Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master Project

Seminar Report: Semantic Data-analysis and Automated Decision Support Systems

written by

Fabian Karl

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Course of Study:	Intelligent Adaptive Systems
Student-ID:	6886047
Advisors:	Anastasiia Novikova, Yevhen Alforov

Hamburg, March 26, 2018

Abstract

Huge amounts of new data are generated, analyzed and stored every day. Scientific research is creating data with every experiment or observation. Since storing more data means spending more money on physical data storage, compressing data becomes increasingly important.

The first issue this paper tackles is the compression of scientific data. Hardware is getting better by the month and compression algorithms are increasing in performance, but is there a way to increase compression performance without increasing the capability of hardware or software?

By finding optimal matches between datasets and compression algorithms based on the semantic information the dataset carries, one can increase the performance of compression algorithms without having to invest in better hard- or software. A benchmark study with scientific datasets was implemented, executed and analyzed thoroughly in this paper.

The second part of this work deals with the question of how to further automate the process of making decisions based on (un)structured and difficult to understand datasets. Machine learning in a unsupervised or supervised fashion is proposed, explained and theoretical use-cases are presented.

Contents

1. Motivation and Introduction

Scientific research is flourishing more than ever before. At the same time observation techniques and devices are getting better and more precise. More raw data can be extracted from experiments or from simply observing processes in nature. More data hopefully means more information and more information can lead to more knowledge about underlying mechanisms and structures.

More and larger data-files mean more storage space is needed and transmitting the data from one place to another takes longer. Compressing files before sending or storing them is one solution to this problem. But to counter the exponential growth of data, compression algorithms have to improve in performance just as the size of datasets increases. Better hardware assists this process, but cannot fully solve this problem alone.

A different approach to the problem of compressing files is to first look at the semantic information carried by the dataset before compressing it with the next best compression algorithm. Semantic information here means, what the data represents in a human sense. The semantics of words and sentences describes their meaning to humans. The same way data has a meaning to us. Does data describe a time-sequence of events, is it a catalog of entities, does it represent changes in a system or does it represent measurements from a medical test?

Using this information can help, when deciding about whether a dataset will have a high or low compression ratio or speed. Different datasets inherently have different characteristics when it comes to compression. Since most algorithms exploit repetitions of numbers or sequences, random data is almost incompressible. A dataset with lots of repetitions on the other hand can be compressed to a fraction of its original size. If knowing a dataset will compress very badly, one might save time on trying to compress it nevertheless, and instead focus on solving the problem in a different way. It goes the same way for the opposite case: If one assumes or knows, that a certain dataset can be compressed quite well, a big file-size will be less of a problem.

The second big advantage arises, when the semantic information of datasets is used to decide on an optimal compression algorithms for a specific dataset. Depending on the dataset, compression algorithms can have different performances. Finding optimal pairs between algorithms and semantic data-types can increase compression ratio as well as compression speed. To summarize, semantic information of different datasets should be exploited as much as possible, since it can save time and allow better compression performance without new hardware or software.

The first research questions of this work is of exploratory nature: How much information about the compression characteristics of a dataset can be extracted by a semantic data analysis?

This includes two subquestion:

1. How much do the semantic information and the structure of a dataset influence the general compression performance for this dataset?

2. How much advantage, with regards to compression ratio, compression speed and decompression speed can be achieved, by using an optimally suited compression algorithm for different datasets?

The second part of this work covers a related but different topic: Decision Support Systems (DSS). More data also means more complex methods for analyzing the data are needed. Analyzing huge, high-dimensional datasets without computer systems is hardly possible. Automated DSS are needed, that either help humans in analyzing data or to make automated decisions all together. This is one of many reasons why machine learning, big data and data science are getting more and more important. These fields try to facilitate or automate the process of interpreting and analyzing data as much as possible.

Human decision making is often based on intuition and gut feeling. Taking every single piece of information into account when making complex decisions is often impossible.

DSS have been around for a long time (Haagsma & Johanns, 1970) in order to assist human decision making. Recent advances in machine learning offer new ways of looking at how automated decision processes can be formalized. A theoretical overview on different unsupervised and supervised methods is given as well as practical examples in order to answer the second research question.

How can decision processes be automated and optimized? More concrete: What possibilities offers machine learning when it comes to DSS?

2. Scientific Data and Compression

In this Chapter various scientific datasets will be presented and described semantically (meaning the information they hold and represent). After that, compression will be explained in a general way. Especially the family of compression algorithms used in later the benchmark study will be explained in more detail.

2.1. Data

Classic compression corpora like the Canterbury corpus (Witten, Moffat, & Bell, 1999) or the Silesia corpus (Deorowicz, 2003) often use images, text, html-files and other multi media datasets in order to compare compression performance. In this study, only scientific datasets were used for the benchmark. So what restrictions are in place here?

The datasets used in this paper are all from scientific sources. This means, they are measurements from an executed experiment or a process in nature that was observed and logged. The data that is compressed, consists only of actual data-points, no header or other information is compressed. Most of the datasets used are arrays of float values, stored in a byte or float representation. Since different fields of science can have wildly different measurements or observations, the resulting dataset can have very varying structures and forms. It was tried to cover as many different scientific research areas as possible in order to create a more representative result. In this chapter every used dataset will be presented and described in a short fashion.

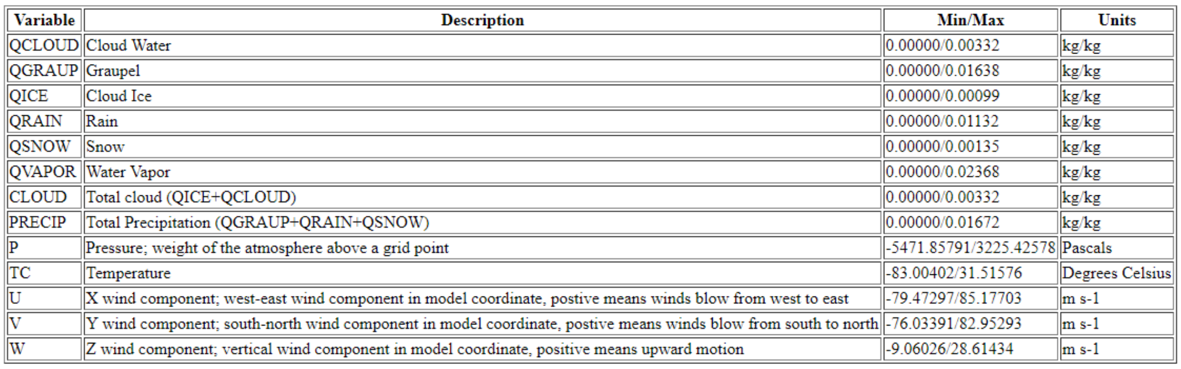
2.1.1. Climate Data

Observing the climate and the changes of climate on our planet can create an almost unlimited amount of data. Every part of the world offers hundreds of different observations every second. In order to analyze and work with all this data, it has to be stored first. The more efficient this is possible, the more data can be collected and studied.

As one representative for climate data, the Isabel Dataset [1] was chosen. Isabel was a hurricane in 2003 (Fredericks, Nagle, & Kranenburg, 2017). The whole dataset covers 13 different measurements over 48 time-steps, each measurement per time-step yielding 500x500x100 float values. The values are stored in a 4-byte representation. Figure 2.1 shows the different measurements, a short description and their range. Uncompressed every file is 97.657 KB large.

[1]`http://www.vets.ucar.edu/vg/isabeldata/`

[2]`http://www.vets.ucar.edu/vg/isabeldata/readme.html`



Variable	Description	Min/Max	Units
QCLOUD	Cloud Water	0.00000/0.00332	kg/kg
QGRAUP	Graupel	0.00000/0.01638	kg/kg
QICE	Cloud Ice	0.00000/0.00099	kg/kg
QRAIN	Rain	0.00000/0.01132	kg/kg
QSNOW	Snow	0.00000/0.00135	kg/kg
QVAPOR	Water Vapor	0.00000/0.02368	kg/kg
CLOUD	Total cloud (QICE+QCLOUD)	0.00000/0.00332	kg/kg
PRECIP	Total Precipitation (QGRAUP+QRAIN+QSNOW)	0.00000/0.01672	kg/kg
P	Pressure; weight of the atmosphere above a grid point	-5471.85791/3225.42578	Pascals
TC	Temperature	-83.00402/31.51576	Degrees Celsius
U	X wind component; west-east wind component in model coordinate, postive means winds blow from west to east	-79.47297/85.17703	m s-1
V	Y wind component; south-north wind component in model coordinate, postive means winds blow from south to north	-76.03391/82.95293	m s-1
W	Z wind component; vertical wind component in model coordinate, positive means upward motion	-9.06026/28.61434	m s-1

2

Figure 2.1.: The different measurements from the Isabel Datasets.

2.1.2. Geographical/Biological Data

Right next to the climate, the processes of nature and life are also ever changing and evolving. Measuring or observing the changes in plant or animal life is one of the oldest and most fundamental forms of research.

The dataset chosen from this research area comes from the University of Frankfurt, Germany and is a measurement about the growing rates of crops. The dataset contains the Monthly Growing Area Grids (MGAG) for 26 irrigated and rainfed crops. The dataset is called MIRCA2000 (Portmann, Siebert, Bauer, & Döll, 2008). It also contains float values stored as bytes.

Each of the 52 files contains information about one plant either being irrigated or rainfed. Each file covers 4320 x 2160 grid cells over the period of 12 months. The values in every grid cell represent the growing area in hectare. Uncompressed every file is 437.400 KB large. Because the files were very big, not all algorithms from the benchmark could be used when testing these files.[3]

`https://www.gfbio.org/data/search` offers further Geographical datasets.

2.1.3. Protein Corpus

The Protein Corpus (Abel, 2002) is a set of four files, which were used in the article 'Protein is incompressible' (Nevill-Manning & Witten, 1999). Every file describes a different protein with a string made up by a sequence of 20 different amino acids. Every amino acid is represented by one character. Protein is hard to compress, because it has little repeating sequences. This makes it a good candidate for compression benchmarks. The files are between 400 to 3.300 KB in size.

`http://gdb.unibe.ch/downloads/` offers further Chemical datasets.

[3]used algorithms: lzlib,xz,zstd,lzma,csc,brotli,lz4fast,pithy,snappy,lzo1x

2.1.4. Electrical Engineering and Robot Data

The used dataset comes from the testing of 193 Xilinx Spartan (XC3S500E) Field Programmable Gate Arrays (FPGAs)[4]. One hundred frequency samples were taken of each of the 512 ring oscillators on the Spartan board. The measurements were taken with the device operating in a regular room-temperature environment (around 25C°) and with an input voltage of the standard 1.2V. Every dataset contains only float-values and was saved as a csv-file with 395 Kb each.

Datasets from robotic sensors can be found on `http://kos.informatik.uni-osnabrueck.de/3Dscans/` or `http://asrl.utias.utoronto.ca/datasets/mrclam/#Download`, but were not tested for this work.

2.1.5. Medical Imaging Data

Medical imaging is a standard procedure in every hospital. There are different ways to gather informations about the inner life of a human or other animal non-invasively. Two example datasets are used in this benchmark.

The Lukas Corpus (Abel, 2002) contains the measurements of radiography (x-ray). The dataset includes four sets of files containing either two or three dimensional results from radiography. Only the data from the first set of measurements was used in this study. The tested files are a set of two dimensional 16 bit radiographs in DICOM format. The files are between 6.000 and 8.000 KB large.

A second dataset comes from a functional Magnetic Resonance Imaging (fMRI) study (Hanke et al., 2015). The data contains high-resolution, ultra high-field (7 Tesla) fMRI data from human test persons. Twenty participants were repeatedly stimulated with a total of 25 music clips, with and without speech content, from five different genres using a slow event-related paradigm. The resulting physiological phenomena in the brain are partly captured by the fMRI. Files are 275.401 KB large.

2.1.6. Astronomical Data

The last set of data comes from the scientific field of Astronomy. The used files contain the entire Smithsonian Astrophysical Observatory (SAO) Star Catalog [5] of 258,996 stars, including B1950 positions, proper motions, magnitudes, and, usually, spectral types in a locally-developed binary format described below. The catalog comes in two formates, one is 7.082 KB, the other one is 8.094 KB large.

[4] `http://rijndael.ece.vt.edu/puf/download.html`

[5] `http://tdc-www.harvard.edu/software/catalogs/sao.html`

2.2. Compression

Compression plays an important role in the digital world we live in today (Chen & Fowler, 2014). Storing and transmitting files, pictures, videos and information in general, is done in every second of every day. Since physical storage is expensive and the time for sending, receiving and storing data increases with its size, compression of data is crucial to a fast and economically working digital information system. Especially in scientific environments, huge amounts of data can result from experiments or observations.

Compression describes the process of decreasing the size of the data needed in order to save the same (or almost the same) information (Sayood, 2000; Lelewer & Hirschberg, 1987). When performing a data compression, the resulting size divided by the original size will result in the compression ratio. The ratio is often the most important metric of a compression.

The time it takes to compress a file is represented in the compression speed. It shows, how many Bytes the algorithm compresses every second. The decompression speed is measured the same way.

Further metrics that might be interesting are the memory usage of the algorithm during the compression and the energy usage of the program. Especially when compression is done on a small hardware device or a microprocessor.

Often the use-case and the situation determine the most important metric and thus the choice of the algorithms. Is the prime aspect of compression to store it with as little space-usage as possible? Or is it important that data is compressed fast, so that it can be send somewhere fast? Or maybe the hardware constraints do not allow for too much memory usage?

The fact, that different use-cases will have different demands on compression, makes it hard to define the optimal compression algorithm for one dataset. One could try to come up with some sort of score to weight all the different metrics together to one value. But this again would be highly subjective. For simplicity in this study, ratio will mostly be taken as the most important metric and will often define the optimal algorithms.

Mahoney (2012) offers a nice theoretical and practical overview on the most commonly used algorithms. It offers nice examples and gives a quick and intuitive understanding about compression. For the ones seeking more theoretical and thorough information on the topic of compression, Salomon and Motta (2010) is recommended.

2.2.1. Lossless and lossy compression

Compression algorithms can be separated in two classes: lossless and lossy compression. As the names already suggest, the first loses nothing where the second one does. The thing they lose is precision or accuracy of information.

This is often accepted, when the information, that is lost was redundant or not visible for humans anyway. Most famously, lossy compression is used for images e.g. JPEG (Acharya & Tsai, 2005), for videos e.g. MPEG (Sikora, 1997) and for audio, like the well known MP3 (Brandenburg & Stoll, 1994). The loss in information, is normally not noticeable for humans. This acceptance of loss on the other hand allows for much better and faster compression.

Lossless compression does not allow this. Every bit has to be the same after compression and decompression. This is often necessary, when a loss of information would destroy the data. A very obvious example is storing any sort of key or password. The exact replication of the original data is crucial. The benchmark used in this study only uses lossless compression algorithms, thus they are more important at this point. Famous lossless compression strategies are: Run-length Encoding (Robinson & Cherry, 1967), Huffman Encoding (Huffman, 1952), Prediction-by-partial matching (Cleary & Witten, 1984) or Lempel-Ziv Compression (Chapter 2.2.3).

