity-based** or **probability-based** methods to check/correct **documents being indexed** or **user queries** to adapt machines to human mistakes.
  - **Proximity-based:** for a given misspelled query, choose the **closest** one for correction
    - **Example:** word a is corrected as b if $\text{Metric}(a, b)$ is optimized.
    - **Metric can be**
      - **Edit distance (aka. Levenshtein distance)**
        - The minimum number of operations (insert/delete/replace one character) required to transform string a to b.
        - Some sources include the swapping of two characters as one operation.
      - **Weighted edit distance**
        - Edit distance, but we add weights given the nature of input (e.g., for keyboard input: q is close to w, but for voice input q is close to k)
        - Formulated as a probability model that takes weight matrix as input.
      - **N-gram overlap**
        - Enumerate all n-grams of the user query.
        - Use the existing n-grams index on the documents to retrieve all lexicon terms matching any of the query n-grams.
        - Define a threshold of matched n-grams, if a term matches a number of query n-grams greater than this threshold, we select it as a correction candidate.

Longest Common Subsequence (LCS) can also be used as a metric (although edit distance is more commonly used).

- **Example:**
  - User searches for "helli"
  - **Query 3-grams:** (he, hel, ell, lli, li)
  - **"hello" term 3-grams:** (he, hel, ell, llo, lo).
  - **Three** 3-grams matched, assuming "hello" was the only term with highest number of matches, it is selected as a correction.

N-gram overlap can be used with to find candidate terms to calculate edit-distance against, or it can itself be used as a metric.	**Weights** can also be introduced in the example to account for the fact that "i" is closer to "o" than "y" (in QWERTY keyboard), preferring "hello" over "helly" (assuming it's a valid term).	**Jaccard Coefficient (aka. Intersection over Union (IoU))** is a commonly used measure for overlapping between two sets. $IoU(X,Y) = \frac{\lvert X \cap Y \rvert}{\lvert X \cup Y \rvert}$ gives a number between 0 and 1, we can use this number as a threshold for n-gram overlap.

- **Probability-based:** for a given misspelled query, choose the most probable one for correction.
  - **Example: "**grnt" is corrected as "grunt" or "grant" depending on which one is being searched the most.
- **Document correction:** the system can preprocess documents as it indexes them.
  - Especially needed for OCRed documents (e.g., correcting rn → m).
  - Can use domain-specific knowledge (e.g., correcting O → I as they are adjacent in QWERTY keyboards).
- **Query correction:** the system can fix the user query automatically and retrieve documents, or suggest one or more corrected queries to the user if not confident.
  - **Isolated words correction:** checking a word for misspelling (e.g., Lazania → Lasagna), taking the lexicon as a reference for correct words (what's not found in it is assumed to be a mistake).
    - **The lexicon can be**
      - Composed from a standard dictionary.
      - An industry-specific lexicon (generated by tools and reviewed by humans).
      - A text corpus: structured (e.g., indexed) collection of words, names, acronyms, or common misspellings.
    - Finding the correction term from lexicon is done using the methods described above.
    - Won't catch typos resulting in correctly spelled words (e.g., form → from).
  - **Context-sensitive**
    - Deduces correction from context (e.g., I flew form A to B).
    - **Assume typos exist:** assume they exist all the time or use a heuristic (e.g., a matrix of words and how often they appear together) to decide whether to check for typos.
      - In this example, "flew" doesn't appear with "form" often.
    - **Method 1: retrieve dictionary terms that are close to each query term**.
      - Choose words that are close to the suspected typo until you find the most probable fix (should also be more probable than the original query).
    - **Method 2: Bi-word statistical approach**
      - Break phrase query into a conjunction of bi-words.
      - Look for bi-words that need only one term corrected.
      - Enumerate only phrases containing common bi-words.

