

Technische Universität Dresden

Faculty of Computer Science  
Institute for Artificial Intelligence  
Applied Knowledge Representation and Reasoning Group

An Empirical Study of K-Means Initialization  
Methods for Document Clustering

Seminar Paper

Submitted by: Marco Zimmerling (2944184)

Advisor: Doz. Dr.-Ing. habil. Uwe Petersohn

Dresden, 25. September 2008



# Abstract

Everyday vast amounts of documents, e-mails, and web pages are generated. In order to handle these data, automatic techniques such as document clustering are needed. The **k-means** method is a clustering technique widely used in practice because of its simplicity and empirical speed. In this paper, the basic **k-means** algorithm is augmented with two special initialization techniques that aim at improving both the accuracy and the speed of **k-means**. Experiments are run on two popular document collections. The results vary substantially between the document sets and the number of clusters, but overall, only marginal improvements in accuracy and a decline in speed are observed.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Representation of Text Documents</b>	<b>3</b>
2.1	Document Preprocessing	3
2.2	The Vector Space Model	5
<b>3</b>	<b>K-Means and Alternative Methods</b>	<b>7</b>
3.1	Preliminaries	7
3.2	The Basic <b>k-means</b> Algorithm	8
3.3	Initialization Techniques	9
3.4	Alternative Clustering Methods	10
<b>4</b>	<b>Evaluation</b>	<b>11</b>
4.1	Document Sets	11
4.2	Metrics	12
4.3	Results	14
4.4	Discussion	17
<b>5</b>	<b>Conclusions</b>	<b>19</b>
	<b>List of Figures</b>	<b>21</b>
	<b>List of Tables</b>	<b>23</b>
	<b>Bibliography</b>	<b>25</b>



## Chapter 1

# Introduction

Everyday vast amounts of documents, e-mails, and web pages are generated. These unstructured data are usually stored on web servers, file servers, or personal workstations and need to be organized into a logical structure for later use. Only by an automatic approach without human assistance or preparation work, this task can be accomplished efficiently.

Document clustering is an automatic technique that groups text documents into cluster. The overall goal is to group similar documents into the same cluster and dissimilar documents into different clusters. Afterwards, the superimposed cluster structure can be used, for instance, to generate complete and precise responses to users querying information contained in the documents. Document clustering has been applied successfully in a wide range of areas including web mining, search engines, information retrieval, and topological analysis.

One of the most popular clustering techniques is the **k-means** method [12]. It belongs to the class of partitioning algorithms that divide a set of data points—for example, vectors in  $\mathbb{R}^d$  representing text documents—into  $k$  clusters. A clustering is implicitly defined by choosing  $k$  centroids (cluster centers) and subsequently assigning each data point to the nearest centroid. The goal of **k-means** is to choose  $k$  centroids so as to minimize the sum of the squared distances between each data point and its closest centroid.

Solving this problem exactly is NP-hard [5], but Lloyd [11] proposed a local search algorithm that is still very widely used today. Indeed, a recent survey of data mining techniques [3] states that it “is by far the most popular clustering algorithm used in scientific and industrial applications.”

Usually referred to simply as **k-means**, Lloyd’s algorithm begins with  $k$  arbitrary centroids, typically chosen uniformly at random from the data points. Each point is then assigned to the nearest centroid, and each centroid is recomputed as the average

of all points assigned to it. These two steps (assignment and centroid calculation) are repeated until the process stabilizes.

Because of its simplicity and speed, **k-means** is very appealing in practice. As for the accuracy of the generated clusterings, however, the algorithm is only guaranteed to find a local optimum. The final clusterings depend highly on the placement of the initial centroids, but even with standard randomized seeding techniques, the accuracy of the results can be arbitrarily bad with high probability.

In this paper, the **k-means** method is used for document clustering. In particular, the basic **k-means** algorithm with uniform, random seeding is compared with two other variants that choose the initial centroids according to a specific scheme—careful initialization may lead to more accurate clusterings and fewer iterations to reach the local optimum. Using two popular document collections, experiments are carried out measuring the improvement in both accuracy and running time of the three approaches. While the results vary substantially between the document sets and the number of clusters, overall, only marginal improvements in accuracy and even a decline in speed are observed.

The paper is organized as follows. Chapter 2 shows how the document collections were preprocessed and represented as sets of term vectors. The basic **k-means** algorithm and the initialization methods considered in this work are described in Chapter 3. This includes also a discussion of the cosine measure, which is used to quantify the similarity between documents, and a brief overview of hierarchical clustering and clustering based on frequent itemsets. Chapter 4 describes the document collections used in the experiments, the evaluation method including the used metrics, and finally presents and discusses the experimental results. Some concluding remarks on the development process are given in Chapter 5.

## Chapter 2

# Representation of Text Documents

The **k-means** algorithm requires the representation of text documents as data points in  $\mathbb{R}^d$ . The necessary transformation steps and the final document representation are described in this section.

