Techniques for Efficient Learning without Search

Houssam Salem, Pramuditha Suraweera, Geoffrey I. Webb, and Janice R. Boughton

Faculty of Information Technology, Monash University, VIC 3800, Australia
{Houssam.Salem, Pramuditha.Suraweera, Geoff.Webb}@monash.edu

Abstract. Averaged n-Dependence Estimators (AnDE) is a family of learning algorithms that range from low variance coupled with high bias through to high variance coupled with low bias. The asymptotic error of the lowest bias variant is the Bayes optimal. The AnDE family of algorithms have a training time that is linear with respect to the training examples, learn in a single pass through the data, support incremental learning, handle missing values directly and are robust in the face of noise. These characteristics make the algorithms particularly well suited to learning from large data. However, for higher orders of n they are very computationally demanding. This paper presents data structures and algorithms developed to reduce both memory and time for training and classification. These enhancements have enabled the evaluation and comparison of A3DE's effectiveness. The results provide further support for the hypothesis that as the number of training examples increases, decreasing error will be attained by members of the AnDE family with increasing levels of n.

Keywords: naive Bayes, semi-naive Bayes, probabilistic prediction

1 Introduction

The classical classification learning paradigm performs search through a hypothesis space to identify a hypothesis that optimizes some objective function with respect to training data. Averaged n-Dependence Estimators (AnDE) [10] is an approach to learning without search or hypothesis selection, which represents a fundamental alternative to the classical learning paradigm.

The new paradigm gives rise to a family of algorithms, of which, Webb *et. al.* [10] hypothesize, the different members are suited for differing quantities of data. The algorithms range from low variance with high bias through to high variance with low bias. Webb *et. al.* suggest that members with low variance are suited for small datasets whereas members with low bias are suitable for large datasets. They claim that the asymptotic error of the lowest bias variant is Bayes optimal.

The algorithms in the family possess a unique set of features that are suitable for many applications. In particular, they have a training time that is linear with respect to the number of examples and can learn in a single pass through the training data without any need to maintain the training data in memory. Thus, they show great potential for very accurate classification from large data.

Further, they have direct capacity for incremental and anytime [6] learning, are robust in the face of noise and directly handle missing values. Importantly, evaluations have shown that their classification accuracy is competitive with the state-of-the-art in machine learning [10].

AnDE extends the underlying strategy of Averaged One-Dependence Estimators (AODE) [9], which relaxes the Naive Bayes (NB) independence assumption while retaining many of Naive Bayes's desirable computational and theoretical properties. The third member of the AnDE family, A2DE, has been shown to produce strong predictive accuracy over a wide range of data sets [10].

Although evaluations to date support the hypothesis that the predictive accuracy of AnDE increases for larger datasets with higher orders of n, the expected increase in accuracy comes at the cost of increased computational requirements. The current implementations further complicate the matter due to their inefficiencies. Thus, efficient implementation is critical. Except in cases of lower dimensional data, the computational requirements defeat a straightforward extension of Weka's AODE [11] to handle A3DE.

This paper presents data structures and algorithms that reduce both memory and time required for both training and classification. These improvements have enabled us to evaluate the effectiveness of A3DE on large datasets. The results provide further evidence that members of the AnDE family with increasing n are increasingly effective at classifying datasets of increasing size.

The remainder of the paper starts by introducing the AnDE family of algorithms. Section 3 outlines the memory representation developed to reduce memory usage. The enhancements made to reduce testing times are outlined in Section 4. Section 5 presents the results of evaluating the effectiveness of the enhancements. It also compares the effectiveness of A3DE with AnDE members with lower n. Finally, conclusions are outlined.

2 The AnDE Family of Algorithms

The classification problem can be stated as estimating, from a training sample τ of classified objects, the probability $\mathrm{P}(y \mid \mathbf{x})$ that an example $\mathbf{x} = \langle x_1, \ldots, x_a \rangle$ belongs to class y, where x_i is the value of the i^{th} attribute and $y \in c_1, \ldots, c_k$ that are k classes. As $\mathrm{P}(y \mid \mathbf{x}) \propto \mathrm{P}(y, \mathbf{x})$, we only need to estimate the latter.

