5243 Introduction to Data Mining

Assignment 2

Johns Gresham

Introduction

In an effort to build automatic article categorization, search, and graphs, the issue of categorizing and comparing the contents of articles arises. To compare the similarity between two articles, we can lump articles into groups using clustering. Using these groups, we can then more easily label classes and analyze documents. A KMeans and DBScan algorithm were fit to the feature vector data created in Assignment 1. The algorithms' scalability and quality was measured to give some insight on the algorithms performance. Two different distance measurements and a few variations in parameters were used to further explore the algorithms.

External Sources

- Sci-kit learn 0.16.0
  - Provided the *DBSCAN* clustering algorithm, with flexibility for specifying a distance matrix, which it also provided a function for *pairwise*_distances (Euclidean and Cosine). It can also normalize a set of feature vectors and find the average silhouette score for a cluster.
- NLTK 3.0.5
  - Provided the *KMeans* clustering algorithm, with flexibility for specifying distance metric (Euclidean and Cosine). *Nltk.tokenize* and *nltk.corpus.stopwords* helped in preprocessing the data.
- NumPy 1.9.2
  - Required *numpy.array* format for Sci-kit learn and has a *numpy.var* function used to find the variance (skew) for a cluster.
- BeautifulSoup4.4.0
  - Used when parsing the article information from the html files.

Data Formatting

Picking up from Assignment 1, we had a set samples and each sample had a feature vector and label. However, the feature vectors were not ready to be input into a clustering algorithm. First, the features had to be a matrix format [#samples, #features] and labels in an array [#samples]. In my case, the features were just the tf-idf value for that sample of every filtered word(non-stop word, occurs more than once, etc.) that appears in the corpus in an article. If there were 1000 filtered words in the corpus and the article had a few words with non-zero tf-idf values, then only a few of the 1000 features are non-zero. So there were no discrete or string features. Then each feature is normalized over the sample set so certain features with high values aren't orders of magnitude larger than others and every feature is in the range 0 – 1.

Scalability

For the scalability and quality measurements for each clustering and distance formula, I will keep consistent the number of clusters for KMeans and epsilon and min_points in DBSCAN. N=25 clusters for KMeans and epsilon=0.2, min_points=3 in DBSCAN. Every test was done using 6151 features over 3179 samples.

	KMeans	DBSCAN
Euclidean	416.46s	429.42s
Cosine	423.26s	431.65s

DBSCAN created 69 clusters in for the Cosine metric and 26 for Euclidean.

Quality

Variance and the silhouette values were calculated using scikit. Entropy was calculated using the class definition and the topic labels. The fewer the number of samples in a cluster with different labels, the lower the entropy, or disorder in the cluster.

	KMeans	DBSCAN
Euclidean		
Avg. cluster entropy	0.28	0.015
Avg. cluster variance	2.7e-3	3.4e-05
Avg. cluster silhouette	-0.163	-0.193
Cosine		
Avg. cluster entropy	1.51	0.017
Avg. cluster variance	3.2e-3	4.6e-05
Avg. cluster silhouette	-0.211	-0.244

Discussion

First I will comment on my brief observations of the effects of locally tuning the number of clusters in KMeans the the epsilon and min_points values. Generally, KMeans lumped most of the samples in a single cluster and had a handful of samples in the rest of the clusters. With clusters in the single digits, the algorithm placed most samples in a single cluster. In the tens, the majority of the samples were dispersed in several clusters. Above 25 clusters, KMeans seemed to just create more single sample clusters. For DBSCAN, I chose min_points=3 due to an observation during testing. I noticed some topics only occurred a handful of times in the sample set. So in order for them to be in a cluster by themselves, min_points would need to be low. Then with any episilon over 1, there was only a single cluster after DBSCAN completed. Lowering it to below 0.5 produced mostly clusters with singe topics, then one cluster with roughly half of the samples.

Secondly I had to chose how to label the samples from the topic information. For the articles, there were often a number of topics, but for many of the measurements and algorithms I needed a single labeling. I chose the first topic listed. I could've chosen a random one, but that seemed arbitrary as well. A better solution could be some vector representation of all the topics and find some tolerable distance between vectors under a label.

Overall, DBSCAN produced more clusters that had the majority of the samples of a single label compared to KMeans which produced more single sample clusters and a few clusters with samples of mixed labels. However, I did not do a complete scan of parameter values for the number of clusters, epsilon, and min_points which could lead to much different results. Also, the number of samples was limited from the original sample size due to missing labels and running time during testing. Increasing the sample size and label selection/comparison could have positive results for quality, and perhaps negative results on run time.

File Locations

directory : /home/7/gresham/5243/lab2/

makefile

readme: contains run instructions

report.docx

kmeans.py

dbscan.py