AI Assisted Web Deep Search

CS-480 Capstone, Spring 2024

Can AI Cite its Sources?

Problem Space

- Centralized Search Engines have SEO incentives that often reduce search quality
- Generative Pre-trained AI obscures training data; veracity of statements can't be verified
- Can we use smaller scale search and AI tools to assist in crawling the web for specific topics?

Abstract

We're exploring basic tools for tunable, topic-specific web search, including rudimentary language-modeling (GenerativeAI) to assist in refining crawler parameters and indexes.

Our goal is to manage small-data, topic-specific, deep-search of the web.

We've set up a basic search engine; we train Bayesian networks on documents we discover with that engine, and then use a 2-gram model to generate Markov chains. Text generation produces a weighted list of contributing source documents.

Authors

Joe Burchett
B.S. CS, Fall 2024

Zac Heimgartner
B.S. CS, Fall 2024

Carrie Nickel
B.S. CS, B.S., Mathematics, Spring 2024

Marcus Pappas
B.S. Cybersecurity Management, Spring 2024

Keelyn Pilcher
B.S. Cybersecurity Management, Spring 2024

slidesmania.com

A Recipe for a Bayesian Network

Ingredients:

1 Input text cheese is delicious, I love cheese and fruit
1 Matrix Math library we like numpy

Makes one Bayesian Network appropriate for generating Markov Chains

0	cheese
1	is
2	delicious
3	I
4	love
5	and
6	fruit

Step 1 — Create token Dictionary
Each token in the input text gets a unique index

Step 2 — Vectorize Input
This vector is an ordered series of token indexes from the dictionary in step 1
[0, 1, 2, 3, 4, 0, 5, 6]

Step 3 — Count Transitions
Each 2-tuple (e.g., 0->1) in the input vector represents column & row in matrix C

C =

	0	1	2	3	4	5	6
0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
5	1	0	0	0	0	0	0
6	0	0	0	0	0	1	0

Step 4 — Make Column Sums Vector
Take the sum of each column in matrix C to produce a column-counts vector
[2,1,1,1,1,1,0]

Step 5 — Form Diagonal Matrix D
Take `1/sum` for each element in the column-counts vector as matrix diagonal

D =

	0	1	2	3	4	5	6
0	0.5	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0
6	0	0	0	0	0	0	1

1 / column sums

Step 6 — Calculate Transition Matrix M
Take product **DC** to form matrix **M**, to represent probabilities that one token follows another
M=DC

M =

	0	1	2	3	4	5	6
0	0	0	0	0	1	0	0
1	0.5	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
5	0.5	0	0	0	0	1	0
6	0	0	0	0	0	0	0

Citing Our Sources

Example Datastructure

```
In [7]: source_struct
Out[7]:
{(0, 1): {'http://source1': 1, 'http://source2': 1, 'http://source3': 2},
 (1, 0): {'http://source1': 2, 'http://source3': 3, 'http://source4': 5},
 (1, 2): {'http://source1': 3, 'http://source4': 1},
 (2, 4): {'http://source3': 2, 'http://source4': 1, 'http://source5': 6}}
```

Building it:

- Inputs are tokenized; we record a source for each ordered pair of tokens (a '2-Gram')
- Every 2-gram represents a column/row in our transition matrix. The 2-gram is represented as a tuple, and is used as a key in a sources dict
- When text is ingested, we add a dict of the form `{source : count}` to the source_struct
- For each 2-gram in the source text, we increment the source count during ingestion

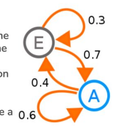
Using it:

- When we generate markov chains, the index-pair for each transition is converted to a tuple
- Each tuple is looked up in the `source_struct`, and we add the contribution count for that source to an accumulator (E.G., if source5 occurred 6 times for transition (2,4), 6 gets added to the total count for that source
- We select the top sources and return these as primary contributors to the generated text
- NB: This process is memory hungry! Our model focuses on 'small data' to mitigate memory costs

Generating Text

Markov Chains

- Markov chains are generated from the Transition Matrix: the current index ID of the current token is the index of a column in the transition matrix
- The column represents a vector of transition probabilities, with row number being the index of the 'next' token
- We select 'next'; the indexes (e.g, 2->1) are a tuple in the source_struct
- Markov chains are generated by repeatedly finding 'next' tokens from the current token



Cosine Similarity

- Users choose a 'search target'
- We generate a markov chain as described above
- We generate vectors in 'n' dimensions – n is the number of unique tokens across both the 'search target' and the markov chain
- Indexes in the vector represent a unique token; values at each index are the count of that token for each of our two texts
- We compute the cosine of the angle between those vectors, to get a value between 0 and 1
- Higher numbers are 'more similar' !

