Using XCS as a Prediction Engine in Data Compression

Mohsen Sharifi Amir Aavani Shahab Tasharrofi

Computer Engineering Department, Iran University of Science & Technology

Abstract

XCS has been used in several fields as a prediction engine before; its population based nature enables XCS to generate sets of properly pruned classification rules. Further, unlike other population based algorithms, it learns using rewards. These properties encouraged us to use it as a predictor engine for lossless data compression. In the compression context, XCS can be used to find the hidden relations in files of the same type. So, we used it as a preprocessor before an entropy encoder, to remove the existing correlations between file's bits or symbols. Removing correlations causes entropy encoders to achieve higher rates of compression. The results support this conclusion.

Keywords: Data Compression, Learning Classifier System, Entropy

1. Introduction

Data compression is a branch of information theory in which the goal is to represent data in a file, stream, and etc using as little as possible bits. In general, data compression consists of taking a stream of symbols and transforming them into codes. If the compression is effective, the resulting stream of codes will be smaller than the original symbols. The decision to output a certain code for a certain symbol or set of symbols is based on a model. The model is simply a collection of data and rules used to process input symbols and determine which code(s) to output [1].

The concept of entropy was first introduced by Shannon in his paper "A Mathematical Theory of Communication" published in 1948. Shannon proved that the entropy rate of a data source means the average number of bits (codes) per symbol needed to encode it [2]. Entropy effectively bounds the performance of the best lossless compression possible, which can be realized in theory by using the typical set [13] or in practice using Huffman, Lempel-Ziv, or arithmetic coding [1]. The entropy is closely related with *Markov Model* of the source. The more complicated the model, the higher the entropy.

If one reformulates the data compression into a *prediction problem*, then to achieve a good compression rate, one needs a prediction engine which, given the current context, guesses the next few symbols of the source as accurately as possible. Classifiers have been used as predictors in several fields such as financial forecasting [5] and opponent modeling [14]. One can use a classifier as a tool to partition data into two classes, namely 0 and 1. The main problem with using classifiers in compression is that there are a huge number of configurations which should be mapped to one of the classes, 0 or 1 [7]. To fulfill the performance measure, the classifiers should meet the need of having a high probability of correct classification.

We have used a special form of classifiers, the XCS classifiers, to predict the input stream. XCS has not been widely in use as a prediction engine before. However, the possibility of such a usage is discussed in [15]. The most important feature of XCS is that it generates a rule-based system as a result of training phase. This feature makes XCS a very effective method of making prediction engines as rule-based systems are one of the most widely used frameworks to develop predicting systems [16], [17]. Furthermore, XCS' population based nature provides a competitive environment which, by itself, results in generating sets of properly pruned classification rules [6]. Further, unlike other population based algorithms, which need an explicit fitness function to operate, XCS learns using rewards. This gives XCS a unique advantage, compared to the other population based algorithms, of being able to learn interactively. These properties encouraged us to use XCS as a predictor engine for lossless data compression. It is noteworthy that there is a distinction between when one is using 'classifier' to refer to a whole system (like Neural Network Classifiers) and when using it to refer to a rule in a classifier system (like XCS). In the context of XCS, when referring to a classifier, one usually has a single rule (instead of the whole rule-based system) in mind.

In what follows, section 2 discusses the definition and usage of "Predictor Functions". It also presents how predictors could be applied to the field of data compression. Section 3 gives a brief description of

XCS classifier systems. Section 4 discusses the architecture which includes XCS as its core. It also describes the logic behind how XCS works. Section 5 focuses on implementation details which we considered while applying XCS to the field of data compression. These details include how to choose every single parameter of XCS algorithm. Section 6 contains some algorithms to increase the overall functionality and some future works to be pursued. Section 7 discusses train set and test set and the results achieved by applying the method on them.

2. Predictor functions

In a predictive encoder, we are interested in predicting the next k input symbols $y = y_1, y_2, \cdots, y_k$, given a vector $\mathbf{x} = x_1, x_2, ..., x_M$ using a predictor function, where each x_i is 1 if a particular feature is present in the input history, and 0 otherwise [7].

In the field of compression, there are many ways to use this schema. One of which is to consider a file as a stream of symbols, and then, by fixing M, the size of input vector, one can use the last M symbols of the stream as the input to the predictor function. It is logical, then, to select k to be 1, i.e. the predictor function should only deal with guessing the next symbol of the stream. As the underlying model of sources is usually thought to be a finite Markov Model, this method can fully be justified.