- **Issues with spellchecking**
  - **Which alternative queries to show to the user?** Use a machine learning model.
    - Show many alternatives, learn a probabilistic model from user choices.
    - **Query log analysis:** user rewrote a query quickly because results were not what they wanted, learn from that.
- **Soundex:** phonetic algorithm for indexing names by sounds, as pronounced in English.
  - **Algorithm:**
    - Retain the first letter of the name.
    - Change all occurrences of the following letters to 0: 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
    - Change (similar) letters to digits as follows:
      - B, F, P, V → 1
      - C, G, J, K, Q, S, X, Z → 2
      - D, T → 3
      - L → 4
      - M, N → 5
      - R → 6
    - Remove all pairs of consecutive digits.
    - Remove all zeros from the resulting string.
    - Pad the resulting string with trailing zeros.
    - Return the first four positions, which will be of the form
      - <uppercase letter> <digit> <digit> <digit>.
    - **Example:** Hermann becomes H655

  - **Generalization to create a lexicon from voice-input:**
    - Turn every query token into a 4-character reduced form.
    - Build and search an index on the reduced forms.

  - **Problems:**
    - Not efficient with typo fixing.
    - High recall, very low precision.

  - **Phonetic string matching:**
    - Improvement over Soundex, represents input as phonemes.
    - Applies the same text algorithm (i.e., edit distance) on input.

Lecture 7: Vector Space Modeling

- **Term-Document Matrix (TDM)**
  - Describes the frequency (or weight) of terms occurring in a collection of documents as a matrix, with **terms as rows** and **documents as columns**.
    - **Document-Term Matrix** ($\text{DTM} = (\text{TDM})^\top$)
  - Column is an n-dimensional vector describing the Bag of Words (BoW) in a document.
  - Row is a vector describing the term.
  - The matrix is sparse (mostly 0's) for short documents.
- **Vector Space Model**
  - Assume an m-dimensional vector space with orthogonal basis composed by TDM rows (let's call them concept vectors).
    - This assumption implies that terms do not correlate.
    - The idea is to express documents as vectors lying in that latent space and then represent our query as another vector and see which document vectors are the **"closest"** to it.
  - **Let:**
    - $\text{doc}_\text{i} = (w_{1i}, w_{2i}, \dots, w_{ki})^\top$
    - $\text{term}_\text{i} = (w_{i1}, w_{i2}, \dots, w_{in})^\top$
    - $\text{query} = (q_1, q_2 \dots, q_m)^\top$

	doc_1	…	doc_n	query
term_1	w_{11}	$\cdots$	w_{1n}	q_1
⋮	⋮	⋱	⋮	⋮
term_m	w_{m1}	$\cdots$	w_{mn}	q_m

  - **Where:**
    - $\text{TDM}_{m\times n} = \{w_{ij}\}$
    - w_{ij} = the weight (significance) of term_i in doc_j (e.g., TF-IDF).
    - q_i = the weight of term_i in query (i.e., how important is w_i in the query).
  - **Then**
    - Relevance of doc_i to query is proportional to $1 - \text{distance}(\text{query}, \text{doc}_\text{i})$ where distance measures irrelevance between two vectors.
    - distance can be the Euclidean distance or $1 -$ Cosine Similarity

$$S_C(A,B) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{m} A_i B_i}{\sqrt{\sum_{i=1}^{m} A_i^2}\sqrt{\sum_{i=1}^{m} B_i^2}}$$

    - $R_{n\times 1} = \text{DTM} * \text{query}$ yields the ranking matrix in which a larger number indicates a document that is more relevant to the query.
  - **Example:**

$$\textbf{TDM} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, q = \text{t}_1\ \text{t}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Longrightarrow R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

    - **Interpretation:** the 4th document is the most relevant to the query, followed by the 1st and the 2nd. The 3rd one is not relevant.
  - **Problems:**
    - Similarity calculation is computationally intensive for large number of documents.
    - Assumption of orthogonal concepts might not be true in practice.

