Nontraditional Search Engines

For Fun and (not much) Profit

Introduction

Hi, I'm Casey Stella!

Traditional Search Engines

Traditional Approach

"In 1911, Professor Lane Cooper published a concordance of William Wordsworth's poetry so that scholars could readily locate words in which they were interested. The 1,136-page tome lists all 211,000 nontrivial words in the poet's works, from Aaliza to Zutphen's, yet remarkably, it took less than 7 months to construct. The task was completed so quickly because it was undertaken by a highly organized team of 67 people – 3 of whom had died by the time the concordance was published – using 3-by-5 inch cards, scissors, glue, and stamps."

— *Managing Gigabytes: Compressing and Indexing Documents and Images*

Traditional Approach

Document Search Engines are really just (sometimes fuzzy) information retrieval systems involving

- Compressed Document Storage – Storing the documents in a space-efficient way

Traditional Approach

Document Search Engines are really just (sometimes fuzzy) information retrieval systems involving

- Compressed Document Storage – Storing the documents in a space-efficient way
- Indexing – Intelligently organizing information so that queries can be resolved efficiently and relevant portions of the data extraced quickly.

Traditional Approach

Document Search Engines are really just (sometimes fuzzy) information retrieval systems involving

- Compressed Document Storage – Storing the documents in a space-efficient way
- Indexing – Intelligently organizing information so that queries can be resolved efficiently and relevant portions of the data extraced quickly.
- Query Scoring – Returning relevant needles from the haystack of documents

Query Scoring: In the beginning, there was counting

In order to rank queries, one important aspect is relevance to the query. Simple ranking systems were constructed around

- Term Frequency - The number of times a word or query term exists in a document.
- Document Frequency - The number of documents which contain a query term

Together, they can be used to form a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

Query Scoring: Vectors

Scoring suffered from search engine result inaccuracy due to the vagaries of the human languages. Synonyms and context which computers lack can make the exercise maddening. What we needed was a system which could give us more fuzzy matches efficiently.

- In 2013, Word2Vec was a vectorization model created by Google
- In 2018, both Google and Facebook released approaches to embed not just words, but also sentences into vectors.

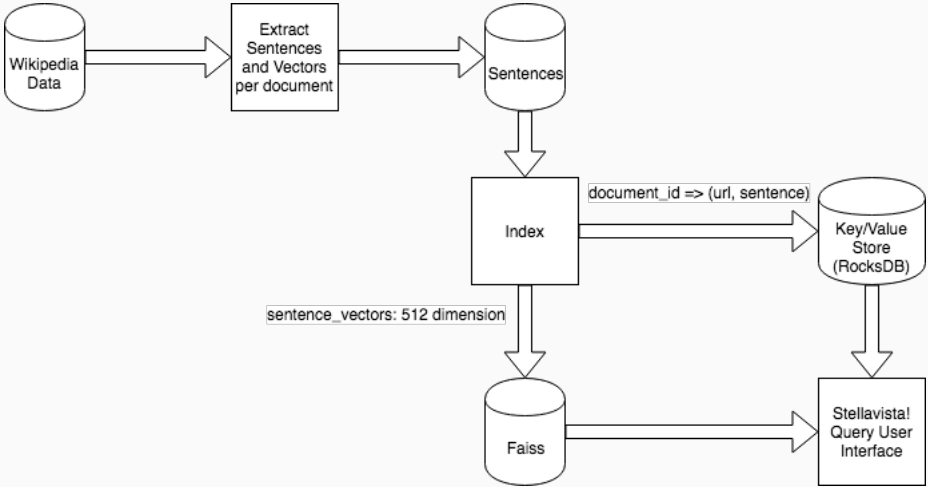
Takeaway: With sentences and words embedded into a vector space, query scoring becomes a nearest neighbors search in a vector space

Stellavista!

A toy search engine, which

- Index 2.1M sentences from wikipedia using the released Google Universal Sentence Encoder in Tensorflow
- Retrieve documents which sentences which are similar to textual queries in ranked order based on textual similarity

Stellavista! Architecture



Challenges

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors
  - Applying a model is embarassingly parallel

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors
  - Applying a model is embarassingly parallel
- The Bad

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors
  - Applying a model is embarassingly parallel
- The Bad
  - Unlike summing or averaging word vectors, it was a lot slower.

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors
  - Applying a model is embarassingly parallel
- The Bad
  - Unlike summing or averaging word vectors, it was a lot slower.
  - Google released a model, not the algorithm, so I can't retrain and add new vocabulary.

Sentence Vectorization via Tensorflow

A host of lessons were learned:

- The Good
  - It was available and (probably) better than just summing or averaging word vectors
  - Applying a model is embarassingly parallel
- The Bad
  - Unlike summing or averaging word vectors, it was a lot slower.
  - Google released a model, not the algorithm, so I can't retrain and add new vocabulary.
  - There was a massive memory leak which caused me NOT to be able to use Spark for this.

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node
  - GPU accelerated where appropriate

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node
  - GPU accelerated where appropriate
  - Supports approximate solutions with latency tradeoffs

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node
  - GPU accelerated where appropriate
  - Supports approximate solutions with latency tradeoffs
- The Bad

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node
  - GPU accelerated where appropriate
  - Supports approximate solutions with latency tradeoffs
- The Bad
  - Not multi-node parallelizable, but parallelizable via sharding.

Finding k-Nearest Neighbors in 2M 512-dimensional Vectors Quickly

This was a really hard problem, frankly. After looking around for a comprehensive solution, I landed on a Facebook library called Faiss

- The Good
  - Blazing fast
  - Scales to billions of vectors on a single node
  - GPU accelerated where appropriate
  - Supports approximate solutions with latency tradeoffs
- The Bad
  - Not multi-node parallelizable, but parallelizable via sharding.
  - Not online updateable.

Demo

Questions

Thanks for your attention! Questions?

- Code & scripts for this talk available on my github presentation page.[1]
- Find me at http://caseystella.com
- Twitter handle: @casey_stella

[1] http://github.com/cestella/presentations/