universität koblenz
weiter:denken

Web Retrieval

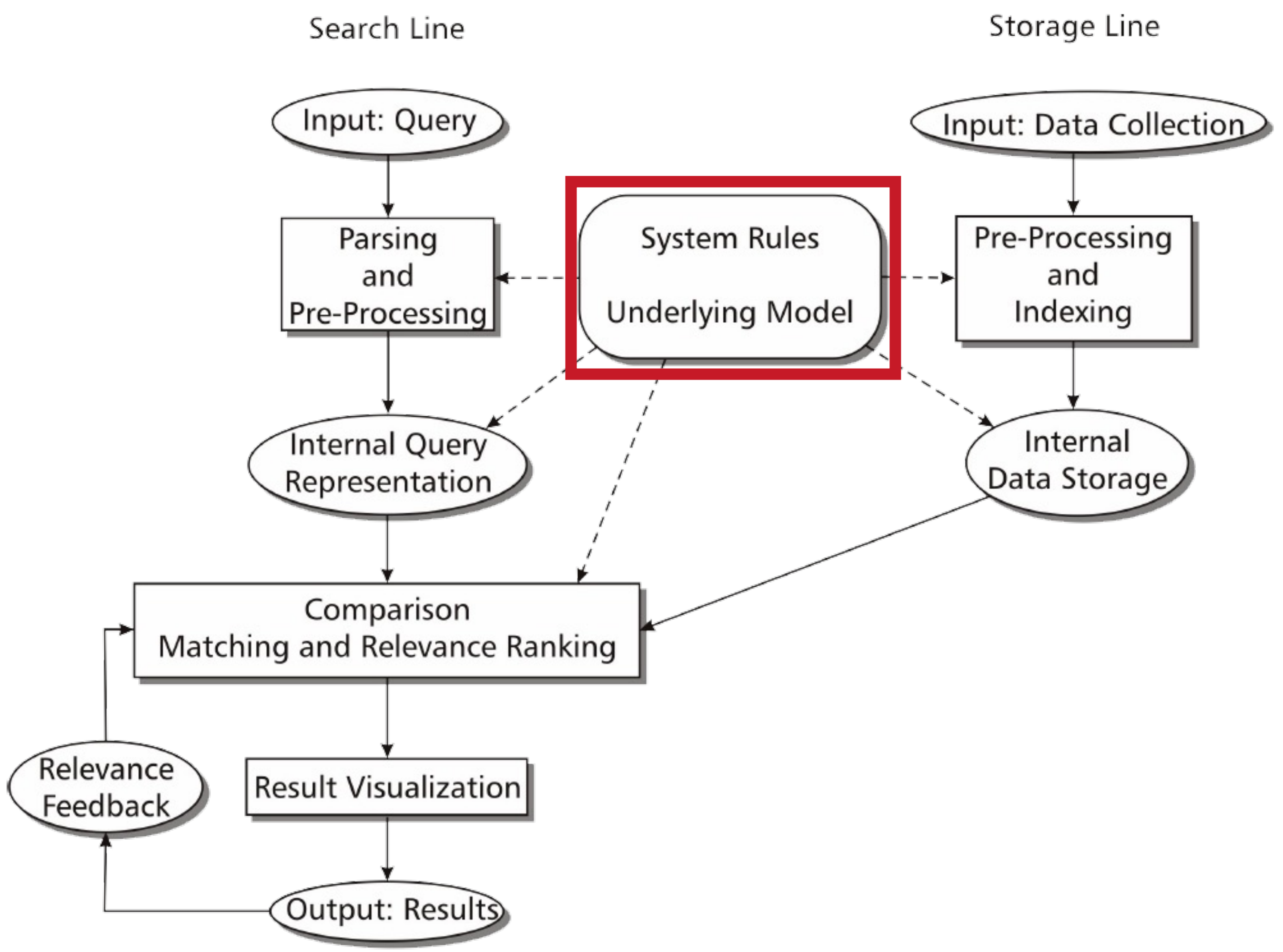
Underlying models (III): Language Models for IR

Frank Hopfgartner
Institute for Web Science and Technologies

Recapitulation

- Boolean Model
- Ranked retrieval model
- Scoring
- Term frequency
- Document frequency
- TF-IDF
- Vector Space Model
- Relevance feedback

IR System Architecture



Intended Learning Outcomes

universität koblenz
weiter:denken

At the end of this lecture, you will be able to:

- Know what a language model is
- Understand differences between different language models (unigram, bigram)
- Understand how to use language modeling for information retrieval
- Learn about different smoothing schemes for LM for IR
- Be able to compare LM for IR with the vector space model

