Data Mining Group Project

Part 2: Text Clustering

Group4

Teacher: Nicolas TURENNE

1730026058 Alvin LIANG

1730014072 Michael XU

1730026153 Hawthorn ZHAO

Data Science

2019.12.23

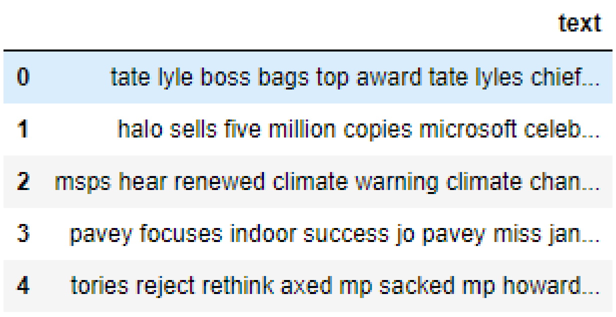
Table of Contents

Background Information

Recent years, with the prevalence of the natural language processing (NLP), people have proposed many ways related to process the natural language. One way of them is text clustering. Text clustering is the application of cluster analysis to textual documents. The goal of text clustering is to divide a given text into different categories according to a certain similarity principle, in which the text within the same category is more similar, while the text of different categories is less similarity. As I have learned many methods in Data Mining course, and also our group is interested in the text mining, and I am interested in the text clustering, so in this part, I will firstly show you how k-means and hierarchical clustering algorithms work in the text data, then introduce a model called Latent Dirichlet Allocation which is useful in extracting the topics among large text data.

Introduction of Dataset

The dataset we used in this part is the collection of news from BBC, it contains 2225 documents which are in the .txt format and 5 categories: business, entertainment, politics, sport, and tech. After cleaning the data (like removing the punctuations and tokenizing the words), we get the following data frame:



	text
0	tate lyle boss bags top award tate lyles chief...
1	halo sells five million copies microsoft celeb...
2	msps hear renewed climate warning climate chan...
3	pavey focuses indoor success jo pavey miss jan...
4	tories reject rethink axed mp sacked mp howard...

Where each row represents a news extracted from the original dataset. In the following contents, we will work on this dataset.

K-means

K-means algorithm is a classical clustering algorithm in supervised learning. So we will firstly use k-means method to divide the articles into different clusters based on their similarities.

Bag of Word (BOW)

The first thing we need to do is use the Bag of Word (BOW) method to transform the word into vector. The idea is to put all the words in a bag, regardless of the morphology and word order, that is, each word is independent. For example, consider the following two sentences:

Jane wants to go to Shenzhen.

Bob wants to go to Shanghai.

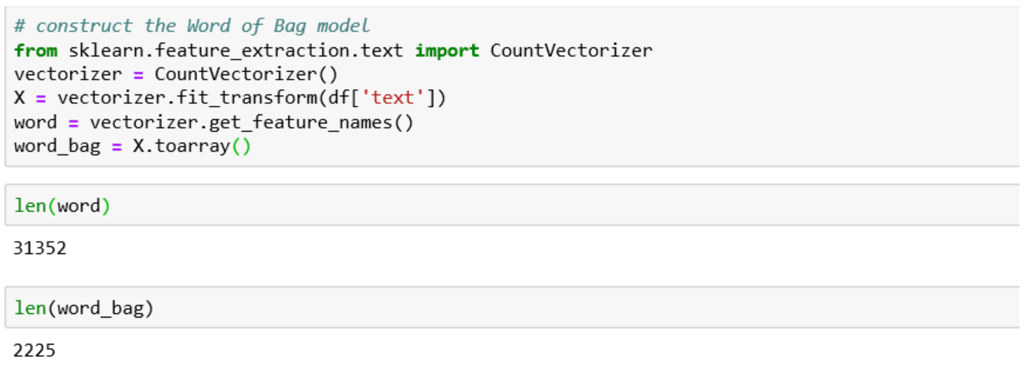
The corresponding bag of words is: [Jane, wants, to, go, Shenzhen, Bob, Shanghai]

The two examples above can then be represented by two vectors that match the index of the mapped array, the number of occurrences of the word, shown as following:

[1,1,2,1,1,0,0]

[0,1,2,1,0,1,1]

In python, there is a useful method in the sklearn library called CountVectorizer that can help us get the word-vector based on the Bag of Word model automatically.



```
# construct the Word of Bag model
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['text'])
word = vectorizer.get_feature_names()
word_bag = X.toarray()
```

```
len(word)
```

```
31352
```

```
len(word_bag)
```

```
2225
```

We can see that there are totally 31352 words in our dataset.