2.2.2. Level of compression

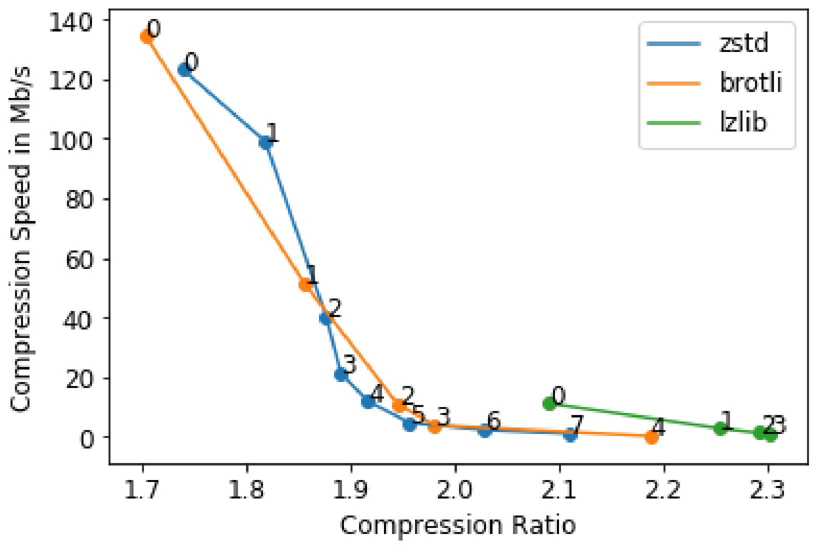


Figure 2.2.: Compression speed and ratio for different compression levels for zstd, brotli and lzlib.

Most compression algorithms have ’levels’ of compression, that the user can define. This allows the user to put an emphasis on either compression speed or compression ratio. The higher the level, the greater is the achieved compression ratio at a loss of compression speed. Compression levels enable flexible, granular trade-offs between compression speed and ratios. So one can use level 1 if speed is most important or the highest level if size is most important. For example Zlib offers nine compression levels and Zstandard 22 levels.

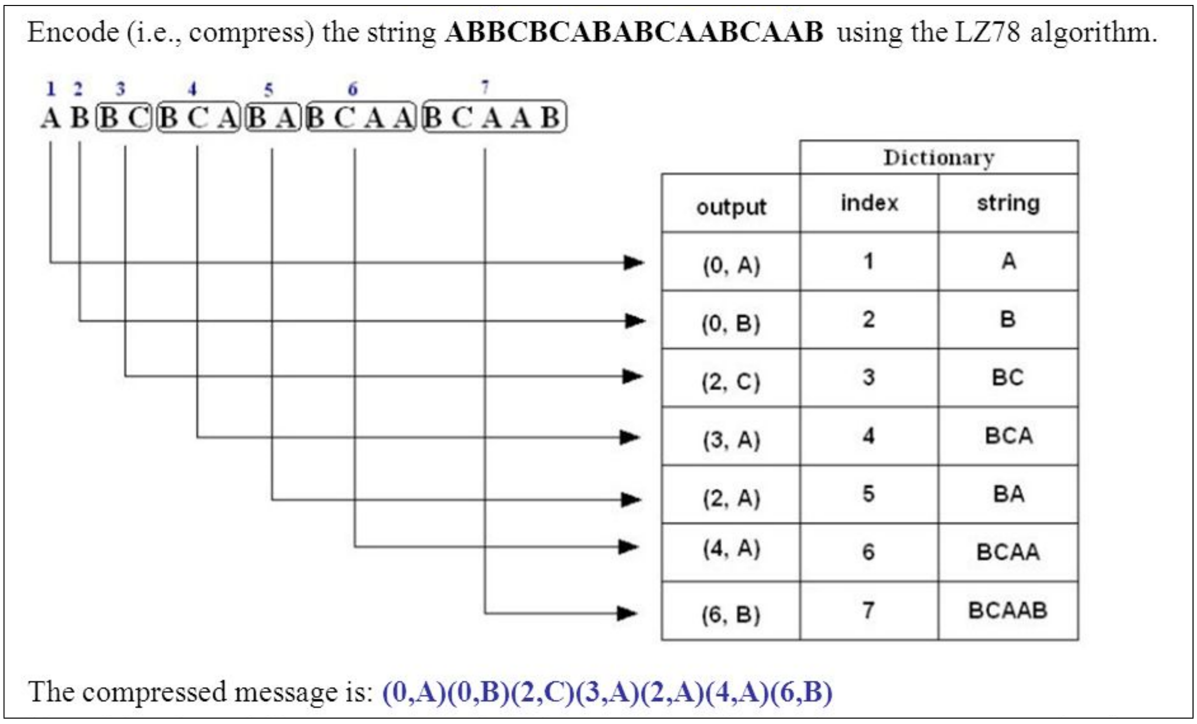
Figure 2.2 shows compression speed (y-axis) and the ratio (x-axis) for different levels of zstd, brotli and zlib. All three compression algorithms clearly show better compression ratio and worse compression speed with every increased level. The user then has the freedom of picking its preferred level for every use-case.

2.2.3. Lempel-Ziv algorithms

The Lempel-Ziv (LZ) compression method is a very famous compression strategy. The original LZ algorithm was published in 1977 by Ziv and Lempel, hence the name LZ77. They published LZ78 (Ziv & Lempel, 1978) in the following year. Many more algorithms based on the same idea were published later on. They are normally easily recognized by having 'lz' somewhere in their name.

Lempel–Ziv–Welch (LZW) (Welch, 1984), Lempel–Ziv–Storer–Szymanski (LZSS) (Storer, 1985), Lempel–Ziv–Markov chain algorithm (LZMA) (Igor Pavlov, 2004), Lempel–Ziv–Oberhumer (LZO)[6] and DEFLATE (Katz, 1991) all incorporate the same basic idea and add some additional features to mostly increase speed or ratio of the algorithm.

The LZ-family is important for this work, because the algorithms of the benchmark (**lz**Bench) used in this work are all LZ algorithms. This is why the LZ algorithms will be explained in more detail. LZ78 will be used as an example and is schematically shown in Figure 2.3.


Encode (i.e., compress) the string **ABBCBCABABCAABCAAB** using the LZ78 algorithm.

	Dictionary	
output	index	string
(0, A)	1	A
(0, B)	2	B
(2, C)	3	BC
(3, A)	4	BCA
(2, A)	5	BA
(4, A)	6	BCAA
(6, B)	7	BCAAB

The compressed message is: (0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)

(SlidePlay, 2016)

Figure 2.3.: Using LZ78 on a string. Every arrow symbolizes one step.

After reading a character, the algorithm checks in the lookup-table, if this character has been seen before. If not, it is added to the lookup-table. If it has seen it before, the next character is also looked at. If the combination of the first two characters has been seen before, then the first three characters are looked at, and so on. If at some point

[6]http://www.oberhumer.com/opensource/lzo/

the algorithm finds a sequence of characters it has not seen before, it will replace all the characters except the last one with the address of that character sequence (because it has to have seen that sequence before) and save it plus the new character. This can be seen in the 'output' column. By doing this, the algorithm builds longer and longer known sequences, that it can reuse.

3. Experiment: Compression Benchmark with lzBench

Compression benchmarks are quite popular and very useful. Because of the flood of different compression algorithms, that all claim to be faster and better than the rest, it is important to create a valid and reliable competition between them.

The Idea is simple: Different algorithms compress, decompress and validate the same data-sample on the same hardware. Important metrics like ratio, speed, memory usage, energy usage and more are collected and can be compared afterwards. This creates a valid and usable comparison between the algorithms and one can decide which algorithm suits one's needs best.

3.1. Experiment Design

3.1.1. lzBench

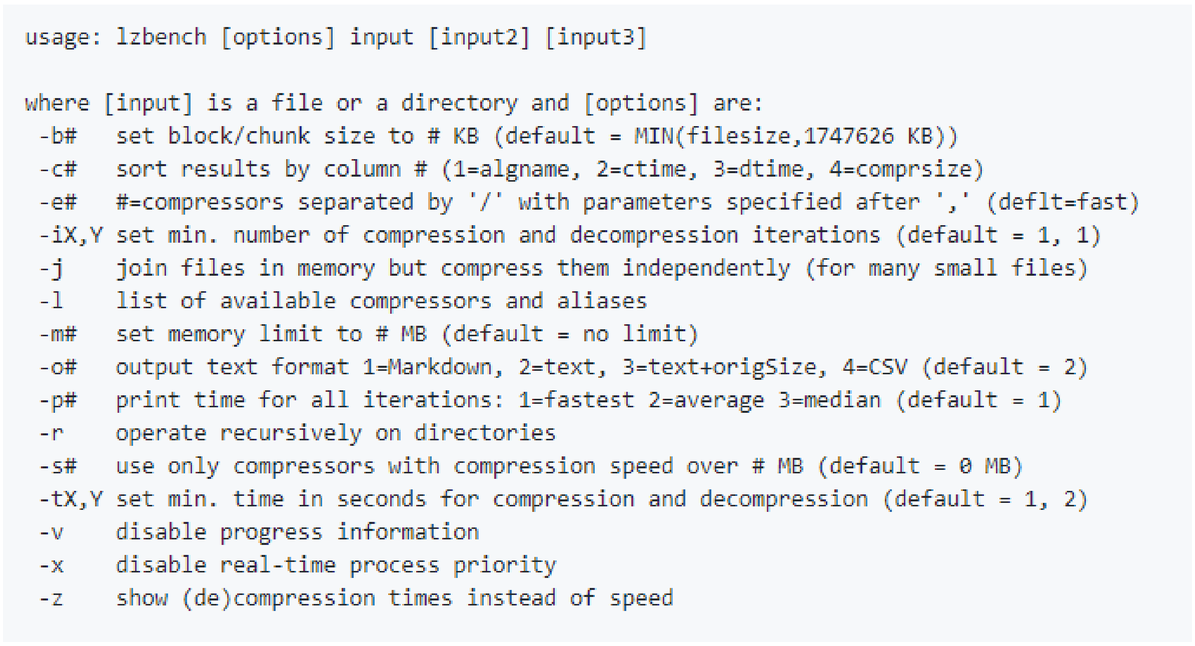
The benchmark chosen for this paper is lzBench[1]. The authors describe lzBench in their Github Readme with the following: *'lzbench is an in-memory benchmark of open-source LZ77/LZSS/LZMA compressors. It joins all compressors into a single exe. At the beginning an input file is read to memory. Then all compressors are used to compress and decompress the file and decompressed file is verified.'*

LZ77 stands for Lempel–Ziv 1977, LZSS for Lempel–Ziv–Storer–Szymanski and LZMA for Lempel–Ziv–Markov Algorihtm. All these algorithms are part of the Lempel-Ziv family described in Section 2.2.3. Even though, they are based on the same general principle, the three subfamilies have their own mechanisms and ideas in order to create better compression algorithms. The same way, every individual implementation can have a focus on a different aspect of compression, making it favorable for specific tasks or datasets.

LzBench was chosen because of its simplicity and its big number of algorithms (see Section 3.1.2). It is freely available and allows the user a high degree of manual adaptation. Figure 3.1 shows the options one can set when running the benchmark tool.

When executing lzBench, it will use all set algorithms (option '-e#') with all their possible levels of compression (see Chapter 2.2.2). It will run X compression runs and Y decompression runs for every algorithm. X and Y are defined by the option '-iX,Y'. '-p#' will then decide what measurement will be printed in the end (fastest, average, median).

[1]`https://github.com/inikep/lzbench/`



```
usage: lzbench [options] input [input2] [input3]

where [input] is a file or a directory and [options] are:
 -b#   set block/chunk size to # KB (default = MIN(filesize,1747626 KB))
 -c#   sort results by column # (1=algname, 2=ctime, 3=dtime, 4=comprsize)
 -e#   #=compressors separated by '/' with parameters specified after ',' (deflt=fast)
 -iX,Y set min. number of compression and decompression iterations (default = 1, 1)
 -j    join files in memory but compress them independently (for many small files)
 -l    list of available compressors and aliases
 -m#   set memory limit to # MB (default = no limit)
 -o#   output text format 1=Markdown, 2=text, 3=text+origSize, 4=CSV (default = 2)
 -p#   print time for all iterations: 1=fastest 2=average 3=median (default = 1)
 -r    operate recursively on directories
 -s#   use only compressors with compression speed over # MB (default = 0 MB)
 -tX,Y set min. time in seconds for compression and decompression (default = 1, 2)
 -v    disable progress information
 -x    disable real-time process priority
 -z    show (de)compression times instead of speed
```

Figure 3.1.: Options available for running lzBench.

Many more options can be altered as seen in Figure 3.1. The official GitHub page of the benchmark provides further information and examples.

3.1.2. Algorithms

LzBench uses 38 compression algorithms, all based on the idea of LZ-compression. Figure 3.1 and Figure 3.2 show all 38 algorithms with their respective version or date and a description, which comes directly from the authors of the algorithm. The reference for the description and the algorithm can be found in the last column. As one can see, most of the algorithms have 'lz' or 'deflate' somewhere in their name.

It should also be remarked, that many of the big IT-companies like Google, Apple or Facebook all have their own compression algorithms and that there is a constant rivalry which algorithm is the fastest and compresses the best. A nice example for this is Facebook's zstd (Collet & Turner, 2016) or Apple's LZFSE. This further shows the importance and prestige of compression algorithms nowadays. It also shows, that the basic ideas of LZ-compression are still very much used.