Selecting the symbols to be bits of input stream is a custom choice [7]. Based on the same assumption that we did in the previous paragraph, i.e. the underlying models of sources are Markov Models, it seems that the adjacent bits have higher correlation compared to the other sets of bits. For example, when one sees the "tha" in an English text, they expect the character "t" after it. In some contexts, however, there may be cases in which the adjacent bits are not so correlated. And, thus, no finite Markov Model could be a good model of the source for these contexts. We will discuss it later in section 7.

This is a common choice among predictive encoders to have distinct symbols, usually a bit with distinct values of 0 and 1, and to put one of them when it has guessed the next source's symbol right and the other followed by the right symbol if not. In the case of predicting only one bit per time, however, saving the correct bit in case of incorrect prediction is not necessary as there is only one way around. It is easy to check that the original symbol (here the input bit) is, then, equal to the result of applying XOR operator on the output bit and the result of the predictor.

This approach to predict one bit of input stream per time results in a stream which has the same length as the input stream. Although this may seem odd for the purpose of compression, having a more precise look at the output stream demonstrates that having a good predictor will result in a low error rate which is, by the method described, equal to a lot of zeros on the output stream, which, usually, decreases the entropy (and achieves a higher compression rate). As a result, existing entropy coders could compress the resulted stream better than the main stream. Like what has been shown in [8].

It is proven that Bayesian Classifiers guarantee the minimum expected error [9]. But we already know that implementing a Bayesian Classifier needs a huge amount of input resources (to have a good estimation of probabilities of symbols) and is impractical for large attribute vectors. So, we need a predictor that is able to handle large attribute vectors and is fast enough to be comparable with existing methods. Mahoney uses Neural Networks to predict the input stream [10]. We have used XCS as a predictor function that is described in section 4.

3. An overview of XCS

XCS, developed by Wilson [12], is a learning classifier system [6]. It uses reinforcement learning along with genetic algorithms in order to search for the best rule-based. XCS is designed for both single-step and multiple-step tasks, but the discussion here is restricted to only the version that works for single-step tasks in which the system, by receiving an input data, makes a decision and selects an action. System does not receive any input before applying the selected action to the environment and, later, receiving the reward of the action back from the environment [6].

XCS is a collection of classifiers [6]. Each classifier has a pair of <CONDITION, ACTION> and a set of parameters. Conditions can be thought as a string of {0, 1, #}, where # matches any character and so called "*don't care*" symbol, but 0 and 1 match only 0 and 1, respectively. Each classifier has three principal parameters:

1) *Prediction,* an estimate for the reward that will be received from environment, if the classifier's action will be selected on an input that matches condition.
2) *Prediction error,* an estimate of the difference between reward and *Prediction*.

After receiving an input, XCS generates a set of classifiers called "match set" whose conditions match the input data. Size of "match set" must be bigger than a specified threshold (θ_{mna}). If the number of matched classifiers is less than θ_{mna}, then XCS tries to generate some classifiers whose conditions match the input.

In next step, an action should be chosen. XCS does not prescribe any particular action-selection method but usually the following method is used. XCS chooses action according to classifiers prediction and classifiers fitness with probability of *1- P_{exp}* and chooses a random action between those which are available in "match set" with probability of P_{exp}. In this way, XCS is allowed to explore the environment.

After choosing an action, "action set" must be generated from the "match set". "Action set" is a subset of "match set", in which all classifiers' actions are the same as selected action. "Action set" is the most important part of XCS, because all updates occur on it.

The agent sends the selected action to the environment and receives the reward (or penalty) from environment. Reward is a measure of how good the selected action had been.

After receiving reward, agent updates the *prediction*, *prediction error* and *fitness* parameters of the classifiers which are in "action set". Update step involves updating the *prediction* parameter based on its last value and the reward, and then, *error estimate* and *fitness* parameters.

XCS runs a genetic algorithm on the member of "action set" time by time. Two classifiers based on their fitness are chosen and a two-point crossover will be applied on them and two new children (classifiers) will be created. These new classifiers will be inserted into the population.

A method for deleting some classifiers exists. If the number of classifiers exceeds a threshold (N), XCS uses it to remove some classifiers. Each classifier has a probability to delete. This probability is calculated according to classifier fitness and the action set size estimate.

4. Using XCS as a predictor function

As it is shown in Fig.1, every bit in stream is fed into both SR box and DCF box. Then, the content of SR box is fed to XCS and the result of XCS box also is sent to DCF box. The result of DCF box is the correspondent output of input bit. The process is the same in both encoding and decoding procedures.

4.1. SR Box

SR box is an M-bit shift register that is initially filled with zeros. It consists of M previous bits of input and is used to represent the attribute vector $X(x_1, x_2 \ldots x_M)$.

