# GSoC 2017 - Clustering of Search Results

## *About me*

Name: Richhiey Geeverghese Thomas

Email: richhiey1996@gmail.com

Github: https://github.com/richhiey1996

Emergency contact: 9167060079 / 9892803695

Nationality: Indian

## *Background Information*

I would like to apply to Xapian Search Engine Library as a part of Google Summer of Code 2017. I was a part of GSoC 2016 with Xapian where I had worked on clustering of search results. I am currently working on getting it merged in and hope to get it merged in before the official start of GSoC 2017. Since I have been working with Xapian, I am fairly used to the codebase and wouldn't require time to get used to the development workflow. (Always learning small things about the workflow on git and otherwise)

My main aim with GSoC 2017 would be to improve the clustering system developed last time by adding in automated performance analysis, dimensionality reduction and developing an agglomerative hierarchical clusterer and merge all of the work either into the cluster branch or master branch.

I will be working from India (Mumbai – UTC+05:30) and my expected work hours will be from IST 12pm to 6pm till the first phase of evaluations and thereafter, from IST 12 pm to 9 pm. I will be able to put in more time during the weekends incase the project requires more attention.

This project will be my main focus during the time of the program. Xapian is the only organization I will be applying to for GSoC 2017 and this is the only project within Xapian I will be applying for.

## *Project Information*

Looking at the current state of the API, once we can merge all the code from last year, we can have a straight forward Spherical KMeans Clusterer which performs fairly well. Ahead from here, there are three main directions I want to work in this year:

## ***Automatic Performance Analysis***

For any clustering system, it is necessary to evaluate the results returned after clustering to find out whether the results are optimal. In case of clustering, result evaluation isn't a trivial operation. Since the documents that the clusterer will be clustering are search results (stored in an MSet), it wouldn't be fair to assume ground truth tables for the documents. We will not have a document to label mapping to evaluate how correctly documents are clustered. Thus, using internal clustering evaluation methods would be more favorable instead of external clustering evaluation methods.

Internal clustering evaluation methods revolve around the requirements of clustering i.e. the intra-cluster distance should be as less and inter-cluster distance should be as high as possible. This can give us an idea of how well a certain clustering configuration worked, and accordingly be able to change one or more of the parameters involved.

I guess a regular use case of this API will not provide a way to have ground truth labels for documents. Thus internal evaluation techniques would be the better option. I would thus like to introduce a few internal clustering evaluation techniques. They are:

1) *Silhouette coefficient*

- Silhouette coefficient is calculated with mean intra-cluster distance (a) of a point and the mean nearest cluster distance (b) :

  S(i) = (b(i) - a(i)) / max(a(i),b(i))

  The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters.
  The silhouette coefficient shows how well a point fits in its cluster, compared to the other clusters.

  Link : https://en.wikipedia.org/wiki/Silhouette_(clustering)

2) *Dunn Index*

- Dunn Index is another internal evaluation method. Dunn Index can be used to find the fitness of a clustering result for different number of clusters. The result with maximum Dunn Index can be chosen.

  Link : https://en.wikipedia.org/wiki/Dunn_index

3) *Root Mean Square Standard Deviation*

- RMSSD is a measure of the homogeneity within clusters

  Link : Page 75-76 of
http://cda.psych.uiuc.edu/multivariate_fall_2012/systat_cluster_manual.pdf

4) *Calinski-Harabasz index*

- The cluster index of Calinski and Harabasz is calculated using the following equation
  CH(k) = [trace B / (K -1)] / [trace A / (N - K)]
  Where,
  trace A = The error sum of squares between different clusters (inter-cluster)
  trace B = the squared differences of all objects in a cluster from their respective cluster center

  Link :
http://stats.stackexchange.com/questions/97429/intuition-behind-the-calinski-harabasz-index

5) *Davies-Bouldin Index*

- This is an internal evaluation scheme, where the validation of how well the clustering has been done is made using quantities and features inherent to the dataset.

