Eksamensnotater IN4120 H23

Truls Hestetræet

Contents

Approximate nearest neighbors....3
Suffix arrays....3
Mean Average Precision (and Precision@K)....3
Support vector machines....3
Bias variance tradeoff....3
K-fold cross validation....3
Soft margin....3
The kernel trick....4
Heaps' Law....4
NDCG....4
DCG....4
Edit Distance....4
Weighted edit distance....4
Levenshtein....5
Cosine Similarity....5
Encoding....5
Gamma encoding....5
Elias' Gamma encoding....5
Example (X=10)....5
Variable Bytes (VB) encoding....6
Gap encoding....6
Simple-9 encoding....6
P-for delta....6
Page rank....6
Random surfer model....6
Markov chain....7
Skip lists....7
DAAT and TAAT....7
F1-score....8
Rocchio....8
Tf-idf....8
Term-, document- and collection frequency....8
Term frequency score....8
Inverse document frequency....8
Tf-idf weighting....8
Score given query....9
Precision & recall....9
Wildcard index handling....9
Permuterm index....9
K-gram index....9
Naïve Bayes....9
Bloom filter....10
Aho-Corasick....10
kNN....10
Field/zone search in inverted index....10
Break-even point....10
Distance....11
Euclidean....11
Manhattan....11
Static quality score....11
Stemming....11
Porter's algorithm....11
Zipf's law....11
Clicks for evaluation....11
Kendall tau....12
Exam 2022 example....12
Lemmatization....12
BSBI and SPIMI....12
Distributed indexing....12
Logarithmic merge....12
Mapreduce....12

Dynamic indexing 13
Query/document processing 13
Query expansion 13
Stop words 13
Tries 13
Boolean retrieval 14
Micro- and macro-averaging 14
Rocchio 1971 14
Generative and discriminative 14

Approximate nearest neighbors

- Find close neighbors faster than kNN at the cost of precision
- Voronoi diagrams: partitioning a plane into regions closest to a given set of objects (Voronoi cell around each object)
- Tree-based: Recursively build a tree until leaf nodes have small enough partitions
- Locality-sensitive hashing: Apply multiple hash functions h to each point to bucket them
- Quantization: Cluster vectors to reduce size of dataset
    - Replace each vector with a leaner, approximate and quantized representation (cluster)
- Graph-based: Construct hierarchical graphs (somewhat similar to skip lists as graphs)

Suffix arrays

- Take all suffixes of string, and give numeric value from 0 to (len(string) – 1)
    - The indexes the "pointer has moved" into the original string
- Sort suffixes lexicographically (this is now a suffix array)
- Can do binary search on the suffix array

Mean Average Precision (and Precision@K)

- Mean of precision across multiple queries
- "Average precision" is the average of precisions within a single query
- Precision is the precision at a specific cut-off point

$$p@k(x) = \frac{relevant\ docs\ in\ [0, x]}{x}$$

- Average precision: Add $p@k$ for all relevant docs, and divide by number of relevant docs
- MAP: Repeat for all queries, add together, and divide by number of queries

Support vector machines

- Basic idea: Find hyperplane with maximum margin of separation
- Creating a decision boundary as far away as both classes as possible
- Locating the "closest" elements from each class, and creating support vectors from these
- The decision boundary is exactly in the middle of the support vectors

Bias variance tradeoff

- Bias: Underfitting (not specified enough for the data/ too generalized)
- Variance: Overfitting (the decision boundary fits too perfectly for the training set)

K-fold cross validation

- Dividing the dataset into k partitions
- Training the model on k-1 partitions and testing on the last
- Repeat training for every possible version of k-1 partitions
- Keep score of which performs best
- Choose the best classifier after all k iterations

Soft margin

- If the training data is not linearly separable, slack variables can be added to allow misclassification of difficult or noisy examples

- Still try to minimize training set errors and place hyperplane “far” from each class

The kernel trick

- Used when mapping to a higher dimension with SVMs
- If there is no linearly separable decision boundary, the vector space can be mapped to a higher dimension, where the classes are linearly separable
- Replace dot product of mapping function to kernel function
- There are different kernels most appropriate to different classification tasks
- Kernel function must be continuous, symmetric and positive definite

