Evolving Artificial Datasets to Improve Interpretable Classifiers

Michael Mayo and Quan Sun
Department of Computer Science
University of Waikato
Hamilton, New Zealand
Email: {mmayo, qs12}@waikato.ac.nz

***Abstract*—Differential Evolution can be used to construct effective and compact artificial training datasets for machine learning algorithms. In this paper, a series of comparative experiments are performed in which two simple interpretable supervised classifiers (specifically, Naive Bayes and linear Support Vector Machines) are trained (i) directly on "real" data, as would be the normal case, and (ii) indirectly, using special artificial datasets derived from real data via evolutionary optimization. The results across several challenging test problems show that supervised classifiers trained indirectly using our novel evolution-based approach produce models with superior predictive classification performance. Besides presenting the accuracy of the learned models, we also analyze the sensitivity of our artificial data optimization process to Differential Evolution's parameters, and then we examine the statistical characteristics of the artificial data that is evolved.**

Keywords*—*Differential Evolution, supervised machine learning, interpretable models, artificial data.

I. Introduction

Machine learning algorithms for building predictive models typically assume that the input or training data for a particular learning algorithm is fixed.

This training data is commonly in the form of a two-dimensional table consisting of features (the columns) and labelled examples (the rows). In data mining practice, it is common to consider subsets of this data if the table is too large along one or both of these dimensions.

For example, many methods exist for feature selection (e.g. Deng & Runger [1]). Less commonly used but none-the-less effective methods also exist for selecting subsets of examples, such as Cano et. al's [2] evolutionary instance selection algorithm.

Practitioners may also project the data in order to reduce its dimensionality. Principle Components Analysis [3] is a standard example of this.

By and large, however, beyond row/column selection and performing projections, the training data for data mining experiments is traditionally an unchanging constant. Yet, ironically, the choice of training data can have a significant impact on classifier performance.

In this paper we explore the possibility that training datasets for machine learning algorithms may be artificially created using state-of-the-art Evolutionary Algorithm methods. The evolutionary method we propose "wraps" an existing machine learning algorithm, optimizing an artificial dataset that is used as input to the machine learning algorithm.

We demonstrate that performance increases are possible for two simple, interpretable machine learning classification algorithms, specifically Naive Bayes and linear Supper Vector Machines (SVMs). The increase appears to be most pronounced for Naive Bayes.

Furthermore, we also demonstrate that only very small artificial datasets are required to train effective classifiers. Whereas real data may have thousands of examples, in all of our experiments, the evolved artificial datasets consist of only ten examples.

In the next section, we briefly describe the algorithms used in this paper. In Section III, we outline our novel algorithms for artificial dataset construction, namely *Instance Optimization for Naive Bayes* (IO-NB) and *Instance Optimization for Support Vector Machines* (IO-SVM). Sections V, VI and VII then detail extensively the result of our experiments, before the paper is concluded in the final section.

II. Background

Differential Evolution (DE) and the simple classification methods that we utilized in our experiments are detailed in this section. We also briefly review the role of artificial data in machine learning.

A. High Dimensional Optimization using Differential Evolution

Differential Evolution (DE) is a meta-heuristic search algorithm for multi-dimensional continuous optimization. It is ideal for situations where there is no gradient function available, and where the cost function is discontinuous and non-linear.

We have focussed on DE in this research (i) because it is simple to implement and understand, with fewer parameters than other methods; and (ii) because it frequently performs at or near the state-of-the-art in current competitions and evaluations [5].

Figure 1 depicts pseudocode for the variant of DE used in our experiments. We chose the "DE/best/1/bin" version of DE because it was found to outperform classic DE in Tasoulis et. al [6], and is therefore a good choice for experimentation.

