International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 2 (2019) pp. 447-451

Automated Text Clustering and Labeling using Hypernyms

Tammishetti Vishnu
Department of Information Technology
Sree Nidhi Institute of Science and Technology

Konda Himakireeti
Department of Information Technology
Sree Nidhi Institute of Science and Technology

Abstract

Automated text clustering and labeling is a process of dividing a corpus of documents into one (or) more cluster(s) and then choosing an appropriate label for each of the clusters which will be descriptive of that particular cluster. This lets the users or anyone wishing to use these documents get an easy and quick understanding of the documents with the help of the labels we provided. These labels are aimed at being the words which are descriptive of the words or phrases in the cluster. This process results in one or a few clusters and their label(s). So, a corpus, a collection of documents is divided into clusters using the K-Means clustering algorithm which divides the data into predefined number of clusters. It works on numbers only, so we calculate Tf-idf(Term Frequency and Inverse Document Frequency) and use this data to cluster the data using K-Means. We use Wordnet API to find the synonyms and then find the hypernyms. We then label the cluster based on the hypernym with highest frequency. This procedure can be used with a light or no variation for document labeling also. When these documents are required, a glance at these cluster labels must be enough to get the basic idea of the corpus under consideration. This might help a lot especially when there are a lot of documents and data to be looked at. So, in this paper of ours we describe an approach of doing so.

1. INTRODUCTION

Data invariably helps us in many places or circumstances. Having data can be considered a boon. Having adequate data helps in many ways like analysis, forecasting and what not. It helps in understanding the situations better, it helps in understanding some patterns like in stocks, oil price and the list goes on. Earlier, data was very scarce. There wasn't adequate amount of data available for analysis, as a matter of fact any other job involving the need of data. Having sufficient data can even play an important role in saving lives. If provided with the data about weather patterns and all, tsunamis can be predicted and required safety measures can be taken. Things can be handled. All these things put emphasis on the importance of data.

With the advent of internet and many more modern technologies, promising results have been shown. A lot of data is now readily available over the internet and most of it is free to access for everyone. According to Forbes article by Bernard Marr, there are about two and a half million bytes of data created every day at the current pace. The article says that 90% of the data available over the internet is created only in the past two years. This will of course commensurate with time.

This increase in data might seem like a boon and it is. But the only problem is that large portion (almost all of it) of the data available is unstructured. This is one of the biggest problems today. Unstructured data is the data which is not in a pre-defined way or simply not organized. This problem is in dire need to be addressed. A lot of unstructured textual is also available which need to be organized for usage in future. Hence the techniques or methods which can be used to mollify this problem are very much necessary.

Clustering and labeling them appropriately might help ease the situation up to a certain extent. Clustering is, taking similar or relative text and making it into one group called a cluster. Whereas labeling is nothing but assigning a word or a phrase which best describes a particular cluster. This optimistically will result in creating structured data out of unstructured ones. This can add meaning to the data. We can identify a particular document or a cluster of words by its label. This can also be used to label documents. This will help the users get a basic idea or overview of what these documents are or what is this cluster about. So, this clustering and labeling will help structuring the textual data and will improve user readability.

This paper of ours is a proposed method to achieve the above described results. We take a corpus that is a collection of documents which will be the data to be clustered and labeled. We calculate the Term frequency-Inverse document frequency and transform the data into one or more clusters. Then, we use the WordNet to find the synonyms of the words followed by finding their hypernyms and finally allocating a good label for that particular cluster.

This paper is further organized into two sections. Section one will be about the proposed approach and section 2 will show the results of the experiments conducted based on the approach described followed by the conclusion.

Note: We used python for this approach. So, all the examples shown in this paper are achieved through python and its various modules. Python has a package "nltk" which provides methods which ease up the process.

2. THE APPROACH

The whole approach is depicted by the following flowchart.

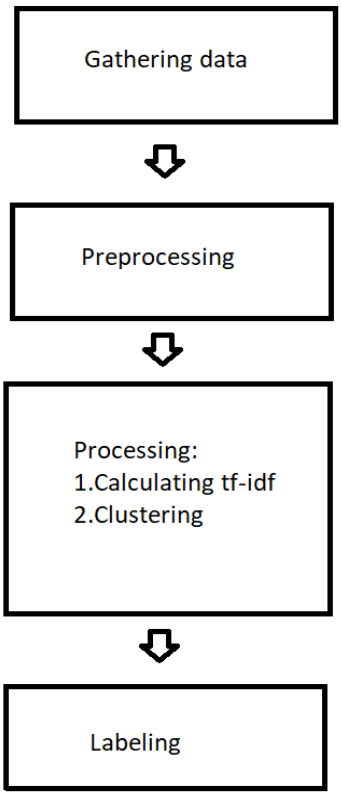

Figure 1. Flow chart of the technique

2.1 Gathering data

This is the first step, the input data. The data to be clustered is gathered. This data maybe anything, some random text or a document or even a collection of documents called a corpus. The random text maybe used to get a label for that and a document can also be used in the same way. A corpus of documents can be input to make some well-defined clusters. For more comprehensive understanding, this paper will be based on the input of a corpus of documents. Let the input be, D_c .