Cosine Similarity

The second step is to determine the method to calculate the distance between two vectors. There are many methods to calculate the distance, and for text data the most appropriate one is to use cosine similarity, because cosine similarity pays more attention to the difference in direction between two vectors than in the length.

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i * B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

To simplification for our task, we normalized the cosine similarity to the interval [0, 1]. The higher the values, the more similarity we have.

Algorithm

The third part is the core part, which we will use K-means method to cluster the text data. The idea is similar to the idea in the lecture notes:

(1) Randomly select k initial points as our centers

(2) Find the nearest of centers for each point in the dataset and assign it to corresponding cluster

(3) Update the center of each cluster to the average of all points in that cluster

(4) Repeat step (2) and (3) until the means are unchanged.

Based on the above idea, I implement the K-means algorithm by myself, the codes are show as follows:

```python
# Construct a random set of k centers of mass
def randCent(k):
    import random
    cent = []
    randomlist = random.sample(range(0,test.shape[0]),5)
    a = test.tolist()
    for i in randomlist:
        cent.append(a[i])
    return cent
```

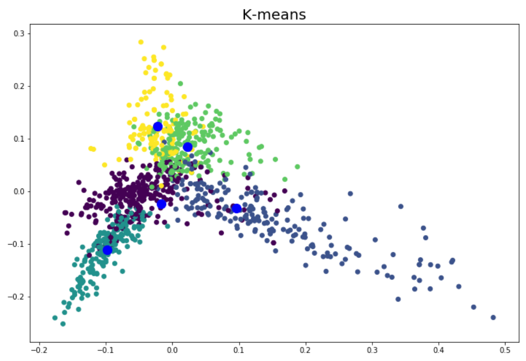
```python
def kMeans(k, distMeas = cos_sim, createCent= randCent):
    from math import inf
    m = test.shape[0]  # total number of vectors

    # The first column is the index of the center of mass, and the second column is the cosine simil
    clusterAssment = np.mat(np.zeros((m, 2)))
    centroids = createCent(k)  # Construct a random set of k centers of mass
    b = test.tolist() # Turn word_bag into list
    i = 0;
    while i<50:
        clusterChanged = False
        for i in range(m):  # Each point given in the total data
            minDist = inf  # INFINITE
            minIndex = -1
            for j in range(k):  # each center
                distJI = distMeas(centroids[j], b[i])  # calculate the cosine similarity
                if distJI < minDist:
                    minDist = distJI
                    minIndex = j # record the corresponding point
            if clusterAssment[i, 0] != minIndex:
                clusterAssment[i,:] = minIndex, minDist

        # print centroids
        for cent in range(k):
            ptsInClust = test[np.nonzero(clusterAssment[:,0].A == cent)[0]]
            centroids[cent] = np.mean(ptsInClust, axis=0)
    i = i+1
    return centroids, clusterAssment
```

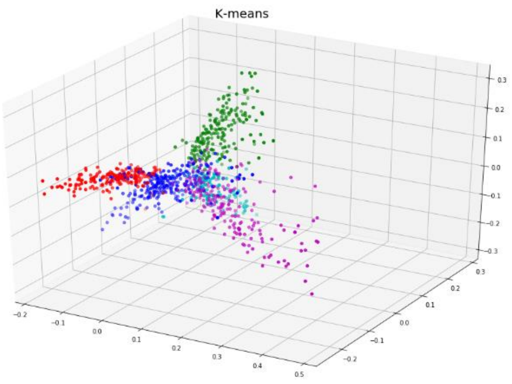
Notice that when I firstly ran the codes, the program died due to the size of data and the memory of machine, in this case, I just selected the first 1000 rows from the dataset and reduce the dimension by PCA to 1000 dimensions. In addition, since the convergence of the K-means is slow, in this case, I just used 50 iterations.

