Knowledge discoveries based on text mining techniques

Weijing CHEN, Xiaokun XIE, Ziyuan HE, Zixin GUO

I.Introduction

In recent years, online pop culture, subcultures and buzzwords are full of network environment. The Oxford English Dictionary defines a subculture as “a cultural group within a larger culture, often having beliefs or interests at variance with those of the larger culture.” According to Wikipedia: a buzzword is a word or phrase, new or already existing, that becomes very popular for a period of time. These kinds of text usually have strong characteristics. They usually have a unique form of expression that comes from a phenomenon or an event and are very frequently used by net users who likes it.

Figure 1.1 several kinds of subcultural groups that exist on the China Internet

It is important to identify and classify such expressions in large amounts of text. If we get a classifier that can recognize subcultural related texts, we can know what users are talking about, and we can know which subcultural group a user belongs to, then we are able to know user interests and further discover discussion topics. For business applications, proper classification of network culture gatherings helps to accurately target advertising and help us to know users’ interests for recommendation tasks.

II. Goal

2.1 Objective

Implementing a subculture text classifier based on the dataset collecting from the Internet. It should be noticed that all the subculture texts are in Chinese.

2.2 How to realize

To realize our project, we mainly based on python. We use libraries like pandas, numpy, gensim, pytorch, sklearn, etc as support for data collecting, data preprocessing, data cleaning and model training, testing and model deploying.

III. Collecting Data

3.1 Requirments

To discover knowledge behind different Internet subculture groups, the first thing we should do is to collect data from different user groups. According to our prior knowledge, subculture user groups often appear in popular social applications or forums. So we sort out about 11 categories of subcultures group. Together with a negative class 'other', in which reviews are from ordinary users who doesn't belong to Any subculture groups, we finally have 12 categories to collect. Table below shows abbreviations of all 12 categories we were going to collect. We are not going to explain stories behind these labels because that's not our topic.

Table 3.1 abbreviations of all 12 categories we were going to collect

baixue	fzl	gdh	hasi
jojo	liuxue	ly	quin
ym	zhexue	goufensi	other(negative examples)

3.2 Collecting data through distributed Spider System

All the datasets we used in this project are manually collected from Sina Weibo and Baidu Tieba. For Sina Weibo, we collected the comments of microblogs which the senders are specified by us. Also, for Baidu Tieba, we collected comments and replies in postbars (similar to forums). These Weibo bloggers and postbars correspond to some online pop culture, subcultures or buzzwords in recent years.

Since this project requires a large amount of data set, we need a data collection framework that supports scalability and fault tolerance. And, nowadays internet companies have mechanisms to protect their data, so we need a framework to overcome their anti-spider system. Therefore, we use weibospider[1] (a repository in GitHub with over 3,000 stars), a distributed crawler for weibo, building with celery and requests. The framework uses redis for communication between nodes, and the captured data is stored in the MySQL database. For ease of deployment, we employ Elastic Compute Cloud (EC2) and Relational Database system (RDS) of Amazon Web Services (AWS).

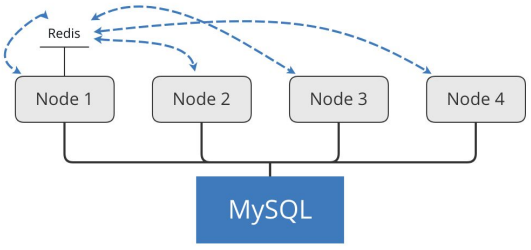

Figure 3.1 The architecture of weibospider

Also, we use another repository in GitHub to crawl Baidu Tieba, which is Tieba_Spider[2]. This framework utilizes scrapy to do web crawling job then stores data in

[1] https://github.com/SpiderClub/weibospider
[2] https://github.com/Aqua-Dream/Tieba_Spider

MySQL database. It can crawl all the posts in a Baidu Tieba, including comments and replies of each post.

It takes us about a week to collect enough data from Sina weibo and Baidu Tieba. In total we deploy four nodes for parallel spiders and we bought several Weibo accounts for preventing banning. With this configuration, we are able to collect 10000 Weibo reviews per hour. Because there's no anti-spider system in Baidu Tieba, so we are able to collect every review in a Tieba in a very short time.

Finally, we collected 870MB dataset. In total we got over 300,0000 Chinese reviews from Sina Weibo and Baidu Tieba, table 3.2 below gives the details of data sources.

