University of Amsterdam

Master's thesis

Clustering by file compression

Author:
Geert Kapteijns

Supervisor:
Dr. Jan van Eijck

Centrum voor Wiskunde en Informatica

June 2015

Contents

Chapter 1

Introduction

1.1 Similarity of data

How do we know that Dutch is more similar to German than it is to French? How do we know that Bob Dylan's music is closer to The Beatles' than it is to Bach's?

Does a computer know?

This thesis concerns a method expressing similarity of data that is feature free: it does not use domain knowledge about the data (for example, word origins or grammar rules in the case of languages.) The method is based on file compression and is rooted in Kolmogorov complexity.

The idea is easy to grasp. If a compressor compresses the concatenation of two files better than it compresses the files separately, it must have found some regularities that appear in both files. This compression gain is used to define a similarity metric that aims to capture the similarity of every dominant feature of the data.

Chapter 2

Normalized compression distance

2.1 Foundations in Kolmogorov complexity

In this chapter we will make explicit the idea of similarity based on file compression. But first, we explain the notion of Kolmogorov complexity. For a complete reference, see [1].

The Kolmogorov complexity of a string x, written $K(x)$, is the length of the shortest program that outputs x.

Intuitively, $111\ldots111$, a string of a million ones, is not very complex. It does not contain much information. Indeed, the Kolmogorov complexity of this string is low. A 27-byte Ruby program produces it:

```
1000000.times { print "1" }
```

I do not claim the Kolmogorov complexity of a string of a million ones is 27 bytes. The true Kolmogorov complexity of a string x cannot be computed in the Turing sense. There is no program that, given x, outputs the (length of) the shortest program that produces x.

The string $011\ldots010$, produced by flipping a fair coin a million times, has, with very high probability, a Kolmogorov complexity close to its own length (by counting the number of different bit strings of each length, you can show that the chance to compress a random string by more than c bits is at most 2^{-c}).

So, printing the literal description may be the best we can do:

```
print "011...010"
```

The Kolmogorov complexity of a string x, given a string y, denoted by $K(x|y)$, is the length of the shortest program that outputs x, given y as input.

2.1.1 Information distance

The Kolmogorov complexity is a measure of information content in an individual object. In the same way, in [2] the information distance $E(x, y)$ between two strings x and y is defined as the length of the shortest program that converts x to y and y to x. It is shown that

$$E(x, y) = \max\{K(x|y), K(y|x)\}$$

$E(x, y)$ is an absolute distance. But similarity is better expressed relatively. To illustrate: if two binary strings of length 100 have a Hamming distance of 50 (i.e. they have different bits in 50 positions), they are not very alike. If, on the other hand, two strings of length 10^6 have a Hamming distance of 50, they are very much alike.

In [3], the normalized information distance is defined as

$$\mathrm{NID}(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}} \tag{2.1}$$

and it is shown that $\mathrm{NID}(x, y)$ minorizes every distance $d(x, y)$ up to a negligible additive term, where $d(x, y)$ belongs to a wide class of normalized distances that includes everything remotely interesting.

This means that if two strings are similar according to some distance (be it Hamming distance, overlap distance, or any other), they are also similar according to the normalized information distance. This is why $\mathrm{NID}(x, y)$ is also called *the* similarity distance.

2.2 Approximating Kolmogorov complexity with a real-world compressor

The remarkable properties of the normalized information distance come at the price of incomputability. But, the Kolmogorov complexity can be approximated by real-world (lossless) compression programs like zlib or liblzma. The normalized compression distance [4] is defined as

$$\mathrm{NCD} = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}} \tag{2.2}$$

where xy is the concatenation of x and y, and $C(x)$ is the length of x, after being compressed by compressor C.

The NCD is central to this work. It is the real-world approximation of the normalized information distance (2.1).

2.2.1 A compressor does not exploit all regularity

Since K is incomputable, we have no idea how far off the length given by C is. Consider 31415..., the string consisting of the first 10^9 digits of π. The Kolmogorov complexity of this string is low: a simple program, perhaps exploiting a converging series formula, will produce it. Any real-world compressor C, however, fails to compress this string by even a few bits[1].

In the following chapters (and extensively in [4]) it is demonstrated that the NCD is adequate for many applications.

[1] A textual representation "31415..." can actually be compressed significantly by almost every compressor, but this is an encoding issue. The file consists only of the bytes 0 through 9, which can be more efficiently encoded using, for example, a system where each decimal number is assigned a four bit code (which is still naive.) For clarity, I do not concern myself with encoding in this chapter. Every finite alphabet can be recoded in binary, so it is customary to only think about binary strings in the literature. Conceptually, nothing changes.

Chapter 3

Hierarchical clustering of data

In this chapter, we cluster vastly different types of data using the normalized compression distance (NCD) (2.2). To explain the method, we first cluster mitochondrial gene sequences of mammals.

3.1 Evolution of placental mammals

Reconstructing an evolutionary tree has intuitive appeal. It should lend itself well to hierarchical clustering, since species emerge from common ancestors. And, within biology, there is agreement on what the true phylogeny tree is: the brown bear and polar bear are closely related, etc. Only higher up on the tree there is ongoing debate. Do the primates first join with the rodents, or are they more closely connected to the ferungulates?

