SANDIA REPORT
SAND2021-11185
Unclassified Unlimited Release
Printed Sept 2021

Sandia National Laboratories

NGram PPM: Compression Analytics without Compression

Travis Bauer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

NGram PPM: Compression Analytics without Compression

Travis Bauer
5554 Computational Decision Science
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0621
tlbauer@sandia.gov

SAND2021-11185

ABSTRACT

Arithmetic Coding (AC) using Prediction by Partial Matching (PPM) is a compression algorithm that can be used as a machine learning algorithm. This paper describes a new algorithm, NGram PPM. NGram PPM has all the predictive power of AC/PPM, but at a fraction of the computational cost. Unlike compression-based analytics, it is also amenable to a vector space interpretation, which creates the ability for integration with other traditional machine learning algorithms. AC/PPM is reviewed, including its application to machine learning. Then NGram PPM is described and test results are presented, comparing them to AC/PPM.

ACKNOWLEDGMENT

I'd like to acknowledge Christina Ting and Andrew Fisher who were partners and developers in our early exploration of PPM. I'd also like to acknowledge Nicole Murchison and Lisa Gribble, colleagues on recent text analysis work in the context of which this algorithm was developed. Finally, I'd like to thank Derek Tucker, Christina Ting, and Nicole Murchison for careful reading and feedback on the text of this paper.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

AC Arithmetic Coding

PPM Prediction by Partial Matching

1. INTRODUCTION

This paper describes a new algorithm, NGram PPM. NGram PPM is an algorithm that assigns a score to a sequence of tokens (generally bytes) based on a comparison of the statistical composition of that sequence to the distribution of tokens of some background training set. These scores can be used in a variety of ways including both rank ordering items and supervised learning problems.

Normalized Compression Distance (NCD) [4, 14] is a method to produce a score that represents the similarity of two items. This ability can be leveraged to accomplish various machine learning tasks. NCD takes advantage of the fact that compression algorithms can be treated as an approximation of Kolmogorov complexity [15]. In so doing, they compute similarity as shared information. Arithmetic Coding with Prediction by Partial Matching (AC/PPM) [18] is a compression algorithm often used for NCD.

There are two key problems with compression-based analytics and with AC/PPM in particular. One problem is that AC/PPM is slow. This has limited use for larger datasets and real-time applications. A second problem is explainability. While an advantage of compression-based algorithms is that they do not require the explicit creation of feature spaces, the downside is that the algorithm collapses a large number of decisions into a single number based on the ratio of the size of the original items to their size when compressed. There is no straightforward way to explain a particular score.

NGram PPM solves both of these problems. It has all of the advantages of AC/PPM-based analytics but is significantly faster. The algorithm is also more explainable and can be used to generate feature spaces for integration with other algorithms. It isolates the components of AC/PPM that contribute to the compressability of an individual data item without actually performing the compression. In our tests, it is approximately 10 times faster for model creation and at least 5 times faster for model usage while returning almost exactly the same results as AC/PPM. It also expresses the scores in terms of weighted ngrams. This enables the creation of features spaces, opening a wide range of possibilities for hybrid compression-based machine learning algorithms with traditional vector-based methods.

This paper describes how PPM is utilized by arithmetic coding to accomplish compression, develops that description into an alternative algorithm that produces a score that correlates with AC/PPM, shows how that can be used as an efficient alternative to AC/PPM on real and synthetic data, and describes some modifications and future developments.

2. USE OF AC/PPM AS A MACHINE LEARNING ALGORITHM

Compression-based machine learning algorithms have been shown to perform well on a wide variety of tasks, including the identification of gene function in biomedical papers [16] and other aspects of computational biology [11], authorship attribution [19, 21], and topic detection [17]. The utility of these algorithms have also been shown on problems as broad as disinformation detection [25], seismic data and binary file analysis [10], and network traffic protocol anomaly

detection [24]. It has also been applied to the problem of hierarchical music clustering [7]. PPM-based compression analytics have been shown to out perform other compression algorithms in software plagiarism detection [5]. Given this broad applicability of compression-based clustering, it has been suggested as a general clustering algorithm [8]. Even just these distances alone have utility for arranging documents in a way that is visually useful to users [23].

In its most basic form, NCD measures the shared information between two individual items by compressing them individually, compressing an object composed of the individual items concatenated, and then comparing the resulting compression. If two items are identical and the compressor (whether based on AC/PPM or some other algorithm) is accomplishing a sufficiently accurate approximation of the Kolmogorov compexity of the two items, the size of the compressed form of an individual item should be almost the same as the size of the compressed form of the two concatenated items compressed. For items that are not identical, the size of the compressed form of the concatenated object should reflect the similarity of the statistical distributions of the bytes in each individual item. This is because a compressor learns the distribution from the first item and then can use that to efficiently compress the second one. By examining the items byte-by-byte, algorithms like this run the risk of losing information because the semantics are often at a higher level (e.g., words reveal the meaning of a text, not individual letters). However, it has been shown that AC/PPM achieves the same performance when it uses this byte-by-byte configuration as when documents are reconfigured so it operates on a word level [3].