Table 3.2 Details of data sources

Sources	Amount	From
Weibo	303,184	@带带大师兄 @Sharon張寶華 @Mr_Quin @六学bot @头条新闻
Tieba	2,787,914	jojo的奇妙冒险, mrquin, 真夏夜的银梦, 真夏夜之银梦, 刃牙, 兄贵, 白色相簿2, 白学, 非主流, 淋语, 三木木2, 中华城市, bilibili, 六学, 六学家, 明星

3.3 Structure of our subculture dataset

After getting origin raw data, we export them out from our MySql database and organize them into csv format. Below is the inner data structure of our collected data.

Table 3.3 CSV Inner structure

Column Name	**content**	**user**	**label**	**time**
Description	review content	user's id	which categories of subculture/ buzzword it may belong to	time this review was posted

So far, we have collected enough data from our project. However, before starting training a classification model, firstly we need various data preprocessing steps to refine our data.

We will introduce our data cleaning and preprocessing procedures in the next sections.

IV. Data preprocessing

4.1 Data Cleaning

At the very beginning of any data mining tasks, data preprocessing is of vital importance.

Firstly we category every review to one of 12 classes (11 subculture class and 1 negative class). A review's label is based on its original sources, we assume that subcultural

users always appear together in related Tieba or microblog so the reviews' contents are highly related to its sources. For example, if we collect a review from Tieba 'liuxue', it will be firstly labelled as 'liuxue', though it's content may not be related to this subcultural topic.

Because raw data from the Internet always contains useless symbols or meaningless review, we then use keywords to filter out those simple meaningless reviews like 'Good', 'Wow, interesting', 'Thank you', etc. Then we set a length limit, review whose length is less than 5 is abandoned. Besides, reviews contain web linkage and picture comments are eliminated. All reply prefixes like '@user:' are replaced by an empty string ''.

We develop a filter class and use it to go through all the collected samples to pick up clean data.

4.2 Chinese Word Segmentations

Unlike English or other languages, there are no spaces between every single Chinese word. So the first step is word segmentation, which is a step of great importance in Chinese natural language processing.

PKUseg [3] is the latest powerful word segmentation package in python. It outperforms traditional segmentation tool 'JieBa' and support for web text segmentation. This is good for doing our tasks.

We develop a parallel computation program to execute PKUseg and it takes us about ten minutes to finish segmentations.

4.3 Word Embedding through Word2vec

In order to represent every word in the documents, traditional method is to use one-hot encode. To extract document features, traditional methods are tf-idf based or bag of words model. However, we think traditional method can't capture semantic information within words well. Since these models need long and spare vector to represent a document or words, it may be very computation expensive and memory consuming. So we consider using word embeddings for text feature extraction.

Recently distributed word representations methods play a very important role in NLP area and Recommender System area. The main idea of distributed word representation is to represent words or documents as a dense low-dimension vector. These vectors bring rich semantic information and are usually used for downstream tasks like reasoning or sentimental analysis. The lateset pre-trained embedding method BERT which is proposed by Google research group has been proven super efficiently in various kinds of NLP tasks.

Word2vec [1] which is proposed in 2013 is a classic, effective and efficient three-layer neural network word embedding algorithms. It's based on Skip-gram model. Its main idea is that words share similar context are similar to each other. The training tasks of word2vec model are to let embedding vectors of words that share the same context have higher dot product with each other, while not similar words have small dot product with each other. By this way, similar words will have similar embeddings vectors, thus semantic are captured. For example, 'Beijing' and 'Paris' often have 'Capital' nearby, so 'Beijing' is similar to 'Paris' and this similarity can be captured by word2vec.

This is a better method than simply represent a word with a gigantic long one-hot encode. So here, every review is viewed as a document, we used gensim library as support and build an embedding library for 525,768 words. Below is the parameters we set when training the word2vec models:

Table 3.4 Word2vec parameters

Parameter	**Value**
window size	2
embedding dimension	100
negsample rate	0.01
epoch	30
min count	1

To check the quality of embeddings vectors, we simply manually check several words' most similar words. The result is quite satisfying. Result are shown in the table below. This proves that the embeddings capture semantic information in text.