Outline

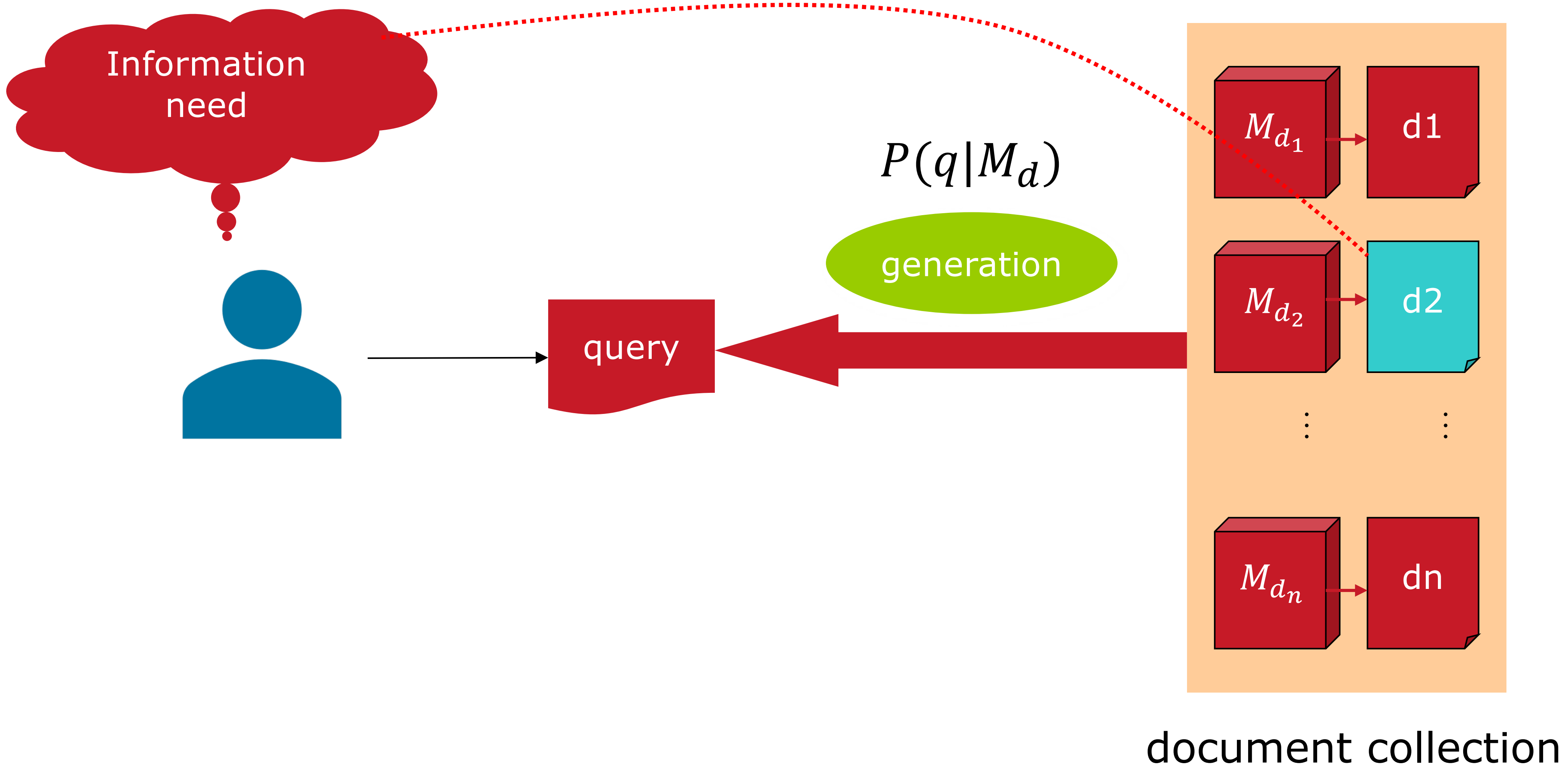
- Motivation
- Language Models
  - Unigram LM
  - Bigram LM
- Query likelihood model for ranking
- Smoothing schemes
- Pre-trained large language models

1. Motivation

Language model for IR: basic concept

- IR views documents as models and considers queries as strings of texts randomly sampled from models
- Users have a reasonable idea of terms that are likely to occur in document of interest. They use words that they expect to find in matching documents as their query
- The LM approach directly exploits this idea! Documents are ranked according to the probability of observing a query q in repeated random samples from the document model $P(q|M_d)$

Language modelling for IR



Language models: in brief

- Novel way of looking at the problem of text retrieval based on probabilistic language modeling
  - Conceptually simple and explanatory
  - Formal mathematical model
  - Natural use of collection statistics, not heuristics
  - LMs provide effective retrieval and can be improved to the extent that the following conditions can be met
    - Our language models are accurate representations of the data
    - Users have some sense of term distribution

Comparison with Vector Space Model

- Similarities
  - Term weights based on frequency
  - Terms often used as if they were independent
  - Inverse document/collection frequency used
  - Some form of length normalization useful
- Differences
  - Based on probability rather than similarity
    - Intuitions are probabilistic rather than geometric
  - Aspects such as usage of document length and term, as well as document and collection frequency differ

2. Language Models (LMs)

Language Modeling (for Information Retrieval)

universität koblenz
weiter:denken

- Language models are **probabilistic models** that capture the probabilities of sequences of words in a language

  - **Unigram model**: How likely is the word “frodo” to appear (in a language)?
    - p(“frodo”) = ?

  - **Bigram model:** given that current word is ”frodo”, what is the probability of next word being “baggins”?
    - p(“baggins” | “frodo”) = ?

  - **Trigram model:** given the current sequence “bilbo baggins”, what is the probability of the next word being ’shire”
    - p(“shire” | “frodo baggins”) = ?

Language Modeling (for Information Retrieval)

- We use the instantiations of the language to estimate the probabilities of words and sequences
  - i.e., large corpora - the larger the corpora, the better the approximation of the true word distributions in language

- In other applications we build language models so large until we cannot compile them anymore

- In Information Retrieval, we build language models
  - from individual documents
  - from the whole document collection

Language Modeling (for Information Retrieval)

- Language models for IR model the **query generation process**
- Given a document **d** and a query **q**, what is the probability of the query being sampled from the language model of the document
- In other words, we want to estimate **P(Q = q | D = d)**
- The probability of a document generating a query is directly the function according to which we rank the documents
  - i.e., we rank the documents in decreasing order of **P(Q = q | D = d)**

Language Bowl metaphor

- Assume we have a document which contain the following occurrences of terms:
  - “frodo” (5x), “baggins” (3x), “sam” (3x), “shire” (2x), “gandalf” (2x), “orcs” (1x)
- Let’s represent each term with balls of one colour
  - “frodo” -> 5 blue balls, “baggins” -> 3 red balls, “sam” -> 3 yellow balls
  - “shire” -> 2 green balls, “gandalf” -> 2 orange balls, “orc” -> 1 purple ball
- We put all balls into one bowl and randomly take them out one by one

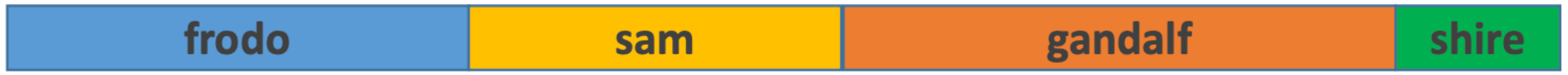
Language Bowl metaphor

- What is the probability of drawing a yellow ball?
  - P(yellow) = 3 / (5+3=3+2+2+1) = 3 / 16

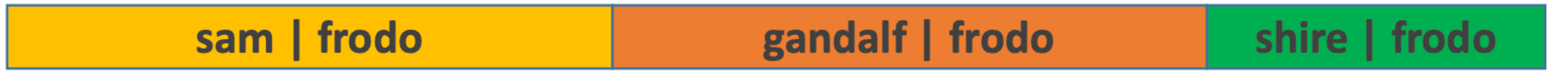
- What is the probability of drawing first an orange then a blue ball?
  - Replacement: P(orange,blue) = P(“gandalf”, “frodo”) = P(“gandalf”) * P(“frodo”) = 2/16 * 5/16
  - No replacement: P(orange,blue) = P(“gandalf”, “frodo”) = P(“gandalf”) * P(“frodo | “gandalf”) = 2/16 * 5/15