As a byproduct of compression, PPM produces a model that encodes the probability distribution of various subsequences in the data, using that for arithmetic coding. By keeping this model and reapplying it, actual concatenation is no longer necessary. The same effect achieved by concatenating and compressing is effectively accomplished by compressing an item and then using that model to compress some other item. This also means that rather than creating a model based on a single item, an entire training set can be compressed to generate a model that can then be used to evaluate new items. At this point, one is not simply comparing two items, but comparing some item to an entire training set.

Compressing a new item with respect to an existing model yields a compressed object. The compression ratio of this object (the ratio of the number of compressed bytes to the bytes in the original item) can be used as a score that reflects how well this new item matches the distribution of the training data, in other words, how similar this new item is to the training data. This can be used for a variety of analytical tasks, including rank ordering a set of items according to a model based on training data, or classifying items according their best match to the set of models. For rank ordering, the compression ratio of each item with respect to some model is computed after which the items can be ordered from the lowest compression ratio to the highest. The item with the lowest compression ratio best matches the training data used to create the model and the item with the highest compression ratio is the poorest match. For classification, the compression ratio of an item can be computed using each model and a label can be assigned based on which model resulted in the best (lowest) compression ratio.

In each of these applications of compression for machine learning, the basic operation is the computation of a compression ratio that serves as a score for the similarity between the item under consideration and the composition of the data in the model. The lower the score, the better

the match. The actual compressed form of the object is irrelevant. Even the precise compression ratio itself is irrelevant. It is the comparison of the scores to one another that is used for classification and rank ordering. As will be described below, NGram PPM computes a score that is neither a compression ratio nor a compressed object, but the scores are such that almost identical results can be computed for machine learning purposes but with more explainability and better computational efficiency.

3. PREDICTION BY PARTIAL MATCHING

3.1. Overview of Arithmetic Coding

Arithmetic coding encodes a sequence of tokens into an infinite precision real valued number. Let K be a sequence of t tokens, $k_0, k_1, ..., k_t$. Each token can take on one of an enumerated set of distinct v values, $b_0, b_1, ..., b_v$, denoted B. When each k is encoded, the relevant range is split into $|B|+1$ intervals, each corresponding to a potential value plus a range for an escape character (ESC), described in more detail below. The size of each interval is proportional to the probability of its corresponding value. Imagine that $B = \{w, x, y\}$ and that w occurs 50% of the time, x occurs 25% of the time, and y occurs 25% of the time. In this case, w would be allocated half of the available range and the other two would be allocated a quarter each. In this scenario, all token sequences starting with w would be encoded between 0 and 0.5. This new range would then be used to encode the next token with sub ranges being proportionally allocated. Thus all token sequences starting with ww would be encoded between 0 and 0.5, the second w splitting the range 0 to 0.5 in half again.

Because computers do not actually have infinite precision real valued numbers, integer ranges are used at each step. Rather than splitting a real valued range between 0 and 1, range coding works by splitting a large integer range roughly proportionally. This same range is used at each step based on the relative probability of each token. While in principle a location in a real valued number is being computed, in practice neither the actual intervals used up to that point nor the specific range ends up being important. The probabilities and a full range of integers are used at each step.

It is also the case that at each token, only that token is encoded. The extent to which that token is compressed is related solely to its probability. For compression, the probabilities of the other tokens are only needed so that the specific start of the interval for that token's value can be specified. This is necessary not to determine how well the token is going to be encoded, only to determine when and which bits to emit for later decompression.

To the extent that the actual probabilities are estimated for each value of B, arithmetic coding results in data being compressed. They do this by using fewer bits to encode the high probability items. The trick is estimating the probabilities correctly. One way to do this is by using PPM. For the purposes of using AC/PPM for machine learning, we do not care about decompression or about the specific bit stream that is emitted. We only care about how well that token is going to be compressed. We can get this value from its probability alone.

3.2. Overview of PPM

At the point where each token is encoded, arithmetic coding assesses the probability of that token's value. There are a variety of ways to estimate probabilities. It would be computationally well defined to simply use the same probability of every potential value, assigning $\frac{1}{|B|+1}$ to each token plus an interval for dealing with situations where a value is seen in a context where it did not occur in the training data (called "ESC"). Unless the individual values were equally probable, however, this would not lead to optimal compression. Another method would be to assign probabilities based on their occurrence in real data. For example, If B was the set of latin letters (a-z), the letter u could be assigned a probability of 2.8% [1].

But it is possible to create better probability estimates given the context within which each item occurs. Remember that arithmetic coding works over a sequence of tokens. When considering the probability of any given value, it is possible to take into account the previous tokens in the sequence. Although a u may have a probability of 2.8% overall in English, the probability of a u given that the previous token was a q is quite high. So when encoding a u after a q, a different probability can be used than in a different context. In fact PPM has been used to study the English langauge [22].

PPM is a way of accumulating and issuing those conditional probabilities. While compressing data, the probabilities are accumulated while compressing. However, when used for model creation as described above, only the accumulated probabilities are relevant, not the actual compressed bits of the training data. To create a model, we do not need to emit any compressed bits. It is only necessary to accumulate the probabilities. When considering a new item, the model is fixed (no more probabilities are accumulated) and the accumulated probabilities are referenced.

