Unsupervised and Semi-Supervised Clustering

Henrik Ljungdahl
Department of Computer Science
C03, Institute of Technology
Lund University, Sweden
c03hlj@efd.lth.se

Abstract

This document describes an effort to cluster a large amount of documents into a given number of categories using unsupervised clustering, an explanation of why this was unmanageable, and a revised, semi-supervised approach. Basic elements and techniques of document clustering are explained, as are evaluation measures. Results do not come close to the state of the art algorithms but the algorithms are basic, conceptual and understandable.

1 Introduction

Clustering means analyzing a number of texts and putting them together in groups where all documents share some property or properties. Depending on the purpose, this could mean different things. The texts in a cluster could for instance all be of the same language, subject, author or any other property. For the analysis, there must be some measure that can be made to examine the similarity between the documents and this measure can be more or less difficult to design, depending on what property is to be used for the clustering. For instance, it is a lot easier to note a difference between two documents written in English and Swedish respectively, than it is to tell documents written by different workers within the same organization apart.

1.1 Practical uses

The Internet is already an incomprehensibly big collection of information with no supervision of content organization whatsoever. Clustering is one way of grouping information so that related information can be found easily. But there are also other uses for these techniques. Work has already been done to use clustering methods to group medical articles and information (Johansson, 2002). As information pools grow and evolve over time, their previous partitions may become obsolete and automatic text clustering is a cheap and efficient way to update these partitions (Rosell, 2005). Refined methods will possibly be valuable in police work to find similarities between crimes described in reports from vast areas, thus helping with identifying criminals that are very hard to find today.

2 Clustering

To perform the clustering itself, what is needed is of course a number of texts, a way to compare them based on the relevant property and a desired number of clusters to put them in. A commonly used way to compare the similarity between texts is to represent them with *vector space models* and then measure the *cosine similarity* between the models. This is also what has been done in this project. In this section, the fundamentals of clustering are explained.

2.1 Corpus

The text collection used in the experiments is the *RCV1-v2/LYRL2004*, a test collection based on the Reuters Corpus (Lewis et al., 2004). It consists of 804,414 news articles, manually annotated with various meta data including categories such as *Market*, *Corporate/Industrial*, *Economics* and *Government/Social*. The collection is distributed as both XML-formatted documents and in a tokenized version, which is the one mainly used for the experiments within this project.

2.2 Document representation

The vector space model is a representation of one text in an arbitrarily large collection of texts. The idea is to define a vector[1] containing the same number of elements as there are unique words in the text collection. Every unique word has its own index in the vector. Each document can then be represented by a vector as long as the word vector, but containing the respective document's word count of every word, rather than the words themselves. Consider the following example:

$D1$: I have a white Aston Martin
$D2$: I have a blue Aston Martin
$D3$: Martin wants a white Aston Martin

The resulting word vector would be:
{`I, have, a, white, Aston, Martin, blue, wants`}

And the documents could be represented by:
$D1$: {`1, 1, 1, 1, 1, 1, 0, 0`}
$D2$: {`1, 1, 1, 0, 1, 1, 1, 0`}
$D3$: {`0, 0, 1, 1, 1, 2, 0, 1`}

Note that the order of the words within the respective documents is discarded in this representation, which is why the vector space model is sometimes described as a *bag-of-words* representation. With a large collection, it is easily understood that the number of unique words and thus the length of each document vector can grow unreasonably large. To avoid this, the lemmatized and normalized form of each word has here been used to construct the word vector.

2.2.1 Term weighting

To increase the importance of words common in one document but not in the others and accordingly decrease the importance of words common throughout the entire collection, *tf-idf weights* can be applied to the word counts. This is a statistical measure used to avoid excess influence from very common words such as 'the' or 'a'. Without using tf-idf weights, such words could play a significant role in the similarity examination, which in turn could lead to an erroneous result. The word count w_t in the example above is replaced by a weighted number calculated as follows:

$$w_t = \mathrm{tf}_{i,j} \cdot \mathrm{idf}_i = \frac{n_{i,j}}{\sum_k n_{k,j}} \cdot \log \frac{|\mathcal{D}|}{|\{d_j : t_i \in d_j\}|} \quad (1)$$

where the first fraction is the tf-part consisting of the count $n_{i,j}$ of the word i in the document j, that is the numbers in the vectors in the example above. The denominator is the total count of all words in that document. Thus, the tf fraction is only based on the importance of a specific word within the current document. We can now understand the non-abbreviated name, *term frequency*. The second fraction is the idf-part, which stands for *inverse document frequency*. It is a way to examine the importance of the word in the entire text collection. The numerator $|\mathcal{D}|$ is the total number of documents in the collection and the denominator is the number of documents in the collection in which the current word t_i appears. The log value of the quotient is the idf-value for the word (Salton et al., 1974).