3.1.3. Methods

The whole process of selecting files for the benchmark, testing them and writing the results first to a file and then into the database was implemented in Python as part of this study. The used database to store the final results of the benchmark is SQLite (Hipp, 2015). Every single dataset is stored as a table where every row contains: Compressor name, Compression speed, Decompression speed, Original size, Compressed size, Ratio

Table 3.1.: The first 18 algorithms used by lzBench

No	Name	Version/Date	Description by Author	Github or Website
1	blosclz	10.11.2015	Blosc is a high performance compressor optimized for binary data.	https://github.com/Blosc/c-blosc/tree/master/blosc
2	brieflz	01.01.2000	BriefLZ is a small and fast open source implementation of a Lempel-Ziv style compression algorithm. The main focus is on speed, but the ratios achieved are quite good compared to similar algorithms.	https://github.com/jibsen/brieflz
3	brotli	12.12.2017	Brotli is a generic-purpose lossless compression algorithm that compresses data using a combination of a modern variant of the LZ77 algorithm, Huffman coding and 2nd order context modeling, with a compression ratio comparable to the best currently available general-purpose compression methods. It is similar in speed with deflate but offers more dense compression.	https://github.com/google/brotli
4	crush	v1.0	CRUSH is a simple LZ77-based file compressor that features an extremely fast decompression.	https://sourceforge.net/projects/crush/
5	csc	13.10.2016	A Loss-less data compression algorithm inspired by LZMA	https://github.com/fusiyuan2010/CSC
6	density	v0.12.5 beta	Superfast compression library. DENSITY is a free C99, open-source, BSD licensed compression library. It is focused on high-speed compression, at the best ratio possible. Streaming is fully supported. DENSITY features a buffer and a stream API to enable quick integration in any project.	https://github.com/centaurean/density
7	fastlz	v0.1	FastLZ - lightning-fast lossless compression library. FastLZ is very fast and thus suitable for real-time compression and decompression. Perfect to gain more space with almost zero effort.	https://github.com/ariya/FastLZ
8	gipfeli	13.07.2016	Gipfeli is a high-speed compression/decompression library aiming at slightly higher compression ratios (around 30 % less bytes produced for text) than other high-speed compression libraries.	https://github.com/google/gipfeli
9	glza	v0.8	GLZA is an experimental grammar compression toolset. Compression is accomoplished by using GLZAformat, GLZAcompress and GLZAdecode, in that order.	https://github.com/jrmuizel/GLZA
10	libdeflate	v0.7	ibdeflate is a library for fast, whole-buffer DEFLATE-based compression and decompression.	https://github.com/ebiggers/libdeflate
11	lizard	v1.0	Lizard (formerly LZ5) is a lossless compression algorithm which contains 4 compression methods: fastLZ4, LIZv1, fastLZ4 + Huffman, LIZv1 + Huffman	https://github.com/inikep/lizard
12	lz4/lz4hc	v1.8.0	LZ4 is lossless compression algorithm, providing compression speed at 400 MB/s per core, scalable with multi-cores CPU.	https://github.com/lz4/lz4
13	lzf	v3.6	LZF-compress is a Java library for encoding and decoding data in LZF format, written by Tatu Saloranta. Data format and algorithm based on original LZF library by Marc A Lehmann.	https://github.com/ning/compress
14	lzfse/lzvn	08.03.2017	Beginning with iOS 9 and OSX 10.11 El Capitan, Apple provides a proprietary compression algorithm, LZFSE. LZFSE is a Lempel-Ziv style data compression algorithm using Finite State Entropy coding. It targets similar compression rates at higher compression and decompression speed compared to deflate using zlib.	https://developer.apple.com/ documentation/compression/ data_compression
15	lzg	1.v0.8	LZG algorithm is a minimal implementation of an LZ77 class compression. The main characteristic of the algorithm is that the decoding routine is very simple, fast, and requires no memory.	https://github.com/mbitsnbites/liblzg
16	lzham	v1.0	LZHAM is a lossless data compression codec written in C/C++ (specifically C++03), with a compression ratio similar to LZMA but with 1.5x-8x faster decompression speed.	https://github.com/richgel999/lzham_codec
17	lzjb	2010	lzjb is a fast pure JavaScript implementation of LZJB compression/decompression. It was originally written by ""Bear"" based on the OpenSolaris C implementations.	https://github.com/cscott/lzjb

Table 3.2.: The second 18 algorithms used by lzBench

No	Name	Version/Date	Description by Author	Github or Website
18	lzlib	v1.8	The lzlib compression library provides in-memory LZMA compression and decompression functions, including integrity checking of the uncompressed data.	https://github.com/LuaDist/lzlib
19	lzma	v16.04	Lempel-Ziv-Markow-Algorithmus	
20	lzmat	v1.01	LZMAT is an extremely fast real-time lossless data compression library!	http://www.matcode.com/lzmat.htm
21	lzo	v2.09	Lempel-Ziv-Oberhumer	
22	lzrw	15.07.1991	Lempel-Ziv Ross Williams	
23	lzsse	14.05.2016		https://github.com/ConorStokes/LZSSE
24	pithy	24.12.2011	pithys roots can be traced back to Googles snappy compression library, but has diverged quite a bit.	https://github.com/johnezang/pithy
25	quicklz	v1.5.0	QuickLZ is the world's fastest compression library, reaching 308 Mbyte/s per core.	http://www.quicklz.com/
26	shrinker	v0.1	A data compression/decompression library for embedded/real-time systems.	https://github.com/atomicobject/heatshrink
27	slz	v1.0.0	SLZ is a fast and memory-less stream compressor which produces an output that can be decompressed with zlib or gzip. It does not implement decompression at all, zlib is perfectly fine for this.	http://www.libslz.org/
28	snappy	v1.1.4	Snappy is a compression/decompression library. It does not aim for maximum compression, or compatibility with any other compression library instead, it aims for very high speeds and reasonable compression.	https://github.com/google/snappy
29	tornado	v0.6a		
30	ucl	v1.03	UCL is a portable lossless data compression library written in ANSI C.	https://github.com/Distrotech/ucl
31	wflz	16.09.2015	wfLZ is a compression library designed for use in game engines. It is extremely cross platform and fast enough to use anywhere I've run into.	https://github.com/ShaneYCG/wflz
32	xpack	02.06.2016	XPACK is an experimental compression format. It is intended to have better performance than DEFLATE as implemented in the zlib library and also produce a notably better compression ratio on most inputs. The format is not yet stable.	https://github.com/ebiggers/xpack
33	xz	v5.2.3	XZ Utils is free general-purpose data compression software with a high compression ratio.	https://tukaani.org/xz/
34	yalz77	19.09.2015	Yet another LZ77 implementation.	https://github.com/ivan-tkatchev/yalz77
35	yappy	22.03.2014		https://github.com/richard-sim/Compression-Test-Suite/tree/master/CompressionSuite/Yappy
36	zlib	v1.2.11	zlib is designed to be a free, general-purpose, legally unencumbered, that is, not covered by any patents, lossless data-compression library for use on virtually any computer hardware and operating system.	https://zlib.net/
37	zling	10.04.2016	Libzling is an improved lightweight compression utility and library	https://github.com/richox/libzling
38	zstd	v1.3.3	Zstandard is a real-time compression algorithm, providing high compression ratios. It offers a very wide range of compression / speed trade-off, while being backed by a very fast decoder	http://facebook.github.io/zstd/

and Filename. Every table contains one row for every used level of algorithm. When two levels have too similar results, only one of them is stored. On average every benchmark run creates 170 to 200 rows of measurements. LzBench is mostly written in C and C++, with an easy to use shell interface. Executing C programs from Python is not impossible, but executing a shell command is much more handy.

Implemented methods for testing include:
A method ($lzBench\_on\_folder$) that will take an input-file or folder and an output-folder as arguments, will then run the benchmark and save the result file in the given directory. Options for lzBench can also be given as parameters. This method creates one bash script for every file, that is executed on a cluster. Number of nodes to use and which partition to use can also be given as parameters for the method. By doing this, the procedure of testing can easily be parallelized on different nodes.

$import\_folder\_to\_DB$ will import the given folder into the given sqlite3 database. Existing tables will be overwritten.

The same Python script contains different methods for analyzing the results stored in the database.

get_table will return the whole table for a given table name. It will be sorted by a given metric (default = ratio) and an n can be given if only the top n rows shall be returned.

get_best_algorithms will return all tested data samples with their n best algorithms regarding a chosen metric (ratio, compression speed, decompression speed). By adding a regex one can filter the tables that are returned.

$get\_best\_files\_per\_algorithm$ will return the previously mentioned dictionary reversed. This function takes the best algorithms for every dataset and creates a dictionary mapping from algorithm to datasets, that this algorithm had the best performance on.

get_averages returns a list of all datasets and their average ratio, compression speed and decompression speed over all algorithms.

3.2. Results

Visualizing and presenting the results of a compression benchmark can be tricky. The benchmark includes different algorithms, which all measure different metrics. And every benchmark is performed on various datasets. This means, three important factors interact (algorithm, metric, dataset) and the different effects between those can be observed. It is very hard, though to show all of them in one plot or table, since at least three dimensions would be needed for that. Effects that might be interesting are: How well is the performance of different algorithms on one dataset? Do different sorts of data share certain characteristics when it comes to compression? What algorithms show overall the best performance? Is there a relation between the different metrics?

In order to answer these questions in the discussion, the results are shown and plotted from different angles in this chapter.

The first view on the data shows the relationship between one dataset and different algorithms and different metrics. Table 3.3 shows part of a single result from lzBench with all algorithms selected (-eall). This is also how every table in the database looks like. The table is sorted by ratios and the complete table can be seen in the Appendix in Figure A.1. The dataset used to create this table is one of the Isabel datasets. The first row of the table shows, that lzlib 1.9 -9 yields the best ratio of 7.86. To create this ratio, the compression speed decreases to 1.23 MB/s and the decompression speed is 56.68 MB/s. The next rows show slightly worse ratios with better compression speed. At row 102 for example, blosclz 2015 -9 has a ratio of 6.37 with a compression speed of 556.74 MB/s and a decompression speed of 1085.87 MB/s. The very last row shows the measurements of copying the data without compression. This creates a reference value for the compression and decompression speed.

Table 3.3.: Part of the results plotted in Figure 3.2 for one dataset from the Isabel datasets

No.	Compressor name	Compression speed	Decompression speed	Original size	Compressed size	Ratio	Filename
0	lzlib 1.8 -9	1.23	56.68	100000000	12718753	7.86	CLOUDf08.bin
1	brotli 2017-03-10 -11	0.72	288.82	100000000	12731988	7.86	CLOUDf08.bin
2	lzlib 1.8 -0	28.21	52.01	100000000	13043559	7.67	CLOUDf08.bin
3	xz 5.2.3 -9	8.16	82.42	100000000	13071183	7.65	CLOUDf08.bin
4	lzma 16.04 -9	5.67	76.84	100000000	13071640	7.65	CLOUDf08.bin
5	lzma 16.04 -0	26.78	86.33	100000000	13068579	7.65	CLOUDf08.bin
6	xz 5.2.3 -6	7.99	84.07	100000000	13065817	7.65	CLOUDf08.bin
7	csc 2016-10-13 -5	9.83	58.56	100000000	13086055	7.64	CLOUDf08.bin
...	...	...	...	...	...	...	...
102	blosclz 2015-11-10 -9	556.74	1085.87	100000000	15707744	6.37	CLOUDf08.bin
103	fastlz 0.1 -2	439.01	1753.47	100000000	15720163	6.36	CLOUDf08.bin
104	slz_zlib 1.0.0 -1	454.04	475.85	100000000	15745116	6.35	CLOUDf08.bin
105	slz_zlib 1.0.0 -2	413.92	476.32	100000000	15755240	6.35	CLOUDf08.bin
106	slz_zlib 1.0.0 -3	461.42	451.4	100000000	15757749	6.35	CLOUDf08.bin
107	lzo1y 2.09 -1	1592.93	892.5	100000000	15769310	6.34	CLOUDf08.bin
108	ucl_nrv2b 1.03 -6	33.43	391.42	100000000	15775324	6.34	CLOUDf08.bin
...	...	...	...	...	...	...	...
169	blosclz 2015-11-10 -1	1037.62	1536.04	100000000	43275920	2.31	CLOUDf08.bin
170	lizard 1.0 -40	157.79	3070.95	100000000	46405992	2.15	CLOUDf08.bin
171	lizard 1.0 -20	447.72	3552.96	100000000	46405992	2.15	CLOUDf08.bin
172	yappy 2014-03-22 -1	106.87	1895.22	100000000	48855449	2.05	CLOUDf08.bin
173	lizard 1.0 -10	497.55	2889.2	100000000	49093035	2.04	CLOUDf08.bin
174	lizard 1.0 -30	465.87	3467.25	100000000	49093035	2.04	CLOUDf08.bin
175	density 0.12.5 beta -1	819.16	729.42	100000000	60061456	1.67	CLOUDf08.bin
176	memcpy	4537.5	4271.75	100000000	100000000	1.00	CLOUDf08.bin

Plotting the columns 'Compression speed', 'Decompression speed' and 'Ratio' for the whole table will result in the graph from Figure 3.2. The figure shows all algorithms

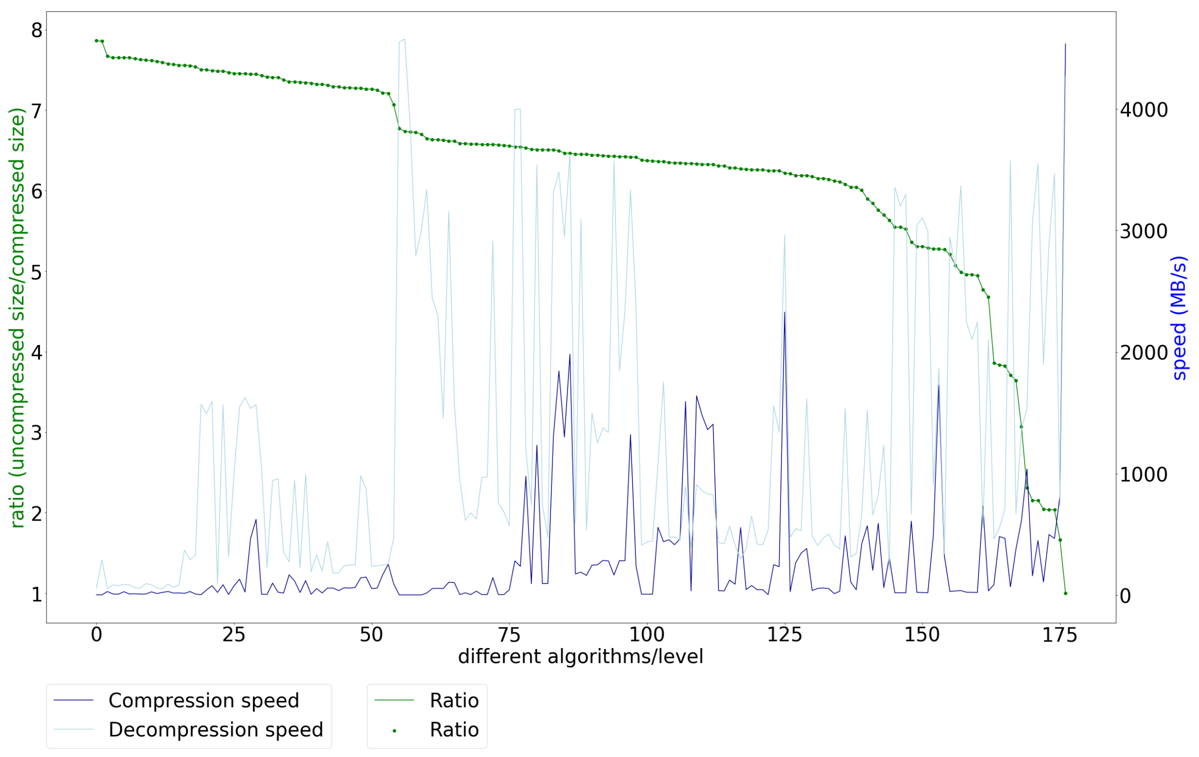


Figure 3.2.: Plot of ratio, compression and decompression speed for one of the Isabel datasets.