Table 3.5 top 10 most similar words

Input words	Top 10 most similar
爸爸（normal words）	妈妈,女儿,老爸,爷爷,母亲,爷爷,爹,老公,儿子
深圳（normal words）	广州，东莞，广东，珠三角，香港，北京，珠海，广西，惠州
哈哈（normal words）	哈哈哈，呵呵，哈哈哈哈，嘻嘻，啊，哎，哈哈哈哈，....
承太郎（subculture words, an animation character）	阿强,DIO,dio,迪奥,神父，乔尼，二乔，乔瑟夫，卡兹，空条 (They are all animation character)

4.4 Generating features for every record

In the previous section we introduce how we extract semantic information through word2vec model. As we mentioned above, traditional method use tf-idf to extract document features. Here, we use two way to generate features for every review:

(1) We add all embedding vectors of words of a review together and take an average. This generates a fixed 100 dimension length features.

(2) We concatenate all words embeddings together to generate a long embedding word sequence whose length is not fixed.

Then, we store extracting features in the format of numpy pickle.

4.5 Refine Data

In section 4.1 we mention that every single review is labelled based on its data sources. However, as we examined our datasets we find out that not always every review are talking about subcultural related topics or are using buzzwords. Therefore, we came up with an idea to further refine our data.

(1) For some classes, we add some subcultural related keywords to filter related samples

(2) For some classes, we manually select hundreds of samples as positive samples, and 'other' class is regarded as negative examples. Then we train a soft margin SVM classifier(Gaussian kernel). Then we use this bi-classifier to find out more related samples in the rest of the data.

By now, we've got enough data of good quality for training a classifier that is expected to learn useful language patterns. The refined data contains 705097 reviews. Table below give details of each class. Notice that sample number of classes are **imbalance**.

Table 3.6 details of refined dataset

class Abbreviation	sample number	class Abbreviation	sample number
baixue	1824	fzl	22767
goufensi	115260	hasi	1390
zhexue	56478	gdh	92916
other	96089	ly	79217
ym	121630	quin	7365
liuxue	15928	jojo	90528

4.6 Whole procedure of data preprocessing

Figure below illustrates the whole procedure of our data preprocessing method clearly, it summarize what we introduce above in this section.

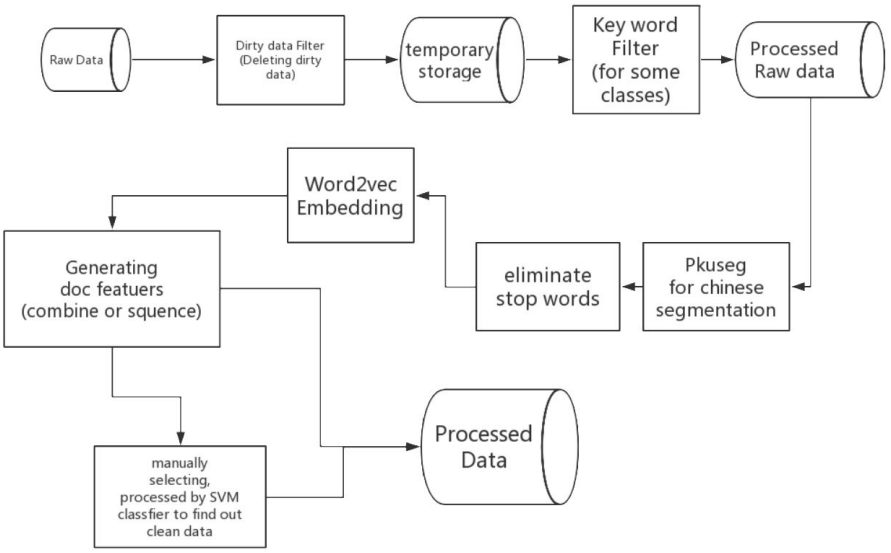

Figure 4.1 The whole procedure of data preprocessing

V. Classifications

As what we introduce in the first section, we would like to build a multi-class classifier that can learn to recognize different kinds of buzzwords and subcultural related review so that we can further to mine more knowledge in users' reviews. 100% accuracies aren't what we want. We want the classifier outputs probabilities of belonging to which classes because sometimes net users will mix different buzzwords and subcultural meme together, it's not reasonable to classify it to a certain class. But training a classifier with high precision and recall can let it learn more from the data. We try 4 different kinds of classification algorithms: they are KNN, RandomForest, MLP, and RNN. **We applied Holdout and stratified sampling here**: for every class we sample 20% as testing sets and the rest 80% are training set.

5.1 KNN

K-nearest neighbour is one of the most famous and classic classification algorithms. As we mention in the data-preprocessing section, we add all word2vec embeddings of words in a review together and take average to generate review features.