Language models – Generative story

- Language models can be observed as a **statistical model** for **generating data**
- Example (toy language, consisting of four words):
  - P(“frodo”) = 0.3, P(“sam”) = 0.25, P(“gandalf”) = 0.35, P(“shire”) = 0.1
  - P(“sam”|”frodo”) = 0.4, P(“gandalf”|”frodo”) = 0.4, P(“shire”|”frodo”) = 0.2
- Generative process
  1. Randomly draw the first word (e.g., from a uniform distribution)



  2. Draw the second word from conditional distribution of the first word (e.g., “frodo”)



Types of language models

- We want to estimate the probability of the sequence:

P(🟡 🟠 🔵 🟢) = P(🟡) * P(🟠|🟡) * P(🔵|🟡 🟠) * P(🟢|🟡 🟠 🔵)

- **Unigram language model**
  - Word independence = probability of the word does not depent on previous words
  - We ignore conditioning

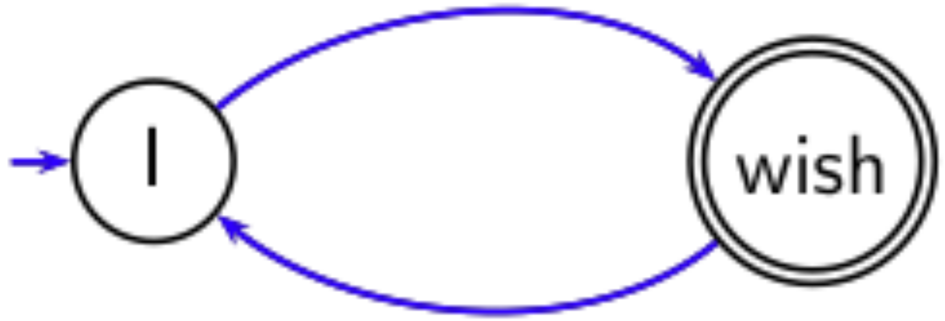
P(🟡 🟠 🔵 🟢) = P(🟡) * P(🟠) * P(🔵) * P(🟢)

- **Bigram language model**
  - The probability of word appearing depends only on the immediate preceding word
  - Containing only one word before

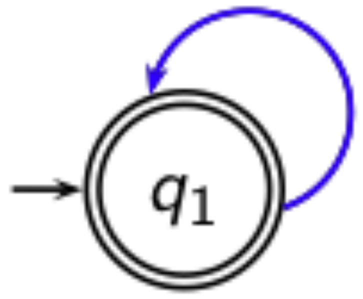
P(🟡 🟠 🔵 🟢) = P(🟡) * P(🟠|🟡) * P(🔵|🟠) * P(🟢|🔵)

Finite state automaton

- We can view a finite state automaton as a deterministic language model
  - I wish
  - I wish I wish
  - I wish I wish I wish
  - I wish I wish I wish ….
- Cannot generate: “wish I wish” or “I wish I”
- Our basic model: each document was generated by a different automaton like this except that these automata are probabilistic



A probabilistic language model



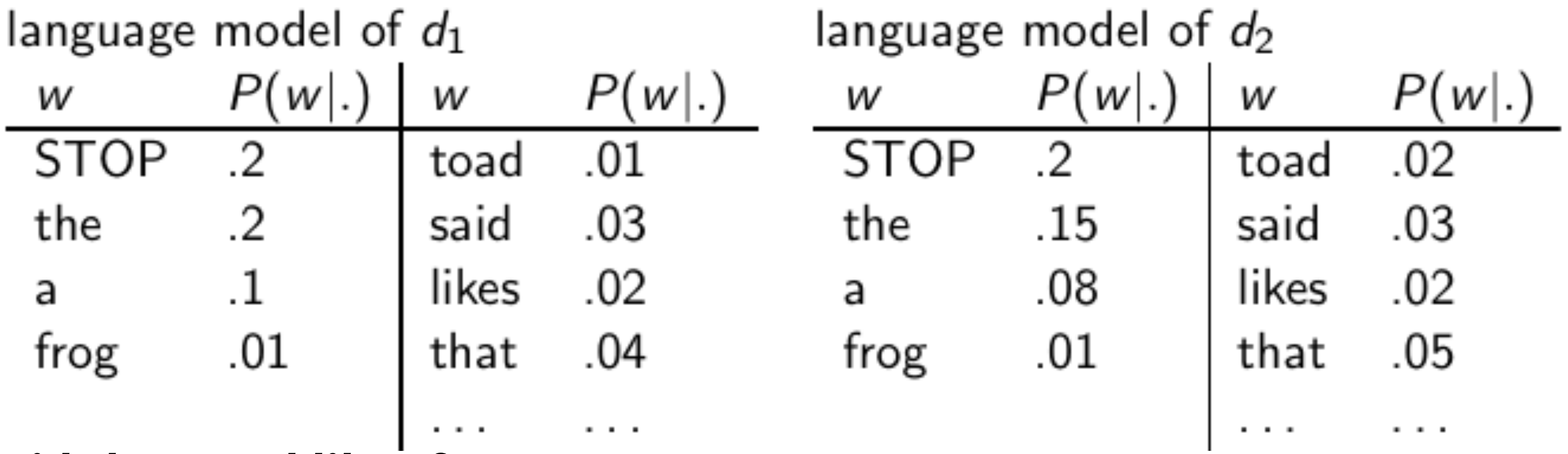
w	$P(w\|q_1)$	w	$P(w\|q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
		...	...

- This is an one-state probabilistic finite-state automaton – a **unigram** language model – and the state emission distribution for its one state q_1. STOP is not a word, but a special symbol indicating that the automaton stops. ***"frog said that toad likes frog STOP"***

$$P(string) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.02$$

$$= 0.0000000000048$$

A different language model for each document



language model of d_1

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.01
the	.2	said	.03
a	.1	likes	.02
frog	.01	that	.04
		...	...

language model of d_2

w	$P(w\|.)$	w	$P(w\|.)$
STOP	.2	toad	.02
the	.15	said	.03
a	.08	likes	.02
frog	.01	that	.05
		...	...