The naive Bayes (NB) algorithm extrapolates to $\hat{\mathrm{P}}(\mathbf{x}, y)$ from each two dimensional probability estimate $\hat{\mathrm{P}}(x_i \mid y)$, by assuming that attributes are independent given the class. Based on this assumption,

$$\mathrm{P}(\mathbf{x} \mid y) = \prod_{i=1}^{a} \mathrm{P}(x_i \mid y). \tag{1}$$

Hence we classify using

$$\hat{\mathrm{P}}_{\mathrm{NB}}(y, \mathbf{x}) = \hat{\mathrm{P}}(y) \prod_{i=1}^{a} \hat{\mathrm{P}}(x_i \mid y). \tag{2}$$

We assume herein that NB and the other AnDE family members are implemented by compiling at training time a table of observed low-dimensional probabilities. Under this strategy, the complexity of building this model is $O(ta)$, where t is the number of training examples and a the number of attributes. As the model simply stores the frequency of each attribute value for each class after scanning the training examples, the space complexity is $O(kav)$, where k is the number of classes and v is the average number of attribute values. As the classifier only needs to estimate the probability of each class for the attribute values of the test case, the resulting complexity at classification time is $O(ka)$.

Despite the attribute independence assumption, NB delivers relatively accurate results. However, greater accuracy can be achieved if the attribute-independence assumption is relaxed. New algorithms based on NB have been developed, referred to as semi-Naive Bayesian techniques, that achieve greater accuracy by doing this, as real-world problems generally do have relationships among attributes [12].

Of numerous semi-naive Bayesian techniques, SP-TAN [7], Lazy Bayesian Rules (LBR) [13] and AODE [9] are among the most accurate. However, SP-TAN has very high computational complexity at training time and LBR has high computational complexity for classification. Contrastingly, AODE a more efficient algorithm, avoids some of the undesirable properties of those algorithms to achieve comparable results.

2.1 AODE

AODE extends NB's strategy of extrapolating from lower dimensional probabilities to make use of three-dimensional probabilities. It averages across over all of a class of three-dimensional models, which are called super-parent one-dependence estimators (SPODE). Each SPODE relaxes the attribute independence assumption of NB by making all other attributes independent given the class and one privileged attribute, the super-parent x_α.

AODE seeks to use

$$\hat{\mathrm{P}}(y, \mathbf{x}) = \sum_{\alpha=1}^{a} \hat{\mathrm{P}}(y, x_\alpha)\hat{\mathrm{P}}(\mathbf{x} \mid y, x_\alpha)/a. \tag{3}$$

In practice, AODE only uses estimates of probabilities for which relevant examples occur in the data. Hence,

$$\hat{\mathrm{P}}_{\mathrm{AODE}}(y, \mathbf{x}) = \begin{cases} \sum_{\alpha=1}^{a} \delta(x_\alpha)\hat{\mathrm{P}}(y, x_\alpha)\hat{\mathrm{P}}(\mathbf{x} \mid y, x_\alpha) / \sum_{\alpha=1}^{a} \delta(x_\alpha) & : \sum_{\alpha=1}^{a} \delta(x_\alpha) > 0 \\ \hat{\mathrm{P}}_{\mathrm{NB}}(y, \mathbf{x}) & : \text{otherwise} \end{cases} \tag{4}$$

where $\delta(x_\alpha)$ is 1 if attribute-value x_α is present in the data, otherwise 0. In other words, AODE averages over all super-parents whose value occurs in the data, and defaults to NB if there is no such parent.

2.2 A*n*DE

A*n*DE [10] generalises AODE's strategy of search free extrapolation from low-dimensional probabilities to high-dimensional probabilities. The first member of the A*n*DE family (where $n = 0$) is NB, the second member is AODE and the third is A2DE. Investigation into the accuracy of higher dimensional models with different training set sizes shows that a higher degree model might be susceptible to variance in a small training sample, and consequently that a lower degree model is likely to be more accurate for small data. On the other hand, higher degrees of A*n*DE may work better for larger training sets as minimizing bias will be of increasing importance as the size of the data increases [3].

For notational convenience we define

$$x_{i,j,\ldots,q} = \langle x_i, x_j, \ldots, x_q \rangle. \tag{5}$$

For example, $x_{2,3,4} = \langle x_2, x_3, x_4 \rangle$.

