Ankai Lou
CSE 5243 - Data Mining
Instructor: S. Parthasarthy

Report # 4 - Clustering Data

Group Members & Contributions

Ankai Lou

- Wrote entire codebase for preprocessing raw sgml input into processable tokens.
- Wrote entire codebase for generating feature vectors using td-idf and normalization.
- Added functionality for paring down original feature vector to 10% size.
- Added functionality for clustering feature vectors using K-means mean-based clustering.
- Added functionality for clustering feature vectors using DBSCAN density-based clustering.
- Added documentation for entire codebase & README
- Added API for adding new clustering algorithms and metrics.

Problem Statement

Cluster feature vector datasets representing Reuters article database. Generate two feature vector datasets of similar content but differing dimensionality. Implement two clustering algorithms (K-means & DBSCAN) and apply to the datasets. Test the scalability and cluster quality (entropy & variance) of the 2x2 set of experiments.

Proposed Solution

To generate the original feature vector dataset:

- Remove stopwords and terms found in topic/place class labels.
- Lemmatize the remaining words using the built-in NLTK Lemmatizer.
- Stem the remaining word using the built-in NLTK Porter Stemmer.
- Remove sufficiently short stems (less than 5 characters).
- Compute the TF-IDF score for each document-word pair using the built-in NLTK TF-IDF module.
- Select the top 5 words from each document as features to be used in the feature vector.
- Remove duplicate terms from the feature set (performed implicitly in Python).

To generate the pared-down feature vector:

- Begin with the set of terms generated for the original feature vector.
- Select the top 10% of words in the feature set by averaging the TF-IDF score across all documents.
- Use the features selected by average TF-IDF score as the feature set for the pared feature vectors.

Implement K-Means Centroid-Based Clustering:

- Use the built-in Scikit-Learn K-Means clustering module in sklearn.cluster.KMEANS.
- Initialize the clustering algorithm as 'kmeans++' to pseudo-randomly generate optimal initial centroids.
- Set the number of clusters (initial centroids) to $n = 10$ and the maximum number of iterations to $m = 10$.
- Set the default distance metric to use euclidean distance (no others were available).

Implement DBSCAN Density-Based Clustering:

- Use the built-in Scikit-Learn DBSCAN clustering module in sklearn.cluster.DBSCAN.
- Set the default epsilon to $\epsilon = 0.3$ and min points to $min\_pts = 10$ (subject to change).
- Set the default distance metric to euclidean distance.

To compute entropy of a clustering:

- Set initial value as `entropy = 0.0`
- For each cluster, set temporary counter as `temp = 0.0`
- For each topic word, compute the number of times that term appears in the cluster as occurrences
- Divide `occurrences` by the length of the cluster and store as `normalized_o`
- Compute the value `-normalized_o * log2(normalized_o)` as `term_entropy`
- Add `(cluster_size / total_no_of_vectors) * term_entropy` to `temp` — repeat with new topic term.
- Add `temp` to `entropy`, reset value `temp = 0.0` — repeat previous steps with a new cluster.
- Return `entropy` variable as the new entropy score of the clustering.

To compute the variance of a clustering:

- Compute the average of the cluster sizes as `avg`.
- Set initial value as `variance = 0.0`.
- For each cluster, add `(cluster_size - avg) ** 2` to `variance`.
- Divide `variance` by the total number of clusters and return.

Note that the GAIN of a clustering is the entropy of the original dataset subtract the entropy of the clustering. Both use the same algorithm to compute entropy. An optimal clustering algorithm would have high GAIN (high reduction in entropy) and low variance (high similarity in cluster sizes).

Explanations & Rationale

- TF-IDF was used to compute word importance because it balanced term frequency against frequency across all documents — which was deemed a sufficient measure of importance without bias.
- The top 5 words for each document hold valuable context for determining the article's purpose and were therefore chosen during feature selection phase — any fewer would impact the quality of the dataset.
- The pared down feature vector was reduced to 10% of the original size in order to have the sizes differ by an order of magnitude — this was deemed large enough of a difference to create disparities in cost and performance. The top 10% of terms were chosen to mitigate the loss of information during paring.
- K-Means clustering was chosen because it is the most naive form of clustering and served as a good baseline for interpreting the quality of clustering algorithms.
- DBSCAN clustering was chosen to compare with K-Means clustering because we wanted to observe the differences between centroid-based clustering and density-based clustering — two sufficiently different approaches to clustering. Hierarchical clustering was deemed too expensive time-wise to perform.
- Only Euclidean distance was used with K-Means clustering because other distance metrics (e.g. cosine) have edge cases that cause the algorithm to diverge and loop infinitely — it is also disallowed by sklearn.

- Only Euclidean distance was used with DBSCAN clustering because the purpose of this assignment is to compare performance between clustering algorithms — so as many variables should be consistent across experiment sets to ensure no confounding variables impact the quality/cost of clustering.
- The number of initial centroids for K-means clustering was set to $n = 10$ because this was the sweet spot found during testing to optimally balance entropy GAIN and performance time for the dataset. Running the same code on a different or extended dataset might require a different optimal number of clusters.
- The epsilon and min_points were set to 0.1 and 10 respectively for DBSCAN clustering because this was the optimal setting such that both clusters had sufficiently large entropy GAIN values. Setting epsilon to be too large had the standard feature vector return 0 GAIN, setting the min_points too low had the pared feature vector return 0 GAIN. Setting the min_points too high lead to both datasets returning 0 GAIN.
- The entropy was computed using the algorithm described above in order to compensate for many articles having multiple TOPIC class labels. The algorithm was created to be as close to the original entropy formula discussed during class as possible — however, the existence of multi-label articles means the entropy will exceed 1. Thus, the entropy scores are not normalized — however, the GAIN score resultant from subtracting two entropy scores will be normalized to [0,1]
- Articles with no topic class labels were still included in the clusters despite contributing negatively to the entropy score. These articles only contributed to the value of the total number of vectors in calculation.

