Student Referral Sheet – Faculty of Computing (Final Examination Jan-June 2025)

Student ID: IT23262690 | Module Code: IT3041

L1 – Intro to IRWA

Is finding documents (mostly text) that satisfy an information need from large collections.
Data types: Structured, Unstructured, Semi-Structured (XML)
Information Need: The topic a user desires to know more about.
Relevance: Documents are relevant if they satisfy the user's information need, which is subjective and context dependent.
Information scarcity problem (or needle-in-haystack problem): hard to find rare information
Information abundance problem (for more clear-cut information needs): redundancy of obvious information
Precision: Fraction of retrieved documents that are relevant.
Recall: Fraction of relevant documents that were retrieved.
Boolean model:
Term-Document incidence matrix (1/0)
Y axis: terms, X axis: documents
Cons: extremely sparse, has a lot of 0's wastes memory, no support for complex query operators like 'near'
Inverted index: Has a dictionary of terms with a posting list for each term. Uses linked lists or variable length arrays.
Term: "Data", doc. freq → Documents: [1, 3, 5]
To **optimize a Boolean query**: start with the smallest set when using AND
For OR use the smallest sum of doc. freq first.
Set-based operations (AND, OR, NOT) can be efficiently executed in linear time if the query terms are processed in increasing order of postings size.

L2 - Term Vocabulary & Posting Lists

Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens.
Token - A token is an instance of a sequence of characters in some document that are grouped together as a use full semantic unit for processing.
Type - A type is the class of all TERM tokens containing the same character sequence.
Ex: increase in home sales in July – 6 tokens and 5 types.
Term - type that is included in the IR system's dictionary – 4 terms if "in" omitted.
Stop words: Extremely common words which would appear to be of little value in helping select documents matching a user need.
Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens.
Case folding: Reduce all letters to lower case
asymmetric expansion: Enter: window - Search: window, windows
Stemming (Porters Algorithm): Reduce terms to their "roots" before indexing, chops off the ends of words. Helps recall but harms precision.
sses -ss – Ex: Caresses → caress
Ies - I – Ex: Ponies → poni
ational - ate – Ex: International -> internate
tional -tion – Ex: Conditional-> condition
Lemmatization is the process of converting words to their base form (lemma) by removing inflectional endings using vocabulary and rules
Ex: am, are, is -> be
If the list lengths are m and n, the **merge takes O(m+n)**.
Skip Pointers: More skips -> shorter skip spans-> more likely to skip. But lots of comparisons to skip pointers. Ignores the distribution of query terms. Easy if the index is relatively static.
Phrase queries: multiple word queries. So can't just store docIDs in posting lists.
Biword indexes: Index every consecutive pair of terms in the text as a phrase
Pro: A long phrase can be represented as
the Boolean query Con: Without the position of the term in specific doc, can have false positives. And Index blowup to due to size.
Positional Indexes: each posting is a docID and a list of positions (offsets)
Proximity queries: ex: employment /k place; k=4 – docs having these 2 words withing 4-word spaces.
A positional index expands postings storage substantially. The complexity will be **O(T)** t = no of terms. 2 -4 times larger than non-positional index
Combination schemes: Biword indexes and positional indexes are combined. Include frequent biwords as vocabulary terms in the
index. Do all other phrases by positional intersection.