Heaps’ Law

- Empirical formula to measure vocabulary in a corpus
- M is the size of the vocabulary. T is the number of tokens in the collection
- Constants with values usually being: $30 \leq k \leq 100$ and $b \approx 0.5$

$$M = kT^b$$

NDCG

- Normalized discounted cumulative gain
- The ideal ranking would return the most relevant documents first, then the next highest…
- NDCG will always be a number from 0 to 1
- Normalize DCG at rank n by dividing query’s DCG by ideal DCG (max possible DCG) up until position p

$$NDCG = \frac{DCG_p}{IDCG_p}$$

DCG

- Popular measure for evaluating web search
- 2 assumptions:
  - Highly relevant docs are more useful than marginally relevant docs
  - The lower the ranked position, the less useful
- Value r is a ranking value (e.g., 0-3)
- Discounted gain is score for a single result based on relevance and position

$$DG = \frac{r_x}{log_2 x}$$

- DCG is the sum of the DG scores for all returned docs up to position n

$$DCG = r_1 + \frac{r_2}{log_2 2} + \frac{r_3}{log_2 3} + \frac{r_n}{log_2 n}$$

Edit Distance

- The number of operations to convert a string to another
- Operations counting as 1: Insert, delete, replace (and sometimes transposition)
- Can use n-gram overlap to reduce candidate dictionary terms

Weighted edit distance

- the weight of an operation depends on the characters

    - The weight is based on the distance between characters on a keyboard (more likely to misspell close characters)
    - Requires weight matrix as input
- Can either show user all possible corrections, or the single best (disempowers user but saves a round of interaction and is faster)

Levenshtein

- Make table with empty string plus all characters in both strings
- Calculate edit distance from empty to all substrings (both vertically and horizontally)
- If characters are the same, insert number value equal to diagonal above left
- Else, choose the lowest value from left, above of diagonal up left, and add 1

Cosine Similarity

- With docs in a vector space, distance is a bad measure of likeness
- Since "word" and "wordword" would be far apart even though they are similar
- Better to rank docs according to angle
- Ranking by cosine(query, doc) is the same as ranking in decreasing order of angle

$$cosine(A, B) = \frac{A \cdot B}{||A|| * ||B||}$$

$$||x|| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}$$

Encoding

Gamma encoding

- Convert document to binary and remove first bit 1
- Write length of of binary as unary and add 0
- Concatenate step one and 2
- Example:

$$13 \rightarrow 1101 \rightarrow 101 \rightarrow 1110 + 101 = 1110101$$

Elias' Gamma encoding

Encoding

- Find largest N where $2^N \leq X$
- Encode N using unary coding (N zeros, 1 one)
- Append the binary of integer $X - 2^N$ using N digits

Decoding

1. Decode unary code from start (this is N) until current digit is 1
2. Read the remaining N digits (corresponds to $X - 2^N$)
3. Add together result from step 2 and 2^N

Example (X=10)

Encoding

1. $N = 3 \rightarrow 2^3 = 8 \leq 10$
2. $unary(N) = 0001$

3. $unary(N) + binary\ with\ N\ digits(X - 2^3) = 0001 + 010 = 0001010$

Decoding

1. $N = 000 = 3$
2. $d = 010 = 2$
3. $X = d + 2^N \rightarrow X = 2 + 2^3 \rightarrow X = 10$

Variable Bytes (VB) encoding

- Every byte starts with a bit telling if this is the last byte in the encoding
    a. 1 means it is last, 0 is not
- Then, the remaining 7 bits are used for the number itself (0-127 for single byte)
- Reading bytes as single number until 1 is the first bit of a byte

Gap encoding

- The size of the gap between two postings list doc IDs as variable byte encoding with either 1 or 2 bytes (from exam 2020)

Simple-9 encoding

- Allocated 32 bits
- Try to pack several numbers into the 32 bits
- 4 control bits and 28 data bits
- 9 possible cases for how many bits per word
    a. From 1 28-bit number to 28 1-bit numbers
- 4 control bits to store what 9 cases should be used