- ***frog said that toad likes frog STOP***
  - $P(string|M_{d1}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.02 = 0.0000000000048 = 4.8 \cdot 10^{-12}$
  - $P(string|M_{d2}) = 0.01 \cdot 0.03 \cdot 0.05 \cdot 0.02 \cdot 0.02 \cdot 0.01 \cdot 0.02 = 0.0000000000120 = 12 \cdot 10^{-12}$
  - $P(string|Md_1) < P(string|Md_2)$
- Thus, document d_2 is “more relevant” to the string “frog said that toad likes frog STOP” than d_1 is

Sparseness issue of language models

- Language models have a major issue
  - The longer the phrase, the harder it is to estimate its true probability in language
  - E.g., P("bilbo" | "frodo ran around house found ring") = ?
- Long phrases have very few appearances even in very large corpora
  - Impossible to compute reliable estimates of their conditional probabilities
  - This is why language models for N ≥ 3 are almost never used
- In practice, we use unigram and bigram language models
  - In IR setting, we build language models from invidual documents
    - Even bigram probability hard to estimate
  - In IR, we most often employ the unigram language model

Estimating probabilities

- For the unigram language model we need to estimate
  - P(term) for every term in the text
- For the bigram language model we additionally need to estimate
  - P(term | previous term) for every pair of terms that appear one after another
- **Q:** How do we estimate these?
  - Unigram language model
    - $P(t_i) = n_i / n_T$
    - n_i is the number of occurrences of term t_i in the collection
    - n_T is the total number of word occurrences (i.e., tokens) in the collection
  - Bigram language model
    - $P(t_i \mid t_{i-1}) = n(t_{i-1}, t_i) / n(t_{i-1})$
    - $n(t_{i-1}, t_i)$ is the number of occurrences of bigram $t_{i-1}t_i$ in the collection
    - $n(t_{i-1})$ is the number of occurrences of term t_{i-1} in the collection

Estimating probabilities – Examples

- We are given a toy collection consisting of three documents
  - d_1: “Frodo and Sam stabbed orcs”
  - d_2: "Sam chased the orc with the sword”
  - d_3: “Sam took the sword”

- Estimating word probabilities for the unigram model:

t_i	Frodo	Sam	orc	chased	sword	…
$P(t_i)$	1 / 16	3 / 16	2 / 16	1 / 16	2 / 16	…

- Estimating word probabilities for the bigram model:

t_{i-1},t_i	Frodo, chased	the, sword	the, orc	…
$P(t_i \mid t_{i-1})$	0	2 / 3	1 / 3	…

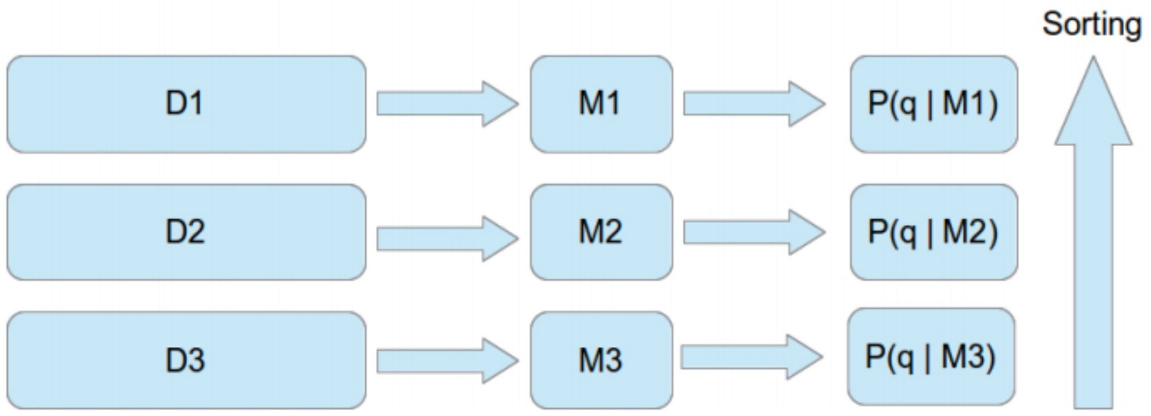
universität koblenz
weiter:denken

3. Query likelihood model for ranking

Query likelihood model for ranking

- Given a document collection D and a query q we need to estimate the probability P(q | d) for every document d in D
- In the **query likelihood model**, we estimate the probability P(q | d) as the probability that the language model built from d generates the query q

- Algorithm
  - Compute the language model M_i for every document d_i in D
  - Compute the probability $P(q \mid M_i)$ for every language model M_i



- **Intuition**: Language models of relevant documents should assign higher probability for the query

Query likelihood model for ranking – Example

- We are given a toy collection consisting of three documents
  - d_1: "Sam chased the orc with the sword"
  - d_2: "Frodo and Sam stabbed orcs"
  - d_3: "Sam took the sword"
- We are given the query "Sam and orc and sword"
- Let's rank the documents according to unigram LM for IR (ignore stopwords)

- **Step 1:** Compute language models of individual documents
  - M1: P("sam") = 0.25, P("chase") = 0.25, P("orc") = 0.25, P("sword") = 0.25
  - M2: P("frodo") = 0.25, P("sam") = 0.25, P("stab") = 0.25, P("orc") = 0.25
  - M3: P("sam") = 0.33, P("took") = 0.33, P("sword") = 0.33

Query likelihood model for ranking – Example

- We are given a toy collection consisting of three documents
  - d_1: “Sam chased the orc with the sword”
  - d_2: “Frodo and Sam stabbed orcs”
  - d_3: “Sam took the sword”
- We are given the query “Sam and orc and sword”
- Let’s rank the documents according to unigram LM for IR (ignore stopwords)

- **Step 2:** Let’s compute the probabilities $P(q \mid M_i)$
  - P(q | M1) = P(“sam” | M1) * P(“orc” | M1) * P(“sword” | M1) = 0.25 * 0.25 * 0.25
  - P(q | M2) = P(“sam” | M2) * P(“orc” | M2) * P(“sword” | M2) = 0.25 * 0.25 * **0**
  - P(q | M3) = P(“sam” | M3) * P(“orc” | M3) * P(“sword” | M3) = 0.33 * **0** * 0.33