- **Distributional Semantics**
  - Research area that involves studying **semantic similarities between linguistic items** based on their distribution properties in large samples of language data.
  - **Distributional hypothesis:**
    - **Equivalent definitions:**
      - Linguistic items with **similar distributions** have similar meanings.
      - Words that are used and occur in **similar contexts** tend to purport similar meanings.
      - You shall know a word by the company it keeps.
    - This idea is the basis of ML-based NLP.
  - **Word Embedding**
    - Representing a word as a **vector of real values** lying in a **latent semantic space** such that the words that are closer in the vector space are expected to be similar in meaning.
- **Dimensionality Reduction** methods are used to reduce the number of dimensions of the TDM, thus making calculations faster.
  - **Approach 1:** use modulus, on collisions, either store the weight of the more recent document, the sum of the existing weight and new one, or the maximum.
    - $\forall i \in \{1, \dots, n\}: doc[i \,\%\, N] = \begin{cases} doc[i] \\ doc[i \,\%\, N] + doc[i] \\ \max(doc[i \,\%\, N], doc[i]) \end{cases}$
    - Then consider only the first N documents.
    - Works well with sparse TDMs
  - **Approach 2:**
    - Random projection of data onto a lower dimension.
    - Or just remove a random dimension.
  - **Approach 3:** applying different mathematical techniques that differ in what they try to preserve from the original data.
    - **Latent Semantic Analysis (LSA):**
      - Used to find a low-rank approximation of TDM (titled X in the image) by calculating its Singular Value Decomposition (SVD).
      - The largest k singular values (σ) with their corresponding u and v vectors **form** the k-dimension approximation of X with the smallest error (the cosine similarity properties between vectors is somewhat preserved).

$$\begin{matrix} & X & & U & & \Sigma & & V^T \\ & (\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\ & \downarrow & & & & & & \downarrow \end{matrix}$$
$$(\mathbf{t}_i^T) \rightarrow \begin{bmatrix} x_{1,1} & \dots & x_{1,j} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \dots & x_{i,j} & \dots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,j} & \dots & x_{m,n} \end{bmatrix} = (\hat{\mathbf{t}}_i^T) \rightarrow \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} & \dots & \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \cdot \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix}$$

    - **Principal Component Analysis (PCA):**
      - Tries to preserve variance, but cosine similarity is changed.
      - Can be used when the irrelevance metric is the Euclidean distance.

PageRank

- **Impact Factor** (of an academic journal)
    - is a scientometric (concerning scholarly literature) index that reflects the yearly mean number of citations of articles published in the last two years in a given journal.

$$\mathrm{IF}_y = \frac{\mathrm{Citations}_y}{\mathrm{Publications}_{y-1} + \mathrm{Publications}_{y-2}}$$

- **PageRank**
    - An algorithm used by Google Search to rank web pages in their search engine results.
    - PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.
- **CheiRank**
- **HITS Algorithm**
- **TrustRank**