(many with different levels) sorted by ratio. Every green dot on the green line is one algorithm. The green line belongs to the left axis and shows the compression ratio. The two blue lines represent the compression and decompression speed. In order to see the speed of an algorithm, one has to go down in a theoretical vertical line from every green dot until one intersects the blue lines. Then the three measurements for ratio, compression speed and decompression speed can be seen. Speed is measured in Megabytes per second (MB/s).

This view shows the effects of different metrics and different algorithms for one dataset. It lacks the information of comparing many datasets with each other, though.

To create a two dimensional view of all algorithms, with different datasets, one has to somehow eliminate the factor 'metric'. The easiest way to do that, is to just decide on a metric beforehand. Ratio was chosen as a metric for the next tables and figures, since it is often the most important metric. When fixating the metric, one can now look at the best performances of different algorithms for different datasets.

Table 3.3 shows the four best performing algorithms for datasets from different scientific data-collections. One can see, that a lot of datasets were best compressed by different levels of lzlib. FMRI Imaging datasets, though, are best compressed by csc 2016 level -1 or -5. Protein datasets, which are supposed to be hard to compress, were compressed best by brotli 2017 level -11 followed by zstd 1.3.1 and csc 2016. The table only shows a part of all the datasets used in the benchmark. The complete list of datasets can be found in the Appendix in Tables A.2 to A.8.

Table 3.3 offers an overview over what algorithm has the best ratio for which dataset. It also offers the possibility to count the 'winning algorithms' from every dataset and create a histogram over the algorithms having the best ratio for at least one dataset. This can be seen in Figure 3.3. The algorithms are aligned on the x-axis and the number of times they achieved the best ratio is shown on the y-axis. The figure shows, that as expected from Table 3.3, different levels of lzlib yield the best ratio in this study. Lzlib levels -9 , -0, -6 and -1 show the best ratio for most of the tested datasets. csc, brotli, xz, lzma and zstd also achieve the best ratio for at least one dataset.

Both these views on the data can easily be replicated with different metrics, offering a different view on the results.

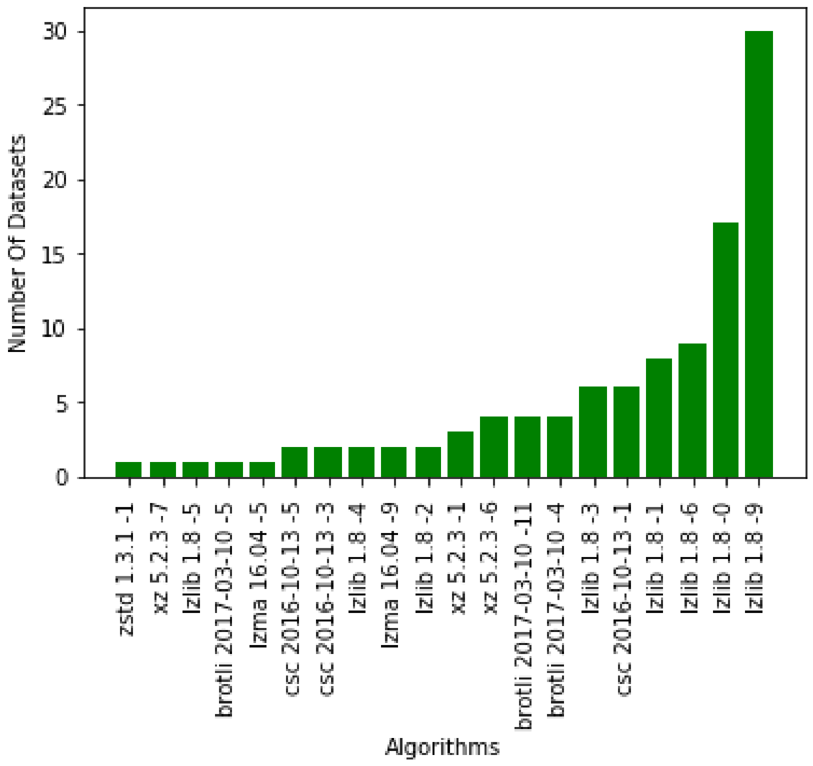


Figure 3.3.: Histogram for algorithms/levels and the number of times, they achieved the best ratio for a dataset.

Another way to view more dimensional data in two dimensions, is to average over one variable. Averaging over all measurements from all algorithms will reduce the table of around 200 measurements to one row of averaged values. Going even further, one could now take these averages and calculate another average over all the datasets, that come

Table 3.4.: Four best algorithms regarding ratio for different datasets.

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Electric_D067966	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Electric_D070707	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Electric_D070835	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6
Electric_D059546	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Electric_D061283	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
FMRI_auditory_perception-01	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
FMRI_auditory_perception-03	csc 2016-10-13 -1	lzlib 1.8 -0	brotli 2017-03-10 -5	zstd 1.3.1 -8
FMRI_auditory_perception-04	csc 2016-10-13 -1	lzlib 1.8 -0	brotli 2017-03-10 -5	zstd 1.3.1 -8
FMRI_auditory_perception-05	csc 2016-10-13 -5	csc 2016-10-13 -3	lzlib 1.8 -9	xz 5.2.3 -9
FMRI_auditory_perception-06	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
FMRI_auditory_perception-08	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
Lukas_breast_0	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Lukas_thorax_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_pelvis_0	csc 2016-10-13 -3	csc 2016-10-13 -5	csc 2016-10-13 -1	lzlib 1.8 -9
Lukas_thorax_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9
Lukas_breast_1	lzlib 1.8 -6	lzlib 1.8 -9	lzma 16.04 -9	xz 5.2.3 -6
Lukas_pelvis_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_sinus_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_food_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9
Isabell_QRAINf01	lzlib 1.8 -9	brotli 2017-03-10 -11	lzma 16.04 -9	xz 5.2.3 -6
Isabell_Pf01	lzlib 1.8 -3	lzlib 1.8 -6	lzlib 1.8 -9	xz 5.2.3 -6
Isabell_QGRAUPf01	lzlib 1.8 -9	xz 5.2.3 -6	xz 5.2.3 -9	lzma 16.04 -9
Isabell_Vf01	lzlib 1.8 -3	lzlib 1.8 -6	xz 5.2.3 -6	lzlib 1.8 -9
Isabell_CLOUDf01	lzma 16.04 -5	lzlib 1.8 -9	lzlib 1.8 -3	lzlib 1.8 -6
Isabell_QVAPORf01	csc 2016-10-13 -5	csc 2016-10-13 -3	csc 2016-10-13 -1	lzlib 1.8 -3
Isabell_QICEf01	lzlib 1.8 -9	xz 5.2.3 -6	lzlib 1.8 -3	lzma 16.04 -5
Isabell_PRECIPf01	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	lzma 16.04 -5
Isabell_TCf01	csc 2016-10-13 -3	csc 2016-10-13 -5	csc 2016-10-13 -1	lzlib 1.8 -3
Protein_mj	brotli 2017-03-10 -11	zstd 1.3.1 -18	zstd 1.3.1 -22	zstd 1.3.1 -1
Protein_sc	brotli 2017-03-10 -11	csc 2016-10-13 -5	zstd 1.3.1 -22	csc 2016-10-13 -3
Protein_hi	brotli 2017-03-10 -11	zstd 1.3.1 -18	zstd 1.3.1 -22	zstd 1.3.1 -1
Protein_hs	brotli 2017-03-10 -11	zstd 1.3.1 -22	csc 2016-10-13 -5	csc 2016-10-13 -3
Stars_SAOra	xz 5.2.3 -6	xz 5.2.3 -9	lzma 16.04 -5	lzlib 1.8 -6
Stars_SAO	lzma 16.04 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6
irrigatedcrop22	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7
irrigatedcrop23	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7
irrigatedcrop24	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfedcrop12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfedcrop13	lzlib 1.8 -4	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2
rainfedcrop14	lzlib 1.8 -3	lzlib 1.8 -4	lzlib 1.8 -5	lzlib 1.8 -6
rainfed_crop_15	lzlib 1.8 -6	xz 5.2.3 -7	xz 5.2.3 -6	xz 5.2.3 -8

from the same 'area' and are semantically similar. By doing this, a very compact view, on how well datasets with similar structure and semantic information are compressible, is created. Table 3.5 shows the result of what was just described. The averages over all datasets from one domain of all three metrics can be seen. The table is again sorted by ratio. The Star Catalog shows the worst results for all three measurements. The Protein and FMRI dataset also show a low ratio (around 1.4) and speed. Lukas, Electrical Engineering and Isabel datasets all have a ratio around 3 to 3.4 and a quite

Table 3.5.: Average performance of all algorithms over different datasets.

No.	Name	Ratio	Compression Speed	Decopmression Speed
1	Stars	1.258214	104.467712	674.257881
2	Protein	1.407821	219.244082	1111.206667
3	FMRI	1.482281	156.532802	666.815015
4	Lukas	3.029618	137.818054	730.667927
5	Electric	3.227880	159.069449	783.240035
6	Isabel	3.478683	293.490520	1186.734199
7	MIRCA rainfed	90.986818	3377.459896	3125.643697
8	MIRCA irrigated	471.882460	3879.590057	3293.802149

low compressions speed (around 150). The two MIRAC datasets show the highest ratios (90 and 470) and the fastest compression (more than 3300 MB/s). The two MIRCA datasets were separated, because their average ratio was quite different.

Looking at the mean is normally one of the first steps when analyzing results or data in general. In order to get more information about the underlying structures, more analysis can be helpful, though. Variance of the data and the detection of outliers can be visualized in a boxplot.

A boxplot consists of a box, a line inside the box, two whiskers and possibly dots on top or below the whiskers. The dots mark outliers, that are further than 1.5 times the interquartile range away from the top or bottom of the end of the box. Interquartile range equals the height of the box. The bottom and top of the box always show the first and thirds quartile of the data. This means inside the box lie 50% of all data points. The line in the box is the median of the distribution.

boxplots show many important informations about a distribution of data. Most importantly, they show the variance. Two distributions can have the same mean or median, but a completely different variance. boxplots allow to see that.

Figure 3.4 shows the data for ratio and compression speed from Table 3.5 in a boxplot. Both sub-datasets from the MIRCA2000 dataset have shown the highest ratio. Their results are so much higher than the rest of the results, that in order to plot them all in one figure, a logarithmic scale on the y-axis had to be used. Aside form that, one can see that the datasets from the Field Programmable Gate Arrays and from the Lucas dataset are compressed better then the rest as well. The star catalog and the Protein sequences have the worst ratio. FMRI measurements and Isabel datasets show slightly better results. The Isabel datasets have a very high variance.

Compression speed and ratio show similar trends, but some differences. The star catalog has a very low compression speed, whereas the Protein dataset shows a faster compression speed than Electric, Lukas and FMRI. Isabel datasets are showing the fastest average compression speed after the MIRCA datasets. The MIRCA datasets are more than ten times faster than the Isabel datasets.

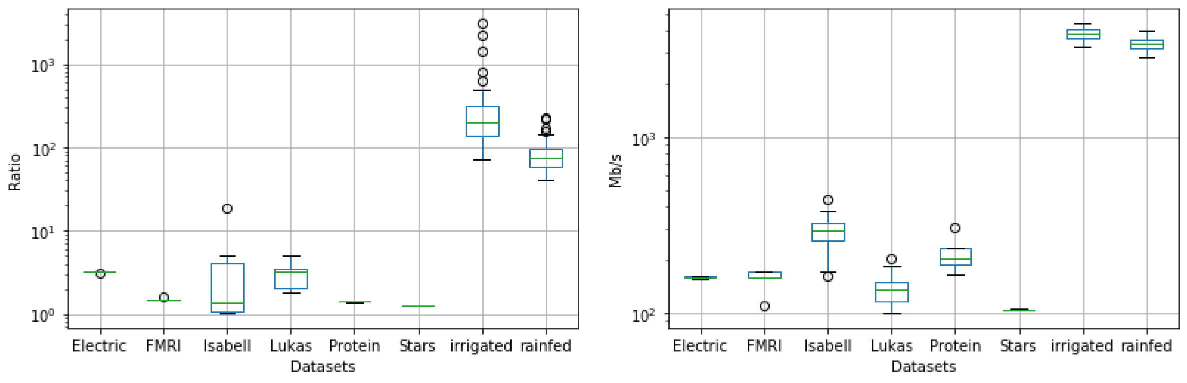


Figure 3.4.: boxplot showing Ratio (left) and Compression Speed (right) for all algorithms for different datasets.

3.3. Discussion

In the last section the results were shown and explained from different point of views. Now the important question is, what information can be drawn from these results about semantic data analysis and compression. In the same order as the results were presented, they will now be discussed and put into context.

Figure 3.2 nicely shows, that compressing files almost always results in some sort of ratio-time trade-off. The more focus is put on achieving a very small compressed output-file, the more time and computational resources are needed. This phenomena seems very reasonable and can be observed in computer science frequently. The main performance of an algorithm can be increased, but there is no free lunch (Wolpert & Macready, 1997). More computational power in form of energy consumption, memory usage and/or time are needed. On the one hand, this can be seen as a disadvantage because there is no optimal solution that maximizes all the desired output metrics. On the other hand it gives the user the freedom to decide for every use-case, what metric should be focused on. It was explained in Section 2.2.2 that compression algorithms often offer the possibility of selecting a level to decide within the same algorithm, if the focus should be put more on the resulting ratio or the compression and decompression speed.

It makes it hard, to compare compression algorithms in a general fashion though. To solve this question of general performance of an algorithm two methods can be used: Either all the metrics have to be put next to each other and looked at together or a score is defined that takes in the information from all the metrics and returns one value, that can be used for a general comparison. This score could for example look like:

$$ratio * x + compression\_speed * y + decompression\_speed * z + further\_metrics * c$$

Assuming that all metrics are normalized and preprocessed in a way that bigger values show better performance, then this score can be calculated for different algorithms and compared directly. The weights for the different metrics are still to be decided by the user.

Taking a closer look at Table 3.4, it seems, that even though lzlib overall shows the best ratio, certain complicated and not well structured datasets are better compressed by other algorithms. The star catalog for example shows little repeating structure, due to it being a catalog, and were best compressed by other algorithms than lzlib. When the data was more structured like in the climate measurements, or the crop growing datasets (irrigatedcrop and rainfedcrop) lzlib outperforms the other algorithms most of the time with regards to ratio.

When averaging over the compression algorithms and also the different datasets from the same dataset corpus, one can have a close look at the overall performance on different classes of datasets. Table 3.5 and Figure 3.4 both offer that view. Especially the boxplot shows clearly, that some datasets with the same underlying semantic information are compressible very good and very fast. It can be assumed, that the MIRCA datasets either have a lot of repeating patterns or a lot of repeating numbers. It makes sense, when thinking about what the datasets represent. The monthly growing areas in hectares for the same plant is likely quite similar at different positions. If the crop is growing very similar, the resulting numbers are all very similar and might often be the same. This allows for a easy and good compression. Protein and Star datasets on the other hand have almost no repeating features and have a very low ratio compared to the rest of the datasets. Especially the Star catalog has a ratio close to one, meaning that it is hardly compressed at all. A catalog often is a list of different items. There is not much connection between the items, that show patterns or repeating numbers.

