Latent Semantic Analysis — Deduce the hidden topic from the document

Sanket Doshi Feb 26, 2020 · 8 min read ★

Making computers learn and understand the human language is still the most difficult task. Language contains huge vocabulary and each work has different meaning based on the context and making computers learn the context is an open question. In this we will try to deduce the hidden topics represented by the text and will use that knowledge for document clustering.

Topic Model

Analytic vidhya — Topic Modeling

Topic model is an unsupervised way of deducing the hidden topics represented by the text or document. This topic is not the actual topics such as sports, news or business instead are the words that can be used for representing the text in the best possible way.

This technique is very powerful and can be used for document clustering in an unsupervised way. If you've used Google News then you've seen the clustering of news from various sources if the news represents similar topic. This is one of the application of topic modelling.

Latent Semantic Analysis

Latent Semantic Analysis is an efficient way of analysing the text and finding the hidden topics by understanding the context of the text.

Latent Semantic Analysis(LSA) is used to find the hidden topics represented by the document or text. This hidden topics then are used for clustering the similar documents together. LSA is an unsupervised algorithm and hence we don't know the actual topic of the document.

Why LSA?

Most simple way of finding similar documents is by using vector representation of text and cosine similarity. Vector representation represents each document in the form of vector. This vector is known as document-term matrix.

For example:

```
a1 = "the petrol in this car is low"
a2 = "the vehicle is short on fuel"
```

Consider above two strings and form the context we can understand that both the strings are similar. We'll try to find how much this strings are similar using vector representation.

The document term matrix for the above example is:

	car	fuel	in	is	low	on	petrol	short	the	this	vehicle
a1	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
a2	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0

Document-Term matrix

The size of the document-term matrix is (number of documents) * (vocabulary size). Vocabulary size is the number of unique words present in all the documents all together. Here the vocabulary size is 11 and number of documents are 2.

Similarity between documents if found out using cosine similarity between the documents matrix. The similarity between documents `a1` and `a2` is 0.3086067 which is too low since the documents are mostly similar in context. This is the disadvantage of document-term matrix and hence, the vector representation technique. Another disadvantage is the vocabulary size as the language has the huge vocabulary causing the matrix to be bigger and computationally expensive.

This disadvantages of the vector representation has led to the requirement of new technique for finding the similarity among the documents and finding the hidden topics. The technique which can solve the problem of synonyms and is also computationally not expensive. And the technique which was suggested was Latent Semantic Analysis.

Working of LSA

Term Co-occurrence Matrix

This matrix is of dimension (vocabulary size) * (vocabulary size). It represents the frequency of the words coming together in the dataset. The matrix helps us understand the words which belongs together.

For the above example the term co-occurrence matrix is:

	car	fuel	in	is	low	on	petrol	short	the	this	vehicle
car	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
fuel	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0
in	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0
is	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	2.0	1.0	1.0
low	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0
on	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0
petrol	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0
short	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0
the	1.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0
this	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0
vehicle	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0

Term co-occurrence matrix

As we can see the words `the` and `is` are the most common but are not very useful in meanings of the sentence. We'll see how to use this matrix and it's benefits later in this blog.

Concepts

The LSA returns concepts instead of topics which represents the given document. The concepts are list of words which represents the document in the best possible way.

