Protocol: blog clustering

Filip Sedlák

May 16, 2011

1 Aim

Cluster the set of blogs based on the similarity of topics they discuss, and compare clustering results to known blog classes.

2 Input data

The file `assignment/feedlist.txt.orig`[1] contains blog domains divided into seven sections. This data served as a reference for automated clustering.

3 Data preparation

For this work, text data from RSS feeds of supplied blogs was used. In order to get RSS feeds from web pages, the pages were fetched and links to RSS were found (`get_feeds.py`). These URLs were then manually completed because not every blog specified its RSS channel in a machine-readable format.

As a next step, the RSS feeds were downloaded (`download_feeds.py`). In this step, not all feeds could have been downloaded. Namely observingthesky.org and lunarsoil.com which were inaccessible at the moment.

The downloaded XML documents were parsed using Universal Feed Parser[2] and HTML to text conversion was performed by BeautifulSoup[3].

3.1 Creating feature matrix

Contrary to the original hint, the feed summary couldn't be used as it contained no or useless information for most of the blogs. Instead, the full RSS content was used with different weights for different parts of the feed (`generate_corpus.py`). These weights should be further optimized because no formal optimization technique was used.

Then, several techniques aided by Natural Language Toolkit[4] were performed (`extract_features.py`).

1. The text was tokenized into words, dividing also words containing a hyphen.
2. The words containing special characters, the words shorter than three characters and the words starting by a digit were filtered out.
3. The words were tagged by their corresponding part of speech and uninformative parts of speech were filtered.

[1] All files referenced can be found in the attachment.
[2] http://www.feedparser.org/
[3] http://www.crummy.com/software/BeautifulSoup/
[4] http://www.nltk.org

4. Stemming by Porter algorithm was performed.

5. Common english stop words plus some frequently encountered words in this context were filtered (e.g. blog, weblog).

6. The counts of the remaining words, multiplied by their RSS section weight, were saved into the feature matrix having observations (individual blogs) as rows and word counts as columns (`feature_matrix.csv`).

The feature matrix was normalized row-wise to account for different word counts in the blogs (`normalize_rows.py`).

As a next step, to leverage the importance of words found only in smaller portion of documents, the counts in the feature matrix were transformed into TF×IDF[5] values (`make_tf-idf.py`).

At the time, the feature matrix had more than 9000 columns because many of the words were not common for different blogs. To reduce dimensions of the feature space, the Principle Component Analysis was used (`pca.R`). Thanks to this technique, the number of dimensions could have been reduced to 25. In this dataset was a number of outliers (e.g. snakesonablog.com which isn't really a movie blog but contains the word "snake" really often). To improve PCA's performance, a robust algorithm was used[6] (the first attempt to address this problem by using logarithms of the frequencies failed).

For Rapid Miner[7], used in some of the next steps, the number format used in data matrix must have been edited.

4 Clustering

4.1 Agglomerative clustering

The PCA transformed data was clustered based on correlation dissimilarity measure rather than the euclidean distance. This measure neglects the differences in absolute word counts. It performs better than the euclidean distance measure (best adjusted Rand index around 0.49 vs. 0.25).

The number of clusters was optimized to achieve a maximal Rand index. Depending on other settings the cluster count varied around 25, having several clusters of size 1. For alternative linkage measures comparison, see the following figures.

[5] $TF \times IDF = f_{term} \times \log(\frac{n_{termDocs}}{N_{docs}})$ where f_{term} is the normalized term frequency in a document, $n_{termDocs}$ is the number of documents containing the term and N_{docs} is the total number of documents in the set.