All materials were taken from the GenBank database [5].

In [6], the authors estimate the likelihood of phylogeny trees based on 12 mitochondrial proteins of 20 placental mammals: rat (*Rattus norvegicus*), house mouse (*Mus musculus*), grey seal (*Halichoerus grypus*), harbor seal (*Phoca vitulina*), cat (*Felis catus*), white rhino (*Ceratotherium simum*), horse (*Equus caballus*), finback whale (*Balaenoptera physalus*), blue whale (*Balaenoptera musculus*), cow (*Bos taurus*), gibbon (*Hylobates lar*), gorilla (*Gorilla gorilla*), human (*Homo sapiens*), chimpanzee (*Pan troglodytes*), pygmy chimpanzee (*Pan paniscus*), orangutan (*Pongo pygmaeus*), Sumatran orangutan (*Pongo abelii*), using opossum (*Didelphis virginiana*), wallaroo (*Macropus robustus*), and the platypus (*Ornithorhynchus anatinus*).

In [4], 4 more mammals were added: Australian echidna (*Tachyglossus aculeatus*), brown bear (*Ursus arctos*), polar bear (*Ursus maritimus*), and the common carp (*Cyprinus*

carpio). The common carp is not a mammal and is used as an outgroup. It should join the phylogeny tree at the very top.

The authors of [4] cluster the complete mitochondrial genome sequences with the normalized compression distance. They use a clustering algorithm described by them in [7]. The algorithm tries to optimize a global criterion, namely the (normalized) summed weights of all consistent quartet topologies (layouts of groups of four items). In a binary tree, only one of three possible pairings of four items ($ab|cd$, $ac|bd$, $ad|bc$) is *consistent*, in the sense that you can connect the two pairs without crossing paths. The sum of the distances (in this case, NCDs) between the items in the consistent pairs is the contribution of quartet *abcd* to the tree score.

This method works especially well if the items you're trying to cluster result from an evolutionary process, since then (without corruption of data) there should exist an evolutionary tree that embeds all the most likely quartet topologies, and, given enough time, the heuristic presented in [7] finds the true tree.

Now, we will cluster the same 24 genome sequences with a much simpler algorithm (average linkage) and compare results.

3.1.1 Distance matrix

The distance matrix contains the distances between all pairs of items. Entry i, j is the distance between item i and item j, i.e. $\mathrm{NCD}(i, j)$. The distance matrix for the 24 animals is displayed in table 3.1.

