Clustering tweets by similarity

Sergio Tobal and Gonzalo Pierola

Abstract

In this document we report our proposal for the application of clustering tweets. For this we used some new alternatives like TF-IDF and Word2vec, and some old, like K-Means. Most of algorithms' implementations are from Scikit Learn, but we used other libraries, like pandas and nltk to preprocessing. At the end we used the Silhouette coefficient to interpret and validate the clusters.

1 Description of the problem

The objective of this project is the classification of tweets about a concrete subject, like for example Machine Learning. This tweets are going to be classified according to the similarity of the topics covered in each one, so each group will contain the tweets corresponding to a subgroup of the chosen subject, for example, in the case of the tweets of Machine learning the groups should correspond to different areas of Machine Learning.

To perform this process we have used a clustering algorithm, in this way we generated clusters showing the distribution of the tweets depending of the topics that are mentioned on it.

For this project we have used two datasets, the first one with 100 tweets about Machine Learning obtained using the python pattern[1] library, and the second one which were 77000 tweets about the Web Summit of 2015, but we took only 500 tweets for speedup the process. In the last one we have the full text of each tweet, so the information obtained from this will be more reliable.

2 Description of our approach

This are the task that we followed for the implementation:

1. Obtain the dataset of tweets about Machine learning with pattern.
2. Preprocess tweets to remove all sections that do not give us relevant information.
3. Apply the Word2vec transformation to the tweet word to obtain the vector representation of them.
4. Use the Tf-idf feature extraction technique to obtain the one hot vector representation of each tweet.
5. Mini Batch K-Means clustering method from sklearn to generate the cluster of the tweets.
6. T-sne to apply dimensionality reduction to the one hot vectors of the tweets in order to obtain the 2D representation a visualize them.
7. Evaluate the clusters from their silhouette metric.

2.1 Creating datasets

We took the recommendation of Roberto Santana of using the Pattern library to get tweets about some topic, but the library didn't work so well, sometimes it didn't return all the tweets we asked or there was a lot of tweets repeated.

At the end we only created a dataset of 100 tweets with this approach, thinking in using it only to test the algorithms quickly.

As we thought, this didn't convince us because it didn't yield some useful clusters, so instead of fighting against Pattern we took a dataset of 77000 tweets about the Web Summit of 2015.

With both datasets we apply some degree of data processing.

2.2 Preprocessing

The first step for the preprocessing it will be to clear the tweets. The text that we obtain from Twitter have some useless information, for example the URLs that appear in a tweet, or the user names of Twitter accounts. So the first step it will be to clean all this information using regular expressions.

Once that we have a clear representation of the tweet data, we need to represent that text information in a numeric way for doing the cluster. We decide to represent each tweet with a "one hot vector" which is a binary vector that represents a list of topics, and for each tweet this vector will sow which one of this topics appears on it.

2.3 Vector representation with Word2vec

In a first instance we thought in using a word representation method because we knew about it and since we were working with text, was the best way we could think of working with tweets.

This method works using 2 algorithms called Continuous Bag of Words (CBOW) or Skip-gram, they are really the opposite, in CBOW you predict a word looking the words around it, and in Skip-gram you receive a word and try to predict the words around. With this you get a number representation of all the words we call word embeddings, which are a finite dimension vector, the longer the length more accurate are the representation of the words, even though they will take up more disk space and memory space.

So, we used a pretrained Word2vec model, trained in Google News, and after tokenize the phrases of the tweets we create a new column in the pandas dataframe with the words that had a representation in Word2vec.

We also tested with a Glove model, trained in 7 billion of tweets, but even though it may seem contradictory, it gave us worse results, besides that it took longer to load in memory, so we discarded the idea.

2.4 Vector representation with TF - IDF

Looking in Internet to other similar ways to approach this way, we found something useful call "Term frequency - Inverse document frequency". The "term frequency" refers to how frequently a term occurs in a document, understanding document as a single tweet, and the "Inverse document frequency" refers to the importance of the term, because some words, like 'is', 'of', 'the' are too common, so we need to weigh down the frequent terms while we scale up the rare ones.

And the implementation of Scikit Learn, combine this algorithm with CountVectorizer, which is nothing more than a One-hot vector, a vector of 0s and 1s, in which the 0 stands for that word has not appeared, and the 1 stands for that word has appeared.

2.5 Mini-Batch K-Means to learn the labels

The main problem we had it was to group the tweets in cluster, and since we didn't had the true labels we had to use Mini-Batch K-Means to find the best number of clusters.

This algorithm is similar to K-Means, it gives a slightly worse results but it reduces the computation time, and we start using it because of the big dataset of 77000 tweets and later we stick with it for all datasets.