L3 - Dictionaries and Tolerant Retrieval

Dictionary Data Structures:
Hash Tables: Provide O(1) lookup time, but do not support prefix search or variant matching. Rehashing is costly when vocabulary grows.
Binary Trees: O(log M) search for balanced trees, where M is the size of the vocabulary. Slower than hashes. Re-balancing is expensive but Support prefix searches. B trees mitigate rebalancing problem.
Wild-card queries:
mon*: easy with a binary tree or B tree
*mon: can use a reverse B-tree
co*tion: ook up co* AND *tion in a B-tree and intersect the two term sets.
Cons: Expensive.
Solution: **Permuterm Index**. Hello -> hello$, ello$h, llo$he...
X lookup on X$ X* lookup on $X* always * at the end
Cons: 4x the size of lexicons
k-gram indexes: April -> a,ap,pr,ri,il,l
need a second inverted index from bigrams to dictionary terms that match each bigram. (bigram index). can give false positives.
k-gram vs. permuterm index
k-gram index is more space-efficient. permuterm index does not require postfiltering always (only need for some cases with multiple wild cards).
Processing wild-card queries: can result in expensive query execution.
Spell correction: Context-sensitive, Isolated word.
Document correction: don't change the documents and instead fix
the query-document mapping.
Lexicon: refers to the complete set of words or vocabulary used in a particular language.
Query misspellings: Edit distance (Levenshtein distance): min no of operations to convert one string to another.ex: replace,insert,delete
Weighted edit distance: Requires weight matrix as input.
Expensive to compute every possible word.
Option 1:n-gram overlap: Enumerate all the n-grams in the query string as well as in the
Lexicon. To measure the overlap:
Jaccard coefficient: X intersec Y / X uninon Y
Option 2: identify words matching 2 of its 3 bigrams and merge.
Soundex: Retain the first letter of the word, then change all occurrences of 'A', 'E', 'I', 'O', 'U', 'H', 'W', 'Y' to '0'. Convert 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; and R → 6. Remove all pairs of consecutive digits. Remove all zeros from the resulting string. Pad the resulting string with trailing zeros and return the first four positions.

Lecture 4: Scoring, term weighting and the vector space model

It takes a lot of skills to come up with a query that produces a
manageable number of hits AND gives too few: OR gives too many.
Pro: large result sets not an issue since only top k results are shown.
Ranking:
Jaccard: cons: does not consider term frequency, Rare Items in a collection are more informative than frequent terms. Jaccard doesn't consider this information. Order not considered.
Term frequency tf: number of times that t occurs in d.
Cons: Relevance does not increase proportionally with term frequency.
Binary term-document incidence matrix: Each document is represented by a binary vector ∈ {0, 1}
Term-document count matrices: **Pro**:considers the number of occurrences of a term in a doc.
Log-frequency weighting: 0 → 0,1 → 1,2 → 1.3,10 → 2,1000 → 4
Document frequency (df): no of documents containing t - >1+log10 tf
We want a high weight for rare terms so,
Idf weight: Inverse doc freq. -> idf = log10(N/df)
ignores the term's frequency within the document itself. has no effect on ranking one-term queries.
Term Frequency (tf): Measures the importance of the term within the document.
Inverse Document Frequency (idf): Measures the importance of the term across the whole collection (rarity).
tf-idf: $Wt,d = 1 + log\ 10(tft,d) * log\ 10(N/dft)$

Score $(q, d) = \sum tf * idf\ t,d$
Vector Space Model (VSM) represents both documents and queries as high-dimensional vectors. Does ranked retrival.
Euclidean distance:cons:too large distances
Solutions use angles: **Cosine similarity** (-1 to 1)
The cosine value ranges from 0 to 1 for non-negative tf-idf weights, and from -1 to 1 in general. It is easier to calculate cosine similarity with length-normalized vectors first.

$$\cos(q,d) = \frac{q \cdot d}{||q||||d||} \xrightarrow{\text{Length-Normalized}} q \cdot d$$

L5 - Scoring and Vector Space Model (VSM)