## 2.1 Document Preprocessing

Before a set of text documents can be represented in a meaningful way, several preprocessing steps are necessary. They aim at converting the documents into a standard format by removing all kinds of clutter, which would otherwise degrade the performance of the clustering algorithms. The preprocessing steps carried out in this work are described below in chronological order.

**Conversion into Uniform File Format** The two document sets used for the experimental evaluation were available in two different file formats. (See Section 4.1 for a detailed description of the document sets.) The documents of the *Reuters* collection were available as 22 SGML files, where each of the first 21 files contained 1000 documents, and the last file contained 578 documents. The documents in the *Classic3* collection were available as three text files containing 1033, 1400, and 1460 documents. All three files had a common but non-standard structure.

To avoid the effort of dealing with many files and two different file formats, the files of a collection were combined into a single file. Furthermore, the two resulting files were converted into XML. This allowed the use of standard XML programming libraries which eased further processing of the document collections. During the conversion, unnecessary data such as the documents' authors and titles were skipped

and not included in the XML files. This reduced the size of the document collections considerably.<sup>1</sup>

**Basic Text Preparation** The first step in preparing the documents involved the removal of digits and punctuation characters, the normalization of white spaces such that all words would be separated by exactly one space, and the conversion to lowercase letters. This further reduced the amount of noise in the documents.

**Elimination of Stopwords** Stopwords are words that do not contribute to the essence of a document—examples include “a”, “and”, “for”, and “the”. They appear in large numbers in all kinds of texts independent of the subject, and are therefore useless for judging whether documents are similar in content. A list of 571 English stopwords<sup>2</sup> was thus eliminated from the documents.

**Elimination of Function Words** A word’s document frequency is the number of documents in a document set in which the word occurs. Words with low or high document frequency are known as function words, where the exact cut-off thresholds are determined using heuristics or information-theoretic criteria [15]. Eliminating function words removes little or no information while speeding up the computation. In this work, words appearing in less than 0.2% of the documents in a collection were considered as low-frequency; words appearing in more than 15% of the documents in a collection were considered as high-frequency [4]. Such function words were eliminated.

**Stemming** Stemming is the reduction of a word to its root or stem, thereby eliminating plurals, tenses, prefixes, and suffixes. Words of the same stem but with different suffixes, for example, are considered as equal after the stemming. This transformation step is important for document clustering, because the similarity of documents should be determined based on their meaning rather than on their syntactical appearance. A well-known stemming algorithm for words in English is the Porter stemmer [13], which was applied to the documents in a final preprocessing step.<sup>3</sup>

---

<sup>1</sup>For the *Reuters* collection, in particular, a reduction to roughly 25% of the original size was achieved.

<sup>2</sup>The list is available at <ftp://ftp.cs.cornell.edu/pub/smart/english.stop>.

<sup>3</sup>To avoid variations in the functionality of the Porter stemmer, the implementations available at <http://tartarus.org/~martin/PorterStemmer/> serve as the definitive version of the algorithm.

**Elimination of Short Documents** Documents with less than three remaining terms were removed from the document collections.

## 2.2 The Vector Space Model

After preprocessing the documents as described above, *term frequency* and *document frequency* were determined for each term. Term frequency counts the occurrences of term  $j$  in document  $i$  and is denoted with  $f_j^i$ . Document frequency, denoted  $d_j$ , is the number of documents that contain term  $j$ . If there are  $t$  unique terms in document set  $\mathcal{D}$ , document  $i$  is represented as a  $t$ -dimensional vector

$$\mathbf{d}_i = (w_1^i, w_2^i, \dots, w_j^i, \dots, w_t^i)^{\mathbf{T}}, \quad (2.1)$$

where  $w_j^i$  is the weight of the  $j$ th term. This representation of documents is called the *vector space model* [16]. The components of each document vector, the term weights, are appropriate combinations of term frequency and document frequency.

According to the *normalized term frequency–inverse document frequency* scheme, the weight of the  $j$ th term is given by

$$w_j^i = f_j^i idf_j nc_j, \quad (2.2)$$

where  $idf_j$  is the *inverse document frequency* of term  $j$ , and  $nc_j$  is the *normalization component*. First, we see that a term receives a large weight if it occurs in a document at high frequency. Term frequency expresses the relative importance of a term in a particular document and can therefore be regarded as a local parameter. Document frequency, on the other hand, can be regarded as a global parameter, because it captures the overall importance of a term in the entire document set.

The inverse document frequency of term  $j$  is defined as

$$idf_j = \log_2 \left( \frac{|\mathcal{D}|}{d_j} \right), \quad (2.3)$$

emphasizing rare terms that appear only in few documents and mitigating terms that appear in many documents—this makes sense since too common terms are not useful in distinguishing documents from each other. All in all, we can say that the largest weights are assigned to those terms that appear with high frequency in individual documents but are at the same time rare in the collection as a whole.

