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

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

The goal of this paper 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.

## 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 nonparametric Bayesian statistics [1] and deep learning [2], 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.

The close connections between sequence prediction and data compression are perhaps underappreciated within the machine learning community. The goal of this paper is to describe a state-of-the-art compression method called *PAQ8* [3] 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 [4]. 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. PAQ8 is a version of PAQ which achieves record breaking compression ratios at the expense of increased speed and memory usage. For example, all of the winning submissions in the Hutter Prize [5], 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].

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 accessible material detailing the algorithms underlying PAQ8. We begin this paper with a detailed description of PAQ8 and its relationship to PPM. We describe the models and algorithms underlying PAQ8 and propose an improvement to the PAQ8 method based on replacing its first-order parameter update scheme with a more sophisticated second-order extended Kalman filter, achieving improved lossless compression results on a text compression benchmark. We demonstrate an extension of PAQ8 to lossy image

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

Table 1: *Cross entropy rates for compression algorithms on the Calgary corpus. Each file is compressed independently and the resulting cross entropy rates are then averaged.*

| PPM-TEST | PPM*C | 1PF | UKN | CPPMII-64 | PAQ8L |
|---|---|---|---|---|---|
| **2.23** | **2.34** | **2.12** | **2.12** | **2.04** | **1.73** |

compression. We show results indicating that it achieves higher perceptual quality than standard JPEG compression with a much better compression ratio. We also apply PAQ8 to the problem of text classification using a compression-for-classification scheme [6]. We achieve near state-of-the-art results on this task. Finally, we discuss several other applications of the predictive models underlying PAQ8 including text prediction and adaptive game playing.

## 2 Predictive coding and PAQ8

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 [7] or Huffman coding [8]. 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. The literature on arithmetic coding is broad and easily accessible. For these reasons, we focus on the predictive aspect of the problem.

### 2.1 PPM

Prediction by Partial Matching (PPM) [4] is a lossless compression algorithm which consistently performs well on text compression benchmarks. It creates predictive probability distributions based on the history of characters in a sequence using a technique called context matching. Context matching involves examining the most recent input and finding the longest match of that string in the input history. A probability distribution for the next character given its context can be computed based on the empirical probability of all context matches in the input history. 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. Instead of generating the probability distribution entirely based on the longest context match, PPM blends the predictions of multiple context lengths and assigns a higher weight to longer matches. A more detailed description of PPM appears in the supplementary material.

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 [1] is a nonparametric model based on an unbounded-depth hierarchical Pitman-Yor process. The stochastic memoizer shares several similarities with PPM implementations and achieves similar compression rates.

Table 1 shows a comparison of several compression algorithms on the Calgary corpus [9], a widely-used compression benchmark. `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 [11]. `cPPMII-64` [12] is currently among the best PPM implementations. `paq8l` outperforms all of these compression algorithms by what is considered to be be a very large margin in this benchmark.

## 2.2 PAQ8 architecture

To the best of our knowledge, there exist only incomplete high-level descriptions of PAQ1 through 6 [3] and PAQ8 [13]. The C++ source code, although available, is very close to machine language and the underlying algorithms are difficult to extract. Most of architectural details in this paper about PAQ8 were understood by examining the source code and are presented here for the first time.

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 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 includes additional language modeling components such as dictionary preprocessing and word-level modeling.

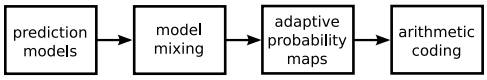


Figure 1: *PAQ8 architecture.*

An overview of the `paq8l` architecture is shown in Figure 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 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 [13]. 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.

## 2.3 Model mixer

The `paq8l` model mixer architecture is shown in Figure 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 2.3.1. In this sense, PAQ8 is a type of ensemble method and closely related to a mixture of experts model. Unlike typical ensembles, the parameters do not converge to fixed values unless the data is stationary. PAQ8 was designed for both stationary and non-stationary data.

