1 Intro & Design choices

1.1 Sampling

We may decide to apply sampling to possibly improve performances and reduce memory and CPU load. In order to apply this step, you only need exchange the corresponding boxes inside the code.

1.2 Preprocessing

We have four main steps:

1. **Regular Expression (re)** ⇒ To remove anything that is not a letter, such as punctuation, numbers and so on. They are not useful to identify the topic;
2. **Lowering** ⇒ Essential part to normalize terms;
3. **Lemmatization** ⇒ I have made one version that performs a complete lemmatization, identifying the correct *POS tag* for each word, and a simpler (and faster) version that only finds the root for verbs (more likely to have inflected forms);
4. **TfidfVectorizer** ⇒ To convert the collection of raw documents to a matrix of *TF-IDF* features. *stop_words* parameter included and other parameters found by heuristic.

1.3 Clustering

Three different attempts (six if we consider also the sampling version):

1. **Only MiniBatchKMeans** ⇒ Simple clustering using only a variant of the KMeans algorithm, which uses mini-batches to reduce the computation time[1];
2. **SVD + MiniBatchKMeans** ⇒ Apply SVD first to reduce the dimensionality to the number of topics[2]. Then, MiniBatchKMeans to find the clousters as before;
3. **SVD + KMeans** ⇒ As before but with standard Kmeans++;

1.4 WordCloud

We have a WordCloud for each attempt and for each cluster to identify the most important terms for each of the two topics, reflecting their relative importance.

2 Comments on results

Let's start considering the sampling step, which is performed taking in consideration one line each n, where n is a number set at the beginning of the code. Indeed, this method may help to maintain the dataset's homogeneity, namely the same ratio for both classes.

This action may be useful in a very large dataset, considering also that Kmeans is prone to give better results with fewer data points (reducing also the fluctuations in the accuracy). However, in this case, I did not find enough relevant difference in the two

[1] Not really required in this case using Colab.

[2] We may also try with a different number of components, but this may lead to more fluctuations in the clustering accuracy, without bringing real improvements.

approaches, and thus there is no real reason to risk generality reductions. Furthermore, using Colab, we do not encounter any memory problem with both Kmeans and SVD.

Now, let's also consider another fact: the final result of Kmeans depends a lot on the initialization, and thus the results may vary at each run[3]. However, with a good preprocessing and parameters tuning, it is possible to stabilize "enough" the outcome.

- **Only MiniBatchKMeans** ⇒ For the accuracy, we have values that generally goes around 78% and 84%, that is more than reasonable considering the expected 50% accuracy for two classes clustering.

  The WordClouds already gives good results. Making a guess:

  - *Topic 0* ⇒ something about animals
  - *Topic 1* ⇒ something about being a father, work and so on

- **SVD + MiniBatchKMeans** ⇒ As expected, using SVD to reduce the dimensionality to the number of topics, generally increases the classification accuracy and decreases the fluctuations. We may state that, on average, it increases by 2-4 points.

  For the WordClouds we have similar results.

- **SVD + KMeans** ⇒ Test made for comparison. However, for this case, there are no relevant differences with the previous variant.

For some more insights directly refer to the code.

[3]We may work on the initialization state and increase the number of random tried to improve the performances slightly.