Finally, each document vector is normalized to be of unit length, that is,

$$nc_j = \left( \sum_{j=1}^t (f_j^i idf_j)^2 \right)^{-1/2}. \quad (2.4)$$

This accounts for documents of different lengths, avoiding a bias towards very long documents.

## Chapter 3

# K-Means and Alternative Methods

### 3.1 Preliminaries

Every document clustering algorithm requires a measure to determine the similarity of documents. Based on the representation of documents as vectors, the most widely used method is the *cosine measure* [14], which defines similarity of documents  $i$  and  $j$  as the cosine of the angle between the corresponding document vectors,

$$\text{sim}(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i \cdot \mathbf{d}_j}{\|\mathbf{d}_i\| \|\mathbf{d}_j\|}. \quad (3.1)$$

For document vectors of unit length this simplifies to  $\text{sim}(\mathbf{d}_i, \mathbf{d}_j) = \mathbf{d}_i \cdot \mathbf{d}_j$ . The cosine measure ranges between 0 and 1, where larger values indicate that the documents are more similar.

The clustering algorithm should group similar documents in the same cluster and dissimilar documents in different clusters, or, in other words, achieve high *intra-cluster similarity* and low *inter-cluster similarity*. To quantify intra-cluster similarity, let us consider the *centroid* of cluster  $\mathcal{C}$ ,

$$\mathbf{c} = \frac{1}{|\mathcal{C}|} \sum_{\mathbf{d} \in \mathcal{C}} \mathbf{d}, \quad (3.2)$$

which is, intuitively speaking, the mean of all documents in the cluster. The intra-cluster similarity of  $\mathcal{C}$  can now be defined as the sum of the squared similarities of all documents in  $\mathcal{C}$  and the centroid,

$$\text{sim}(\mathcal{C}) = \sum_{\mathbf{d} \in \mathcal{C}} \text{sim}(\mathbf{d}, \mathbf{c})^2. \quad (3.3)$$

The higher the  $\text{sim}(\mathcal{C})$  value, the more similar are the documents in cluster  $\mathcal{C}$ .

## 3.2 The Basic k-means Algorithm

The **k-means** method is the most popular clustering technique in practice because of its simplicity and speed. It is a general method that partitions any set of data points in  $\mathbb{R}^d$  into  $k$  clusters. In our documents' world, the goal is to identify  $k$  clusters in the set of documents such that the sum of intra-cluster similarities (3.3) is maximal.

For the following discussion, however, we adopt an equivalent formulation of the **k-means** problem. Given an integer  $k$  and a set of document vectors  $\mathcal{D}$ , choose  $k$  centroids  $\bar{\mathcal{C}}$  so as to maximize  $\phi$ , the sum of the squared similarities of each document vector and its most similar centroid,

$$\phi = \sum_{\mathbf{d} \in \mathcal{D}} \max_{\mathbf{c} \in \bar{\mathcal{C}}} \text{sim}(\mathbf{d}, \mathbf{c})^2. \quad (3.4)$$

Choosing these centroids implicitly defines a clustering—for each centroid, we set one cluster to be the set of document vectors that are more similar to that centroid than to any other.

Solving this problem exactly is NP-hard, even with just two clusters [5]. But Lloyd [11] proposed a local search algorithm that attempts to improve an arbitrary **k-means** clustering. It works as follows.

1. Arbitrarily choose  $k$  initial centroids  $\bar{\mathcal{C}} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k\}$ .
2. For each  $i \in \{1, 2, \dots, k\}$ , set cluster  $\mathcal{C}_i$  to be the set of document vectors in  $\mathcal{D}$  that are more similar to  $\mathbf{c}_i$  than they are to  $\mathbf{c}_j$  for all  $j \neq i$ .
3. For each  $i \in \{1, 2, \dots, k\}$ , set  $\mathbf{c}_i$  to be the centroid of  $\mathcal{C}_i$  according to (3.2).
4. Repeat Steps 2 and 3 until  $\mathcal{D}$  no longer changes.

It is standard practice to choose the initial centroids uniformly at random from  $\mathcal{D}$ . In Step 2, ties may be broken arbitrarily as long as the method is consistent.

The **k-means** algorithm is attractive in practice because it is simple and requires usually very few iterations. Unfortunately, it is guaranteed only to find a local optimum, which can often be quite poor. Both the runtime of **k-means** and the accuracy of the produced clusterings, in fact, depend on the choice of the initial centroids. Therefore, many initialization methods have been developed that aim at improving **k-means** by choosing more suitable initial centroids, that is, to find more accurate clusterings in fewer iterations. Two of these methods are presented in the next section.

### 3.3 Initialization Techniques

Arthur and Vassilvitskii proposed the **k-means++** algorithm, which uses a randomized seeding technique [2]. At any given time, let  $S(\mathbf{d})$  denote the largest similarity value (cosine measure) of document vector  $\mathbf{d}$  and the most similar centroid already chosen. The **k-means++** algorithm can then be defined as follows.