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 2. We refer to the leftmost rectangle as set 1 and the rightmost 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 $552 \times 3{,}080 = 1{,}700{,}160$ weights in the first layer, only $552 \times 7 = 3{,}864$ 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 hidden 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

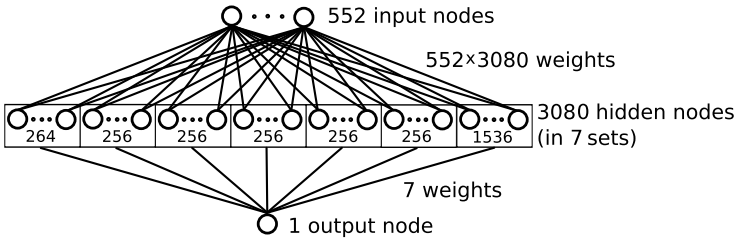


Figure 2: *PAQ8 model mixer.*

**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) $\times 2^{8-\mathrm{bitPosition}}$
  **if** bitPosition = 1 **then**
    set7Index← set7Index + history(3)/2
  **end if**
  set7Index← min(bitPosition,5) × 256 + history(1)/32 + 8 × (history(2)/32) + bitMask(set7Index,192)
**end if**

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).

### 2.3.1 Online parameter updating

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

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

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 $\mathrm{sigm}(\eta) = 1/(1 + e^{-\eta})$ 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 $\pi_t = \mathrm{sigm}(\mathbf{w}^T\mathbf{x}_t)$. The negative log-likelihood of the $t$-th bit is given by

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

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 $\eta$ is held constant (typically a value between 0.002 and 0.01) to ensure ongoing adaptation.

Table 2: *PAQ8 compression rates on the Calgary corpus.*

| paq8l-1 | paq8l-5 | paq8l-8 | paq8-8-tuned | paq8-8-ekf |
|---|---|---|---|---|
| **1.93382** | **1.72964** | **1.72698** | **1.7248** | **1.72328** |

### 2.3.2 EKF parameter updates

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 [14]. 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: $\mathbf{Q} = 0.15 \times \mathbf{I}_{7\times7}$, $\mathbf{P}_0 = 60 \times \mathbf{I}_{7\times7}$, $\mathbf{w}_0 = 150 \times \mathbf{1}_{7\times1}$, 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 $\mathbf{G}_{1\times7}$ 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 2. 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 tradeoff 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 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`.

## 3 Applications

### 3.1 Classification and PAQclass

Compression-based classification was discovered independently by several researchers [6]. One of the main benefits of compression-based methods is that they are very easy to apply as they usually

Table 3: *Comparative results on the 20news dataset. Our results are in boldface.*

| METHODOLOGY | PROTOCOL | PERCENT CORRECT |
|---|---|---|
| EXTENDED VERSION OF NAIVE BAYES [15] | 80-20 TRAIN-TEST SPLIT | 86.2 |
| SVM + ERROR CORRECTING OUTPUT CODING [16] | 80-20 TRAIN-TEST SPLIT | 87.5 |
| LANGUAGE MODELING [17] | 80-20 TRAIN-TEST SPLIT | 89.23 |
| AMDL USING RAR COMPRESSION [6] | 80-20 TRAIN-TEST SPLIT | 90.5 |
| MULTICLASS SVM + LINEAR KERNEL [18] | 70-30 TRAIN-TEST SPLIT | 91.96 |
| **PAQclass** | **80-20 train-test split** | **92.35** |
| MULTINOMIAL NAIVE BAYES + TFIDF [19] | 80-20 TRAIN-TEST SPLIT | 93.65 |

require little 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.

Marton et al [6] describe three common compression-based classification procedures: standard minimum description length (SMDL), approximate minimum description length (AMDL), and best-compression neighbor (BCN). Suppose each 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 $M_i$. 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 $T$.

We describe the details and computational complexity of AMDL and BCN in the supplementary material. It suffices to state here that using PAQ8 for classification on large datasets would be unfeasibly slow using AMDL or BCN, but not using SMDL with the modifications described subsequently. 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` 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). `fork()` is a system call on Unix-like operating systems which creates an exact copy of an existing process. 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 the model $M_i$ continues to be adaptively modified while it is processing test file $T$.

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. We used 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.

We evaluated PAQclass using five randomized 80-20 train-test splits. Using randomized 80-20 splits and taking the average over multiple runs seems to be the most common evaluation protocol used on the dataset. No document preprocessing was performed. The results are shown in Table 3. Our result of 92.35% correct is competitive with the best results published for 20news. PAQclass outperforms classification using the RAR compression algorithm [6] on this dataset.