  Link : https://en.wikipedia.org/wiki/Davies%E2%80%93Bouldin_index

Evaluation methods like these are useful as heuristics to compare various clustering solutions. Thus if we have two clusterers, we can use these indices to compare the quality of clustering, using different parameters for some clusterers.
As Olly rightly suggested, it would be better to have a namespace with these functions. The function names can be named after the corresponding evaluation technique. These function can give an overview of the clustering done.
// Use Clusterer and get the ClusterSet ‘cset’

```
double metric = silhouette(cset);
metric = dunn_index(cset);
metric = rmssd(cset);
metric = calinski_harabasz(cset);
metric = davies_bouldin(cset);
```

## ***Dimensionality Reduction***

In text clustering, we see that the term vectors created from the text documents turn out to be very high dimensional since a document can contain a large number of terms. This high dimensionality causes problems such as too common or too uncommon words which dilute the information in the vector and increase the size, thus increasing run time.

The main ideas used in dimensionality reduction are :

- Stemming
- Removing frequent words (Stopword removal)
- Removing uncommon words
- Semantic dimension reduction (Using LSA)

Stemming, stopword removal and removal of uncommon words will be done as a part of noise reduction in the preprocessing step. In addition to this, we can pass an option within the constructor of the clusterer if it requires LSA.

### *Stemming :*

Xapian Document termlist already contains stemmed words as well unstemmed words. Thus currently, simply removing the unstemmed words, to get all the stemmed words in the document

### *Stopword removal and uncommon words:*

The Document termlists contain a lot of terms, out of which a few are terms that are used frequently in language like articles and prepositions. These terms do not add value to the termlist and can thus be removed. Xapian has a list of stopwords which can be loaded into Xapian::SimpleStopper to identify stopwords before calculating their TF-IDF weights.

In case of uncommon words, most such terms are because of spelling mistakes or words that carry very less information. Thus, these words can be removed by checking how many times these same words have occurred in other documents. If they have occurred too few times (for example, only once), we could easily remove it by keeping a threshold. One way to do this would be by keeping the threshold at 1, so that any noise in the document termlists which do not contribute to distance calculations are removed.

### *Semantic dimension reduction*

This is the main thing that needs to be implemented in GSoC this year. The above three techniques are methods that are easy to add into the process, but adding in an unsupervised method to reduce dimensionality could be a tad bit harder. This will be applied on termlists that have already been stemmed and scanned for stopwords. For this, I would like to implement Latent Semantic Analysis for reduction of text document vector size.

For this, we could create a DimReduction class to be inherited by LSA class, which will implement the functionality required. A method within this LSA class called reduce() will reduce a point given to it and give back a Point with the reduced termlist.

The LSA class will implement a transform function which will take the initialized points and return these Points with the reduced term document vectors. The most computationally intensive part of LSA is calculating the Singular Value Decomposition of the vector. We will be doing SVD with the help of randomized algorithm. This improves the speed. Follow the below links for more information on the randomized algorithm-
https://research.fb.com/fast-randomized-svd/
https://arxiv.org/abs/0909.4061

The decomposed vectors will help us get the new values for the terms, and the ‘n’ highest values are selected from the matrix for each document.
Thus, the method for getting LSA-reduced vector would be :

1) Create term document matrix (term frequency)
2) Compute SVD of the term matrix using randomized SVD
3) Compute the reduced document vector

Using LSA, we will choose the top ‘n’ terms as required, and then create a document vector with the ‘n’ top ranked terms for the document, with respect to the corpus.

## ***Hierarchal Clustering***

It is necessary to have both partitional as well as hierarchal clustering algorithms for clustering search results because both these methods have their advantages and disadvantages. Hierarchal clustering is divided into two broad types based on their technique, divisive or agglomerative.

Within a document corpus, it is better to see which document is closer to another cluster or document, than checking which documents are dissimilar to an entire corpus. Thus, it would be better to develop an agglomerative strategy to cluster documents.

For hierarchical clustering, we could subclass Clusterer and create a new Clusterer named HierarchicalClusterer. This clusterer will take the MSet as an input and give ClusterSet as output.

For the hierarchical clusterer, the one of the new challenge to face is the merge step where we merge two different clusters. The two clusters having shortest distance are grouped together and merged. In this, we would have to specify various linkage criterions for merging the clusters. I would like to name three linkage criterions :

- *Maximum or complete-linkage clustering*
  In this, distance between two clusters is the maximum distance between two points (one in each cluster)

- *Minimum or single-linkage clustering*
  In this, distance between two clusters is the minimum distance between two points (one in each cluster)

- *UPGMC or Centroid linkage clustering*
  In this, distance between two clusters is the distance between their centroids

All of the above techniques seem possible with the current API and by letting Xapian::Cluster have a function to merge two clusters.

