The handle [http://hdl.handle.net/1887/20358](http://hdl.handle.net/1887/20358) holds various files of this Leiden University dissertation.

**Author:** Witsenburg, Tijn  
**Title:** Hybrid similarities: a method to insert relational information into existing data mining tools  
**Date:** 2012-12-20
Chapter 6

\textit{K}-means and \textit{K}-medoids

Centroid-based clustering methods form clusters around a certain number of prototypes. Implementing the hybrid similarities in this type of clustering methods is not straightforward. It is not always possible to define the concept of a ‘mean node’ while at the same time having a meaningful hybrid similarity to it. Two ways around this problem are proposed. Experimental research will show both if there is a difference in performance for these two solutions and what the effect of using the hybrid similarities is.

6.1 Introduction

In previous chapters, we expressed the need for generative methods to create hybrid data mining techniques. We proposed two hybrid similarities, which can be used instead of the original, content-based similarity. The first experiments were set up to demonstrate the effect for agglomerative hierarchical clustering. Chapter 5 describes how this is done, and what the effects of using the hybrid similarities instead of the content-based similarities are on the quality of the found clusterings.

These experiments showed that the hybrid similarities performed significantly better than the content-based similarity. This leaves the question whether the hybrid similarities also will improve other data mining techniques. This chapter explores the possibilities to implement the hybrid similarities in two centroid-based clustering methods: \textit{k}-means and \textit{k}-medoids.

It will become clear that there are no obstacles in using the hybrid similarities with \textit{k}-medoids; this is not the case for \textit{k}-means. The hybrid similarities define a similarity between two elements in the dataset. However, \textit{k}-means uses similarities between an element and a mean that is not part of the dataset. Since these kinds of similarities have not yet been defined, using the hybrid
similarity with $k$-means is not so straightforward. We propose two possible solutions to work around this problem.

The rest of this chapter is organized as follows. Section 6.2 explains the working of $k$-means and $k$-medoids. Section 6.3 describes how the hybrid similarities could be implemented for $k$-means and $k$-medoids. It also describes the problem that arises when trying to implement it in $k$-means, and proposes two ways around this problem. Section 6.4 describes the experimental setup and Section 6.5 describes the results. Finally, Section 6.6 draws conclusions from the found results.

### 6.2 The Methods

There are many different clustering methods. Amongst them are two important categories: hierarchical clustering and centroid clustering. Hierarchical clustering forms clusters by recursively splitting up or merging already existing clusters. Centroid clustering forms clusters around some centroids that are iteratively adapted to fit the actual clusters in the dataset more every step. The algorithms $k$-means and $k$-medoids are examples of centroid clustering.

#### 6.2.1 $K$-means

Officially, $k$-means was created by MacQueen in 1967 [61]. He was the first to use the term ‘$k$-means’, although Steinhaus already mentioned this idea in 1957 [92]. Originally, the algorithm aims to cluster a dataset with elements that are vectors in an $n$-dimensional space. The Euclidean distance is used to calculate the distance between points in this $n$-dimensional space.

The algorithm starts by choosing $k$ random points in the $n$-dimensional space. These can be considered the prototypes of the clusters. Then, the algorithm iteratively executes an assignment step, followed by an update step. During the assignment step, each element is assigned to the prototype that it is closest to (i.e. the prototype to which the Euclidean distance is the smallest). During the update step, for each cluster, the prototype is recalculated as the mean value of all elements in that cluster at that time. These steps are repeated until the means do not change anymore and all elements stay in the same cluster. More formally:

1. Randomly choose $k$ different points $M_i$ $(i = 1, \ldots, k)$ in the data space; each $M_i$ will serve as a prototype for a cluster $C_i$.
2. Assign each data element to the cluster with the prototype that has the lowest Euclidean distance to it.
3. Recalculate each $M_i$ as the mean of all elements of $C_i$.
4. Repeat steps 2 and 3 until the $M_i$ and $C_i$ no longer change.
Here, step 2 is known as the assignment step and step 3 is known as the update step. An example of the working of this algorithm is shown in Figure 6.1.