Visualization and Explanation of the Results

In fact, we need to adjust the parameter k based on some evaluation methods like DBI (Davies-Bouldin Index) to find the best k for our model. However, our dataset has already divided into 5 categories, in this situation, I just used k = 5 for demo.

I used both 2D figure and 3D figure to give you the intuitionistic results





The big blue point in the 2D figure represents the centers after 50 iterations. From the result, we can see that most articles can be clustered but some of them are overlapped. Possible reasons behind it include the less number of iterations and the high dimensional of the data.

Hierarchical Clustering Algorithms

Introduction

As shown above, since K-means method are not efficient in the high dimensional data, so next I will introduce the Hierarchical clustering algorithms that is helpful in clustering the high dimensional data.

Hierarchical clustering algorithm has two types, including the Agglomerative (bottom-up) and Divisive (top-down). In our project, we used the Agglomerative clustering algorithms.

Before using the Agglomerative clustering algorithms, we need to extract the features from the text. We used the same method in K-means, using the python built-in method CountVectorizer.

Algorithm

The idea of the Agglomerative clustering algorithms is shown as follows:

(1) Compute the distance matric over our dataset

(2) Initialize each point as different cluster

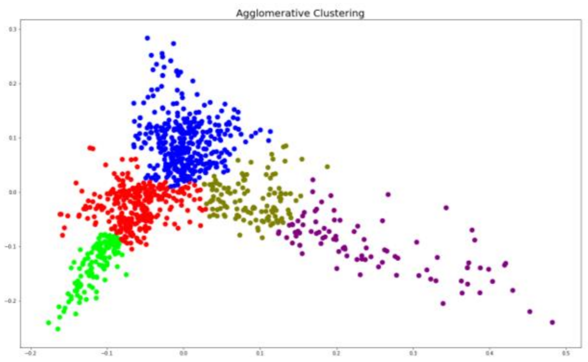
(3) Merge the two closest clusters

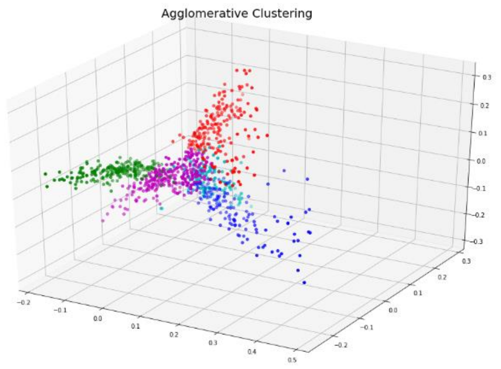
(4) Update the distance matrix

(5) Repeat step (3) and (4) until only one cluster remains

For this algorithm, I used the method called AgglomerativeClustering in the sklearn library.

Visualization and Explanation of the Results





From these graphs we can see that the result is better than the previous one. There are clearly 5 clusters in the graph although the boundary areas are ambiguous. This result also prove that Hierarchical Clustering algorithm is better in the high-dimensional space.

Latent Sematic Analysis

Introduction

Latent sematic analysis (LSA) is an unsupervised learning method which is very useful in extracting the topics among large text data. In text information processing, traditionally, the semantic content of text is represented by word vector and the semantic similarity between texts is represented by the measure of word vector space. However, sometimes the word has polysemy, which means that the word has many meanings. In this case, traditional way cannot recognize these different meanings. LSA model aims to solve the problem that this method cannot accurately represent the semantic meaning, and tries to find the potential topic from a large amount of text data. The topic vector is used to represent the semantic content of the text, and the topic vector space is used to measure the semantic similarity between the texts more accurately.

Definitions

Before giving you the LSA algorithm, let's give you some definitions first.

Topic Vector Space

Given a text set $D = \{d_1, d_2, \ldots, d_n\}$and a corresponding word set W= $\{w_1, w_2, \ldots w_m\}$, the word vector space (also called word-document matrix) X can be obtained as:

$$X = \begin{bmatrix} x_{11}, x_{12}, \ldots, x_{1n} \\ x_{21}, x_{22}, \ldots, x_{2n} \\ \ldots \\ x_{m1}, x_{m2}, x_{mn} \end{bmatrix}$$

Where X is the original word vector space and each column is the representation of the text in the word vector space.