[6] http://cran.r-project.org/web/packages/pcaPP/pcaPP.pdf

[7] http://rapid-i.com/content/view/181/196/

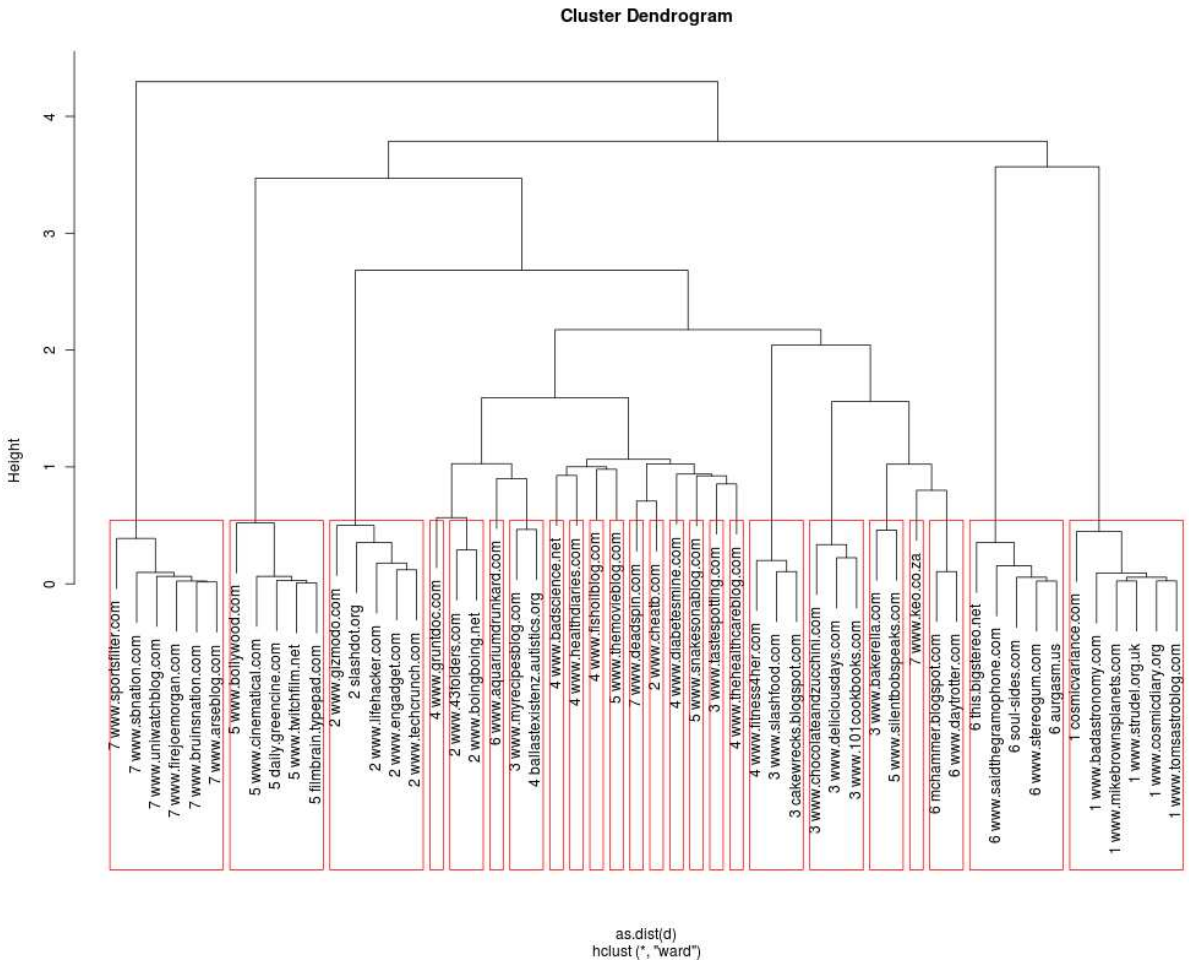

Figure 1: Clustering by ward linkage (adj. Rand index 0.49). The numbers correspond to the original section. This linkage creates a well balanced tree where some clusters are erroneously divided (2, 3, 6). The cluster 4 (medicine) wasn't found at all.