blueWhale	0.01	0.72	0.86	0.67	0.78	0.60	0.83	0.24	0.81	0.80	0.67	0.66	0.62	0.78	0.77	0.82	0.81	0.77	0.83	0.70	0.79	0.78	0.83	0.63
brownBear	0.71	0.01	0.89	0.63	0.84	0.72	0.85	0.69	0.84	0.82	0.56	0.56	0.69	0.81	0.80	0.84	0.84	0.82	0.84	0.11	0.82	0.83	0.84	0.64
carp	0.87	0.88	0.01	0.88	0.89	0.87	0.89	0.89	0.89	0.89	0.89	0.88	0.87	0.88	0.86	0.88	0.89	0.89	0.90	0.88	0.88	0.89	0.88	0.88
cat	0.67	0.59	0.87	0.01	0.79	0.68	0.84	0.69	0.79	0.79	0.59	0.57	0.61	0.81	0.77	0.80	0.80	0.80	0.82	0.60	0.75	0.81	0.78	0.59
chimpanzee	0.78	0.83	0.90	0.81	0.01	0.77	0.87	0.82	0.49	0.34	0.80	0.79	0.79	0.29	0.84	0.88	0.47	0.17	0.89	0.83	0.84	0.47	0.86	0.80
cow	0.61	0.73	0.87	0.68	0.78	0.01	0.84	0.61	0.79	0.78	0.66	0.66	0.62	0.81	0.77	0.82	0.81	0.77	0.84	0.71	0.76	0.81	0.81	0.62
echidna	0.84	0.87	0.89	0.85	0.85	0.83	0.01	0.86	0.85	0.86	0.87	0.87	0.84	0.88	0.81	0.81	0.87	0.85	0.55	0.87	0.84	0.86	0.84	0.83
finWhale	0.25	0.72	0.89	0.71	0.81	0.62	0.85	0.01	0.81	0.82	0.70	0.69	0.64	0.81	0.80	0.84	0.80	0.78	0.85	0.72	0.81	0.81	0.85	0.64
gibbon	0.83	0.85	0.90	0.81	0.51	0.81	0.88	0.85	0.01	0.51	0.82	0.83	0.81	0.50	0.85	0.89	0.53	0.50	0.90	0.85	0.85	0.54	0.87	0.79
gorilla	0.79	0.81	0.90	0.82	0.35	0.79	0.88	0.83	0.50	0.01	0.79	0.82	0.83	0.35	0.83	0.89	0.46	0.35	0.87	0.82	0.82	0.47	0.88	0.80
graySeal	0.69	0.56	0.88	0.60	0.79	0.66	0.84	0.69	0.79	0.80	0.01	0.16	0.64	0.80	0.77	0.82	0.80	0.78	0.85	0.55	0.78	0.80	0.80	0.61
harborSeal	0.67	0.54	0.88	0.58	0.78	0.64	0.85	0.69	0.80	0.80	0.16	0.01	0.61	0.79	0.76	0.83	0.77	0.77	0.84	0.54	0.77	0.77	0.80	0.60
horse	0.63	0.65	0.88	0.63	0.76	0.62	0.85	0.64	0.77	0.79	0.62	0.60	0.01	0.79	0.78	0.81	0.78	0.78	0.83	0.66	0.76	0.79	0.80	0.49
human	0.78	0.83	0.88	0.83	0.30	0.79	0.87	0.80	0.50	0.35	0.82	0.82	0.78	0.01	0.82	0.87	0.46	0.30	0.88	0.83	0.82	0.46	0.86	0.78
mouse	0.77	0.79	0.86	0.76	0.81	0.76	0.81	0.80	0.81	0.82	0.77	0.76	0.75	0.83	0.01	0.78	0.83	0.82	0.83	0.78	0.54	0.83	0.77	0.74
opossum	0.82	0.81	0.87	0.80	0.87	0.81	0.84	0.82	0.87	0.88	0.82	0.82	0.81	0.87	0.80	0.01	0.88	0.87	0.84	0.82	0.82	0.88	0.68	0.83
orangutan	0.80	0.83	0.90	0.82	0.46	0.82	0.88	0.80	0.52	0.46	0.82	0.79	0.83	0.45	0.84	0.88	0.01	0.47	0.89	0.83	0.83	0.25	0.86	0.82
pigmyChimpanzee	0.77	0.81	0.90	0.81	0.17	0.77	0.88	0.81	0.49	0.34	0.82	0.79	0.79	0.29	0.83	0.87	0.47	0.01	0.89	0.81	0.83	0.47	0.86	0.78
platypus	0.83	0.85	0.89	0.86	0.89	0.82	0.56	0.84	0.89	0.87	0.84	0.84	0.87	0.87	0.83	0.82	0.88	0.88	0.01	0.85	0.84	0.89	0.84	0.85
polarBear	0.70	0.10	0.89	0.62	0.81	0.70	0.86	0.71	0.84	0.81	0.56	0.56	0.66	0.82	0.79	0.83	0.82	0.81	0.85	0.01	0.78	0.82	0.82	0.65
rat	0.78	0.82	0.89	0.75	0.80	0.76	0.83	0.78	0.81	0.82	0.79	0.77	0.74	0.82	0.55	0.82	0.81	0.83	0.86	0.82	0.01	0.81	0.84	0.75
sumatranOrangutan	0.78	0.82	0.89	0.80	0.47	0.80	0.86	0.79	0.52	0.47	0.80	0.80	0.77	0.44	0.84	0.87	0.24	0.47	0.88	0.82	0.82	0.01	0.87	0.78
wallaroo	0.81	0.82	0.87	0.80	0.85	0.81	0.85	0.83	0.86	0.86	0.80	0.79	0.81	0.85	0.81	0.68	0.86	0.85	0.83	0.83	0.82	0.87	0.01	0.80
whiteRhinoceros	0.64	0.66	0.89	0.65	0.78	0.64	0.86	0.65	0.78	0.79	0.61	0.61	0.51	0.79	0.78	0.84	0.82	0.79	0.86	0.68	0.77	0.81	0.82	0.01

TABLE 3.1: Distance matrix of genome sequences of 24 animals. To obtain the NCD, I used ruby-xz, the Ruby binding to compressor liblzma, with settings `compression_level = 9`, `extreme = true`, `check = :none`.

3.1.2 Clustering method

In hierarchical clustering, the goal is to build a hierarchy of clusters from a distance matrix. The hierarchy can be displayed as a binary tree. One of the simplest ways to do this is to put every object in its own cluster, and then merge greedily based on a linkage criterion, until all items have been merged into a single cluster.

In single linkage, at each step the two clusters are merged with the smallest pairwise distance. This is also called *nearest neighbor* clustering.

In complete linkage, or *farthest neighbor* clustering, at each step the two clusters with the largest pairwise distance are merged.

Average linkage is a compromise between single and complete linkage. The distance between two clusters is defined as the average between the pairwise distances, i.e.

$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$$

3.1.3 Phylogeny tree

Figure 3.1 shows the phylogeny tree obtained by average link clustering.

Comparison with [6] shows that the dendrogram in this paper is a little bit off. Here, $((\text{finWhale}, \text{blueWhale}), \text{cow})$ joins $(\text{whiteRhinoceros}, \text{horse})$ first, while in the cited paper $((\text{harborSeal}, \text{greySeal}), \text{cat})$ first joins $(\text{horse}, \text{rhinoceros})$. Also, higher up in the tree, in this paper, the rodents join the ferungulates, and then the primates join, while in the cited paper the primates and ferungulates join first, and then the rodents join.

[4], which also uses NCD, but clusters with an algorithm described in [7], does not make these mistakes. But, the algorithm they use works especially well for evolutionary data, and they attain a global fitness score of $S(T) = 0.996$ for their tree T, which is exceptionally high. Other trees, containing, for example, random correlated data, have $S(T) = 0.905$ for the best found tree.