The algorithm has three key parameters: NP, the population size; F, the amplification parameter; and CR, the

```
1: procedure MINIMIZECOSTDE(V)                    ▷ DE/best/1/bin
2:     Pop ← InitializePopulation(V, NP)
3:     Costs^Pop ← EvaluatePop(Pop)
4:     while MaxFEsNotExceeded() do
5:         TrialPop ← {}
6:         D* ← Best(Pop)
7:         for D^x ∈ Pop do                        ▷ Mutate/Recombine
8:             r ← RandomInt(1, Length(D^x))
9:             D^r1, D^r2 ← RandomlyPick(Pop)
10:            T^x ← EmptyVector()
11:            for i = 1 to Length(D^x) do
12:                if Random(0, 1) ≤ CR||i == r then
13:                    T^x[i] ← D*[i] + F(D^r1[i] − D^r2[i])
14:                else
15:                    T^x[i] ← D^x[i]
16:                end if
17:            end for
18:            TrialPop ← TrialPop ∪ {T^x}
19:        end for
20:        Costs^TrialPop ← EvaluatePop(TrialPop)
21:        for T^x ∈ Pop do                        ▷ Select
22:            if Costs^TrialPop[x] ≤ Costs^Pop[x] then
23:                D^x ← T^x
24:            end if
25:        end for
26:    end while
27:    return Best(Pop)
28: end procedure
```

Fig. 1. Differential Evolution algorithm.

crossover rate. The dimensionality of the vectors that are evolved is fixed and dependent on the problem.

Essentially, the algorithm maintains a population of individual vectors, where NP is the fixed population size.

After initialization and evaluation of the starting population, the main loop (from line 4 to line 26) commences and runs until some terminating condition is true. In our case termination occurs when the total number of cost function evaluations exceeds a fixed limit.

For each iteration of the main loop, a *trial population* of vectors, the same size as the main population, is generated. The construction of trial vectors is undertaken by using DE's mutation and recombination operators, both of which are applied in unison for efficiency (although they may be applied separately if need be). Lines 7 to 19 of the algorithm are the mutation/recombination steps.

It is important to note that the mutation and recombination operators use several very specific random selections.

The first random number, r, is a random index between 1 and the dimensionality of the optimization problem. This is used to ensure that at least one element from the trial vector will always be crossed over (see line 12), so that trial vectors will never be identical to their parents.

The second important random selection occurs on line 8 of the algorithm. Essentially, the idea is to pick two random vectors from the population, D^{r1} and D^{r2}, such that $D^{r1} \neq D^{r1}$, and neither are equal to the current best population member, D^*.

The final random number selected is a random value between 0 and 1 on line 12. This is compared to the crossover rate, CR, and if the value is less than or equal to CR (or if the current index happens to be equal to r, which is tested in the second part of the if statement), then the element of the trial vector at position i is replaced by the ith element of the current best vector D^* plus the scaled difference between the ith elements of D^{r1} and D^{r2}. This step is known as a mutation, with the parameter F (the amplification parameter) regulating the degree of scaling.

If the random number between 0 and 1 exceeds CR (and the current index is not equal to r) on the other hand, then the ith element of T^x is simply copied across from the parent D^x. In this case, mutation is not applied.

It should be evident therefore that lower values for CR ensure that trial vectors will be more similar to their parent vectors.

Higher values of both CR and F, conversely, result in more diverse trial vectors.

Once the trial vectors are generated, they are evaluated using the cost function (line 20). Trial vectors then replace their parents in the next generation if and only if their cost is not greater than that of their parents (see the second nested loop inside the main loop, lines 22 to 25).

Numerous papers have been written about DE and its variants, and the reader is referred to Das & Suganthan [5] for more information.

B. Machine Learning Algorithms

In this paper, we make use of two well-known machine learning algorithms for supervised classification, Naive Bayes [7] and Support Vector Machines (SVMs) with a linear kernel [8], [9]. Space prevents a detailed description of these algorithms, but the reader can find descriptions in most standard machine learning textbooks.

C. Artificial Data in Machine Learning

Most applications of artificial data in machine learning are for testing algorithms, i.e. artificial data with a known pattern is generated, and a classifier is trained using that data in order to determine if the pattern can be learned correctly. For example, see the "people database" techniques developed by Agrawal et. al [11], which is frequently used in the literature.

The closest related work to our work is from the field of nearest neighbour classification. Nearest neighbour classification is typically slow at testing time if the training data is large because each test example must be compared to every single training example. Prior work as surveyed by Triguero et. al [12] concerns techniques for both selecting and creating appropriate small sets of "prototype" examples that can replace the training data and therefore speed up nearest neighbour classification.

