Latent Semantic Analysis

20165953 주광우

Problem with VSM

We have already tired to use VSM to do the document retrieval. It works well. We treat documents and queries as vectors, compute the score by cosine similarity, thus we can rank the retrieved results.

But there are some problems with VSM. One is the term–document matrix's size may be huge. That means both the compute complexity and space complexity will increase. Another problem is, in VSM, we treat terms independently, so it can not cope with two problems arising in natural languages: *synonymy* and *polysemy*.

What Is Latent Semantic Analysis

latent semantic analysis can solve these two problems in VSM. In LSA, we use the *singular value decomposition* to construct a low–rank approximation to the term–document matrix. Each row/column be mapped into the low–rank dimensional space. With the SVD on original term–document matrix, we are not just reduced the size of matrix, but also find the relationships between the terms and topics contained in unstructured texts. The idea is the terms that are closed in meaning will occur in similar pieces of text. The new low–dimensional space reflects semantic association.

SVD decomposes a matrix into three. Say have a term–document matrix A, then by SVD we can get $A = U\Sigma V^T$. Matrix U is the orthogonal eigenvectors of AA^T, and V is the orthogonal eigenvectors of A^TA. Σ contains the *singular values*. U is known as the *SVD term matrix*. The V^T is known as the *SVD document matrix*.

Usually, the singular values in Σ are in descending order. If we select the k largest singular values, and their corresponding singular vectors from U and V, we can get the rank k approximation to A with the smallest error. Since we only choose the k largest singular values, the other elements in Σ will be 0. After the production, both U and V^T will be truncated. We call the truncated matrices U_k and V_k^T.

When we want to find the similarity between documents. First, map query into the low-dimensional space by $q_k = q^T U_k \Sigma_k^{-1}$. Then, calculate the cosine similarities with V_k^T. We can treat the k as number of topics. Terms and documents are clustered into k categories.

To summarize, there are roughly a few steps to using SLA:

1. Build term-document matrix
2. Do SVD on term-document matrix
3. Choose a appropriate k-dimensional space
4. Mapping query into that k-dimensional space
5. Compute similarity.

Testing

Use different k

When $k = 2$

Search "theodore", the word only occurs in movie "Her". The result is not good, "Her" is not the 1st document, and scores between documents are similar.

k=2	
rank	score
logan.txt	0.00011
beauty-and-the-beast.txt	0.00011
war-of-the-planet-of-the-apes.txt	0.00011
her.txt	0.00011
happy-feet.txt	0.00011
dark-knight-rises.txt	0.00011
coco.txt	0.00011
big.txt	0.000109
12-monkeys.txt	0.00008

Search "coco". This time, movie "Coco" is the 1st document in the result, but the scores between each document still similar.

coco.txt	0.000169
dark-knight-rises.txt	0.000169
beauty-and-the-beast.txt	0.000169
logan.txt	0.000169
war-of-the-planet-of-the-apes.txt	0.000167
her.txt	0.000167
happy-feet.txt	0.000167
big.txt	0.000163
12-monkeys.txt	0.000138
lord-of-war.txt	0.000055

When $k = 5$

Search "theodore". This time "Her" is the first rank document, and we can see the gap between scores became bigger.

k=5	
rank	score
her.txt	0.000126
happy-feet.txt	0.000125
war-of-the-planet-of-the-apes.txt	0.000124
big.txt	0.000115
beauty-and-the-beast.txt	0.000107
coco.txt	0.00008
logan.txt	0.000059
12-monkeys.txt	0.00004
dark-knight-rises.txt	0.00003
lord-of-war.tx	0.000029

Search "theodore coco". They are two non-common terms. Movie "Coco" is the rank one document, but "Her" is not on the high ranking.

rank	score
coco.txt	0.001386
beauty-and-the-beast.txt	0.001247
happy-feet.txt	0.001021
war-of-the-planet-of-the-apes.txt	0.000888
her.txt	0.000628
big.txt	0.000391
12-monkeys.txt	-0.000039
lord-of-war.txt	-0.000045
dark-knight-rises.txt	-0.000188
logan.txt	-0.000247

When $k = 10$

Search "theodore coco" again. This time, both "Coco" and "Her" are at the high ranking, and only these two documents got positive scores.

k=10	
rank	score
her.txt	0.004865
coco.txt	0.001543
lord-of-war.txt	-0.000155
dark-knight-rises.txt	-0.000159
war-of-the-planet-of-the-apes.txt	-0.000169
beauty-and-the-beast.txt	-0.000181
logan.txt	-0.00019
happy-feet.txt	-0.000191
12-monkeys.txt	-0.000195
big.txt	-0.00039

We can find, as k increase, the search results get better. Like what we said before, we can treat k as the number of topics. Since these 10 movies that I have chosen are in different genres, when k equals to the number of movies, we get the best result.

Use different queries

We've already known LSI can help us find the semantic relations between terms, so here we focus on multi–term queries.

Descriptive String

Search "man who sales weapon". The expected answer is "Lord of war". We can see, the result is good. The answer that we want is on the top, and the scores between the answer and other documents have big gap.

man who sales weapon	
rank	score
lord-of-war.txt	0.000428
dark-knight-rises.txt	0.000004
war-of-the-planet-of-the-apes.txt	0.000003
beauty-and-the-beast.txt	0.000001
logan.txt	-0.000002
coco.txt	-0.000021
happy-feet.txt	-0.000022
12-monkeys.txt	-0.000024
her.txt	-0.000037
big.txt	-0.00004

Related topics

This time, we search "monkey apes war for life". The related movies should be "War of the planet of the apes", "12–monkeys" and "Lord of war".

monkey apes war for live	
war-of-the-planet-of-the-apes.txt	0.002158
coco.txt	0.000137
12-monkeys.txt	0.000103
big.txt	-0.000018
her.txt	-0.000128
lord-of-war.txt	-0.00013
happy-feet.txt	-0.000137
dark-knight-rises.txt	-0.000156
beauty-and-the-beast.txt	-0.000189
logan.txt	-0.000191

Precision@3 is $2/3$.

Commonly occurs, related terms.

Search "boys girls". Many documents contain these two words, but the term frequency is not high. It's hard for VSM to handle this situation, but LSI can do better. Because LSI reduced the dimension and cluster related element to near each other.

boys girls	
happy-feet.txt	0.00011
dark-knight-rises.txt	0.000088
beauty-and-the-beast.txt	0.000085
lord-of-war.txt	0.000054
logan.txt	0.000047
big.txt	0.00001
her.txt	-0.000008
coco.txt	-0.000018
12-monkeys.txt	-0.000025
war-of-the-planet-of-the-apes.txt	-0.000032

Summary

From above contents, we can find, the SVD reduces the size of matrix that we need to compute with queries. But SVD needs lot time to compute, serialize the result of SVD into file maybe a better choice.

Dimension of LSI space k affects the accuracy of searching. When k matching the number of topics in the documents collection, we have the best searching result.