BIG DATA COMPUTING
Text Clustering

Alessandra Monaco

23-12-2020

Contents

1 Introduction

Dealing with Big Data requires searching for alternative solutions that improve time and space efficiency but still preserving effectiveness. In this notebook, that can be found in `BD_HW2.ipynb` file, we experimented different solutions for clustering.
A standard Kmeans algorithm on our dataset of 314808 documents takes more or less one hour of computations. To overcome this efficiency problem, we can either use a **MiniBatchKmeans**, or somehow reduce the size of the dataset, considering **less data** (e.g. sampling), or **less features** (e.g dimensionality reduction).
Possible solutions that we took into account are:

- **HashingVectorizer** instead of TfIdfVectorizer, to speed up the time required for the vectorization of documents;
- reducing the number of features used for clustering, with a lot of **preprocessing and normalization** (stop words removal, noise removal, stemming AND lemmatization...), or setting **min_df** of the vectorizer (such that only words appearing in at most a predefined number of docs are considered in the vocabulary), and also **Dimensionality Reduction** techniques, such as **Truncated SVD**;
- reducing the number of documents using a **sample** of the dataset, to speed up the preprocessing and clustering time, and reducing the space required;
- using **MiniBatch Kmeans** algorithm with the most suitable batch size.

In the next sections we will describe our design choices and results in detail. Results are evaluated basically using the **accuracy** metric (since we had the real cluster labels).

2 Preprocessing and featurization

Preprocessing is generally one of the most important steps to achieve good results. We defined the function `preprocessing(doc)` that uses tools provided by `nltk`, performing the following steps:

- word tokenization;
- upper case removal;
- punctuations and digits removal;
- english and spanish stop words removal (we found out that there were also few spanish documents);
- stemming with Snowball Stemmer algorithm;
- lemmatization with WordNetLemmatizer algorithm.

We tried to preprocess both with and without lemmatization, to inspect its effect on the results.
After that, we tried different featurizations: `HashingVectorizer` that just considers the frequency of terms, `TfidfVectorizer` and `TfidfVectorizer(min_df=10)`.
Preprocessing the whole dataset required some time (about 10 minutes, still affordable), while the vectorizer required about 10 seconds.

3 Clustering using Sampling and Dimensionality Reduction

As a first trial, we tried to **randomly sample** the dataset, keeping about the 10% of the original one and ending up with 31480 documents.
Doing a random sampling has pros and cons: we can significantly reduce the time needed for the clustering and also for the preprocessing (from 10 minutes to 70 seconds), but, since it is *random*, we can not really be sure that the sampled dataset is representative enough for the population.
After doing preprocessing and featurization (with `TfidfVectorizer`), we ended up with 20287 features. We reduced the number of these features with **Truncated SVD**. The number of singluar values to keep (parameter `k`) was chosen empirically: we started with 2, that is the number of topics that we expected to have, and incrementally increased it up to 8, that was able to give us the best accuracy.

Cluster sizes:

```
|Cluster  0 |:  19358
|Cluster  1 |:  12122
```

The 20 most relevant terms of the created clusters:

```
Cluster 0:  use babi one great like love would get work well easi month littl time realli product fit
            bottl old put
Cluster 1:  dog cat toy love one food like treat get product would use chew great good work time eat
            play realli
```

Evaluation results:

```
Accuracy: 0.839
Homogeneity: 0.405
Completeness: 0.422
V-measure: 0.413
Adjusted Rand-Index: 0.460
Silhouette Coefficient: 0.206
```

4 Clustering with Mini Batch Kmeans

As second attempt, we tried to consider the entire dataset and cluster using **Mini Batch Kmeans** algorithm.
Mini Batch K-means algorithm's main idea is to use small random batches of data of a fixed size (batch_size), so they can be stored in memory. At each iteration a new random sample from the dataset is obtained and used to update the clusters and this is repeated until convergence. The effect of the randomness of the batches is visible from the fact that we obtain different accuracies and clusterings if we run the same algorithm many times, because they depend on the `random_state`.
Initially, we run the algorithm many times with different batch sizes, without specifying any random state: most of the time the accuracy was about 0.8, sometimes about 0.7, and very few times 0.9. In the end, we were able to find (empirically) values to achieve the highest accuracy: with `random_state=123456` and `batch_size=10000` we obtained the following results[1]:

Cluster sizes:

```
|Cluster  0 |:  153486
|Cluster  1 |:  161322
```

20 most relevant terms of the clusters:

```
Cluster 0:  babi use love one great like easi month bottl seat would get littl fit diaper well old
            realli daughter work
Cluster 1:  dog cat love one toy like food get use product work great good would treat well time chew
            realli seem
```

Evaluation results:

```
Accuracy: 0.913
Homogeneity: 0.575
Completeness: 0.575
V-measure: 0.575
Adjusted Rand-Index: 0.683
Silhouette Coefficient: 0.005
```

5 Clustering using Dimensionality Reduction

As last attempt, we combined **Truncated SVD** (on the whole dataset) with the standard **Kmeans** algorithm, that is now able to run in "reasonable" time since the dimensionality of the data is reduced just to few singular values.
The preprocessing and the featurization are the same used for the MiniBatch Kmeans algorithm; we reduced the dimensionality from 12473 to 3 (keeping 3 singular values), obtaining the following results:

Cluster sizes:

```
|Cluster  0 |:  193518
|Cluster  1 |:  121290
```

20 most relevant terms of the clusters:

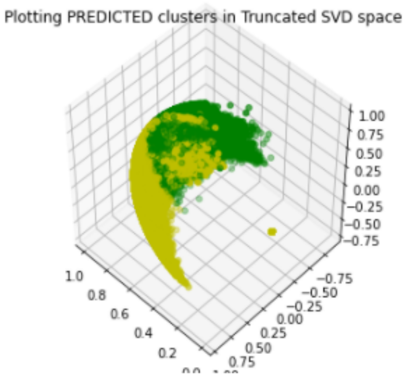
```
Cluster 0:  use babi one great like love get would seat easi work bottl month well littl time realli
            fit diaper old
Cluster 1:  dog cat love toy food like one treat get would product good chew use great eat play time
            realli work
```

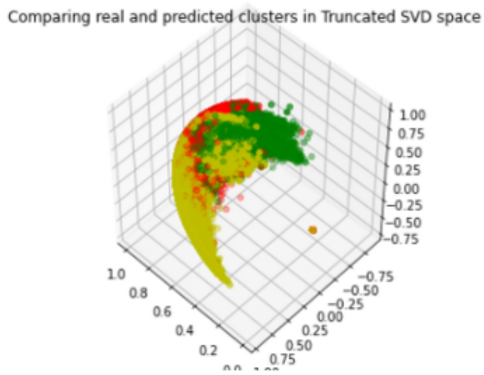
Evaluation results:

```
Accuracy: 0.839
Homogeneity: 0.404
Completeness: 0.420
V-measure: 0.412
Adjusted Rand-Index: 0.461
Silhouette Coefficient: 0.461
```

[1]the results refer to the featurization with TfidfVectorizer(min_df=10)

Differently than before, our Cluster 0 is now bigger than Cluster 1, and the accuracy is lower, so the algorithm is making some mistake in assigning some documents to the clusters. This is also visible by plotting the clusters in the dimensionality defined by the singular values:





The green points belong to cluster 0, the yellow points belong to cluster 1 and the red points are the wrongly predicted. This probably happens because the clusters are not well separated and distinct: the Silhouette score, in fact, is not really high (0.4).