3.2 Random correlated data

In this section, we will cluster files containing random bytes, which we have partially correlated, so we know what the clustering should be like. This method of validation was

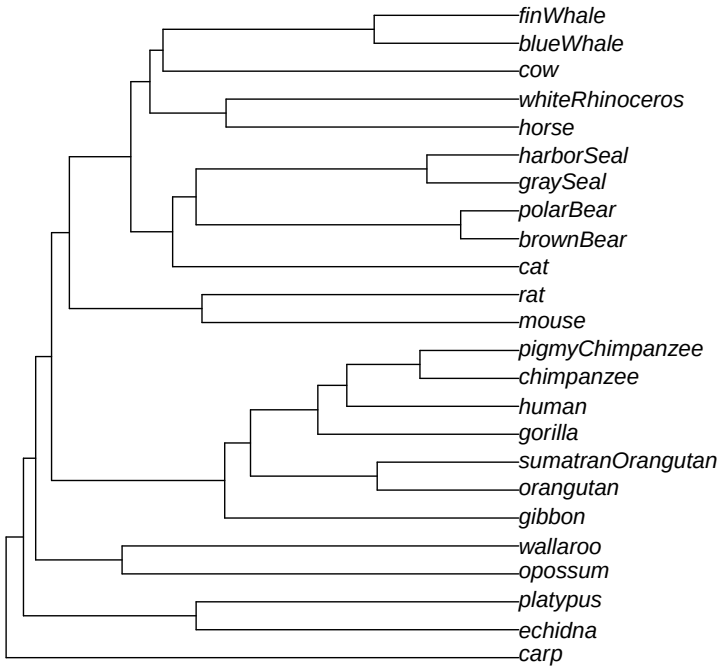

FIGURE 3.1: Result of average linkage clustering of the distance matrix shown in table 3.1. The height at which clusters join is proportional to the distance between them.

taken from [4]. All byte sequences have been generated with the random_bytes function of the Ruby library SecureRandom.

Let *tags* a, b, c be blocks of 1000 random bytes. We create file a in the following way: generate 80.000 random bytes, and at 10 distinct positions (picked from $0 \ldots 79$) replace a 1000 byte block with tag a. To create file ab, we insert tag a at 10 distinct positions, then insert tag b at 10 distinct positions, possibly overwriting some of the earlier insertions of tag a. In this way, we create files a, b, c, ab, ac, bc, and abc. The clustering is shown in 3.2.

I opted for single linkage clustering, i.e. the distance between clusters A and B is $d(A, B) = \min_{a \in A, b \in B} \mathrm{d}(a, b)$, because it shows the fact that ab, ac and bc have about the same distance to abc, while a, b and c are farther away, but also at an even distance, exactly as you would expect.

In figure 3.3, the experiment is repeated with 22 files, mimicking the experiment done in [4]. The result is what you would expect, and you could argue that this clustering

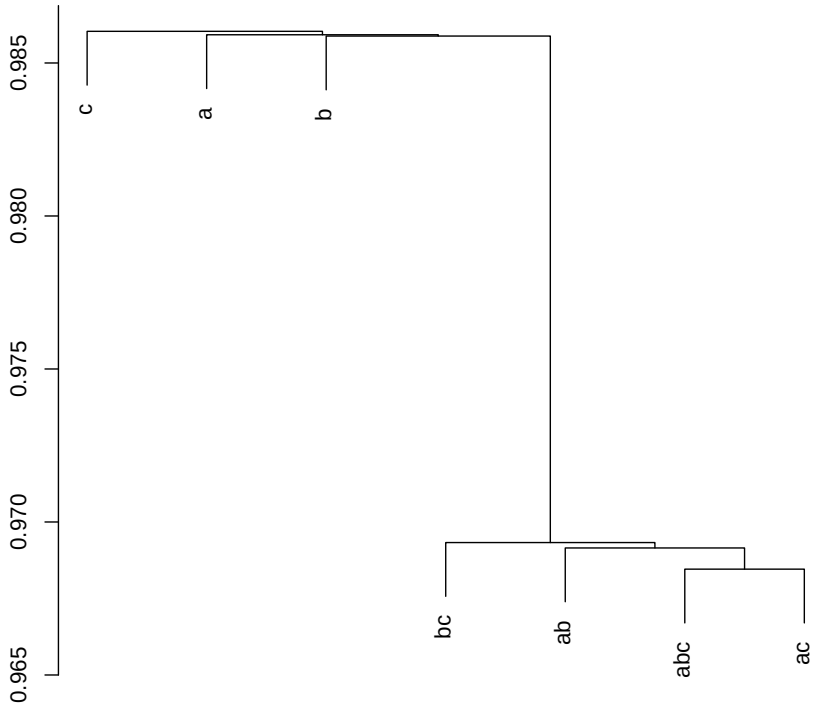

FIGURE 3.2: Single link clustering of seven 80 kB files containing random bytes, which are partially correlated. The height at which clusters join is the distance between them, according to the linkage criterion.

is more insightful than the quartet tree method, since it shows the degree in which two clusters are related. *abcd* is closer to *abce* than *jk* is to any file labeled by more than two tags, for example.

