Text Clustering

Kevin Chuang, 012480052

Overview

For this project, I implemented the DBSCAN (density-based spatial clustering of applications with noise) clustering algorithm from scratch to cluster text data (news records). DBSCAN is an unsupervised clustering algorithm that is density-based, meaning the algorithm will group together points that are closely packed together, and mark points in low-densities as outliers. The input data consists of 8580 text records in document-term sparse matrix (CSR) format with no labels provided. For evaluation purposes (leaderboard ranking), the Normalized Mutual Information Score (NMI), an external index metric for evaluating clustering solutions, will be the metric used for scoring the clustering algorithm.

Rank & NMI

At the time of writing this report, my current rank on CLP public leaderboard is **2nd** with a NMI score of **0.6238**.

NMI is an external metric (meaning it needs labels or external information) for evaluating different clusterings. It has a score between 0 (no mutual information) and 1 (perfect correlation). The equation of NMI is shown below, where **Y = class labels, C = cluster labels, H(.) = Entropy, and I(Y;C) = Mutual Information between Y and C**.

$$NMI(Y, C) = \frac{2 \times I(Y; C)}{[H(Y) + H(C)]}$$

Data Preprocessing (Feature Selection & Dimensionality Reduction)

Since the input data was in document-term sparse matrix format, the input data was specially parsed using custom functions, and converted to a CSR sparse matrix data structure with **inds**, **vals**, and **ptrs** provided. The input data was provided in a bag-of-words format with word index and word frequency in the document as pairs (word index, word frequency).

Then, I implemented a custom function to remove words that did not show up in any documents or very few documents (<1-5 documents), and showed up to all documents or most documents (95%> of all documents). Essentially, what this does is remove very frequent functional terms (i.e. the, a, at,) and very infrequent terms. These infrequent and frequent terms usually do not have much discriminative power, thus they are not helpful in determining whether a document belongs in a cluster. In addition, this is done to further clean the data and reduce the dimensionality of features (or words) that do not really contribute more information for the task at hand and avoid the Curse of Dimensionality. However, after experimentation, I realized that this did not improve the clustering results. This may be due to the dimensionality reduction technique mentioned below that

already accounts for the unimportant features, thus I did not use this in the final prediction. In addition, this may be because tf-idf (explained below) already weights the unimportant features appropriately to account for this.

Next, I scaled the input data using tf-idf (term frequency–inverse document frequency) to reflect how important a word is to a document in a corpus by scaling down the impact of tokens that occur very frequently in a given corpus, and thus less informative. Then, the term vectors are normalized using the L2-norm, which is important, since I will be using the cosine distance proximity measure.

Because of the sparsity of the data and the Curse of Dimensionality, I used a linear dimensionality reduction technique called truncated singular value decomposition (SVD) to further reduce the number of features (mostly sparse features) in the dataset. This essentially reduces down the features to a smaller set that still contain most of the information. When truncated SVD is applied to term-document matrices (such as bag-of-words approach or tf-idf approach), this transformation is known as latent semantic analysis (LSA), because it transforms such matrices to a "semantic" space of low dimensionality. LSA is good for combining words with multiple meanings (synonyms, polysemy) together. These words cause term-document matrices to be overly sparse, and exhibit poor similarity under measures such as cosine similarity. I experimented and validated with different dimensionalities (values between [2, 100]), and determined the optimal number of components or output features to be 4. Experiments with other unsupervised linear and nonlinear dimensionality reduction techniques were conducted including PCA, LLE, t-SNE, and ICA, but TruncatedSVD seemed to perform the best.

DBSCAN Overview and Pseudocode

For my implementation of the DBSCAN clustering algorithm, I utilized cosine distance as the proximity measure for determining a point's nearest neighbors. I loop through each point, and find its nearest neighbors based on a given epsilon. If the number of neighbors is below the MinPts parameter, then it is marked as a "noise" point. However, this "noise" point does have the potential to change to a border point. If the number of neighbors is equal to or above the MinPts parameter, then this point is labeled as a core point.

This leads to the cluster generation for that core point. The cluster generation is similar to a first in first out (FIFO) queue, where it loops through a core point's nearest neighbors. In this stage, previously classified "noise" points can be converted to border points and thus assigned to a cluster if they are neighbors to a core point. In addition, if the neighbor has not been classified to a specific cluster id or as a noise point, they can be claimed by the core point, and the neighbors for that specific neighbor are generated; adding to the cluster generation queue.

Lastly, since the noise points must also be assigned to a cluster, I experimented with different ways of assigning the noise points including assigning noise points to closest border point, closest core point, closest K-nearest neighbors, and closest centroid. Using an internal measures to tune and validate, I determined that assigning noise points to the closest centroid (with recalculation of centroids) performed the best.