PPM takes a parameter d (depth) indicating the maximum size of the context that it considers for calculating conditional probabilities. It keeps $d+1$ models, one for each context less than d and a base model of simple probabilities with no context. Each model, C_j records the probability of each value of a token in all contexts of size j as shown in Equation 1. For example, given a sequence abc and a depth of two, after encoding a and b, c would be encoded using the probability of c given the context of ab.

$$C_j(k_i) = p(k_i|k_{i-j}, k_{i-j+1}, ...k_{i-1}) \tag{1}$$

This works if, for every token encoded, that token has already been seen in a context of size d. In practice, this is often not the case. In this situation, an *escape* character (ESC) is encoded. ESC is a value that cannot occur in the input data. It exists solely to deal with a situation where a token has not been seen in a context being considered. AC encodes ESC the same way it encodes any character and then it checks another context C_{j-1}. In the scenario just described, if c has never before occurred preceded by ab (which it would know by checking C_2, AC would emit an ESC and then try to encode c by checking C_1, corresponding to the context b. If c has also never occurred preceded by a b, it would fall back to C_0 which would be a baseline that encodes a

[1] https://en.wikipedia.org/wiki/Letter_frequency

probability for every possible token. ESC tokens are encoded the same way that normal tokens are. However, that leaves a problem of determining the probability of an ESC token. A approach described by Cleary and Witten [9] and used in NGramPPM is to treat each context as containing a single occurrence of an ESC character.

As mentioned earlier, this algorithm can be applied for machine learning. To use AC/PPM for machine learning, a model is created from a training set by passing it through AC using PPM, accumulating counts of the numbers of times each token has been seen in each context. For each C_j that AC visits in the course of encoding some k_i, regardless of whether it results in an ESC character being issued, it increments a counter for k_i in C_i corresponding to the current context.

4. NGRAM PPM

4.1. Explanation

The use of AC/PPM as an anlytical algorithm requires both that the underlying statistics are collected (using PPM) and that the item being analyzed is compressed (using arithmetic coding). NGram PPM collects the statistics needed for AC/PPM, but then uses those statistics directly without performing arithmetic coding. Instead, it computes a score that reflects how well the item would have been compressed had arithmetic coding been conducted. This results in a smaller computation time.

PPM determines probabilities by counting the number of times each token occurs in each context, tracking the number of times each context occurs, accounting for ESC tokens. The analytic power of PPM-based base analytics is rooted in the ability to compress some new item. To perform full compression using AC/PPM, we need to find a specific starting point for each token being compressed which leads to additional computation during compression. But the compression ratio of each token is function solely of the probability of each token alone and is independent of the compression ratio of any other token. Therefore, we can define a score for each K by simply looking at the accumulated probabilities of each of the underlying tokens, taking into account ESC tokens. This reduces the number of computations needed in NGram PPM compared to using full AC/PPM.

We define a function σ that computes a score for any k_i indicating how well k_i will get compressed [2]. The overall score for a token sequence is simply the sum of those scores. This is defined in Equation 2. The rest of this section will describe how σ is computed.

$$ngramppm(K) = \sum_{i=0}^{t} \sigma(k_i) \tag{2}$$

The function σ will be a combination of the probability of a character occurring given its context and any ESC tokens that need to emitted as part of the compression. As described above, AC/PPM looks at the longest context first, encoding an ESC if the token does not occur in that context and

[2]Implied in the rest of this discussion is that anytime k_i is referenced that the previous d tokens from K are available.

falling back to a shorter context. We can think of this as visiting each context length for every token, computing a context score based on whether an ESC would have been issued or it was the appropriate context for returning a probability. In order to define the component score, we need to compute the count that would be contributed for a given content. Let $\gamma_j(k_i)$ be the number of times $k_{i-j}, k_{i-j+1}, ..., k_i$ has occurred in the training data. Then the count (ε) of a token in a given context taking into account escapes would be defined as in Equation 3. A one is returned if an escape character would have been issued. This occurs if no longer context would have contained that character. If the context contains the character in the longest possible context, it is the count itself that is returned. Otherwise the context is ignored so the count returned would be zero.

$$\varepsilon_j(k_i) = \begin{cases} 1 & \text{if } \gamma_j(k_i) = 0 \wedge \text{for all } j < n \leq d \ \gamma_n(k_i) = 0 \\ \gamma_j(k_i) & \text{if } \gamma_j(k_i) > 0 \wedge \text{for all } j < n \leq d \ \gamma_n(k_i) = 0 \\ 0 & \text{if for some } j < n \leq d \ \gamma_n(k_i) > 0 \end{cases} \tag{3}$$

To convert this to a probability, we then need to know the number of times a given context has been seen. We add one to account for ESC tokens, but can otherwise reuse γ, see Equation 4. The reuse of γ is significant for implementing the algorithm. The number of times a context occurs is the same as the number of times the last character has occurred in a context shorter by one token. For the base case where there is no context, the number of potential tokens plus the tokens actually seen in the training data is used.

$$\alpha_j(k_i) = \begin{cases} |B| + \gamma_0(k_i) & \text{if } j = 0 \\ \gamma_{j-1}(k_{i-1}) + 1 & \text{otherwise} \end{cases} \tag{4}$$

We can now define the score contributed by each context. This is simply the count provided by the context divided by the number of times the context has been seen, accounting for ESC, see Equation 5 [3]. Remember that a higher probability leads to smaller compression ratio. Computing one minus the probability results in a higher score for low probability tokens given their context.

$$\rho_j(k_i) = \begin{cases} 0 & \text{if } \varepsilon_j(k_i) = 0 \\ 1 - \frac{\varepsilon_j(k_i)}{\alpha_j(k_i)} & \text{else} \end{cases} \tag{5}$$