3.3 Literature

We proceed to clustering of utf-8 files of books obtained from `www.gutenberg.org`. Here, the result is hard to validate, except for human intuition. The books used in this experiment are (1) A Week on the Concord and Merrimack Rivers, (2) Walden, and On The Duty of Civil Disobedience, both by Thoreau, (3) The Adventures of Sherlock Holmes, (4) The Hound of the Baskervilles, both by Doyle, (5) The Beautiful and the Damned, (6) This Side of Paradise, both by Fitzgerald, (7) Sons and Lovers, (8) White Peacock, both by Lawrence. The result is shown in figure 3.4.

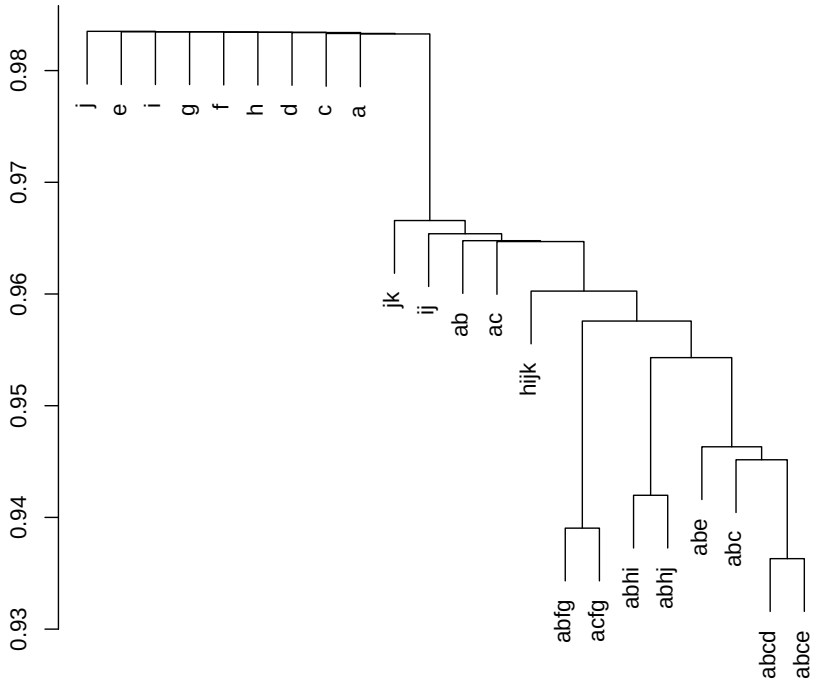

FIGURE 3.3: Single link clustering of twenty-two 80 kB files containing random bytes, which are partially correlated.

3.4 Source files

We cluster ten files from the Ruby standard library [8] at commit `c722f8ad1d`, ten files from the Clojure standard library [9] at commit `41af6b24dd` and ten files from the Glasgow Haskell Compiler 6.10.1 [10]. All comments and blank lines were stripped. This removes, among other things, copyright notices which were shared by files belonging to the same library. The result is shown in figure 3.5. No preprocessing gives similar results.

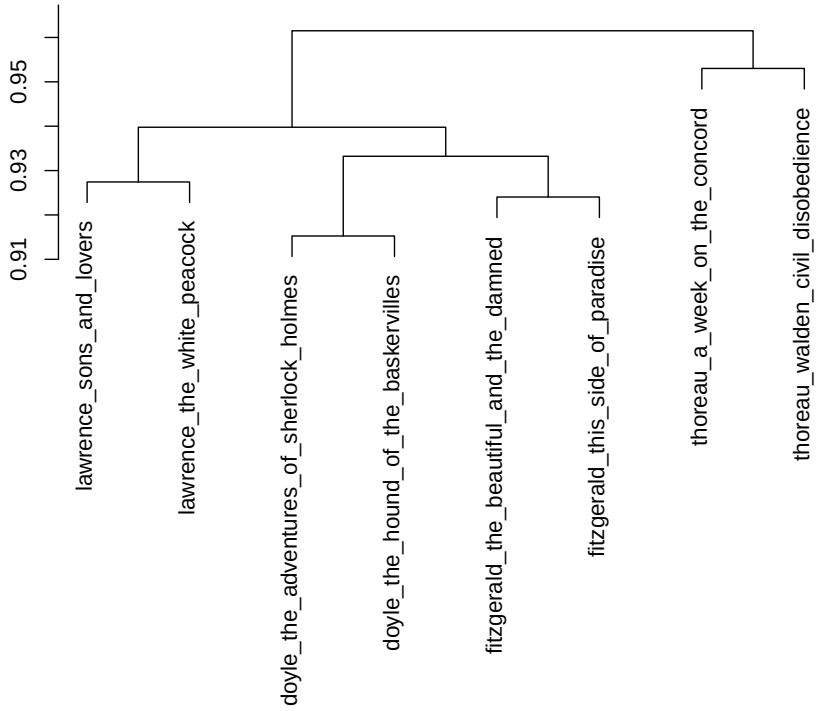

FIGURE 3.4: Average link clustering of eight books by english and american writers in utf-8 format.

