Efficient Vector Space Retrieval

Information Retrieval Project

Marlena Kuc, Elina Tugaeva, Eliana Ruslanova
University of Mannheim

FSS 2020

Outline

1. Dataset
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Outline

1. Dataset
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Dataset

We used development subset of NFCorpus.

- 3193 documents
- 325 queries (full text) with corresponding 325 query titles
- Relevance links between documents and queries

Outline

1. Dataset
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents → punctuation removal

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents → punctuation removal → number removal

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents → punctuation removal → number removal → lower case

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents → punctuation removal → number removal → lower case → stopwords removal

Preprocessing

Documents came in the preprocessed form, though we still needed to stem words.

Queries and query titles needed preprocessing.

removing terms not occuring in documents → punctuation removal → number removal → lower case → stopwords removal → stemming

Outline

1. Structure and size of the data set
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Basic retrieval

Term frequency:

$$tf(t_i, d_j) = 1 + \log(ft_d)$$

Inverse document frequency:

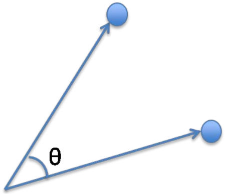
$$idf(t_i) = \log(\frac{N}{df_i})$$

TF-IDF:

$$idf(t_i) \cdot tf(t_i, d_j)$$

Cosine similarity:

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



Basic retrieval

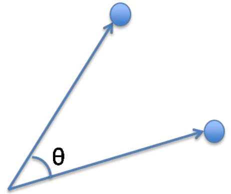
Term frequency: { 0: { alkylphenol: 0.008, human: 0.026, milk: 0.026, .. }

Inverse document frequency: { alkylphenol: 6.122 }

TF-IDF: { 0: { alkylphenol: 0.052, human: 0.035, milk: 0.080, .. }

Cosine similarity:

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



Basic retrieval

Shortcomings:

- for each query we have to compute all cosine scores
- we do not make use of sparsity in term space

For large corporas the retrieval cannot be done in a feasible time.

Basic retrieval

Shortcomings:

- for each query we have to compute all cosine scores
- we do not make use of sparsity in term space

For large corporas the retrieval cannot be done in a feasible time.

Idea: Reduce the number of cosine computations

Outline

1. Structure and size of the data set
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Tiered index

Inverted index with tf scores:

{**dtt:** {135: **0.017**, 136: **0.007**, 1117: **0.006**, 1118: **0.014**, 1264: **0.008**, 1444: **0.006**, 2327: **0.018**, 3020: **0.004**} }

Tiered index

Inverted index with tf scores:

{**dtt:** {135: **0.017**, 136: **0.007**, 1117: **0.006**, 1118: **0.014**, 1264: **0.008**, 1444: **0.006**, 2327: **0.018**, 3020: **0.004**} }

Sorted inverted index by tf scores:

{**dtt:** {2327: **0.018**, 135: **0.017**, 1118: **0.014**, 1264: **0.008**, 136: **0.007**, 1444: **0.006**, 1117: **0.006**, 3020: **0.004**} }

Tiered index

Inverted index with tf scores:

{**dtt:** {135: **0.017**, 136: **0.007**, 1117: **0.006**, 1118: **0.014**, 1264: **0.008**, 1444: **0.006**, 2327: **0.018**, 3020: **0.004**} }

Sorted inverted index by tf scores:

{**dtt:** {2327: **0.018**, 135: **0.017**, 1118: **0.014**, 1264: **0.008**, 136: **0.007**, 1444: **0.006**, 1117: **0.006**, 3020: **0.004**} }

Chunked inverted index into K = 4 tieres:

{**dtt:** {0: [2327, 135], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

Tiered index

Inverted index with tf scores:

{**dtt:** {135: **0.017**, 136: **0.007**, 1117: **0.006**, 1118: **0.014**, 1264: **0.008**, 1444: **0.006**, 2327: **0.018**, 3020: **0.004**} }

Sorted inverted index by tf scores:

{**dtt:** {2327: **0.018**, 135: **0.017**, 1118: **0.014**, 1264: **0.008**, 136: **0.007**, 1444: **0.006**, 1117: **0.006**, 3020: **0.004**} }

Chunked inverted index into K = 4 tieres:

{**dtt:** {0: [2327, 135], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

How to effectively merge tieres from different postings?