Where, D_c stands for a corpus of documents.

$D_c = D_1, D_2, D_3, \ldots, D_n$.

Where, D_i being a document on its own.

2.2. Preprocessing

This is one of the pivotal steps in this approach to achieve good results. As a matter of fact, this preprocessing plays an important role in any of the data related methods. Preprocessing deals with transforming data into an acceptable format for the process. It is basically readying the data, making it suitable to work on. Not doing this will have a negative effect on further process and will have a bad effect on the outcome.

Preprocessing for this approach takes 4 simple yet effective steps.

2.2.1 Tokenizing:

Tokenizing is the first and foremost step. Tokenizing refers to dividing a piece of text into tokens, words in our case, based on specific parameters. When the documents are input, they must be divided into sentences and these must further be divided into words using a Tokenizer. This is quite easy and it returns a list of words which are in the documents. This list allows us to implement the process in an efficient manner. We perform all the further operations on this set of tokens only.

2.2.2 Punctuation removal:

Punctuations like " !, ?, .." will only be a burden to deal with. Whatever we are doing, is on the words. We don't have to care about the punctuation. These punctuations only delay the process but not help it in any way unless we are trying to get the emotion of the text. Which we are not. We are only looking for a label which best describes a particular set of words. Hence, we didn't find a reason to keep these under consideration any further.

So, in this step, we are going to strip of any punctuations to proceed further.

2.2.3 Stop words removal:

Stop words removal is the most important step. Stop words are those which have to be filtered off before proceeding with natural language processing. These are the words which are most common in the language. A stop word will only be slowing down the process as they don't really carry a lot of information. A stop word can be "the" or "is" or "at" or "on" etc. As we consider each and every word or token, having these in the list will not be of use and moreover will serve as a disadvantage in the further process. Hence removing these will help a lot.

So, in this preprocessing, we perform the mentioned operations so that our process can be more efficient.

2.2.4 Stemming and Lemmatization:

Stemming basically refers to stripping of the end or beginning of the word by taking some common prefixes and suffixes into consideration.

Consider the following example,

Form	Suffix	Stem
studies	-es	studi
studying	-ing	study

Figure 2. Understanding Stemming

In figure 2, the words studies and "studying" have been stripped, actually stemmed to "studi" and study. "studi" actually cannot be used when we search for synonyms or hypernyms. Hence we consider Lemmatization. Lemmatization considers the morphological information of the words also.

For the same example,

Form	Morphological information	Lemma
studies	Third person, singular number, present tense of the verb study	study
studying	Gerund of the verb study	study

Figure 3. Understanding Lemmatization

Now after Lemmatization (Figure 3), we got study which can be worked with.

Note: All these pre-processing methods can be implemented in python as well as some other languages.

2.3 Processing

Processing in this paper's sense is clustering the textual data into different clusters.

This happens in two steps,

1) Finding tf-idf vector
2) Clustering

After these two steps have been applied, we would be having a desired number of clusters of words based on their significance calculated using tf-idf.

2.3.1 tf-idf:

tf-idf stands for Term frequency-Inverse Document frequency. This is widely used in text mining and information retrieval. This is a highly effective method.

What tf-idf does is it takes into account all the words in the collection of documents and computes each of the words significance with respect to that document. Simply put, it tells how important a particular word is.

Tf-idf is found for each and every word that made out of the preprocessing stage. To get a clear idea,

2.3.1.1 Term frequency:

This is the frequency of a particular word in a particular document. Term frequency of a word is the ratio of no. of occurrences of that word to the total no. of words in the document.

Ex: Consider a document d1,

"Hello this is a sentence"

Now, the term frequency,

tf("this",d1)= 1/5

As there is one occurrence of "this" in a total of 5 words, term-frequency of "this" in d1 is 1/5.

2.3.1.2 Inverse document frequency:

Idf is how much information that particular word provides. It answers questions like how frequent this occurs or how rare this occurs. This helps in finding out the significance or importance of a particular word.

Ex: Consider two documents d1, d2.

d1="this is first document"

d2="this is second document"

Now, idf("this",D)=log(2/2)=0.

Where D=d1, d2.

Similarly, idf("first",D)=log(2/1)=0.301

Now to calculate tf-idf we just have to multiply the term frequency and inverse document frequency.

