A Machine Learning Perspective on Predictive Coding with PAQ8 and New Applications

by

Byron Knoll

B. Sc. Computer Science, The University of British Columbia, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

June 2011

Abstract

The goal of this thesis is to describe a state-of-the-art compression method called PAQ8 from the perspective of machine learning. We show both how PAQ8 makes use of several simple, well known machine learning models and algorithms, and how it can be improved by exchanging these components for more sophisticated models and algorithms. We also present a broad range of new applications of PAQ8 to machine learning tasks including language modeling and adaptive text prediction, adaptive game playing, classification, and lossy compression using features from the field of deep learning.

Table of Contents

List of Tables

List of Figures

Glossary

AMDL Approximate Minimum Description Length

BCN Best-Compression Neighbor

EKF Extended Kalman Filter

PPAM Prediction by Partial Approximate Matching

PPM Prediction by Partial Matching

RNN Recurrent Neural Network

SMDL Standard Minimum Description Length

Acknowledgments

This thesis would not have been possible without the help and support of many people. Most of all, I would like to thank my supervisor Nando de Freitas for his invaluable guidance and giving me the freedom to explore my own ideas. I would also like to thank fellow UBC CS students for discussing ideas, answering questions, and providing feedback on my research: David Buchman, John Chia, Aram Ebtekar, Matt Hoffman, Andrej Karpathy, Ben Marlin, Kevin Swersky, and Paul Vanetti. I would also like to thank Matt Mahoney for answering questions and discussing new applications related to PAQ8. Matt Mahoney played a crucial role in the development of the PAQ8 algorithm, which is a major component of this thesis. Finally, I would like to thank my family for their tireless love and support.

Chapter 1

Introduction

Detecting temporal patterns and predicting into the future is a fundamental problem in machine learning. It has gained great interest recently in the areas of non-parametric Bayesian statistics [39] and deep learning [35], with applications to several domains including language modeling and unsupervised learning of audio and video sequences. Some researchers have argued that sequence prediction is key to understanding human intelligence [14].

The close connections between sequence prediction and data compression are perhaps underappreciated within the machine learning community. The goal of this thesis is to describe a state-of-the-art compression method called *PAQ8* [23] from the perspective of machine learning. We show both how PAQ8 makes use of several simple, well known machine learning models and algorithms, and how it can be improved by exchanging these components for more sophisticated models and algorithms.

PAQ is a family of open-source compression algorithms closely related to the better known Prediction by Partial Matching (PPM) algorithm [9]. PPM-based data compression methods dominated many of the compression benchmarks (in terms of compression ratio) in the 1990s, but have since been eclipsed by PAQ-based methods. Compression algorithms typically need to make a trade-off between compression ratio, speed, and memory usage. PAQ8 is a version of PAQ which achieves record breaking compression ratios at the expense of increased time and memory usage. For example, all of the winning submissions in the Hutter Prize

Table 1.1: Comparison of cross entropy rates of several compression algorithms on the Calgary corpus files. The cross entropy rate metric is defined in Section 2.3.

FILE	PPM-TEST	PPM*C	1PF	UKN	CPPMII-64	PAQ8L
BIB	1.80	1.91	1.73	1.72	1.68	1.50
BOOK1	2.21	2.40	2.17	2.20	2.14	2.01
BOOK2	1.88	2.02	1.83	1.84	1.78	1.60
GEO	4.65	4.83	4.40	4.40	4.16	3.43
NEWS	2.28	2.42	2.20	2.20	2.14	1.91
OBJ1	3.87	4.00	3.64	3.65	3.50	2.77
OBJ2	2.37	2.43	2.21	2.19	2.11	1.45
PAPER1	2.26	2.37	2.21	2.20	2.14	1.97
PAPER2	2.22	2.36	2.18	2.18	2.12	1.99
PIC	0.82	0.85	0.77	0.82	0.70	0.35
PROGC	2.30	2.40	2.23	2.21	2.16	1.92
PROGL	1.57	1.67	1.44	1.43	1.39	1.18
PROGP	1.61	1.62	1.44	1.42	1.39	1.15
TRANS	1.35	1.45	1.21	1.20	1.17	0.99
Average	**2.23**	**2.34**	**2.12**	**2.12**	**2.04**	**1.73**

[17], a contest to losslessly compress the first 100 MB (10^8 bytes) of Wikipedia, have been specialized versions of PAQ8. Indeed, there are dozens of variations on the basic PAQ8 method[1]. As stated on the Hutter Prize website, "This compression contest is motivated by the fact that being able to compress well is closely related to acting intelligently, thus reducing the slippery concept of intelligence to hard file size numbers."

The stochastic sequence memoizer [13] is a language modeling technique recently developed in the field of Bayesian nonparametrics. Table 1.1 shows a comparison of several compression algorithms on the Calgary corpus [3], a widely-used compression benchmark. A summary of the Calgary corpus files are in Table 1.2. `PPM-test` is our own PPM implementation used for testing different compression techniques. `PPM*C` is a PPM implementation that was state of the art in 1995 [10]. `1PF` and `UKN` are implementations of the stochastic sequence memoizer [13]. `cPPMII-64` [33] is currently among the best PPM implementations. `paq8l` out-

[1] http://cs.fit.edu/~mmahoney/compression/paq.html

Table 1.2: File size and description of Calgary corpus files.

FILE	BYTES	DESCRIPTION
BIB	111,261	ASCII TEXT IN UNIX "REFER" FORMAT - 725 BIBLIOGRAPHIC REFERENCES.
BOOK1	768,771	UNFORMATTED ASCII TEXT - "FAR FROM THE MADDING CROWD"
BOOK2	610,856	ASCII TEXT IN UNIX "TROFF" FORMAT - "PRINCIPLES OF COMPUTER SPEECH"
GEO	102,400	32 BIT NUMBERS IN IBM FLOATING POINT FORMAT - SEISMIC DATA.
NEWS	377,109	ASCII TEXT - USENET BATCH FILE ON A VARIETY OF TOPICS.
OBJ1	21,504	VAX EXECUTABLE PROGRAM - COMPILATION OF PROGP.
OBJ2	246,814	MACINTOSH EXECUTABLE PROGRAM - "KNOWLEDGE SUPPORT SYSTEM".
PAPER1	53,161	"TROFF" FORMAT - ARITHMETIC CODING FOR DATA COMPRESSION.
PAPER2	82,199	"TROFF" FORMAT - COMPUTER (IN)SECURITY.
PIC	513,216	1728 X 2376 BITMAP IMAGE (MSB FIRST).
PROGC	39,611	SOURCE CODE IN C - UNIX COMPRESS V4.0.
PROGL	71,646	SOURCE CODE IN LISP - SYSTEM SOFTWARE.
PROGP	49,379	SOURCE CODE IN PASCAL - PROGRAM TO EVALUATE PPM COMPRESSION.
TRANS	93,695	ASCII AND CONTROL CHARACTERS - TRANSCRIPT OF A TERMINAL SESSION.

performs all of these compression algorithms by what is considered to be be a very large margin in this benchmark.

Despite the huge success of PAQ8, it is rarely mentioned or compared against in machine learning papers. There are reasons for this. A core difficulty is the lack of scientific publications on the inner-workings of PAQ8. To the best of our knowledge, there exist only incomplete high-level descriptions of PAQ1 through 6 [23] and PAQ8 [24]. The C++ source code, although available, is very close to machine language (due to optimizations) and the underlying algorithms are difficult to extract. Many of the architectural details of PAQ8 in this thesis were understood by examining the source code and are presented here for the first time.

1.1 Contributions of this Thesis

In this thesis, we provide a detailed explanation of how PAQ8 works. We believe this contribution will be of great value to the machine learning community. An understanding of PAQ8 could lead to the design of better algorithms. As stated in [24], PAQ was inspired by research in neural networks: "Schmidhuber and Heil

(1996) developed an experimental neural network data compressor. It used a 3 layer network trained by back propagation to predict characters from an 80 character alphabet in text. It used separate training and prediction phases. Compressing 10 KB of text required several days of computation on an HP 700 workstation." In 2000, Mahoney made several improvements that made neural network compression practical. His new algorithm ran 10^5 times faster. PAQ8 uses techniques (*e.g.* dynamic ensembles) that could lead to advances in machine learning.

As a second contribution, we demonstrate that an understanding of PAQ8 enables us to deploy machine learning techniques to achieve better compression rates. Specifically we show that a second order adaptation scheme, the extended Kalman filter (EKF), results in improvements over PAQ8's first order adaptation scheme.

A third contribution is to present several novel applications of PAQ8. First, we demonstrate how PAQ8 can be applied to adaptive text prediction and game playing. Both of these tasks have been tackled before using other compression algorithms. Second, we show for the first time that PAQ8 can be adapted for classification. Previous works have explored using compression algorithms, such as RAR and ZIP, for classification [25]. We show that our proposed classifier, PAQclass, can outperform these techniques on a text classification task. We also show that PAQclass achieves near state-of-the-art results on a shape recognition task. Finally, we develop a lossy image compression algorithm by combining PAQ8 with recently developed unsupervised feature learning techniques.