Tiered index

Inverted index with tf scores:

{**dtt:** {135: **0.017**, 136: **0.007**, 1117: **0.006**, 1118: **0.014**, 1264: **0.008**, 1444: **0.006**, 2327: **0.018**, 3020: **0.004**} }

Sorted inverted index by tf scores:

{**dtt:** {2327: **0.018**, 135: **0.017**, 1118: **0.014**, 1264: **0.008**, 136: **0.007**, 1444: **0.006**, 1117: **0.006**, 3020: **0.004**} }

Chunked inverted index into K = 4 tieres:

{**dtt:** {0: [**2327, 135**], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

Sorted tieres:

{**dtt:** {0: [**135, 2327**], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

Tiered index

{**dtt:** {0: [135, 2327], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

{**human:** {0: [23, 24, 135, 210, 2327], 1: [13, 28, 111, 1118, 2000], 2: [139, 789, 830, 921, 1000], 3: [1, 3, 12, 201, 1190] } }

Tiered index

{**dtt:** {0: [135, 2327], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

{**human:** {0: [23, 24, 135, 210, 2327], 1: [13, 28, 111, 1118, 2000], 2: [139, 789, 830, 921, 1000], 3: [1, 3, 12, 201, 1190] } }

Retrieving at least top ***3*** documents = finding at least ***3*** intersections

Tiered index

{**dtt:** {0: [135, 2327], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

{**human:** {0: [23, 24, 135, 210, 2327], 1: [13, 28, 111, 1118, 2000], 2: [139, 789, 830, 921, 1000], 3: [1, 3, 12, 201, 1190] } }

Retrieving at least top ***3*** documents = finding at least ***3*** intersections

1. Merging [135, 2327] and [23, 24, 135, 210, 2327] → [135, 2327] $< K$

Tiered index

{**dtt:** {0: [135, 2327], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

{**human:** {0: [23, 24, 135, 210, 2327], 1: [13, 28, 111, 1118, 2000], 2: [139, 789, 830, 921, 1000], 3: [1, 3, 12, 201, 1190] } }

Retrieving at least top ***3*** documents = finding at least ***3*** intersections

1. Merging [135, 2327] and [23, 24, 135, 210, 2327] → [135, 2327] $< \boldsymbol{K}$
2. Expanding tieres: [135, 2327, 1118, 1264], **[**23, 24, 135, 210, 232 , 13, 28, 111, 1118, 2000]

Tiered index

{**dtt:** {0: [135, 2327], 1: [1118, 1264], 2: [136, 1444], 3: [1117, 3020] } }

{**human:** {0: [23, 24, 135, 210, 2327], 1: [13, 28, 111, 1118, 2000], 2: [139, 789, 830, 921, 1000], 3: [1, 3, 12, 201, 1190] } }

Retrieving at least top ***3*** documents = finding at least ***3*** intersections

1. Merging [135, 2327] and [23, 24, 135, 210, 2327] → [135, 2327] < ***K***
2. Expanding tieres: [135, 2327, 1118, 1264], [23, 24, 135, 210, 232 , 13, 28, 111, 1118, 2000]
3. Sorting tieres: [135, 1118, 2327, 1264], [13 23, 24, 28, 111, 135, 210, 232 , 1118, 2000] → [135, 1118, 2327] >= ***K***

Tiered index

In this algorithm we perform intersection between tieres from term postings. What if no intersection is found between all query terms?