For example, in the dataset of sports document the concepts can be

Concept 1: ball, shoes, goals, win

Concept 2: ball, bat, score, umpire

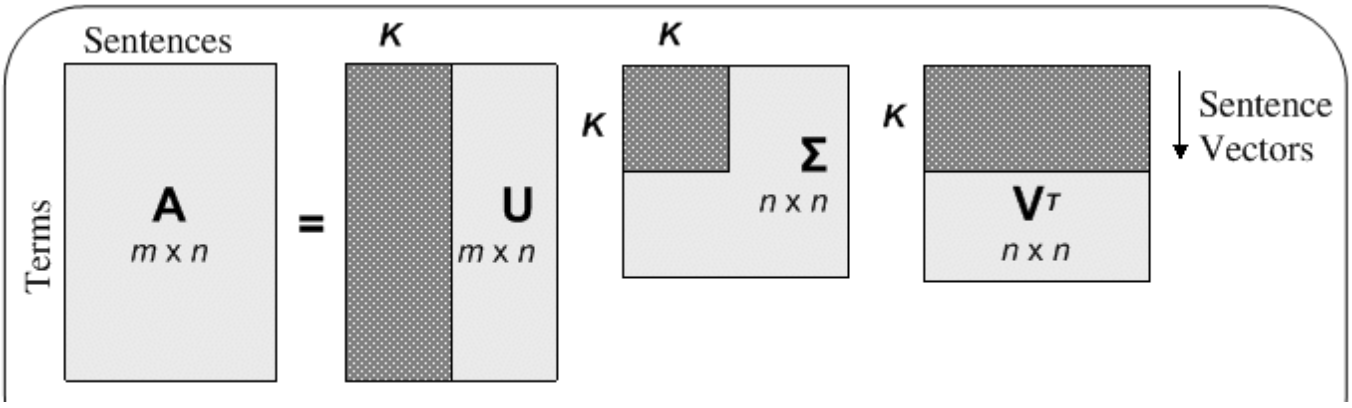
As we can see two concepts — concept 1 represents football and concept 2 represents cricket. But we can see that the concepts can have overlapping of words so, the whole set of words represent one concept together rather than the individual word. LSA tries to find the best set of words known as concept to represent the document using term co-occurrence matrix.

Concept is also a way of representing the document through dimension reduction.

Singular Value Decomposition (SVD)

We can see that the document-term matrix is very sparse and large in size. The computation on such a large matrix is expensive along with not very significant results and much of the values in the matrix are zero. To reduce the computation complexity and to get the more relevant and useful result SVD is used.

SVD decomposes the matrix into three different matrixes: orthogonal column matrix, orthogonal row matrix and one singular matrix.



SVD (researchgate)

The main advantage of SVD is that we can reduce the the size of the matrix substantially from millions to 100 or 1000. In the above image `K` is the rank of the matrix. It resembles that if we use only k columns and rows then also we can approximately calculate the matrix A without any major loss.

During SVD calculations we calculate `A*(A'T)` which represents the term co-occurrence matrix. That means the value with index `(i,j)` in the above matrix represents the number of times term(i) and term(j) exist together in a dataset of documents. You learn about SVD more here.

Implementation

Implementation is the best way to understand the concept. We'll implement LSA using a small example that will help us understand the working and output of LSA.

The documents we'll be using are

```
a1 = "He is a good dog."
a2 = "The dog is too lazy."
a3 = "That is a brown cat."
a4 = "The cat is very active."
a5 = "I have brown cat and dog."
```

Here we can see that 2 concepts must be generated one which represents cat and another represents dog.

Convert this list of documents to DataFrame:

```
import pandas as pd
df = pd.DataFrame()
df["documents"] = [a1,a2,a3,a4,a5]
df
```

The `df` would look like:

	documents
0	He is a good dog.
1	The dog is too lazy.
2	That is a brown cat.
3	The cat is very active.
4	I have brown cat and dog.

Documents dataframe

Preprocessing

The most important part of any machine learning algorithm is data preprocessing. More the noise present in data lesser the accuracy of model.

We'll perform four types of processing on data:

1. Remove all the special characters from the text.
2. Remove all the words with less than 3 letters.
3. Lowercase all the characters.
4. Remove stop words.

```
#remove special characters
df['clean_documents'] = df['documents'].str.replace("[^a-zA-Z#]", " ")

#remove words have letters less than 3
df['clean_documents'] = df['clean_documents'].fillna('').apply(lambda x:
' '.join([w for w in x.split() if len(w)>2]))

#lowercase all characters
df['clean_documents'] = df['clean_documents'].fillna('').apply(lambda x:
x.lower())
```

For removing stop words we'll tokenise the string and than again append all the words which are not stop words.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')