1.2 Organization of this Thesis

In Chapter 2 we provide general background information about the problem of lossless data compression, including a description of arithmetic coding and PPM. In Chapter 3 we present a detailed explanation of how PAQ8 works. This chapter also includes a description of our improvement to the compression rate of PAQ8 using EKF. We present several novel applications of PAQ8 in Chapter 4. Chapter 5 contains our conclusions and possible future work. Appendix A contains information on how to access demonstration programs we created using PAQ8.

Chapter 2

Lossless Data Compression

An arbitrary data file can be considered as a sequence of characters in an alphabet. The characters could be bits, bytes, or some other set of characters (such as ASCII or Unicode characters). Lossless data compression usually involves two stages. The first is creating a probability distribution for the prediction of every character in a sequence given the previous characters. The second is to encode these probability distributions into a file using a coding scheme such as arithmetic coding [32] or Huffman coding [16]. Arithmetic coding is usually preferable to Huffman coding because arithmetic coders can produce near-optimal encodings for any set of symbols and probabilities (which is not true of Huffman coders). Since the problem of encoding predicted probability distributions to a file has been solved, the performance difference between compression algorithms is due to the way that they assign probabilities to these predictions. In the next section we give an overview of how arithmetic coding works.

2.1 Arithmetic Coding

Arithmetic coders perform two operations: encoding and decoding. The encoder takes as input a sequence of characters and a sequence of predicted probability distributions. It outputs a compressed representation of the sequence of characters. The decoder takes as input the compressed representation and a sequence of predicted probability distributions. It outputs the original sequence of characters (exactly equivalent to the encoder's input). The sequence of predicted probability distributions needs to be exactly the same for the encoder and the decoder in order for the decoder to reproduce the original character sequence.

From the perspective of the component of a compression program which creates the predicted probability distributions, compression and decompression is equivalent. In order to generate a predicted probability distribution for a specific character, it takes as input all previous characters in the sequence. In the case of compression, it has access to these characters directly from the file it is trying to compress. In the case of decompression, it has access to these characters from the output of the arithmetic decoder. Since the predictor has to output the exact same probability distributions for both compression and decompression, any randomized components of the algorithm need to be initialized with the same seed.

The process used to encode a sequence of characters by an arithmetic encoder is essentially equivalent to storing a single number (between 0 and 1). We will present how this process works through a small example. Suppose we have an alphabet of three characters: A, B, and C. Given the file "ABA" our goal is to compress this string using arithmetic encoding. For the first character prediction, assume that the predictor gives a uniform probability distribution. We can visualize this on a number line, as seen in the top of Figure 2.1. For the second character, assume that the predictor assigns a probability of 0.5 to A, 0.25 to B, and 0.25 to C. Since the first character in the sequence was A, the arithmetic encoder expands this region (0 to 1/3) and assigns the second predicted probability distribution according to this expanded region. This is visualized in the middle layer of Figure 2.1. For the final character in the sequence, assume that the predictor assigns a probability of 0.5 to A, 0.4 to B, and 0.1 to C. This is visualized in the bottom of Figure 2.1. Now all the arithmetic coder needs to do is store a single number between the values of 1/6 and 5/24. This number can be efficiently encoded using a binary search. The binary search ranges would be: "0 to 1", "0 to 0.5", "0 to 0.25", and finally "0.125 to 0.25". This represents the number 0.1875 (which falls in the desired range). If we use "0" to encode the decision to use the lower half and "1" to encode the decision to use the upper half, this sequence can be represented in binary as "001".

Now consider the task of decoding the file. As input, the arithmetic decoder has the number 0.1875, and a sequence of predicted probability distributions. For the

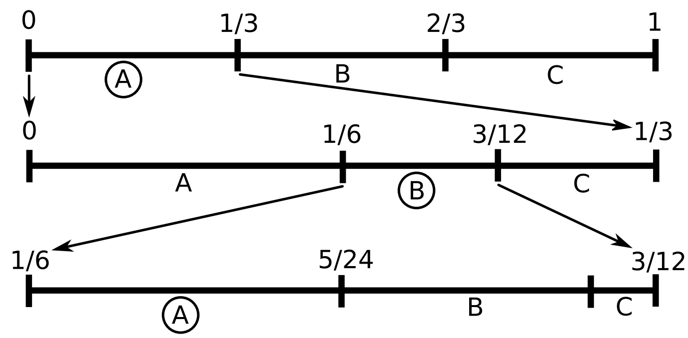

Figure 2.1: An example of arithmetic coding. The alphabet consists of three characters: A, B, and C. The string being encoded is "ABA".

first character, the predictor gives a uniform probability distribution. The number 0.1875 falls into the 'A' sector, so the arithmetic decoder tells the predictor that the first character was 'A'. Similarly, for the next two characters the arithmetic decoder knows that the characters must be 'B' and 'A'. At this point, the arithmetic decoder needs some way to know that it has reached the end of the sequence. There are typically two techniques that are used to communicate the length of the sequence to the decoder. The first is to encode a special "end of sequence" character, so that when the decoder reaches this character it knows it reached the end of the string. The second technique is to just store an additional integer along with the compressed file which represents the length of the sequence (this is usually more efficient in practice).

Although arithmetic coding can achieve optimal compression in theory, in practice there are two factors which prevent this. The first is the fact that files can only be stored to disk using a sequence of bytes, so this requires some overhead in comparison to storing the optimal number of bits. The second is the fact that precision limitations of floating point numbers prevent optimal encodings. In practice both of these factors result in relatively small overhead, so arithmetic coding still produces near-optimal encodings.

Table 2.1: PPM model after processing the string “abracadabra” (up to the second order model). This table is a recreation of a table from [10].

Order k = 2				Order k = 1				Order k = 0			Order k = -1		
Predictions		c	p	Predictions		c	p	Predictions	c	p	Predictions	c	p
ab	→ r	2	$\frac{2}{3}$	a	→ b	2	$\frac{2}{7}$	→ a	5	$\frac{5}{16}$	→ A	1	$\frac{1}{\lvert A\rvert}$
	→ Esc	1	$\frac{1}{3}$		→ c	1	$\frac{1}{7}$	→ b	2	$\frac{2}{10}$			
					→ d	1	$\frac{1}{7}$	→ c	1	$\frac{1}{16}$			
ac	→ a	1	$\frac{1}{2}$		→ Esc	3	$\frac{3}{7}$	→ d	1	$\frac{1}{16}$			
	→ Esc	1	$\frac{1}{2}$					→ r	2	$\frac{2}{16}$			
				b	→ r	2	$\frac{2}{3}$	→ Esc	5	$\frac{5}{16}$			
ad	→ a	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{3}$						
	→ Esc	1	$\frac{1}{2}$										
				c	→ a	1	$\frac{1}{2}$						
br	→ a	2	$\frac{2}{3}$		→ Esc	1	$\frac{1}{1}$						
	→ Esc	1	$\frac{1}{3}$										
				d	→ a	1	$\frac{1}{2}$						
ca	→ d	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{2}$						
	→ Esc	1	$\frac{1}{2}$										
				r	→ a	2	$\frac{2}{3}$						
da	→ b	1	$\frac{1}{2}$		→ Esc	1	$\frac{1}{2}$						
	→ Esc	1	$\frac{1}{2}$										
ra	→ c	1	$\frac{1}{2}$										
	→ Esc	1	$\frac{1}{2}$										

2.2 PPM

PPM [9] is a lossless compression algorithm which consistently performs well on text compression benchmarks. It creates predicted probability distributions based on the history of characters in a sequence using a technique called context matching.

Consider the alphabet of lower case English characters and the input sequence “abracadabra”. For each character in this string, PPM needs to create a probability distribution representing how likely the character is to occur. For the first character in the sequence, there is no prior information about what character is likely to occur, so assigning a uniform distribution is the optimal strategy. For the second character in the sequence, ‘a’ can be assigned a slightly higher probability because it has been observed once in the input history. Consider the task of predicting the character after the entire sequence. One way to go about this prediction is to find the longest match in the input history which matches the most recent input. In this case, the longest match is “abra” which occurs in the first and eighth positions. Based on the longest match, a good prediction for the next character in the sequence is simply the character immediately after the match in the input history. After the string “abra” was the character ‘c’ in the fifth position. Therefore ‘c’ is a good prediction for the next character. Longer context matches can result in better predictions than shorter ones. This is because longer matches are less likely to occur by chance or due to noise in the data.

PPM essentially creates probability distributions according to the method described above. Instead of generating the probability distribution entirely based on the longest context match, it blends the predictions of multiple context lengths and assigns a higher weight to longer matches. There are various techniques on how to go about blending different context lengths. The strategy used for combining different context lengths is partially responsible for the performance differences between various PPM implementations.

One example of a technique used to generate the predicted probabilities is shown in Table 2.1 [10]. The table shows the state of the model after the string “abracadabra” has been processed. ‘k’ is the order of the context match, ‘c’ is the occurence count for the context, and ‘p’ is the computed probability. ‘Esc’ refers to the event of an unexpected character and causes the algorithm to use a lower order model (weighted by the probability of the escape event). Note that the lowest order model (-1) has no escape event since it matches any possible character in the alphabet A.