The boxplot shows similar results as Table 3.4. What can be seen in the right figure is that the Isabel dataset has a very high variance. This is due to the fact, that the Isabel datasets contains 13 different measurements, all merged into one average. The high variance shows, that these different measurements have different properties when it comes to compression. Testing all 13 datasets individually would be very time intensive and create many more different datasets. Since there is some assumed similarity between the different Isabel datasets and to save time and space, the datasets were averaged together. The very low variance of the Star dataset is because it only contains two files, which are two representations of the same catalog. Protein, Electric and FMRI datasets all have a very low variance and all seem to be semantically very similar. All datasets show a low variance when it comes to compression speed.

This discussion shows, that the semantic information carried by different datasets is highly responsible for the general compressibility of a dataset and also has a big impact on how well compression algorithms will perform on said dataset. By knowing the semantic information of a dataset beforehand, one could now assume if the file will compress well and what algorithm to use.

3.4. Conclusion

This chapter covered the compression of different scientific datasets, analyzing the results from different angles and discussing the implications of the results.

It was clearly visible, that different datasets with different semantic information react very differently to compression. It was also shown, that certain algorithms perform better or worse depending on the dataset. Knowing where the dataset comes from and what sort of semantic information it carries, can thus be highly informative about the general compressibility and maybe even the performance of different algorithms.

The connection between the datasets used in this study and the field of research they represent should only be drawn with care. Mordern scientific research in one filed is no longer limited to one or two different methods of measuring or observing, resulting in few different datasets. On the contrary, as seen in the Isabel dataset, one scientific filed can create various datasets with different measurements and data-structures.

This chapter builds a foundation for testing and analyzing scientific datasets in order to find optimal compression algorithms for different datasets from scientific sources. The implementations created alongside this work are general and can be used for different benchmarks and datasets, in order to create an easy and fast way to benchmark various algorithms against many datasets.

4. Decision Support Systems with Machine Learning

This chapter will cover Decision Support Systems (DSS) (Power, Sharda, & Burstein, 2015; Arnott & Pervan, 2008). DSS is a quite broad term, covering basically all systems from facilitating decisions slightly to systems making decisions on their own. Support Systems also have changed a lot in the past 50 years, and will keep changing in the future (Shim et al., 2002). This paper will focus on recently developed DSS (Marakas, 2003): Systems that are sophisticated and proactive. They try not only to aid decision making, but to propose a decision for the user or to even make a decision on its own (Bojarski et al., 2016).

In the end of this chapter, the use-case of choosing an optimal compression algorithm for new and unknown data will be used to give theoretical ideas a more practical use-case.

4.1. Decisions

DSS can look very differently. They can be a simple program that restructures, visualizes or highlights data (Li, Feng, & Li, 2001). They can make easy mathematic operations like averaging over a list of values. In this case, the decision itself will still be placed in human hands though. The system will not propose anything, it just makes the data more easily accessible. In machine learning, unsupervised learning falls in this category. When the only thing available is data, clustering can help to structure the data in a more understandable way. Then a human can make a decision.

On the other hand one can think of a DSS that is aware of the different decisions that are available (Goldman, Hartman, Fisher, & Sarel, 2004). It could for example be explicitly implemented, that there are 38 different compression algorithms to choose from in the end. In many use-cases this decision will be binary. A system predicting the development of shares on the stock market (Zhang, Fuehres, & Gloor, 2011; Huang, Nakamori, & Wang, 2005) for example should produce either a one for buying or a zero for not-buying. Systems like that are often stochastic systems, that will return a probability distribution over all existing categories or decision-options. Every decision process can be modeled as a categorization problem. There is some sort of information available in order to make a decision. The information will be used as the input for the categorization algorithms and the decision will be the discrete output. So basically every decision is a function mapping from information (input) to a decision (discrete value). If one has a dataset of inputs and correct outputs, that data can be used for a supervised learning model. These models will optimize a function to map every input as close as possible to every given output.

4.2. Unsupervised Learning

Unsupervised Learning is one of the two main classes of machine learning (Hastie, Tibshirani, & Friedman, 2009). It describes the discovery of structures or underlying informations in data. One can use unsupervised learning for unlabeled data. That means just the plain data. Labeled data on the other hand describes data, that has already been solved in a way (Seeger et al., 2001). Labeled data is only used to train a model. More information will be given in Section 4.3.

Clustering data (Berkhin, 2006) can allow humans to make better judgment about data and maybe help them come to a decision. Clustering means, to split the dataset into a number of clusters or groups of data-points. A very famous and simple algorithm to do this is the k-Means Algorithm (MacQueen, 1967; Lloyd, 1982).

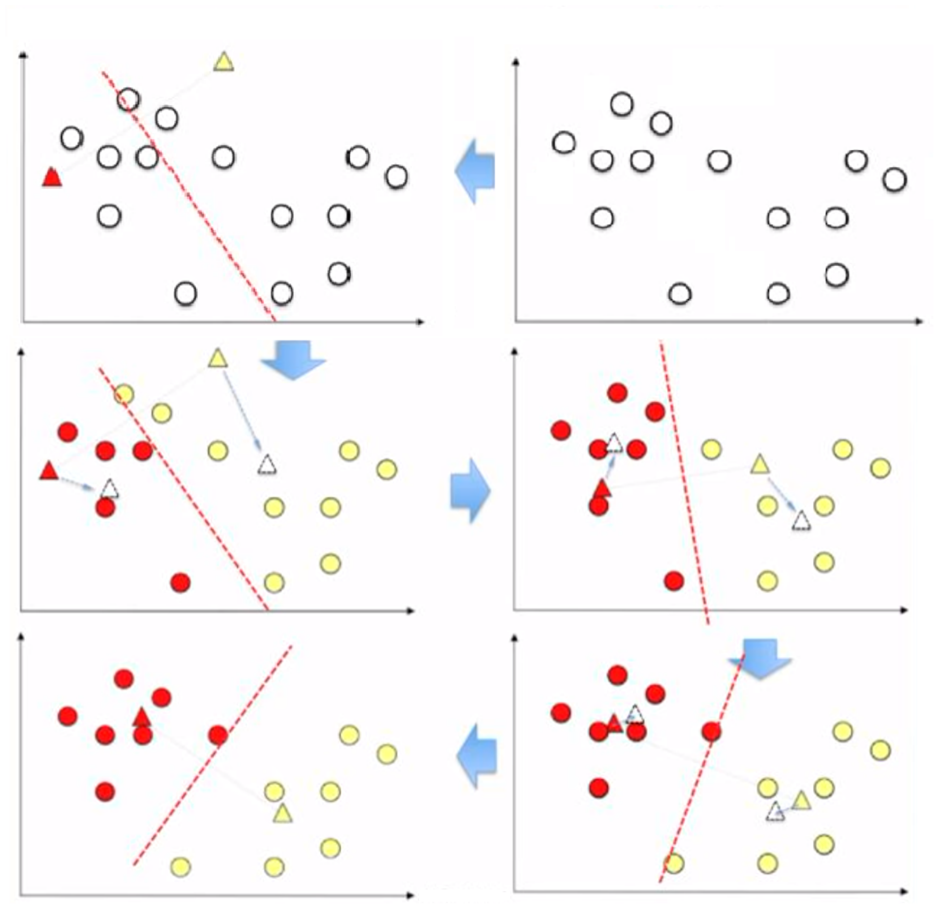
The k-Means Algorithm is a very widely and quite simple algorithm to cluster data in an unsupervised fashion. It needs the number of clusters (k) and a distance measurement (often euclidean distance) beforehand. Then it clusters the data-samples by repeating two steps until the clusters do not change any more.

After initializing k centroids randomly:

1. Every data-point is assigned its closest centroid.
2. Every centroid moves to the center of all points, assigned to it.
3. Repeating steps (1) and (2) until the algorithm converges and all the data-points are clustered.

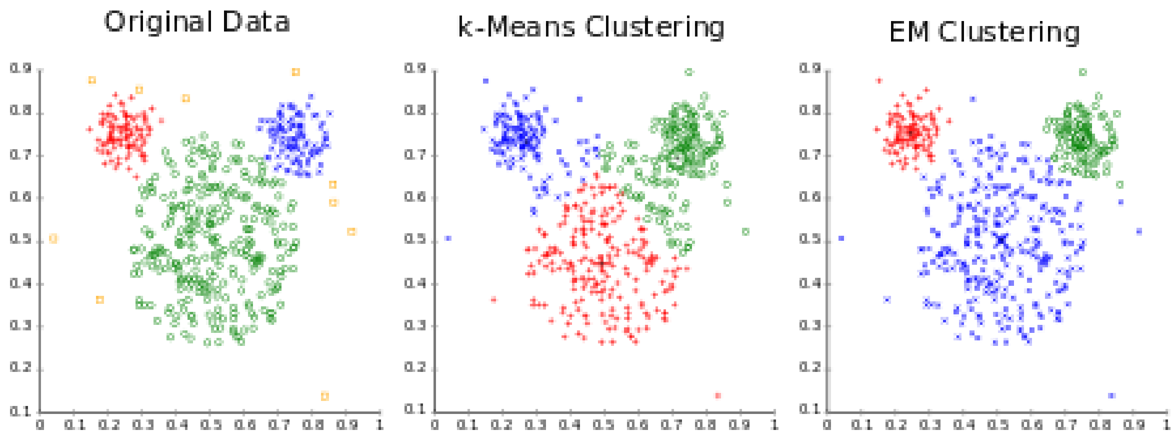
Figure 4.1 shows this for a simple 2-dimensional example. The algorithm can be used for more complex data, though. K-Means is an easy and effective way of clustering data without requiring any other knowledge about it. The results can be used for further analysis of the data.

The disadvantages of k-Means are that we assume, that distance equals similarity, which is not necessary in this case. It also does not care for internal structures in the data and assumes that clusters are always spherical. Unsupervised algorithms like Expectation-Maximization (EM) (Dempster, Laird, & Rubin, 1977) can handle cases like that much better. Figure 4.2 shows the limitation of k-Means when the data has internal structures. This structure can not be captures by simple calculating distances between data points. Another negative point is, that k has to be chosen beforehand. Self-Organizing-Maps (SOM) (Kohonen & Honkela, 2007) for example is an unsupervised algorithm where that is not necessary.



(Lavrenko, 2014)

Figure 4.1.: K-Means: Start with unlabeled data (top-right); Random initialization of 2 centroids (top-left); Every point gets the label of its closest centroid (middle-left); The centroids move to the center of all their points (middle-right); Repeat until convergence (bottom)



(Chire, 2010)

Figure 4.2.: K-Means does not assume any underlying structures. Expectation-Maximization can have a better prediction in that case.

4.3. Supervised learning

Unsupervised algorithms do not have the ability to propose any decision. Supervised algorithms (Caruana & Niculescu-Mizil, 2006) do. In order to do so, they need labeled data. That means, data-samples, that have already been labeled with the correct decision or prediction. Coming back to the example of predicting the rise or the fall of a share: In this case labeled data would contain the actual data (information about the company and the economical situation e.g.) and the actual outcome: did the share actually rise or fall in the situation, that is described by the input data.

Only when data, with already known solutions is available, we can train a system to learn the correlation between the input data and the output, the target.

4.3.1. k-Nearest-Neighbor

To start with an easy example of a supervised algorithm, the k-Nearest-Neighbor (Altman, 1992; Burba, Ferraty, & Vieu, 2009) algorithm will be explained. This algorithm does not actually learn any mapping from input to output, but it uses the same heuristic as k-means: Distance.

If there are a number of data-points, that already have their correct label, than one can do something very similar to what k-Means does. The k-Nearest-Neighbor algorithm calculates the distance between a new unlabeled data-point and every other already labeled data-point. It then looks at the k closest data-points and at their label. The new data-point simply gets the most common label under the k closest neighboring data-points.

This again is a very simple, but very effective way of assigning a label and thus a class to a new data-point. The disadvantages are almost the same as for k-Means: Distance as a measurement for similarity is assumed, structures in the data are not taken into account and k has to be decided on beforehand. Additionally, computation time can be quite high, if many, high dimensional data-points are available, since for every new data-point, the distances to all other points have to be calculated.

4.3.2. Neural Networks

Neural Networks (Schmidhuber, 2015; Haykin & Network, 2004) are probably the most famous algorithm in machine learning right now. Even though there are many more sophisticated models like Convolutional Neural Networks (LeCun et al., 1989) and Recurrent Neural Networks (Hochreiter & Schmidhuber, 1997), the principles of Neural Networks are always the same. Every Neural Network - and every supervised learning algorithm - ultimately is doing function approximation (Ferrari & Stengel, 2005). The thing, that makes Neural Networks so unique and popular is, that they are able to model very complex and high dimensional functions better than many other techniques.

Every network has n input-nodes and m output-nodes. The number of the input nodes is defined by the number of features ones data has. A feature can be pretty much anything, as long as it is a numerical value. In an intuitive way, one feature is one single information value from the data. So when we want to know if a company is going to increase or decrease their value in the next week, one would use information about the company and the economy in general as features. One feature could for example be the gross income of the company in the last month.

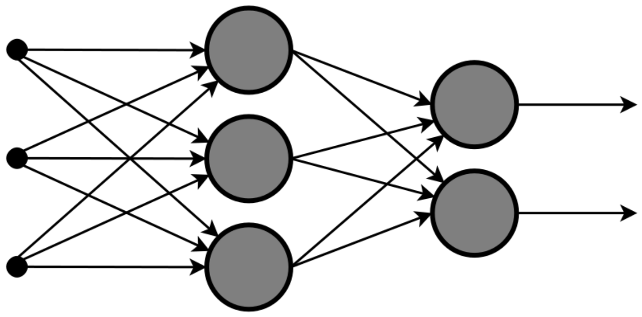
After selecting all features, they are stacked together into a feature vector. One feature vector is now one data-point or one input vector. The size of the feature vector determines the number of input neurons of the network.

The number of possible outcomes determines the number of output neurons. In the same example the network would have two output neurons, one for 'rise' and one for 'fall' of the share. The output of these two neurons can be interpreted as the probability of this action being the correct one. Figure 4.3 shows how a Neural Network looks schematically.

The question now is, how is the input propagated through the network to finally create the values for the output neurons? Figure 4.4 shows, what happens in every single node. The inputs, coming either from the direct input or from previous nodes, are multiplied by the weights on the connecting edges and summed up. The bias can be ignored for now. The resulting value is put into an activation function and passed on to the next node. If the node is an output node, it creates an output value.