Another challenge is to do all this quickly, because the complexity of a simple agglomerative clustering would go upto O(N^3), since we have to calculate O(N^2) distances and O(N) iterations. A good way to do this would be to store the k-nearest neighbours for every point so that we can reduce the complexity by reducing the number of distance computations being carried out. With a double linked list, we can get a complexity of O(kN log(N)) which is considerably lesser. A paper talking about this algorithms is :

https://www.researchgate.net/publication/268324352_Efficient_agglomerative_clustering_using_k_nearest_neighbor_graph

We can evaluate the results obtained by Hierarchical clusterer and KMeans in terms of quality of clusters returned on different datasets as well as the run time over a number of documents.

## ***Triangle Inequality***

This is a technique that can reduce the number of distance computations required in Clusterers while finding the closest clusters. When doing a normal KMeans run, we check the distance of each point with every other point in the ClusterSet. A lot of these computations are redundant, because a point that is far away will not be a part of the cluster. We can reduce redundant computations by finding such points.

Here is a paper which shows us how to do it for KMeans :

http://users.cecs.anu.edu.au/~daa/courses/GSAC6017/kmeansicml03.pdf

The criteria to find out which points don't need to be calculated can be extended to the AgglomerativeClusterer too. It can considerably reduce the amount of time taken for the Clusterer to run.

Here is a paper which shows us how we can implement it with the AgglomerativeClusterer :

Page 7, Heading 4 in
http://ai2-s2-pdfs.s3.amazonaws.com/1fcd/8971378cfbfe9d52889493af5c05b9a345e5.pdf

## ***Final Notes***

At the end of this, we should have a clustering interface which can take in an MSet and return a ClusterSet containing the clusters and their respective documents. We will have 3 clusterers, a partitional Spherical KMeans clusterer, an agglomerative hierarchical clusterer and a dummy Round Robin clusterer which should look like :

KMeans (num_of_clusters, max_iters, distance_metric, init_method, dim_reduction)
RoundRobin (num_of_clusters)
HierarchialClusterer (num_of_clusters, linkage, distance_metric, dim_reduction)

# TIMELINE :

Through experience, I have learnt that this is the most important part of the proposal and due to less foresight in my previous year, I wasn't able to make a perfect timeline for the project, because of the way the project developed. This year, I will try to keep in mind these mistakes and make it well planned.

I would like to plan weekly from May 4th onwards since that would be a good level of granularity.

**Present - May 4th :**
*Merge existing PR for KMeans Clusterer:*

The main aim for this part is to merge the current PR open for the KMeans clusterer after reviewing the entire code and adding in the needed changes. In case this gets done earlier, a stretch goal for this period is to add in multiple distance metrics. Currently, we only have the Cosine Similarity. It would be good to add in Euclidian and Manhattan distance metrics with appropriate tests.

**May 5th - May 11th :**
*Silhouette coefficient:*

Start by setting up the ClusterEvaluation class and create a function for Silhouette which should find the average silhouette coefficient over the ClusterSet. This implementation will have a docstring and tests will be written for this evaluation metric. We will also need to update the User Guide. This will also act as buffer time for merging the old PR if more time is needed, since only one task is taken up for the week.

**May 12th - May 18th :**
*Dunn Index and RMSSD :*

Similar to Silhouette class, we can create functions for Dunn Index and RMSSD and write appropriate docstrings to explain the indices and tests for these two evaluation metrics.

**May 19th - May 25th :**
*Calinski-Harabasz index and Davies-Bouldin index*

Similar to the previous week, we can create functions for CalinskiHarabasz and DaviesBouldin and write appropriate documentation to explain the indices and tests for these two evaluation metrics.

**May 26th - May 31st :**
*Buffer time for fixes and optimizations*

Incase the project runs behind schedule or to add in optimizations that could help performance, we could use this time for finishing the work up on the ClusterEvaluation class.
The stretch goal for this period would be to add in the KMeans++ initialization.
KMeans++ is a way to initialize the centroids of the KMeans Clusterer so that these initial centroids are as far away from each other as possible. Since the output of KMeans depends on the initial selection, KMeans++ gives faster conversion and better clusters too. It will be useful evaluate the difference in clustering results with the help of ClusterEvaluation class.