Tf-idf("first",d1,D)=tf("first",d1) * idf("first",D)

Tf-idf("first",d1,D)=(1/4) * 0.301

Tf-idf("first",d1,D) ≈ 0.075.

The generic algorithm is as follows,

Using python and scikit learn we can directly generate something called TfidfVectorizer which automaticall computes the tf-idfs for us. Let's take a look,

texts = ["Penny bought bright blue fishes.", "Penny bought bright blue and orange fish.", "The cat ate a fish at the store.", "Penny went to the store. Penny ate a bug. Penny saw a fish.", "It meowed once at the bug, it is still meowing at the bug and the fish", "The cat is at the fish store. The cat is orange. The cat is meowing at the fish.", "Penny is a fish"].

	ate	blue	bought	bright	bug	cat	fish
0	0.000000	0.447123	0.447123	0.447123	0.000000	0.000000	0.000000
1	0.000000	0.452762	0.452762	0.452762	0.000000	0.000000	0.259093
2	0.571840	0.000000	0.000000	0.000000	0.000000	0.571840	0.327236
3	0.302651	0.000000	0.000000	0.000000	0.302651	0.000000	0.173192
4	0.000000	0.000000	0.000000	0.000000	0.768166	0.000000	0.219792
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.830757	0.316934
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.610644

fishes	meowed	meowing	orange	penny	saw	store	went
0.538647	0.000000	0.000000	0.000000	0.331817	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.452762	0.336002	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.488790	0.000000
0.000000	0.000000	0.000000	0.000000	0.673806	0.364602	0.258696	0.364602
0.000000	0.462702	0.384083	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	0.276919	0.276919	0.000000	0.000000	0.236701	0.000000
0.000000	0.000000	0.000000	0.000000	0.791905	0.000000	0.000000	0.000000

Figure 4. Results of tf-idf.

We input some text sentences in form of "texts", we used python's "TfidfVectorizer" to generate the tfidf values in our required form to further process.

Pre-processing is all done implicitly in python 3.7.x.

2.3.2 Clustering:

The main objective of this paper is to structure the data and label it. What other way to structure than group similar data together. Clustering is basically a process aimed at doing that very operation. It is used to group similar things into a cluster(think it of like a group or collection of similar things). So, clustering is a process of making clusters out of the input data.

There are quite a few clustering algorithms like K-Means clustering, Hierarchical clustering and their versions.

These two are the most widely used ones. However, we are focused only on K-Means clustering as we found it is better in terms of application and results.

2.3.2.1 K-Means clustering:

K-Means clustering is used to cluster n observations into k groups where k≤n.

In other words, K-Means clustering is use to make "K" clusters out of the given observations. Each cluster will be formed in such a way that all the observations in it are similar to each other in the very same cluster than other clusters. Let's take an example, given some random names and addresses, there can be a cluster of names and another cluster of addresses. Hence, it formed a cluster of only similar things. A name is conceptually more similar to another name than it is to an address.

This can be called a lazy method as it performs most of the computation when it has to perform the classification of the new observation. This is an instance based clustering technique. Its training experience is all the observations it previously encountered. Its performance increases with increasing number of observations fed to it.

K-Means creates "K" clusters, given K an integer. If K=3, it creates 3 clusters. That is, it classifies each of the observation into either one of the three clusters.

2.3.2.2 Working:

Initially, k is chosen and k base points are taken randomly. These are called centroids. Each centroid is a data point, or an observation. Whole k-means works on Euclidean distance property i.e. the distance between two observations in the Euclidean space. Euclidean space is where the points are plotted. Whenever a new observation is encountered, its distance to the k number of clusters is calculated. It is classified to be a part of the cluster whose centroid is the closest to the new point.

After classification of every new observation, the centroid is recomputed (Figure 6).

Let's take an example where k=2, we have the centroids represented by "x" mark in the following figure.

Figure 5. K-Means working - Initial stage

After re-computing the centroids,

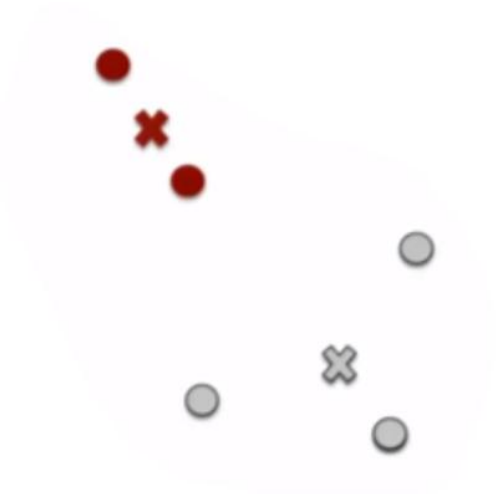

Figure 6. K-Means working - After re-compting the centroid

The following (Figure 7) is a figure of outcome of K-means clustering with k=3.