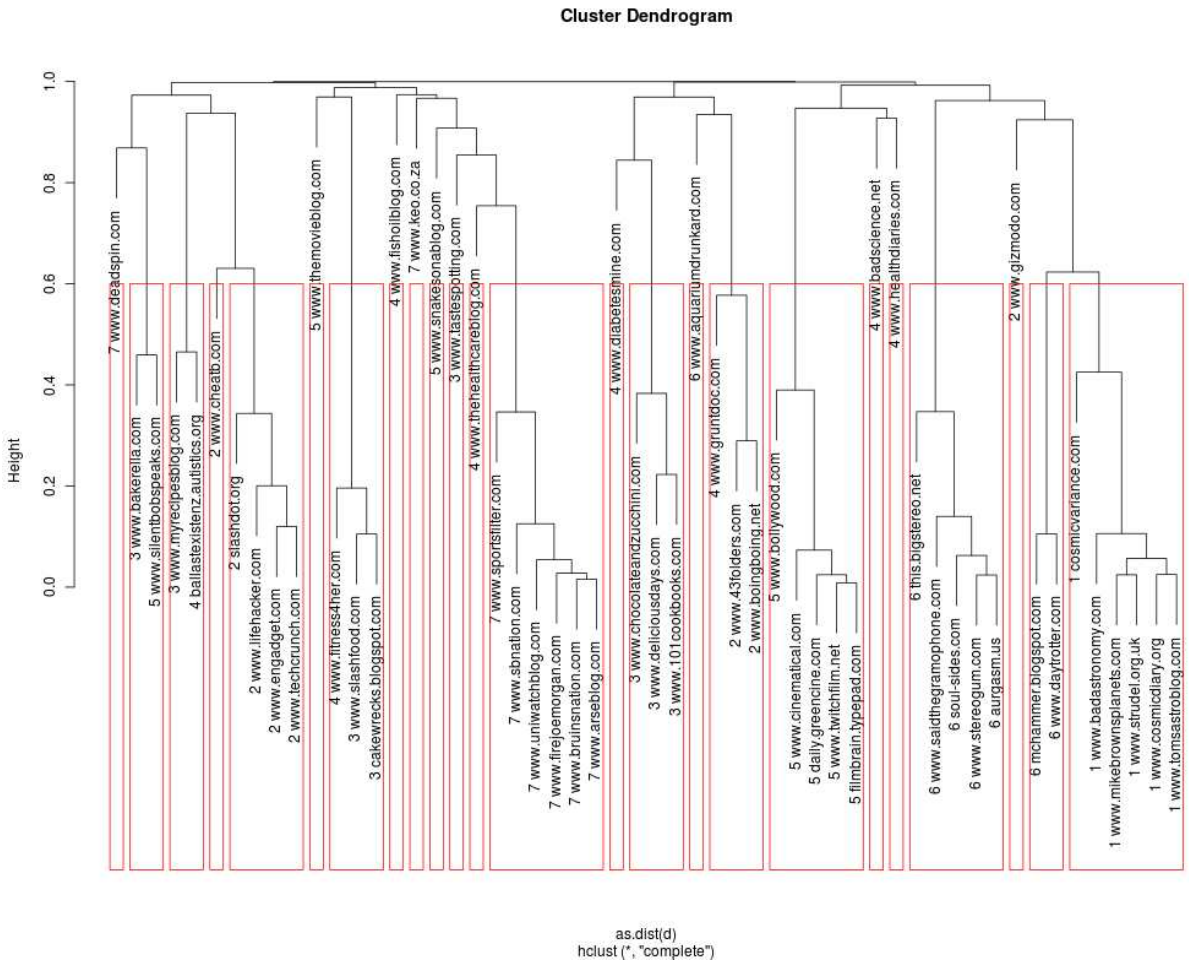

Figure 2: Clustering by complete linkage (adj. Rand index 0.46). Now the tree is not so well balanced as with the ward linkage but the clusters are similar to the previous ones.