PPM is a nonparametric model that adaptively changes based on the data it is compressing. It is not surprising that similar methods have been discovered in the field of Bayesian nonparametrics. The stochastic memoizer [39] is a nonparametric model based on an unbounded-depth hierarchical Pitman-Yor process. The stochastic memoizer shares several similarities with PPM implementations. The compression performance of the stochastic memoizer is currently comparable with some of the best PPM implementations.

2.3 Compression Metrics

One way of measuring compression performance is to use the file size of compressed data. However, file size is dependent on a particular type of coding scheme (such as arithmetic coding or Huffman coding). There are two common metrics used to measure the performance directly based on the predicted probability distributions: cross entropy and perplexity. Cross entropy can be used to estimate the average number of bits needed to code each byte of the original data. For a sequence of N characters x_i, and a probability $p(x_i)$ assigned to each character by the prediction algorithm, the cross entropy can be defined as: $-\sum_{i=1}^{N} \frac{1}{N} \log_2 p(x_i)$. This gives the expected number of bits needed to code each character of the string. Another common metric used to compare text prediction algorithms is perplexity which can be defined as two to the power of cross entropy.

2.4 Lossless Image Compression

Compressing images represents a significantly different problem than compressing text. Lossless compression algorithms tend to work best on a sequence of characters which contain relatively little noise. They are well suited for natural language because the characters are highly redundant and contain less noise than individual pixels. The problem with noise is that it reduces the maximum context lengths which an algorithm like PPM can identify. It has been shown that a variant of PPM called prediction by partial approximate matching (PPAM) shows competitive compression rates when compared to other lossless image compression algorithms [40]. PPAM uses approximate matches in order to find longer context lengths which can improve the compression ratio for images.

Another fundamental difference between text and images is that text is a single dimensional sequence of characters while images are two dimensional. Applying PPM to image compression requires mapping the pixels to a single sequential dimension. A trivial way of performing this mapping is using a raster scan (*i.e.*

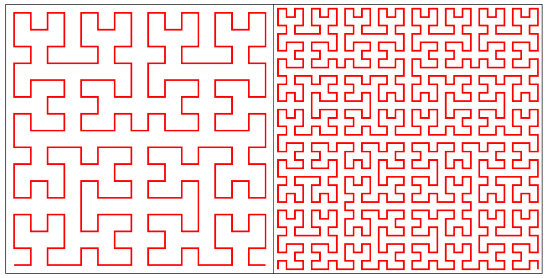

Figure 2.2: The fourth and fifth iteration of the Hilbert curve construction. Image courtesy of Zbigniew Fiedorowicz.

scanning rows from top to bottom and pixels in each row from left to right). Another mapping known as the Hilbert curve [15] maximizes spatial locality (shown in Figure 2.2).

2.5 Distance Metrics

Keogh et al [19] demonstrate how compression algorithms can be used as a distance metric between time series data. This distance metric can be used to solve several interesting problems such as clustering, anomaly detection, and classification. For example, the distance metric can be used to cluster a variety of types of files such as music, text documents, images, and genome sequences. Li et al [22] propose the following metric to measure the distance between the strings x and y:

$$d_k(x,y) = \frac{K(x|y) + K(y|x)}{K(xy)}$$

$K(x)$ is defined to be the Kolmogorov complexity of a string x. That is the length of the shortest program capable of producing x on a universal computer. $K(x|y)$ is the length of the shortest program that computes x when y is given as auxiliary input to the program. $K(xy)$ is the length of the shortest program that outputs y concatenated to x. Kolmogorov complexity represents the best possible compression that can be achieved. Since the Kolmogorov complexity can not be directly computed, a compression algorithm can be used to approximate $d_k(x,y)$:

$$d_c(x,y) = \frac{C(x|y)+C(y|x)}{C(xy)}$$

$C(x)$ is the compressed size of x and $C(x|y)$ is the compressed size of x after the compressor has been trained on y. The better a compression algorithm, the closer it approaches the Kolmogorov complexity and the closer $d_c(x,y)$ approximates $d_k(x,y)$. Keogh et al [19] use the following dissimilarity metric:

$$d_{CDM}(x,y) = \frac{C(xy)}{C(x)+C(y)}$$

The main justification made for using $d_{CDM}(x,y)$ over $d_c(x,y)$ is that it does not require the calculation of $C(x|y)$ and $C(y|x)$. These can not be calculated using most off-the-shelf compression algorithms without modifying their source code. Fortunately, PAQ8 is open source and these modifications can be easily implemented (so d_c can be used). For the purposes of classification, we investigated defining our own distance metrics. Using cross entropy, a more computationally efficient distance metric can be defined which requires only one pass through the data:

$$d_{e1}(x,y) = E(x|y)$$

$E(x|y)$ is the cross entropy of x after the compressor has been trained on y. We also investigated a symmetric version of this distance metric, in which $d_{e2}(x,y)$ is always equal to $d_{e2}(y,x)$:

$$d_{e2}(x,y) = \frac{E(x|y)+E(y|x)}{2}$$

Finally, Cilibrasi et al [8] propose using the following distance metric:

$$d_{NDM}(x,y) = \frac{C(xy)-min(C(x),C(y))}{max(C(x),C(y))}$$

In section 4.2 of this thesis, we use a compression-based distance metric to perform classification. Keogh et al [19] use the ZIP compression algorithm as a distance metric to perform experiments in clustering, anomaly detection, and classification. Since PAQ8 achieves better compression than ZIP, it should theoretically result in a better distance metric. Although we do not perform the experiments in this thesis, it would make interesting future work to compare ZIP and PAQ8 in the experiments by Keogh et al.

Chapter 3

PAQ8

3.1 Architecture

PAQ8 uses a weighted combination of predictions from a large number of models. Most of the models are based on context matching. Unlike PPM, some of the models allow noncontiguous context matches. Noncontiguous context matches improve noise robustness in comparison to PPM. This also enables PAQ8 to capture longer-term dependencies. Some of the models are specialized for particular types of data such as images or spreadsheets. Most PPM implementations make predictions on the byte-level (given a sequence of bytes, they predict the next byte). However, all of the models used by PAQ8 make predictions on the bit-level.

Some architectural details of PAQ8 depend on the version used. Even for a particular version of PAQ8, the algorithm changes based on the type of data detected. For example, fewer prediction models are used when image data is detected. We will provide a high-level overview of the architecture used by `paq8l` in the general case of when the file type is not recognized. `paq8l` is a stable version of PAQ8 released by Matt Mahoney in March 2007. The PAQ8 versions submitted to the Hutter prize include additional language modeling components not present in `paq8l` such as dictionary preprocessing and word-level modeling.

An overview of the `paq8l` architecture is shown in Figure 3.1. 552 prediction models are used. The model mixer combines the output of the 552 predictors into a single prediction. This prediction is then passed through an adaptive probability

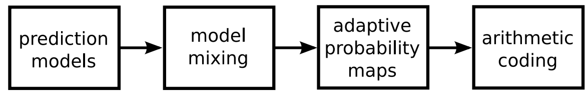

Figure 3.1: PAQ8 architecture.

map (APM) before it is used by the arithmetic coder. In practice, APMs typically reduce prediction error by about 1%. APMs are also known as secondary symbol estimation [24]. APMs were originally developed by Serge Osnach for PAQ2. An APM is a two dimensional table which takes the model mixer prediction and a low order context as inputs and outputs a new prediction on a nonlinear scale (with finer resolution near 0 and 1). The table entries are adjusted according to prediction error after each bit is coded.

3.2 Model Mixer

The `paq8l` model mixer architecture is shown in Figure 3.2. The architecture closely resembles a neural network with one hidden layer. However, there are some subtle differences that distinguish it from a standard neural network. The first major difference is that weights for the first and second layers are learned online and independently for each node. Unlike back propagation for a multi-layer network, each node is trained separately to minimize the predictive cross-entropy error, as outlined in section 3.2.2. In this sense, PAQ8 is a type of ensemble method [27]. Unlike typical ensembles, the parameters do no converge to fixed values unless the data is stationary. PAQ8 was designed for both stationary and non-stationary data[1].

The second major difference between the model mixer and a standard neural network is the fact that the hidden nodes are partitioned into seven sets. For every bit of the data file, one node is selected from each set. The set sizes are shown in the rectangles of Figure 3.2. We refer to the leftmost rectangle as set 1 and the right-most rectangle as set 7. Only the edges connected to these seven selected nodes are updated for each bit of the data. That means of the 552×3,080 = 1,700,160

[1] We refer to "non-stationary data" as data in which the statistics change over time. For example, we would consider a novel to be non-stationary while a text document of some repeating string (*e.g.* "abababab...") to be stationary.