- 1a. Choose an initial centroid  $\mathbf{c}_1$  uniformly at random from  $\mathcal{D}$ .
- 1b. Choose the next centroid  $\mathbf{c}_i$ , selecting  $\mathbf{c}_i = \mathbf{d}' \in \mathcal{D}$  with probability

$$\frac{S(\mathbf{d}')^2}{\sum_{\mathbf{d} \in \mathcal{D}} S(\mathbf{d})^2}. \quad (3.5)$$

- 1c. Repeat Step 1b until  $k$  centroids are chosen.
- 2.–4. Proceed as with the standard **k-means** algorithm.

In Step 1b, the next centroid is chosen with probability proportional to the centroid's contribution to the overall similarity. The basic idea is to construct clusters of large intra-cluster similarity beforehand, thus making it more likely that the actual **k-means** algorithm will find a good solution in short time.

By contrast, the initialization technique proposed by Katsavounidis et al. aims at low inter-cluster similarities among initial clusters [7]. Their method, called **kkz** in the following, has the same structure as **k-means++** but uses only the pure pairwise similarities. In particular, in Step 1b, the **kkz** method chooses the document vector with the smallest  $S(\cdot)$  value as the next centroid, that is,

$$\mathbf{c}_i = \underset{\mathbf{d} \in \mathcal{D}}{\operatorname{argmin}} S(\mathbf{d}), \quad (3.6)$$

where  $S(\cdot)$  is defined as above. **kkz** chooses always the document vector that is most different from the existing centroids, separating the initial centroids as much as possible from each other. Compared with **k-means++**, the initialization is deterministic after choosing the first centroid.

**k-means++** as well as **kkz** are user-friendly extensions of the basic K-Means algorithm, as none of them introduces additional parameters. Also, as argued in the original papers, the improvements in accuracy of the final **k-means** clusterings justify the computational overhead. For these reasons, **k-means++** and **kkz** were considered in this work.

## 3.4 Alternative Clustering Methods

Hierarchical methods are another class of algorithms widely used in the area of document clustering. Hierarchical clustering builds a tree of clusters, also known as a dendrogram, rather than a flat cluster structure. Every cluster node contains child clusters, and sibling clusters partition the points covered by their common parent. This allows exploring data on different levels of granularity.

Hierarchical clustering methods are categorized into agglomerative (bottom-up) and divisive (top-down) [8]. An agglomerative clustering starts with one-point (singleton) clusters and recursively merges two or more most appropriate clusters. A divisive clustering starts with one cluster of all data points and recursively splits the most appropriate cluster. The process continues until a stopping criterion is satisfied.

Advantages of this method are that a number of clusters need not be supplied in advance, and that the resulting cluster hierarchy is *browsable*. On the other hand, hierarchical algorithms do not scale (at each merge phase, similarities must be compared), and so are not appropriate for real-time applications or large data sets. Furthermore, it was shown that partitional algorithms perform better than hierarchical ones [17]. In the cases where a hierarchy is desired, superior results have been achieved by using partitional approaches at each level of the hierarchy [19].

The idea of *frequent itemsets* is used in [6] to define a cluster, to organize the cluster hierarchy, and to reduce the dimensionality of the document sets. The intuition is that there are some frequent itemsets for each cluster in the document set, and different clusters share few frequent itemsets. A frequent itemset is a set of words that occur together in some minimum fraction of documents in a cluster. Therefore, a frequent itemset describes something common to many documents in a cluster. It is shown that this approach outperforms the best known partitioning and hierarchical algorithms.

## Chapter 4

# Evaluation

In order to evaluate the different initialization techniques in practice, they were implemented and tested in C++. In this section, the empirical results of these experiments are discussed.

### 4.1 Document Sets

The performance of `k-means`, `kkz`, and `k-means++` was evaluated on two document sets widely used in document clustering research [4, 9]. These document sets differ particularly in the number of classes and the range of the class sizes. Each document was assigned a single topic, called *natural class* below. This information is used to measure the accuracy of the clustering result, but it was hidden from the algorithms during cluster construction.

The first document set, *Classic3*, contains 3891 documents and was obtained by merging the Medlars, Cisi, and Cranfield collections.<sup>1</sup> Medlars consists of 1033 abstracts from medical journals, Cisi consists of 1460 abstracts from information retrieval papers, and Cranfield consists of 1398 abstracts from aeronautical systems papers. Based on the documents' original membership in one of these collections, the documents were assigned to natural class “med,” “cisi,” or “cran.”

The *Classic3* collection was preprocessed as outlined in Section 2.1. After removing common stopwords, the collection contained 23667 unique words from which 19554 low-frequency words, appearing in less than 0.2% of the documents (about 8 documents), and 6 high-frequency words, appearing in more than 15% of the documents (about 584 documents), were eliminated. After the stemming, 2588 unique terms were left in the *Classic3* document set. Because all documents contained at least three terms, none had to be removed from the collection. Then

---

<sup>1</sup>All three document collections are available at <ftp://ftp.cs.cornell.edu/pub/smart/>.

