Documentation

Agglomerative Clustering with k means final reclustering

Authors: Jakub Kamiński
Michał Grabowski

Table of contents

Problem Description

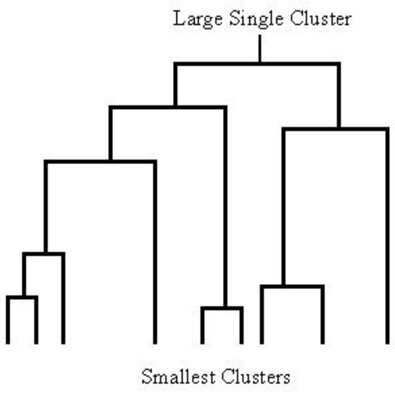
- Main Goal:
    - Finding similarities between words in text documents
        - Formulating a dictionary based on a document.
            - Could be used in spell-checking.
            - Searching prompts
        - Language analysis
    - Grouping words by similarity to observe how the groups are formed in a different sized documents
- Side Goal
    - Experimental, what are the most used words in
        - Web sites
        - Short stories

Technical Analysis

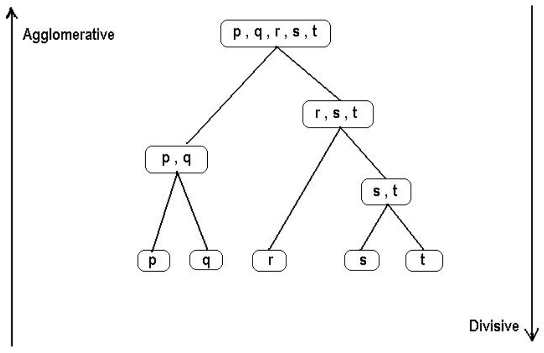
1. Target platform
    a. The Application’s target will be PC with Windows Operating Systems.
2. Development platform: Visual Studio
    a. Windows Presentation Forms in C#
        i. MahApps.Metro Toolkit
        ii. Loading Animations
3. Supported file types:
    a. Txt

Background

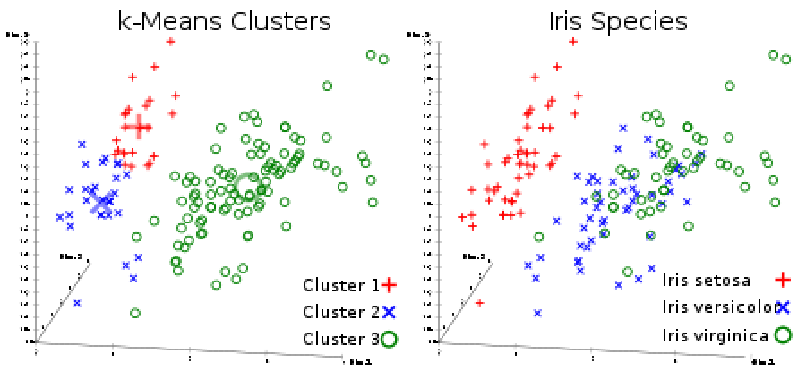
- Used Algorithms:
  - **Hierarchical clustering**: (
    - A distance measure is needed (ex. Hamming, Euclidean) to calculate distances between clusters.
    - Each point is a cluster
    - We merge nearest clusters depending on distance.



  - **Agglomerative clustering** (bottom up Hierarchical)
    - We start with each example (word) being it's own cluster
    - Compare all pairs of clusters and pair with the nearest
    - Stop when stop condition is satisfied



- **k-means algorithm**
  - Take k cases, each case is a centroid of it’s own cluster
  - Each other object is assigned to the nearest centroid.
  - Calculate the new position of the centroid of each cluster
  - Repeat steps 2 and 3 until all centroids don't change positions.