Assume that all texts have totally k topics, and each topic is represented by m-dimension vector defined on the word set W, called topic vector, that is:

$$t_l = \begin{bmatrix} t_{1l} \\ t_{2l} \\ \ldots \\ t_{ml} \end{bmatrix}, l = 1,2, \ldots, k$$

Where t_{il} is the weight of the word w_i in the topic t_l, the higher the weight, the higher importance of the word in this topic.

Topic Vector Space T (also called word-topic matrix) is defined as follows:

$$T = \begin{bmatrix} t_{11}\ t_{12} \ldots\ t_{1k} \\ t_{21}\ t_{22} \ldots\ t_{2k} \\ \ldots \\ t_{m1}\ t_{m2} \ldots\ t_{mk} \end{bmatrix}$$

Topic-Document Matrix

Now consider one element dj in the text set D, which is represented by a vector x_j in the word vector space, let's project x_j to the topic vector space T, we can get a vector y_j:

$$y_j = \begin{bmatrix} y_{1j} \\ y_{2j} \\ \ldots \\ y_{kj} \end{bmatrix}, j = 1,2, \ldots, n$$

Where y_{lj} is the weight of the text dj in the topic t_l $l = 1,2, \ldots, k$

Therefore, the representation of the text in the topic vector space, also called topic-document matrix, is defined as:

$$\mathrm{Y} = \begin{bmatrix} y_{11}\ y_{12} \cdots\ y_{1n} \\ y_{21}\ y_{22} \cdots\ y_{2n} \\ \cdots \\ y_{k1}\ y_{k2} \cdots\ y_{kn} \end{bmatrix}$$

Algorithm

The idea of LSA model is shown as followings:

> The text set is represented as the word-text matrix, and the singular value decomposition of the word-text matrix is carried out to obtain the topic vector space and the representation of the text in the topic vector space.

The first step is to transform the given text set and word set into the word-document matrix X:

$$\mathrm{X} = \begin{bmatrix} x_{11}, x_{12}, \dots, x_{1n} \\ x_{21}, x_{22}, \dots, x_{2n} \\ \cdots \\ x_{m1}, x_{m2}, x_{mn} \end{bmatrix}$$

Then we will use the singular value decomposition method to decompose the matrix based on the user defined topic number k, that is:

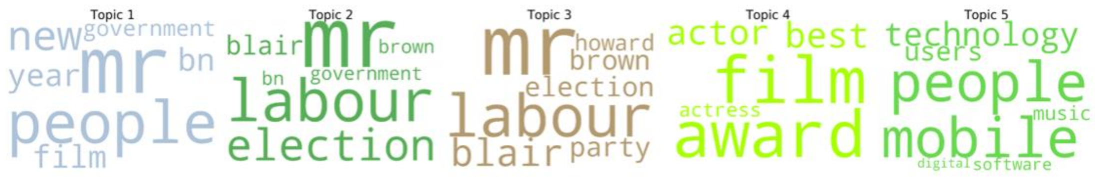
$$\mathrm{X} = U_k \Sigma_k {V_k}^T = [u_1, u_2, \dots, u_k] \begin{bmatrix} \sigma_1\ 0\ 0\ 0 \\ 0\ \sigma_2\ 0\ 0 \\ \cdots \\ 0\ 0\ 0\ \sigma_3 \end{bmatrix} \begin{bmatrix} {V_1}^T \\ {V_2}^T \\ \cdots \\ {V_k}^T \end{bmatrix}$$

Where the topic vector space is U_k and topic-document matrix is $\Sigma_k {V_k}^T$

Visualization and Explanation of Result

Following this idea, we will use our dataset to extract the topic of our model.

The result is shown as follows:



From the result we can see that the fifth topic represents the technology, the fourth topic is about entertainment. However, for the first three topics, the boundary is ambiguous, we cannot easily determine which topic they represent.