4.2. XCS Box

The input of the XCS box, as it is shown in Fig.1, is the vector **X** from SR box and its output is the action 0 or 1 that will be sent to DCF box.

This box does not completely simulate XCS. It only generates "match set" and selects an action based on it (with $P_{exp} = 0.0$). This box also does not contain any update procedures of XCS. It uses predefined classifiers to generate the "match set". If "match set"

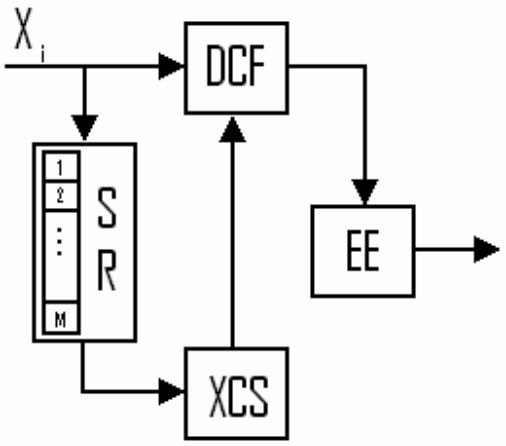

Fig. 1: Schema of predictor function.

is empty, an action that has the maximum occurrence will be taken. Fig.2 shows the contents of XCS box.

The classifiers of XCS box are trained offline on large sets of input. Compression with predefined classifiers is called to be offline, but we have also designed (not implemented yet) an online version that will be described briefly in section 6.

Although a general purpose data compressor ideally treats all input files in the same manner, but the poor results of training a set of classifiers for all the sample files together led us to train different sets of classifiers for different file types such as "text files", "Pascal sources", "HTML files", "monochrome bitmaps", and "color bitmaps".

It should be noted that although training of XCS box is done off-line, compression actually occurs at run time and is desired to be as fast as possible.

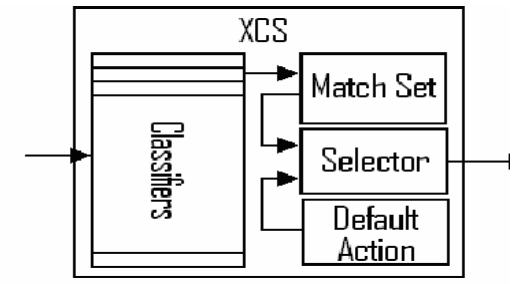

Fig. 2: Schema of XCS box.

4.2.1. Benefits of XCS in compression context

XCS keeps the classifiers which have greater impact due to their higher support and confidence. So, the classifiers with low support (classifiers which are rarely used) or low confidence (classifiers which are not reliable) will be removed. This leads to better expected error rates compared to other classification systems.

The parameter N of XCS enables us to restrict its size. So, it will not use huge memory resources like Bayesian classifiers. It also provides control over generality of classification.

XCS could be applied to various file types and the resultant set of classifiers could be used without any further considerations. Its results are expected to be extremely better than general classifiers as it is described in section 7.

Finding an action based on classifiers is a really fast process and is linear respect to N and M. It processes about 14000 characters per second.

The "*don't care*" symbol in classifiers' conditions and the ability of merging two classifiers with the same action and nearly similar conditions causes the resultant classifiers' conditions to focus only on important features which have high correlation with the action. This process is called "subsumption procedure".

4.2.2. How does XCS Reduce the Entropy?

If the classifiers predict the input almost accurately, the resultant file will contain lots of zeros, and so the probability of zeros and patterns that have zeros will be increased. So, based on Shannon's formula, the entropy of the message will be decreased [2]. XCS, in this context, is used as a learner that learns the input language and predicts the next symbols. For example, in case of Pascal sources on which we trained the XCS, it was clear (by checking the generated rules) that it has learnt keywords of Pascal language and some common Pascal patterns like the new line code after "begin" keyword or "for i:=" token.

About the "colored bitmaps", we trained the system in two different ways. First, we trained the XCS on the whole file. Resulting set of classifiers learned relation of colors (small regions have the same color). Second, we separated each bitmap to 24 files (each containing the correspondent bits of colors). And we also trained 24 sets of classifiers for each group of files (one for the files containing most significant bits, and so on). The resultant sets of classifiers learned the properties of each bit. For example, the set of classifiers for the most significant bit learned how to predict the changes in luminosity of pictures.

4.3. Difference calculator function (DCF)

The DCF box simply implements an XOR on the bit from input stream and the predicted bit of XCS box. It assures us that the whole process is reversible and the resulting stream contains more zeros if the predictor acts accurately. It also makes the encoder and decoder processes like each other.