Cons: costly to calculate cosine for every doc
Use Max Heap (binary tree) for Selecting Top K,
Bottleneck: The primary computational bottleneck is cosine computation. construction takes 2J operations and reading 2log J. Cosine as a Proxy: Cosine similarity is a proxy for user happiness. Getting a list of K docs "close" to the true top K is often acceptable.
Index Elimination: Only consider documents containing high-idf query terms (low-idf terms like "in" or "the" contribute little to ranking) or documents containing many query terms (e.g., at least 3 out of 4 terms).
Champion Lists: Precompute, for each term t, a list of r documents with the highest weights in t's postings. At query time, only compute scores for documents in the champion lists of the query terms.
Static Quality Scores: Assign a query-independent quality score g(d) in between [0, 1] (e.g., PageRank, number of citations) to each document, modeling authority. The net score combines this with cosine relevance: net-score(q,d) = g(d) + cosine(q,d).
g(d)-ordering: Order all postings by g(d) (a common ordering). This allows high-scoring documents to appear early in the posting's traversal, enabling early termination in time-bound applications.
With ordered postings, the system can prioritize processing the most promising documents. Since its just one ranking for all the terms, Faster Postings Intersection, Parallelizing Computation of Cosine Scores.
Can **combine champions lists with g(d) ordering**:
Maintain for each term a champion list of the r docs with highest g(d) + tf-idf.
High and Low ListsFor each term, maintain two postings lists: high (champion list) and low. Traverse high lists first and only proceed to low lists if less than K documents are found.
Cluster Pruning Preprocessing: Pick √N leaders randomly. Assign every other document as a follower to its nearest leader.
Query Processing: Find the query's nearest leader L and only seek K nearest documents from among L's followers.
parametric Index: Postings for each field value (e.g., Year = 1601)
Zone Index: Inverted indexes built on specific zones to permit querying (e.g., "merchant in the title zone")
Tiered Indexes: Break postings into a hierarchy of lists (Tier 1: Most important, etc.) based on g(d) or another measure. At query time, use the top tier first.
Impact-ordered Postings: Sort each postings list by weighted term-frequency.
2 strategies for this,
Early Termination: Stop traversing a term's postings after a fixed number of documents or when Wf{t,d} drops below a threshold.
IDF-ordered Terms: Traverse query terms' postings in order of decreasing idf, as high-idf terms contribute most to the score.
Query parsers: first run the query as a phrase query
If <K docs contain the phrase rising interest rates, run the two phrase
Queries. If we still have <K docs, run the vector space query
rising interest rates, Rank matching docs by vector space scoring.
Evaluation of IR Systems: need 3 things,
A benchmark document collection, benchmark suite of queries, An assessment of either Relevant or Nonrelevant for each query and
each document.
Precision(P): Fraction of retrieved docs that are relevant:
P = tp/(tp+fp) = P(relevant/retrieved).
Recall (R): Fraction of relevant docs that are retrieved: R = tp/(tp+fn) = P(retrieved/relevant).
F Measure: A single measure that trades off precision vs. recall:

$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

Precision@K(P@K): The percentage of relevant documents in the top K retrieved results. Ex: prec@3 of relevant/retrieved top 3

Mean Average Precision (MAP): average of Precision@K for each rank position where a relevant document is found, averaged across multiple queries/rankings. (only take precision when a relevant doc is found))

Avg precision = (prec@1+prec@2+..Prec@n)/n

Mean Avg Precision do the previous for multiple queries and divide by the no. of queries.

L-11: Web Crawling

Purpose: Web crawlers, also known as spiders or automatic indexers, systematically browse the web to collect up-to-date data for search engines.

Crawlers start from "seed" URLs, fetch pages, extract links, and repeat the process with the new URLs. Challenges: Malicious pages, spam, non-malicious challenges, latency, bandwidth issues, and politeness, Spider traps: spider trap (or crawler trap) is a set of web pages that may intentionally or unintentionally be used to cause a web crawler or search bot to make an infinite number of requests or cause a poorly constructed crawler to crash.

Must be,

Robust: Be immune to spider traps and other malicious behavior from web servers. **Polite**: Respect implicit and explicit politeness considerations. **Explicit politeness**: specifications from webmasters on what portions of site can be crawled (robots.txt)

Implicit politeness: even with no specification, avoid hitting any site too Often.

Robots.txt: A protocol used by websites to define which parts of a site can be crawled and which cannot.

Ex: No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

User-agent: searchengine
Disallow:
User-agent: *
Disallow: /yoursite/temp/

Should be:

Be capable of **distributed operation**, **be scalable**: designed to increase the crawl rate by adding more machines, **Performance/efficiency**: permit full use of available processing and network resources, Fetch pages of "higher quality" first, **Continuous operation**: Continue fetching fresh copies of a previously fetched page**, Extensible**: Adapt to new data formats, protocols.

URL frontier: It's a data structure (often a priority queue) holding URLs that have been discovered but not yet fetched by the crawler. Must try to keep all crawling threads busy.