P-for delta

- Idea: Compress/decompress many values at a time
- How many bits per number?
    a. Choose so 90% can be encoded, and handle 10% as exceptions
- Example: for next 128 numbers, most numbers are <= 32, so 5 bits is usually sufficient
    a. Exceptions are stored as 4-byte ints at the end
    b. To decode, copy 128 numbers into int array and traverse linked list and patch exceptions
    c. Always uncompress the next 128 posts into temp array.
        i. Use max among next 128 numbers to choose number of bits
        ii. 10-20% better compression with basically same speed

Page rank

- If a random surfer traverses the web, what is the fraction of time the surfer would be on page d?
- Fundamentally based on calculating stationary distributions of Markov chains
- Since Markov chains can't handle dead ends, a random surfer with the ability to teleport to other sites is better
- The PageRank algorithm is the distribution of how much a user would be on a page
- The power iteration method is one way to calculate the PageRank

Random surfer model

- "Randomly" visiting web pages
- The surfer either follows a direct link from the current page or teleports

$$p(specific\ direct\ link) = \frac{1-\alpha}{direct\ links}$$

- To avoid getting stuck, α is the probability of random teleport

$$p(specific\ teleport) = \frac{\alpha}{websites\ not\ linked}$$

- The surfer will always either teleport or follow a direct link

$$p(teleport) + p(direct\ link) = 1$$

- The random surfer walk can be represented as a Markov chain

Markov chain

- The transition of states with probabilistic rules
- Can be represented as directed acyclic graph with edges representing the probability of transition from one state to another (from one website to another in the context of pagerank)
- The future state only depends on the current state, never the previous states
- The sum of all possibilities from a state is always 1
- Ergodic: A unique stationary distribution exists for irreducible and aperiodic Markov chain. In other words, if it is possible to go from every state to every state
- Probability matrix: The matrix used to describe the probabilities of each transition within a Markov chain

Skip lists

- Can be used if index is not changing to often
- Can check if the skip successor is lower than the next element in the other list
- Tradeoff between number of comparisons and successful skips. Can be a good alternative to place pointers on every sqrt(L)
    a. Ignores the distribution of query terms
    b. Easy if the index is relatively static
    c. The extra time of loading bigger posting lists might outweigh the gains from memory merging

DAAT and TAAT

DAAT: Making a score for a single document, and then moving on to the next

- Pointer to all postings lists that are not zero
- Adding the values for all the terms pointing to the same lowest docID
- When values are added, lowest pointers are increased, and the next lowest document will be added scorewise
- Repeat until all postings lists (or less than m) are null
- Only emits non-zero docs

TAAT: Compute the scores for all documents at the same time incrementally

- Fetch the entire postings list for the current query term
- Add value to every document's score
- Fetch entire postings lists for every term until all terms are processed
- Incrementing the score of the document by the term-value multiplied by the number of occurrences the term has in the document
- Extract the result set (return the non-zero entries)
- O(n +N) (total length of all lists + #docs)

F1-score

- A Harmonic mean between precision and recall (conservative average)

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Rocchio

- Finds the centroid for each class
- A new point x will be classified as the closest centroid (lowest distance)
- Fails to deal with non-contiguous regions
- Assumes the classes correspond to spheres of equal radii

Tf-idf

Term-, document- and collection frequency

- Term frequency $tf(t, d)$: The number of times term t occurs in document d
- Document frequency: The number of documents d that contain term t (invers measure of the informativeness of t)
- Collection frequency: The number of occurrences of term t in a collection, counting multiple occurrences. If a term t is present exactly once in the documents it is present in, $cf_t == df_t$

Term frequency score

- We want positive weights for common words, but not as high as rare terms
- Log frequency weighting gives the score of a document query pair. It is 0 if none of the query terms are in the document
- For all terms t element in both query q and document d, sum $(1 + \log(tf_{t,d})$

$$LF\ score = \sum_{t \epsilon q \cap d} (1 + \log(tf_{t,d})$$

Inverse document frequency

- Idf weight is the inverse document frequency weight
- Log used to dampen effect of idf. Base does not matter since it is a constant
- The idf of t is defined as:

$$idf_t = log_{10}(\frac{N}{df_t})$$

Tf-idf weighting

- The product of tf and idf weight

$$tf.idf_{t,d} = \log(1 + tf_{t,d}) \cdot log_{10}(\frac{N}{df_t})$$

- Increases with number of occurrences in document d
- Increases with the rarity of the term in the collection