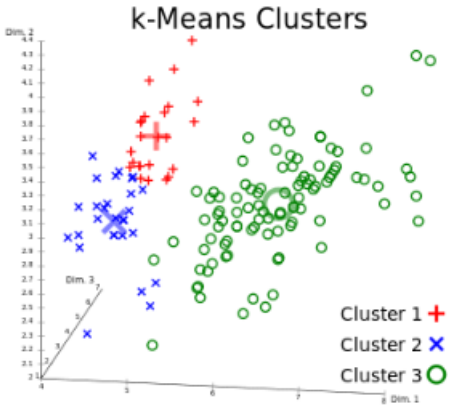

Figure 7. Example of K-Means having 3 centroids i.e., K=3

This clustering is a major step as this is how we cluster the textual data into clusters which will be labeled. Like most of the algorithms, even K-Means works with only numerical data. So, to convert the textual data we have into numerical data, we calculate the tf-idf of each and every word of the data we have after preprocessing. We generate a vector of Tf-idf values and we feed it to the K-Means algorithm and obtain "K" clusters.

This whole operation can be done using python its sklearn module. Now after generating the clusters, we label them.

2.4 Labeling:

This is the last phase of our procedure. In this step, we label the obtained clusters. A good label is the one which best describes the data in it. So far in our approach, we tokenized the data, we removed the stopwords, we calculated Tf-idf and then we clustered them. Now we have different clusters containing words. We have to choose a best descriptor of these words. We do so by generating the synonyms of all the words in the clusters using the Wordnet interface provided by Python. We have to first check if the word in in WordNet. If not we remove it from our cluster. We then generate the hypernyms of the remaining.

2.4.1 Hypernym:

Hypernym is a word broad in meaning which encompasses words more specific in meaning.

For example, color is a hypernym for red, blue etc.

So, we generate the hypernyms for each of the words including their synonyms present in our cluster.

Python code:

```
In [85]: syns_final[7]

Out[85]: Synset('field_hockey.n.01')

In [86]: syns_final[7].hypernyms()

Out[86]: [Synset('field_game.n.01')]
```

Figure 8. Understanding hypernyms

From Figure 8, we can see that it generated "field_game" as the hypernym for Hockey.

This hypernym generation helps as it broadens the context. It helps broadening the meaning of the words.

After these hypernyms are generated, we calculate the frequency the hypernyms in a particular cluster. We assign the hypernym with highest frequency or top hypernyms (as per requirement) as the label of the cluster.

Example:

Consider a simple cluster.
cluster=['dog','cat','leopard','melon','india','rat','lion']

By finding the synonyms and hypernyms, we found that the highest frequency hypernym we found was 'feline'. This is actually a good description of the above mentioned cluster as Cat, Leopard and Lion fall under feline species.

3. CONCLUSION AND FUTURE DIRECTIONS

This process of clustering and labeling is aimed at dividing textual data into chunks of labelled blocks of data. This is very useful in organizing data and further helps in understanding data quickly and efficiently. When properly done, a quick glance will be enough to understand the brief point of the data under consideration. However, this process is highly dependent on the performance of the clustering mechanism we used. If the cluster is comprising of vaguely related words, the label assigned will invariably be vague as well. For the future, we are aiming at developing or tweaking the present clustering models which will help in forming better clusters, which will lead to a concise label.

REFERENCES

[1] Rajaraman, A.; Ullman, J. D. (2011)."Data Mining". Mining of Massive Datasets (PDF).*pp. 1–17.* doi:10.1017/CBO9781139058452.002. ISBN 9781139058452.

[2] Comprehensible and Accurate Cluster Labels in Text Clustering Jerzy Stefanowski and Dawid Weiss Institute of Computing Science, Poznan University of Technology Piotrowo 2, 60–965 Poznan, Poland{jerzy.stefanowski,dawid.weiss}@cs.put.poznan.pl

[3] Article by Bernarnd marr https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#13c947f960ba

[4] Samuel J. Rivera, Barbara S. Minsker, Daniel B. Work, Dan Roth. "A text mining framework for advancing sustainability indicators", Environmental Modelling & Software,2014.

[5] N. Yuvaraj, A. Sabari. "Twitter Sentiment Classification Using Binary Shuffled Frog Algorithm", Intelligent Automation & Soft Computing, 2016.

[6] Deniz Iren, Hajo A. Reijers. "Leveraging business process improvement with natural language processing and organizational semantic knowledge", Proceedings of the 2017 International Conference on Software and System Process - ICSSP 2017, 2017.

[7] Text Documents clustering using K Means Algorithm Mrs Sanjivani Tushar Deokar Assistant professor.