Process:1. **Fetching**: Retrieve pages based on URLs. 2.**Parsing**: Extract useful data and links. 3.**Duplicate Detection**: Check if the page is already indexed or in the crawl queue. 4.**URL Filtering**: Apply filters to avoid crawling unnecessary or duplicate pages.

Basic crawl architecture:

The URL frontier, containing URLs yet to be fetched in the current crawl (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching), **A DNS resolution module** that determines the web server from which to fetch the page specified by a URL. **A fetch module** that uses the http protocol to retrieve the web page at a URL, **A parsing module** that extracts the text and set of links from a fetched web page, **A duplicate elimination module** that determines whether an extracted link is already in the URL frontier or has recently been fetched.

Common OS implementations of DNS lookup are blocking only one outstanding request at a time

Solutions: DNS caching, Batch DNS resolver – collects requests and sends them out together

Parsing: URL normalization: convert relative links to absolute links.

Duplicates: This is verified using document fingerprints or shingles.

Filters– regular expressions for URLs to be crawled/not. Once a robots.txt file is fetched from a site, need not fetch it repeatedly, Doing so burns bandwidth, hits web server, Cache robots.txt files

Duplicate URL elimination: For a non-continuous (one-shot) crawl, test to see if an extracted+filtered URL has already been passed to the frontier.

Run multiple crawl threads, under different processes – potentially at different nodes, Partition hosts being crawled into nodes, Hash used for partition.

Host Splitter: It directs URLs to the appropriate node responsible for crawling that host. This ensures that all crawling threads within the system (across all nodes) don't repeatedly hit the same server.

URL frontier: two main considerations:

Politeness: do not hit a web server too frequently.

Freshness: crawl some pages more often than others.

These goals may conflict with each other. (E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.

Politeness – challenges: Even if we restrict only one thread to fetch from a host, can hit it repeatedly (add a time gap between requests)

Mercator scheme (FIFO): is a specific design for the URL frontier within a web crawler architecture, designed to address the conflicting goals of politeness and freshness.

1. **Front queues**: Manage prioritization, 2. **Back queues**: Enforce politeness.

1. **Front Queues (Prioritization)**

The front queues manage which pages are most important to crawl.

• Structure: There are K front queues (numbered 1 to K).

• **Prioritizer**: An incoming URL is assigned an integer priority (1 to K) by a Prioritizer and appended to the corresponding queue. Priorities can be based on heuristics, such as the refresh rate sampled from previous crawls, or be application-specific (e.g., favoring news sites).

• **Selection**: When a back queue needs a URL, the Biased front queue selector chooses a front queue from which to pull a URL. This selection process may be randomized or a round robin approach biased toward queues of higher priority.

2. **Back Queues (Politeness)**

The back queues enforce the politeness constraints.

• **Invariants**: Each of the B back queues must contain URLs from only a single host. A table is maintained to map host names to their corresponding back queues. Ideally, each back queue is kept non-empty while the crawl is in progress.

• **Politeness Mechanism (Heap)**: Politeness is enforced using a Back queue heap. The heap contains one entry for each back queue, storing the earliest time (t_e) at which the host corresponding to that queue can be accessed again. This time is calculated based on the last access and a chosen time buffer heuristic.

• **Processing**: When a crawler thread needs a URL, it extracts the root of the heap and fetches the URL at the head of the corresponding back queue (q). If queue q becomes empty, the system pulls a URL (v) from the front queues to replenish it, ensuring the politeness invariant is maintained.

• **Recommended Size**: To ensure all crawler threads remain busy while respecting politeness, the Mercator scheme recommends having three times as many back queues (B) as there are crawler threads.

Duplication: Exact match can be detected with fingerprints

Near-Duplication: Approximate match (Compute syntactic similarity with an edit-distance measure)

Shingles (Word N-Grams) can be used for Jaccard coefficient.

Computing exact set intersection of shingles between all pairs of documents is expensive, Approximate using a cleverly chosen subset of shingles from each (a sketch), Estimate (size_of_intersection / size_of_union) based on a short sketch.

Web scraping is basically extracting data from websites in an automated Manner. Its automated.

The 3 Steps of Web Scraping:

1**. Request-Response**: first step is to request the contents of a specific URL from the target website. In response, the scraper receives the requested information in HTML format.