Although $k$-means always converges to an optimum, there is a chance that this is just a local optimum. The proof of this involves the fact that the sum of all squared Euclidean distances from one prototype to all the elements in that cluster can only decrease in each update and assignment step. Only a finite number of such decrements is possible (see, for instance, the book by Manning et al. [62]).

The size of $k$ is left open. Much research has been done to determine the best value for $k$, given a specific dataset. One well-known method is the elbow method, which regards the percentage of variance explained by the clusters. This is the ratio of the between-group variance to the total variance. When the amount of clusters is increased, this percentage will begin to rise fast, meaning that an extra cluster will provide much extra information. At some point, the marginal gain will drop, leaving an elbow in the graph. This is the amount of clusters that should be chosen. The first one to come up with this idea was Thorndike in 1953 [95]. There are other methods, like, using information criteria (see, for instance, the work of Goute et al. [30]), or analyzing the kernel matrix (see, for instance, the work of Honarkhah and Caers [37]). If all methods fail, one can always use the rule of thumb to choose $k$ as $\sqrt{\frac{n}{2}}$ (see the book by Mardia et al. [63]).

### 6.2.2 K-medoids

As a variant of $k$-means, Kaufman and Rousseeuw came up with $k$-medoids in 1987 [45]. It differs from $k$-means in that the prototype of a cluster (in this case known as the medoid) is always an element from the data set: in the beginning (step 1) it is a random data element; and during an update step (step 3), $M_i$ becomes the element of $C_i$ for which the overall similarity to all other elements of $C_i$ is maximal.

Using $k$-medoids can be a good alternative for $k$-means. They both have their own benefits and downsides. For instance, $k$-medoids is known to be better than $k$-means when it comes to handling outliers, as shown by Han et al. [32]. This can be seen easily. Consider, for instance, an outlier $v$, which is assigned to a cluster $C$. The element $v$ does not look like any of the other elements in this cluster. Nonetheless, it does not look like any of the elements in the dataset anyway. It still needs to be assigned to a cluster. Therefore, it was assigned to cluster $C$. When the prototype is calculated in the update step during $k$-means, the outlier $v$ has a big influence on the mean value of all the elements in the cluster $C$. To make matters worse, the more $v$ differs from the other elements in $C$, the bigger its influence is on the mean. This is illustrated in Panel 5 of Figure 6.1, where the mean of the top cluster is shifted a bit to the top left because of the outliers in that corner.
With \( k \)-medoids, during the update step, the medoid is chosen that has the highest overall similarity to all other elements in the cluster. The influence that \( v \) has on the sum of similarities for any possible medoid is small, and more importantly, it is roughly the same for any possible medoid in the cluster. If outliers have a big influence, then this is not good for the performance of an algorithm. Hence, \( k \)-medoids performs better in handling outliers than \( k \)-means.

A downside of \( k \)-medoids with respect to \( k \)-means, is that it is more limited in its choice of prototypes, as shown by Kirsten and Wrobel [48]. This can be a problem when clusters have a relatively sparse center, and are close to each other. In this case it is difficult for \( k \)-medoids to find the right medoid for those clusters. This is illustrated in Panel 6 of Figure 6.1. Here, the medoid of the bottom left cluster cannot be chosen somewhere in the center of the cluster. The effect is that some of the elements in this cluster are assigned to the top cluster.

Also, Han et al. showed that \( k \)-medoids is less efficient when handling big data sets [32]. To calculate the new prototype of a cluster in \( k \)-medoids, one needs to calculate the distance from every node in that cluster to every other node in that cluster. However, to calculate the new prototype of a cluster in \( k \)-means, one only needs to calculate the mean of all nodes in it.

6.3 Implementing the Hybrid Similarities

6.3.1 \( K \)-means and the Hybrid Similarities

\( K \)-means was designed for an \( n \)-dimensional space where Euclidean distances are used to determine which elements are close and which are far away. The properties of Euclidean space are used to prove that \( k \)-means always converges. This is an important aspect. Still, \( k \)-means has been used successfully in other environments like radial basis function networks (Moody and Darken [69]), feature classifiers (Huang and Lippman [39]), and astronomy time series (Lei et al. [57]).