**Table 4.1:** Characteristics of the preprocessed document sets used in the experiments.

Data set	Documents	Natural classes	Class sizes	Unique terms
<i>Classic3</i>	3891	3	1033 – 1460	2588
<i>Reuters</i>	8193	65	1 – 3274	1984

3891 document vectors of dimension 2588 were created (see Section 2.2). On average, each document vector contained only 40 nonzero components and was about 98.5% sparse.

The *Reuters* document set is a subset of the Reuters-21578 text categorization test collection, Distribution 1.0.<sup>2</sup> The Reuters-21578 collection contains 21578 stories which appeared on the Reuters newswire in 1987. All stories were indexed by experts with categories from five category sets (exchanges, organizations, people, places, topics). For the *Reuters* document set, only the documents that had been indexed with exactly one topic were selected. These 8654 documents were classified into natural classes according to their assigned topic.

The *Reuters* collection was also preprocessed as outlined in Section 2.1. After removing common stopwords, the collection contained 26422 unique words from which 23533 low-frequency words, appearing in less than 0.2% of the documents (about 17 documents), and 13 high-frequency words, appearing in more than 15% of the documents (about 1298 documents), were eliminated. After the final stemming, 1984 unique terms were left in the *Reuters* document set. From the initial 8654 documents, 461 documents with less than three remaining terms were removed from the collection; these documents, for the most part, were made up of only a few words, abbreviations, or numbers even before preprocessing. Finally, 8193 document vectors of dimension 1984 were created, which had on average only 31 nonzero components and were roughly 98.5% sparse.

Table 4.1 summarizes the main characteristics of the document collections after preprocessing.

## 4.2 Metrics

For each document set 3, 10, 25, 50, and 65 clusters were tested. (Note that *Classic3* consists of 3 natural classes, *Reuters* of 65 natural classes.) Since all three algorithms use randomized seeding techniques, 20 trials were run for each case.

<sup>2</sup>The Reuters-21578 document collection, Distribution 1.0, is available at <http://www.daviddlewis.com/resources/testcollections/reuters21578/>.

The overall F-measure was used for measuring the accuracy of the generated clusterings, for which the maximum and the average values are reported. Furthermore, the mean number of iterations and the mean total running times are reported to assess the empirical speed of the algorithms. The implementations are standard with no special optimizations.

The used measures are described in more detail in the next two sections.

**Measuring Accuracy** The accuracy of a clustering can be measured by comparing the produced clusters to the natural classes. This is the basic idea of the *F-measure* [18], a standard external measure<sup>3</sup> for the evaluation of both flat and hierarchical clustering structures. It combines the ideas of *recall* and *precision* from information retrieval, where these parameters are often used to measure how accurate a system responds to user queries. Recall is the ratio of relevant items actually retrieved; precision is the ratio of retrieved items actually relevant.

We can adapt recall and precision to the world of document clustering by treating each cluster as if it were the result of a query and each natural class as if it were the relevant set of documents for a query. Suppose that  $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$  denotes the clustering of a document set, and that the “correct” set of natural classes is given by  $\mathcal{C}^* = \{\mathcal{C}_1^*, \mathcal{C}_2^*, \dots, \mathcal{C}_l^*\}$ . The recall of cluster  $\mathcal{C}_i$  with respect to natural class  $\mathcal{C}_j^*$  is defined as

$$r(\mathcal{C}_i, \mathcal{C}_j^*) = \frac{|\mathcal{C}_i \cap \mathcal{C}_j^*|}{|\mathcal{C}_j^*|}. \quad (4.1)$$

The precision of cluster  $\mathcal{C}_i$  with respect to natural class  $\mathcal{C}_j^*$  is defined as

$$p(\mathcal{C}_i, \mathcal{C}_j^*) = \frac{|\mathcal{C}_i \cap \mathcal{C}_j^*|}{|\mathcal{C}_i|}. \quad (4.2)$$

For a cluster, then, recall reflects the ratio of documents from some natural class that are captured by the cluster, and precision reflects the ratio of documents in the cluster that belong to some natural class. The F-measure combines recall and precision in the following form,<sup>4</sup>

$$f(\mathcal{C}_i, \mathcal{C}_j^*) = \frac{2 r(\mathcal{C}_i, \mathcal{C}_j^*) p(\mathcal{C}_i, \mathcal{C}_j^*)}{r(\mathcal{C}_i, \mathcal{C}_j^*) + p(\mathcal{C}_i, \mathcal{C}_j^*)}. \quad (4.3)$$

---

<sup>3</sup>A quality measure is called *external* if it uses knowledge that is not available from the clustering solution itself. As for the F-Measure, these are the natural classes of documents.