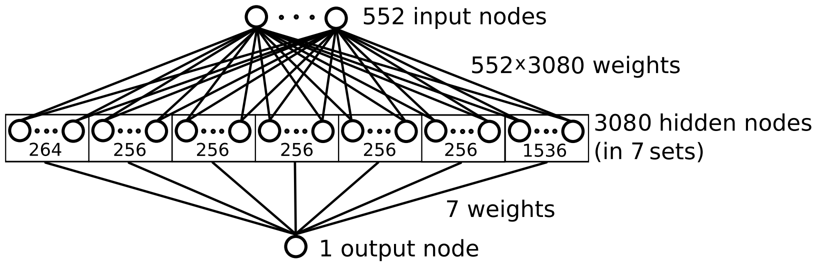

Figure 3.2: PAQ8 model mixer architecture.

weights in the first layer, only 552×7 = 3,864 of the weights are updated for each bit. This makes training the neural network several orders of magnitude faster.

Each set uses a different selection mechanism to choose a node. Sets number 1, 2, 4, and 5 choose the node index based on a single byte in the input history. For example, if the byte for set 1 has a value of 4, the fifth node of set 1 would be selected. Set 1 uses the second most recent byte from the input history, set 2 uses the most recent byte, set 4 uses the third most recent byte, and set 5 uses the fourth most recent byte. Set 6 chooses the node based on the length of the longest context matched with the most recent input. Sets 3 and 7 use a combination of several bytes of the input history in order to choose a node index. The selection mechanism used by `paq8l` is shown in Algorithm 1. $history(i)$ returns the i'th most recent byte, $lowOrderMatches$ the number of low-order contexts which have been observed at least once before (between 0 and 7), $lastFourBytes$ is the four most recent bytes, $longestMatch$ is the length of the longest context match (between 0 and 65534), $bitMask(x, y)$ does a bitwise AND operation between x and y, and $bitPosition$ is the bit index of the current byte (between 0 and 7).

3.2.1 Mixtures of Experts

In the previous section we compared the PAQ8 model mixer to a multilayer neural network. The PAQ8 model mixer can also be compared to a technique known as "mixtures of experts" [18]. Although PAQ8 does not use the standard mixtures of experts architecture, they do share some similarities. Jacobs et al [18] state: "If backpropagation is used to train a single, multilayer network to perform different

Algorithm 1 `paq8l` node selection mechanism.

```
set1Index← 8 + history(1)
set2Index← history(0)
set3Index← lowOrderMatches + 8 × ((lastFourBytes/32) mod (8))
if history(1) = history(2) then
    set3Index← set3Index + 64
end if
set4Index← history(2)
set5Index← history(3)
set6Index← round(log_2(longestMatch) × 16)
if bitPosition = 0 then
    set7Index← history(3)/128 + bitMask(history(1),240) + 4×(history(2)/64) +
    2×(lastFourBytes / 2^31)
else
    set7Index← history(0) ×2^(8−bitPosition)
    if bitPosition = 1 then
        set7Index← set7Index + history(3)/2
    end if
    set7Index← min(bitPosition,5) × 256 + history(1)/32 + 8 × (history(2)/32) + bit-
    Mask(set7Index,192)
end if
```

subtasks on different occasions, there will generally be strong interference effects which lead to slow learning and poor generalization. If we know in advance that a set of training cases may be naturally divided into subsets that correspond to distinct subtasks, interference can be reduced by using a system composed of several different 'expert' networks plus a gating network that decides which of the experts should be used for each training case." Their architecture is shown in Figure 3.3.

PAQ8 and the mixtures of experts architecture both use a gating mechanism to choose expert models. The same problem-specific properties which lead to the development of mixtures of experts also applies to compression - data can be naturally divided into subsets and separate 'experts' can be trained on each subset. Using a gating mechanism has the additional computational benefit that only one expert needs to be trained at a time, instead of training all experts simultaneously. Increasing the number of expert networks does not increase the time complexity of the algorithm.

One difference between the mixtures of experts model and the PAQ8 model mixture is the gating mechanism. Jacobs et al use a feedforward network to learn the gating mechanism, while PAQ8 uses a deterministic algorithm (shown in Algo-

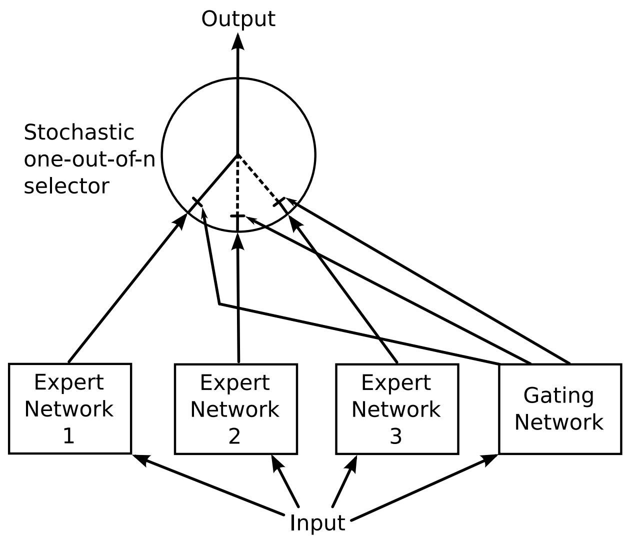

Figure 3.3: Mixtures of experts architecture. This figure is a recreation of a figure in [18]. All of the experts are feedforward networks and have the same input. The gating network acts as a switch to select a single expert. The output of the selected expert becomes the output of the system. Only the weights of the selected expert are trained.

rithm 1) which does not perform adaptive learning. The gating algorithm used by PAQ8 contains problem-specific knowledge which is specified a priori. One interesting area for future work would be to investigate the effect of adaptive learning in the PAQ8 gating mechanism. Adaptive learning could potentially lead to a better distribution of the data to each expert. Ideally, the data should be uniformly partitioned across all the experts. However, using a deterministic gating mechanism runs the risk of a particular expert being selected too often.

There is a direct mapping between the mixtures of experts architecture and the PAQ8 model mixer architecture. Each "set" in the hidden layer of Figure 3.2 corresponds to a separate mixtures of experts model. The seven mixtures of experts are then combined using an additional feedforward layer. The number of experts in each mixtures of experts model corresponds to the number of nodes in each set (*e.g.* there are 264 experts in set 1 and 1536 experts in set 7). As with the mixtures of experts architecture, only the weights of the expert chosen by the gating mechanism are trained for each bit of the data. Another difference between the standard mixtures of experts model and PAQ8 is the fact that mixtures of experts models typically are optimized to converge towards a stationary objective function while PAQ8 is designed to adaptively train on both stationary and non-stationary data.

3.2.2 Online Parameter Updating

Each node of the `paq8l` model mixer (both hidden and output) is a Bernoulli logistic model:

$$p(y_t|\mathbf{x}_t, \mathbf{w}) = \text{Ber}(y_t|\text{sigm}(\mathbf{w}^T\mathbf{x}_t))$$

where $\mathbf{w} \in \mathbb{R}^{n_p}$ is the vector of weights, $\mathbf{x}_t \in [0,1]^{n_p}$ is the vector of predictors at time t, $y_t \in \{0,1\}$ is the next bit in the data being compressed, and $\text{sigm}(\eta) = 1/(1+e^{-\eta})$ is the sigmoid or logistic function. n_p is the number of predictors and is equal to 552 for the first layer of the neural network and 7 for the second layer of the network. Let $\pi_t = \text{sigm}(\mathbf{w}^T\mathbf{x}_t)$. The negative log-likelihood of the t-th bit is given by

$$NLL(\mathbf{w}) \quad = \quad -\log[\pi_t^{\mathbb{I}(y_t=1)} \times (1-\pi_t)^{\mathbb{I}(y_t=0)}] = -[y_t \log \pi_t + (1-y_t)\log(1-\pi_t)]$$

where $\mathbb{I}(\cdot)$ denotes the indicator function. The last expression is the cross-entropy error (also known as coding error) function term at time t. The logistic regression weights are updated online with first order updates:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \nabla NLL(\mathbf{w}_{t-1}) = \mathbf{w}_{t-1} - \eta(\pi_t - y_t)\mathbf{x}_t$$

The step size η is held constant to ensure ongoing adaptation.

3.2.3 Extended Kalman Filter

To improve the compression rate of `paq8l`, we applied an extended Kalman filter (EKF) to adapt the weights. We assume a dynamic state-space model consisting of a Gaussian transition prior, $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathcal{N}(0, \mathbf{Q})$, and a logistic observation model $y_t = \pi_t + \mathcal{N}(0, r)$. The EKF, although based on local linearization of the observation model, is a second order adaptive method worthy of investigation. One of the earliest implementations of EKF to train multilayer perceptrons is due to Singhal and Wu [34]. Since EKF has a $O({n_p}^2)$ time complexity, it would be unfeasibly slow to apply EKF to the first layer of the neural network. However, EKF can be used to replace the method used by `paq8l` in the second layer of the neural network without significant computational cost since there are only seven weights.