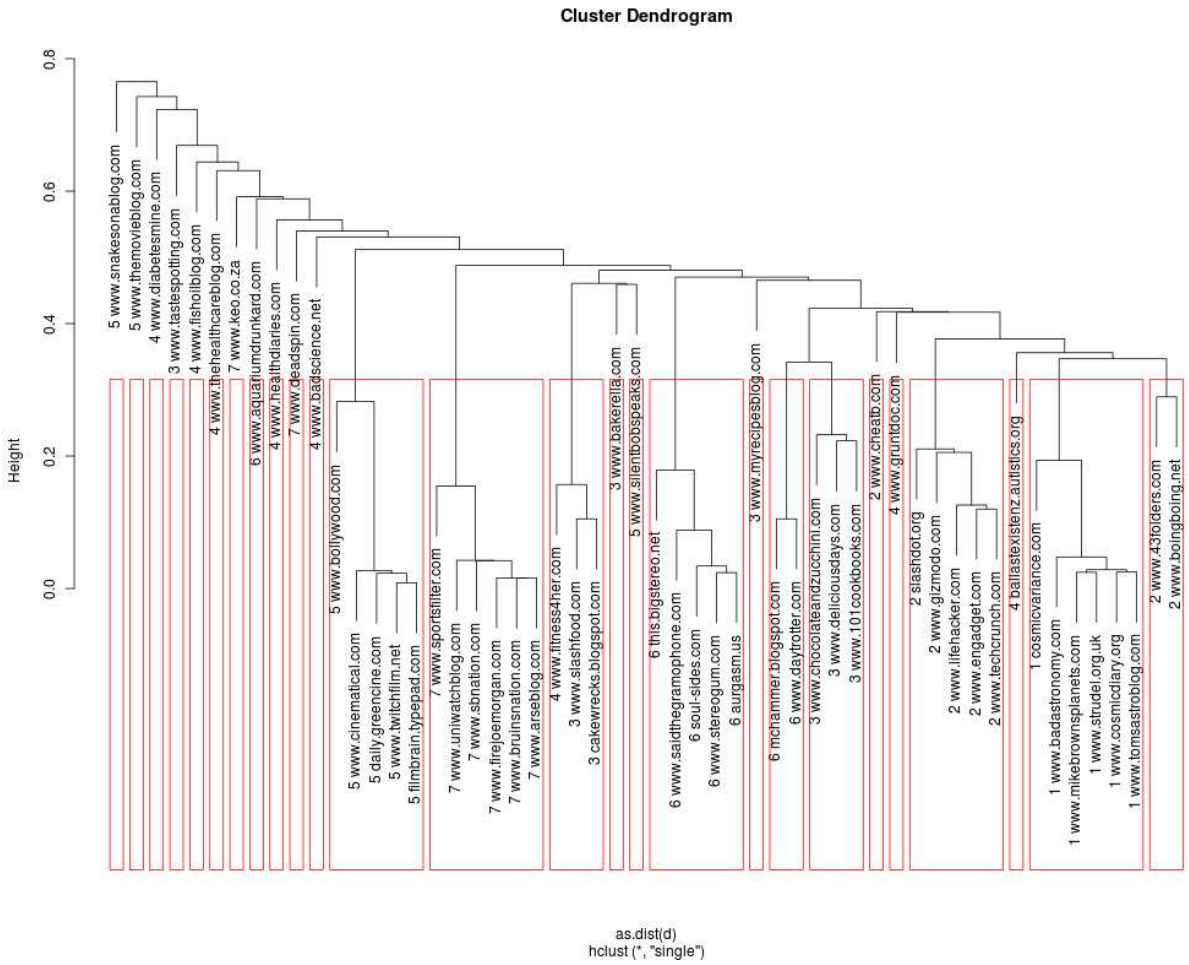

Figure 3: Clustering by single linkage (adj. Rand index 0.49). This linkage leaves most of the outliers at one side of the tree while the rest is relatively well clustered. Like with the other linkage measures, the category 4 (medicine) couldn't be clearly separated.