A*n*DE classifies using:

$$\hat{\mathrm{P}}_{\mathrm{A}n\mathrm{DE}}(y, \mathbf{x}) = \begin{cases} \sum\limits_{s \in \binom{\mathcal{A}}{n}} \delta(x_s) \hat{\mathrm{P}}(y, x_s) \hat{\mathrm{P}}(\mathbf{x} \mid y, x_s) / \sum\limits_{s \in \binom{\mathcal{A}}{n}} \delta(x_s) : \sum\limits_{s \in \binom{\mathcal{A}}{n}} \delta(x_s) > 0 \\ \hat{\mathrm{P}}_{\mathrm{A}(n-1)\mathrm{DE}}(y, \mathbf{x}) \quad : \text{otherwise.} \end{cases} \tag{6}$$

Attributes are assumed independent given the parents and the class. Hence, $\mathrm{P}(\mathbf{x} \mid y, x_s)$ is estimated by

$$\hat{\mathrm{P}}(\mathbf{x} \mid y, x_s) = \prod_{i=1}^{a} \hat{\mathrm{P}}(x_i \mid y, x_s) \tag{7}$$

Given sufficient training data, A2DE has lower error than AODE, but at the cost of significantly more computational resources.

3 Optimising Memory Consumption

In order to support incremental learning, A*n*DE classifiers compile a table of observed joint frequencies of attribute-value combinations during training. The frequencies table is used in testing to calculate posterior probabilities of class membership. The A*n*DE classifier requires the joint frequencies of n attribute value combinations per class. Additionally, as the classifier defaults to lower orders of n, for super-parents whose values do not occur in the data, the classifier also requires frequencies of all combinations of length up to n per class. As the space requirement for storing these joint frequencies for higher orders of n is undesirable, we developed a new representation that reduces the required space.

The frequency matrix for AODE is a 3-D matrix, where each cell holds the frequency of a (class, parent, child) combination. As an example, consider the frequency matrix for a dataset with two attributes (A and B). Attribute A has two values (a_1 and a_2), while attribute B has three values (b_1, b_2 and b_3). The

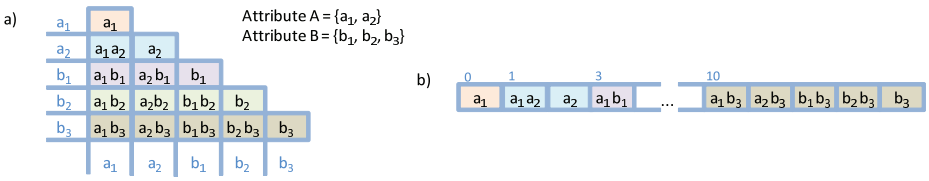


Fig. 1: AODE Parent Child Combinations

parent and child dimensions of the frequency matrix is illustrated in Fig. 1a. It contains cells for each parent-child combination and the (n,n) locations are reserved for frequencies of parents. The 2-D structure is replicated for each class to form the 3-D frequency matrix for AODE.

The representation for A2DE is a 4-D matrix that is a collection of tetrahedral structures for each class. Each cell contains the frequencies of (class, parent1, parent2, child) combinations. The matrix reserves (class, parent1, parent1, parent1) cells for storing frequencies of class-parent1 combinations and (class, parent1, parent2, parent2) cells for storing class-parent1-parent2 combinations.

AnDE requires a matrix of $n + 2$ dimensions to store frequencies of all attribute value combinations. The outer dimension has k elements for each class. The n middle dimensions represent the n parent attribute values and the final dimension represents the child attribute values. The inner dimensions have av elements, where a is the number of attributes and v is the average number of attribute values (including missing values). Consequently, as the size of the frequency matrix is determined by figurate numbers ($P_{n+1}(av) = \binom{av+n}{n+1}$), resulting in a memory complexity of $O(k\binom{av+n}{n+1})$.

Although this representation allows for straight forward access of the frequency of a class-parent-child combination, the matrix has to be implemented as a collection of arrays. This incurs overhead and the does not guarantee that a contiguous block of memory is allocated for the matrix, reducing the possibility that required parts of the matrix are available in the system's cache.

The frequency matrix can be stored compactly with the elements of each row stored in consecutive positions. This representation minimises the overheads that can occur with multi-dimensional arrays. Taking AODE as an example, the rows in the 2-D matrix, which are all combinations involving the corresponding parent, can be stored sequentially in a 1-D array as shown in Fig. 1b.

