Fast Text based Clustering

Name: Sampad Kumar Kar
Shankar Ram Vasudevan

Summary

- We look at various techniques of clustering while using the ***Jaccard Similarity*** as a measure of closeness.
- We first used the built-in `KMeans` by processing the documents as binary vectors with dimension equal to the no. of words in the vocabulary. We used this for *KOS* and *NIPS*
- Since, the above techniques turned out to be too slow or unreliable (the `MiniBatchKMeans`) for *Enron*, we created two algorithms from scratch to tackle while considering the ***Jaccard Distance***. This turned out to be more efficient and accurate with better scores and clusters.

We finally decide on the following results based on our observation:
- *KOS*: `n_clusters = 2`
- *NIPS*: `n_clusters = 3`
- *Enron*: `n_clusters = 4`

Strategy 1

We will be treating the data corresponding to each document as a binary W dimensional vector, where W is the no. of words in the vocabulary of the dataset (the notation being obvious, we ignore the frequency counts, and presence of a word in a document gives the corresponding feature value of `1` and `0` otherwise).

Strategy 1.1

We find the centroid points in each cluster. We do this by taking the argmin of the euclidean distance of each of the data points in a cluster from the cluster centroid as found by the `KMeans` methods. We do so in order to evaluate the Jaccard Score of each cluster w.r.t. this data point and we can't do so if our centroid contains floating point values.

Strategy 1.2

We tweak the `closestocentroid` method based on the thresholds. Now, instead of using the argmin method (with Euclidean Distance) and defining a particular data point in the cluster itself as a centroid, we instead use threshold value to approximate the centroid as computed by the `KMeans` method. So, if for a certain attribute `value > threshold`, we give that particular attribute `1` and `0` otherwise. This also creates a binary vector and we can move on to calculate the jaccard score of a particular cluster w.r.t. this centroid. We treat this `threshold` as a hyperparameter which we tweak while trying to improve the overall jaccard score.

Strategy 2

We will be treating the data corresponding to each document as a set of words i.e., if a word is present in that document, we will add the `wordID` of that particular word to the set corresponding to that document. This is a more efficient way to process the data, instead of the sparse binary representation we were using before.

Strategy 2.1

In this particular strategy, we define our own `jaccardDistance` function, because of the change to a sparse set of word representation.

In the `KMeans_Pro` method, we first set the early stopping criterion, and while that is fulfilled, we continue to update our clusters. The key to this algorithm is that we store our data points in sets (clusters are represented as sets) and we update these sets everytime based on the average jaccard distance of a data point from a set to a cluster (we use the jaccard matrix to calculate the average jaccard distance from the data point in interest to all the points in that particular cluster and assign the cluster with minimum average distance to the data point). We iterate through the dataset until the cluster stabilises.

Strategy 2.2

In this strategy, we again use the same sparse (set of words) representation to optimise the storage and processing of the data points (which is again what makes this algorithm faster compared to the ones in **Strategy 1**).

In this algorithm we again use thresholding to come up with an appropriate centre w.r.t. a particular cluster (which is used to assign a cluster to any example based on the jaccard distance from these centres, like normal `KMeans`), which we update after each iteration over the whole data set.

The question that remains is *how we do this*. Remember that each data point in our dataset is a set of words (`wordID`s). So, we just use thresholding to determine whether a particular word in the vocabulary deserves to be in the centre of this particular cluster. For this we calculate

$$\alpha = \frac{\text{no. of documents in this cluster in which this word is present}}{\text{total documents in this cluster}}$$

for each word. If $\alpha > \text{threshold}$, we add the `wordID` corresponding to this word to the set of words corresponding to the centre we are constructing (and we skip the word otherwise).