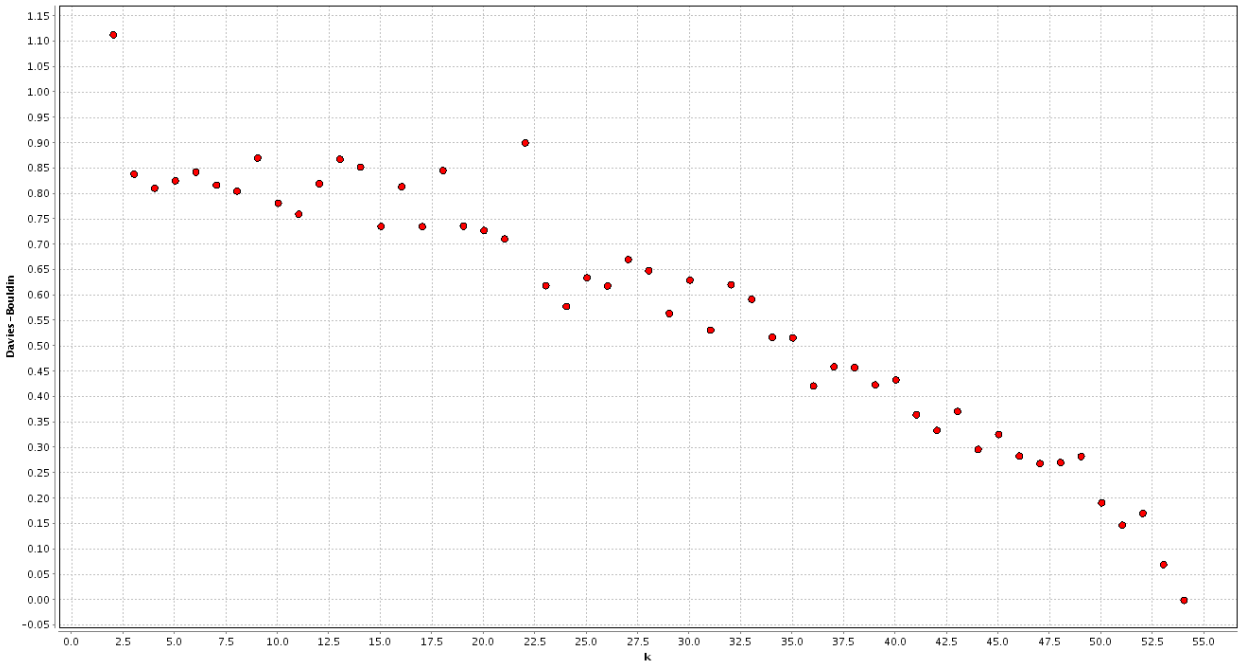

Figure 4: Davies-Bouldin criterion depending on the number of centroids. The real criterion for this data is negative and should be minimized. The positive values come from Rapid Miner which is only able to optimize for a maximal value thus needing negative of the Davies-Bouldin index. The points between k=3 and 18 form a plateau. Knowing the real number of classes, we can choose k equal to it (as 7, the real number of classes belongs to the plateau). Without this knowledge, another validation measure would be necessary to estimate k.

4.2 K-means clustering

For k-means clustering, the data was preprocessed by a self-organizing map (Kohonen network) to further reduce dimensionality. Then the optimal number of clusters was assessed using a Davies-Bouldin criterion. When using a large number of dimensions, the k-means clustering performed badly because the data points were sparse in the high-dimensional space.

According to Rand index (being 0.27), the k-means clustering performed worse than hierarchical clustering. This is presumably the effect of additional dimensionality reduction which was needed for k-means clustering to work at all. However, the higher dimensional self-organizing map could solve this problem. The trade-off for this would be the much higher computation complexity.