Description of our solution

In our program as an input the program takes a txt file. It clusters the items, where item is a string of characters separated by blank spaces (‘ ‘) and newline characters (‘\n’). Agglomerative clustering algorithm is used for the program, which means, that at the beginning the program assumes that each item forms a separate cluster. Then, comparing each clusters - the program merges two closest and continues iteration until there aren’t clusters that would exceed a specified minimum distance. As a distance measure in the string context, the program takes Levenst ein distance for comparing. It basically means that it calculated the amount of steps to transform from one word to another. Then the program uses k-means re-clustering to

To initialize k-means algorithm we need to find the centroid of words for given cluster. To do that, we opted to represent words in vector space. First of all, we need to remove repetition of given word w so that every word wi of our cluster is unique. In each cluster, we will have vector of words such that *V = [w1, w2, w3…..wn]* where wi represents the length of unique word with index.
Then we easily find the central word through calculating mean length of word in given cluster and finding the word with closest to the mean.

As a solution the program returns clustered data in a form of a graphical representation of centroids and the sets of words that are linked with them.

Program Architecture

Our program will consist of 2 modules, one responsible for program-user interaction and the second to create and merge clusters according to algorithms used

1. **Visual**
    a. User Interface
    b. Document to be clustered
    c. Visualisation of final solution
2. **Mathematical**
    a. **Algorithms**
        i. **Agglomerative Clustering**
            1. *InitializeClusters(List<Cluster>)* -takes an empty list of words and adds new word as a seperate entry to the list of clusters.
            2. *Cluster(List<Cluster> Clusters, int Constant)* Takes the list of clusters and begins the hierarchical agglomerative clusterization from the list list of clusters and with Constant as the stop value (the difference between clusters so large it stops computing)
            3. *Merge Clusters(cluster a, cluster b)* - Takes two clusters and merges them into a single cluster.
            4. *CalculateLevensteinDistance(string a, string b)* -Calculates distance between two words based on Levenstein algorithm i.e. how many letters do we need to change to make one word match the other.

        ii. **K - means**
            1. *KMeans(List<Cluster> Clusters)* -begins reclusterisation using the k-means algorithm.
            2. *FindCentroid(List<Cluster> Clusters)* - looks for ‘mean word’ in each of the clusters. We define the mean word as the word to which the rest of the words have a same distance.
            3. *CalculateClosests(List<Cluster> Clusters, List<string> Words)* - This function we use during reclustering, so that each cluster is added words that are closest to the centroid we found in previous iteration