Here we present the EKF algorithm for optimizing the second layer of the PAQ8 neural network. The following values were used to initialize EKF: $\mathbf{Q} = 0.15 \times \mathbf{I}_{7\times 7}$, $\mathbf{P}_0 = 60 \times \mathbf{I}_{7\times 7}$, $\mathbf{w}_0 = 150 \times \mathbf{1}_{7\times 1}$, and $r = 5$. The following are the EKF update equations for each bit of data:

$$\begin{aligned}
\mathbf{w}_{t+1|t} &= \mathbf{w}_t \\
\mathbf{P}_{t+1|t} &= \mathbf{P}_t + \mathbf{Q} \\
\mathbf{K}_{t+1} &= \frac{\mathbf{P}_{t+1|t}\mathbf{G}'_{t+1}}{r + \mathbf{G}_{t+1}\mathbf{P}_{t+1|t}\mathbf{G}'_{t+1}} \\
\mathbf{w}_{t+1} &= \mathbf{w}_{t+1|t} + \mathbf{K}_{t+1}(y_t - \pi_t) \\
\mathbf{P}_{t+1} &= \mathbf{P}_{t+1|t} - \mathbf{K}_{t+1}\mathbf{G}_{t+1}\mathbf{P}_{t+1|t},
\end{aligned}$$

where $\mathbf{G}_{1\times 7}$ is the Jacobian matrix: $\mathbf{G} = [\partial y/\partial w_1 \ \cdots \ \partial y/\partial w_7]$ with $\partial y/\partial w_i = y(1-y)x_i$. We compared the performance of EKF with other variants of `paq8l`. The results are shown in Table 3.1. The first three columns are `paq8l` with different settings of the *level* parameter. *level* is the only `paq8l` parameter that can be changed via command-line (without modifying the source code). It makes a trade-off between speed, memory usage, and compression performance. It can be set to an integer value between zero and eight. Lower values of *level* are faster and use less memory but achieve worse compression performance. *level*=8 is the slowest setting and uses the most memory (up to 1643 MiB) but achieves the best compression performance. *level*=5 has a 233 MiB memory limit. `paq8-8-tuned` is a customized version of `paq8l` (with *level*=8) in which we changed the value of the weight initialization for the second layer of the neural network. We found changing the initialization value from 32,767 to 128 improved compression performance. Finally, `paq8-8-ekf` refers to our modified version of `paq8l` with EKF used to update the weights in the second layer of the neural network. We

Table 3.1: PAQ8 compression rates on the Calgary corpus

FILE	paq8l-1	paq8l-5	paq8l-8	paq8-8-tuned	paq8-8-ekf
BIB	1.64592	1.49697	1.49645	1.49486	1.49207
BOOK1	2.14158	2.00573	2.00078	2.00053	1.99603
BOOK2	1.73257	1.59531	1.5923	1.59198	1.58861
GEO	3.70451	3.43456	3.42725	3.42596	3.43444
NEWS	2.07839	1.90573	1.90284	1.90237	1.89887
OBJ1	3.25932	2.77358	2.77407	2.76531	2.76852
OBJ2	1.85614	1.45499	1.43815	1.43741	1.43584
PAPER1	2.09455	1.96543	1.96542	1.96199	1.95753
PAPER2	2.09389	1.99046	1.99053	1.9882	1.98358
PIC	0.6604	0.35088	0.35083	0.35073	0.3486
PROGC	2.07449	1.91574	1.91469	1.91037	1.9071
PROGL	1.31293	1.18313	1.18338	1.1813	1.18015
PROGP	1.31669	1.1508	1.15065	1.14757	1.14614
TRANS	1.1021	0.99169	0.99045	0.98857	0.98845
Average	**1.93382**	**1.72964**	**1.72698**	**1.7248**	**1.72328**

find that using EKF slightly outperforms the first order updates. The improvement is about the same order of magnitude as the improvement between *level*=5 and *level*=8. However, changing *level* has a significant cost in memory usage, while using EKF has no significant computational cost. The initialization values for `paq8-8-tuned` and `paq8-8-ekf` were determined using manual parameter tuning on the first Calgary corpus file ('bib'). The performance difference between `paq8l-8` and `paq8-8-tuned` is similar to the difference between `paq8-8-tuned` and `paq8-8-ekf`.

Chapter 4

Applications

4.1 Adaptive Text Prediction and Game Playing

The fact that PAQ8 achieves state of the art compression results on text documents indicates that it can be used as a powerful model for natural language. PAQ8 can be used to find the string x that maximizes $p(x|y)$ for some training string y. It can also be used to estimate the probability $p(z|y)$ of a particular string z given some training string y. Both of these tasks are useful for several natural language applications. For example, many speech recognition systems are composed of an acoustic modeling component and a language modeling component. PAQ8 could be used to directly replace the language modeling component of any existing speech recognition system to achieve more accurate word predictions.

Text prediction can be used to minimize the number of keystrokes required to type a particular string [12]. These predictions can be used to improve the communication rate for people with disabilities and for people using slow input devices (such as mobile phones). We modified the source code of `paq8l` to create a program which predicts the next n characters while the user is typing a string. A new prediction is created after each input character is typed. It uses `fork()` after each input character to create a process which generates the most likely next n characters. `fork()` is a system call on Unix-like operating systems which creates an exact copy of an existing process. The program can also be given a set of files to train on.

M—ay the contemplation of so many wonders extinguish the spirit ofvengeance in him!
My— companions and I had decided to escape as soon as the vessel cameclose enough for us to be heard
My n—erves calmed a little, but with my brain so aroused,I did a swift review of my whole existence
My name i—n my ears and some enormous baleen whales
My name is B—ay of Bengal, the seas of the East Indies, the seasof China
My name is Byr—on and as if it was an insane idea. But where the lounge.I stared at the ship bearing
My name is Byron K—eeling Island disappeared below the horizon,
My name is Byron Kn—ow how the skiff escaped from the Maelstrom'sfearsome eddies,
My name is Byron Knoll.— It was an insane idea. Fortunately I controlled myselfand stretched
My name is Byron Knoll. My name is B—yron Knoll. My name is Byron Knoll. My name is Byron Knoll.

F—or example, consider the form of the exponential family
Fi—gure~\ref{fig:betaPriorPost}(c) shows what happens as the number of heads in the past data.
Figure o—f the data, as follows: \bea\gauss(\mu|\gamma, \lambda(2 \alpha-1))
Figure ou—r conclusions are a convex combination of the prior mean and the constraints
Figure out Bayesian theory we must. Jo—rdan conjugate prior
Figure out Bayesian theory we must. Jos—h Tenenbaum point of the posterior mean is and mode of the
Figure out Bayesian theory we must. Josh agrees. Long live P(\vtheta—|\data)

Figure 4.1: Two examples of PAQ8 interactive text prediction sessions. The user typed the text in boldface and PAQ8 generated the prediction after the "—" symbol. We shortened some of the predictions for presentation purposes. In the top example, PAQ8 was trained on "Twenty Thousand Leagues Under the Seas" (Jules Verne, 1869). In the bottom example, PAQ8 was trained on the LaTeX source of an anonymous ML book.

Some preliminary observational studies on our text prediction system are shown in Figure 4.1. Note that PAQ8 continuously does online learning, even while making a prediction (as seen by the completion of "Byron Knoll" in the top example). The character predictions do capture some syntactic structures (as seen by completion of LaTeX syntax) and even some semantic information as implied by the training text.

Sutskever et al [35] use Recurrent Neural Networks (RNNs) to perform text prediction. They also compare RNNs to the sequence memoizer and PAQ8 in terms of compression rate. They conclude that RNNs achieve better compression than the sequence memoizer but worse than PAQ8. They perform several text prediction tasks using RNNs with different training sets (similar to the examples in Figure 4.1). One difference between their method and ours is the fact that PAQ8 continuously does online learning on the test data. This feature could be beneficial for text prediction applications because it allows the system to adapt to new users and data that does not appear in the training set.

We found that the PAQ8 text prediction program could be modified into a rock-paper-scissors AI that usually beats human players. Given a sequence of the opponent's rock-paper-scissors moves (such as "rpprssrps") it predicts the most likely next move for the opponent. In the next round the AI would then play the move that beats that prediction. The reason that this strategy usually beats human players is that humans typically use predictable patterns after a large number of rock-paper-scissors rounds. The PAQ8 text prediction program and rock-paper-scissors AI are available to be downloaded (see Appendix A).

4.2 Classification

In many classification settings of practical interest, the data appears in sequences (*e.g.* text). Text categorization has particular relevance for classification tasks in the web domain (such as spam filtering). Even when the data does not appear to be obviously sequential in nature (*e.g.* images), one can sometimes find ingenious ways of mapping the data to sequences.

Compression-based classification was discovered independently by several researchers [25]. One of the main benefits of compression-based methods is that they are very easy to apply as they usually require no data preprocessing or parameter tuning. There are several standard procedures for performing compression-based classification. These procedures all take advantage of the fact that when compressing the concatenation of two pieces of data, compression programs tend to achieve better compression rates when the data share common patterns. If a data point in the test set compresses well with a particular class in the training set, it likely belongs to that class. Any of the distance metrics defined in Section 2.5 can be directly used to do classification (for example, using the k-nearest neighbor algorithm). We developed a classification algorithm using PAQ8 and show that it can be used to achieve competitive classification rates in two disparate domains: text categorization and shape recognition.