In the tokenized RCV1-v2/LYRL2004 corpus, a variant of tf-idf weighting has been used, called *ltc* term weighting. The formula is a bit different and looks like this:

$$w_t = (1 + \log_e n_{i,j}) \cdot \log_e \frac{|\mathcal{D}|}{|\{d_j : t_i \in d_j\}|} \quad (2)$$

with the same mappings as in (1) (Lewis et al., 2004).

2.3 Similarity measure

My goal was to replicate the text categorization in the corpus using automatic clustering and see how close I could get to the manual annotation. Since I wanted to measure the 'overall' similarity between the documents, I chose the standard way of measuring document similarity, namely by calculating the *cosine similarity* between the vectors.

The normalized document vectors constitute a spanning set for an n-dimensional space $\mathbb{R}^n$ where n is the length of the word vector. In the experiments, in which only the stemmed and de-capitalized word forms were used, the vectors spanned a space

[1] Here, the term *vector* is used in a computer programming context rather than a mathematical context.

$\mathbb{R}^{47,237}$. Each of the document vectors end in a point in this space, and by calculating the distance between these points, one can get a measure of how similar the documents are. The cosine similarity between two documents $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ is calculated with the formula below:

$$\text{sim}(\overrightarrow{d_1}, \overrightarrow{d_2}) = \frac{\overrightarrow{d_1} \bullet \overrightarrow{d_2}}{|\overrightarrow{d_1}| \cdot |\overrightarrow{d_2}|} \tag{3}$$

where the numerator is the *dot product* of the document vectors and the denominator is the multiplied lengths of the vectors (Manning et al., 2008). However, all of the document vectors in the corpus used were length normalized, so the dot product alone was enough to produce a similarity measure.

2.4 Clustering

With all documents represented in a comparable way and with a similarity measure, the only thing remaining is the clustering itself. There are several known algorithms to use for clustering, all with different performance regarding both speed and correctness. For this project, the rather straightforward *K-means* algorithm was chosen.

2.4.1 The K-means algorithm

With the K-means algorithm, the idea is to define a number of clusters and then assign a random text from the collection to every cluster. These texts are now the initial *centroid* of every cluster. A centroid is the *mean vector* of all the vectors that are currently in the cluster. Whenever a new document is added to the cluster ω the centroid $\overrightarrow{c}$ needs to be recalculated and this is done as follows:

$$\overrightarrow{c}_\omega = \frac{1}{|\omega|} \sum_{\overrightarrow{d} \in \omega} \overrightarrow{d} \tag{4}$$

Another measure used in the algorithm is the *residual sum of squares* or RSS, which is a measure of how well a centroid actually represents the texts it is made of, that is all the texts in its cluster. It is calculated as follows:

$$\text{RSS}_\omega = \sum_{\overrightarrow{d} \in \omega_k} |\overrightarrow{d} - \overrightarrow{c}(\omega_k)|^2 \tag{5}$$

The K-means algorithm creates the initial clusters, adds all the texts in the collection to their appropriate clusters and then starts 'moving' the centroids around by reassigning texts to other clusters and recalculating the centroids. The moving stops either after a pre-defined number of iterations or when the sum of the clusters' RSS value reaches a pre-defined threshold. In other words, the algorithm tries to assign texts to the clusters in a way which maximizes the centroids' potential to represent the texts in their respective clusters (Manning et al., 2008).

2.4.2 Downsides of unsupervised clustering

There is a major problem with clustering big collections of text using the strategies described so far. The problem is memory, or rather the lack of it. The initial assigning of text to clusters is no problem, since the centroids can be calculated accumulatively without saving any information from earlier documents other than the centroid itself, which of course is a mean of all earlier documents. However, in order to perform the RSS calculations, the algorithm needs to examine the full vectors of all the texts in the cluster. A word vector has 47,237 elements of the type `double`. A double is 8 bytes, which means that every vector occupies ≈369 KB of memory. Analyzing the Reuters Corpus would thus require 283.1 GB of RAM. In other words, no regular computer can hold all of this information in the random access memory, so the hard drive must be used to store the information. This makes the clustering so slow it is practically impossible to carry out without a new approach. This is why unsupervised clustering in this project was discarded in favor of *semi-supervised.*

2.4.3 Semi-supervised clustering

The semi-supervised clustering techniques is rather a way of *categorizing* than clustering. The difference might seem subtle, but is really quite clear. Categorizing means that the texts in a collection are sorted into given categories with specified properties, while unsupervised clustering tries to find patterns that connect texts into clusters without any previous knowledge. Semi-supervised clustering is a hybrid between these two procedures.