<sup>4</sup>This is also known as the  $F_1$  measure, because recall and precision are equally weighted. In van Rijsbergen’s general  $F_\alpha$ -measure [18], either of them has weight  $\alpha$  while the other has weight 1.

Intuitively speaking, it measures the quality of cluster  $\mathcal{C}_i$  in describing the natural class  $\mathcal{C}_j^*$ ; a larger F-measure indicates a higher quality.

Let  $\hat{\mathcal{C}}_i$  denote the cluster that best describes natural class  $\mathcal{C}_j^*$ , that is,  $\hat{\mathcal{C}}_i$  has the largest F-measure among all clusters in  $\mathcal{C}$  with respect to  $\mathcal{C}_j^*$ :

$$f(\hat{\mathcal{C}}_i, \mathcal{C}_j^*) = \max_{\mathcal{C}_i \in \mathcal{C}} \{f(\mathcal{C}_i, \mathcal{C}_j^*)\}. \quad (4.4)$$

To measure the overall quality of the clustering  $\mathcal{C}$ , we use the weighted sum of these maximum F-measures for all natural classes. This measure is called the *overall F-measure* of  $\mathcal{C}$  and is given by

$$f(\mathcal{C}) = \sum_{\mathcal{C}_j^* \in \mathcal{C}^*} \frac{f(\hat{\mathcal{C}}_i, \mathcal{C}_j^*) |\mathcal{C}_j^*|}{|\mathcal{D}|}, \quad (4.5)$$

where  $|\mathcal{D}|$  denotes the number of documents in the document set. The overall F-measure ranges between 0 and 1, whereas larger values indicate a higher accuracy of the clustering.

**Measuring Efficiency** Besides accuracy, it is worthwhile measuring the efficiency at which the clustering result was generated. The number of **k-means** iterations required to calculate a solution is one measure of efficiency. Given a set of initial centroids, this characterizes the quality of the internal workings of the algorithm. Ultimately, however, the empirical running time is what really matters, including the time to choose the initial centroids. Only if the time needed to determine the initial centroids does not eat up the gain in speed of the actual clustering, a sophisticated initialization method will be beneficial.

During the experiments the number of iterations as well as the individual running times of initialization and clustering were recorded.

## 4.3 Results

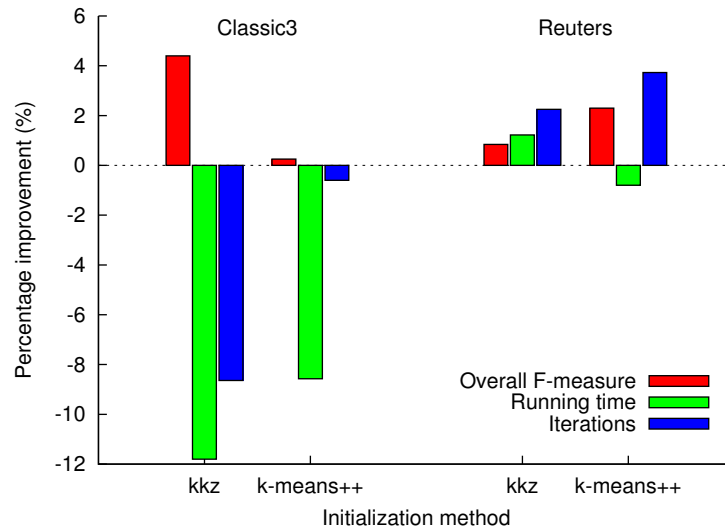
The results for **k-means**, **kkz**, and **k-means++** are displayed in Tables 4.2 through 4.5, showing the absolute results for **k-means** and the percentage improvements for **kkz** and **k-means++**. In each case the best value is printed in bold. The percentage improvement in the overall F-measure was calculated as follows,

$$100\% \cdot \left( \frac{\text{kkz or k-means++ value}}{\text{k-means value}} - 1 \right), \quad (4.6)$$

and in the running time and the number of iterations,

$$100\% \cdot \left( 1 - \frac{\text{kkz or k-means++ value}}{\text{k-means value}} \right). \quad (4.7)$$

Overall, the special initialization methods of **kkz** and **k-means** brought marginal improvements in accuracy and led to longer total running times and more iterations. The results, however, varied considerably between the document sets and the cluster sizes.



**Figure 4.1:** Percentage improvements of **kkz** and **k-means++** over **k-means**, averaged over all cluster sizes, on the *Classic3* and *Reuters* document sets.

Figure 4.1 shows the average percentage improvements of **kkz** and **k-means++**. While **kkz** achieved on *Classic3* the largest F-measure scores, its total running time increased, as it took more iterations to determine the final clustering. The largest decline in speed was at 10 clusters, where both **kkz** and **k-means++** were about 30% slower than **k-means**. The shorter clustering times of **k-means++** at 3, 50, and 65 clusters failed to outweigh the time spent in choosing the initial centroids. No improvement in accuracy was observed for **k-means++** on *Classic3*.