4.2.1 Techniques

Martin et al [25] describe three common compression-based classification procedures: standard minimum description length (SMDL), approximate minimum description length (AMDL), and best-compression neighbor (BCN). Suppose each

Table 4.1: The number of times each bit of data gets compressed using different compression-based classification methods. N_X is the number of training files, N_Y is the number of test files, and N_Z is the number of classes.

METHOD	TRAINING DATA	TEST DATA
SMDL	1	N_Z
AMDL	$N_Y + 1$	N_Z
BCN	$N_Y + 1$	N_X

data point in the training and test sets are stored in separate files. Each file in the training set belongs to one of the classes $C_1, ..., C_N$. Let the file A_i be the concatenation of all training files in class C_i. SMDL runs a compression algorithm on each A_i to obtain a model (or dictionary) M_i. Each test file T is compressed using each M_i. T is assigned to the class C_i whose model M_i results in the best compression of T. While compressing T, the compression algorithm does not update the model M_i.

For a file F, let $f(F)$ be the file size of the compressed version of F. Also, let A_iT be the file A_i concatenated with T. For AMDL, T is assigned to the class C_i which minimizes the difference $f(A_iT)$ - $f(A_i)$. Let B be a file in the training set. BCN checks every pair BT and assigns T to the class which minimizes the difference $f(BT)$ - $f(B)$.

A speed comparison between these methods can be made by considering how many times each bit of the data gets compressed, as shown in Table 4.1. It should be noted that the primary difference between SMDL and AMDL is the fact that SMDL only processes the training data once, while AMDL reprocesses the training data for every file in the test set. For many datasets, the number of training and test files (N_X and N_Y) are much larger than the number of classes (N_Z). That means that SMDL can be orders of magnitude faster than AMDL (and AMDL faster than BCN). Although PAQ8 achieves state of the art compression rates, it is also extremely slow compared to the majority of compression algorithms. Using PAQ8 for classification on large datasets would be unfeasibly slow using AMDL or BCN.

4.2.2 PAQclass

AMDL and BCN both work with off-the-shelf compression programs. However, implementing SMDL usually requires access to a compression program's source code. Since PAQ8 is open source, we modified the source code of `paq8l` (a version of PAQ8) to implement SMDL. We call this classifier PAQclass. To the best of our knowledge, PAQ has never been modified to implement SMDL before. We changed the source code to call `fork()` when it finishes processing data in the training set (for a particular class). One forked process is created for every file in the test set. This essentially copies the state of the compressor after training and allows each test file to be compressed independently. Note that this procedure is slightly different from SMDL because the model M_i continues to be adaptively modified while it is processing test file T. However, it still has the same time complexity as SMDL. `paq8l` has one parameter to set the compression level. We used the default parameter setting of 5 during classification experiments.

Compression performance for AMDL and BCN is measured using file size. The use of file size is fundamentally limited in two ways. The first is that it is only accurate to within a byte (due to the way files are stored on disk). The second is that it is reliant on the non-optimal arithmetic coding process to encode files to disk. Cross entropy is a better measurement of compression performance because it is subject to neither of these limitations. Since we had access to the `paq8l` source code, we used cross entropy as a measure of compression performance instead of file size.

4.2.3 Text Categorization

Text categorization is the problem of assigning documents to categories based on their content. We evaluated PAQclass on the 20 Newsgroup (20news) dataset. This dataset contains 18,828 newsgroup documents partitioned (nearly) evenly across 20 categories. The number of documents in each category is shown in Table 4.2. We used J. Rennie's version of the corpus[1]. In this version of the corpus duplicate postings to more than one newsgroup were removed. Most message headers were also removed, while the "Subject" and "From" fields were retained.

[1] http://people.csail.mit.edu/people/jrennie/20Newsgroups/20news-18828.tar.gz

Table 4.2: Number of documents in each category of the 20news dataset.

CLASS	COUNT
ALT.ATHEISM	799
COMP.GRAPHICS	973
COMP.OS.MS-WINDOWS.MISC	985
COMP.SYS.IBM.PC.HARDWARE	982
COMP.SYS.MAC.HARDWARE	961
COMP.WINDOWS.X	980
MISC.FORSALE	972
REC.AUTOS	990
REC.MOTORCYCLES	994
REC.SPORT.BASEBALL	994
REC.SPORT.HOCKEY	999
SCI.CRYPT	991
SCI.ELECTRONICS	981
SCI.MED	990
SCI.SPACE	987
SOC.RELIGION.CHRISTIAN	997
TALK.POLITICS.GUNS	910
TALK.POLITICS.MIDEAST	940
TALK.POLITICS.MISC	775
TALK.RELIGION.MISC	628
Total	**18,828**

We evaluated PAQclass using randomized 80-20 train-test splits. The 80-20 split seems to be the most common evaluation protocol used on this dataset. No document preprocessing was performed. The results are shown in Table 4.3.

Our result of 92.3526% correct is competitive with the best results published for this dataset. Table 4.4 shows some comparative results. It should be noted that there are several versions of the 20news dataset and many publications use different evaluation protocols. Some of these published results can not be directly compared. For example, Zhang & Oles [41] report a figure of 94.8% correct on a version of the dataset in which the "Newsgroup:" headers were not removed from messages. The four best results (including PAQclass) in Table 4.4 [20, 25, 38] all seem to use the same version of 20news. PAQclass outperforms classification using the RAR compression algorithm [25] on this dataset.

Table 4.3: Classification results on the 20news dataset. Each row shows one run of a randomized 80-20 train-test split.

CORRECT CLASSIFICATIONS (OUT OF 3766)	PERCENT CORRECT
3470	92.1402
3482	92.4588
3466	92.034
3492	92.7244
3480	92.4057
Average	**92.3526**

Table 4.4: Comparative results on the 20news dataset. Our results are in boldface.

METHODOLOGY	PROTOCOL	PERCENT CORRECT
EXTENDED VERSION OF NAIVE BAYES [31]	80-20 TRAIN-TEST SPLIT	86.2
SVM + ERROR CORRECTING OUTPUT CODING [30]	80-20 TRAIN-TEST SPLIT	87.5
LANGUAGE MODELING [28]	80-20 TRAIN-TEST SPLIT	89.23
AMDL USING RAR COMPRESSION [25]	80-20 TRAIN-TEST SPLIT	90.5
MULTICLASS SVM + LINEAR KERNEL [38]	70-30 TRAIN-TEST SPLIT	91.96
PAQclass	**80-20 train-test split**	**92.35**
MULTINOMIAL NAIVE BAYES + TFIDF [20]	80-20 TRAIN-TEST SPLIT	93.65

4.2.4 Shape Recognition

Shape recognition is the problem of assigning images to categories based on the shape or contour of an object within the image. We evaluated PAQclass on the

Table 4.5: Number of images in each class of the chicken dataset.

CLASS	COUNT
BACK	76
BREAST	96
DRUMSTICK	96
THIGH AND BACK	61
WING	117
Total	**446**

chicken dataset[2] [1]. This dataset contains 446 binary images of chicken parts in five categories (see Figure 4.5). Example images from this dataset are shown in Figure 4.2. The chicken pieces in the images are not set to a standard orientation. The images are square and vary in resolution from 556×556 to 874×874.

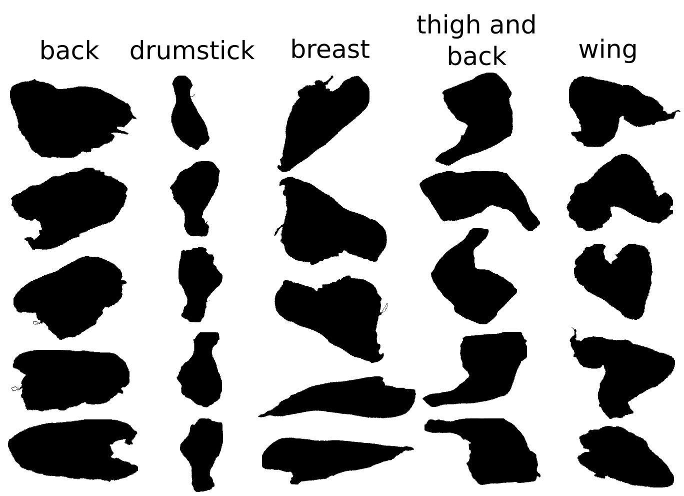

Figure 4.2: Five example images from each class of the chicken dataset. The images have not been rotated.