2. **Parse and Extract**: Parsing is the general process of taking code as text and producing a structured representation in memory that a computer can work with. In web scraping, HTML parsing involves taking the HTML code and extracting the specific, relevant information, such as the page title, headings, paragraphs, links, or bold text.

3. **Download Data**: The final step is to download and save the extracted data. This is typically stored in a structured format like CSV, JSON, or a database, allowing the data to be easily retrieved, used manually, or employed by other programs.

Data Scraping	Data Crawling
Involves extracting data from various sources including web	Refers to downloading pages from the web
Can be done at any scale	Mostly done at a large scale
Deduplication is not necessarily a part	Deduplication is an essential part
Need crawl agent and parser	Needs only crawl agent

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$

L6- Text Classification Using Machine Learning

Text Classification: An NLP task where the goal is to categorize or predict the class of unseen text documents using supervised machine learning.

Types: Rule Based, Machine learning Based

Rule-based Classification: Uses manually constructed language rules to categorize text based on specific keywords or linguistic elements. While intuitive, this method is time-consuming and difficult to scale or maintain.

Machine Learning-based Classification: Involves supervised learning, where models are trained using raw text data and corresponding labels. Text **Preprocessing Steps**: Raw Text -> Tokenization -> Text Cleaning -> POS Tagging (Part of Speech Tagging): assigns a grammatical category (like noun, verb, adjective) to each token. For instance -> Stopwords Removal -> Lemmatization ->cleaned text -> ML model

Machine Learning: **Supervised Learning**: The model learns from a labeled dataset to make predictions (regression, classification), **Unsupervised Learning**: The model works with data without labels, often used in clustering, **Reinforcement Learning**: The model learns from actions and feedback to maximize reward.

Feature extraction: converting text data (strings) into numeric features so machine learning model can be trained.

Bag of Words (BoW): Represents text as a collection of words with frequencies in a document using a vector format, ignoring grammar and word order.

TF-IDF (Term Frequency-Inverse Document Frequency):TF measures how often a word appears in a document, while IDF evaluates the importance of a word across all documents. Helps in identifying meaningful terms in a document.

Why tf-idf better: considers the frequency of the words in the document, as well as the inverse document frequency so its likely to find important words more.

Model Evaluation:

Validation Set Approach: Splits data into training and testing datasets to evaluate model performance, **Cross-Validation**: Data is divided into multiple subsets, training and testing the model on different combinations to ensure robustness,

Confusion Matrix: A table used to evaluate the performance of a classification model by comparing predicted versus actual outcomes (e.g., accuracy, precision, recall), **Accuracy:** Proportion of correctly predicted instances(1-MCE), **Misclassification Error**: The proportion of incorrect predictions.

Underfitting refers to a model that can neither model the training data nor generalize to new data. **Overfitting:** Occurs when the model learns the training data too well, including noise, which harms its generalization ability on new data. **Reasons** - Complex Model – use simpler models with few params, Insufficient Training Data – gather more labeled data, test set is used during the training – separate test and training sets, Too Many Epochs in Training – early stopping **Solutions**: Regularization, ensemble methods, and data augmentation. **Class Imbalance**: When one class has more instances than others, causing the model to bias predictions toward the majority class.

Solutions: Undersampling: Reduce the number of instances in the majority class.

Oversampling: Increase the minority class by duplicating samples or using techniques like SMOTE (Synthetic Minority Over-sampling Technique). **SMOTE** generates synthetic samples for the minority class by interpolating new points between existing ones. each sample in the minority class, it selects k nearest neighbors from the same class.

Ensemble Methods: Such as Balanced Random Forest, to focus on correctly classifying minority class instances. **Balanced Random Forest (BRF**) is an extension of the traditional Random Forest algorithm, specifically designed to address the problem of imbalanced classes. **Key techniques of BRF: Bootstrap sampling** is the process of randomly sampling the dataset with replacement to create multiple subsets. These subsets are used to train different decision trees in the Random Forest model, **Balancing Class Weights**: During the training of each decision tree, BRF assigns higher weights to minority class instances. **Voting or averaging** refers to combining the predictions of the individual decision trees in the Random Forest.