# tokenization
tokenized_doc = df['clean_documents'].fillna('').apply(lambda x:
x.split())

# remove stop-words
tokenized_doc = tokenized_doc.apply(lambda x: [item for item in x if item
not in stop_words])

# de-tokenization
detokenized_doc = []
for i in range(len(df)):
    t = ' '.join(tokenized_doc[i])
    detokenized_doc.append(t)

df['clean_documents'] = detokenized_doc
```

After this preprocessing our data will look like:

	documents	clean_documents
0	He is a good dog.	good dog
1	The dog is too lazy.	dog lazy
2	That is a brown cat.	brown cat
3	The cat is very active.	cat active
4	I have brown cat and dog.	brown cat dog

After cleaning the documents

Document-Term matrix

We'll use `sklearn` for generating the document-term matrix.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english', smooth_idf=True)

X = vectorizer.fit_transform(df['clean_documents'])
```

We used `TfidfVectorizer` instead of `CountVectorizer` as tf-idf is more efficient vectorizer. You can learn about various params passed to `TfidfVectorizer` here and to learn about tf-idf you can check this link.

The shape of `X` will be `(5,6)` where rows represents the number of documents that is 5 and columns represents the terms which are 6.

To see the terms

```
dictionary = vectorizer.get_feature_names()
dictionary
```

which will give an array of words

```
['active', 'brown', 'cat', 'dog', 'good', 'lazy']
```

Singular Value Decomposition

```
from sklearn.decomposition import TruncatedSVD

# SVD represent documents and terms in vectors
svd_model = TruncatedSVD(n_components=2, algorithm='randomized',
n_iter=100, random_state=122)

lsa = svd_model.fit_transform(X)
```

`TruncatedSVD` performs SVD function on the document-term matrix and gives us the vector after dimensionality reduction. If you want the matrix without dimensionality reduction you should use `fit` instead of `fit_transform` .

`n_components` is the dimension of output data. The value of `n_components` represents number of different topics. You can learn more about sklearn SVD here.

Now we'll check the topics assigned to our documents

```
pd.options.display.float_format = '{:,.16f}'.format
topic_encoded_df = pd.DataFrame(lsa, columns = ["topic_1", "topic_2"])
topic_encoded_df["documents"] = df['clean_documents']
display(topic_encoded_df[["documents", "topic_1", "topic_2"]])
```

The output will look like

	documents	topic_1	topic_2
0	good dog	0.3413834191239959	0.7199781067501040
1	dog lazy	0.3413834191239958	0.7199781067501034
2	brown cat	0.8609490919302158	-0.3659836550739513
3	cat active	0.5166658991993210	-0.3850046207843266
4	brown cat dog	0.9494117370834857	0.0236302940661153

LSA

We can see the topics assigned to each document. The documents regarding dog's is represented by topic_2 and the documents regarding cats is represented by topic_1. The last documents which has both cat and dog is represented more by topic_1 but belong to topic_2 too. It's more resembled by topic_1 as the document contains word `brown` and `cat` both which has higher weight in topic_1.

We can also see the weight given to the terms in each topic.

```
encoding_matrix = pd.DataFrame(svd_model.components_, index =
["topic_1","topic_2"], columns = (dictionary)).T
encoding_matrix
```

	topic_1	topic_2
active	0.2003541259081117	-0.2424408501618364
brown	0.5965117122287049	-0.2018098984872574
cat	0.6293380994160956	-0.3298859088715313
dog	0.4158307960649448	0.6169033286639758
good	0.1323826028466488	0.4533766476433699
lazy	0.1323826028466494	0.4533766476433687

We can see above that both term `brown` and `cat` has higher weightage in topic_1 than topic_2.

We've seen the implementation and working of LSA.

Applications

The LSA is the pioneer for LSI and dimensionality reduction algorithms.

1. The LSA is used for dimensionality reduction. We can reduce the vector size drastically from millions to mere thousands without losing any context. This will help us in reducing the computation power and the time taken to perform the computation.
2. The LSA is used in search engines. Latent Semantic Indexing(LSI) is the algorithm developed on LSA. The documents matching the search query are found using the vector developed from LSA.
3. LSA can also be used for document clustering. As we can see that the LSA assigns topics to each document based on the assigned topic we can cluster the documents.