Lecture 8: Vector Space Modeling with ML

- **word2vec**
- **doc2vec**
- **DSSM**
- **BERT**

Lecture 9-10: Approximate Nearest Neighbor Search

- **Approximate Nearest Neighbor Search (ANNS)**
  - Used in IR systems to get approximate ranking of search results for a given query, as doing exact ranking will require more time and is not strictly required.
  - Works by preselecting a number of potential neighbors (**pre-ranking set)** then filtering out the irrelevant results.
    - **Example:** to get the top k results, select $c * k$ approximate neighbors, and filter out the irrelevant $c * k - k$ ones.
- **ANNS using graphs:**
  - **Locality-sensitive hashing**
  - **Clustering**
    - Grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups.
    - Similarity is not strictly defined; different algorithms achieve different results based on how they implement **linkage** (when to say that two elements belong to the same cluster).
      - **Single linkage (smallest distance):** an element joins the closest cluster
      - **Average distance and centroid-based approaches:** an element joins the cluster where the average distance to all cluster members (or the distance to the cluster centroid) is the minimum.
      - **Complete linkage (maximum distance):** builds clusters such that the maximum distance within a cluster is minimized.
      - **Minimum energy:** uses energy (e.g., the variance between elements) instead of distance, since complete linkage is affected by outliers.
    - **Divisive vs agglomerative hierarchical clustering**
      - **Agglomerative:** a bottom-up approach; each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
      - **Divisive:** a top-down approach; all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.
    - **Case studies:**
      - **K-means (divisive method)** tries to split the data by defining leaders (cluster **centroid**) and groups (elements belonging to the same cluster).
      - **DBSCAN (agglomerative method)** tries to split the space into strongly connected groups, it can be said that
    - **Example (using clustering for ANNS):**
      - Split the documents into $\sqrt{N}$ clusters (preprocessing step).
      - For a given query, compute the distance to all cluster centroids, the closest cluster defines the pre-ranking set.
      - From within that cluster, filter out the irrelevant results to get the kNN.
      - Time complexity is reduced from $O(N)$ to $O(\sqrt{N})$
  - **Facebook AI Similarity Search (FAISS)**
    - Builds an inverted index on cluster centroids instead of the original lexicon terms.
    - Clusters are constructed using k-means, they appear as Voronoi diagrams (space is divided into polygons with each one having a representative centroid).

        - Significantly saves memory in exchange of precision loss (two different terms are mapped to the same vector, resulting in **vector compression**).
    - **Proximity graphs**
        - **Small world experiment**
        - **Small world network**
            - **Watts-Strogatz model**
            - **Navigable Small World**
            - **Hierarchical Navigable Small World**

- **ANNS using trees**
    - **Search tree**
    - **Quadtree**
    - **K-d tree**
    - **Ball tree**
    - **Range tree**
    - **Vantage Point (VP) tree**
    - **Johnson–Lindenstrauss lemma**
    - **Annoy Index**

    - **Off-topic:**
        - **Interval tree**
        - **BSP tree**
        - **M-tree**
        - **R-tree**
        - **Octree**

Lecture 11: Query Improvement

- **Relevance feedback**
  - Using the user's explicit or implicit input as a navigator in the vector space to drift towards better search results.
  - The IR system iteratively recomputes better representations (tuning) of the query vector based on user feedback (e.g., the user marks a certain result as relevant and they want more results like this, or non-relevant and results like it should be excluded).
  - The IR system uses information from the original query q_0 as well as from the document that was marked as relevant q_r (or marked as irrelevant q_{nr}) to compute an optimized query that moves towards the relevant documents and away from irrelevant ones.
  - **Examples:**
    - "Similar pages" in google search (2009).
    - "Related articles" and "star" in google scholar.

  - **Assumptions and violations**

Assumption	Violations
User has sufficient knowledge for initial query: they know the exact lexicon term for what they are looking for.	- Misspelled words - Different vocabulary background (e.g., cosmonaut/astronaut refer to the same word but the first one is used by Russians). - Cross Language IR (CLIR): retrieving documents written in a language different from the query's.
Relevant documents are similar to each other (i.e., they cluster together) and irrelevant ones are not different from them.	- Different documents using different vocabulary for the same thing (e.g., Kyiv/Kiev). - Queries where the answer is inheritably disjunctive (e.g., Pop stars who worked at Burger King).

  - **Problems**
    - Relevance feedback expands the query and makes it harder to resolve.
    - Users are often lazy to provide explicit feedback.
    - It's not easy to understand how it works, so users won't use it right.

- **Theoretically optimal query:**

$$q_{opt} = \underset{q}{\text{argmax}}\ [sim(q, v_r) - sim(q, v_{nr})] = \frac{\sum_{d \in C_r} d}{|C_r|} - \frac{\sum_{d \in C_{nr}} d}{|C_{nr}|}$$

  - **Where:**
    - C_r: the set of documents (vectors) relevant to q_r (or irrelevant to q_{nr})
    - $C_{nr} = D - C_r$ where D is the set of all documents.
    - v_r: the centroid vector (aka. Center of gravity/mass) for C_r.
    - v_{nr}: the centroid vector for C_{nr}
    - $sim(a, b)$: the similarity metric between two vectors (e.g., cosine similarity).