Figure 6.1 (preceding page): Example of the working of \( k \)-means. Panel 1 shows the initial situation. Here, the circles are the elements that need to be clustered, and the crosshairs are the randomly chosen prototypes. It is clear to see that there are three clusters and two outliers, and that the initial prototypes are chosen poorly. Panel 2 shows that each element is assigned to the prototype it is closest to. Here, the exact center of a circle determines to which cluster it is assigned. In Panel 3, the prototypes are being updated, and shift towards the centers of the obvious clusters. In Panel 4, the elements are assigned again, but now to the updated prototypes. It is clear to see that the borders of the clusters shift to their desired locations. Panel 5 shows the end result for \( k \)-means. This can be compared with Panel 6 which shows the end result for \( k \)-medoids.
When $k$-means is implemented for a dataset that is an annotated graph, as defined in Section 3.3, it uses a prototype function $P : 2^V \rightarrow A$, that, given a subset of vertices, returns a prototype. Normally, this is the average of the annotations of the vertices in a cluster. Also, it needs a similarity function $S' : A \times V \rightarrow \mathbb{R}$, that calculates the similarity between a prototype and an element. The prototype function is used in the update step, and the similarity function is used in the assignment step.

There must be some aggregate function of the similarities between elements and prototypes (e.g. increasing sum of similarities) to guarantee convergence. During each assignment step and each update step, this function must increase (or decrease) monotonically. When this function does not increase (or decrease) anymore, then, and only then, the algorithm has reached its convergence. Thus, the prototype function and the similarity function must be compatible with each other (i.e. they both must have a similar effect on the aggregate function) to guarantee that the algorithm converges.

The prototype can be defined as an annotation that is not a part of the graph. However, the hybrid similarity measures are functions $S : V \times V \rightarrow \mathbb{R}$, that define a similarity between two elements in a graph, and thus need two vertices of the graph as input. Fortunately, it is possible to rewrite the hybrid similarities in such way that they can be used to calculate the similarity between an element and a prototype. The problem is, that these similarity measures will not be compatible with the previously described prototype measure. We will discuss two ways around this problem. The first one will redefine the prototype measure in such a way that it will be compatible with the slightly altered versions of the hybrid similarity measures. The second one will be a more efficient approximation of the first. We next discuss these two methods.

**$K$-means-NAM:**

**$K$-means with Neighbor Annotation Means**

Recall that the content-based similarity from (3.1) only uses the annotations of the elements. Since the prototype is in the same domain as the annotations, the content-based similarity between a prototype $M$ and an element $v$ can be defined as:

$$S'_{\text{content}} : A \times V \rightarrow \mathbb{R} : S'_{\text{content}}(M, v) = S(M, \alpha(v)) \quad (6.1)$$

The contextual similarity $S_{\text{context}}$ is a symmetrized version of $S_{\text{neighbor}}$. The definition of the latter (see (3.2)) uses for the first element ($v$) only its annotation $\alpha(v)$, not its location in the graph. Thus, the neighbor similarity $S_{\text{neighbor}}$ can be rewritten as a function $S'_{\text{neighbor}} : A \times V \rightarrow \mathbb{R}$ that is defined by:

$$S'_{\text{neighbor}}(M, v) = \frac{\sum_{w \in \mathcal{N}(v)} S_{\text{content}}(M, w)}{|\mathcal{N}(v)|} \quad (6.2)$$
6.3. Implementing the Hybrid Similarities

We can use this asymmetric neighbor similarity instead of the contextual similarity as described in (3.3). Also the combined similarity can then be rewritten as a function $S'_{\text{combined}} : \mathcal{A} \times V \rightarrow \mathbb{R}$ that is defined by:

$$S'_{\text{combined}}(M, v) = c \cdot S_{\text{content}}(M, v) + (1 - c) \cdot S'_{\text{neighbor}}(M, v) \quad (6.3)$$

Now that the hybrid similarities are rewritten, they can also calculate the similarity between an element and an annotation that is not part of the graph. Now, it is possible to calculate the prototype as the average of the annotations of all elements and still have a similarity measure that can be used to assign the elements to the most similar prototype.