Allocating slots for all combinations of attribute values in the frequency matrix simplifies access. However, this produces a sparse matrix containing unused slots allocated for impossible combinations. As training and testing cases have only single valued attributes, combinations of attribute values of the same attribute are impossible. In the case of the AODE example, the frequency matrix contains slots to record frequencies of a_1a_2, b_1b_2, b_1b_3 and b_2b_3, which are impossible combinations (shaded in black in Fig. 2a). The size of the frequency matrix can be reduced by avoiding the allocation of memory for such impossible

combinations. In the AODE example, the size of the 2-combinations matrix can be reduced from 10 to 6. The size of the n combinations matrix is $\binom{a}{n+1}v^{n+1}$.

To avoid allocating space for impossible combinations and simplify indexing, the frequency matrix is decomposed into a series of structures for storing attribute value combinations of a specific length. Taking the AODE example, the set of 1-D arrays for storing only possible attribute value combinations is shown in Fig. 2b. Array `freq1` contains frequencies of each attribute value and Array `freq2` contains frequencies of all valid attribute value pairs.

4 Optimising Testing Time

AnDE classifies a test instance by calculating posterior probabilities of class membership. They are calculated by iterating through all parent-child permutations, resulting in a time complexity of $O(ka^{(n+1)})$. We reduced the overall testing time by reorganising the frequency matrix and the looping structure to taking advantage of locality of reference.

The CPU cache is a fast but limited memory resource, which stores copies of most frequently used data. It is used to reduce average time to access memory. We reorganized the memory representation and minimized data retrieval from memory to improve the likelihood of availability of data in the CPU cache.

The compact memory representation for the frequency matrix is a 2-D array, which contains k copies of arrays that record n-combination attribute value frequencies per class. For example, Fig. 3a illustrates the memory representation for a dataset with two attributes (A and B) and two classes (c_1 and c_2). The 2-D representation is poorly suited for accessing all class frequencies of some attribute-value combination. Especially, in the case of datasets with large collection of attributes, this representation reduces the likelihood of all the per-class

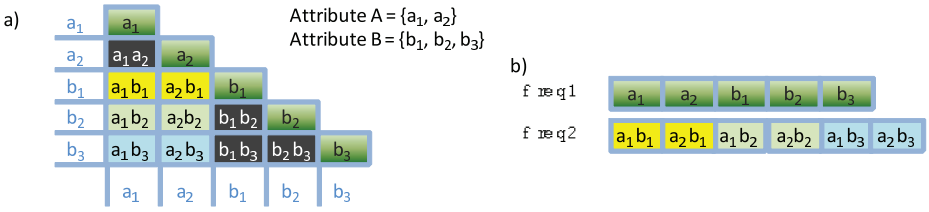


Fig. 2: Valid AODE Parent Child Combinations

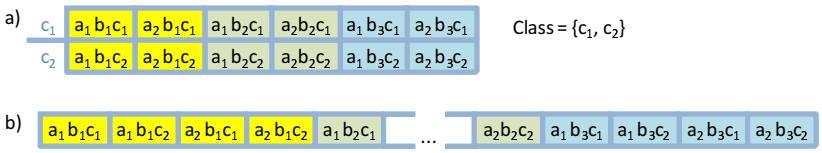


Fig. 3: Storing per Class Frequencies in Sequence

frequencies of some attribute value combination being available in the system's high-speed access cache.

The locality of reference of attribute-value combinations for all classes can be improved by storing them next to each other. Taking AODE as an example, the 2-D frequency matrix (Fig. 3a) can be represented in a 1-D array by interleaving the per class frequencies as shown in Fig. 3b. This representation improves the chances of the frequencies of attribute-value combinations for both the classes being available in high-performance memory. In order to take full advantage of locality of reference of class frequency combinations the looping structure of the classifier also had to be rearranged from looping through each class, parent and child to loop through each parent, child and class.

AnDE requires conditional probabilities for all parent child permutations. Iterating through all permutations requires all relevant offsets to be retrieved, indexes to be calculated and the relevant frequencies to be retrieved. Although these retrievals would be loaded into the CPU cache, they are only used once. In order to reuse data and improve the likelihood of data being available in the CPU cache, we modified the implementation to only iterate over unique combinations. During each iteration conditional probabilities for all permutations of each combination are calculated. This results in reducing the iterations from $ka^{(n+1)}$ to $ka^{(n)}$ and reducing the total number of memory accesses.