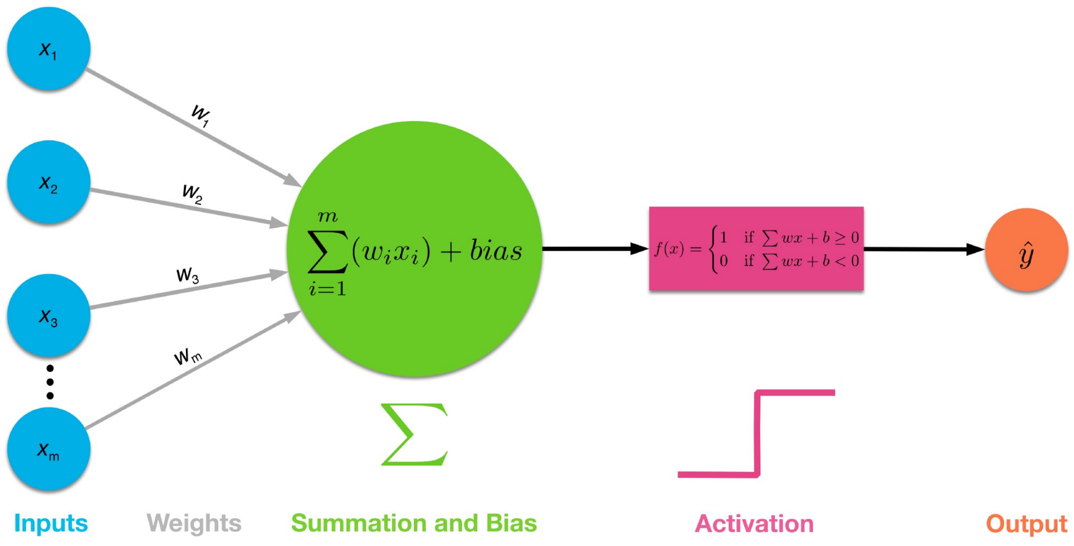
The weights are initialized randomly and are the only thing, that changes in the network during its training phase. The activation function for the last layer will map all values from all nodes to probabilities, simply by dividing each one of them by the sum of all values of output nodes. This activation function is called softmax and is normally used in any categorization task.

When training a Neural Network for one step, one will use one labeled data-point. The feature vector of this data-point is used as input and its label is transformed in a one-hot



(Chrislb, 2015)

Figure 4.3.: Neural Network with three input-neurons and two output-neurons.



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

(Kang, 2017)

Figure 4.4.: A perceptron: Incoming values are multiplied by the weights on the connecting edges and summed up. The result goes through an activation function and is passed on to the next node or is one output node.

encoding and used as target. The input-vector will be propagated through the network until the output nodes yield the result for the given input. Now one can calculate the difference between the actual observed output and the desired target output.

It is the goal of the network, to minimize this error. If this error between actual outcome and target is close to zero, it shows, that the network effectively maps the given input to the desired outcome.

To achieve this goal, different optimization techniques can be used (Pelikan, Goldberg, & Lobo, 2002). The most frequently used one is a form of gradient based optimization called gradient descent (Bottou, 2010). One formulates the error function, which is a mapping from all changeable parameters of the network (its weights) to the average error made for all data-points. Then the first derivative of this error function is calculated. Now the gradient can be calculated for this error value. The gradient will be used to change all the parameters in the direction of decreasing error. By repeatably doing this, the error will converge to an optimal or suboptimal state, depending on the problem.

Neural Networks can model very complex functions and they are relatively easy to build. Training can be time intensive, but after that the prediction for new data-samples is rather fast.

One of their big disadvantages is the fact, that it is hard to interpret what the network actually learns. Depending on the task it will learn underlying rules of the problem and use those to solve the problem. But to extract these rules to a level that it is easy for humans to understand is very hard to impossible. Models like Decision Trees do not suffer so much from this disadvantage.

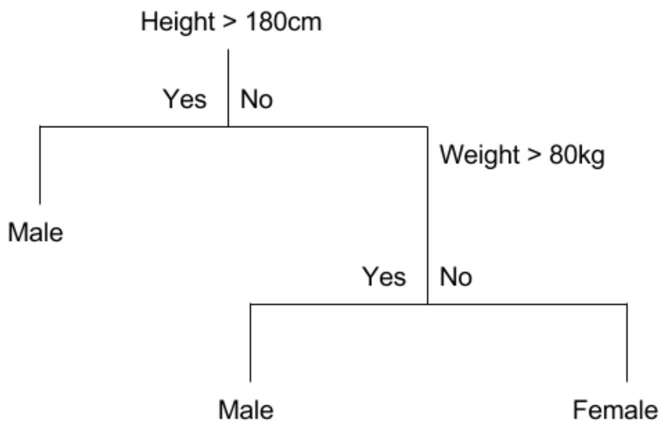
4.3.3. Decision Tree

Where Neural Networks work only with continuous values, Decision Trees (Quinlan, 1986) still use discrete boundaries to categorize data. There are various forms of Decision Trees and even Decision Forests (Ho, 1995, 1998), which use many Decision Trees at the same time.

In Figure 4.5 a very simple Decision Tree is shown. This simple example shows nicely how a Decision Tree works in principal. Just like Neural Networks, they need labeled training data in order to optimize themselves. The general idea is, to adapt the decision boundaries in an optimal way. That means, that for as many training samples as possible the correct outcome is predicted.

In the concrete example of Figure 4.5 one can assume, that if the training set contains a lot of women who are taller than 180cm, the decision boundary will change in favour of that. The model will also optimize where it has decision boundaries in the first place. Adding a weight decision boundary after the first 'Yes' might be a good idea in order to also categorize tall women better. Especially when the input-vector gets bigger it is an important question, between time and performance, how many boundaries a Decision Tree needs.

As mentioned before, Decision Trees are easier to interpret than Neural Networks. Especially when the problem is more discrete in its nature and not a purely mathematical problem, interpreting a Decision Tree is much nicer than a Neural Network. When the



(Brownlee, 2016)

Figure 4.5.: Very simple example of a Decision Tree. The sex of a person is to be determined by other information about the person.

problem becomes very complex and high-dimensional, Decision Trees might perform worse than more complex models, like Neural Networks.

4.3.4. Example: Compression Algorithms

In order to link the two chapters together, this section will describe, how a decision unit for the task of choosing an optimal compression algorithm for new, unseen data could look like. Two situations could be imagined.

The first is, that there are plenty of different datasets and none of them are tested with a compression benchmark yet. In this case unsupervised learning could be used, to cluster the different data-samples into groups of similar data-samples. K-Means for example could be used to cluster all the datasets in a fast and simple way. Then a certain number of data-samples could be run through a compression benchmark and the results could be analyzed. If many samples from one cluster are all compressed in a good or optimal fashion by the same algorithm, one can choose that algorithm for the rest of the data-samples from that cluster. This is a fast way to determine a compression algorithm for every data-sample, without having to test all of them.

The second case would be, if there is already a number of data-samples, tested with a benchmark and their results are stored. One can now take every data-sample that is already tested and use an optimal compression algorithm with regards to a metric. E.g. one could always take the algorithm with the best ratio as label for every data-sample. By doing this, a labeled dataset is created. This dataset can now be used for any supervised learning technique. A Neural Network or a Decision Tree with one dataset as input and 38 different compression algorithms as output could be created. In the training phase, the algorithm would be optimized, to map every dataset as close as possible to the previously selected optimal compression algorithm. After the training phase, new, untested data can be run through the trained model and a compression algorithm will be predicted. This will save time and should yield a good accuracy with regards to the optimal algorithm.

4.4. Conclusion

Machine learning can be of a great help when it comes to making decisions. Either in structuring unlabeled data or in proposing the optimal decision when trained with labeled data. DSS were originally built to assist humans in their decision making process, but are evolving to systems, that are trained to make better decisions than humans ever could. In all this, one should always try to understand why decisions are proposed or taken, even if that can be hard sometimes.

5. Final Conclusion

This paper covered two important topics. The compression benchmark of scientific datasets and the theoretical construction of DSS.

It was shown, how important semantical differences in datasets can be, when compressing them. Data is defined by its underlying information and the structure of how this information is stored. By acquiring and using this semantic information in a smart way, time an resources can be saved. Realizing that certain datasets will only compress very little while other can be reduced to a fraction of their original size can safe precious of time.

This benchmark study should show, that these differences between datasets exists and that they can be visualized by rigorous testing. More importantly it should have shown, that creating an intuition for datasets and the information they represent can often save the time and effort of analyzing ever single dataset.

After the data is stored in an efficient way, it can then be used to gather further informations and make decisions based on those insights. The usage of computer based models, automated visualizations, data clustering or even supervised models can greatly increase the performance of any company or research institute. DSS help when making decisions based on complex and unstructured data.

Deciding if data will be compressible well and if so which algorithm will achieve optimal solutions is of course also a decision problem. This is why all the Methods presented in the second chapter can be applied to this problem.

This trend of big data will further increase. The amount of data produced every day is rising exponentially. Systems for measurements are getting better every day. Social media creates millions of data-samples in the form of images, videos, tweets or comments every second. Dealing with this massive amount of unstructured and high-dimensional data will be a big task of computer science and data science. The goal: To store data as efficient as possible, to automatically acquire all the information the dataset contains and finally, to make the optimal decisions based on the give data.

References

Abel, D.-I. J. (2002). *The data compression resource on the internet.* Retrieved March 26, 2018, from www.data-compression.info

Acharya, T., & Tsai, P.-S. (2005). *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures.* Hoboken, New Jersey: John Wiley & Sons.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, *46*(3), 175-185. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879 doi: 10.1080/00031305.1992.10475879

Arnott, D., & Pervan, G. (2008). Eight key issues for the decision support systems discipline. *Decision Support Systems*, *44*(3), 657–672.

Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data* (pp. 25–71). Springer.

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... Zieba, K. (2016). End to end learning for self-driving cars. *CoRR*, *abs/1604.07316*. Retrieved from http://dblp.uni-trier.de/db/journals/corr/corr1604.html#BojarskiTDFFGJM16

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of compstat'2010* (pp. 177–186). Springer.

Brandenburg, K., & Stoll, G. (1994). ISO/MPEG-1 audio: A generic standard for coding of high-quality digital audio. *Journal of the Audio Engineering Society*, *42*(10), 780–792.

Brownlee, J. (2016). *Classification and regression trees for machine learning.* Retrieved March 26, 2018, from https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/

Burba, F., Ferraty, F., & Vieu, P. (2009). k-nearest neighbour method in functional nonparametric regression. *Journal of Nonparametric Statistics*, *21*(4), 453-469. Retrieved from https://doi.org/10.1080/10485250802668909 doi: 10.1080/10485250802668909

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on machine learning* (pp. 161–168).

Chen, M., & Fowler, M. L. (2014). The importance of data compression for energy efficiency in sensor networks. In *Hopkins university.*

Chire. (2010). *Different cluster analysis results on 'mouse' data set.* Retrieved March 26, 2018, from https://en.wikipedia.org/wiki/K-means_clustering

Chrislb. (2015). *Multi-layer neural network-english version.* Retrieved March 26, 2018, from https://commons.wikimedia.org/wiki/File:Single-Layer

_Neural_Network-Vector-Blank.svg

Cleary, J., & Witten, I. (1984). Data compression using adaptive coding and partial string matching. *IEEE transactions on Communications*, *32*(4), 396–402.

Collet, Y., & Turner, C. (2016). *Smaller and faster data compression with zstandard.* Retrieved March 26, 2018, from https://code.facebook.com/posts/1658392934479273/smaller-and-faster-data-compression-with-zstandard/

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

Deorowicz, S. (2003). *Silesia compression corpus.* Retrieved March 26, 2018, from http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia

Ferrari, S., & Stengel, R. F. (2005). Smooth function approximation using neural networks. *IEEE Transactions on Neural Networks*, *16*(1), 24–38.

Fredericks, X., Nagle, D. B., & Kranenburg, C. J. (2017). *Eaarl coastal topography cape hatteras, north carolina, pre- and post-hurricane isabel, 2003.* U.S. Geological Survey. doi: 10.5066/f76w9879

Goldman, A. J., Hartman, J., Fisher, J., & Sarel, S. (2004, November 16). *Method and tool for data mining in automatic decision making systems.* Google Patents. (US Patent 6,820,070)

Haagsma, I., & Johanns, R. (1970). Decision support systems: An integrated and distributed approach. *WIT Transactions on Ecology and the Environment*, *6*.

Hanke, M., Dinga, R., Häusler, C., Guntupalli, J., Casey, M., Kaule, F., & Stadler, J. (2015). High-resolution 7-tesla fmri data on the perception of musical genres – an extension to the studyforrest dataset [version 1; referees: 2 approved with reservations]. *F1000Research*, *4*(174). doi: 10.12688/f1000research.6679.1

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Unsupervised learning. In *The elements of statistical learning* (pp. 485–585). Springer.

Haykin, S., & Network, N. (2004). A comprehensive foundation. *Neural networks*, *2*(2004), 41.

Hipp, D. R. (2015). *Sqlite.* Retrieved March 26, 2018, from https://www.sqlite.org/download.html

Ho, T. K. (1995). Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on* (Vol. 1, pp. 278–282).

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, *20*(8), 832–844.

Hochreiter, S., & Schmidhuber, J. (1997, November). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780. Retrieved from http://dx.doi.org/10.1162/neco.1997.9.8.1735 doi: 10.1162/neco.1997.9.8.1735

Huang, W., Nakamori, Y., & Wang, S.-Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, *32*(10), 2513–2522.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, *40*(9), 1098–1101.

Kang, N. (2017). *Multi-layer neural networks with sigmoid function— deep learning for rookies.* Retrieved March 26, 2018, from https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Katz, P. (1991, September 24). *String searcher, and compressor using same.* Google Patents. Retrieved from https://www.google.com/patents/US5051745 (US Patent 5,051,745)

Kohonen, T., & Honkela, T. (2007). Kohonen network. *Scholarpedia*, *2*(1), 1568. (revision #122029) doi: 10.4249/scholarpedia.1568

Lavrenko, V. (2014). *K-means clustering: how it works.* Retrieved March 26, 2018, from http://bit.ly/K-means

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989, December). Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, *1*(4), 541–551. Retrieved from http://dx.doi.org/10.1162/neco.1989.1.4.541 doi: 10.1162/neco.1989.1.4.541

Lelewer, D. A., & Hirschberg, D. S. (1987, September). Data compression. *ACM Comput. Surv.*, *19*(3), 261–296. Retrieved from http://doi.acm.org/10.1145/45072.45074 doi: 10.1145/45072.45074

Li, T., Feng, S., & Li, L. X. (2001). Information visualization for intelligent decision support systems. *Knowledge-Based Systems*, *14*(5-6), 259–262.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, *28*(2), 129–137.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability, volume 1: Statistics* (pp. 281–297). Berkeley, Calif.: University of California Press. Retrieved from https://projecteuclid.org/euclid.bsmsp/1200512992

Mahoney, M. (2012). Data compression explained. *mattmahoney. net, updated May*, *7*.

Marakas, G. M. (2003). *Decision support systems in the 21st century* (Vol. 134). Prentice Hall Upper Saddle River, NJ.