The score for an individual token in a token stream is then simply the sum of the scores among all the contexts, as defined in Equation 6.

$$\sigma(k_i) = \sum_{j=0}^{d} \rho_j(k_i) \tag{6}$$

These scores will not be the same as the compression ratios that come from a full implementation of AC using PPM. However, the relative ordering of the scores should be almost the same. The

[3] See the appendix.

key difference is that there is some rounding that takes place in AC when encoding into a range of integers rather than into actual real values. The rounding does not take place in the algorithm described here. It is also the case that generally a baseline token probability is used in PPM. As defined here, each potential value is given a count of 1 to which is added the actual value counts. This creates a more accurate baseline probability for tokens seen in a completely new context. So where this algorithm differs from PPM, it should more accurately reflect the underlying probabilities and stay truer to the underlying intention of the algorithm because it avoids rounding errors.

4.2. Implementation and Testing

NGram PPM is expressed in terms of conditional probabilities, but the entire algorithm is built on γ. Because of this, the entire algorithm can be implemented in terms of counting ngrams. A reference implementation was written in Python according to the specification above. Both a pure Python implementation was tested along with a Cython version of the same code (with modest adaptation to Cython).

Three studies were conducted to show that NGram PPM produces comparable results to AC/PPM. For these studies, the NGram PPM implementation was compared to a form of PPM compression analytics based on a pure Python implementation of AC/PPM [4].

The memory used by PPM grows non linearly with respect to context size (d). In Moffat's paper [18], they experiment with contexts sizes between one and five. A context size of zero would be the equivalent of examining the probability of individual items without context. For retrieval problems in English language text, a context size of three or four works well. The experiments described here use a context size of four.

A full description of the algorithmic complexity of AC/PPM is covered in other literature (e.g. [20]). NGram PPM (as used in these experiments) computes the same statistics as AC/PPM, but then it skips the compression step. Because of this, the computational complexity of the algorithm is of the same order of magnitude as the AC/PPM, but as we will show, it is faster for the datasets studied and because it is able to simply skip computations that AC/PPM performs. This would be true for any dataset. The memory requirements for the tests shown here are similar to AC/PPM because the same statistics are collected. However, as will be discussed in Section 5.2, these statistics can be pruned in a way that would lead to better memory efficiency.

These tests were run on a 2.9 GHz 6-Core Intel Core i9 Macbook Pro with 32GB of RAM.

4.2.1. Testing Ranking Using Synthetic Data

A basic way to use these algorithms for machine learning is to create a model from some training data and then produce a score from a test item. The scores serve as a kind of distance measure, with a lower score for either algorithm (compressing to a smaller number of bytes or generating a

[4] https://github.com/nayuki/Reference-arithmetic-coding

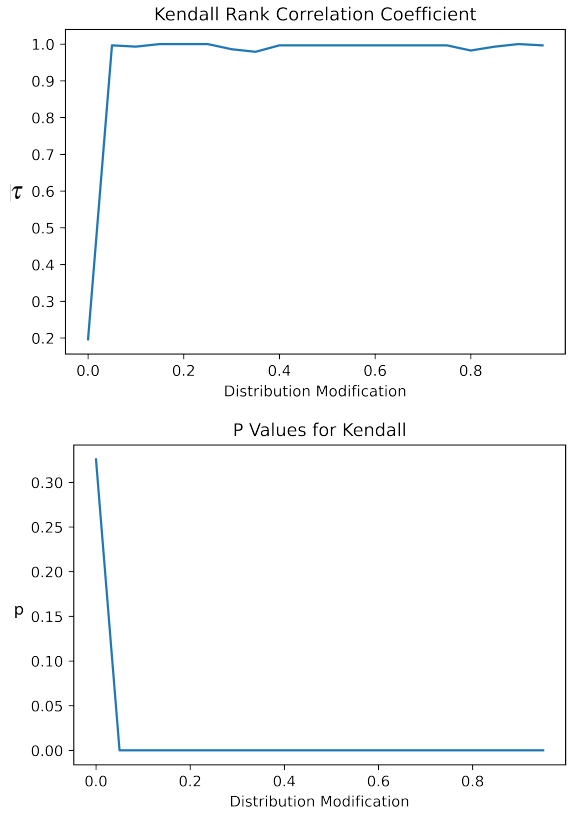

Figure 4-1 Ordering study showing that AC/PPM and NGram PPM order data in the same way

lower ngramppm score) indicating that the test item's composition matches that of the training data. A set of items can be rank ordered by generating a score for each item against a model and ordering according to the scores returned. If ordered from smallest to largest score, the items would be ordered according to how well they match the training data, from the best to the worst match.

The first test comparing AC/PPM and NGram PPM consisted of created a training set generated from a single distribution and then creating a test data set where each item was systematically different from the training data in some predictable way. If each algorithm orders these test items in the same way, we can have confidence that they are accomplishing an equivalent computation, at least for our purposes.