This approach causes a new problem though: the proposed prototype measure and similarity measure are not compatible. In the update step, the new prototype is the mean of the cluster element’s annotations, which increases the average content similarity between $M$ and the nodes, but not necessarily the neighbor similarity between $M$ and these nodes. As a result, $k$-means may no longer converge. Consider an element $v$ that has an annotation that differs a lot from the annotations of its neighbors. When the contextual similarity is used, $v$ will be placed in a cluster with a mean that is close to the annotations of the neighbors of $v$. However, when the mean for this cluster is updated, this will be done using the annotation of $v$, instead of the annotations of the neighbors of $v$. This will pull the mean towards $v$’s own annotation, and away from the annotation of its neighbors. The effect could be that the new mean will be far from the annotations of $v$’s neighbors, so the monotonic increase of the aggregate function is no longer guaranteed, and the algorithm may not converge.

To ensure convergence, we need to redefine the prototype measure to be compatible with the similarity measure. For this, we first need to limit the space of possible annotations a little more. From now on we assume that this space allows for a meaningful definition of the basic arithmetic operations: addition, subtraction, multiplication and division. As described in Section 3.3, $\alpha : V \rightarrow \mathcal{A}$ assigns an annotation to a vertex. Now let $\alpha' : V \rightarrow \mathcal{A}$ be a new function that assigns another annotation to a vertex:

$$\alpha'(v) = \frac{\sum_{w \in N(v)} \alpha(w)}{|N(v)|} \quad (6.4)$$

So $\alpha'(v)$ is the mean annotation of all neighbors of $v$. It is clear that calculating the prototype as the average of these new annotations is compatible with the proposed similarity measure.

Following the same reasoning, when using the combined similarity instead of the contextual one, the annotation function $\alpha'' : V \rightarrow \mathcal{A}$ should be used:

$$\alpha''(v) = \frac{\alpha(v) + \alpha'(v)}{2} \quad (6.5)$$
This setup, which we call \textit{k-means-NAM} (\textit{k}-means with Neighbor Annotation Means) is the first solution proposed to enable the use of a \textit{k}-means-like algorithm with a hybrid similarity measure.

\textbf{K-means-NAMA: K-means-NAM Efficiently Approximated}

The previous solution is less efficient than the original \textit{k}-means algorithm. To calculate the similarity between a prototype and an element, the original \textit{k}-means only needs to calculate one content-based similarity. The hybrid similarity for \textit{k}-means-NAM, on the other hand, is based on the neighbor similarity. This similarity needs to calculate the content-based similarity between the prototype and all neighbors of that element. In every assignment step, this needs to be done for every prototype-element combination. Therefore, \textit{k}-means-NAM is several times slower than the original \textit{k}-means.

A more efficient alternative is to average the neighbor annotations themselves, instead of averaging the similarities. This would mean that the algorithm would use the content-based similarity on annotations that are adapted to indirectly include the relational information. Then, the formula to calculate this similarity would be:

$$S_{\text{content}}(M, \frac{\sum_{w \in N(v)} \alpha(w)}{|N(v)|}) = S_{\text{content}}(M, \alpha'(v))$$  \hspace{1cm} (6.6)

These two calculations are mathematically different, and generally do not give the same outcome, but they approximate each other well when the annotations of the neighbors ($\alpha(w)$) are “in the same direction” as $\alpha(v)$. The advantage of this additional approximation is that for each $v$, $\alpha'(v)$ can be computed once in advance, and substituted for $\alpha(v)$, after which the standard \textit{k}-means algorithm can be run. This new annotation $\alpha''$ approximates \textit{k}-means-NAM with the combined distance in the same way that using $\alpha'$ instead of $\alpha$ allows us to approximate \textit{k}-means-NAM with contextual distance. We call this approximation \textit{k}-means-NAMA: \textit{k}-means with Neighbor Annotation Means efficiently Approximated.