In short the algorithm tries to separate samples in groups with equal variance, minimizing some metric. Sometimes the people call this algorithm Lloyd's algorithm, and it has 3 steps, the first one we choose the initial centroids, and then we loop the other 2 steps, consisting of creating new centroids and evaluating this new centroids with the old ones until the difference is less than a threshold, in other words, repeat until the centroids do not move much.

2.6 Dimensionality reduction with T-SNE

The problem with Word2vec it was we had a matrix with the number of rows as the number of tweets in the dataset, and the number of columns as the dimensions in the word embeddings. And the same problem with the TF-IDF approach, we had matrices with number of tweets by number of words.

And we couldn't draw this in a 2D space, so we had to use an algorithm of dimensionality reduction, which in a nutshell converts the input into a probability distribution. Then it takes 2-dimensional (because we wanted 2 dimensions) points and converts to another probability distribution.

At the end we had matrices with the same rows but only 2 columns without losing the relative similarity of the higher dimension space. We used this to draw the clusters and calculate the metric for evaluation.

2.7 Evaluation of clusters

For the evaluation of the clusters we used the silhouette metric, which validates the consistency of a cluster of data. This method provides a graphical representation of how well each object it matches with the cluster in which it is.

The silhouette value ranges from -1 to 1, where a high value represents that the objects matches better with them cluster. If most objects have a high value the cluster configuration will be correct. If many of the objects have low or negative values means that you have too many or too few clusters.

3 Main Algorithms

For the sake of simplicity the unspecified parameters are the default values of the libraries.

1. Word2vec [2]
   - Field = Google News
   - Dimensions = 300
   - Training algorithm = Continuous Bag Of Words
   - Negative = 5
   - Sorted_vocab = 0
2. TF-IDF (with CountVectorizer inside) [3]
   - Min_df = 4
   - Max_features = 5000
   - Ngram_range = (1, 1)
3. T-SNE [4]
   - Learning_rate=200
   - Perplexity=50
   - Random_state=10

4. MiniBatchKMeans [5]

   - Init = 'k-means++'
   - N_clusters = 3
   - N_init = 1
   - Init_size = 1000
   - Batch_size = 1000
   - Verbose = False
   - Max_iter = 1000

4 Implementation

All the project steps were implemented in Python. For getting the the dataset we had to use Python 2 because of the Pattern library. But for the rest of the tasks we used Python 3, pandas and numpy for reading and prepocessing the dataset, and scikit-learn for the classification tasks. We separate the implementation in several Jupyter Notebooks.

`1. gettingDataset.ipynb`. Using Pattern library and some regex to clear most we can when we recollect data.

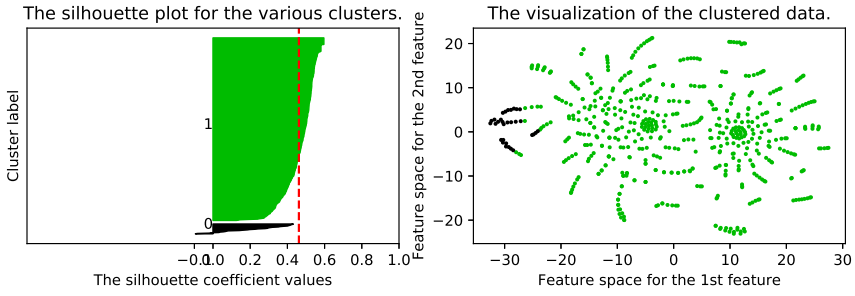
`2. Preprocessing.ipynb`. There are 2 files, one for Word2vec and the other are for the 77000 tweets which are in a JSON file.

`3. Dimensionality Reduction.ipynb`. At the beginning we created a lot of notebooks, one for each file in the prepocessing step, but at the end we only keep one for TF-IDF, and other for using Word2vec.

5 Results

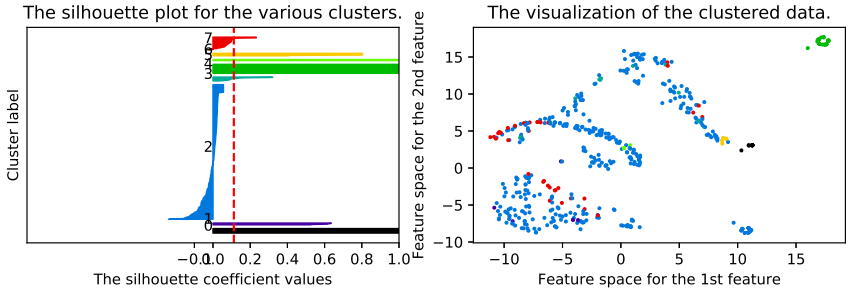
We obtained several clusters each one corresponding to one of the test that we have done, and also the graph representing the value of the silhouette given one of the clusters. For the tests we have used two different datasets, and also two different ways of obtaining the numeric representation of the text of the tweets, so we have four different groups of results. For each group we have executed the Mini-Batch K-Means clustering method for different k values, so for each of one them we have a different result. In this section we are going to show the results of the best K value for each one of the four groups.