This test was conducted with synthetic data to compare the *ngramppm* score from Equation 2 with the compression ratio achieved using AC/PPM. Two hundred randomly generated training byte arrays of 5000 bytes each were used to create models. Each array was created by drawing from an alphabet $|B| = 10$, each having a different probability (0.0, 0.022, 0.044, 0.067, 0.089, 0.111, 0.133, 0.156, 0.178, 0.2). Then a series of test datasets were created. For each test dataset, twenty different test items were created. For each item in a test dataset, the probability of the most likely byte in the training dataset (probability of 0.2) was decreased by some value and the probability of the least likely byte (0.0 and thus not in the training set) was increased by the

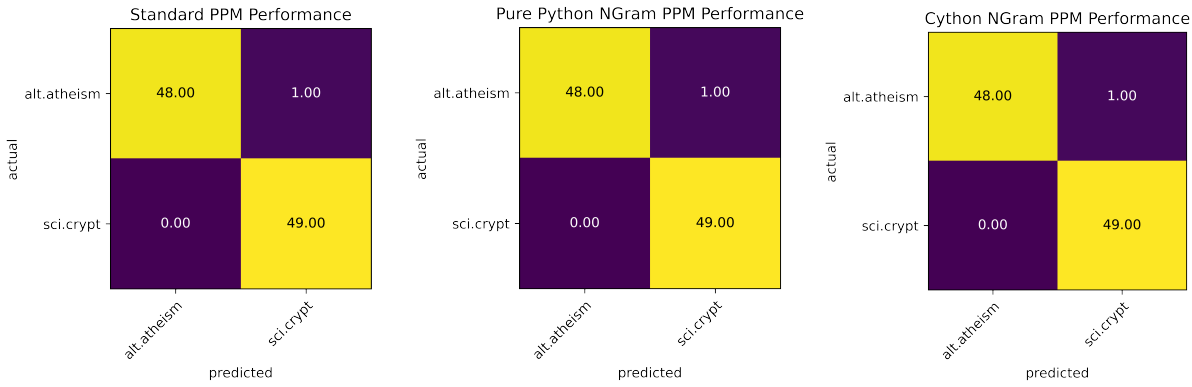

Figure 4-2 Identical classification performance of standard PPM and both implementations of NGram PPM for classifying by topic

corresponding amount. This means that there was a natural ordering for each item in a test dataset according to how well it matched the training dataset. Each test dataset used a different range for these modifications, 0 indicating that the distribution in the test dataset matched the training dataset exactly and 1 meaning that the probability of the least and most probable bytes in the training sets were fully reversed. The Kendall rank correlation coefficient [12] was then computed between the ordering of the test item according to AC/PPM and NGram PPM. In this measure, τ is a measure of how well the ordering match and the p values indicate the results of a statistical test where the null hypothesis is that τ is 0. Figure 4-1 shows how τ and the p values changed as the items were more differentiated from each other.

When all the test data closely match the training data, the ordering is arbitrary and small differences in the algorithm results in a low correlation between the two algorithms. But as soon as the test items were drawn from different distributions, the correlation rapidly approached one and the p values were near zero. This shows that the compression ratios achieved from PPM correspond to the NGram PPM scores except in cases where the items are identical in their composition.

4.2.2. Testing Classification - Topic Classification

To compare the speed of the implementations, a classification experiment was conducted on documents in the classic "20 newsgroup dataset." This dataset is available from a variety of sources including in scikit-learn [5]. Specifically, 195 documents each from sci.crypt and alt.atheism were used for classification. The documents were split into a stratified crossfold four ways and one of the folds was testing and timed. Figure 4-2 shows a confusion matrix with the results. The confusion matrices demonstrate identical performance. Given the combination of the strong results from the ordering test and identical classification performance we conclude that NGram PPM captures the value of AC/PPM for machine learning purposes.

[5] `https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html`

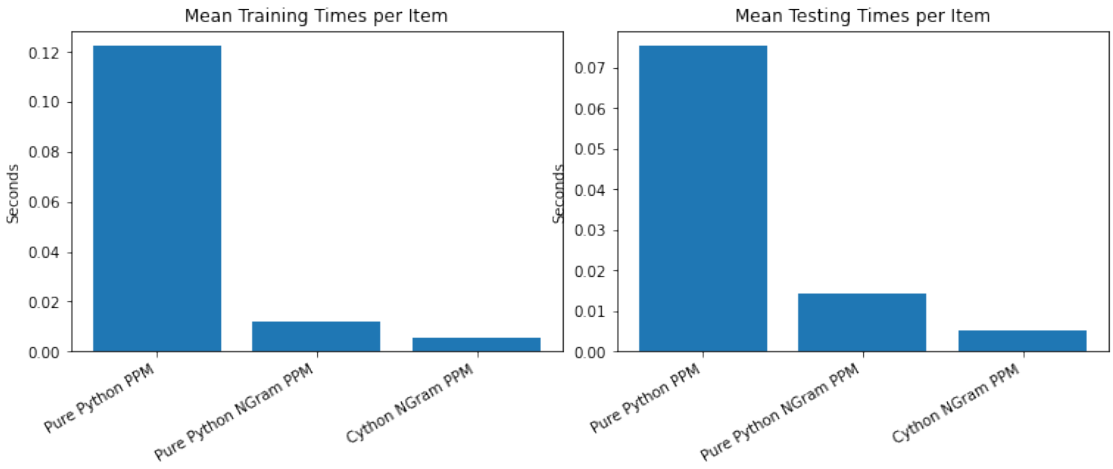

Figure 4-3 Comparison of timing for training and testing (seconds per item)

It is possible for the results between the two algorithms to vary slightly. This can happen when an item is an almost equal match between the two models. This is for two reasons. The first is that PPM/AC ultimately has to effectively round the probabilities to fit them within an integer range. The second is that while normal PPM has a baseline probability for tokens with no contexts, NGram PPM can accumulate the actual base probabilities of the individual tokens. Figure 4-2 shows that the classification performance between the two algorithms was identical in the test run. Other experimentation has shown that there can be minor differences between the two algorithms. As described above, in the cases where the two differ NGram PPM would be preferred because it is keeping more information about the underlying data.