### 3.2 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

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

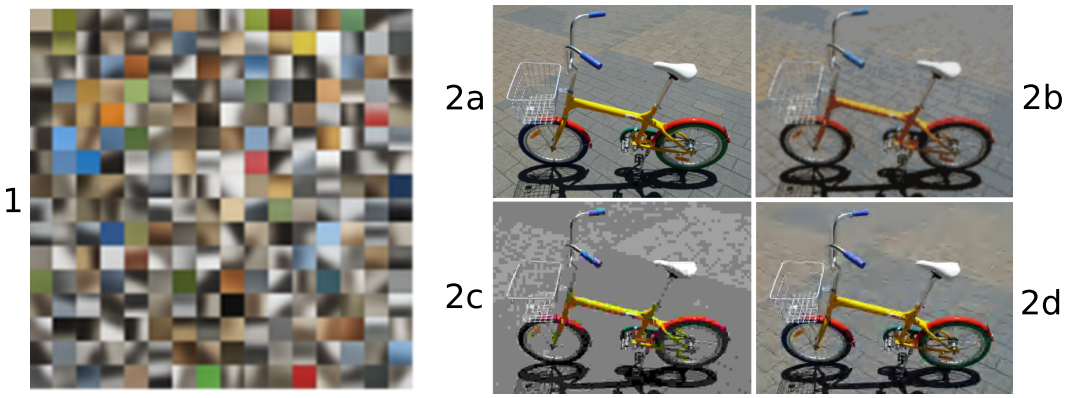


Figure 3: *1: The 256 6×6 image filters learned on the CIFAR-10 dataset. 2a: original image (700×525 pixels). 2b: our compression method (4083 bytes). 2c: JPEG (16783 bytes). 2d: JPEG2000 (4097 bytes).*

the lossy representation) and then performs lossless compression more efficiently. To create a lossy image compression algorithm, we first created a set of filters based on the method described by Coates et al [20]. 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 image patches. The filters are shown on the left of Figure 3.

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`. One example image compressed using this method is shown on the right of Figure 3. We also evaluated our method on several other test images. 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 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 [13]. Our method was not designed to exploit these limitations. 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).

### 3.3 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 machine learning 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 [22]. 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. The program can also be given a set of files to train on. We performed some text prediction experiments shown in Figure 4. Note that PAQ8 continuously does online learning, even while making a prediction.

**N**—ed Land some devilfish and are commonplace, the very homeland of the northern shore the day of March 13
**Ni**—ght and day of June 10, 1770. The compass indicated that it was letting the warship approach.
**Nips paper submission hinges on thi**—s manuscript that containson the horizon,
**Nips paper submission hinges on this demo. G**—ulf Stream is seventy-fivemiles wide and again and couldn't be recovered
**Nips paper submission hinges on this demo. Geoffrey** —didn't answer him.I could feel the anger building in him.
**Nips paper submission hinges on this demo. Geoffrey Hinton will like it. Hopefully. Thanks.**—In any event, it's better to perish
**Nips paper submission hinges on this demo. Geoffrey Hinton will like it. Hopefully. Thanks. G**—eoffrey Hinton will like it. Hopefully.
**Nips paper submission hinges on this demo. Geoffrey Hinton will like it. Hopefully. Thanks. Geoffrey is a** —deep

**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)) \Ga(\lambda|\alpha,\beta)\eea
**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 prior,and the conjugate prior mean
**Figure out Bayesian theory we must. Josh agrees. Long live P(\vtheta**—|\data)

Figure 4: *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). Note that as the user types an input string, the algorithm adapts (as seen by the completion of "Geoffrey Hinton"). In the bottom example, PAQ8 was trained on the LaTeX source of an anonymous ML book. 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. We invite readers to try testing PAQ8 text prediction using the program included in the supplementary materials.*

Sutskever et al [2] 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). 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. We invite the readers to try playing rock-paper-scissors against PAQ8 using the program included in the supplementary materials.

## 4 Conclusions and future work

We believe this technical exposition of PAQ8 will make the method more accessible and stimulate 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. An interesting possible extension is Rao-Blackwellized particle filtering for online logistic regression [23]. 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. 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.

## References


[1] 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.

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

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

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

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

[6] 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.

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

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

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

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

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

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

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

[14] 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.

[15] 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.

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

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

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

[19] 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.

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

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

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

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