\subsection{K-medoids and the Hybrid Similarities}

It is obvious that the hybrid similarity measure can be applied to \textit{k}-medoids without problems. Since the prototype is always an element in the data set (i.e. a node in the graph) the similarity with other elements can be calculated using the hybrid similarities from Section 3.4. To compute the new prototype, one only needs to compute for each element the sum of the similarities to all other elements in that cluster in order to determine which is the largest.
6.4 Experimental Setup

Three methods have been proposed: $k$-means-NAM, $k$-means-NAMA, and $k$-medoids. Each of these methods can be used with one of three similarity measures: the content-based similarity, the contextual similarity and the combined similarity. This leaves nine combinations of a method with a similarity measure. Two of these combinations are the same: $k$-means-NAM with the content-based similarity, and $k$-means-NAMA with the content-based similarity. They both are the original version of $k$-means that we aim to improve.

To evaluate the usefulness of the proposed methods, a few questions need to be answered experimentally:

1. For $k$-means, we have to choose between standard $k$-means with the content-based similarity, or an approximative variant that takes contextual information into account; does the use of contextual information compensate for a possible quality loss for having to use an approximate method?

2. When choosing such an approximate method, is there a difference in performance between $k$-means-NAM and $k$-means-NAMA?

3. One can use $k$-medoids both with content-based or hybrid similarities; does the use of a hybrid similarity yield better results?

4. Which of the method/similarity-combination performs the best?

Note that it is unlikely that answers are obtained that hold for all datasets. See, for instance, the article of Wolpert and Macready [100]. It is likely that on different datasets, a different method will perform best. A proposed method can be considered useful if it has advantages over some (realistic) datasets. In this paper, we evaluate the methods on the Cora dataset by McCallum et al. [65], which is a commonly used benchmark. This is a dataset that contains scientific papers with their abstracts and the citation graph. From this dataset, five subsets are created. A more detailed description can be found in Section 5.2.

For every element in these subsets, a vector can be created which denotes the useful words that appear in the abstracts. This vector will be used as the annotation for the vertices. The cosine of the angle between two vertices is used as the content-based similarity measure. The value for $k$ was varied from 100 to 2. When a clustering is found, it is compared to the clustering as defined by the labels of the vertices in the dataset. This is done by using the $F$-score, the harmonic mean between the precision and the recall of a cluster/label-combination. Keep in mind that $k$-means-NAM(A), used with the content-based similarity, is actually the regular $k$-means algorithm.
CHAPTER 6. *K*-means and *K*-medoids

<table>
<thead>
<tr>
<th>DATASET</th>
<th>CONTEXTUAL SIMILARITY</th>
<th>COMBINED SIMILARITY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSOLUTE DIFFERENCE</td>
<td>AVERAGE DIFFERENCE</td>
</tr>
<tr>
<td>Cora-1</td>
<td>2.4%</td>
<td>-1.2%</td>
</tr>
<tr>
<td>Cora-2</td>
<td>2.4%</td>
<td>+1.0%</td>
</tr>
<tr>
<td>Cora-3</td>
<td>1.6%</td>
<td>-0.7%</td>
</tr>
<tr>
<td>Cora-4</td>
<td>0.9%</td>
<td>+0.2%</td>
</tr>
<tr>
<td>Cora-5</td>
<td>1.3%</td>
<td>+0.4%</td>
</tr>
</tbody>
</table>

Table 6.1: Percentual difference in quality of the found clusters by *k*-means-NAMA compared to the found clusters by *k*-means-NAM.

6.5 Results

6.5.1 *K*-means-NAM vs. *K*-means-NAMA

In Section 6.3.1 we hypothesized that *k*-means-NAMA would get similar results as *k*-means-NAM, but faster. This can be tested by regarding the percentual difference in score between the results with *k*-means-NAMA and *k*-means-NAM. Here, a positive percentage means that *k*-means-NAMA outperformed *k*-means-NAM, and a negative percentage means the opposite. Also the absolute value of these percentages are taken into concern. Table 6.1 shows these results.

First, the averages of all runs in a dataset for the absolute value of these percentages are small, indicating there is not a lot of difference in performance. Second, when the signs of the percentual differences are also taken into account, the differences are even smaller, indicating that one is not overall better than the other. These conclusions hold for both the contextual and the combined similarity.