- **Problems:**
  - We can't precisely calculate C_r
  - Original query is not considered in this formula.
  - The approach may drift results away from the desired cluster.

- **Rocchio algorithm** introduces regularization of the above approach to mitigate the mentioned problems.

$$q_m = \alpha q_0 + \beta \frac{\sum_{d \in D_r} d}{|D_r|} - \gamma \frac{\sum_{d \in D_{nr}} d}{|D_{nr}|}$$

  - **Where:**
    - q_m: the modified (improved) query.
    - D_r, D_{nr}: the known subsets from C_r, C_{nr}
    - α, β, γ: hyperparameters to be optimized (recommended values: 1, 0.75, 0.15).
  - **Comments:**
    - The higher the size of D_r, D_{nr}, the higher the required values for β, γ
    - The approach works well with a single q_{nr}, unlike the above one.
    - The algorithm improves recall, but no guarantee on high precision (irrelevant documents might still show up in results, but it's not a problem as long as relevant ones are there).

- **Evaluation of relevance feedback**
  - To check whether q_m is better that q_0 we need some metric.
  - We can calculate precision-recall graph on all documents before and after applying relevance feedback, but this will always bring better or the same results (it will never be worse since the user already assessed some documents and they will be included in q_m)
  - To make fair evaluation, we consider only documents from a residual collection (all documents without those assessed by the user), this may give results that are worse than with q_0 which is more useful for evaluation.
  - Applying relevance feedback is often very useful, while applying it twice won't improve much.

- **Pseudo relevance feedback**
  - Automates the manual part of true relevance feedback as users won't likely provide explicit feedback.
  - Works by retrieving initial results and assuming that top k are marked as relevant, then implements relevance feedback (e.g., Rocchio algorithm).
  - Works very well on average but can go horribly wrong in some cases, several iterations will cause query drift.

- **Implicit relevance feedback**
  - Uses implicit information about user (which documents they clicked previously, their search history) to do relevance feedback on behalf of them.
  - **Example:**
    - User searches for "reverse string" and the IR system appends "C++" based on user previous activity.
    - If there are no known activities, the system can use most commonly clicked documents and do relevance feedback based on them (i.e., pseudo feedback).

- **Query Expansion**
  - Reformulating a given query to improve retrieval performance, by appending/altering keywords implicitly, or asking the user before doing so.
  - The user gives feedback about the words/phrases in the query itself instead of the results as in relevance feedback.
  - **Example:** suggestions for different or similar queries that appear in Google images.
  - **Methods to augment user query:**
    - **Global analysis:** uses global, static information from all documents (e.g., common queries from query logs, statistics about words that typically occur together).
    - **Local analysis:** expands query based on analysis on the resulting set of documents.
    - **Thesaurus:**
      - Expanding query given the language dictionary (e.g., by adding word synonyms or related words)
      - **Example:** doctor → doctor +doc +medico +hospital
      - The thesaurus can be generated manually or automatically.
        - Manual thesaurus is hard to generate and update.
        - Automatic generation is based on word similarity analysis in documents.
          - **Definition 1 (more robust):** two words are similar if they co-occur with similar words.
          - **Definition 2 (more accurate):** two words are similar if they occur in a given grammatical relation with the same words.
      - Generally, increases recall, but may significantly reduce precision (e.g., in case of term ambiguity)
        - **Example:** "Apple computer" → "Apple +red +fruit computer"
        - May not retrieve additional documents since terms are already highly correlated (documents where you find one are almost the same ones where you find the other).
- **Search suggest**
  - Query feature used in computing to show the searcher shortcuts while the query is being typed into a text box.
  - Before the query is complete, a drop-down list with the suggested completions appears to provide options to select.
  - Trie is the most useful data structure to implement suggestions
  - **Issues:**
    - Blacklisting bad suggestions.
    - Complaints on certain suggestions (bots, law violations, insults).