We import KNN algorithms from sklearn libraries, then we simply input all 700,000+ review features and corresponding labels into the algorithm. We enable kd tree to speed up prediction process.

We record performances of different k value. It seems that KNN can't do well in classifying these 12 categories. This is shown in the figure below:

For convenience, we simply use accuracies to check the performances in our preliminary experiments. If an algorithm can't get high accuracy, it's not necessary for us to go further. From figure 5.1 we can see KNN doesn't perform well. No matter how we adjust the k value. The best accuracies we get is **66.7%** when k value is 10. When k is large, for example, 30 or when k is small we get lower accuracies.

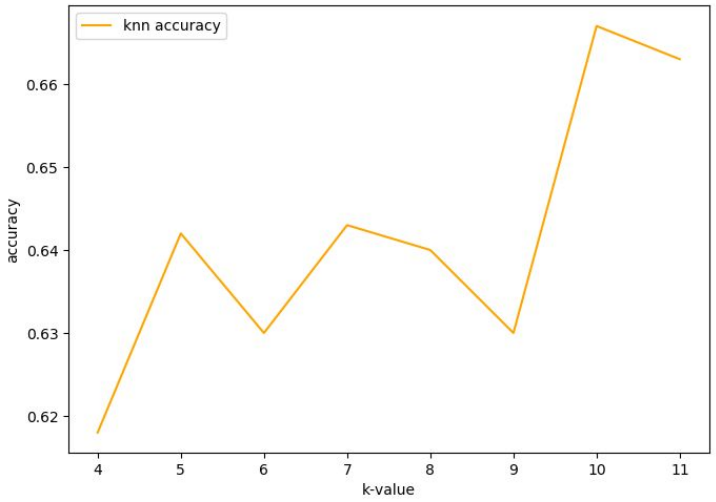

Figure 5.1 The accuracies of KNN with different k

5.2 RandomForest

We also test Random Forest algorithms on our dataset. We import random forest algorithms from sklearn libraries. The parameters are set as below:

n_estimator: 200, max_features:'auto', bootstrap: True

However, it seems that Random Forest is still not working as good as we expected. Under this set of parameters, it's accuracy is only **67.5%**, which is the best accuracies we can get. If we set n_estimator to a large number, it takes too long to output a result.

5.3 Multilayer Perceptron:

We also try to use MLP - "Multilayer Perceptron" to do classification tasks, which is also called "Fully Connected Layer" in deep learning. It is a semi-automatic algorithm, requiring us to set the loss function of the final output, and then it could run and optimize by itself through optimizers, such as Stochastic Gradient Descent (SGD) or Adam Optimization algorithms .etc.

To be specific, for the first stage, all the weights are initialized through some initializers, subjecting to some distributions and may having some statistics. Input is processed by perceptron together with activation function, which means that any input could be processed by linear and nonlinear transformation. From the angle of theory, any function could be simulated after being propagated by one layer in the algorithm.

For setting parameters, we try a lot and get the best result during experiments. For loss function, we use "cross entropy", which is shown below. In addition, we also add "L2 norm" item to alleviate the overfitting problem.

"Cross entropy" could measure the difference between predictive distribution and real distribution of data, which is a special condition of "KL divergence".

In the theory of machine learning, with the increasing of parameters, more data is required for training. This requirement is huge and subjects to power. If we do not have so much data, then overfitting will happen. "L2 norm" could alleviate overfitting greatly.

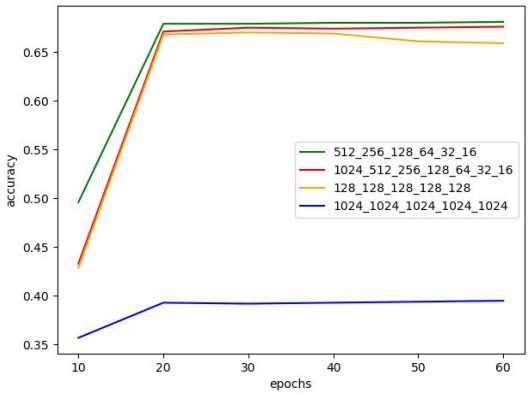

Figure 5.2 The accuracies with different settings of MLP

Based on pytorch framework, we realize a Multilayer Neural Networks whose structures can be easily modified by us. However, deep MLP networks still can't produce a satisfying result. We try 4 kinds of neural networks of different width and depth but however the best result we can get is **67.6%**, can't outperform random forest or KNN algorithms significantly. The results are shown in figure above. Legend '512_256_128_64_32_16' means the MLP has 512 units in the first layer, 256 units in the second layer, and so on. During experiments, the parameters are set as below:

optimizer: Adam, learning rate: 0.001, batch_size: 10000, epochs, l2_regularization: 0.001