Nevill-Manning, C. G., & Witten, I. H. (1999). Protein is incompressible. In *Data compression conference, 1999. proceedings. dcc'99* (pp. 257–266).

Pelikan, M., Goldberg, D. E., & Lobo, F. G. (2002). A survey of optimization by building and using probabilistic models. *Computational optimization and applications*, *21*(1), 5–20.

Portmann, F. T., Siebert, S., Bauer, C., & Döll, P. (2008). *Global dataset of monthly growing areas of 26 irrigated crops : version 1.0* (No. 6).

Power, D. J., Sharda, R., & Burstein, F. (2015). *Decision support systems.* Wiley Online Library.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Robinson, A., & Cherry, C. (1967). Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, *55*(3), 356–364.

Salomon, D., & Motta, G. (2010). *Handbook of data compression.* Springer Science & Business Media.

Sayood, K. (2000). *Introduction to data compression.* Morgan Kaufman Publishers.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Seeger, M., et al. (2001). *Learning with labeled and unlabeled data* (Tech. Rep.). technical report, University of Edinburgh.

Shim, J. P., Warkentin, M., Courtney, J. F., Power, D. J., Sharda, R., & Carlsson, C. (2002). Past, present, and future of decision support technology. *Decision support systems*, *33*(2), 111–126.

Sikora, T. (1997, Feb). The MPEG-4 video standard verification model. *IEEE Transactions on Circuits and Systems for Video Technology*, *7*(1), 19-31. doi: 10.1109/76.554415

SlidePlay. (2016). *Lempel-ziv compression techniques.* Retrieved March 26, 2018, from `http://slideplayer.com/slide/5173265/`

Storer, J. A. (1985). Textual substitution techniques for data compression. In A. Apostolico & Z. Galil (Eds.), *Combinatorial algorithms on words* (pp. 111–129). Berlin, Heidelberg: Springer Berlin Heidelberg.

Welch, T. A. (1984, June). A technique for high-performance data compression. *Computer*, *17*(6), 8-19. doi: 10.1109/MC.1984.1659158

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes: Compressing and indexing documents and images* (2nd ed.). San Francisco, CA: Morgan Kaufmann.

Wolpert, D. H., & Macready, W. G. (1997, April). No free lunch theorems for optimization. *Trans. Evol. Comp*, *1*(1), 67–82. Retrieved from `http://dx.doi.org/10.1109/4235.585893` doi: 10.1109/4235.585893

Zhang, X., Fuehres, H., & Gloor, P. A. (2011). Predicting stock market indicators through twitter "i hope it is not as bad as i fear". *Procedia-Social and Behavioral Sciences*, *26*, 55–62.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*, *23*(3), 337–343.

Ziv, J., & Lempel, A. (1978, September). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, *24*(5), 530-536. doi: 10.1109/TIT.1978.1055934

A. Appendix

Table A.1.: Results of one full run with lzBench: This table is plotted in Figure 3.2

No.	Name	Comp. speed	Decomp. speed	Ratio	Filename
0	lzlib 1.8 -9	1.23	56.68	7.86	CLOUDf08.bin
1	brotli 2017-03-10 -11	0.72	288.82	7.86	CLOUDf08.bin
2	lzlib 1.8 -0	28.21	52.01	7.67	CLOUDf08.bin
3	xz 5.2.3 -9	8.16	82.42	7.65	CLOUDf08.bin
4	lzma 16.04 -9	5.67	76.84	7.65	CLOUDf08.bin
5	lzma 16.04 -0	26.78	86.33	7.65	CLOUDf08.bin
6	xz 5.2.3 -6	7.99	84.07	7.65	CLOUDf08.bin
7	csc 2016-10-13 -5	9.83	58.56	7.64	CLOUDf08.bin
8	lzlib 1.8 -6	6.87	58.37	7.63	CLOUDf08.bin
9	lzma 16.04 -5	8.07	94.03	7.62	CLOUDf08.bin
10	xz 5.2.3 -0	25.28	83.71	7.62	CLOUDf08.bin
11	lzlib 1.8 -3	11.87	56.35	7.60	CLOUDf08.bin
12	csc 2016-10-13 -1	20.92	50.71	7.59	CLOUDf08.bin
13	lzma 16.04 -2	28.62	89.44	7.58	CLOUDf08.bin
14	csc 2016-10-13 -3	15.71	61.52	7.57	CLOUDf08.bin
15	xz 5.2.3 -3	16.55	83.33	7.56	CLOUDf08.bin
16	brotli 2017-03-10 -8	13.2	371.42	7.56	CLOUDf08.bin
17	brotli 2017-03-10 -5	27.96	289.68	7.55	CLOUDf08.bin
18	lzham 1.0 -d26 -0	7.39	327.84	7.54	CLOUDf08.bin
19	zstd 1.3.1 -22	2.2	1569.28	7.50	CLOUDf08.bin
20	zstd 1.3.1 -15	40.8	1492.43	7.50	CLOUDf08.bin
21	zstd 1.3.1 -11	76.53	1594.26	7.49	CLOUDf08.bin
22	lzma 16.04 -4	20.92	87.97	7.49	CLOUDf08.bin
23	zstd 1.3.1 -8	84.35	1566.48	7.49	CLOUDf08.bin
24	lzham 1.0 -d26 -1	3.81	313.51	7.47	CLOUDf08.bin
25	xpack 2016-06-02 -9	74.95	998.53	7.46	CLOUDf08.bin
26	zstd 1.3.1 -5	129.61	1549.42	7.46	CLOUDf08.bin
27	zstd 1.3.1 -18	23.82	1625.43	7.45	CLOUDf08.bin
28	zstd 1.3.1 -2	464.77	1535.61	7.45	CLOUDf08.bin
29	zstd 1.3.1 -1	622.5	1566.92	7.45	CLOUDf08.bin
30	libdeflate 0.7 -12	5.86	1047.23	7.43	CLOUDf08.bin
31	tornado 0.6a -16	4.43	228.65	7.41	CLOUDf08.bin

32	xpack 2016-06-02 -6	97.7	942.11	7.41	CLOUDf08.bin
33	libdeflate 0.7 -9	22.2	956.07	7.41	CLOUDf08.bin
34	zlib 1.2.11 -9	15.85	351.8	7.37	CLOUDf08.bin
35	brotli 2017-03-10 -2	165.87	273.31	7.35	CLOUDf08.bin
36	libdeflate 0.7 -6	115.02	944.34	7.35	CLOUDf08.bin
37	tornado 0.6a -13	20.55	221.39	7.35	CLOUDf08.bin
38	xpack 2016-06-02 -1	118.27	992.5	7.34	CLOUDf08.bin
39	tornado 0.6a -10	5.59	191.75	7.34	CLOUDf08.bin
40	zlib 1.2.11 -6	52.31	329.83	7.32	CLOUDf08.bin
41	tornado 0.6a -7	16.66	196.07	7.32	CLOUDf08.bin
42	lzfse 2017-03-08	57.05	438.09	7.31	CLOUDf08.bin
43	tornado 0.6a -5	56.58	177.78	7.29	CLOUDf08.bin
44	tornado 0.6a -6	35.83	181.82	7.29	CLOUDf08.bin
45	zling 2016-01-10 -3	58.85	240.19	7.28	CLOUDf08.bin
46	zling 2016-01-10 -4	56.75	246.7	7.28	CLOUDf08.bin
47	zling 2016-01-10 -2	62.23	248.78	7.27	CLOUDf08.bin
48	libdeflate 0.7 -3	140.65	982.2	7.27	CLOUDf08.bin
49	libdeflate 0.7 -1	146.88	870.22	7.26	CLOUDf08.bin
50	zling 2016-01-10 -1	52.11	237.27	7.26	CLOUDf08.bin
51	zling 2016-01-10 -0	56.85	239.18	7.25	CLOUDf08.bin
52	tornado 0.6a -4	166.16	248.54	7.22	CLOUDf08.bin
53	tornado 0.6a -3	253	253.95	7.21	CLOUDf08.bin
54	zlib 1.2.11 -1	90.34	470.36	7.07	CLOUDf08.bin
55	lizard 1.0 -29	0.22	4551.26	6.78	CLOUDf08.bin
56	lizard 1.0 -19	0.49	4577.43	6.73	CLOUDf08.bin
57	lizard 1.0 -39	0.48	3860.41	6.73	CLOUDf08.bin
58	lizard 1.0 -49	0.21	2793.22	6.72	CLOUDf08.bin
59	lz4hc 1.7.5 -12	0.03	2995.85	6.70	CLOUDf08.bin
60	lz4hc 1.7.5 -9	15.92	3338.65	6.65	CLOUDf08.bin
61	lizard 1.0 -45	52.71	2445.4	6.64	CLOUDf08.bin
62	lizard 1.0 -25	53.97	2307.52	6.63	CLOUDf08.bin
63	lizard 1.0 -15	53.34	1455.77	6.63	CLOUDf08.bin
64	lz4hc 1.7.5 -4	104.48	3157.39	6.62	CLOUDf08.bin
65	lizard 1.0 -35	102.11	1585.19	6.61	CLOUDf08.bin
66	lzo1y 2.09 -999	3.15	944.97	6.59	CLOUDf08.bin
67	yalz77 2015-09-19 -12	17.26	614.85	6.58	CLOUDf08.bin
68	lzo1b 2.09 -999	3.08	676.82	6.58	CLOUDf08.bin
69	yalz77 2015-09-19 -8	31.11	624.39	6.58	CLOUDf08.bin
70	lzo1z 2.09 -999	3.53	967.14	6.57	CLOUDf08.bin
71	lzo1x 2.09 -999	3.43	972.77	6.57	CLOUDf08.bin
72	lz4hc 1.7.5 -1	142.82	2913.56	6.57	CLOUDf08.bin
73	lzo1c 2.09 -999	3.14	757.78	6.57	CLOUDf08.bin
74	lzo1f 2.09 -999	2.88	686.15	6.56	CLOUDf08.bin
75	yalz77 2015-09-19 -4	43.65	568.68	6.56	CLOUDf08.bin

76	lizard 1.0 -22	280.6	3996.42	6.54	CLOUDf08.bin
77	lizard 1.0 -42	235.78	4001.14	6.54	CLOUDf08.bin
78	shrinker 0.1	978.08	1189.52	6.53	CLOUDf08.bin
79	lzo1c 2.09 -99	92.94	739.61	6.51	CLOUDf08.bin
80	pithy 2011-12-24 -9	1233.34	3542.66	6.51	CLOUDf08.bin
81	lzo1b 2.09 -99	93.43	740.43	6.51	CLOUDf08.bin
82	yalz77 2015-09-19 -1	94.45	475.09	6.51	CLOUDf08.bin
83	pithy 2011-12-24 -6	1305.52	3315.2	6.51	CLOUDf08.bin
84	pithy 2011-12-24 -3	1844.14	3483.27	6.49	CLOUDf08.bin
85	lz4 1.7.5	1300.46	2950.81	6.47	CLOUDf08.bin
86	pithy 2011-12-24 -0	1982.51	3630.78	6.46	CLOUDf08.bin
87	lzo1b 2.09 -9	173.16	590.87	6.46	CLOUDf08.bin
88	lizard 1.0 -12	187.58	3093.69	6.45	CLOUDf08.bin
89	lzo1c 2.09 -9	160.82	534.91	6.45	CLOUDf08.bin
90	lzo1b 2.09 -6	245.2	1498.06	6.44	CLOUDf08.bin
91	lzo1b 2.09 -3	248.15	1251.94	6.44	CLOUDf08.bin
92	lzo1c 2.09 -6	284.69	1375.07	6.44	CLOUDf08.bin
93	lzo1c 2.09 -3	279.87	1339.57	6.43	CLOUDf08.bin
94	lizard 1.0 -32	164.08	3575.91	6.43	CLOUDf08.bin
95	lzo1f 2.09 -1	282.66	1844.08	6.42	CLOUDf08.bin
96	lzo1c 2.09 -1	281.7	2342.11	6.42	CLOUDf08.bin
97	lz4fast 1.7.5 -3	1320.86	3334.5	6.42	CLOUDf08.bin
98	lzo1b 2.09 -1	239.92	2376.15	6.41	CLOUDf08.bin
99	ucl_nrv2b 1.03 -9	6	409.16	6.38	CLOUDf08.bin
100	ucl_nrv2e 1.03 -9	5.9	435.95	6.37	CLOUDf08.bin
101	ucl_nrv2d 1.03 -9	6.01	443.9	6.37	CLOUDf08.bin
102	blosclz 2015-11-10 -9	556.74	1085.87	6.37	CLOUDf08.bin
103	fastlz 0.1 -2	439.01	1753.47	6.36	CLOUDf08.bin
104	slz_zlib 1.0.0 -1	454.04	475.85	6.35	CLOUDf08.bin
105	slz_zlib 1.0.0 -2	413.92	476.32	6.35	CLOUDf08.bin
106	slz_zlib 1.0.0 -3	461.42	451.4	6.35	CLOUDf08.bin
107	lzo1y 2.09 -1	1592.93	892.5	6.34	CLOUDf08.bin
108	ucl_nrv2b 1.03 -6	33.43	391.42	6.34	CLOUDf08.bin
109	lzo1x 2.09 -1	1639.28	908.71	6.33	CLOUDf08.bin
110	lzo1x 2.09 -15	1479.33	858.54	6.33	CLOUDf08.bin
111	lzo1x 2.09 -12	1361.29	831.88	6.33	CLOUDf08.bin
112	lzo1x 2.09 -11	1405.92	821.76	6.33	CLOUDf08.bin
113	ucl_nrv2d 1.03 -6	35.35	430.37	6.31	CLOUDf08.bin
114	ucl_nrv2e 1.03 -6	33.08	421.98	6.31	CLOUDf08.bin
115	lzo1a 2.09 -99	122.57	565	6.28	CLOUDf08.bin
116	lzo1 2.09 -99	88.67	408.79	6.28	CLOUDf08.bin
117	brotli 2017-03-10 -0	554.79	296.91	6.27	CLOUDf08.bin
118	ucl_nrv2b 1.03 -1	44.63	379.55	6.27	CLOUDf08.bin
119	lzvn 2017-03-08	77.34	649.35	6.26	CLOUDf08.bin