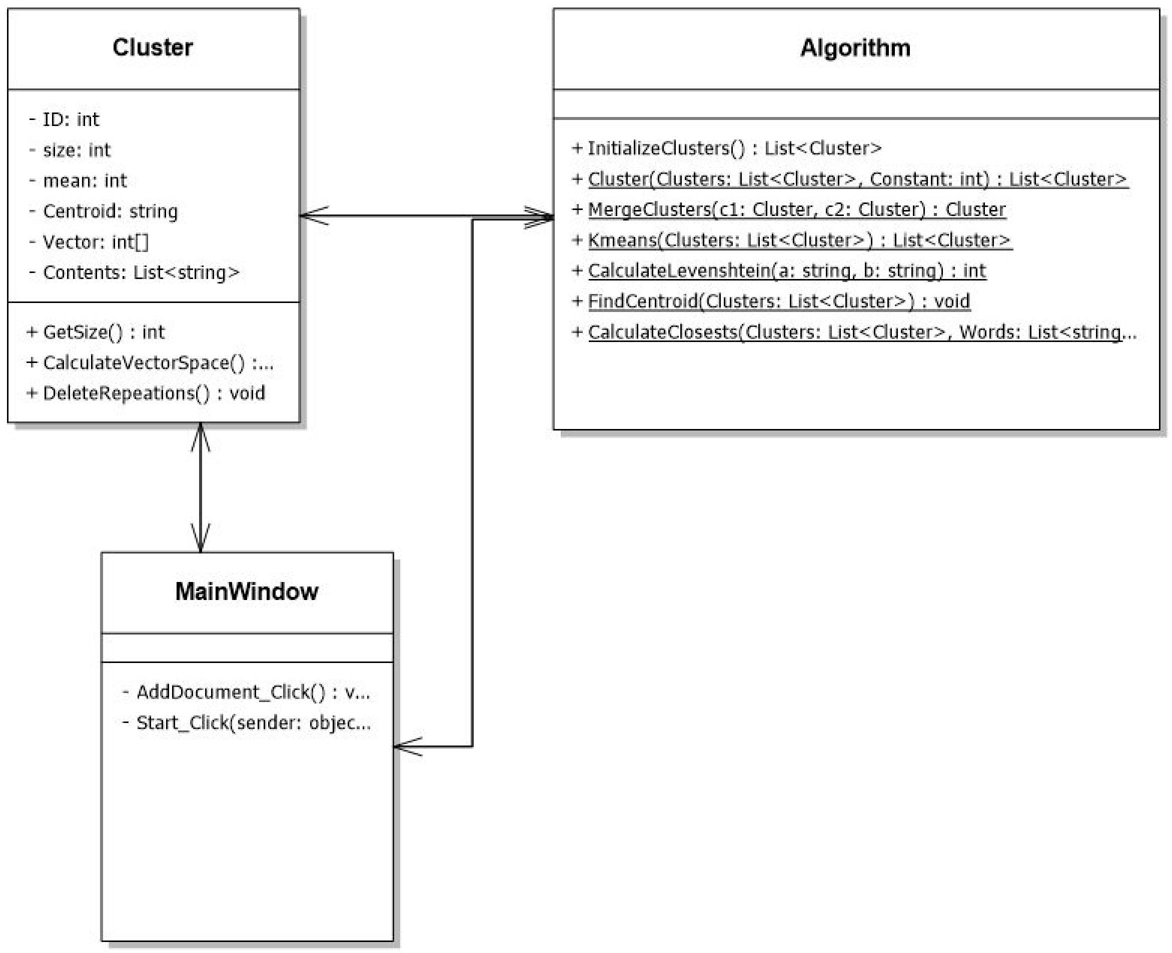
b. **Classes**

   i. **Cluster(int id, int size, List<string> Contents)** - The main class for the whole framework. It contains the id, information about the amount of words in the cluster and the words themselves stored in a List.
   We use lists as in c# they have many useful functionalities such as resizing on the fly and checking whether it contains some item.

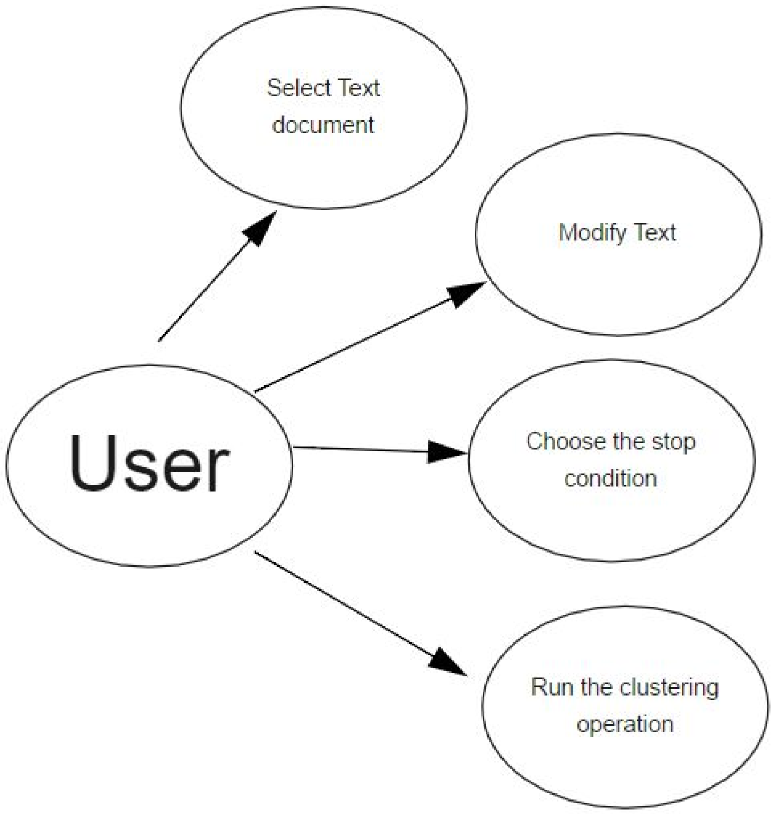
   ii. MainWindow()

   1. *AddDocumentClick()* - Opens the file dialog, we search for txt documents to check.
   2. *Start()* - We start the computation, then we return the results in the second tab