5.4 Recurrent Neural Network

After previous preliminary experiments, we still can't get a satisfying result. So we reflect, what leads to this? Because word sequence plays an important role in semantic, so simply adding all word vector together and then take average to generate a review features is not a good idea for feature extractions. The best way is to keep the sequence structure. Recurrent Neural Network is a kind of neural network good at learning from sequential data. RNN is proven effective and help researchers to achieve a great improvement in the area of NLP these years.

RNN has a memorizing mechanism that can memorize what it goes through. So it can handle data sequences of different lengths and react according to what it memorized.

In our implementation, we construct a 10 layer Recurrent Neural Network. The basic unit of our network is LSTM. We take the output of the last timestamp as the review features and feed it into a one layer Linear neural network for classification. Figure below shows the structure of the constructed Recurrent neural network.

Training RNN is quite slow, so we didn't make many experiments to test parameters. Now, using a 10 layer in depth, 128 units in width RNN gives us the best result. To train the recurrent neural network, here we use the sequential features dataset generate in the data preprocessing step. We construct our RNN classifier using python pytorch framework. During experiments, our parameters are set as below:

epoch: 20, learning rate: 0.0001, optimizer: Adam, batch_size:500

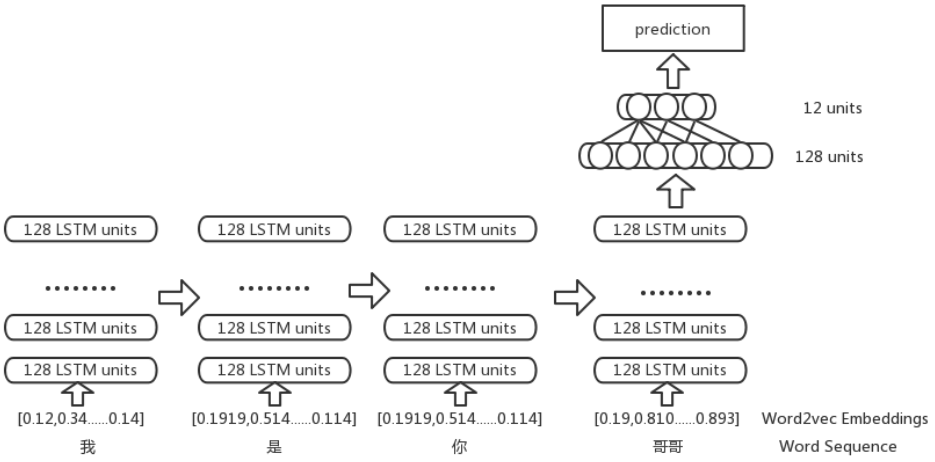

Figure 5.3 RNN structure

5.5 Performances Analysis

Finally, we compare best performances of all 4 classification methods we mentioned above. We can see RNN based method give the best result.

Table 5.1 Accuracies Comparison

Method	**Best Accuracies**
Baseline - ZeroR	16.3%
KNN	66.7%
Random Forest	67.5%
MLP	67.6%
RNN	**87.0%**

We further go deep to analyze the classification quality of RNN. We compute recall and precision for every class:

Table 5.2 The precisions and recalls of each class

Class Label	**Precision**	**Recall**
baixue	0.865	0.75
fzl	0.993	0.986
gdh	0.978	0.977
goufensi	0.902	0.903
hasi	0.957	0.845

liuxue	0.936	0.910
ly	0.940	0.861
other	0.741	0.771
quin	0.910	0.874
ym	0.800	0.857
zhexue	0.983	0.980
jojo	0.905	0.9417

The result is quite satisfying. Though it seems that it can't well recognize 'other' class, but this RNN model works well in the downstream tasks. We will talk about how we going to handle some downstream tasks using RNN as a feature extractor.

