PROJECT REPORT

AKSHAY KULKARNI kulka182@umn.edu

PRE-PROCESSING

Following extra steps were taken during pre-processing

- Stop words were removed. Hard-coded list of stop words was included in code and tokens/ngrams from the files were removed if they belonged in the list containing stop words.

- The tokens/ngrams that occurred in less than **two** documents or in more than **90%** documents were removed. This is because words occurring very frequently or very rarely has very less discrimination power.

- All the files where tag "Lines:" occurred before subject were ignore. There were around 24 files like that. Also, if "Lines: 25" tag existed then only next 25 lines from file were read and processed.

- For each term in each document, its tf-idf was score was computed.

- In addition to this, one can also generate required files that contain only frequency of the term in the document (not td-idf computation) by enabling option of **–use_tfidf=0**

  $ ./preprocess –use_tfidf=0

- It takes around 4 minutes to process all the files across all folders generate all relevant files.

DATA STRUCTURES USED FOR PRE-PROCESSING

- **TRIE** data-structure was used to store different words across all documents. It helped in removal of duplicate words/tokens across different documents and kept only the unique ones.

- Dictionary(map) was used to store frequency of each token in the document and frequency of token across all documents for computing tf-idf score for each term.

DATA STRUCTURES USED FOR CLUSTERING

- Since Document-Term matrix is a high-dimensional matrix, it was stored in CSR format. High dimensional matrix can be stored in CSR format by using 3 arrays namely **rowind, colind and values.**

    - **rowind** => rowind[i] stores the index of starting location of object i in colind.
    - **colind =>** colind[i] stores the column index (or feature id) of the non-zero value.
    - **values =>** values[i] stores frequency corresponding the feature stored at colind[i]

- **Centroids** were stored as dense vectors because centroids may have many non-zero values as its requires adding feature vectors of varying and different dimensions.

OTHER FEATURES

- In addition to traditional K-mean, I have also implemented incremental K-means and tested against both data-sets (tf-idf and regular frequency) and traditional K-means performed better. To run incremental K-means use following command:-
  $ **sphkmeans input-file class-file #clusters #trials output-file –method=inc**

- In addition to this I have also plotted heat-map for clustering showing distribution of points from different class across various clusters. All code for this can be found here[https://goo.gl/LzgnT5]

STATISTICS OF DATA AFTER PRE-PROCESSING

Feature Type	# Objects	#Dimensionality	#Non-Zeroes
Bag of words	6486	25289	519768
3-gram	6486	22382	2574060
5-gram	6481	158265	4028844
7-gram	6478	364400	3796451

STATISTICS FOR CLUSTERING

All tests were ran on CSE Lab machines and the times reported are time taken for clustering (time for IO not included)

- **Traditional K-means with TF-IDF Frequency**

Feature Type	Clusters	Entropy	Purity	Time(in secs)
bag.csv	20	2.182011	0.522510	4.455370
bag.csv	40	2.086442	0.536232	7.877103
bag.csv	60	2.079652	0.541628	23.129917
char3.csv	20	2.503133	0.427999	23.409864
char3.csv	40	2.448055	0.434166	41.066676
char3.csv	60	2.273687	0.473019	64.895686
char5.csv	20	2.472509	0.426940	79.784500
char5.csv	40	2.233677	0.491128	204.289603
char5.csv	60	2.276458	0.489893	248.750000
char7.csv	20	2.870251	0.367706	98.286769
char7.csv	40	2.724683	0.386076	197.823933
char7.csv	60	2.647474	0.407533	277.585092

- **Incremental K-means with Normal Frequency**

Feature Type	Clusters	Entropy	Purity	Time(in secs)
bag.csv	20	2.916804	0.359698	3.849724
bag.csv	40	2.699375	0.390842	4.856060
bag.csv	60	2.601136	0.409497	8.195541
char3.csv	20	3.277300	0.261795	9.981542
char3.csv	40	2.859616	0.350139	16.557323
char3.csv	60	2.758969	0.379124	26.133482
char5.csv	20	3.054948	0.317544	47.382010
char5.csv	40	2.638709	0.416294	83.242808
char5.csv	60	2.529238	0.455022	113.937403
char7.csv	20	3.210340	0.279407	71.605894
char7.csv	40	2.981128	0.339148	104.921835
char7.csv	60	2.658469	0.417104	141.104003

- **Traditional K-means with TF-IDF Frequency**

Feature Type	Clusters	Entropy	Purity	Time(in secs)
bag.csv	20	2.836039	0.361548	5.289126
bag.csv	40	2.583120	0.425532	8.761991
bag.csv	60	2.559492	0.419981	20.043705
char3.csv	20	3.215103	0.284613	25.593709
char3.csv	40	3.042405	0.308665	49.693332
char3.csv	60	2.801406	0.361702	71.856026
char5.csv	20	2.768103	0.367536	118.159745
char5.csv	40	2.742456	0.372474	187.858844
char5.csv	60	2.546572	0.427095	304.072020
char7.csv	20	2.950348	0.354739	118.068670
char7.csv	40	2.824477	0.381599	248.247578
char7.csv	60	2.729939	0.392096	308.875009

- **Incremental K-means with Normal Frequency**

Feature Type	Clusters	Entropy	Purity	Time(in secs)
bag.csv	20	3.051830	0.324391	4.214258
bag.csv	40	2.778600	0.362781	5.345085
bag.csv	60	2.459973	0.470860	8.258685
char3.csv	20	3.328178	0.237897	11.589281
char3.csv	40	3.034307	0.314678	21.877292
char3.csv	60	3.050046	0.316528	31.570499
char5.csv	20	3.050185	0.313841	46.995278
char5.csv	40	2.776748	0.368307	88.686004
char5.csv	60	2.569251	0.422157	120.284469
char7.csv	20	3.098096	0.316456	74.070257
char7.csv	40	2.978804	0.337295	111.205401
char7.csv	60	2.734745	0.380519	149.886136