Table 6.2 shows the average computation time that each method needed to perform one clustering. With the content-based similarity, there is not much difference. This is as expected, since here, *k*-means-NAM and *k*-means-NAMA both boil down to regular *k*-means. It could be said that *k*-means-NAMA is a little slower, but this is explainable by the implementation of the algorithm and it is not considered a significant difference.

When the hybrid similarities are applied, a big difference in time occurs. With respect to the content-based similarity, *k*-means-NAMA takes only 10 to 20 percent more time than the content-based similarity, but *k*-means-NAM takes about four times as much time with the contextual similarity and about five times as much time with the combined similarity. This makes *k*-means-NAMA roughly three times faster than *k*-means-NAM when the contextual similarity is applied, and roughly four times faster when the combined similarity is applied.
6.6. Conclusions

Since there is no real difference in quality between $k$-means-NAM and $k$-means-NAMA, from now on we only consider the results for $k$-means-NAMA.

6.5.2 Quality Improvement

Figure 6.2 shows the results for clustering the subsets CORA-1 and CORA-2, and Figure 6.3 shows the results for clustering the subsets CORA-3, CORA-4, and CORA-5. Table 6.3 shows the best found $F$-scores for all datasets for $k$-medoids and $k$-means-NAMA. Also, it shows the $k$ for which this $F$-score was found. This table clearly shows that using the hybrid similarities improves the $F$-score significantly, as compared to using the content-based similarity. This is the case for both $k$-medoids and $k$-means-NAMA.

When $k$-means-NAMA is compared to $k$-medoids, it shows that $k$-means-NAMA outperforms $k$-medoids. Also, the $k$ for which the found $F$-score is maximal, is much closer to the actual number of classes in the dataset, when using $k$-means-NAMA instead of $k$-medoids.

6.6 Conclusions

We have discussed how a hybrid similarity can be used with centroid-based clustering methods in an annotated graph. Although $k$-means cannot be employed in a straightforward way because the concept of a “mean node” cannot be defined, it can be approximated by two newly proposed methods. These two methods boil down to using approximate similarity measures so that $k$-means becomes applicable. With $k$-medoids, the hybrid similarities can be used without any difficulties.

The main conclusions from this work are:

1. Despite the fact that the hybrid similarities can only be applied on approximate versions of $k$-means, this setting still will lead to better results.

2. Of these approximate versions of $k$-means, there is no significant difference in quality of the found clusterings, but $k$-means-NAMA is much faster than $k$-means-NAM.

3. The hybrid similarities can be used to improve $k$-medoids.

4. When clustering these subsets of Cora, $k$-means-NAMA with the combined similarity leads to the best results. This does not mean that it always will be the best setting for every dataset. It does show, however, that the more sophisticated versions $k$-means-NAM(A) were worth developing, since they work better than the more straightforward approach of applying $k$-medoids.
### Content-based Similarity

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(k)-means-NAM Time</th>
<th>Factor</th>
<th>(k)-means-NAMA Time</th>
<th>Factor</th>
<th>Factor nama/nam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora-1</td>
<td>(2.8 \cdot 10^0)</td>
<td>N/A</td>
<td>(3.3 \cdot 10^0)</td>
<td>N/A</td>
<td>1.19</td>
</tr>
<tr>
<td>Cora-2</td>
<td>(1.8 \cdot 10^1)</td>
<td>N/A</td>
<td>(2.3 \cdot 10^1)</td>
<td>N/A</td>
<td>1.23</td>
</tr>
<tr>
<td>Cora-3</td>
<td>(3.7 \cdot 10^1)</td>
<td>N/A</td>
<td>(4.6 \cdot 10^1)</td>
<td>N/A</td>
<td>1.24</td>
</tr>
<tr>
<td>Cora-4</td>
<td>(1.3 \cdot 10^2)</td>
<td>N/A</td>
<td>(1.6 \cdot 10^2)</td>
<td>N/A</td>
<td>1.20</td>
</tr>
<tr>
<td>Cora-5</td>
<td>(4.2 \cdot 10^2)</td>
<td>N/A</td>
<td>(5.0 \cdot 10^2)</td>
<td>N/A</td>
<td>1.18</td>
</tr>
</tbody>
</table>