120	ucl_nrv2d 1.03 -1	43.1	418.23	6.26	CLOUDf08.bin
121	ucl_nrv2e 1.03 -1	43.35	413.27	6.26	CLOUDf08.bin
122	lzo2a 2.09 -999	3.35	537.22	6.25	CLOUDf08.bin
123	lzo1a 2.09 -1	248.73	1555.44	6.25	CLOUDf08.bin
124	lzo1 2.09 -1	233.31	1340.64	6.25	CLOUDf08.bin
125	lz4fast 1.7.5 -17	2328.88	2961.52	6.22	CLOUDf08.bin
126	lzmat 1.01	26.57	474.59	6.21	CLOUDf08.bin
127	lzf 3.6 -1	264.44	545.69	6.19	CLOUDf08.bin
128	lzf 3.6 -0	345.37	529.25	6.19	CLOUDf08.bin
129	fastlz 0.1 -1	382.46	1615.6	6.19	CLOUDf08.bin
130	lzg 1.0.8 -8	36.05	481.4	6.18	CLOUDf08.bin
131	lzg 1.0.8 -6	54.16	406.89	6.15	CLOUDf08.bin
132	lzg 1.0.8 -4	57.4	468.22	6.15	CLOUDf08.bin
133	lzg 1.0.8 -1	52.2	501.36	6.14	CLOUDf08.bin
134	crush 1.0 -2	11.01	409.37	6.12	CLOUDf08.bin
135	crush 1.0 -1	30.5	378.36	6.11	CLOUDf08.bin
136	wflz 2015-09-16	486.82	1535.09	6.08	CLOUDf08.bin
137	brieflz 1.1.0	106.6	312.96	6.05	CLOUDf08.bin
138	crush 1.0 -0	42.88	342.36	6.04	CLOUDf08.bin
139	tornado 0.6a -2	421.51	665.59	6.01	CLOUDf08.bin
140	blosclz 2015-11-10 -6	569.43	1523.19	5.90	CLOUDf08.bin
141	quicklz 1.5.0 -2	202.46	657.28	5.84	CLOUDf08.bin
142	quicklz 1.5.0 -1	590.15	820.59	5.76	CLOUDf08.bin
143	quicklz 1.5.0 -3	60.25	1233.96	5.70	CLOUDf08.bin
144	density 0.12.5 beta -3	310.62	210.26	5.63	CLOUDf08.bin
145	lzsse8 2016-05-14 -12	18.5	3356.71	5.55	CLOUDf08.bin
146	lzsse8 2016-05-14 -16	18.2	3202.31	5.55	CLOUDf08.bin
147	lzsse8 2016-05-14 -6	17.52	3299.13	5.52	CLOUDf08.bin
148	tornado 0.6a -1	607.72	665.96	5.36	CLOUDf08.bin
149	lzsse4 2016-05-14 -16	21.09	3044.37	5.31	CLOUDf08.bin
150	lzsse4 2016-05-14 -12	20.28	3105.63	5.31	CLOUDf08.bin
151	lzsse4 2016-05-14 -6	18.21	2998.13	5.29	CLOUDf08.bin
152	gipfeli 2016-07-13	491.8	907.54	5.28	CLOUDf08.bin
153	snappy 1.1.4	1728.1	1869.66	5.27	CLOUDf08.bin
154	lzjb 2010	322.33	339.16	5.27	CLOUDf08.bin
155	lzsse8 2016-05-14 -1	28.59	2945.76	5.21	CLOUDf08.bin
156	lzsse4 2016-05-14 -1	33.04	2686	5.07	CLOUDf08.bin
157	yappy 2014-03-22 -100	36.17	3370.26	4.99	CLOUDf08.bin
158	lzsse2 2016-05-14 -12	22.56	2249.47	4.96	CLOUDf08.bin
159	lzsse2 2016-05-14 -16	21.1	2104.69	4.96	CLOUDf08.bin
160	lzsse2 2016-05-14 -6	20.67	2245.36	4.94	CLOUDf08.bin
161	density 0.12.5 beta -2	732.78	552.4	4.77	CLOUDf08.bin
162	lzsse2 2016-05-14 -1	33.43	2103.93	4.68	CLOUDf08.bin
163	lzrw 15-Jul-1991 -5	85.05	461.25	3.86	CLOUDf08.bin

164	lzrw 15-Jul-1991 -4	480.92	553.15	3.83	CLOUDf08.bin
165	lzrw 15-Jul-1991 -3	465.74	709.06	3.83	CLOUDf08.bin
166	yappy 2014-03-22 -10	67.01	3575.89	3.71	CLOUDf08.bin
167	lzrw 15-Jul-1991 -1	377.61	671.38	3.64	CLOUDf08.bin
168	blosclz 2015-11-10 -3	612.34	1370.12	3.07	CLOUDf08.bin
169	blosclz 2015-11-10 -1	1037.62	1536.04	2.31	CLOUDf08.bin
170	lizard 1.0 -40	157.79	3070.95	2.15	CLOUDf08.bin
171	lizard 1.0 -20	447.72	3552.96	2.15	CLOUDf08.bin
172	yappy 2014-03-22 -1	106.87	1895.22	2.05	CLOUDf08.bin
173	lizard 1.0 -10	497.55	2889.2	2.04	CLOUDf08.bin
174	lizard 1.0 -30	465.87	3467.25	2.04	CLOUDf08.bin
175	density 0.12.5 beta -1	819.16	729.42	1.67	CLOUDf08.bin
176	memcpy	4537.5	4271.75	1.00	CLOUDf08.bin

Table A.2.: Results from Geographical Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
irrigated_crop_01_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -2	lzlib 1.8 -3	xz 5.2.3 -1
irrigated_crop_02_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4
irrigated_crop_03_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_04_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	xz 5.2.3 -1
irrigated_crop_05_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_06_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_07_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_08_irrigated_12	lzlib 1.8 -2	lzlib 1.8 -1	lzlib 1.8 -3	lzlib 1.8 -4
irrigated_crop_09_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -4	xz 5.2.3 -1
irrigated_crop_10_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4
irrigated_crop_11_irrigated_12	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7	brotli 2017-03-10 -8
irrigated_crop_12_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_13_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4
irrigated_crop_14_irrigated_12	zstd 1.3.1 -1	zstd 1.3.1 -2	zstd 1.3.1 -3	zstd 1.3.1 -4
irrigated_crop_15_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -6	xz 5.2.3 -1	xz 5.2.3 -2
irrigated_crop_16_irrigated_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4
irrigated_crop_17_irrigated_12	lzlib 1.8 -1	xz 5.2.3 -0	xz 5.2.3 -1	xz 5.2.3 -2
irrigated_crop_18_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_19_irrigated_12	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7
irrigated_crop_20_irrigated_12	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -9
irrigated_crop_21_irrigated_12	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4	xz 5.2.3 -0
irrigated_crop_22_irrigated_12	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7
irrigated_crop_23_irrigated_12	brotli 2017-03-10 -4	brotli 2017-03-10 -5	brotli 2017-03-10 -6	brotli 2017-03-10 -7
irrigated_crop_24_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_25_irrigated_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
irrigated_crop_26_irrigated_12	lzlib 1.8 -6	xz 5.2.3 -1	xz 5.2.3 -2	xz 5.2.3 -3
rainfed_crop_01_rainfed_12	lzlib 1.8 -4	lzlib 1.8 -0	lzlib 1.8 -3	lzlib 1.8 -6
rainfed_crop_02_rainfed_12	xz 5.2.3 -1	lzma 16.04 -0	lzma 16.04 -1	lzma 16.04 -2
rainfed_crop_03_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -6	xz 5.2.3 -0	xz 5.2.3 -1
rainfed_crop_04_rainfed_12	lzlib 1.8 -6	xz 5.2.3 -7	xz 5.2.3 -8	lzlib 1.8 -0
rainfed_crop_05_rainfed_12	lzlib 1.8 -6	xz 5.2.3 -5	xz 5.2.3 -6	xz 5.2.3 -8
rainfed_crop_06_rainfed_12	xz 5.2.3 -6	xz 5.2.3 -8	lzlib 1.8 -0	lzlib 1.8 -1
rainfed_crop_07_rainfed_12	lzlib 1.8 -6	lzlib 1.8 -1	lzlib 1.8 -3	xz 5.2.3 -1
rainfed_crop_08_rainfed_12	xz 5.2.3 -1	xz 5.2.3 -6	xz 5.2.3 -7	lzma 16.04 -0
rainfed_crop_09_rainfed_12	xz 5.2.3 -7	xz 5.2.3 -8	lzlib 1.8 -1	lzlib 1.8 -2
rainfed_crop_10_rainfed_12	xz 5.2.3 -1	xz 5.2.3 -2	xz 5.2.3 -3	lzma 16.04 -0
rainfed_crop_11_rainfed_12	lzlib 1.8 -5	lzlib 1.8 -6	xz 5.2.3 -4	xz 5.2.3 -6
rainfed_crop_12_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_13_rainfed_12	lzlib 1.8 -4	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2
rainfed_crop_14_rainfed_12	lzlib 1.8 -3	lzlib 1.8 -4	lzlib 1.8 -5	lzlib 1.8 -6
rainfed_crop_15_rainfed_12	lzlib 1.8 -6	xz 5.2.3 -7	xz 5.2.3 -6	xz 5.2.3 -8
rainfed_crop_16_rainfed_12	xz 5.2.3 -6	xz 5.2.3 -8	lzlib 1.8 -1	lzlib 1.8 -6
rainfed_crop_17_rainfed_12	xz 5.2.3 -6	lzlib 1.8 -1	xz 5.2.3 -1	xz 5.2.3 -3
rainfed_crop_18_rainfed_12	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3	lzlib 1.8 -4
rainfed_crop_19_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_20_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_21_rainfed_12	lzlib 1.8 -6	xz 5.2.3 -1	xz 5.2.3 -2	xz 5.2.3 -5
rainfed_crop_22_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_23_rainfed_12	lzlib 1.8 -3	lzlib 1.8 -4	lzlib 1.8 -5	xz 5.2.3 -4
rainfed_crop_24_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_25_rainfed_12	lzlib 1.8 -0	lzlib 1.8 -1	lzlib 1.8 -2	lzlib 1.8 -3
rainfed_crop_26_rainfed_12	lzlib 1.8 -3	lzlib 1.8 -0	lzlib 1.8 -2	xz 5.2.3 -1

Table A.3.: Results from Electrical Engineering Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Electric_D067966	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Electric_D070707	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Electric_D070835	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6
Electric_D070436	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Electric_D059546	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Electric_D061283	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Electric_D070802	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9
Electric_D070437	lzlib 1.8 -6	lzlib 1.8 -9	lzma 16.04 -5	xz 5.2.3 -6

Table A.4.: Results from FMRI Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
FMRI_task-01	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
FMRI_task-03	csc 2016-10-13 -1	lzlib 1.8 -0	brotli 2017-03-10 -5	zstd 1.3.1 -8
FMRI_task-04	csc 2016-10-13 -1	lzlib 1.8 -0	brotli 2017-03-10 -5	zstd 1.3.1 -8
FMRI_task-05	csc 2016-10-13 -5	csc 2016-10-13 -3	lzlib 1.8 -9	xz 5.2.3 -9
FMRI_task-06	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
FMRI_task-07	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5
FMRI_task-08	csc 2016-10-13 -1	lzlib 1.8 -0	lzma 16.04 -0	brotli 2017-03-10 -5

Table A.5.: Results from Climate Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Isabell_Wf01	lzma 16.04 -9	xz 5.2.3 -9	lzham 1.0 -d26 -1	lzlib 1.8 -9
Isabell_Uf01	lzlib 1.8 -3	lzlib 1.8 -6	xz 5.2.3 -6	lzlib 1.8 -9
Isabell_QSNOWf01	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6
Isabell_QCLOUDf01	lzlib 1.8 -9	brotli 2017-03-10 -11	lzma 16.04 -9	xz 5.2.3 -6
Isabell_QRAINf01	lzlib 1.8 -9	brotli 2017-03-10 -11	lzma 16.04 -9	xz 5.2.3 -6
Isabell_Pf01	lzlib 1.8 -3	lzlib 1.8 -6	lzlib 1.8 -9	xz 5.2.3 -6
Isabell_QGRAUPf01	lzlib 1.8 -9	xz 5.2.3 -6	xz 5.2.3 -9	lzma 16.04 -9
Isabell_Vf01	lzlib 1.8 -3	lzlib 1.8 -6	xz 5.2.3 -6	lzlib 1.8 -9
Isabell_CLOUDf01	lzma 16.04 -5	lzlib 1.8 -9	lzlib 1.8 -3	lzlib 1.8 -6
Isabell_QVAPORf01	csc 2016-10-13 -5	csc 2016-10-13 -3	csc 2016-10-13 -1	lzlib 1.8 -3
Isabell_QICEf01	lzlib 1.8 -9	xz 5.2.3 -6	lzlib 1.8 -3	lzma 16.04 -5
Isabell_PRECIPf01	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	lzma 16.04 -5
Isabell_TCf01	csc 2016-10-13 -3	csc 2016-10-13 -5	csc 2016-10-13 -1	lzlib 1.8 -3

Table A.6.: Results from Lukas Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Lukas_lukas_2d_16_breast_0	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Lukas_lukas_2d_16_thorax_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_pelvis_0	csc 2016-10-13 -3	csc 2016-10-13 -5	csc 2016-10-13 -1	lzlib 1.8 -9
Lukas_lukas_2d_16_thorax_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9
Lukas_lukas_2d_16_breast_1	lzlib 1.8 -6	lzlib 1.8 -9	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_pelvis_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_sinus_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_spine_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9
Lukas_lukas_2d_16_head_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	lzma 16.04 -5
Lukas_lukas_2d_16_food_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_knee_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_leg_0	lzlib 1.8 -9	xz 5.2.3 -6	xz 5.2.3 -9	lzlib 1.8 -6
Lukas_lukas_2d_16_hand_0	lzlib 1.8 -6	lzlib 1.8 -9	lzma 16.04 -5	lzma 16.04 -9
Lukas_lukas_2d_16_head_1	lzlib 1.8 -9	lzlib 1.8 -6	xz 5.2.3 -6	xz 5.2.3 -9
Lukas_lukas_2d_16_spine_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6
Lukas_lukas_2d_16_sinus_1	lzlib 1.8 -9	lzma 16.04 -9	xz 5.2.3 -6	xz 5.2.3 -9
Lukas_lukas_2d_16_hand_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_knee_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_leg_1	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -9	xz 5.2.3 -6
Lukas_lukas_2d_16_food_0	lzlib 1.8 -9	lzlib 1.8 -6	lzma 16.04 -5	lzma 16.04 -9

Table A.7.: Results from Protein Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Protein_mj	brotli 2017-03-10 -11	zstd 1.3.1 -18	zstd 1.3.1 -22	zstd 1.3.1 -1
Protein_sc	brotli 2017-03-10 -11	csc 2016-10-13 -5	zstd 1.3.1 -22	csc 2016-10-13 -3
Protein_hi	brotli 2017-03-10 -11	zstd 1.3.1 -18	zstd 1.3.1 -22	zstd 1.3.1 -1
Protein_hs	brotli 2017-03-10 -11	zstd 1.3.1 -22	csc 2016-10-13 -5	csc 2016-10-13 -3

Table A.8.: Results from Stars Data

Data sample	best ratio	second best ratio	third best ratio	fourth best ratio
Stars_SAOra	xz 5.2.3 -6	xz 5.2.3 -9	lzma 16.04 -5	lzlib 1.8 -6
Stars_SAO	lzma 16.04 -9	lzlib 1.8 -6	lzma 16.04 -5	xz 5.2.3 -6

List of Figures

List of Tables