100 tweets from Machine Learning dataset + Word2vec representation (K=2):



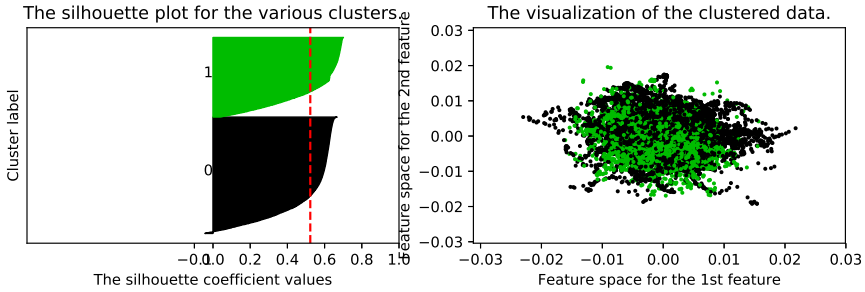
The silhouette value for this cluster if of 0.46, and we obtained two clusters. From the silhouette graphic we can see that there are no negative values, meaning that the samples are assigned to a right cluster, but we can see there are a large number of tweets in cluster 1, so we think this means is overfitting, because is not good to have a silhouette this big if most of the tweets are in one cluster.

100 tweets from Machine Learning dataset + tf-idf representation (K=8):



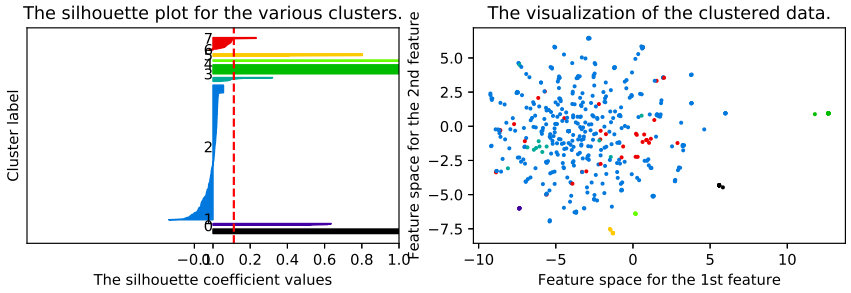
In this case we have a silhouette value of 0.11 with 8 different clusters. In this case there are some negative values, this mean that some values are wrong classified, but most of the samples are assigned to a right cluster. On the other hand, the silhouette values is near to 0, and that mean there are overlapping clusters.

500 tweets from Web Summit 2015 dataset + Word2vec representation (K=2):



The average silhouette value of this clusters is 0.52 which is the best score that we have obtained, and the number of clusters is 2. Like in the first one all the values are positive so all the all the samples are assigned to a right cluster. But still annoying us they are one on top of another.

500 tweets from Web Summit 2015 dataset + tf-idf representation (K=8):



The last cluster haves a silhouette value of 0.11 and the samples are divided in to 8 different clusters. Like in the second case from some of the samples the silhouette value is negative, that means that this samples are assigned to a wrong cluster. An also as in the second cluster the silhouette value is near the 0, which indicates a overlapping cluster.

We had more figures but we thought it was going to be too many for the paper, so we include it in the github repository.[6]

6 Conclusions

In this project we have applied methods of clusterings, dimensionaly reduction, word embeddings, one hot vectors, prepocessing, in which we agrupate tokenization, regex and some other methods of cleaning documents.

We have seen is not easy to solve this problems, is not like web development where you can find most of the information in StackOverflow or in tutorials. For this is difficult to see what needs to be done in some tasks, and there are not much information, because most which we find was too arid to understand it.

And drawing the cluster we have discover how to draw the average of the silhouette metric and how to measure the accuracy of clusters, and how we can see the overfitting more clearly when you have few clusters.

References

[1] Daelemans W. Smedt, T. Pattern. https://github.com/clips/pattern, 2012.

[2] Gensim. Word2vec implementation. https://radimrehurek.com/gensim/models/word2vec.html.

[3] Scikit Learn. Term frequency inverse document frequency. http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction.

[4] Scikit Learn. Word2vec implementation. https://radimrehurek.com/gensim/models/word2vec.html.

[5] Scikit Learn. Mini-batch k-means clustering. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html.

[6] Github. Repository of the project. https://github.com/set92/mlnn_class2017/tree/master/P1%20-%20self-organizing%20maps.

[7] kylemcdonald. Coloring t-sne. https://github.com/kylemcdonald/Coloring-t-SNE, 2017.