The algorithm requires some input consisting of some of the texts from the collection that is supposed to be clustered. These texts have to be *manually annotated* with information about which cluster(s) they should be part of. When these 'seeds' are planted, the centroids for the clusters are calculated and the rest of the texts in the collection are put in the cluster(s) they match best. No RSS calculations are made, and thus the algorithm does not require as much memory.

2.5 Our experiments

After realizing that we would not be able to use unsupervised clustering without extremely slow workarounds, we decided to try and classify all of the texts into the four main categories of the Reuters Corpus using semi-supervised clustering. The clustering application creates four clusters and seeds them with 100 texts each. Many of the texts in the corpus belong to more than one of the four main categories, but to achieve a sufficient initial distance between the clusters, documents that belonged to no more than one category were chosen as seeds.

2.5.1 Clustering semi-supervised

In the algorithm, all of the texts were compared to each of the clusters, and if the euclidean distance turned out to be below a certain threshold, the text was considered to belong to that cluster. This meant that each text could be put in more than one category. This matched the way the texts were annotated from the beginning. After a text was added to a cluster, we decided *not* to recalculate the centroid, since this would allow influence from texts that belonged to other categories as well.

To find an appropriate threshold value, we experimented with a range of different values until the one that gave the best $F_{1.0}$-score was found (more about this in section 3.1). The best threshold we found for the euclidean distance was 1.00722.

2.5.2 Revision one

The results were disheartening. When evaluating the clustering results (as described in section 3) we realized that even random clustering gave better results. We suspected that that the problem lied in the comparison with the centroids created from the seeds. Because of this, the algorithm was altered so that instead of creating centroids from the seed documents, each document $\overrightarrow{d}$ was compared to all of the seed documents S of each cluster ω and a score σ_ω was created from the sum of the similarity measures (see formula below).

$$\sigma_\omega = \sum_{\overrightarrow{s} \in S_\omega} \text{sim}(\overrightarrow{d}, \overrightarrow{s}) \qquad (6)$$

The document was assigned to the cluster ω if the similarity score σ_ω was greater than a specified threshold.

3 Evaluation

The Reuters Corpus is manually annotated with category information. This made it fairly uncomplicated to compare the clustering results with reference annotations and in this way evaluate the results. $F_{1.0}$-score measures were used for the evaluation.

3.1 $F_{1.0}$-score

A commonly used measure of test accuracy is the $F_{1.0}$ score. To calculate it, we must first calculate the *recall* R and *precision* P of the test. This is done as shown in equations (7) and (8) below.

$$R = \frac{C}{C + M} \qquad (7)$$

$$P = \frac{C}{C + I} \qquad (8)$$

C is the number of documents correctly assigned to a category, I is the number of documents that were incorrectly assigned to a category and M is the number of documents that did not get assigned to a category they should have been assigned to. We now calculate the $F_{1.0}$ measure as follows (Lewis et al., 2004):

$$F_{1.0} = \frac{2RP}{R + P} \qquad (9)$$

4 Results

4.1 State-of-the-art results

It is hard to find results that serve as valid candidates for comparison, since this would require exactly the same experimental setup. Lewis et al. has used four different algorithms to categorize the collection with impressive results, and they have been able to reach $F_{1.0}$-scores as high as around 0.9 (Lewis et al., 2004), but this has been done with supervised techniques rather than semi-supervised. Moreover, the scores comprise categorization based on other properties than topics, such as for instance region categories. This renders the results useless for comparison, but they still give an idea of what performance can be reached with state-of-the-art techniques.

4.2 Our results

The table below shows the evaluation results for the different categories.

Category	$F_{1.0}$**-score**
Market	.280
Economics	.161
Corporate	.500
Governmental	.373

5 Conclusions

It is fairly obvious that the results are quite far from the state-of-the-art ones. The reason for this is probably (at least partly) that the word vector is very large in comparison to the individual text lengths. The consequence of this is that every document vector contains mostly zero-value elements, which can affect the performance of clustering algorithms in a negative way (Lewis et al., 2004). There is a way to milden these effects, though beyond the scope of this project, called *supervised feature selection*. In short, this means that only the words most relevant for the individuality of the texts are used to create the word vector, and the length of the word vector can therefore be set arbitrarily. Probably the correctness of the algorithm would have been better if this technique was implemented.

References

Peter Johansson. 2002. *Klustring av svenska texter*. Kungliga Tekniska Högskolan, Stockholm, Sweden.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. *RCV1: A New Benchmark Collection for Text Categorization Research. Journal of Machine Learning Research*, 5:361–397.

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. 2008. *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England.

Magnus Rosell. 2005. *Clustering in Swedish*. Kungliga Tekniska Högskolan, Stockholm, Sweden.

Gerard Salton, A. Wong and C. S. Yang. 1974. *A Vector Space Model for Automatic Indexing* (TR74-218). Department of Computer Science, Cornell University, Ithacha, NY.