VI. Downstream Tasks

6.1 User features extraction and clustering

After getting an RNN model which can output probabilities of which classes a review may belong to, this RNN can be used as a features extractor. The RNN model outputs a 12-dimension vectors in which each entry represent a probability value. If we input all users' review into the RNN model, add all vectors together and take an average, then we get a user feature vector. Users who belong to the same subcultural group may have very similar features vector. By this way we can mine users' user interests hidden in their reviews automatically and divide users into groups.

Further, we collect all reviews from a Tieba called 'bilibili'. In total, we have 20,000 users in this Tieba. We use python library to group users. Then, every users' history reviews are transformed into word2vec embeddings and fed into RNN model. Finally all users' features are extracted from all of their reviews.

However, now we get 20,000 users features, they are 12 dimension vectors which are very hard to visualize.

T-SNE[2] is one of the most famous algorithms for dimension reduction and data visualization. By reducing dimension using T-SNE we are able to keep high-dimensional structure while convenience for visualizing clustering qualities. It's not expensive to reduce 20,000 12-dim vectors to 20,000 2-dimension vectors so we transform all user features into 2D space.

Then, we start to run clustering algorithms on the 2D user features.

The first K-means algorithms we tried is K-mean algorithms, a classic and standard clustering algorithm. However, we found out that no matter how we adjust the number of cluster center, still some points obviously should not belong to a cluster will be put in the same cluster. We adjust cluster number from 20 to 50, the problem still exists, as shown in

Figure below. This is not acceptable for us because this shows it is unable to discover structural pattern in the 2D features.

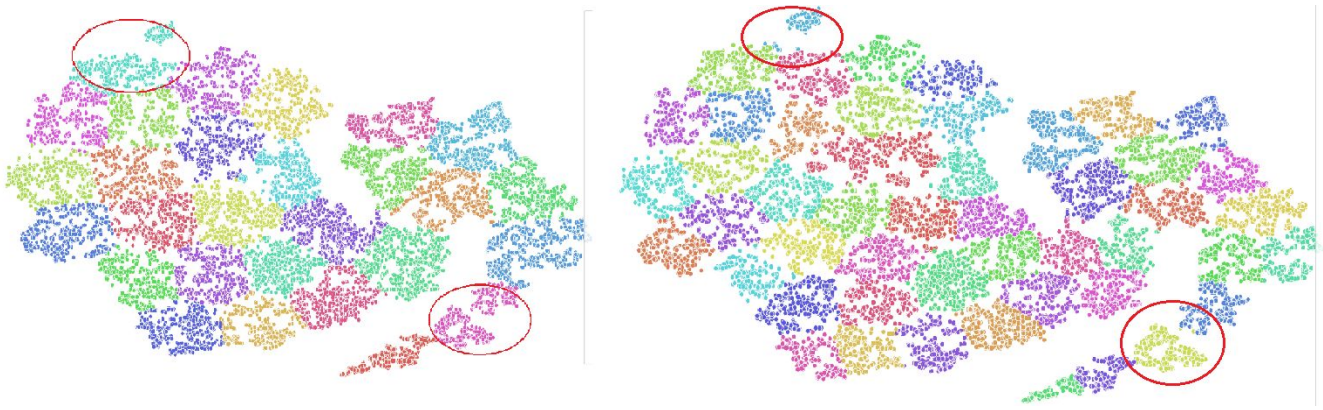

Figure 6.1 Problems of Kmean

Then, we try DBSCAN which is taught in our class. DBSCAN works well if clusters are well separated from each other and have the same density. However, according to our experiment results, no matter how we adjust the epsilon value and min_sample value, it will only lead to two results:

(1) form a super large number of clusters, up to 200, however, many samples are labelled as noises, as shown in the left one in figure below.

(2) Form a large cluster. However, in the right figure, we can't say that user on the most right is similar to the user on the most left. Though it can discover some structural pattern, result still not satisfying.

Figure 6.2 Problems of DBSCAN

Finally, we try hierarchical clustering, using 'ward'[5] method and 'average' method to evaluate cluster distance. This gives the best result: a structural pattern can be discovered and won't form a super large cluster, every cluster members are concentrated in a local area, so they may are similar to each other.

Both ward method and average method give a good visualizing result, however it's hard to tell which one is better.