The conditional probability of a parent-child attribute permutation is calculated by dividing the frequency of parent-child attributes occurring together by the frequency of parents. The numerator is constant for all permutations of a parent combination. The improved implementation also allows this numerator to be reused, reducing the amount of frequency fetches and the number of index calculations. Overall, this reduces the number of frequency accesses of parent-child attribute value combinations to $\frac{1}{2}$ for AODE, $\frac{1}{3}$ for A2DE and $\frac{1}{4}$ for A3DE.

5 Evaluation

The effectiveness of the improvements to reduce memory usage and testing times were evaluated on a collection of Datasets from the UCI machine learning repository[1]. The evaluation was focused on three members of the AnDE family of algorithms: AODE, A2DE and A3DE. Although NB is the first member of the AnDE family, it was not evaluated as the improvements are unlikely to have any impact. The improvements were compared against the Weka version of AODE and naive versions of A2DE and A3DE.

5.1 Test Environment

We selected nine datasets, described in Table 1, from the UCI machine learning repository for the comparisons. The chosen collection includes small, medium and large datasets with small, medium and high dimensionality. The datasets were split into two sets, with 90% of the data used for training and the remaining 10% used for testing. The experiments were conducted on a single CPU single

Dataset	Cases	Att	Values	Classes	Dataset	Cases	Att	Values	Classes
Abalone	4177	8	24	3	House Votes 84	435	16	48	2
Adult	48842	14	117	2	Sonar	208	60	180	2
Connect-4	67557	42	126	3	SPAM E-mail	4601	57	171	2
Covertype	581012	54	118	7	Waveform-5000	5000	40	120	3
Dermatology	366	34	132	6					

Table 1: Datasets used for experiments

core virtual Linux machine running on a Dell PowerEdge 1950 with dual quad core Intel Xeon E5410 processor running at 2.33GHz with 8 GB of RAM.

The implementations of the three algorithms of the AnDE family are limited to categorical data. Consequently, all numerical attributes are discretized. When MDL discretization [5], a common discretization method for NB, was used within each cross-validation fold, we identified that many attributes have only one value. So, we discretized numerical attributes using three-bin equal-frequency discretization prior to classification for these experiments.

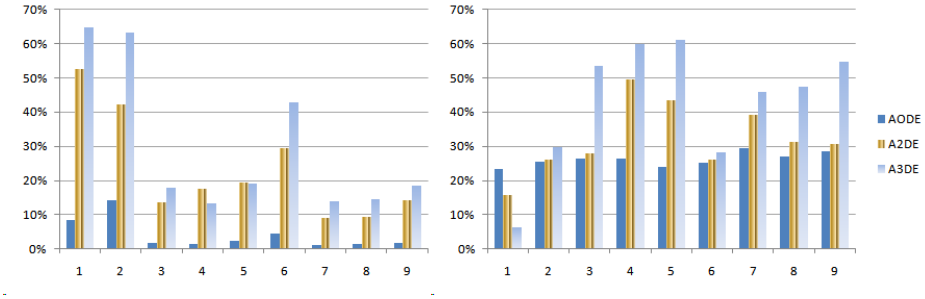
The memory usage of the classifier was measured by the 'Classmexer' tool [4], which uses Java's instrumentation framework to query the Java virtual machine (JVM). It follows relations of objects, so that the size of the arrays inside arrays are measured, including their object overhead and padding.

Accurately measuring execution time for the Java platform is difficult. There can be interferences due to a number of JVM processes such as garbage collection and code profiling. Consequently, to make accurate execution time measurements, we use a Java benchmarking framework [2] that aims to minimize the noise during measurements. The framework executes the code for a fixed time period (more than 10 seconds) to allow the JVM to complete all dynamic optimizations and forces the JVM to perform garbage collection before measurements. All tests are repeated in cases where the code is modified by the JVM. The code is also executed a number of times with the goal of ensuring the cumulative execution time to be large enough for small errors to be insignificant.

5.2 Optimised Memory Consumption

The memory usage for AnDE was reduced by the introduction of a new data structure that avoids the allocation of space for impossible combinations. The reductions in memory usage for the enhanced AnDE implementations were compared against the respective versions of AnDE that stores the frequency matrix in a single array. We do not present the memory reductions of compacting the multi-dimensional array into one dimension as they are specific to Java.

The reductions in memory usages are summarised in Fig. 4a. Results show that the memory reduction for AODE ranged from 1% to 14%. The highest percentage in reduction was observed for the adult dataset, which had a reduction of 9.67KB. The main reason for the large reduction is the high average number



(a) Memory Usage (b) Mean Testing Times

Fig. 4: Proportions of Reductions in Memory Usage and Testing Times

of attribute values of 8.36 for the adult dataset. In contrast, the other datasets have average number of attribute values of around 3.