Class Diagram

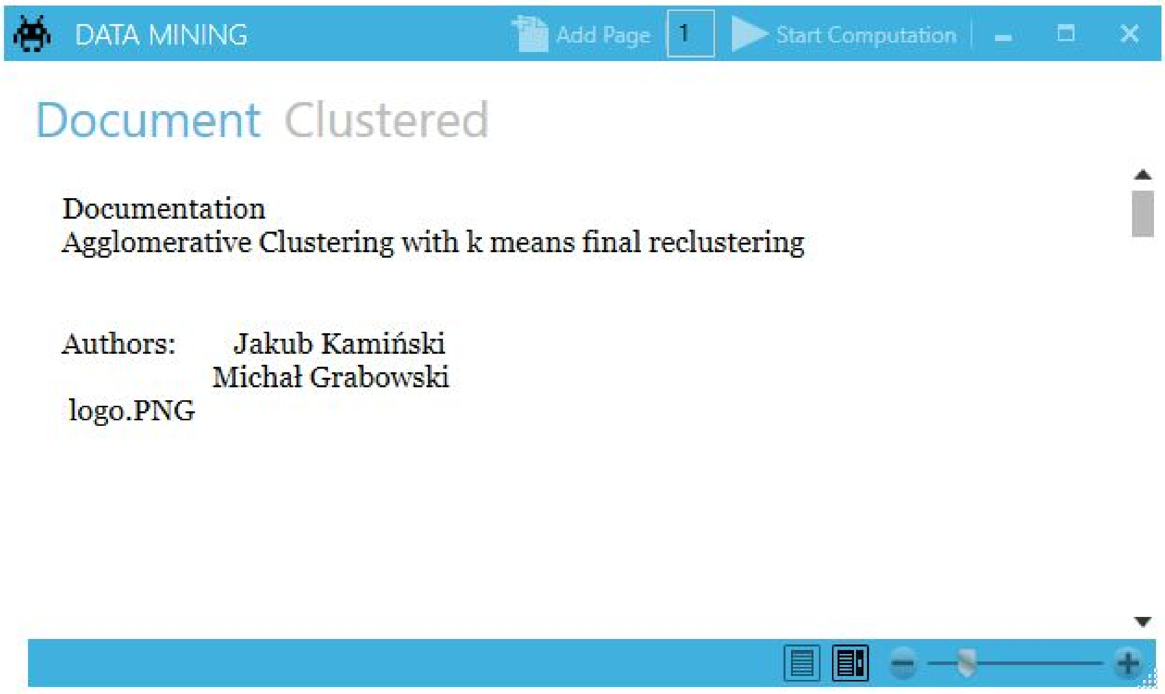

Cluster
- ID: int
- size: int
- mean: int
- Centroid: string
- Vector: int[]
- Contents: List<string>
+ GetSize() : int
+ CalculateVectorSpace() :...
+ DeleteRepeations() : void
Algorithm
+ InitializeClusters() : List<Cluster>
+ Cluster(Clusters: List<Cluster>, Constant: int) : List<Cluster>
+ MergeClusters(c1: Cluster, c2: Cluster) : Cluster
+ Kmeans(Clusters: List<Cluster>) : List<Cluster>
+ CalculateLevenshtein(a: string, b: string) : int
+ FindCentroid(Clusters: List<Cluster>) : void
+ CalculateClosests(Clusters: List<Cluster>, Words: List<string...
MainWindow
- AddDocument_Click() : v...
- Start_Click(sender: objec...