As discussed in Section 2.4, compressing images poses a significantly different problem compared to compressing text. There does not seem to be a large body

[2] http://algoval.essex.ac.uk/data/sequence/chicken

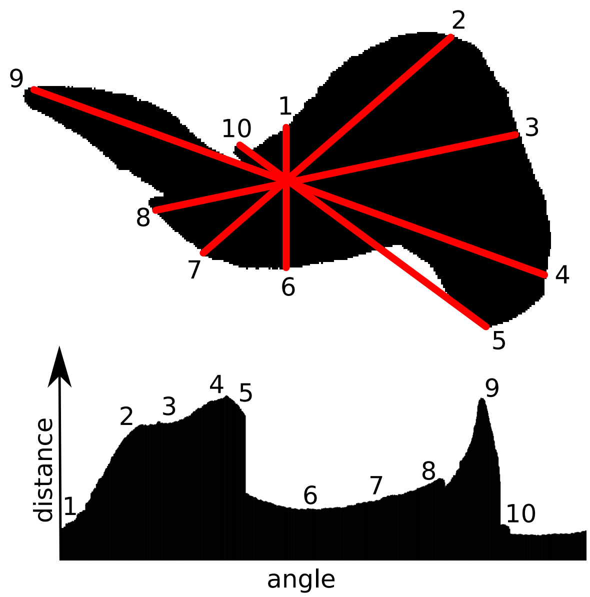

Figure 4.3: An example of converting a shape into one-dimensional time series data. The original image is shown on top (part of the "wing" class of the chicken dataset) and the time series data shown on the bottom. Points along the contour have been labeled and the corresponding points on the time series are shown.

of research on using compression-based methods for image classification (in comparison to text categorization). This may be due to the fact that compression-based methods tend to be slow and may be infeasible for the large datasets often used for object recognition tasks.

Lossy compression algorithms can be used for performing compression-based classification. There are several options for creating one-dimensional lossy representations of images. For example, Watanabe et al [36] demonstrate a method of converting images to text. They show that their system is effective for image classification tasks. Wei et al [37] describe a method of converting shape contours into time series data. They use this representation to achieve successful classification results on the chicken dataset. Based on these results, we decided to combine this representation with PAQ8 classification.

Figure 4.3 demonstrates how we convert images in the chicken dataset into one-

Table 4.6: Leave-one-out classification results on the chicken dataset with different settings of the "number of measurements" parameter. There are a total of 446 images. The row with the best classification results is in boldface.

NUMBER OF MEASUREMENTS	CORRECT CLASSIFICATIONS	PERCENT CORRECT
1	162	36.3229
5	271	60.7623
10	328	73.5426
30	365	81.8386
35	380	85.2018
38	365	81.8386
39	363	81.3901
40	**389**	**87.2197**
41	367	82.287
42	367	82.287
45	359	80.4933
50	352	78.9238
100	358	80.2691
200	348	78.0269
300	339	76.009

dimensional time series data. The first step is to calculate the centroid of the shape. We project a ray from the centroid point and measure the distance to the edge of the shape. If the ray intersects the shape edge at multiple points, we take the furthest intersection (as seen at point 5 in Figure 4.3). We rotate the ray around the entire shape and take measurements at uniform intervals. The number of measurements taken along the shape contour is a tunable parameter. Once the Euclidean distance is measured for a particular angle of the ray, it is converted into a single byte by rounding the result of the following formula: $(100 * distance/width)$. $distance$ is the Euclidean distance measurement and $width$ is the width of the image. PAQclass is then run on this binary data.

The classification results for different settings of the "number of measurements" parameter are shown in Table 4.6. We used leave-one-out cross-validation since this seems to be the most common evaluation protocol used on this dataset.

The "number of measurements" parameter had a surprisingly large effect on classification accuracy. Adjusting the parameter by a single measurement from the best value (40) resulted in ≈ 5 to 6% loss in accuracy. Another unfortunate property of adjusting this parameter is that the classification accuracy is not a convex function (as seen at the parameter value 35). This means finding the optimal value of the parameter would require an exhaustive search. Due to time constraints, we did not perform an exhaustive search (only the experiments in Table 4.6 were performed). Table 4.7 shows a confusion matrix at the best parameter setting.

Our result of 87.2197% correct classifications is among the best results published for this dataset. Table 4.8 shows some comparative results.

The classification procedure we used is not rotationally invariant. Since the chicken pieces in the dataset can be in arbitrary orientations, this could lead to a decrease in classification accuracy. Wei et al [37] use the same one dimensional image representation as we use, but they use a rotationally invariant classification procedure. They use a 1-nearest-neighbor classifier combined with a Euclidean distance metric. When comparing the distance between two images, they try all possible rotations and use the angle which results in the lowest Euclidean distance between the time series representations. This same procedure of trying all possible orientations could be used to make the PAQclass classification procedure

Table 4.7: Confusion matrix for chicken dataset with the "number of measurements" parameter set to 40. C1=back, C2=breast, C3=drumstick, C4=thigh and back, C5=wing.

		Predicted				
		C1	C2	C3	C4	C5
Actual	C1	**55**	10	2	2	7
	C2	0	**93**	0	3	0
	C3	0	5	**84**	0	7
	C4	0	8	0	**48**	5
	C5	3	1	3	1	**109**

Table 4.8: Comparative results on the chicken dataset. Our results are in boldface.

METHODOLOGY	PROTOCOL	PERCENT CORRECT
1-NN + LEVENSHTEIN EDIT DISTANCE [26]	LEAVE-ONE-OUT	≈ 67
1-NN + HMM-BASED DISTANCE [4]	LEAVE-ONE-OUT	73.77
1-NN + MBM-BASED FEATURES [4]	LEAVE-ONE-OUT	76.5
1-NN + APPROXIMATED CYCLIC DISTANCE [26]	LEAVE-ONE-OUT	≈ 78
1-NN + CONVERT TO TIME SERIES [37]	LEAVE-ONE-OUT	80.04
SVM + HMM-BASED ENTROPIC FEATURES [29]	LEAVE-ONE-OUT	81.21
SVM + HMM-BASED NONLINEAR KERNEL [6]	50-50 TRAIN-TEST SPLIT	85.52
SVM + HMM-BASED FISHER KERNEL [5]	50-50 TRAIN-TEST SPLIT	85.8
PAQclass + convert to time series	**leave-one-out**	**87.22**

rotationally invariant (although it would increase the algorithm's running time). Alternatively, we could use a single rotationally invariant representation such as always setting the smallest sampled edge distance to be at $angle = 0$. The effect of rotational invariance on classification accuracy would make interesting future work.

4.3 Lossy Compression

PAQ8 can also be used for performing lossy compression. Any lossy representation can potentially be passed through PAQ8 to achieve additional compression. For example, `paq8l` can losslessly compress JPEG images by about 20% to 30%. `paq8l` contains a model specifically designed for JPEG images. It essentially undoes the lossless compression steps performed by JPEG (keeping the lossy rep-

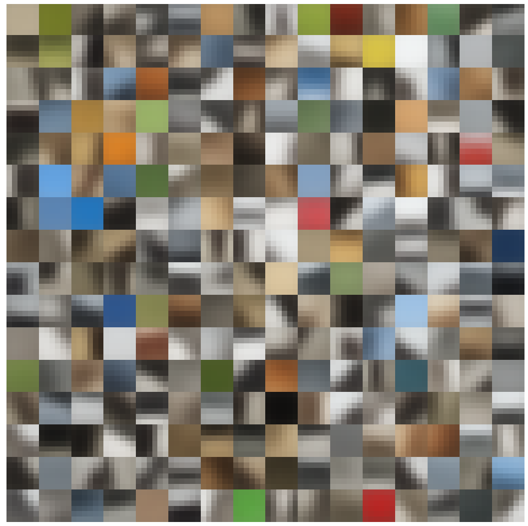

Figure 4.4: 256 6×6 image filters trained using k-means on the CIFAR-10 dataset.

resentation) and then performs lossless compression more efficiently. To create a lossy image compression algorithm, we first created a set of filters based on a method described by Coates et al [11]. We used the k-means algorithm to learn a set of 256 6×6 filters on the CIFAR-10 image dataset [21]. The filters were trained using 400,000 randomly selected images patches. The filters are shown in Figure 4.4.