Score given query

- Total score is calculated as the sum of tf-idf weights from all terms present in both query q and document d

$$score_{q,d} = \sum_{t \epsilon q \cap d} tf.idf_{t,d}$$

Precision & recall

- Precision: Of my retrieved docs, how many are relevant?

$$p = \frac{tp}{tp + fp}$$

- Recall: Of my relevant docs, how many were retrieved?

$$r = \frac{tp}{tp + fn}$$

- Accuracy (classification): Of all docs, how many docs were classified correctly?

$$a = \frac{tp}{N}$$

Wildcard index handling

Permuterm index

- Used for general wildcard queries
- All rotations of a term are linked to the original vocabulary term
- When handling a wildcard query, rotate until * is at the end. Then, via a search tree, the string can be searched in the permuterm index, locating wildcard terms that fit the query
- Allows to locate original vocabulary terms from wildcard queries
- After finding the original term, it can be used in an inverted index
- A drawback is the increased size, since all rotations are stored

K-gram index

- Enumerate k-grams occurring in any term
- Maintain a second inverted index from k-grams to dictionary terms that match each bigram
- Useful to find terms with low edit distance to query. Also possible to only find terms with multiple k-grams matching with query
- Can use Jaccard coefficient to measure overlap between k-grams

Naïve Bayes

- Machine learning model to classify one of multiple classes

$$class = argmax(P(c|d))$$

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

- Denominator is usually ignored since it is a normalization factor

$$Posterior = Likelighood \cdot Prior$$

- Likelihood is the probability of the document given the class
- Prior is the probability of the class itself
- Posterior is the probability of the class given the document

Bloom filter

- Probabilistic data structure for checking set membership
- No false negatives, maybe false positives
- Runs query through hash functions and checking resulting values against bit-array
- If all indexes after hash functions are 1 in array, element is probably member. If not, it is definitely not member
- m bits in array and k hash functions
- In order to add element to set, hash element with all hash functions, and set bit in corresponding memory location to 1
- P(fp)= f(m, k, n)

Aho-Corasick

- Trie with all terms in vocabulary
- Failure links: For every node, connect a link to the node representing the longest suffix that exists of that node

kNN

- Classifies based on the k closest neighbors
- Must search all neighbors to find k closest
- Can make complex decision boundaries

Field/zone search in inverted index

- Term.field as term in the inverted index
    - Fast to search for something in a specific field
- Representing fields by adding name of field to the elements in the postings list
    - Fast if we want to search all fields
    - Slow for specific field
- Creating an inverted index per field
    - Fast for searching specific fields
    - Uses more space
    - Don't need the entire index in memory, can use only the one for the field we are searching in

Break-even point

- In a ranked list of documents at a position k, where precision equals recall
- Can only be one breakeven point, unless precision and recall are 0 (then there can be multiple)

Distance

Euclidean

- The distance in a straight line between two points
- Using Pythagoras theorem

Manhattan

- The sum of distances for all axis
- Euclidean can measure a straight line between two points, Manhattan follows the directions of the dimensions between the points

Static quality score

- We want top ranking docs to be relevant and authoritative
- Authority is usually query-independent
- Quality score [0, 1] can be assigned to a doc and be presented as g(d)
- Can now scale a quantity (e.g., citations) to a number [0, 1]
- Net score is relevance and authority

$$net.score_{q,d} = g(d) + cosine(q, d)$$

Stemming

- Reduce terms to their "root" before indexing
- Automates, automatic, automation -> automat

Porter's algorithm

- For English words
- 5 rules to reduce and of term
- Sequence of stripping suffixes
- Are not real words (are chunks of words)
- Makes mistakes (policy, police -> polic)

Zipf's law

- The i-th most frequent term has frequency proportional to $\frac{1}{i}$
- Collection frequency of term i is equal to $\frac{K}{i}$ where K is a normalizing constant

$$cf_i = \frac{1}{i} = \frac{K}{i}$$

Clicks for evaluation

- Strong position bias
- Can use pairwise relative ratings: docA better than docB
    - Doesn't mean docA is relevant for query
    - Assess in terms of conformance with historical pairwise preferences recorded from user clicks
- If there is a significant spike compared to expected results, it might imply the document should be ranked higher