4.4. Entropy encoder (EE)

EE box could contain an implementation of any compression algorithm like LZW, Huffman, Arithmetic coding, etc. Our implementation takes advantage of a 1-bit arithmetic coder [1].

5. Implementation

As Fig.1 suggests, the main focus of the system should be on the XCS box. Our implementation of XCS takes the following parameters into account:

- $P_{\#}$: This parameter represents the probability of "don't care" symbol and is highly related to the nature of the file type which we are working on. For example, $P_{\#}$ could have higher values in text files (0.2 in our implementation) than that of monochrome bitmaps (0.12 in our implementation). So, we need some experts that could define the best value of $P_{\#}$. Though we have defined it by "trial and error", we could use some control mechanisms that has been described in section 6. As it is described in more detail there, we have always underestimated the $P_{\#}$ value. This value was set to 0.15 for "Pascal sources" and "HTML files" and 0.1 for both implementations of "color bitmaps".
- N: This parameter represents the size of population (i.e. the maximum number of classifiers in XCS). This parameter has high correlation with entropy of the file type which we are working on. For classifying a file type with high information content comparing with the one with lower information content, more classifiers are needed.

We have set this parameter (in our implementation) to 5000 for "text files", 2000 for "html files" and "Pascal sources" and 7000 for "monochrome bitmaps" and 15000 for both implementations of "color bitmaps". This parameter was set based on the fact that for example, "Pascal sources" have less information content than "text files" because they have some structural grammar.

- M: This parameter represents the number of features used for prediction. Unlike other AI problems in which the number of features (the number of inputs of the system) is known, this problem (compression using prediction) has unbounded number of features. In fact, as it could be inferred from what Shannon says, if you could use more bits for prediction, the order with which you are compressing the file will increase and so, you could compress the file better. We have set this parameter (in our implementation) to 64 (i.e. last 8 characters) for "text files", 48 for "Pascal sources", 40 for "html files", 48 for that implementation of "color bitmaps" which does not separate the colors and 24 for both "monochrome bitmaps" and the other implementation of "color bitmaps".
- P_{exp}: This parameter represents the probability with which we explore the environment. Like every other reinforcement learning algorithm, this parameter is initialized with high values (0.7 in our implementation) and will be decreased through the train process by a coefficient (0.95 in our implementation that is applied every 50 steps). This value would not go under a lower bound (0.05 in our implementation).

There were also other parameters that either we did not take into account or we did not change the values from what is stated in [6].

A more general subsumption procedure has also been implemented. This procedure combines the classifiers with same action which their condition's Hamming distance is one and they have near fitness. The new subsumption procedure will be applied every I_{sub} step (where I_{sub} is a new parameter that we have introduced).

The reward in our implementation is calculated by simply comparing the predicted value by XCS and the correspondence input bit, i.e. if the XCS has predicted the bit correctly, it would receive a positive reward and otherwise a zero reward. There could also be another reward assignment procedure which is described in section 6.

6. Some notes

XCS could be used in an adaptive compression too. Adaptive compression means that the model is generated on the fly [1]. In our context, this means that the classifiers of XCS should be generated in an online manner. Adaptive compressors benefit from the identical model update algorithms and initial models. XCS algorithm in our context plays the role of both "model initializer" and "model updater". The "model initializer" should be an XCS with or without predefined classifiers. The "model updater" should be a common XCS algorithm in which the random choices are handled so that they are not random anymore (to make the process deterministic which is a necessity of adaptive algorithms).

As mentioned before, a mechanism for automatic control over value of $P_{\#}$ exists that we will describe it here. It has previously been reported [11] that XCS can work with large $P_{\#}$ provided only that some minor adjustments are made. It is also claimed that larger $P_{\#}$ values are better for generalization and they carry the promise of more compact populations. However, our experiments with $P_{\#}$ near one (a big probability of "don't care" symbol occurrence) resulted in random or greedy behavior which means that the system was not able to generalize the concepts and over fits on the last pieces of data given to it.

Conversely, setting $P_{\#}$ to a small value (near 0.0) causes delayed convergence if population size is allowed to be large enough. But the process would not converge if population size were not allowed to be large enough.

Our mechanism for automatic control over $P_{\#}$ is based on the fact that the best value of $P_{\#}$ is near to average number of "don't care" symbols in successful classifiers (classifiers with high fitness). This fact leads us to start with a small $P_{\#}$ and update it, time by time, to the weighted average over successful classifiers.

The mechanism used for reward calculation was a single step reward mechanism, but we already know that XCS learning algorithm supports delayed reward mechanisms (the reward belongs to all actions done till now). For example, here, the reward could be given every eight steps based on the decrease of entropy between input and output streams.