Latent Dirichlet Allocation

Introduction

From above we can see that the LSA model can only extract part of the topics, there are some topics that are hard to determine. In this case, we will use another model called Latent Dirichlet Allocation (LDA). LDA is a probability model for generating text sets. Model assumes that the topic is represented by multinomial distribution of words, the text is represented by the multinomial distribution of topics, and the prior distributions for both word distribution and topic distribution are Dirichlet distribution. The difference of text contents is caused by the difference of topic distribution.

Dirichlet Distribution

Dirichlet distribution is a probability distribution of multiple continuous random variables, which is an extension of beta distribution.

If the probability density function of multiple continuous random variables $\theta = (\theta_1, \theta_2, \dots, \theta_n)$ is:

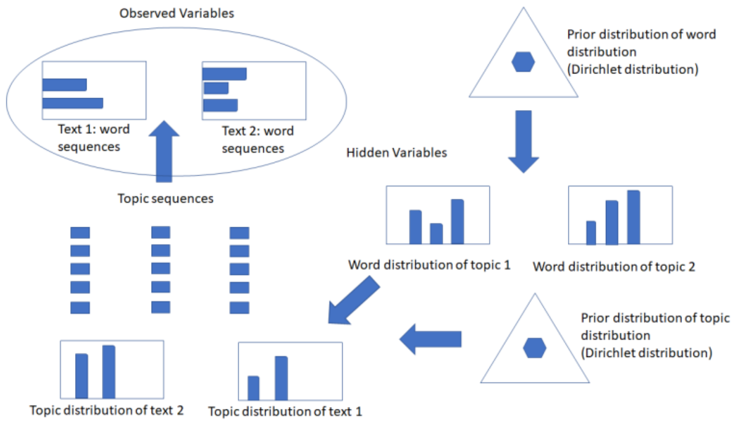
$$p(\theta|\alpha) = \frac{\Gamma\left(\sum_{i=1}^{k} \alpha_i\right)}{\prod_{i=1}^{k} \Gamma(\alpha i)} * \prod_{i=1}^{k} \theta_i^{\alpha_i - 1}$$

Where $\sum_{i=1}^{k} \theta_i = 1, \theta_i \geq 0, \alpha = (\alpha_1, \alpha_2, \dots, \alpha_n),\ \alpha_i > 0, i = 1,2, \dots, k$. Therefore, we call the random variable θ follows Dirichlet distribution with parameter α, denoted as $\theta \sim \mathrm{Dir}(\alpha)$.

Algorithm

The idea of LDA model is: firstly, based on the prior distribution of word distribution (Dirichlet distribution), we generate multiple word distribution, that is to determine multiple topic contents; then based on the prior distribution of topic distribution (Dirichlet distribution), we generate multiple topic distribution, that is to determine multiple documents contents. Then we generate topic sequence based on each topic distribution, as well as for each topic, based on the word distribution in that topic, we generate words that compose word sequences as a whole, which is to generate the texts. Repeat this process until generating all texts.

The graph below shows the generation of text by LDA model:


Observed Variables
Text 1: word sequences
Text 2: word sequences
Topic sequences
Prior distribution of word distribution (Dirichlet distribution)
Hidden Variables
Word distribution of topic 1
Word distribution of topic 2
Prior distribution of topic distribution (Dirichlet distribution)
Topic distribution of text 2
Topic distribution of text 1

The core formula of LDA model is:

$$p(w|\mathrm{d}_m) = \mathrm{p}(\mathrm{w}|\mathrm{z}_\mathrm{k}) * \mathrm{p}(\mathrm{z}|\mathrm{d}_m)$$

Where w is the word, d_m is the document and z_k is the topic.