4. Smoothing schemes

Smoothing language models

- **Zero frequency problem:** Models we've considered so far give probability of **0** to queries containing any term that does not occur in the document
- We can prevent this by using **smoothing techniques**
- Smoothing techniques
  - Change the probability distribution of terms in the language model
  - Assign some small probability to unseen words
- Three prominent smoothing schemes
  - Laplace smoothing
  - Jelinek-Mercer smoothing
  - Dirichlet smoothing

Laplace smoothing

1. Adds a fixed small count (often it’s 1) to all word counts
2. Renormalizes to get a probability distribution

$$P(t_i|M_d) = \frac{n_{i,d} + \alpha}{n_d + |V| \cdot \alpha}$$

- The probability of any unseen word equals

$$P(t_{uns}|M_d) = \frac{\alpha}{n_d + |V| \cdot \alpha}$$

Jelinek-Mercer smoothing

- Laplace smoothing assumes that all unseen words are equally likely
- **Jelinek-Mercer smoothing** (also known as **interpolated smoothing**)
  1. Additionally builds a language model M_D from the whole document collection D
  2. Interpolates between probabilities of the query according to the
     - **Local LM** – language model M_d built from the particular document d
     - **Global LM** – language model M_D built from the whole collection
- The probability of a word unseen in the document d still gets some probability from the global language model
  - Probability of an unseen word depends on its frequency in whole collection

Smoothing

- Jelinek-Mercer smoothing
  - Linear combination of corpus and document model

$$P(t|d) = \lambda \cdot P(t|M_d) + (1 - \lambda) \cdot P(t|M_c)$$

    - $0 < \lambda < 1$: parameter to adjust the influence of the document model

- Bayesian Update smoothing
  - Used for updating estimates in probabilistic relevance feedback
  - Adjusts estimates

$$P(t|d) = \frac{\mathrm{tf}_{(t,d)} + \alpha \cdot P(t|M_c)}{dl_d + \alpha}$$

    - α: parameter to adjust the influence of the document model

Dirichlet smoothing

- **Dirichlet smoothing** can be seen as a generalization of the Laplace smoothing
- Each word unseen in the document gets an artificial extra count
- But the extra count is not fixed, depends on the global probability of the term
  - In this respect, Dirichlet smoothing is similar to Jelinek-Mercer smoothing

$$P(t_i|M_d) = \frac{n_{i,d} + \mu \cdot P(t_i|M_D)}{n_d + \mu}$$

- Less frequent words in the document get more probability from the global component
  - The value of the constant *μ* determines the scale of the global probability's contribution

Example

- Query: „cup jar“
- Estimate term probabilities for document models, e.g.

$$P(\text{"jar"}|d_2) = \frac{2}{5}$$

t_j	d_1	d_2	d_3	d_4	d_5
P(coffee)	1	0	0.25	0.3	0
P(cup)	0	0.2	0.50	0.3	0
P(jar)	0	0.4	0.25	0.3	0.5
P(tea)	0	0.4	0	0.1	0
P(water)	0	0	0	0	0.5

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

Example

- Jelinek-Mercer smoothing

  $\lambda = 0.5$

- Estimate term probabilities for corpus
- models, e.g.

$$P(\text{"jar"}|M_c) = \frac{8}{25}$$

- Smooth document models

$$P(\text{"jar"}|d_2) = 0.5 \cdot 0.4 + 0.5 \cdot 0.32 = 0.36$$

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

t_j	C
P(coffee)	0.24
P(cup)	0.24
P(jar)	0.32
P(tea)	0.12
P(water)	0.08

Example

t_j	d_1	d_2	d_3	d_4	d_5
P(coffee)	0.62	0.12	0.25	0.27	0.12
P(cup)	0.12	0.22	0.37	0.27	0.12
P(jar)	0.16	0.36	0.28	0.31	0.41
P(tea)	0.06	0.26	0.06	0.11	0.06
P(water)	0.04	0.04	0.04	0.04	0.29

1. coffee, coffee
2. cup, jar, jar, tea, tea
3. coffee, cup, cup, jar
4. coffee, coffee, coffee, cup, cup, cup, jar, jar, jar, tea
5. jar, jar, water, water

- Compute retrieval values, e.g.

$$P(q|d_2) = 0.22^1 \cdot 0.36^1 = 0.0792$$

Rank	Document	ρ
1	d_3	0.105
2	d_4	0.084
3	d_2	0.079
4	d_5	0.049
5	d_1	0.019

Note: d_1 at rank 5 does not contain any search term!

Parameter choice for smoothing

- How to choose λ or α?
- Exploit reference corpus
  - Use (comparable) evaluation corpus
  - Optimise retrieval performance on goldstandard
- Incorporate knowledge about query
  - Query length
    - Long queries: strong smoothing
      - ➢ Missing terms have a less strong influence
    - Short queries: weak smoothing
      - ➢ All/most terms should be present

Language models for IR vs. VSM

- Let's compare the query likelihood model with the VSM model

1. Do we have a term frequency component in LM?
   - **Q:** do query terms that are more frequent in the document contribute more to the relevance score?
   - **A:** Yes! $P(t_i) = \mathbf{n_i} / n_T$
2. Do we have a document frequency component in LM?
   - **Q:** does the global document frequency of the query term affect the relevance scores?
   - **A:** No! If we use Jelinek-Mercer or Dirichlet smoothing, we take into consideration collection frequency, but not document frequency
   - However, mixing term frequency (within document) and collection frequency has an effect similar to using IDF