Tiered index

In this algorithm we perform intersection between tieres from term postings. What if no intersection is found between all query terms?

why deep fried foods may cause cancer in the latest study on dietary patterns and breast cancer risk among women , healthier eating was associated with eliminating three-quarters of the odds of breast cancer , whereas less healthy eating was associated with up to nearly eight times the odds . included in the unhealthy eating pattern was the consumption of deep-fried foods , which have previously been linked to breast cancer , pancreatic cancer , lung cancer , oral and throat cancers , esophageal cancer , and cancer of the voicebox . no deep fried foods ? what ' s a southern belle to do ? instead of deep fried foods , how about the traditional southern diet , characterized by high intakes of cooked greens , beans , legumes , cabbage , sweet potatoes and cornbread , which may reduce the risk of invasive breast cancer significantly . what about the consumption of deep-fried foods and risk of prostate cancer ? researchers at the fred hutchinson cancer research center and the university of washington found that eating french fries , fried chicken , fried fish , and doughnuts was associated with about a third greater odds of prostate cancer . after stratifying for tumor aggressiveness , they found slightly stronger associations with more aggressive disease , suggesting that regular intake of deep-fried foods may contribute to the progression of prostate cancer as well . what in deep fried foods is so bad for us ? just heating oil that hot can generate potentially carcinogenic compounds , and then known carcinogens such as heterocyclic amines and polycyclic aro .....

Tiered index

In this algorithm we perform intersection between tieres from term postings. What if no intersection is found between all query terms?

All cosine similarities need to be computed.

Tiered index

In this algorithm we perform intersection between tieres from term postings. What if no intersection is found between all query terms?

All cosine similarities need to be computed.

Conclusion: Tiered index in the form we implemented makes sense only for short queries - justification for using query titles in evaluation of model.

Outline

1. Structure and size of the data set
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Pre-clustering

Text Clustering is a grouping a set of documents into classes of similar documents

Pre-clustering

Text Clustering is a grouping a set of documents into classes of similar documents

The Cluster Hypothesis: Documents if the same cluster behave similarly with respect to relevance to information need

Pre-clustering

Text Clustering is a grouping a set of documents into classes of similar documents

The Cluster Hypothesis: Documents if the same cluster behave similarly with respect to relevance to information need

Use to improve retrieval speed

Cluster pruning: consider only documents in a small number of clusters as candidates for which we compute similarity scores

Pre-clustering

Offline preparation:

1. Pick $\sqrt{N}$ documents from a collection to be leaders of future clusters
2. Assign each document that is not a leader (call it follower) to the cluster with nearest leader

Pre-clustering

Offline preparation:

1. Pick √N documents from a collection to be leaders of future clusters
2. Assign each document that is not a leader (call it follower) to the cluster with nearest leader

Online process ‘on-the-fly’:

1. Compute the similarities of the query with all leaders
2. Choose the most similar documents from the cluster with closest leader

Pre-clustering

Cosine similarity

measures the cosine of the angle between two documents

Pre-clustering

Cosine similarity

measures the cosine of the angle between two documents

FAISS

assures faster similarity searches

focuses on methods that compress the original vectors

Pre-clustering

Cosine similarity

measures the cosine of the angle between two documents

FAISS

assures faster similarity searches

focuses on methods that compress the original vectors

K-means

is arguably the most common technique

non-parametric, distance-based clustering method

Outline

1. Structure and size of the data set
2. Preprocessing
3. Basic retrieval: TF-IDF & cosine similarity
4. Tiered index
5. Pre-clustering
6. Random projections
7. Evaluation of performance and results

Random Projections

Johnson-Lindenstrauss lemma:

A set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved.

Johnson, W. B., Lindenstrauss, J., & Schechtman, G. (1986).

Random Projections

Johnson-Lindenstrauss lemma:

A set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved.

Idea:

Reduce the number of cosine computations by dimensionality reduction.

Johnson, W. B., Lindenstrauss, J., & Schechtman, G. (1986).