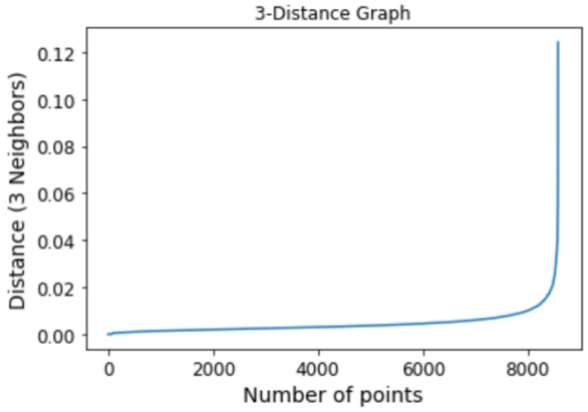
The pseudocode of the DBSCAN algorithm is shown below.

DBSCAN Algorithm:

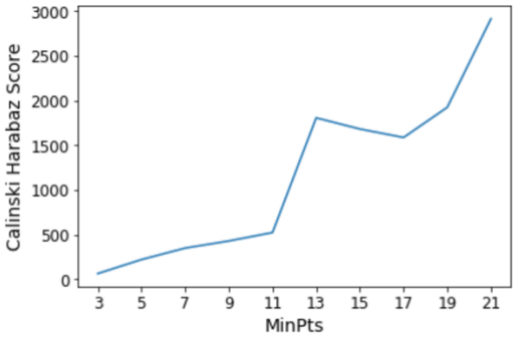
```
for each point in data:
        if label of point is defined, continue
        Find all neighbors of point given radius epsilon
        if number of neighbors < MinPts, then
          Label point as noise (can change)
        else
          Label point as core point
          Generate clusters for core point by
            for each neighbor of core point
              if neighbor was labeled as noise, then
                    Assign this point to current cluster (border point)
                  elif neighbor was not labeled, then
                    Assign point to current cluster
                    Find the neighbors of this point
                    if number of these neighbors < MinPts, then
                      Append neighbors to current queue of neighbors
```

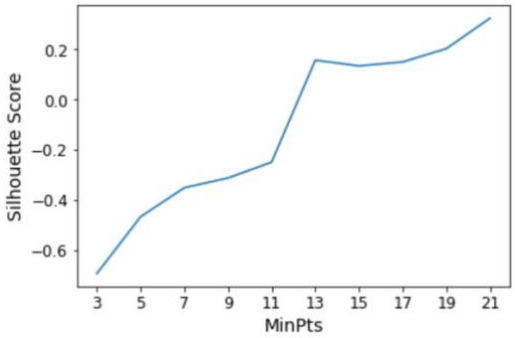
Determination of Eps and MinPts

First, I determined a range for the radius Eps by plotting a k-distance graph using a nearest neighbor algorithm and K = 9 nearest neighbors. The k-distance graph plots all the sorted distances of each point to its K nearest neighbors. The idea is that for points in a cluster, their K nearest neighbors are roughly at the same distance, and for noise points, their K nearest neighbors are at a farther distance. By plotting the distances of every point using the elbow k-distance graph, I visually choose a range for Eps containing where the sharp curve changes (the elbow of the line). The elbow k-distance graph (where k = 3) is shown below.



With the range of Eps determined using the k-distance graph, I use MinPts varying from 3 to 75 in steps of 2 to further tune the radius Eps parameter by using internal evaluation metrics (Silhouette, Calinski-Harabaz). More emphasis was placed on the Calinski-Harabaz score, which is explained more in detail below. A plot of the Calinski-Harabaz score (y-axis) and the MinPts (x-axis) between 3 to 21 is shown below. In addition, another plot of Silhouette score is also shown. The given optimal epsilon is around 0.00319.





Internal Evaluation Metric

Internal index is used to measure the goodness of a clustering structure without respect to external information, such as class labels. I experimented with two different internal measures of cluster validity: **Silhouette coefficient** and **Calinski-Harabaz score.** In the end I decided to move forward with the **Calinski-Harabaz** score, because it was a better indicator of performance and seemed more correlated with **NMI** results compared to the **Silhouette coefficient.** For both metrics, the score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

Silhouette coefficient – Silhouette combines cluster cohesion with cluster separation for individual points. The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around 0 indicate overlapping clusters.

The Silhouette coefficient equation is shown below, where **a** = mean distance between a sample and all other points in the same class, and **b** = mean distance between a sample and all other points in the *next nearest cluster*.

$$s = \frac{b - a}{max(a, b)}$$

Calinski-Harabaz score – Also known as Variance Ratio Criterion, this internal evaluation metric is defined as the ratio between the within-cluster dispersion and the between-cluster dispersion. Thus, this means that a higher Calinski-Harabaz score relates to a model with better defined clusters.

The Calinski-Harabaz score **s (k)** is calculated as a ratio of the between cluster variation (**Bk)** and within cluster variation (**Wk**) scaled by **N** = number of data points and **k** = # clusters.

$$s(k) = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{N - k}{k - 1}$$