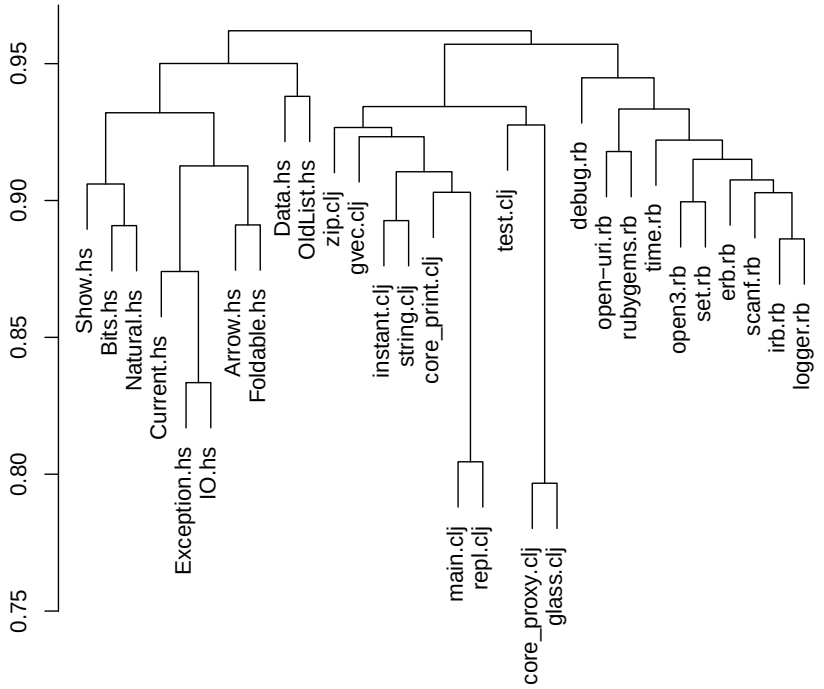

FIGURE 3.5: Average link clustering of 30 source files. The comments and blank lines were stripped using the *cloc* utility.

Chapter 4

Classification

Classification is a fundamental learning task. Given a training set of objects for which the class is known, we want to identify the class of an unknown object.

For example, we want to know if an incoming mail is *spam* (unsolicited message sent in bulk) or *ham* (genuine message.) We need an algorithm that extracts certain features from the incoming message, compares it to features extracted from, say, a million messages labeled as spam and a million labeled as ham, and decides to which class it belongs.

Most real-world applications take a domain specific approach. Spam filters count occurences of words and hyperlinks.

In this chapter, the normalized compression distance (2.2) is used to extract features from objects. No domain knowledge is used. Then, a support vector network is used to classify new data. The results are compared to literature.

4.1 Classifying file fragments

In forensics, file reconstruction is a big issue. Corrupted or wiped disks may contain scattered fragments of files without metadata. File fragment classification is a preliminary step in the reconstruction process.

The Govdocs1 dataset [11] is used throughout. It contains nearly one million files of various types, acquired by querying search egines with pseudo-random keywords.

Many authors, like [12], [13] and [14], try to classify compound file types (e.g. .doc or .pdf.) Since a .ppt file may contain a .doc file, or a .pdf file may contain a .jpg image,

.csv	0.91	0.94	0.94	0.86	0.95	1.01	1.00	1.01	1.02	1.02
.jpg	1.01	1.01	1.01	1.01	1.01	1.00	1.00	1.00	1.00	1.00

TABLE 4.1: Example feature vector for a .csv and a .jpg fragment. The first five anchors are .csv fragments, the last five are .jpg fragments. All fragments are randomly drawn from the Govdocs1 corpus.

there is no way of telling the filetype of a 512 or 4096 byte block originating from such a file. If you misclassify .pdf as .jpg, did the classifier make a mistake, or did you stumble upon an embedded .jpg withing a .pdf? The answer is unknowable. This oversight is noted in [15].

We will only use filetypes that are not compound, e.g. .html, .csv and .jpg.

4.1.1 Feature extraction

Many statistical learning models, like artificial neural networks or support vector machines, take as a training set fixed dimensional vectors that correspond to a label:

$$D = \{(\vec{x_i}, y_i) \mid \vec{x_i} \in \mathbb{R}^k, i = 1, 2, \ldots, n\} \tag{4.1}$$

The mapping of actual objects onto feature vectors is the crucial step. In [12], byte frequencies are used as features, so $\vec{x_i}$ is effectively a histogram, and y_i is one of .dll, .exe, .pdf, .mp3, .jpg.

In [16] a method is described to extract features from objects using the normalized compression distance. For each file type, (randomly) select some *anchors* from the corpus of file fragments. For simplicity, let's assume a binary classification problem of fragments that are either from a .jpg or a .csv file. After picking 5 anchor fragments for each type (totaling 10), we calculate the feature vector $\vec{x}$ for a file f as follows:

$$\vec{x} = (NCD(f, a_1), NCD(f, a_2), \ldots, NCD(f, a_{10})) \tag{4.2}$$

Table 4.1 shows an example feature vector of a fragment for both file types. It becomes clear why a fragment can be characterized this way: it's easy to see that the first fragment is a .csv file, since it is close to the first five anchors (the .csv anchors.) .jpg is a compressed format, so it has a distance of 1 from both the .csv and .jpg anchors.