In contrast, our work is more general and can be applied to any supervised classification (or regression) method. It is not at all confined to only nearest neighbour methods.

```
1: procedure COST(D^x,V)
2:     C ← LearnClassifier(D^x)
3:     m ← Evaluate(C,V)
4:     return 1 − m
5: end procedure
```

Fig. 2. Cost function. D^x is an artificial dataset, V is the validation set ($V \subseteq D$), C is a classifier, and m is a measure of performance (AUROC) of C against V.

III. Instance Optimization for Classification

This section outlines the Instance Optimization algorithms that are the main contribution of this paper.

Most machine learning algorithms for supervised classification assume the existence of a training set of labeled examples.

In this paper, we refer to this "real" training data as D and to the number of real training examples as $|D|$. Equation 1 describes the training data. Each example or instance comprises of a vector of features x_i along with a singular label or target y_i.

$$D = \{x_1, y_1), (x_2, y_2), \ldots, (x_{|D|}, y_{|D|})\} \tag{1}$$

In contrast to the real training data, Figure 1 and the rest of this paper refer to the artificial data as D^x and the artificial examples as (x'_i, y'_i). Equation 2 defines D^x, where N is the fixed size of the artificial dataset (which, in our experiments, is $N = 10$).

$$D^x = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_N, y'_N)\} \tag{2}$$

A. Cost Function

The heart of our proposed Instance Optimization algorithms is the cost function which is used to evaluate individual artificial datasets.

The cost function is defined as the predictive performance of a classifier C learned on an artificially-created dataset D^x when tested against an a-priori fixed validation set of V, which is a subset of the original "real" data D.

In other words, we use artificial data to learn a classifier and then evaluate it using part of the real data. Figure 2 depicts this cost function.

One difficulty when evaluating classifier performance is that the classes may be imbalanced. For example, a dataset (such as the "Churn" dataset in our experiments) may comprise two classes, but one class may be considerably less represented than the other in terms of the number of examples. In the "Churn" dataset, to illustrate, one class consists of only 8% of the examples.

Optimizing directly for accuracy in that case may be problematic, as an algorithm that simply predicts the majority class can appear to be highly accurate (e.g. 92% correct) without any learning occurring. Accuracy consequently may cause a significant local optima to be produced in the optimization space.

To avoid this problem, we therefore use an alternative performance metric, specifically Area Under the Receiver Operating Characteristics Curve (AUROC) [10].

AUROC has the characteristic that it is insensitive to class imbalance. A classifier that achieves an AUROC value of 0.5 is either predicting randomly or simply predicting the same single class label for all examples. Alternatively, an AUROC value of 1.0 indicates perfect predictive performance.

To use AUROC in our cost function, we subtract it from 1.0 in order to obtain a value to minimize, as Figure 2 shows.

B. Optimization Constraints

We also add the constraint that the dataset must conform in its specification to the real data.

That is, the artificial dataset D^x must consist of the same number of numeric and non-numeric attributes, and the same set of class labels, as the original data D has. Equation 3 states this as a constraint on the optimization problem when D^* is the optimal artificial dataset that is found.

$$\begin{aligned} \underset{D^*}{\text{minimize}} \quad & Cost(D^*, V) \\ \text{subject to} \quad & Schema(D^*) = Schema(D), V \in D \end{aligned} \tag{3}$$

One difficulty encountered in the application of DE to artificial dataset generation is that datasets for machine learning are two dimensional tables with constraints on some of the columns (which we have characterized as the "schema" mentioned in Equation 3). DE, on the other hand, is designed to optimize only one dimensional continuous vectors.

In order to apply the standard DE recombination and mutation operators to two-dimensional datasets with rows and columns, therefore, we must first of all "unroll" each dataset into a one-dimensional vector.

For example, with $N = 10$ examples in the artificial dataset and ten features, the dataset is unrolled to a vector of length 100. If the dataset has 335 features (which is the case for one of the datasets we use), then the dimensionality of the problem is 3,350. It should be clear from these examples that artificial dataset optimization is therefore a very challenging, very high dimensional problem!