7. Result

"Table 1" shows the experimental results of applying the system on the train set (the set of files that is used for XCS train process). Test and train sets are collected randomly out of more than 200 mega bytes of files on computers in the university laboratory. The random selection of files is unbiased, that is the train

set is not assumed to contain any special information. Thus the trained set of classifiers could be assumed to be general.

"Table 2" shows the experimental results of applying the system on the test set. The first column of both tables shows the file type which is used. The last two rows on each table are labeled with 1st and 2nd methods of "color bitmaps" where the "1st method" indicates the method in which colors are separated to 24 bits, while the "2nd method" indicates the method in which we have encountered the bitmaps as ordinary binary files.

File Type	**Train set size**	**Comp. rate by Arithmetic coding (bpc)**
Text Files	1.1 MB out of 4.7 MB	2.401
Pascal Sources	4.3 MB out of 14.5 MB	2.234
HTML Sources	6.9 MB out of 45.9 MB	1.982
Monochrome Bitmaps	3.1 MB out of 13.2 MB	2.784
Color Bitmaps (1st method)	22 MB out of 145 MB	3.310
Color Bitmaps (2nd method)	22 MB out of 145 MB	3.781

Table 1: Results on train set. The total of 5000 classifiers are used in all methods.

System is equipped with a 20-bit adaptive arithmetic coder in place of "EE box". The results in the third column are the bpc (bit per character) of output.

"Table 3" shows the result of comparing proposed method against some well known compressors such as "compress", "gzip" and "pkzip". "Average Compression Rate" is a weighted average over the compression rate of each file in test set.

File Type	Test set size	Comp. rate by Arithmetic coding (bpc)
Text Files	0.6 MB out of 4.7 MB	2.731
Pascal Sources	1.4 MB out of 14.5 MB	2.390
HTML Sources	5.0 MB out of 45.9 MB	2.192
Monochrome Bitmaps	0.9 MB out of 13.2 MB	2.995
Colored Bitmaps (1st method)	10 MB out of 145 MB	3.453
Colored Bitmaps (2nd method)	10 MB out of 145 MB	3.896

Table. 2: Results on test set (based on XCS classifiers with 5000 classification rules).

Compressor	Average Compression Rate
compress	3.610
gzip	3.280
pkzip	3.807
XCS Compressor	3.302

Table. 3: Comparison with other compressor.

References

[1] M. Nelson, *The Data Compression Book*, T M&T, New York, NY, 1995.

[2] C.E. Shannon, A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, 27, 1948.

[3] TA.P. Berg, W.B. Mikhael, Survey of techniques for lossless compression of signals. *Midwest Symposium on Circuits and Systems*, 2: 943-946, 1994.

[4] T. R. A. Stine, Model Selection Using Information Theory and MDL Principle. *Sociologist Methods & Research*, 33(2): 230-260, 2004.

[5] C. Lee Giles, Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Inference. *Machine Learning*, 44: 161-183, 2001.

[6] M. V. Butz, S. W. Wilson, An Algorithmic Description of XCS. *IlliGAL Report No 2000017*, 2000.

[7] T.M. V. Mahoney, Text Compression as a Test for Artificial Intelligence. *AAAI*, 970T, 1999.

[8] Blue Book, Lossless Data Compression. *CCSDS 121.0-B-1*, 1997.

[9] T.P. Domingos, M. Pazzani, TOn the Optimality of the Simple Bayesian Classifier under Zero-One LossT. *Machine Learning*, 29: 103-130, 1997.

[10] TM. V. Mahoney, Fast Text Compression with Neural Networks. *Proceedings of the American Association of Artificial Intelligence, FLAIRS conference,* pp. 230-234, 2000.

[11] M. V. Butz, D. E. Goldberg, K. Tharakunnel, Analysis and improvement of fitness exploitation

in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11: 239-277 , 2003.
[12] S. Wilson, Classifier fitness based on accuracy. *Evolutionary Computation*, 3: 149-175, 1995.
[13] T. M. Cover, J.A. Thomas, *Elements of information theory*, Wiley, New York, 1991.
[14] R. D. Bulos, et. al., A data mining approach in opponent modeling. *proceedings of Australian Conference on Artificial Intelligence*, Springer, pp. 993-996, 2005.
[15] S. W. Wilson, State of XCS Classifier System Research. *Lecture Notes in Computer Science*, 1813: 63-81, 2000.
[16] S. M. Weiss, C. A. Kulikowski, *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*, Morgan Kaufmann Publishers Inc., 1991.
[17] T. M. Mitchell, *Machine Learning*, McGraw-Hill Higher Education, 1997.