Language models for IR vs. VSM

- Let's compare the query likelihood model with the VSM model

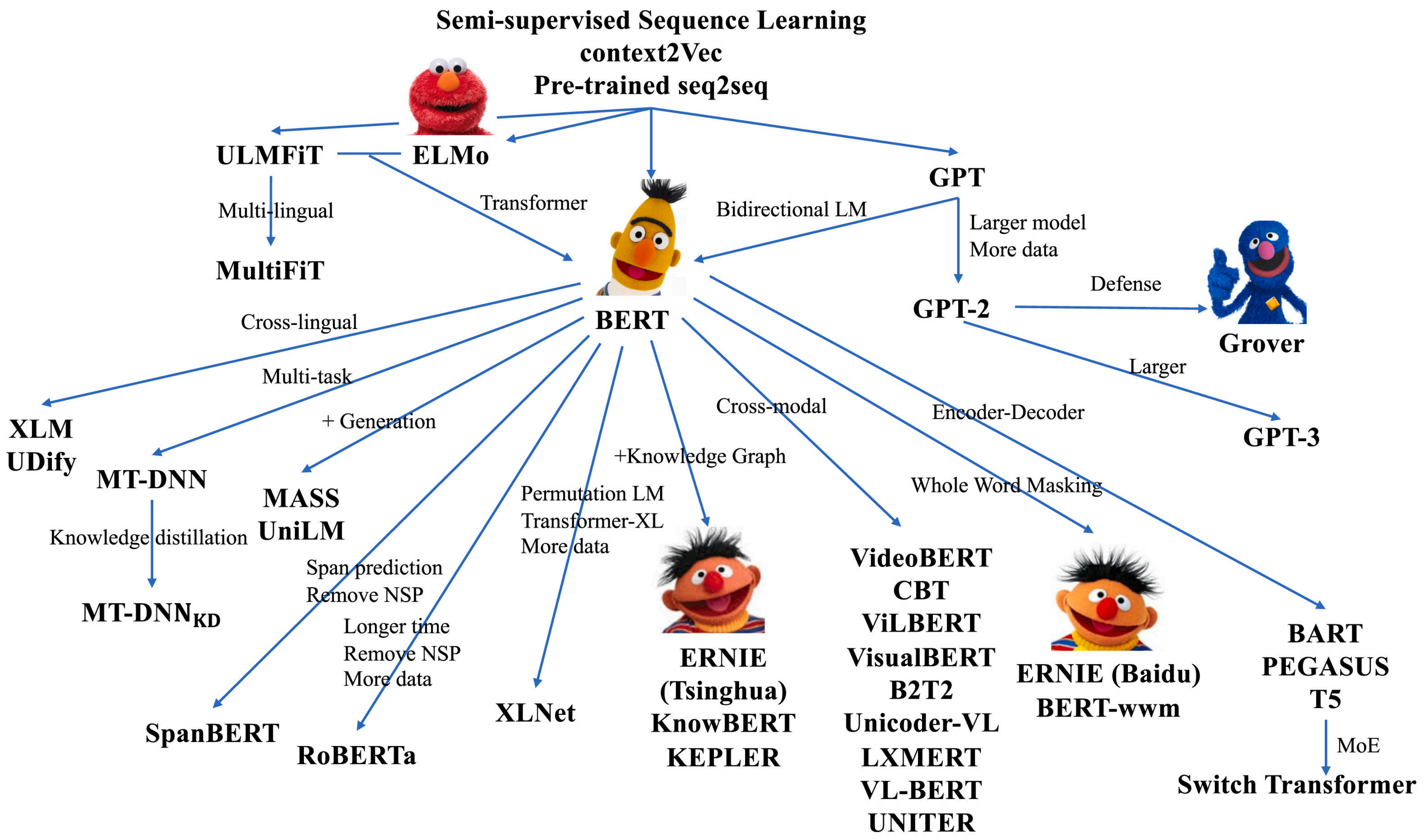
3. Does LM for IR account for different lengths of documents?
   - **Q:** Does it somehow normalize the frequencies of query terms in documents with the document length?
   - **A:** Yes! $P(t_i) = n_i / \mathbf{n_T}$

- LM for IR vs. VSM: **commonalities**
  1. Term frequency directly in the model
  2. Contributions of terms are normalized to account for document length
- LM for IR vs. VSM: **differences**
  1. LM for IR is based in probability theory, VSM in vector algebra
  2. Collection frequency (LM) vs. Document frequency (VSM)

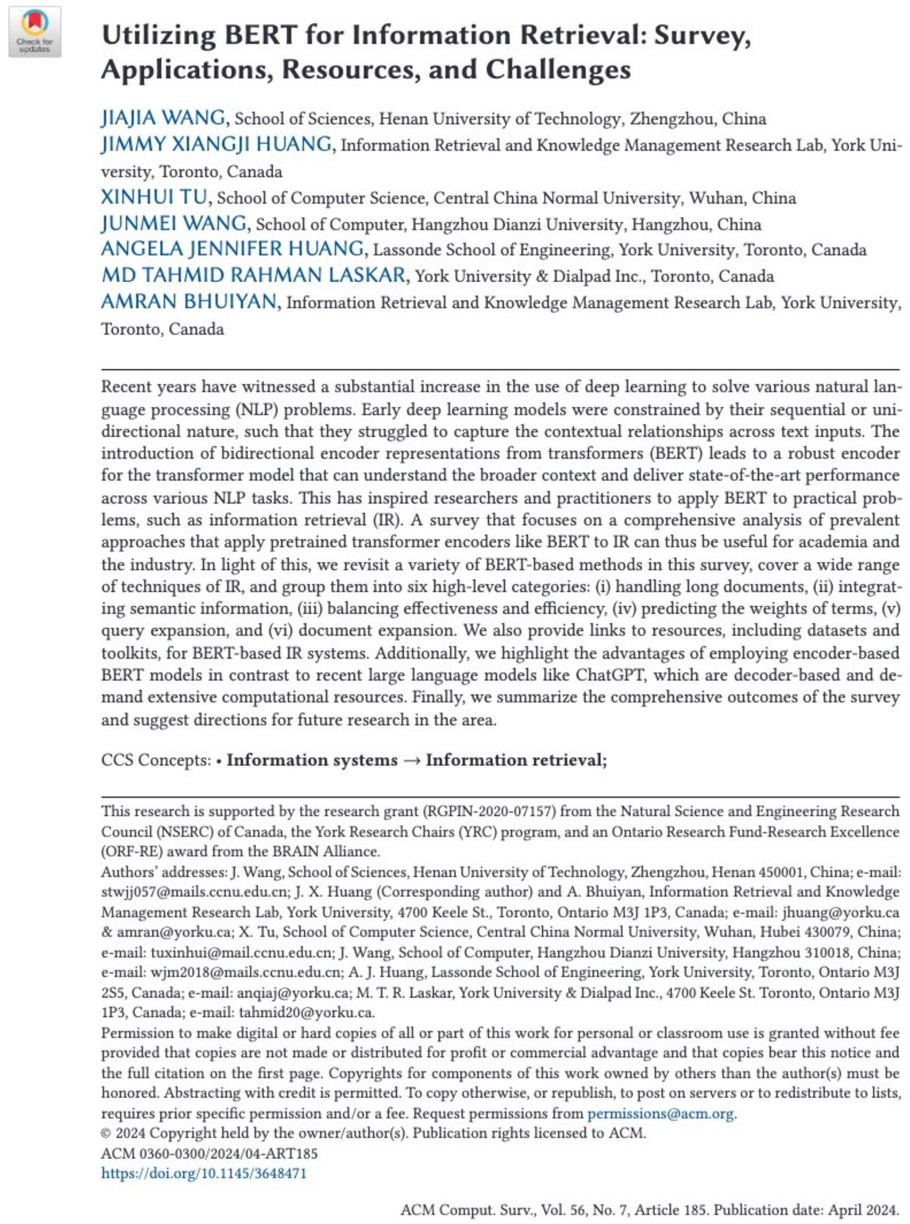
universität koblenz
weiter:denken

5. Pre-trained large language models

Pre-trained large language models



Han et al., Pre-trained models: Past, present and future, *AI Open*, Volume 2, 2021, pages 225-250



Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges

JIAJIA WANG, School of Sciences, Henan University of Technology, Zhengzhou, China
JIMMY XIANGJI HUANG, Information Retrieval and Knowledge Management Research Lab, York University, Toronto, Canada
XINHUI TU, School of Computer Science, Central China Normal University, Wuhan, China
JUNMEI WANG, School of Computer, Hangzhou Dianzi University, Hangzhou, China
ANGELA JENNIFER HUANG, Lassonde School of Engineering, York University, Toronto, Canada
MD TAHMID RAHMAN LASKAR, York University & Dialpad Inc., Toronto, Canada
AMRAN BHUIYAN, Information Retrieval and Knowledge Management Research Lab, York University, Toronto, Canada

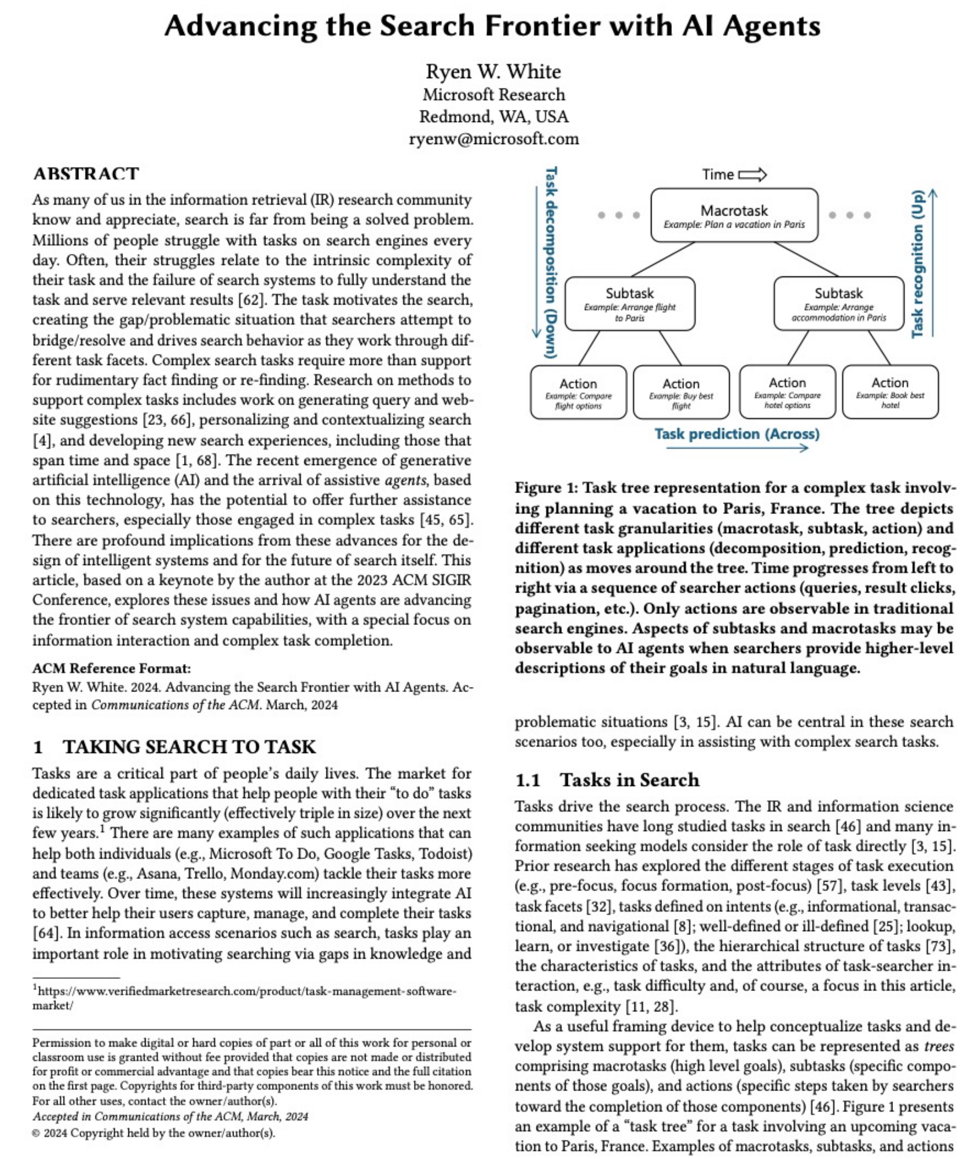
Recent years have witnessed a substantial increase in the use of deep learning to solve various natural language processing (NLP) problems. Early deep learning models were constrained by their sequential or unidirectional nature, such that they struggled to capture the contextual relationships across text inputs. The introduction of bidirectional encoder representations from transformers (BERT) leads to a robust encoder for the transformer model that can understand the broader context and deliver state-of-the-art performance across various NLP tasks. This has inspired researchers and practitioners to apply BERT to practical problems, such as information retrieval (IR). A survey that focuses on a comprehensive analysis of prevalent approaches that apply pretrained transformer encoders like BERT to IR can thus be useful for academia and the industry. In light of this, we revisit a variety of BERT-based methods in this survey, cover a wide range of techniques of IR, and group them into six high-level categories: (i) handling long documents, (ii) integrating semantic information, (iii) balancing effectiveness and efficiency, (iv) predicting the weights of terms, (v) query expansion, and (vi) document expansion. We also provide links to resources, including datasets and toolkits, for BERT-based IR systems. Additionally, we highlight the advantages of employing encoder-based BERT models in contrast to recent large language models like ChatGPT, which are decoder-based and demand extensive computational resources. Finally, we summarize the comprehensive outcomes of the survey and suggest directions for future research in the area.