Using silhouette score supported by sklearn we are able to evaluate clustering qualities. Start from cluster center number of 27, we compute score for two methods and we get a line chart below:

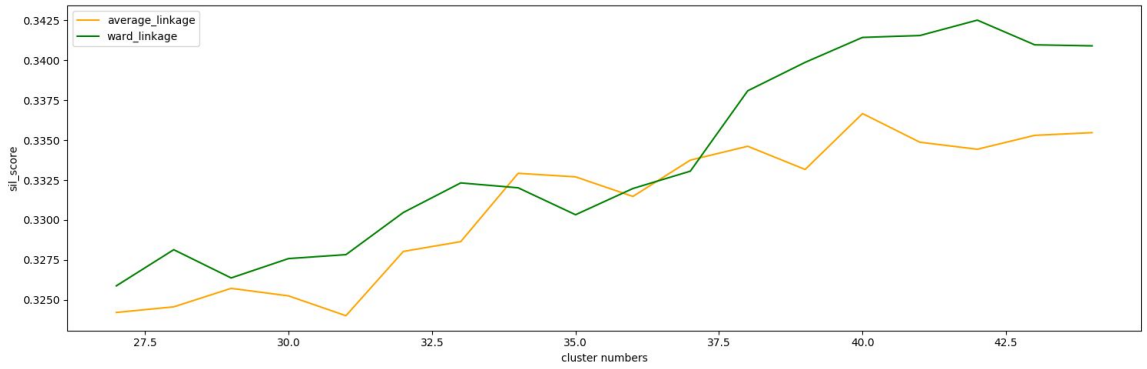

Figure 6.3 silhouette scores with different numbers of clusters

We can see when the cluster number is 42 and using ward linkage it gives the highest score, so finally we choose 42 and ward linkage. Figure below is our final result:

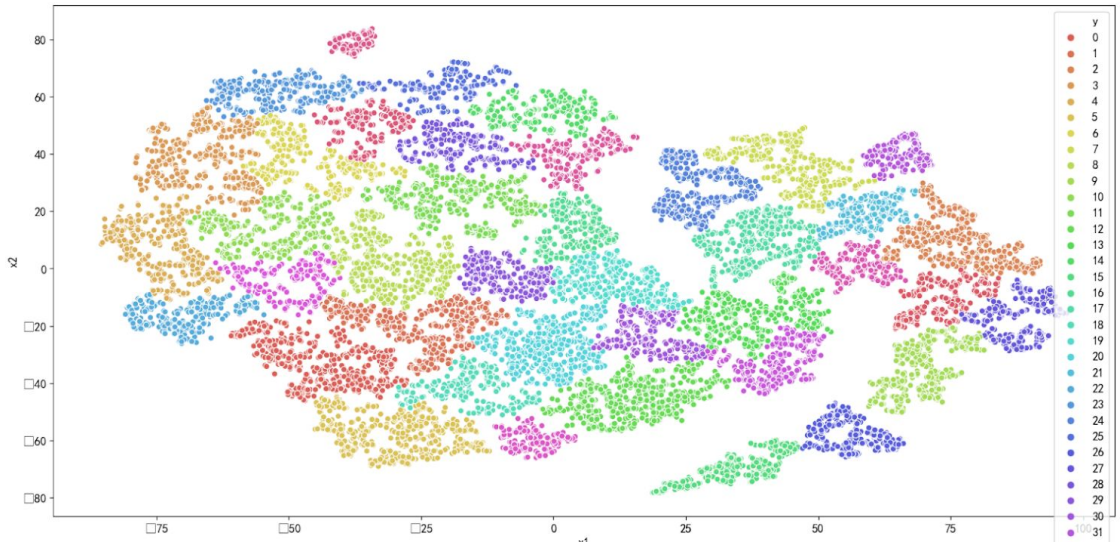

Figure 6.4 hierarchical clustering, n_cluster = 42

To check the quality of the clustering, at last we randomly take some users from a same cluster, and load their reviews. To our expectation, the sampled users' review have similar style(belong to some subcultural or use similar buzzwords), this is shown in figure below.