Figure 4-3 compares the mean time it took per item for training and testing. The relative time does, of course, depend on the nature of the implementation. Neither implementation is optimized for speed. An earlier form of NGram PPM was 26 times faster for training and 12 times faster for testing, but this came at the expense of being inefficient in memory usage and would not scale to larger datasets. The form shown here implemented a context data structure similar the PPM's implementation for memory efficiency. It is approximately 10 times faster for training and 5 times faster for testing. The variance for NGram PPM was smaller than the variance for AC/PPM.

This suggests that NGram PPM captures the predictive power of AC/PPM but in a much more efficient algorithm. The Cython version of NGram PPM is almost twice as fast as the pure Python implementation for training and almost 2.5 times faster for testing.

4.2.3. Testing Classification - Detecting Toxic Tweets

As a second test to compare the classification performance of the two algorithms, a dataset of toxic and non-toxic tweets was downloaded [6] from Kaggle. From the dataset, 1000 toxic and 1000 non-toxic tweets were selected at random. The results of training on three fourths of the data and testing on one fourth are showin in Figure 4-4. Any substantial difference between the two

[6] https://www.kaggle.com/ashwiniyer176/toxic-tweets-dataset

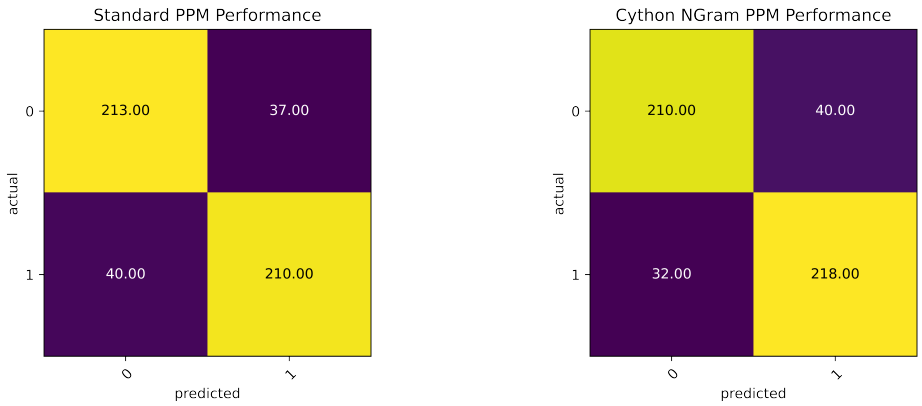

Figure 4-4 Near Identical Performance of standard PPM and NGram PPM on identifying Toxic Tweets

algorithms could easily lead to significantly different results, especially given the small size of individual tweets.

5. FURTHER IMPROVEMENTS

This form of the algorithm is not only faster, but it is expressed in a way that lends itself to further expansion and improvements. A few of these ideas are outlined here.

5.1. Ignoring Escapes

Other research [25] has demonstrated that AC/PPM suffers from a class imbalance problem and that these problems can be mitigated by altering AC/PPM to not emit escapes. Implementing this strategy in NGram PPM is trivial. The first condition resulting in a 1 being returned in Equation 3 is the algorithm emitting an escape. Changing that condition to 0 would have the same effect. This work also suggests that changing Equation 4 so that it does not add 1 could be useful. Especially for poorly represented contexts, adding a 1 to the context count drastically changes the probability of the other tokens in the same context, resulting in a significantly different token score (σ, see Equation 6).

5.2. Controlling for Infrequent Contexts through Thresholds

Contexts that occur infrequently in the training data can have an unusually large effect on scores. Given Ziff's law, [27] contexts that are individually infrequent make up a significant portion of the actual data that gets encoded. Although infrequent items contribute less to the overall scores, there are more of them and explicitly controlling for them can be useful. This is often done in statistical text analysis, where all terms occurring less than some threshold might be dropped from further analysis. This can be accomplished in this algorithm by changing the threshold in the

second two conditions of Equation 3. By requiring a larger count, say $\gamma_j(k_i) > 5$, a token will need to have been in a particular context at least 6 times before it would contribute to the score. Removing low frequency counts from the counter dictionary (γ) altogether might also be useful.

5.3. Controlling for Infrequent Contexts through Smoothing

The context score in Equation 5 computes a probability. At that point it loses the information about how many times that context has been seen. Probabilities based on small sample sizes can have a disproportionate effect in computations such as pointwise mutual information (PMI) [6]. It has been shown that for PMI computations, smoothing can mitigate the effects of low probability items [13]. Something analogous could be done with NGram PPM that could not be done for full AC/PPM to help control for these effects.