After unrolling, mutation and recombination can then be applied within the DE algorithm to generate trial one-dimensional vectors, but before these trial vectors can be "re-rolled" back into two-dimensional datasets, modifications must be made to ensure that the schema of the new trial artificial datasets still matches that of the original one.

This amounts to firstly preventing numeric values from going higher or lower than those observed in the real dataset, and secondly to preserving the discreteness of some features. If an attribute in a trial artificial dataset does exceed the allowable range, then our method simply clips the value to either the maximum or minimum allowed value.

Next, some attributes in the original, real dataset may be discrete (e.g. taking allowable values {1,2,3} only), but trial datasets may contain values outside of the allowable set due

1: **procedure** TRAINCLASSIFIER(D)
2: $V \leftarrow Sample(D)$
3: $D^* \leftarrow MinimizeCostDE(V)$
4: **return** $LearnClassifier(D^*)$
5: **end procedure**

Fig. 3. Classifier training function. D is the original training set, V is a sample of it (the validations set), and D^* is the artificial dataset that minimizes cost with respect to V. The classifier is learned from the optimal artificial dataset rather than the real dataset.

to the addition of scaled differences (see line 13 of Figure 1). For example, suppose that we are calculating the value of T^x at position i, with $D^*[i] = 3$, $D^{r1}[i] = 1$, $D^{r1}[i] = 2$ and $F = 0.4$. According to the DE algorithm, the value of the trial vector $T^x[i]$ will be $D^*[i] + F(D^{r1}[i] - D^{r2}[i]) = 3 + 0.4(1 - 2) = 2.6$.

To solve this problem, we round all values that should be discrete to the nearest allowable discrete value, so therefore in the above example the invalid value 2.6 will be rounded up to the nearest valid value of 3.

These two measures ensure that the schema consistency constraint between the artificial and the real datasets as specified in Equation 3 holds, and they must be applied before the cost function is evaluated on each artificial dataset.

A similar process is taken during the initialisation phase of the DE algorithm to create the initial population: random one dimensional vectors are generated uniformly using maximum and minimum allowable values as determined from the real data, and then the constraints and "rolling up" process is applied in order to turn the initial vectors into valid two-dimensional datasets.

C. Supervised Training Procedure

We now come to describe the method of training a classifier using Instance Optimization.

The basic procedure is outlined in Figure 3. Essentially, the classifier training steps are firstly to select a subsample of the real data to use as a validation set. Next, the best artificial dataset D^* minimizing the cost function with respect to V is found using DE. Finally, a Naive Bayes or linear SVM classifier is built from this artificial dataset (as opposed to the real dataset).

Note that the real dataset is used during the training process, but its use is *implicit* as a part of the cost function that DE attempts to minimize.

IV. EXPERIMENTAL SETUP

Having described the Instance Optimization algorithm, we now describe the experiments we performed.

A. Parameters

As mentioned previously, DE requires a only small number of parameters, specifically the amplification parameter (F), the crossover rate (CR) and the population size (NP).

For our initial experiments, we used arbitrary initial values for F and CR that seemed to work well during initial testing. We also chose a deliberately small value for the population size, specifically $NP = 20$, which in initial testing also worked well.

With respect to timeliness, the maximum number of cost function evaluations was set to 4,000. That is, DE was terminated as soon as 4,000 cost function evaluations had occurred, which amounted to $\frac{4000}{20} = 200$ generations per run of DE. A fixed value for the number of evaluations was chosen primarily because the dimensionality of the datasets varies considerably, and therefore had the number of function evaluations been a function of the dimensionality (as it usually is), then this parameter would have varied considerably between datasets. It was felt therefore that a low fixed number of function evaluations would be the most suitable, at least for initial exploratory testing.

Table I gives the definitive set of parameters used in our experiments. The additional parameter V is the size of the validation set as a proportion of the size of original, real, training dataset. The validation set is chosen by random subsampling in our approach.

TABLE I. PARAMETERS USED IN THE EXPERIMENTS.