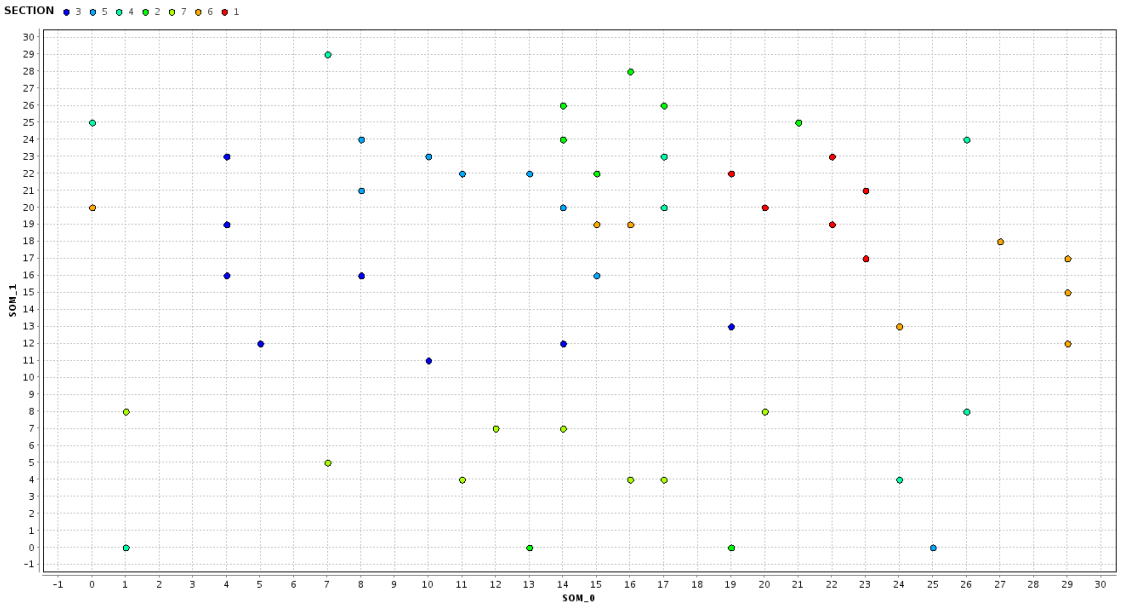

Figure 5: Correct clustering of the data. We can see the section no. 4 is very unlikely to be found as a cluster because of its wide spread. The sections 7 and 2 are likely to be divided into multiple clusters because section 7 forms a cloud which is not round but rather long and sect. no. 2 doesn't have all of its points near each other (although there is some center around 16, 25).

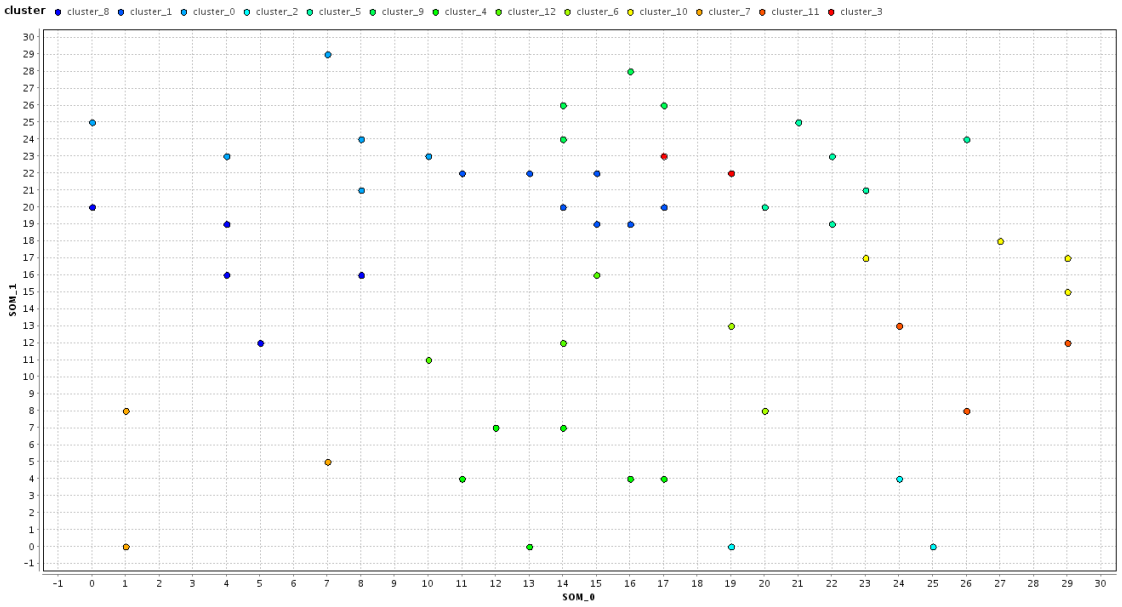

Figure 6: Machine-learnt clustering from the 2D data.

Conclusion

The success of the text-based clustering depends largely on the feature matrix quality. That's why the different methods for hierarchical clustering performed similarly. The preprocessing steps can improve success substantially (e.g. the dimensionality reduction from 9000 to 25 dimensions) but they can also ruin it (like taking logarithm of the term frequency instead of the plain frequency).

Also, even the first step, the text extraction, proved to be more difficult than originally estimated – the websites don't often present content suited for machine processing and the file formats are not always valid. Thus most of the time was spent by a human review of various intermediate data. To ensure a next step can be made with a sensible input.

The prediction ability of these clustering models wasn't estimated at all. This is another factor which should be taken into account. However the sample data is too little set to make a general model.