In order to create a lossy image representation, we calculated the closest filter match to each image patch in the original image. These filter selections were encoded by performing a raster scan through the image and using one byte per patch to store the filter index. These filter selections were then losslessly compressed using `paq8l`. Some example images compressed using this method are shown in Figures 4.5, 4.6, 4.7, and 4.8. At the maximum JPEG compression rate, the JPEG images were still larger than the images created using our method. Even at a larger file size the JPEG images appeared to be of lower visual quality compared to the images compressed using our method. We also compared against the more advanced lossy compression algorithm JPEG2000. JPEG2000 has been designed

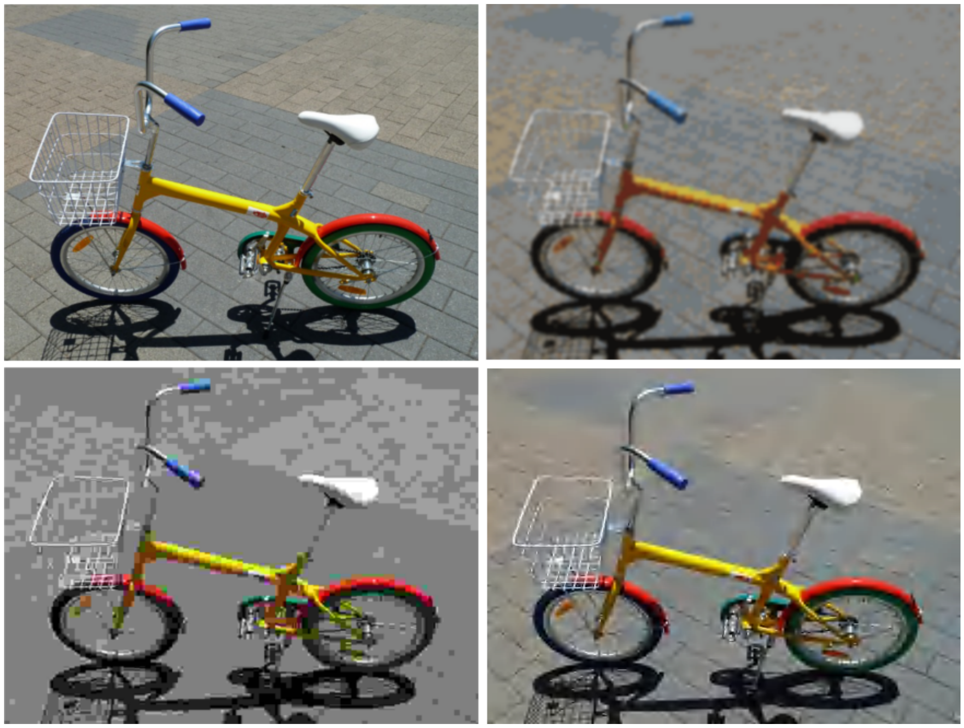

Figure 4.5: top-left image: original (700×525 pixels), top-right image: our compression method (4083 bytes), bottom left: JPEG (16783 bytes), bottom-right: JPEG2000 (4097 bytes)

to exploit limitations of human visual perception: the eye is less sensitive to color variation at high spatial frequencies and it has different degrees of sensitivity to brightness variation depending on spatial frequency [24]. Our method was not designed to exploit these limitations (we leave this as future work). It simply uses the filters learned from data. Based on the set of test images, JPEG2000 appears to outperform our method in terms of visual quality (at the same compression rate).

Figure 4.6: top-left image: original (525×700 pixels), top-right image: our compression method (1493 bytes), bottom left: JPEG (5995 bytes), bottom-right: JPEG2000 (1585 bytes)

Figure 4.7: top-left image: original (700×525 pixels), top-right image: our compression method (3239 bytes), bottom left: JPEG (16077 bytes), bottom-right: JPEG2000 (2948 bytes)

Figure 4.8: top-left image: original (700×525 pixels), top-right image: our compression method (3970 bytes), bottom left: JPEG (6335 bytes), bottom-right: JPEG2000 (3863 bytes)

Chapter 5

Conclusion

We hope this technical exposition of PAQ8 will make the method more accessible and stir up new research in the area of temporal pattern learning and prediction. Casting the weight updates in a statistical setting already enabled us to make modest improvements to the technique. We tried several other techniques from the fields of stochastic approximation and nonlinear filtering, including the unscented Kalman filter, but did not observe significant improvements over the EKF implementation. One promising technique from the field of nonlinear filtering we have not yet implemented is Rao-Blackwellized particle filtering for online logistic regression [2]. We leave this for future work.

The way in which PAQ8 *adaptively* combines predictions from multiple models using context matching is different from what is typically done with mixtures of experts and ensemble methods such as boosting and random forests. A statistical perspective on this, which allows for a generalization of the technique, should be the focus of future efforts. Bridging the gap between the online learning framework [7] and PAQ8 is a potentially fruitful research direction. Recent developments in RNNs seem to be synergistic with PAQ8, but this still requires methodical exploration.

On the application front, we found it remarkable that a single algorithm could be used to tackle such a broad range of tasks. In fact, there are many other tasks that could have been tackled, including clustering, compression-based distance metrics, anomaly detection, speech recognition, and interactive interfaces. It is equally remarkable how the method achieves comparable results to state-of-the-art in text classification and image compression.

Bibliography

[1] G. Andreu, A. Crespo, and J. M. Valiente. Selecting the toroidal self-organizing feature maps (TSOFM) best organized to object recognition. In *International Conference on Neural Networks*, volume 2, pages 1341–1346, 1997.

[2] C. Andrieu, N. de Freitas, and A. Doucet. Rao-Blackwellised particle filtering via data augmentation. *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[3] T. Bell, J. Cleary, and I. Witten. *Text compression*. Prentice-Hall, Inc., 1990.

[4] M. Bicego and A. Trudda. 2D shape classification using multifractional brownian motion. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342 of *Lecture Notes in Computer Science*, pages 906–916. Springer, 2008.

[5] M. Bicego, M. Cristani, V. Murino, E. Pekalska, and R. Duin. Clustering-based construction of hidden markov models for generative kernels. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 5681 of *Lecture Notes in Computer Science*, pages 466–479. Springer, 2009.

[6] A. Carli, M. Bicego, S. Baldo, and V. Murino. Non-linear generative embeddings for kernels on latent variable models. In *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 154–161, 2009.

[7] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[8] R. Cilibrasi and P. M. B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[9] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[10] J. Cleary, W. Teahan, and I. Witten. Unbounded length contexts for PPM. *Data Compression Conference*, 1995.

[11] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. *AISTATS 14*, 2011.

[12] N. Garay-Vitoria and J. Abascal. Text prediction systems: a survey. *Universal Access in the Information Society*, 4:188–203, 2006.

[13] J. Gasthaus, F. Wood, and Y. W. Teh. Lossless compression based on the sequence memoizer. *Data Compression Conference*, pages 337–345, 2010.

[14] J. Hawkins and S. Blakeslee. *On Intelligence*. Owl Books, 2005.

[15] D. Hilbert. Ueber die stetige abbildung einer line auf ein flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.

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

[17] M. Hutter. The human knowledge compression prize. http://prize.hutter1.net, accessed April 15, 2011.

[18] R. Jacobs, M. Jordan, S. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[19] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, 2004.

[20] A. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive Bayes for text categorization revisited. In *Advances in Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2005.

[21] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[22] M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001.

[23] M. Mahoney. Adaptive weighing of context models for lossless data compression. Florida Tech. Technical Report, CS-2005-16, 2005.

[24] M. Mahoney. Data compression explained. http://mattmahoney.net/dc/dce.html, accessed April 15, 2011.

[25] Y. Marton, N. Wu, and L. Hellerstein. On compression-based text classification. In *Advances in Information Retrieval*, volume 3408 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2005.

[26] R. A. Mollineda, E. Vidal, and F. Casacuberta. Cyclic sequence alignments: Approximate versus optimal techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 16:291–299, 2002.

[27] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study. In *Journal of Artificial Intelligence Research 11*, pages 169–198, 1999.

[28] F. Peng, D. Schuurmans, and S. Wang. Augmenting naive Bayes classifiers with statistical language models. *Information Retrieval*, 7:317–345, 2004.

[29] A. Perina, M. Cristani, U. Castellani, and V. Murino. A new generative feature set based on entropy distance for discriminative classification. In *Image Analysis and Processing ICIAP 2009*, volume 5716 of *Lecture Notes in Computer Science*, pages 199–208. Springer, 2009.

[30] J. D. M. Rennie. Improving multi-class text classification with naive Bayes. Master's thesis, M.I.T., 2001.

[31] J. D. M. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumptions of naive Bayes text classifiers. In *International Conference on Machine Learning (ICML)*, pages 616–623, 2003.

[32] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM J. Res. Dev.*, 23: 149–162, 1979.

[33] D. Shkarin. PPM: one step to practicality. In *Data Compression Conference*, pages 202–211, 2002.

[34] S. Singhal and L. Wu. Training multilayer perceptrons with the extended kalman algorithm. In *Advances in neural information processing systems (NIPS)*, pages 133–140, 1989.

[35] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2011.

[36] T. Watanabe, K. Sugawara, and H. Sugihara. A new pattern representation scheme using data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):579–590, 2002.

[37] L. Wei, E. Keogh, X. Xi, and M. Yoder. Efficiently finding unusual shapes in large image databases. *Data Mining and Knowledge Discovery*, 17: 343–376, 2008.

[38] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, 2009.

[39] F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y. W. Teh. A stochastic memoizer for sequence data. In *International Conference on Machine Learning (ICML)*, pages 1129–1136, 2009.

[40] Z. Yong and D. A. Adjeroh. Prediction by partial approximate matching for lossless image compression. *IEEE Transactions on Image Processing*, 17 (6):924–935, 2008.

[41] T. Zhang and F. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.

Appendix A

PAQ8 Demonstrations

Two of the applications described in Section 4.1 are available to be downloaded[1]. The first demonstration is text prediction and the second demonstration is a rock-paper-scissors AI. Instructions are provided on how to compile and run the programs in Linux.

[1] http://cs.ubc.ca/~knoll/PAQ8-demos.zip