Use Case Diagram



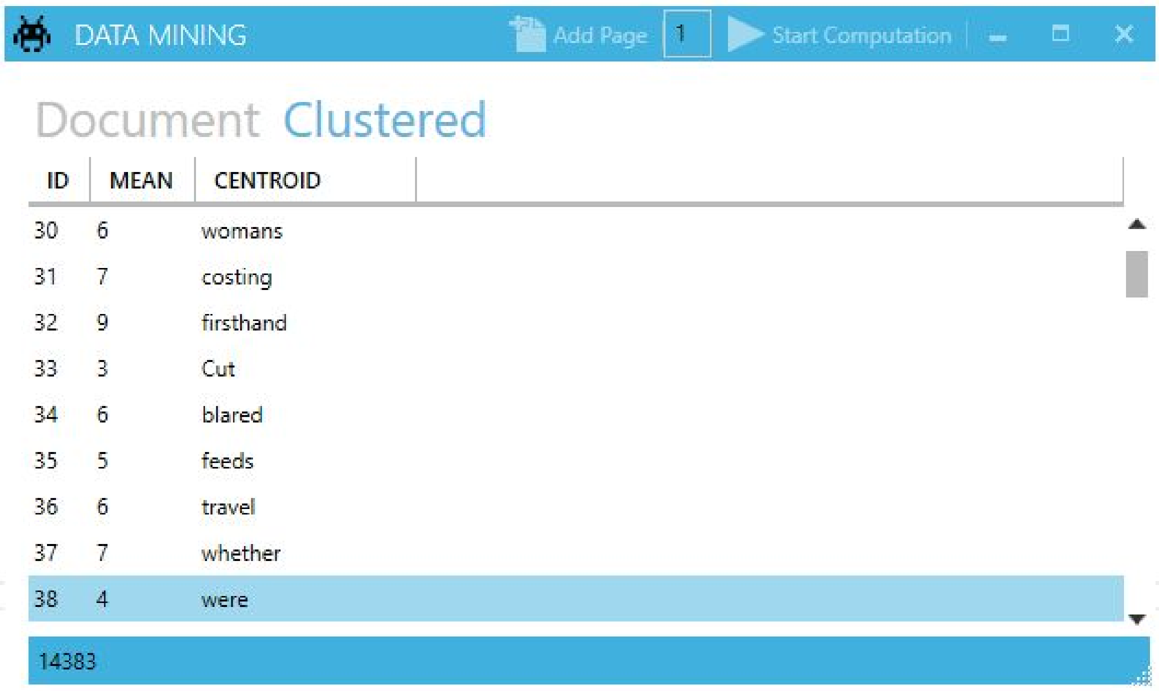
1. Startup DataMining.exe
2. Select first ‘Text’ tab.
3. Click ‘Load File’ and select a file to be calculated. The file must be in a .txt extension
    a. User can use built-in text editor to change the text if he so desires
4. Choose the maximal distance between clusters [stop property]
5. Run the program
6. Observe results on the second tab.

Graphical User Interface



Documentation
Agglomerative Clustering with k means final reclustering

Authors: Jakub Kamiński
Michał Grabowski
logo.PNG



ID	MEAN	CENTROID
30	6	womans
31	7	costing
32	9	firsthand
33	3	Cut
34	6	blared
35	5	feeds
36	6	travel
37	7	whether
38	4	were

14383

Data Sets

- Website articles saved as text files.
- Short stories. (up to 1000 words)
- Downloaded websites.
- Long stories (2000+ words)

Results:

Length(words)	**Time(ms)**
2000	866
3000	1871
2500	3700
4800	4500
10000	213712

As we can observe it’s not a linear complexity.

1. Sample dictionary:
   Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2. Sample output:
   Centroid: velit
velit, enim, veniam, mollit, nisi, sed, quis, Ut, ut, cillum, pariatur, voluptate, adipiscing
   Centroid: magna
magna, anim, minim, sint, non, fugiat, ad, ea, aliqua, ullamco, officia, cupidatat, exercitation
   Centroid: dolore
dolore, dolor, labore, do, culpa, nulla, tempor, commodo, laborum, laboris, nostrud, Excepteur, consectetur
   Centroid: irure
irure, aute, sunt, Lorem, in, id, Duis, ipsum, eu, eiusmod, proident, incididunt, reprehenderit
   Centroid: elit
elit, est, et, sit, ex, amet, esse, qui, aliquip, occaecat, deserunt, consequat

Conclusion

Our program gives satisfactory results for documents up to 10000 words. The initial clustering is choking the program with the high number of Levensthein distance which then speeds up rapidly with each iteration
The groups of words obtained through a numbers of tests can be easily added to a dictionary.

Sources

https://e.mini.pw.edu.pl/en/course_details/5357
http://www.cs.cmu.edu/~knigam/15-505/clustering-lecture.ppt

http://www.cs.utah.edu/~piyush/teaching/4-10-print.pdf