4.1.2 Support vector machine

A support vector machine is a trainable binary classifier[1], first introduced in its current form in [17]. For an extensive introduction, see [18]. In all experiments, I used the Python library scikit-learn [19], which uses [20] and [21] internally.

In short, we want to separate the data points in the d-dimensional feature space with a hyperplane that maximizes the margin (Euclidian distance) to the closest data points.

New data points fall on either side of the hyperplane and are classified accordingly.

It is not possible to separate every set of points in all possible ways with a linear function (i.e. a hyperplane.) You can show that hyperplanes in $\mathbb{R}^n$ can, at most, separate $n+1$ points in all possible ways. So, if the training set is larger than the number of features plus one, it may be inevitable that points in the training set are misclassified by every hyperplane. The user-specified parameter C weighs the penalty for misclassified points in the training set. A higher value of C results in a hyperplane that misclassifies less training items. (This may actually hurt the classifier's predictive power, a phenomenon known as overfitting.)

With some mathematical footwork (see [18]), the optimization problem can be formulated in terms of only the dot products $\langle \vec{x_i}, \vec{x_j} \rangle$ of feature vectors.

Nonlinear-classifiers can be created by replacing all dot products with a *kernel function* $k(\vec{x_i}, \vec{x_j})$. This effectively applies a higher-dimensional transformation to the feature space. A hyperplane is then constructed in that space, so that non-separable data may become separable. This introduces extra parameters, is more memory intensive and is prone to overfitting, but often gives better results.

Two common kernel functions are the polynomial one

$$k(\vec{x_i}, \vec{x_j}) = (\langle \vec{x_i}, \vec{x_j} \rangle + r)^d$$

which introduces parameters r and d, and the radial basis function (RBF)

$$k(\vec{x_i}, \vec{x_j}) = e^{-\gamma ||\vec{x_i} - \vec{x_j}||^2}$$

which introduces γ.

[1] A classifier for more than two classes can be created from a combination of binary classifiers, for example with a one-vs-rest strategy.

File type	# of fragments
.csv	47782
.jpg	33630
.gz	11507
.gif	44367
.txt	19539
.log	25625
.xml	9540
.html	19934

TABLE 4.2: Number of 512-byte fragments per file type.

4.1.3 Preparing the data set

At the time of writing, it is possible to download samples of the Govdocs1 dataset [11], called threads, of 1000 files each. I used thread 0 through thread 7 as a corpus.

I created two corpora of fragmented files by chopping the files up into 512 or 4096 byte blocks, throwing away the first and last block of each file. The reasoning is that those blocks often contain header information, which is atypical for the file at large. It also ensures that all blocks are of equal size.

Table 4.2 shows the number of fragments of each file type.

4.1.4 Training the classifier

In each run of the experiment, randomly n_{anchors} fragments per filetype are selected as anchors, n_{training} fragments per type are selected as training data and n_{testing} fragments per type are selected as testing data.

4.1.4.1 Scaling the feature vectors

As is suggested in the scikit-learn documentation ([19]), we rescale the features (the NCDs with the anchors) to have zero mean and unit variance.

4.1.4.2 Parameter estimation

In [22], the authors describe a practical method for training a support vector machine, and show that a simple grid search on the classifier's parameters can dramatically improve results.

When using a radial basis function kernel, for example, we need to find the best combination of C and γ. First, we try all combinations $\{(C, \gamma) \mid C, \gamma \in 2^n, n = -10, -9, \ldots, 9, 10\}$. Then, after the best C and γ in that grid have been determined, we can perform successively more precise estimations by forming a finer grid around the current optimum.

The fitness of the parameters is assessed by k-fold cross validation. The training set is partitioned into k chunks of equal size. $k - 1$ chunks are used as training data and the remaining chunk as test data. In k runs, every chunk is used once as test data. The results are averaged.

4.1.5 Experiments

4.1.5.1 Classifying .csv and .jpg fragments

As a sanity check, we classify .csv and .jpg fragments. They should be easily distinguishable, since .jpg has high entropy (it is a compressed format) and .csv is very regular. Table 4.3 shows this to be the case. The recall rate is almost perfect, and it doesn't seem to depend on the amount of anchors per type.

anchors/type	.csv recall %	.jpg recall %
10	99.96	99.52
5	99.84	99.76
2	99.88	99.34

TABLE 4.3: The results were acquired by averaging recall rates over five independent runs. In each run, 1000 distinct training fragments and 500 distinct test fragments per file type were selected. After feature extraction, the support vector machine with an RBF kernel was trained by doing a grid search for $C \in \{2^1, 2^2, \ldots, 2^9\}$ and $\gamma \in \{2^{-8}, 2^{-7}, \ldots, 2^0\}$, optimized for the training set with a two-fold cross validation. All fragments are 512 bytes.