5.4. Analogy to and Creation of Feature Spaces

Probably the most significant implication of this algorithm is that it shows how to express the predictive power of AC/PPM in terms of scores over ngrams, which enables us to interpret the scores as a feature space. For a model of depth d, these sequences in some K are essentially decomposed into d-shingles. Each distinct shingle has a score defined by Equation 6 that is invariant with respect to where in K the shingle occurs. For NGram PPM, these scores are simply added together to reflect a compression ratio (Equation 2). Let S_K be an enumerated set of distinct d shingles in K and s_i be a shingle that can be treated as a short token sequence (so that s_{id} refers to the last token of shingle i) and β is the number of times each shingle s_i occurs in the testing document, Equation 2 can be expressed as:

$$ngramppm(K) = \sum_{i=0}^{|S_k|} \beta(s_i)\sigma(s_{id}) \tag{7}$$

This formulation can be leveraged into creating a feature space. When converting a document corpus into a matrix for text analysis, each term in each document is given a score. This score is often a product of a local score that reflects the significance of the term in some document (a local weight) and the significance of the term in the corpus as a whole (the global weight). Equation 7 can be thought of treating β as a local weight and σ as a global weight, computing a product [1]. Rather than simply adding all the products together, they can be kept separate and be treated as a feature space. Because our formulation of NGram PPM gives lower scores to items that contribute the most to compression, σ^{-1} will be used as a global weight. Insight from traditional statistical analysis and conventional ML can now be applied.

Figure 5-1 shows the results of applying this methodology to the newsgroups datasets described previously. An NGram PPM model was created for each category. Vectors were then created by finding all the ngrams that occurred in more than five documents. There were 16,724. A vector space was then created with one vector per document. The model corresponding to the document's newsgroup was used for the global weights. Then t-SNE [26] was used to embed each

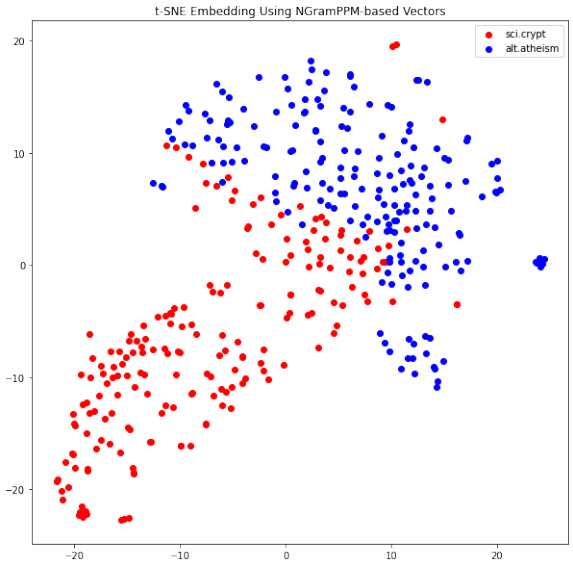

Figure 5-1 t-SNE embedding of NGram PPM vector space

post into a two dimensional space. As can be seen, there is a clear separation among the documents from vectors created using this methodology.

AC/PPM has outperformed other vector-based techniques that require more feature engineering on problems such as authorship detection. But vector-based techniques have other advantages and a large body of research. NGram PPM brings these two different families of algorithms under a common framework.

5.4.1. Alternative Weightings

Rather than using simple shingle counts, alternatives to β can be used. It is common in statistical text analysis to use the *log* of the count rather than the count directly.

5.4.2. Explainability

The individual components of the summation can be used for explainability. The components with the smallest product are contributing the least to the total sum and most to the compressibility. The question "why does this sequence compress better with model A than with model B," can be answered by extracting frequent ngrams with low scores in model A and high scores in model B.

A more general question about the difference between two categories can also be computed. For situations where there are two known categories, the global weights can take into account those categories. In particular, if the goal is to classify one of the categories, the global weights can be computed by subtracting the weight of the target category from the weight of the other categories.

NGram	sci.crypt	alt.atheism
\n___	64	54
Marc	16	1
cmay	16	0
\t\t\t\t	13	2
_fir	59	32

Table 5-1 Top NGrams for a specific sci.crypt document

NGram	sci.crypt	alt.atheism
\n>_>	2	8
___*	1	7
3Apr	25	60
du>_	25	36
@po.	0	8

Table 5-2 Top NGrams for a specific alt.atheism document

This would result in a feature space where individual weights could be used to understand what ngrams best characterize each category.

To illustrate this idea of explainability, two sample documents from the newsgroup dataset were analyzed. For each token in each post, the difference between the token score (σ) in each model was computed. Specifically, σ from the sci.crypt model was subtracted from the alt.atheism score and multiplied by the number of times the ngram occured in the document. Because small scores means higher probability (and better compression), a small score indicates that the term is more indicative of alt.atheism. A large score indicates that the score is more indicative of sci.crypt.

Table 5-1 shows the top terms for a sample sci.crypt document. The table shows how many documents in each category contained the corresponding ngram. For the sci.crypt document, the top scoring ngrams were ones that occurred more often in the other sci.crypt documents. Table 5-2 shows the same table, but for a sample alt.atheism document. What these values mean is that if regular PPM compression was applied to each of these documents, these are the ngrams that would have compressed the best, contributing the most to the document compressing well. However, with NGram PPM, these computations can be performed without the complexity of actually compressing the documents.