CCS Concepts: • **Information systems → Information retrieval;**

This research is supported by the research grant (RGPIN-2020-07157) from the Natural Science and Engineering Research Council (NSERC) of Canada, the York Research Chairs (YRC) program, and an Ontario Research Fund-Research Excellence (ORF-RE) award from the BRAIN Alliance.

Authors' addresses: J. Wang, School of Sciences, Henan University of Technology, Zhengzhou, Henan 450001, China; e-mail: stwjj057@mails.ccnu.edu.cn; J. X. Huang (Corresponding author) and A. Bhuiyan, Information Retrieval and Knowledge Management Research Lab, York University, 4700 Keele St., Toronto, Ontario M3J 1P3, Canada; e-mail: jhuang@yorku.ca & amran@yorku.ca; X. Tu, School of Computer Science, Central China Normal University, Wuhan, Hubei 430079, China; e-mail: tuxinhui@mail.ccnu.edu.cn; J. Wang, School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China; e-mail: wjm2018@mails.ccnu.edu.cn; A. J. Huang, Lassonde School of Engineering, York University, Toronto, Ontario M3J 2S5, Canada; e-mail: anqiaj@yorku.ca; M. T. R. Laskar, York University & Dialpad Inc., 4700 Keele St. Toronto, Ontario M3J 1P3, Canada; e-mail: tahmid20@yorku.ca.

ACM 0360-0300/2024/04-ART185
https://doi.org/10.1145/3648471



Advancing the Search Frontier with AI Agents

Ryen W. White
Microsoft Research
Redmond, WA, USA
ryenw@microsoft.com

ABSTRACT

As many of us in the information retrieval (IR) research community know and appreciate, search is far from being a solved problem. Millions of people struggle with tasks on search engines every day. Often, their struggles relate to the intrinsic complexity of their task and the failure of search systems to fully understand the task and serve relevant results [62]. The task motivates the search, creating the gap/problematic situation that searchers attempt to bridge/resolve and drives search behavior as they work through different task facets. Complex search tasks require more than support for rudimentary fact finding or re-finding. Research on methods to support complex tasks includes work on generating query and website suggestions [23, 66], personalizing and contextualizing search [4], and developing new search experiences, including those that span time and space [1, 68]. The recent emergence of generative artificial intelligence (AI) and the arrival of assistive *agents*, based on this technology, has the potential to offer further assistance to searchers, especially those engaged in complex tasks [45, 65]. There are profound implications from these advances for the design of intelligent systems and for the future of search itself. This article, based on a keynote by the author at the 2023 ACM SIGIR Conference, explores these issues and how AI agents are advancing the frontier of search system capabilities, with a special focus on information interaction and complex task completion.

ACM Reference Format:
Ryen W. White. 2024. Advancing the Search Frontier with AI Agents. Accepted in *Communications of the ACM*. March, 2024

1 TAKING SEARCH TO TASK

Tasks are a critical part of people's daily lives. The market for dedicated task applications that help people with their "to do" tasks is likely to grow significantly (effectively triple in size) over the next few years.[1] There are many examples of such applications that can help both individuals (e.g., Microsoft To Do, Google Tasks, Todoist) and teams (e.g., Asana, Trello, Monday.com) tackle their tasks more effectively. Over time, these systems will increasingly integrate AI to better help their users capture, manage, and complete their tasks [64]. In information access scenarios such as search, tasks play an important role in motivating searching via gaps in knowledge and problematic situations [3, 15]. AI can be central in these search scenarios too, especially in assisting with complex search tasks.

[1]https://www.verifiedmarketresearch.com/product/task-management-software-market/

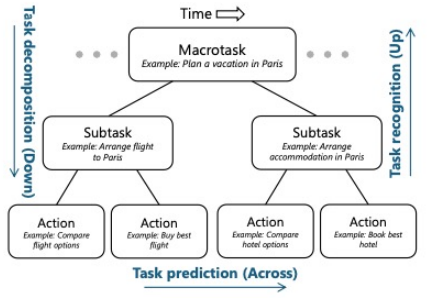

Time
Task decomposition (Down)
Macrotask
Example: Plan a vacation in Paris
Task recognition (Up)
Subtask
Example: Arrange flight to Paris
Subtask
Example: Arrange accommodation in Paris
Action
Example: Compare flight options
Action
Example: Buy best flight
Action
Example: Compare hotel options
Action
Example: Book best hotel
Task prediction (Across)

Figure 1: Task tree representation for a complex task involving planning a vacation to Paris, France. The tree depicts different task granularities (macrotask, subtask, action) and different task applications (decomposition, prediction, recognition) as moves around the tree. Time progresses from left to right via a sequence of searcher actions (queries, result clicks, pagination, etc.). Only actions are observable in traditional search engines. Aspects of subtasks and macrotasks may be observable to AI agents when searchers provide higher-level descriptions of their goals in natural language.

1.1 Tasks in Search

Tasks drive the search process. The IR and information science communities have long studied tasks in search [46] and many information seeking models consider the role of task directly [3, 15]. Prior research has explored the different stages of task execution (e.g., pre-focus, focus formation, post-focus) [57], task levels [43], task facets [32], tasks defined on intents (e.g., informational, transactional, and navigational [8]; well-defined or ill-defined [25]; lookup, learn, or investigate [36]), the hierarchical structure of tasks [73], the characteristics of tasks, and the attributes of task-searcher interaction, e.g., task difficulty and, of course, a focus in this article, task complexity [11, 28].

As a useful framing device to help conceptualize tasks and develop system support for them, tasks can be represented as *trees* comprising macrotasks (high level goals), subtasks (specific components of those goals), and actions (specific steps taken by searchers toward the completion of those components) [46]. Figure 1 presents an example of a "task tree" for a task involving an upcoming vacation to Paris, France. Examples of macrotasks, subtasks, and actions

universität koblenz
weiter:denken

6. Summary

Intended Learning Outcomes

You are now able to:

- Know what a language model is
- Understand differences between different language models (unigram, bigram)
- Understand how to use language modeling for information retrieval
- Learn about different smoothing schemes for LM for IR
- Be able to compare LM for IR with the vector space model