On the *Reuters* collection, **k-means++** performed better than **kkz**, both by achieving a larger F-measure value and also by reducing the number of iterations. In the end, however, the long initialization times of **k-means++**, which were in general twice as long as those of **kkz**, led to a longer total running time. At 50 and 65 clusters, **kkz** and **k-means++** showed the highest improvement in accuracy. But at 65 clusters, the standard seeding was much faster.

**Table 4.2:** Accuracy results on the *Classic3* document set. For **k-means**, the absolute overall F-measure is listed. For **kkz** and **k-means++**, the percentage improvement over **k-means** (4.6) is listed.

$k$	Average F-measure			Maximum F-measure		
	k-means	kkz	k-means++	k-means	kkz	k-means++
3	0.95	-3.48%	<b>2.92%</b>	0.99	<b>0.03%</b>	<b>0.03%</b>
10	0.57	<b>6.15%</b>	1.29%	0.63	<b>10.37%</b>	7.55%
25	0.35	<b>4.98%</b>	0.15%	0.40	0.90%	<b>1.40%</b>
50	0.23	<b>6.87%</b>	-2.93%	0.25	<b>13.67%</b>	8.95%
65	0.20	<b>7.48%</b>	-0.16%	0.23	<b>9.00%</b>	-1.47%

**Table 4.3:** Efficiency results on the *Classic3* document set. For **k-means**, the total running time  $T$  in seconds and the number of iterations  $I$  are listed. For **kkz** and **k-means++**, the percentage improvement over **k-means** (4.7) is listed.

$k$	Average $T$			Average $I$		
	k-means	kkz	k-means++	k-means	kkz	k-means++
3	3.70	-2.70%	<b>2.70%</b>	10.80	-1.39%	<b>12.50%</b>
10	<b>21.90</b>	-30.59%	-26.26%	<b>20.40</b>	-30.39%	-21.08%
25	<b>61.15</b>	-10.22%	-16.93%	<b>23.25</b>	-6.67%	-8.39%
50	112.60	-10.97%	<b>1.60%</b>	22.00	-4.77%	<b>8.86%</b>
65	<b>139.95</b>	-4.50%	-3.97%	21.45	0.00%	<b>5.13%</b>

**Table 4.4:** Accuracy results on the *Reuters* document set. For **k-means**, the absolute overall F-measure is listed. For **kkz** and **k-means++**, the percentage improvement over **k-means** (4.6) is listed.

$k$	Average F-measure			Maximum F-measure		
	k-means	kkz	k-means++	k-means	kkz	k-means++
3	0.46	-1.20%	<b>3.90%</b>	0.57	-0.02%	<b>0.34%</b>
10	<b>0.44</b>	-8.66%	-1.54%	<b>0.51</b>	-8.56%	-2.04%
25	<b>0.38</b>	-4.26%	-3.73%	<b>0.43</b>	-9.05%	-5.63%
50	0.32	<b>8.85%</b>	6.91%	0.35	<b>7.74%</b>	3.07%
65	0.31	<b>9.47%</b>	5.96%	0.34	<b>5.34%</b>	1.63%

**Table 4.5:** Efficiency results on the *Reuters* document set. For **k-means**, the total running time  $T$  in seconds and the number of iterations  $I$  are listed. For **kkz** and **k-means++**, the percentage improvement over **k-means** (4.7) is listed.

$k$	Average $T$			Average $I$		
	k-means	kkz	k-means++	k-means	kkz	k-means++
3	8.90	<b>11.24%</b>	-3.37%	16.15	<b>14.24%</b>	3.10%
10	46.40	-1.08%	<b>0.22%</b>	28.25	0.35%	<b>4.60%</b>
25	141.15	0.46%	<b>9.78%</b>	32.80	-2.74%	<b>11.74%</b>
50	262.50	<b>5.26%</b>	3.77%	30.85	5.02%	<b>6.00%</b>
65	272.45	-9.76%	-14.39%	<b>25.75</b>	-5.63%	-6.80%

## 4.4 Discussion

`kkz` and `k-means++` failed to substantially improve on the standard algorithm probably due to the nature of the data sets. In document clustering, one has to deal with sparse data points of high dimensionality. In the original `k-means++` paper, the algorithm was tested on four data sets with dimensionalities varying from 10 to 58. Average improvements in accuracy between 22% and 99% and in running time between 43% and 80% were achieved, with increasing values for more clusters. The data sets in this study had dimensionalities 1984 and 2588 and were about 98% sparse.

For this reason, the similarity measures are small and thus are the differences between document vectors. Presumably, this makes choosing the initial centroids vague, and the actual clustering algorithm cannot benefit from it.



## Chapter 5

# Conclusions

This section gives some personal remarks on problems and experiences encountered during this work.

Although the experimental results are not very encouraging, I enjoyed working on this paper very much. Indeed, the results indicate that the performance of some clustering algorithm (or initialization technique) actually depends on the data set it ought to be working on. The results in the original papers were very promising, but applied to diverse sets of documents, these look quite different.