Name	Definition	Value
F	DE amplification factor	0.5
CR	DE crossover rate	0.1
NP	DE population size	20
FE	max. function evals	4,000
N	num. artificial examples	10
V	validation set size	1.0

B. Algorithm Variants

For evaluating the effectiveness of evolutionary optimisation methods, a comparison of the optimizer's performance with random search is warranted. If random search performs as well as the optimizer, then it may be the case that the cost function is simply too noisy to permit effective optimization.

To investigate the performance of random search, therefore, we developed two simple random search algorithms: RS-NB and RS-SVM.

These algorithms essentially generate 4,000 random artificial datasets using the same initialization method as our DE-based algorithms. The 4,000 random datasets are evaluated and the best single dataset is returned.

Table II summarizes the four algorithms that we evaluated in our experiments.

TABLE II. SEARCH ALGORITHM VARIANTS USED IN THE EXPERIMENTS.

Name	Description
RS-NB	Random search for Naive Bayes
RS-SVM	Random search for Linear SVM
IO-NB	Instance Optimization for Naive Bayes
IO-SVM	Instance Optimization for Linear SVM

C. Datasets

We tested the four algorithms in Table II on a broad selection of challenging supervised classification datasets.

The datasets can be divided into two classes: artificial and real. The artificial datasets are all low dimensional and have

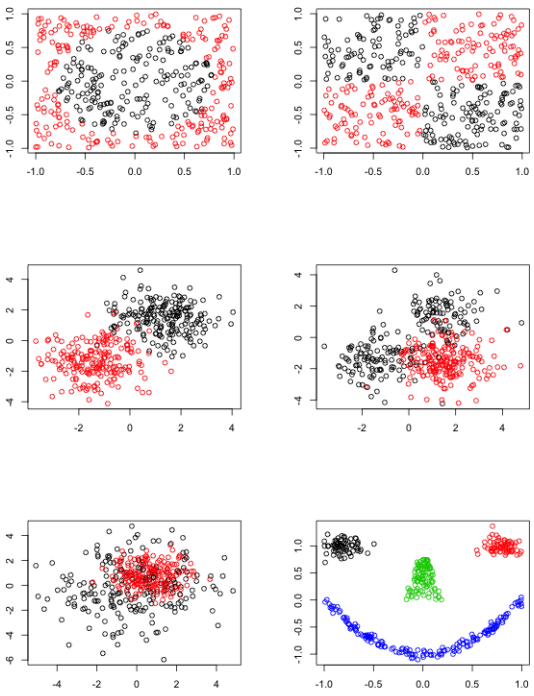

Fig. 4. Artificial datasets used in the experiment. From top to bottom, left to right, they are (i) circle, (ii) xor, (iii) two-norm, (iv) three-norm, (v) ring-norm, and (vi) smiley.

a small number of instances, but despite this deceptive simplicity, some of these datasets are designed to be specifically challenging to certain classes of learning algorithm.

The second group of datasets that we used, on the other hand, were much more challenging in terms of both the dimensionality (ranging up to 335 for one of the datasets), size (up to 10,000 examples per dataset) and difficulty of the classification tasks. They are also real-world datasets used in recent difficult machine learning competitions.

All of the datasets utilised in this research are currently available online[1].

1) Artificial Datasets: The artificial classification datasets we used were generated using algorithms in the R package MLBench [13]. Each dataset consists of two features (x_1 and x_2) and a single class y, and they are all depicted visually in Figure 4.

These datasets were chosen for several reasons.

Firstly, the "circle" and "xor" datasets represent problems where the attributes are clearly not independent. For example, the "circle" dataset consists of one class of all points falling inside a unit circle, and another class of all points falling outside of the circle.

Similarly, "xor" is a well-known challenging problem with dependent features where the class label is only predictable when the features are combined in a non-linear way.

[1]https://github.com/mmayo888/MayoSunDatasets

The datasets "two-norm", "three-norm" and "ring-norm" were suggested by Breiman [14] as good tests for classification methods, and are also included. Finally, the four-class "smiley" dataset has been added for brevity.

2) Real Datasets: Table III enumerates the ten large and challenging real datasets used in our experiments. These datasets were originally downloaded by the second author from (i) the UCI repository, (ii) the UCSD FICO data mining contest 2011 website and (iii) the KDD Cup 2009 website (please see [15] for the original legacy links).