6 Conclusions

Considering the accuracy metric, the best model found so far is obtained with the "complete" preprocessing (both stemming and lemmatization, english and spanish stop words removal), Tf-idf featurization with minimum document frequency of 10, MiniBatch Kmeans algorithm with batches of 10.000 docs and a certain random state (123456), discovered empirically.
Using this model and its vectorizer, we plotted the most relevant terms in a **Word Cloud** fashion, with a pre-attemptive visualization that reflects the importance of words in terms of tf-idf scores, to interpret the clusters. The most relevant terms are discovered through the information contained in the centroids of the clusters.
The plots of the Word Clouds are shown below:

Inspecting the visualization, we can guess the topics of the two clusters. Our corpus seems to be about products reviews. The first cluster regards a product for babies, and this is proved by the presence of terms like *babi* (that is also the most relevant word of the cluster), but also *daughter*, *diaper*, *kid*, *child*. We can guess that we are talking about a product (*use*, *product*, *bought*, *buy*, *item*) that seems to be mostly recommended (*love*, *recommend*, *great*, *like*, *good*, *perfect*, *well*, *nice*, *comfort*, *super*); we can not be sure that each of these words was referred to the product itself in the reviews, but the presence of only positive terms in the word cloud should at least reflect a general positive feeling about the product. Without inspecting further the clusters and the documents, this visualization is not enough to understand exactly what is the product (or products?), but we can guess that it may be related to a *seat* for the *car*, or a toy (*toy*, *play*), maybe in *plastic*, or that fits in a bag (*fit*, *bag*), that it is *small*, easy to be used (*easi*, *easili*) and comfortable (*comfort*). Another relevant term is *one*; it may be related to the age of the baby for whome the product is for, considering also the presence of the terms *month*, *year*, *two* and *old*.

The second cluster is about reviews of a product for dogs or cats (*dog*, *cat*, *kitten puppi*, *pet*). Again, it is not clear what kind of product (or products) it is, but it may be related to some *food* (*eat*, *flavor*, *smell*, *chew*, *bowl*, *teeth*) or to a *toy* (maybe a *ball*?) for pets. The terms *help*, *treat* and *problem* may suggest that the product is used (*use*, *give*) to treat some problematic behaviour in pets, probably related to eating and food.
The reviews also talk about the *price* of this product, but from the visualization we are not able to understand whether it is high or affordable. The general feeling about the product seems to be positive (*love*, *like*, *great*, *well*); notice also that the terms *good* and *product* have the same size, meaning that their importance scores are similar so it is possibile that they are used together to state that the product is good.

Lastly, we conclude with some technical comments regarding the experimentation:

- Using also **lemmatization** after **stemming** improves the results (just a bit), mostly for the MiniBatch algorithm, while for the SVD model the kind of preprocessing[2] did not really impact the effectiveness.
- **HashingVectorizer** produces worse results than **TfidfVectorizer** in terms of accuracy with all the models. For instance, using the MiniBatch algorithm with HashingVectorizer we obtained an average of 0.6965 $\pm$ 0.01 accuracy with a batch of size 10000 and 0.6767 $\pm$ 0.06 with a batch of size 1000. With TfidfVecorizer we obtained an average of 0.8175 $\pm$ 0.08 for the first batch size, and 0.7674 $\pm$ 0.08 for the second one. The improvement is visible also doing dimensionality reduction: 0.726 accuracy with HashingVectorizer and 0.839 with TfidfVectorizer. This happens because the tf-idf is a more precise representation of the document, since it considers also the inverse document frequency, while the HashingVectorizer considers only the frequency of words.
- The **min_df** parameter set to 10 gave us the possibility to work with less features, without loosing effectiveness (no significant differences in accuracies were observed): starting from 64421 features we were able to work with just 12473 terms.
- The **dimensionality reduction** produces more dinstinct clusters, with a higher silhouette score; this is probably correlated to the curse of dimensionality and to the fact that in a high dimensional space everything seems to be almost equi-distant.
- The best solution to apply dimensionality reduction to our dataset is to use **TruncatedSVD**, since it supports very large sparse matrices, while PCA explicitely requires centering the data and therefore the dense representation of the dataset; converting the sparse matrix to a dense one causes a memory crash, even with the sampled dataset.
- The results produced with sampling seem to well approximate the results on the original dataset; most of the relevant terms are indeed the same, therefore our sampling was representative enough for the population, even if it was random.
- Regarding **computational time**, we were able to cluster in very few seconds (2.34 both with dimensionality reduction and minibatch kmeans using a batch of 10000, 0.57 seconds on the sample set); preprocessing took some times anyway, because it's quite heavy and it has to inspect every document.

[2]lemmatization or not, min_df or not