Kendall tau

- Measure of relative distance between rankings
- X is the number of agreements, and Y the number of disagreements
- 1 is perfect agreement and -1 is perfect disagreement

$$KT = \frac{X - Y}{X + Y}$$

Exam 2022 example

$P_{foo} = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$

$R_{foo} = [1,3,2,4]$

$X = 5, Y = 1$

$KT = \frac{5-1}{5+1} = \frac{4}{6} = 0.667$

Lemmatization

- Reduce a word to its base form
    - Am, are, is -> be

BSBI and SPIMI

- BSBI: When there are too much to store in memory, divide into blocks
    - Accumulate postings for each block, sort write to disk
    - Merge blocks into one long sorted order
    - Can do binary merges with layers
    - During each layer, read into memory runs in blocks, merge and write back
    - More efficient to do a multi way merge where we are reading form all blocks simultaneously
    - Not killed by disk seeks if block chunks are large enough
- SPIMI: 2 key ideas
    - Generate separate dictionaries for each block – don't need term-termID mapping across blocks
    - Don't sort. Accumulate postings in postings lists as they occur
    - Can generate a complete inverted index for each block
    - These separate indexes can be merged into one big index
    - Compression makes SPIMI more efficient

Distributed indexing

Logarithmic merge

- Maintain a series of indexes, each twice as large as the previous one
- Keep smallest in memory
- If smallest gets too big, write to disk as first on disk, or merge with first on disk
- Each posting is merged O(log T) times, so complexity is O(T log T)
- Main and auxiliary is faster for query processing, but log index is faster for index construction

Mapreduce

- Map: Collection -> (termID, docID)

- Reduce: (<termID1, list(docID)>, <termID2, list(docID)>, …) -> (postings list 1, postings list 2, …)
- Map phase maps the terms to their respective document and passes to correct inverter based on lexicographic value. Then, the inverter reduces to termID-postings list
- MAP finds terms in docs, Reduce makes and handles the postings lists for the terms
- Since map won't find all the terms with the same lexicographic value, this is handled in reduce. Map only sends it to the correct place

Dynamic indexing

- Documents are often inserted, modified and deleted from the corpus
- Must update postings for terms already in dictionary and add new terms to dictionary
- Simplest approach: Maintain big main index, new docs go into small auxiliary index
    - Search across both, merge results
    - Can use invalid bit-vector for deleted docs
    - Periodically re-index into one main index
- This might be a problem for frequent merges
- Merging the auxiliary index into the main index is efficient if we keep a separate file for each postings list

Query/document processing

- Need to reduce both (if any)
- Only need to expand query
- Query expansion might find relevance that reduction algorithms wouldn't
- Reducing to base form might lose some information
- Need to have some knowledge of what equals what when expanding query

Query expansion

- In relevance feedback, users give input(relevant/not) on docs, which is used to reweight terms in query for the docs
- In query expansion, users give input (good/bad search term) on query words or phrases, possibly suggesting additional query terms
- Web search engines suggest related queries in response to a query. The user can then use one of these alternatives
- Improve recall by adding related words to a query, increasing the number of returned docs

Stop words

- Exclude the most common words from the dictionary
- 30% of top 30 words are "most common"
- The trend is going away from this practice
    - Good compression and query optimization negate the need
    - Some phrases need stop words ("King of Denmark")

Tries

- Tree structure. Each node is usually a character
- The path from the root node to a leaf node will be a string in the vocabulary
- Allows for very fast lookups of either word or prefixes

Boolean retrieval

- Ask a query that is a Boolean expression
    - Returns Boolean matches
- Boolean queries use AND, OR and NOT to join query terms

Micro- and macro-averaging

- Macro: Compute performance for each class, then average
- Micro: Collect decisions for all classes, compute contingency table, evaluate

Rocchio 1971

- Moves query in direction of centroid of known relevant docs
- Tries to find the query that has the best cosine similarity with known relevant docs and lowest wit known non-relevant docs
- New query moves toward relevant docs and away from irrelevant docs

Generative and discriminative

- Generative: Estimate P(c|x) indirectly
    - Generative because they can generate new datapoints
    - Explanatory power
    - Tries to model “more” than what is necessary for the task
- Discriminative: Directly estimates the posterior
    - Logistic regression
    - Usually more accurate when lots of data is available