$$\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Search Engine Basics

Crawling
Apache Nutch
Scrapes seed URLs, generates new links from scaped pages, fetches generated links

Indexing
Apache SOLR
Nutch stores fetched documents and metadata in this searchable storage backend

Ranking
Apache SOLR
The page rank algorithm determines top-results from a query. We're using SOLR defaults

Presenting
Apache SOLR
SOLR's Lucene Search interface presents a query interface that our software uses to find and fetch documents

Deployed Components

Azure Infra
- **DNS** — Hosted Zone (A record, CNAME)
- **Network Security Group** — Basic Firewall
- **Ubuntu VM** — Hosts assorted server software

3rd Party Middleware
- **Nutch** — Crawler
- **Certbot** — Manages LetsEncrypt SSL Cert
- **SOLR** — Index, Lucene Interface
- **NGNIX** — Webserver

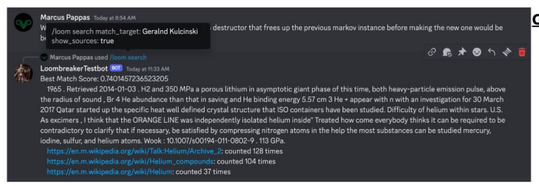
Custom App Code
- **AI core** — Bayes Network, markov process, text pipeline
- **Flask** — REST interface
- **Ionic** — Web Interface
- **Loombot** — Discord Bot Interface

Crawl / Retrain Loop

- **CRAWL** — Crawl Script – Runs Nutch crawler phases, starting with seeds
- **TRAIN** — Query SOLR, use results to build Bayesian Network
- **GENERATE** — Markov process generates text from documents, cites source pages

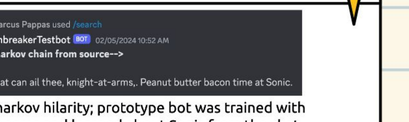
User sets new seeds, triggers crawl again

Examples, Demo & Source



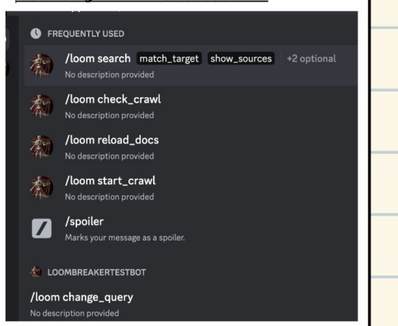
Generated result:

- **Target markov phrase** (for cos similarity): "Geralnd Kulcinski"
- **SOLR Query** (to build network): "Argon"
- **Seeds** (to crawl web): wikipedia pages



Early markov hilarity; prototype bot was trained with Keats poems and learned about Sonic from the chat

Search Engine Control & bot actions



Query parameters to request generated text

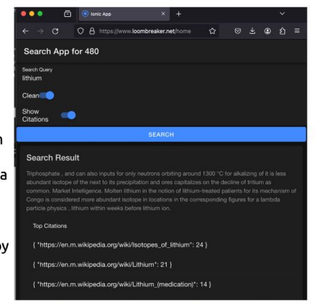
- Target score is 'ideal' cosine similarity
- Max_rounds is number of attempts made to exceed target score

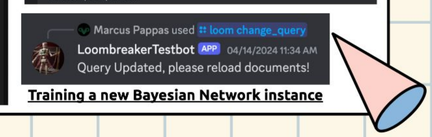
https://www.loombreaker.net

Github Repository

Webapp (Ionic)

- Queries a REST endpoint
- Returns markov generated text with citations
- Website is live – but currently using a rest mock instead of live API!
- Search for 'cats' or 'lithium' to see results
- 'lithium' results are real (produced by our software), but currently static (returned by a mock REST api)





Training a new Bayesian Network instance

slidesmania.com

Text Processing Pipeline

Text Cleaning
Regular Expressions remove certain patterns (like text citations)

Bayesian Network
Tokenize text into n-grams, build transition matrix

Markov Chains
Do weighted random walk over Bayesian network; collect tokens

Cosine Similarity
Generate Markov chains and compare to target string until a close-ish match is found

Learnings / Future Work

It works
We can train a bayesian network on web content, and generate text with citations. We can adjust crawlers and train a search engine.

Progress
Making this useful for real-world web search will require much more work – this is at best a proof-of-concept

Limits
We'll never be 'fast' like Pre-trained generative chatbots – but being retrainable means we can adapt to user needs

A Niche
Topic specific, deep search of the web may be useful for researchers and for working around ad-supported search

Future
We've laid a foundation for future work, using more advanced AI concepts (E.G., Transformers, RAGs) and for granular crawler retraining

slidesmania.com