4.1.5.2 Classifying .gz and .jpg fragments

Now, we train the classifier on two compressed formats. The recall rate is very bad and .jpg gets many false positives. Training on .gif and .mp3 instead of .gz gives similar results. Increasing the training set from 1000 to 10000 sample fragments per type doesn't improve results, either.

anchors/type	.gz recall %	.jpg recall %
10	31.24	96.76
2	19.52	96.84

TABLE 4.4: The results were acquired by averaging recall rates over five independent runs. In each run, 1000 distinct training fragments and 500 distinct test fragments per file type were selected. The kernel is RBF, $C \in \{2^{-3}, 2^2, \ldots, 2^9\}$ and $\gamma \in \{2^{-8}, 2^{-7}, \ldots, 2^4\}$, optimized for the training set with a two-fold cross validation. All fragments are 512 bytes.

4.1.5.3 Classifying .csv, .html, .jpg and .log

Table 4.5 shows that classification still works very well on four different file types of low entropy. Note that adding more anchors now significantly improves results.

anchors/type	.csv recall %	.html recall %	.jpg recall %	.log recall %
2	94.00	92.32	98.84	87.28
10	98.48	95.04	99.12	95.24

TABLE 4.5: The results were acquired by averaging recall rates over five independent runs. In each run, 1000 distinct training fragments and 500 distinct test fragments per file type were selected. The kernel is RBF, $C \in \{2^0, 2^1, \ldots, 2^5\}$ and $\gamma \in \{2^{-7}, 2^{-6}, \ldots, 2^{-1}\}$, optimized for the training set with a two-fold cross validation. All fragments are 512 bytes.

Bibliography

[1] Ming Li and Paul M B Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2009.

[2] Charles H. Bennett, Péter Gâcs, Ming Li, Paul M B Vitânyi, and Wojciech H. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44(4): 1407–1423, 1998. ISSN 00189448. doi: 10.1109/18.681318.

[3] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitanyi. The Similarity Metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004. ISSN 0018-9448. doi: 10.1109/TIT.2004.838101.

[4] Rudi Cilibrasi and P. M B Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005. ISSN 00189448. doi: 10.1109/TIT.2005.844059.

[5] 0aa01af9101ad7f92eedf17301e3269ae811fb50 @ www.ncbi.nlm.nih.gov. URL `http://www.ncbi.nlm.nih.gov/genbank/`.

[6] Ying Cao, Axel Janke, Peter J. Waddell, Michael Westerman, Osamu Takenaka, Shigenori Murata, Norihiro Okada, Svante Pääbo, and Masami Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *Journal of Molecular Evolution*, 47(3):307–322, 1998. ISSN 00222844. doi: 10.1007/PL00006389.

[7] Rudi L. Cilibrasi and P. M B Vitnyi. A Fast Quartet tree heuristic for hierarchical clustering. *Pattern Recognition*, 44(3):662–677, 2011. ISSN 00313203. doi: 10.1016/j.patcog.2010.08.033.

[8] ruby @ github.com. URL `https://github.com/ruby/ruby`.

[9] clojure @ github.com. URL `https://github.com/clojure/clojure`.

[10] The Glasgow Haskell Compiler. URL `https://www.haskell.org/ghc/`.

[11] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6 (SUPPL.):2–11, 2009. ISSN 17422876. doi: 10.1016/j.diin.2009.06.016.

[12] Q Li, a Ong, P Suganthan, and V Thing. A Novel Support Vector Machine Approach to High Entropy Data Fragment Classification. *Proceedings of the South African Information Security Multi-Conference (SAISMC 2010)*, 2010. URL `http://www1.i2r.a-star.edu.sg/~vriz/Publications/WDFIA_SAISMC2010_SVM_High_Entropy_Data_Classification.pdf$\delimiter"026E30F$npapers2://publication/uuid/0A3CD98F-9AA4-4DAF-A2C7-95CB9E080FB4`.

[13] Cor J. Veenman. Statistical disk cluster classification for file carving. *Proceedings - IAS 2007 3rd Internationl Symposium on Information Assurance and Security*, pages 393–398, 2007. doi: 10.1109/IAS.2007.75.

[14] Stefan Axelsson. The normalised compression distance as a file fragment classifier. *Digital Investigation*, 7(SUPPL.):0–7, 2010. ISSN 17422876. doi: 10.1016/j.diin.2010.05.004.

[15] Vassil Roussev and Candice Quates. File fragment encoding classi fi cation d An empirical approach. 10:69–77, 2013.

[16] Rl Cilibrasi. Statistical inference through data compression. 2007. URL `http://www.narcis.nl/publication/RecordID/oai:uva.nl:217776`.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 08856125. doi: 10.1007/BF00994018.

[18] Christopher J C Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998. ISSN 13845810. doi: 10.1023/A:1009715923555. URL `http://www.springerlink.com/index/Q87856173126771Q.pdf`.

[19] Fabian Pedregosa, Ron Weiss, and Matthieu Brucher. Scikit-learn : Machine Learning in Python. 12:2825–2830, 2011.

[20] Rong-en Fan, Xiang-rui Wang, and Chih-jen Lin. LIBLINEAR : A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(2008):1871–1874, 2014.

[21] Chih-chung Chang and Chih-jen Lin. LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2: 1–39, 2011. ISSN 21576904. doi: 10.1145/1961189.1961199.

[22] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. *BJU international*, 101(1):1396–400, 2008. ISSN 1464-410X. doi: 10.1177/02632760022050997. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.