### Contextual Similarity

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(k)-means-NAM Time</th>
<th>Factor</th>
<th>(k)-means-NAMA Time</th>
<th>Factor</th>
<th>Factor nama/nam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora-1</td>
<td>(1.2 \cdot 10^1)</td>
<td>4.17</td>
<td>(4.2 \cdot 10^0)</td>
<td>1.28</td>
<td>0.37</td>
</tr>
<tr>
<td>Cora-2</td>
<td>(5.8 \cdot 10^1)</td>
<td>3.13</td>
<td>(2.3 \cdot 10^1)</td>
<td>1.02</td>
<td>0.40</td>
</tr>
<tr>
<td>Cora-3</td>
<td>(1.5 \cdot 10^2)</td>
<td>4.12</td>
<td>(5.0 \cdot 10^1)</td>
<td>1.08</td>
<td>0.33</td>
</tr>
<tr>
<td>Cora-4</td>
<td>(5.0 \cdot 10^2)</td>
<td>3.71</td>
<td>(1.8 \cdot 10^2)</td>
<td>1.12</td>
<td>0.36</td>
</tr>
<tr>
<td>Cora-5</td>
<td>(1.7 \cdot 10^3)</td>
<td>4.09</td>
<td>(6.2 \cdot 10^2)</td>
<td>1.23</td>
<td>0.36</td>
</tr>
</tbody>
</table>

### Combined Similarity

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(k)-means-NAM Time</th>
<th>Factor</th>
<th>(k)-means-NAMA Time</th>
<th>Factor</th>
<th>Factor nama/nam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora-1</td>
<td>(1.5 \cdot 10^1)</td>
<td>5.56</td>
<td>(4.2 \cdot 10^0)</td>
<td>1.28</td>
<td>0.28</td>
</tr>
<tr>
<td>Cora-2</td>
<td>(8.1 \cdot 10^1)</td>
<td>4.38</td>
<td>(2.2 \cdot 10^1)</td>
<td>0.98</td>
<td>0.28</td>
</tr>
<tr>
<td>Cora-3</td>
<td>(1.9 \cdot 10^2)</td>
<td>5.05</td>
<td>(5.0 \cdot 10^1)</td>
<td>1.08</td>
<td>0.27</td>
</tr>
<tr>
<td>Cora-4</td>
<td>(6.5 \cdot 10^2)</td>
<td>4.86</td>
<td>(1.7 \cdot 10^2)</td>
<td>1.05</td>
<td>0.26</td>
</tr>
<tr>
<td>Cora-5</td>
<td>(2.1 \cdot 10^3)</td>
<td>4.91</td>
<td>(5.4 \cdot 10^2)</td>
<td>1.08</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Table 6.2: CPU time for \(k\)-means-NAM and \(k\)-means-NAMA to do one clustering (on a Intel® Quad Core™ 2, 2.4GHz with 4 Gb memory), for the content-based, contextual, and combined similarities. ‘Dataset’ denotes the used dataset; ‘Time’ denotes the time in seconds; ‘Factor’ in the third and fifth column denotes how much more time it takes to do a clustering using that similarity with regards to the content-based similarity; ‘Factor nama/nam’ in the sixth column denotes the factor of the time it takes \(k\)-means-NAMA to compute a clustering with regards to \(k\)-means-NAM.
6.6. Conclusions

Figure 6.2: Average $F$-score for the resulting clusterings for different values of $k$, for $k$-medoids and $k$-means-NAMA on Cora-1 (top) and Cora-2 (bottom).
Figure 6.3: Average $F$-score for the resulting clusterings for different values of $k$, for $k$-medoids and $k$-means-NAMA on respectively CORA-3 (top), CORA-4 (middle), and CORA-5 (bottom).
Table 6.3: Best $F$-scores found for $k$-medoids and $k$-means-NAMA, for the different similarities and datasets. Also, the appropriate $k$ is shown. This can be compared to $|L|$ (i.e. the amount of labels in the dataset).