**June 1st - June 7th :**
*Implement various linkage criterions*

Implement and test the three linkage criterions i.e complete-linkage, single-linkage and centroid-linkage clustering. These will be implemented as private methods within the ClusterSet class

**June 7th - June 13th :**
*Start with work on hierarchical clustering*

Create the class HierarchicalClusterer by inheriting Clusterer class and implement the helper functions (mainly the merge method) needed to implement the cluster method. Implementing tests for the merge method would be necessary.

**June 14th - June 20th :**
*Implement the agglomerative clusterer :*

Using the helper methods and the linkage criterions, we can implement the clustering functionality within the cluster method, inherited from Clusterer. This clusterer should return a ClusterSet containing Cluster objects.

**June 21th - June 27th :**
*Fixing, optimizing and profiling the agglomerative clusterer :*

Checking whether the hierarchical clusterer is running fast enough to be useful by profiling and checking which parts take the most time, adding in language optimizations if any, and adding any fixes

**June 27th - June 28th :**
*Phase 1 evaluations and mentor feedback on project so far*

**29st July - 4th July :**
*Evaluate KMeans and Agglomerative Clusterer on data with ClusterEvaluation*

We can evaluate both these clusterers in terms of running time and quality of clustering with the help of ClusterEvaluation class on different datasets such as BBC dataset and 20 newsgroups datasets.

The links to both these datasets are:
20 newsgroups dataset:
http://qwone.com/~jason/20Newsgroups/

BBC datasets:
http://mlg.ucd.ie/datasets/bbc.html

Other datasets that are found better can be used too.
If not already implemented, kmeans++ should be implemented and evaluated in this period.
By the end of this period, we should have a benchmark of how well KMeans with both its initializations and the HierarchicalClusterer on **different datasets,** with **different corpus sizes**, and **without dimensionality reduction**.

**5th July - 8th July :**
*Dimension Reduction in the preprocessing*

1) Stemming - This is already done by Xapian, so this shouldn't be a problem.
2) Stopword removal - Xapian provides stopper class and a list of stopwords. Thus we can identify stopwords and remove them.
3) Uncommon words - Remove terms that occur only once in the corpus

Make sure a reduced termlist can be created by 8th July and evaluate, mainly for run time. If time permits, we could even see changes in quality evaluations with reduced termlists.

**9th July - 28th July :**
*Implementing LSA for dimension reduction*

**9th - 13th July :**
Add in a way to create a document-term matrix with the entry for each term being its frequency among documents in the corpus. Create this to pass to an SVD solver.

**14th-21th July :**
Implement the randomized SVD solver and find the decomposed matrices to calculate the reduced document termlist. Write tests for components of the SVD system.

**22nd - 25th July :**
Use the decomposed matrices to calculate the reduced document termlists and return the Point with reduced termlist. Write tests for the reduced document list being returned.

**26th - 28th July :**
Update documentation and buffer time to make up for any loss of work in the month.

**29th - 30st July :**
*Phase 2 evaluations and mentor feedback on project so far*

**31th - 3rd August :**
*Testing LSA and time for fixes and optimizations*

Profiling and optimizing the written code will be done in this period. This period with the above two days will act as a buffer to finish up DimReduction by this time

**4th August - 5th August :**
*Compare clusterer results without LSA and with LSA*

Compare the running time changes and the quality of the clusters returned and document these changes for future reference.

**5th August - 10th August :**

Implement triangle inequality for KMeans to speed up the the Clusterer and test on a dataset to check the speed up in runtime.

**11th August - 15th August :**

In a similar way, add in triangle inequality to the hierarchical clusterer and test on a dataset to check the speed up in runtime.

**15th - 20th August :**
*Merging all PR's that have been opened and cleaning up all the code*

Make sure all the work that has been done has been merged into the right target branch

**20th - 25th August :**
*Updating documentation wherever needed*

I'm not sure where all we would need documentation for the Clustering functionality, but it is safe enough to assume 5 days to finish off all the documentation work.

**25th August - 29th August :**
*Finish up coding and documentation (Final submission)*

Get mentor feedback on the project, get the work product report ready

**RELEVANT DISCUSSIONS:**

https://lists.xapian.org/pipermail/xapian-devel/2017-March/003095.html