Random Projections

Given a matrix of document vectors - $X_{d \times N}$

(N – number of documents, d – the length of document vector)

Random Projections

Given a matrix of document vectors - $\boldsymbol{X_{d \times N}}$

(N – number of documents, d – the length of document vector)

1. Create random matrix - $\boldsymbol{R_{m \times d}}$

Random Projections

Given a matrix of document vectors - $\boldsymbol{X_{d \times N}}$

(N – number of documents, d – the length of document vector)

1. Create random matrix - $\boldsymbol{R_{m \times d}}$

2. Compute the projection of the documents onto a new ***m***-dimensional subspace

$$\boldsymbol{X^{RP}_{m \times N} = R_{m \times d} X_{d \times N}}$$

Random Projections

Given a matrix of document vectors - $\boldsymbol{X_{d \times N}}$

(N – number of documents, d – the length of document vector)

1. Create random matrix - $\boldsymbol{R_{m \times d}}$

2. Compute the projection of the documents onto a new ***m***-dimensional subspace

$$\boldsymbol{X^{RP}_{m \times N} = R_{m \times d} X_{d \times N}}$$

3. Apply a hash function to each element of the resulting matrix, creating a new hashed matrix $\boldsymbol{H^{RP}_{m \times N}}$, where

$$\boldsymbol{h^{RP}_{ij}} = \begin{cases} \boldsymbol{1} & \boldsymbol{if\ x^{RP}_{ij} \geq \theta} \\ \boldsymbol{0} & \boldsymbol{otherwise} \end{cases}$$

Random Projections

What randomization function should we choose?

Random Projections

What randomization function should we choose?

- Real positive numbers $\mathbb{R} \in [0,1]$
  => **Uncertain threshold selection**

Random Projections

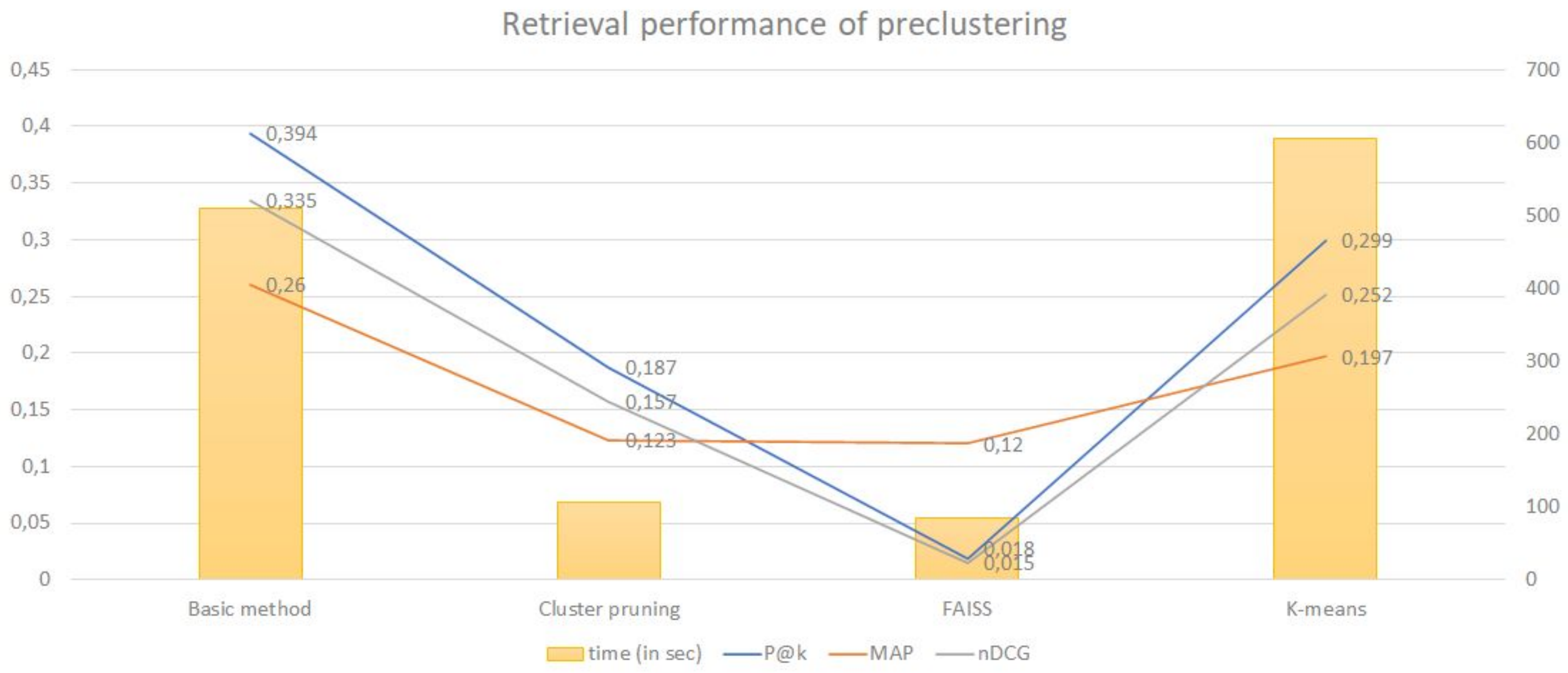
What randomization function should we choose?

