Homework #5: Document Clustering

Due: April 14, Friday
100 points

In this homework, you will implement two document clustering algorithms. The first is hierarchical (agglomerative/bottom-up) clustering HAC (in Python), and the second is (spherical) k-means (in Spark). K-means on unit vectors is often called spherical k-means.

A. Problem Description

In both algorithms, represent each document as a unit vector. The unit vector of a document is obtained from tf*idf vector of the document, normalized (divided) by its Euclidean length. Recall that tf is term frequency (# of occurrences of word in the document) and idf is given by $\log \frac{N+1}{\mathrm{df}+1}$, where N is the number of documents in the given collection and df is the number of documents where the word appears.

Use Cosine function $\frac{\mathrm{A}\cdot\mathrm{B}}{\|\mathrm{A}\|\|\mathrm{B}\|}$ to measure their similarity/distance.

Use the distance between centroids to measure the distance of two clusters in hierarchical clustering.

In both algorithms, the centroid is given by taking the average over the unit vectors in the cluster. You don't need to normalize the centroid.

Both algorithms should return the clusters found for a desired number of clusters.

B. 1st Task: HAC

You are required to use heap-based priority queue to store pairwise distances of clusters. You may use Python heapq module (https://docs.python.org/2/library/heapq.html) which implements a heap. Take care on how you handle the removal of nodes that involve the clusters which have been merged: your implementation should not enumerate all nodes in the queue which would take O(n^2) time where n is the number of data points to be clustered. Since python doesn't provide an efficient way to delete node from heap, you may use delayed deletion here:

1. Instead of removing old nodes from heap, just keep these nodes in the heap.
2. Every time you remove an element from the heap, check if this element is valid (does not contain old clusters) or not. If it is invalid, continue to pop another one.

Use log of base 2 to compute idf.

Write your code in Python to implement the HAC algorithm until it produces a desired number of clusters. Output documents IDs for each cluster.

Execution Format

python hac.py docword.txt k

The program takes 2 arguments:

- docword.txt is a document-word file. We will describe its format in the Data Sets section.
- k is the desired number of clusters.

Output Format

In this task, **DO NOT** write the output to any files. Use **standard output** to print them instead.
For each cluster, output documents IDs that belong to this cluster in one line. Each ID is separated by comma. For example,

```
96,50
79,86,93
97
4,65,69,70
…
```

The order doesn't matter

C. 2nd Task: Spherical k-means

In this part, you are asked to modify the Spark k-means code located at: <your spark installation directory>/examples/src/main/python/kmeans.py (which is covered in class).

Your document-vector RDD needs to be sorted in the increasing order of their integer ID (as given in the input file). You need to call takeSample(False, k, 1) on the sorted RDD to obtain the initial centroids.

You are required to use Cosine function to measure their similarity/distance. The k-means should start from k initial points, and stop when the sum of Euclidean distances between new and old centers is less than the given convergence threshold (note that here we use Euclidean distance as the stop criterion, not the Cosine similarity). Then output the number of nonzero values that each cluster centroid has into a file.

Execution Format

bin/spark-submit k-means.py inputfile k convergeDist output.txt

The program takes 4 arguments:

- inputFile is a document-word file. We will describe its format in Data Sets section.
- k is the number of initial points we get.
- convergeDist is the given threshold. We use this float number as the k-means stopping criterion.
- output.txt is the output file.

Output Format

In this task, you need to save the output into a file.

For each final center, output the number of its nonzero values as following:

```
687
600
509
560
…
```

It means that #0 cluster center is a sparse vector that has 687 nonzero values.
The order doesn't matter

D. Data Sets

We will use the Bag-of-words dataset (https://archive.ics.uci.edu/ml/datasets/Bag+of+Words) at UCI Machine Learning Repository to test your algorithms.

The data set contain five collections of documents. Documents have been pre-processed and each collection consists of two files: vocabulary file and document-word file. We will only use document-word file.

For example, "vocab.enron.txt" is the vocabulary file for the Enron email collection which contains a list of words, e.g., "aaa", "aaas", etc. "docword.enron.txt.gz" is the (compressed) document-word file, which has the following format:

```
39861
28102
3710420
1 118 1
1 285 1
1 1229 1
1 1688 1
1 2068 1
…
```

The first line is the number of documents in the collection (39861). The second line is the number of words in the vocabulary (28102). Note that the vocabulary only contains the words that appear in at least 10 documents. The third line (3710420) is the number of works that appear in at least one document.

Starting from the fourth line, the content is <document id> <word id> <tf>. For example, document #1 has word #118 (i.e., the line number in the vocabulary file) that occurs once.

- We will only use part of the document-word file to test your program, and the number of documents in the collection we use will be less than 2000. Since the number of words in the vocabulary may be very big, you are **required** to use SparseVector or csc_matrix (see https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.linalg.Sparse

Vector, https://spark.apache.org/docs/latest/mllib-data-types.html, and https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csc_matrix.html#scipy.sparse.csc_matrix) to store unit vectors.

E. Submissions

For task 1, name your python script as <FirstName>_<LastName>_hac.py.
For task 2, name your python script as <FirstName>_<LastName>_kmeans.py

Important Notes:

- **We will be using Moss for plagiarism detection**. **Do not copy from each other or you will face serious consequences!**
- For task 1, use the standard output to print results. For task 2, save results into a file.
- The input file for HW5 already contains the TF vectors of the documents. So you need to convert the TF vector to the TF-IDF vector. In task 2, you may use the TF-IDF method provided by Spark (see https://spark.apache.org/docs/latest/ml-features.html#tf-idf) to do this.
- You may use scipy package in this homework. To install it on EC2, execute “pip install --user numpy scipy”.