The memory usage for A2DE was reduced by a minimum of 9% to a maximum of 53%. The maximum amount of reduction in memory was observed for the high-dimensional Covertype dataset, which had a reduction of around 4.68MB.

The enhanced version of A3DE resulted in the highest reduction in memory usage with reductions ranging from 13% to 64%. The reductions in memory usage for the high dimensional Covertype, Dermatology, Sonar Classification and SPAM E-mail datasets were over 100MB.

5.3 Optimised Testing

The total testing times of the algorithms were compared using the test environment. The proportions of reduction in mean test times for AnDE are given in Fig. 4b. The optimisations result in reductions in average testing times for all three algorithms. The reductions for A3DE were highest, with a 61% (0.83s) mean reduction for the small but high-dimensional Dermatology dataset and 60% (8.89ks) reduction for the large and high-dimensional Covertype dataset. The improvements also reduced the testing times of low dimensional datasets of Abalone (6%) and House Votes (28%)

The reductions in testing times were substantial for A2DE, with reductions ranging from 16% (for Abalone) to 50% (Covertype). The improvements to AODE also resulted in reduced total execution times ranging from 23% to 30%. The highest percentage of reduction was exhibited for the dataset with the largest number of attributes, Sonar Classification.

6 The Evaluation of A3DE

We evaluated the classification accuracy of AnDE algorithms comparing how their performance varies as n increases within the AnDE framework. Previous research [10] has compared the effectiveness of AODE to A2DE, but only limited

experimental results were presented for A3DE as the Weka implementation failed on high dimensional datasets due to its high memory requirements.

We compared the effectiveness the AnDE members using the enhanced versions implemented in the Weka workbench on the 62 datasets that were used to evaluate the performance of A2DE [10]. Each algorithm was tested on each dataset using the repeated cross-validation bias-variance estimation method [8]. We used two-fold cross validation to maximise variation in the training data between trials. In order to minimise the variance in our measurements, we report the mean values over 50 cross-validation trials.

The experiments were conducted on the same virtual machine used to evaluate the effectiveness of the improvements. Due to technical issues, including memory leaks in the Weka implementation, increasing amounts of memory is required when multiple trials are conducted. Consequently, we were unable to get bias-variance results for four datasets (Audiology, Census-Income, Covertype and Cylinder bands), that were of high dimensionality. We compared the relative performances of AODE, A2DE and A3DE on the remaining 58 datasets. The lower, equal or higher outcomes when the algorithms are compared to each other is summarised as win/draw/loss records in Tab. 2.

The results show that the bias decreases as n increases at the expense of increased variance. The bias of A3DE is lower significantly more often than not in comparison to A2DE and AODE. The bias of A2DE is lower significantly more often relative to AODE. In contrast, the variance of AODE is lower significantly more often than A2DE and A3DE. The variance of A2DE is lower significantly more often relative to A3DE.

None of the three algorithms have a significantly lower zero-one loss or RMSE on the evaluated datasets. We believe that this is due to the wide range sizes of datasets used in the evaluation. We hypothesize that members of the AnDE family with lower n, that have a low variance, are best suited for small datasets. In contrast, members with higher degrees of n are best suited for larger datasets.

6.1 A3DE performance on Large datasets

To assess the hypothesis that increasing values of n within the AnDE family are suited to increasing data quantity, we compared A3DE to lower-order family members on datasets with over 10,000 cases. Out of the 58 datasets, seven

	A3DE vs A2DE		A3DE vs AODE		A2DE vs AODE	
	W/D/L	p	W/D/L	p	W/D/L	p
Bias	34/3/21	**0.052**	40/1/17	**0.002**	43/0/15	**<0.001**
Variance	17/2/39	**0.002**	15/2/41	**<0.001**	16/1/41	**<0.001**
Zero-one loss	24/3/31	0.209	24/2/32	0.175	29/2/27	0.447
RMSE	24/3/31	0.209	28/0/30	0.445	31/1/26	0.298

Table 2: Win/Draw/Loss: AnDE, n = 1, 2 and 3 on 58 data sets

	A3DE vs A2DE		A3DE vs AODE	
	W/D/L	p	W/D/L	p
Bias	6/0/1	0.063	7/0/0	**0.008**
Variance	2/0/5	0.227	1/0/6	0.063
Zero-one loss	7/0/0	**0.008**	7/0/0	**0.008**
RMSE	7/0/0	**0.008**	7/0/0	**0.008**

Table 3: Win/Draw/Loss: AnDE, n=1,2 and 3 on large data sets

datasets (Adult, Connect-4 Opening, Letter Recognition, MAGIC Gamma Telescope, Nursery, Pen Digits and Sign) satisfied this criterion. The number of cases of the chosen datasets ranged from just over 10,000 cases (Pen Digits) to over 60,000 cases (Connect-4 Opening).