- Real positive numbers $\mathbb{R} \in$ **[0,1]**
  => Uncertain threshold selection

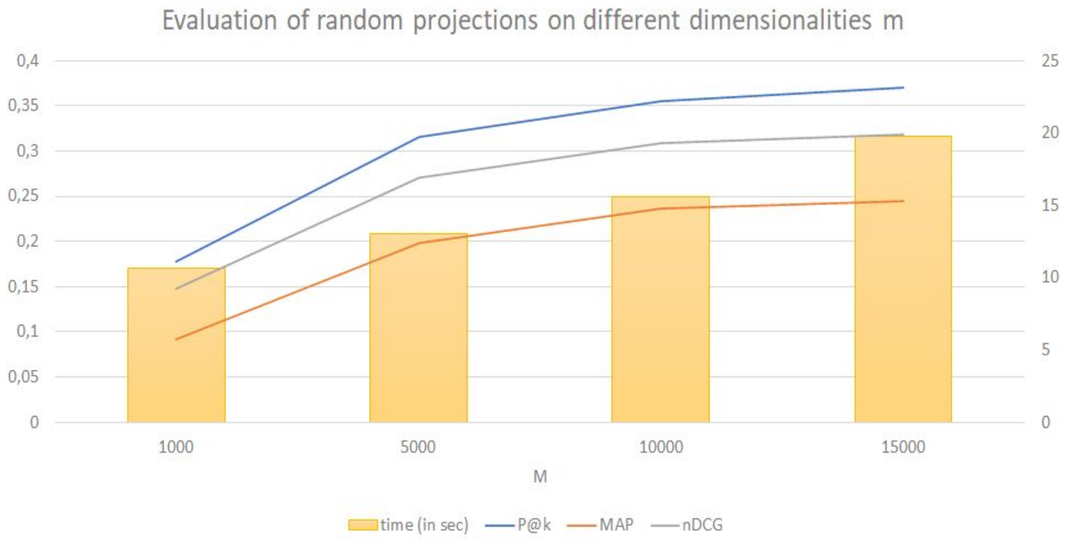
- **Gaussian distribution with mean 0 and variance 1:** $\boldsymbol{X \sim \mathcal{N}(0,1)}$
  => **Threshold = 0**

Outline

Pre-clustering



Random Projections



Basic retrieve

P@k	MAP	nDCG	Time (in sec)
0.394	0.260	0.336	495

Tiered index

Baseline: basic retrieval for query titles

Tiered index

Baseline: basic retrieval for query titles

K = 4 number of tieres

Accuracy:

	P@k	MAP	nDCG
Basic retrieval	0.328	0.238	0.286
Tiered index	0.165	0.11	0.141

Tiered index

15m 59se

Baseline: basic retrieval for query titles

K = 4 number of tieres

Accuracy:

	P@k	MAP	nDCG
Basic retrieval	0.328	0.238	0.286
Tiered index	0.165	0.11	0.141

Time:

10 min 3 sec
16min 1s

Tiered index

15m 59se

Baseline: basic retrieval for query titles

K = 4 number of tieres

Accuracy:

	P@k	MAP	nDCG
Basic retrieval	0.328	0.238	0.286
Tiered index	0.165	0.11	0.141

Time:

10 min 3 sec
16min 1s

Still better, since vectorization of queries is performed outside of basic retrieval model, while tiered index performs is on the fly

Thank you for attention!