5.4.3. Parallelization

The core of the algorithm is counting shingles. This can be done in parallel. For situations such as crossfold validation or "leave-one-out" protocols, the creation of models can also be made significantly more efficient by computing shingles on a per-document basis and then creating new models by adding different subsets of the individual document shingle counts together. The

computation of individual document scores can also be parallelized, although at the level of an individual document, it is probably not necessary.

5.4.4. Streaming

This algorithm is also, unlike AC/PPM, amenable to streaming. It only keeps track of counts. These counts could be managed by keeping a fixed size window, decrementing the oldest counts as new ones are added. This makes the algorithm amenable to techniques that operate over sliding windows [10]. It could also be accomplished by using a probabilistic method to delete infrequent ngrams (e.g. [2]). Such methods would minimize the memory impact incurred if utilizing memoized scores.

6. CONCLUSION

This paper has described how AC/PPM works and formalized the utility of AC/PPM for machine learning in a way that results in a new algorithm, NGram PPM. NGram PPM is faster than full compression, yields nearly identical results, and presents an opportunity for future development of new algorithms.

REFERENCES

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[2] Travis Bauer and David B Leake. Word sieve: A method for real-time context extraction. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 30–44. Springer, 2001.

[3] Victoria Bobicev. Comparison of word-based and letter-based text classification. *Recent Advances in Natural Language Processing V, Bulgaria*, pages 76–80, 2007.

[4] Manuel Cebrián, Manuel Alfonseca, Alfonso Ortega, et al. Common pitfalls using the normalized compression distance: What to watch out for in a compressor. *Communications in Information & Systems*, 5(4):367–384, 2005.

[5] Xin Chen, Brent Francia, Ming Li, Brian Mckinnon, and Amit Seker. Shared information and program plagiarism detection. *IEEE Transactions on Information Theory*, 50(7):1545–1551, 2004.

[6] Kenneth Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[7] Rudi Cilibrasi, Paul Vitányi, and Ronald de Wolf. Algorithmic clustering of music based on string compression. *Computer Music Journal*, 28(4):49–67, 2004.

[8] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

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

[10] Richard Field, Tu-Thach Quach, and Christina Ting. Efficient generalized boundary detection using a sliding information distance. *IEEE Transactions on Signal Processing*, 68:6394–6401, 2020.

[11] Raffaele Giancarlo, Davide Scaturro, and Filippo Utro. Textual data compression in computational biology: a synopsis. *Bioinformatics*, 25(13):1575–1586, 2009.

[12] William R Knight. A computer method for calculating kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.

[13] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[14] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004.

[15] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[16] Malika Mahoui, William John Teahan, Arvind Kumar Thirumalaiswamy Sekhar, and Satyasaibabu Chilukuri. Identification of gene function using prediction by partial matching (ppm) language models. In *Proceedings of the 17th ACM conference on information and knowledge management*, pages 779–786, 2008.

[17] Yuval Marton, Ning Wu, and Lisa Hellerstein. On compression-based text classification. In *European Conference on Information Retrieval*, pages 300–314. Springer, 2005.

[18] Alistair Moffat. Implementing the ppm data compression scheme. *IEEE Transactions on communications*, 38(11):1917–1921, 1990.

[19] Anderson Rocha, Walter J Scheirer, Christopher W Forstall, Thiago Cavalcante, Antonio Theophilo, Bingyu Shen, Ariadne RB Carvalho, and Efstathios Stamatatos. Authorship attribution for social media forensics. *IEEE Transactions on Information Forensics and Security*, 12(1):5–33, 2016.

[20] Amir Said. Comparative analysis of arithmetic coding computational complexity. In *Data compression conference*, page 562, 2004.

[21] Efstathios Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology*, 60(3):538–556, 2009.

[22] William J Teahan and John G Cleary. The entropy of english using ppm-based models. In *Proceedings of Data Compression Conference-DCC'96*, pages 53–62. IEEE, 1996.

[23] Guilherme P Telles, Rosane Minghim, and Fernando Vieira Paulovich. Normalized compression distance for visual analysis of document collections. *Computers & Graphics*, 31(3):327–337, 2007.

[24] Christina Ting, Richard Field, Andrew Fisher, and Travis Bauer. Compression analytics for classification and anomaly detection within network communication. *IEEE Transactions on Information Forensics and Security*, 14(5):1366–1376, 2018.

[25] Christina L Ting, Andrew N Fisher, and Travis L Bauer. Compression-based algorithms for deception detection. In *International Conference on Social Informatics*, pages 257–276. Springer, 2017.

[26] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[27] George Kingsley Zipf. Human behavior and the principle of least effort: an introd. to human ecology. 1949.

APPENDIX A. ADDENDUM TO NGRAMPPM

After this work was completed, but before this report was finalized, the algorithm was modified. Rather than using the probability directly as a score, it is more accurate to use the log of the probability, which yields the number of bits that a particular token would require when compressed.

Specifically, Equation 5 should be rewritten as Equation 8.

$$\rho_j(k_i) = \begin{cases} 0 & \text{if } \varepsilon_j(k_i) = 0 \\ -\log_2 \frac{\varepsilon_j(k_i)}{\alpha_j(k_i)} & \text{else} \end{cases} \tag{8}$$

In subsequent experimentation, scores that use this equation match the results of NGramPPM almost exactly. This form of the algorithm will be used in future work.

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	L. Martin, LDRD Office	1910	0359
1	Legal Intellectual Property	11500	0161

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.