The evaluation results are summarised as win/draw/loss records in Table 3. As expected, the results show A3DE has a lower bias and higher variance than A2DE and AODE. The zero-one loss and the RMSE of A3DE are lower for all the evaluated datasets in comparison to A2DE and AODE (p=0.008). These results confirm that A3DE performs better than its lower-dimensional variants at classifying larger datasets.

7 Conclusions

The AnDE family of algorithms perform search-free learning. The parameter n controls the bias-variance trade-off such that $n = a$ provides a classifier whose asymptotic error is the Bayes optimum. We presented techniques for reducing the memory usage and the testing times of the AnDE implementations that make A3DE feasible to employ for higher-dimensional data. As A3DE is superior to AnDE with lower values of n when applied to large data, and as the linear complexity and single pass learning of AnDE make it particularly attractive for learning from large data, we believe these optimizations have potential for considerable impact.

We developed a new compact memory representation for storing the frequencies of attribute-value combinations that stores all frequencies in a 1-D array avoiding the allocation of space for impossible attribute-value combinations. The evaluation results showed that the enhancements substantially reduced the memory requirements. The enhancements reduced the overall A3DE memory requirements ranging from 13% to 64%, including reductions of over 100MB for the high-dimensional datasets.

The classification times of the AnDE algorithms were improved by reorganising the memory representation to maximise locality of reference and minimising memory accesses. These enhancements resulted in substantial reductions to the total testing times for the AnDE family of algorithms. In the case of A3DE, the maximum reduction in total testing time was 8.89ks, which was a reduction of 60%, for the Covertype dataset.

The enhancements to the AnDE algorithms opened the door for evaluating the performance of A3DE. As expected, the results showed that A3DE has lower bias in comparison to A2DE and AODE. The results for zero-one error between A3DE, A2DE and AODE did not produce a clear winner. However, A3DE produced the lowest error for large datasets (with over 10,000 cases).

The computational complexity of AnDE algorithms is linear with respect to the number of training examples. Their memory requirements are dictated by the number of attribute values in the data. Consequently, the most accurate and feasible member of the AnDE algorithm for a particular dataset will have to be decided based on the dataset's size and its dimensionality.

References

1. C. L. Blake and C. J. Merz. UCI Repository of Machine Learning Databases . `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
2. B. Boyer. Robust Java benchmarking. `http://www.ibm.com/developerworks/java/library/j-benchmark1.html`, 2008.
3. D. Brain and G. Webb. The Need for Low Bias Algorithms in Classification Learning From Large Data Sets. In *Proc. of the 6th European Conference (PKDD 2002)*, pages 62–73. Springer-Verlag, 2002.
4. N. Coffey. Classmexer agent. `http://www.javamex.com/classmexer/`.
5. U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of the 13th Int. Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1993.
6. B. Hui, Y. Yang, and G. I. Webb. Anytime classification for a pool of instances. *Machine Learning*, 77(1): 61-102, 2009.
7. E. Keogh and M. Pazzani. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proc. of the International Workshop on Artificial Intelligence and Statistics*, pages 225–230, 1999.
8. G. I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.
9. G. I. Webb, J. Boughton, and Z. Wang. Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.
10. G. I. Webb, J. Boughton, F. Zheng, K. M. Ting, and H. Salem. Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. *Machine Learning*, in-press.
11. I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.
12. F. Zheng and G. I. Webb. A comparative study of semi-naive Bayes methods in classification learning. In S. J. Simoff, G. J. Williams, J. Galloway, and I. Kolyshakina, editors, *Proc. of the 4th Australasian Data Mining Conference (AusDM05)*, pages 141–156, 2005.
13. Z. Zheng and G. I. Webb. Lazy learning of Bayesian rules. *Machine Learning*, 41(1):53–84, 2000.