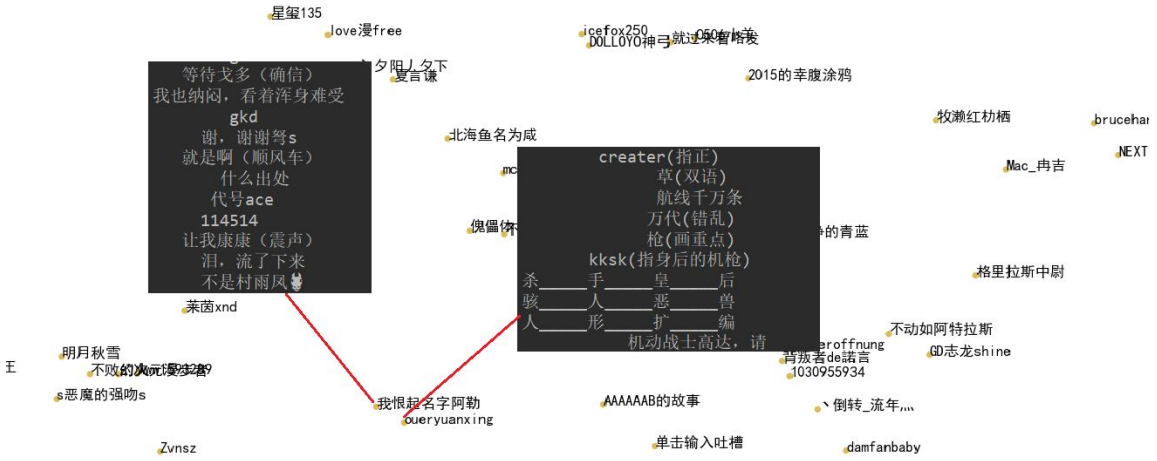

Figure 6.5 similar user

6.2 Video tagging based on review feature extraction

Our another mining task is video tagging. Video comments provided rich information about video topics. Especially for subcultural related video, comments under it are very different from ordinary videos and can be easily recognized. Using our RNN feature extractor we are able to guess it's content and tagging the video. This useful because tagged video can be used for content-based recommendation tasks to improve users' experiences.

We further collect hundreds of videos' comments from a famous Chinese bullet screen(danmaku) website - bilibili and we give them tags based on their contents. We want to know if we can predict a videos' tag based on it's comments. Due to time limitation, we are only able to collect 6 kinds of videos of different subculture. Table below gives details of collected data.

Table 6.1 Number of each subculture texts we collected

Content	Number
goufensi	313
liuxue	40
ly	80
other	175
quin	7
ym	251

Like what we have done before, every comment is transfromed into word embeddings, and are fed into RNN model. All comment features are added together and we take an average. Then again we get a 12 dimension long vector. This video will be tagged as one of the 12 class with the highest value.

The experiment result is quite good. This method gives very high precision and for some classes. However, for some classes the recall rate is quite low. Anyway, we think by this method our program can recognize contents of some video accurately, it has learnt knowledge from our training data.

Table 6.2 The precisions and recalls

Class	Percision	Recall
goufensi	0.992	0.972
liuxue	0.845	0.942
ly	0.964	0.700
other	0.980	0.864
quin	1	1
ym	0.929	0.961

VII. Conclusion and Team Distribution

7.1 Team Distribution

Tasks are evenly distributed to our teammates:

Weijing Chen: word2vec embeddings, builder of KNN, SVM, RandomForest and RNN network. Clustering and video taggings and quality evaluations, writer of section 5.4, 5.5, 6.

Xiaokun XIE: Responsible for all data collection tasks: distributed spider system design and implementation, database manager, version management, writer of Section 2, 3.

Zixin Guo: Manually data preprocessing of class: liuxue, etc. Builders of MLP algorithms, early experiments on RNN, evaluations of algorithms. Writer of section 1,5.3,7.

Ziyuan He: Plan for data preprocessing, Data preprocessing of class ym,renya,jojo , word2vec embeddings, word segmentation, evaluations of embedding qualities. Writers of section 4.

7.2 Conclusion

This project follows the whole procedure of a data mining task: data collecting, data preprocessing, model building, model evaluations and model deploying.Firstly we build a distributed spider system to collect data from social websites. We test several models on our self-collected dataset and get a well-trained RNN classifier which is widely used for NLP tasks in recent years. And its good performances in downstream tasks prove that it has learnt useful language pattern from labelled user reviews.

VIII. References

[1]Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
[2]Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of machine learning research*9.Nov (2008): 2579-2605.
[3]Xu Sun, Houfeng Wang, Wenjie Li. Fast Online Training with Frequency-Adaptive Learning Rates for Chinese Word Segmentation and New Word Detection. ACL. 253–262. 2012
[4]Mikolov, Tomáš, et al. "Recurrent neural network based language model." *Eleventh annual conference of the international speech communication association*. 2010.
[5] Ward, J. H., Jr. (1963), "Hierarchical Grouping to Optimize an Objective Function", Journal of the American Statistical Association, 58, 236–244.