Resources

This assignment was implemented and tested using Python 2.7.5 and used the following libraries/modules:

- os, sys, time
- string
- math
- operator
- BeautifulSoup4
- NLTK
  - nltk.stem.porter.*
  - nltk.corpus.stopwords
  - nltk.corpus.wordnet
  - nltk.stem.wordnet.WordNetLemmatizer
- sklearn
  - sklearn.cluster.kmeans
  - sklearn.cluster.dbscan

Other `sklearn` modules were used for classification but not included in the list because the classification portion of the module has been commented out of `lab4.py` for runtime purposes. The entire preprocessing and clustering module has been optimized to run in under a minute for a single file and under 21 minutes for the entire dataset. Future iterations may include multithreading during preprocessing and entropy computation.

Experiment

Note that the centroid to the K-Means clustering algorithms were generated randomly by the sklearn modules. Therefore, the clusters may differ across multiple executions of the code. The following 2x2 set of experiments were performed on a dataset of 1 file of approximately 1000 reuters articles.

- K-means clustering on the standard feature vector dataset.
- K-means clustering on the pared down feature vector dataset.
- DBSCAN clustering on the standard feature vector dataset.
- DBSCAN clustering on the pared down feature vector dataset.

Results & Interpretation

Note that depending on the machine used to test the code and the quality of the centroids generated by the sklearn.cluster.kmeans module, the time and performance may differ across multiple executions of the test code. The following results are based on a test dataset of 1000 Reuter articles from 1 sgm file:

Scalability (Offline Cost):

The offline cost for the 2x2 experiment set is represented in the table below:

	Standard FV	Pared FV
K-Means	1.031291008 s	0.143668889999 s
DBSCAN	0.766928911209 s	0.230996131897 s

Note that DBSCAN is significantly faster than K-Means for the standard feature vector — i.e. larger datasets. While the difference in offline cost between K-Means and DBSCAN for the pared feature vector only differs by 0.1 seconds. The conclusion being that paring down the feature vector leads to less disparity between DBSCAN (density-based) and K-means (centroid-basing) clustering in terms of performance.

Furthermore, DBSCAN seems to be preferable to K-means for larger datasets and seems to scale better. This is due to K-means time cost increasing linearly with the number of data points and it clusters points by computing the closest centroid to each point. On the other hand, DBSCAN computes clusters by computing the core points of a dataset — which depending on the dataset might not be linearly tied to the number of data points.

Paring down the feature vector improves the offline cost for K-Means clustering by an order of magnitude. This is because the K-means algorithm scales in number of computations linearly with respect to dimensionality and dataset size. Note that paring down improves the offline cost for DBSCAN as well — albeit at less than an order of magnitude due to performance scaling being non-linear for DBSCAN clustering.

Quality (Entropy Gain & Variance)

The entropy GAIN for the 2x2 experiment set is represented in the table below:

	Standard FV	Pared FV
K-Means	0.486546880943	0.277629969515
DBSCAN	0.66008065956	0.101235296799

Note that paring down the feature vector seems to lead to a significant decrease in the quality of performance. This is due to the loss of dimensionality in the data. The reduction of dimensions in the feature vectors brings the data points closer together, which reduces the granularity of the dataset — this causes both clustering algorithms to have a more difficult time distinguishing points.

The drop in performance from paring down the feature vector is more significant for the DBSCAN than K-means. This is because DBSCAN is a density-based clustering algorithms, which means different values for epsilon and min_points may be optimal for different subsections of the dataset. K-means does not have this problem because the clustering is centroid-based and independent from the variance of the dataset.

For the standard feature vector — and presumably for datasets of larger size and higher dimensionality — the DBSCAN algorithm performs better. This is because density-based clustering can find non-convex clusters and other bizarrely shaped cluster. Note that K-means outperforms DBSCAN by a smaller margin for the pared-down feature vector than DBSCAN outperforms K-means for the standard feature vector.

The cluster variance for the 2x2 experiment set is represented in the table below:

	Standard FV	Pared FV
K-Means	18786.6	50615.4
DBSCAN	244036.0	49020.2222222

Note that K-Means has far more balanced cluster sizes than DBSCAN. This is because K-Means is centroid-based and also optimally selects initial centroids to ensure a more even clustering. Density-based algorithms such as DBSCAN rely on taking the union of the neighborhoods of core points to form clusters — which may lead to larger clusters depending on the choice of epsilon and min_points.

Interpretation of Output

- Paring down the feature vector will lead to a significant improvement in runtime for K-Means.
- Paring down the feature vector will not improve the runtime for DBSCAN enough to compensate for the decrease in cluster quality and entropy GAIN. Furthermore, it seems to DBSCAN scales better than K-means for larger dataset with higher dimensionality. Therefore, the DBSCAN algorithm should be chosen for clustering if runtime or scalability of the problem is the primary concern.
- DBSCAN seems to outperform K-Means in terms of entropy GAIN for larger dataset — while K-Means is still the best choice for smaller datasets. Paring down the feature vector leads to a more significant performance drop in DBSCAN than K-means. Therefore, DBSCAN should be the chosen clustering algorithm for larger datasets with higher dimensionality — K-Means is the stronger clustering algorithm when scaling down the dimensionality of the problem (a legitimate concern for runtime).
- Paring down the feature vector leads to an increase in variance in cluster size in the K-means tests.
- Paring down the feature vector leads to a decrease in variance in cluster size in the DBSCAN tests.
- The K-means algorithm is preferable if evenness of cluster size is a primary concern — as the K-means algorithm seems to improve as the dataset and dimensionality increase.