After making myself familiar with the topic, I spent a lot of time looking for appropriate document sets and transforming the documents step-by-step into document vectors. Unfortunately, the up-to-date *TREC* [1] and *RCV1/RCV2* [10] document collections used in many papers are not available for free. So, after all, I decided to go with *Classic3* and *Reuters*. These collections are rather old and especially *Reuters* contains many documents only made up of numbers and abbreviations. Also, their original format is inconvenient for further processing, which is one important improvement of its successor *RC1/RCV2*.

But even after conversion into XML, the preprocessing was not straightforward. Particularly annoying were original, unprocessed documents that had no content at all; I noticed these not until calculating the term weights. I used Python for document preprocessing, which turned out to be a good choice because of Python's many programming libraries and the development speed usually associated with scripting languages.

For the clustering, however, Python turned out to be a bad choice. A single run of **k-means** on the *Classic3* document set with 3 clusters took between 140 and 170 seconds. In order to run 20 trials for each test case in reasonable time, I reimplemented the algorithms in C++, where the same run took only about 5 seconds.



# List of Figures

- 4.1 Percentage improvements of `kkz` and `k-means++` over `k-means`, averaged over all cluster sizes, on the *Classic3* and *Reuters* document sets.

15



# List of Tables

4.1	Characteristics of the preprocessed document sets used in the experiments.	12
4.2	Accuracy results on the <i>Classic3</i> document set. For <b>k-means</b> , the absolute overall F-measure is listed. For <b>kkz</b> and <b>k-means++</b> , the percentage improvement over <b>k-means</b> (4.6) is listed.	16
4.3	Efficiency results on the <i>Classic3</i> document set. For <b>k-means</b> , the total running time $T$ in seconds and the number of iterations $I$ are listed. For <b>kkz</b> and <b>k-means++</b> , the percentage improvement over <b>k-means</b> (4.7) is listed.	16
4.4	Accuracy results on the <i>Reuters</i> document set. For <b>k-means</b> , the absolute overall F-measure is listed. For <b>kkz</b> and <b>k-means++</b> , the percentage improvement over <b>k-means</b> (4.6) is listed.	16
4.5	Efficiency results on the <i>Reuters</i> document set. For <b>k-means</b> , the total running time $T$ in seconds and the number of iterations $I$ are listed. For <b>kkz</b> and <b>k-means++</b> , the percentage improvement over <b>k-means</b> (4.7) is listed.	16



# Bibliography

- [1] “Text retrieval conference TREC.” [Online]. Available: <http://trec.nist.gov/>
- [2] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” in *Proceedings of the 18th ACM SIAM Symposium on Discrete Algorithms*, 2007, pp. 1027–1035.
- [3] P. Berkhin, “Survey of clustering data mining techniques,” Accrue Software, San Jose, Tech. Rep., 2002.
- [4] I. S. Dhillon and D. S. Modha, “Concept decompositions for large sparse text data using clustering,” *Machine Learning*, vol. 42, no. 1/2, pp. 143–175, Jan. 2001.
- [5] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, “Clustering large graphs via the singular value decomposition,” *Machine Learning*, vol. 56, no. 1–3, pp. 9–33, Jan. 2004.
- [6] B. C. M. Fung, K. Wang, and M. Ester, “Hierarchical document clustering using frequent itemsets,” in *Proceedings of the SIAM International Conference on Data Mining*, 2003.
- [7] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, “A new initialization technique for generalized lloyd iteration,” *IEEE Signal Processing Letters*, vol. 1, no. 10, pp. 144–146, Oct. 1994.
- [8] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley and Sons, 1990.
- [9] B. Larsen and C. Aone, “Fast and effective text mining using linear-time document clustering,” in *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 16–22.

- 
- [10] D. D. Lewis, Y. Yang, T. Rose, and F. Li, “RCV1: A new benchmark collection for text categorization research,” *Journal of Machine Learning Research*, vol. 5, pp. 361–397, Dec. 1994.
- [11] S. P. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.
- [12] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the 5th Berkley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.
- [13] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130–137, July 1980.
- [14] G. Salton, *The Smart Retrieval System – Experiments in Automatic Document Retrieval*. Englewood Cliffs: Prentice Hall, 1971.
- [15] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw-Hill Companies, 1983.
- [16] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975.
- [17] M. Steinbach, G. Karypis, and V. Kumar, “A comparison of document clustering techniques,” Department of Computer Science and Engineering, University of Minnesota, Minneapolis, Tech. Rep., 2000.
- [18] C. J. van Rijsbergen, *Information Retrieval*. London: Butterworths, 1979.
- [19] Y. Zhao, G. Karypis, and U. Fayyad, “Hierarchical clustering algorithms for document datasets,” *Data Mining and Knowledge Discovery*, vol. 10, no. 2, pp. 141–168, Mar. 2002.