Algorithm:

(1) For each topic z_k, k = 1,2, … K

Generate multinomial distribution parameter $\varphi_\mathrm{k} \sim \mathrm{Dir}(\beta)$, as the word (w) distribution of topic, denoted as $\mathrm{p}(\mathrm{w}|\mathrm{z}_\mathrm{k})$

(2) For each text d_m, m = 1,2, … M

Generate multinomial distribution parameter $\theta_\mathrm{m} \sim \mathrm{Dir}(\alpha)$, as topic distribution of text, denoted as $\mathrm{p}(\mathrm{z}|\mathrm{d}_m)$

(3) For each word w_{mn} in the document d_m, m = 1,2, … M, n = 1,2, …,N_m

(a) generate topic $Z_{mn} \sim Mult(\theta_\mathrm{m})$, as the word corresponding topic

(b) generate word $w_{mn} \sim Mult\left(\varphi_{z_{mn}}\right)$

Model Implementation

At this point, I have introduced the behind theoretical knowledge, next we will use python lda library to implement.

First step is to vectorize the text data. For LDA model, based on the article(Blei, 2003), the features extraction model I use is the Bag of Word (BOW) model, so I should use the CountVectorizer function in sklearn library.

The next step is to train the model. In fact, we need to adjust the parameter “number of topics” based on some evaluation methods to find the best parameter for our model.

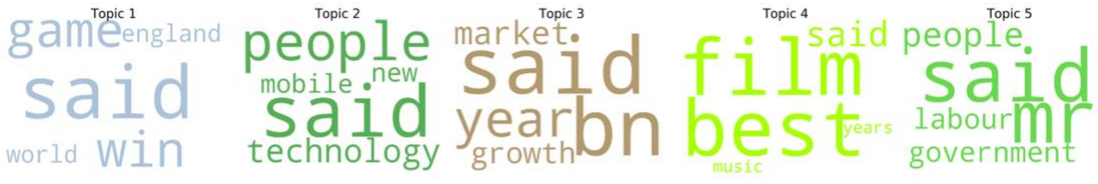
However, our dataset has already divided into 5 categories, in this situation, I just used the parameter = 5 for demo. And I used 500 iterations.

Visualization and Explanation of Results

After training the model, we get the following information:

```
INFO:lda:n_documents: 2225
INFO:lda:vocab_size: 9106
INFO:lda:n_words: 389205
INFO:lda:n_topics: 5
INFO:lda:n_iter: 500
```

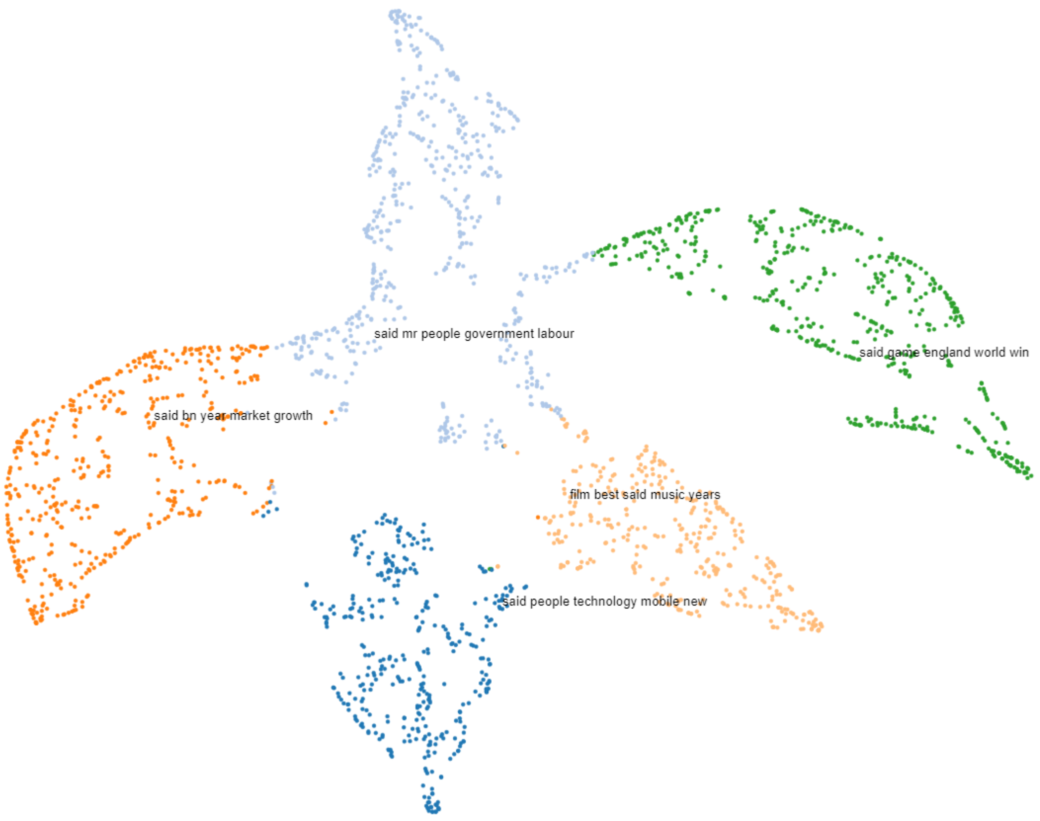
To give you the intuitionistic feeling, I extracted the top 7 words in each topic and draw the graph:



From the graph we can see that the topic can be determined much easier than the LSA model. Topic 1 is about sport, topic 2 is about tech, topic 3 is about business, topic 5 is about entertainment and topic 5 is about politics.

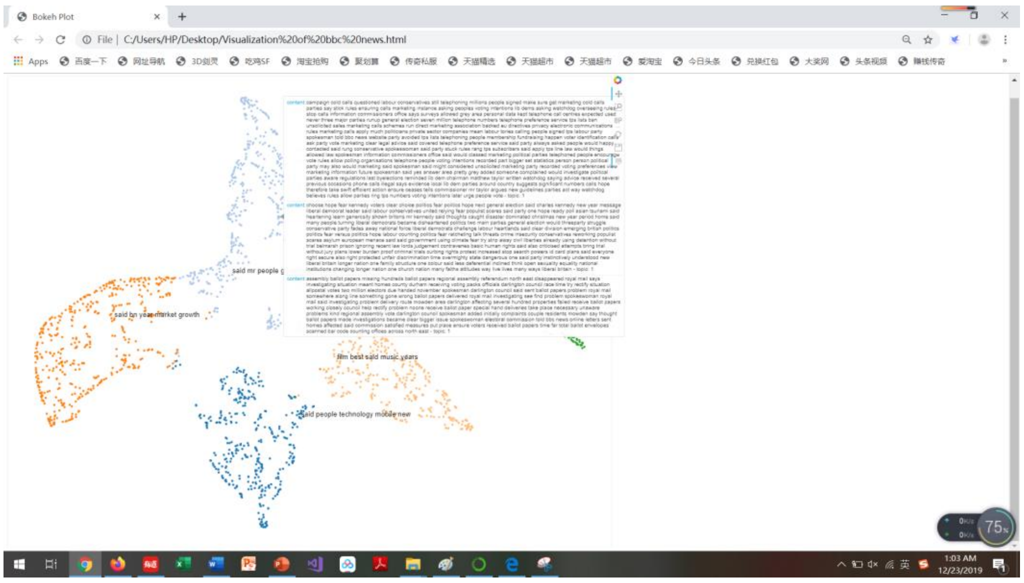
Further Work

At this point, we only extract 5 topics from our dataset. However, we want to know the exact the belongness of each article and to get an intuitionistic feeling, in this case, I will use one powerful library called bokeh and the dimension-reduce tool tsne. The result is shown as follows:



Each point in the graph represents an article, while the text represents the extracted topics.

If move the mouse to the point, we can get the contents of the article. (The graph is quite ambiguous, if you want, you can check in the Visualization of bbc news.html file)


Bokeh Plot
File | C:/Users/HP/Desktop/Visualization%20of%20bbc%20news.html
said mr people
said bn year market growth
film best said music years
said people technology mobile new
1:03 AM
12/23/2019

References

BleiJordanNg,. (2003, 3). Latent Dirichlet Allocation. Journal of Machine Learining Research, 933-1022.

LandauerKThomas. (2006). Latent Semantic Analysis. London: Latent Semantic Analysis.

李航. (2012). 统计学习方法. 北京: 清华大学出版社.

Bokeh: https://docs.bokeh.org/en/latest/

tsne: http://hackage.haskell.org/package/tsne

Sklearn: https://scikit-learn.org/