The datasets were selected because they are large and come from very diverse research and industrial areas. To make timely experiments feasible, any datasets that were originally multiclass were converted into binary classification problems by retaining only the two largest classes and deleting all of the remaining classes. Following this, the size of each dataset was evaluated. For datasets having more than 10,000 examples, a subset of 10,000 examples were randomly selected. The interested reader is referred to Sun and Phafringer [15] for more details on these datasets as well as the results of previous machine learning experiments utilising these datasets.

TABLE III. REAL DATASETS USED IN THE EXPERIMENTS. ALL DATASETS ARE BINARY CLASS DATASETS WITH 10,000 EXAMPLES EXCEPT FOR CHESS THAT COMPRISES 8,747 EXAMPLES.

Dataset	Num. Attributes	Minority Class Size
Adult	14	23%
Chess	6	48%
Connect-4	42	26%
Covtype	54	43%
Churn	190	8%
Localization	8	37%
MAGIC	11	35%
MiniBooNE	50	28%
Poker	11	45%
UCSD FICO	335	9%

D. Evaluation Methodology

For all experiments, we took each dataset and split it into 20% training data and 80% testing data.

Classifiers were then trained using all four algorithms from Table II on the same training split. The final best models learned using the algorithm in Figure 3 was evaluated against the test split.

This training/testing process was repeated thirty times with different random training and testing splits in each repetition in order to obtain averaged performance metrics.

The performance metric of each classifier on the test data that we recorded was the same one that was optimized on the training data, specifically the AUROC. In order to determine the significance of the difference between the Instance Optimization classifiers and their base classifiers alone, as well as between Instance Optimization and Random Search, we compared the results using a standard paired t-test. A paired t-test is appropriate here because the samples from each group are paired, i.e. the same training/test split is shared across algorithms for each of the thirty repetitions.

V. EXPERIMENT 1: CLASSIFICATION RESULTS

The first set of results that we present are given in Tables IV and V.

TABLE IV. RESULTS USING IO-NB. KEY: ○, ● INDICATES STATISTICALLY SIGNIFICANT IMPROVEMENT, DEGRADATION COMPARED TO NB.

Dataset	NB	RS-NB	IO-NB
circle	0.94	0.96 ○	0.98 ○
xor	0.52	0.61 ○	0.59 ○
twonorm	1.00	0.99 ●	0.99 ●
threenorm	0.96	0.96 ●	0.96 ●
ringnorm	0.86	0.84 ●	0.85 ●
smiley	1.00	0.98 ●	1.00
Adult	0.89	0.81 ●	0.89
Chess	0.76	0.71 ●	0.77 ○
Connect4	0.83	0.66 ●	0.81 ●
Covtype	0.80	0.77 ●	0.83 ○
Churn	0.57	0.61 ○	0.64 ○
Localiz.	0.88	0.80 ●	0.88
MAGIC	0.75	0.83 ○	0.86 ○
MiniBooNE	0.52	0.83 ○	0.88 ○
Poker	0.51	0.51	0.51 ○
UCSD FICO	0.60	0.58 ●	0.61 ○
Average	0.77	0.78	**0.82**

TABLE V. RESULTS USING IO-SVM. KEY: ○, ● INDICATES STATISTICALLY SIGNIFICANT IMPROVEMENT, DEGRADATION COMPARED TO SVM.

Dataset	SVM	RS-SVM	IO-SVM
circle	0.51	0.59 ○	0.60 ○
xor	0.50	0.58 ○	0.61 ○
twonorm	0.99	1.00	0.99
threenorm	0.93	0.93	0.93
ringnorm	0.71	0.71 ●	0.71 ●
smiley	1.00	0.96 ●	1.00
Adult	0.88	0.81 ●	0.86 ●
Chess	0.77	0.70 ●	0.74 ●
Connect4	0.88	0.66 ●	0.78 ●
Covtype	0.85	0.72 ●	0.82 ●
Churn	0.49	0.61 ○	0.65 ○
Localiz	0.72	0.68 ●	0.72
MAGIC	0.83	0.83 ●	0.84 ○
MiniBooNE	0.88	0.88 ●	0.90 ○
Poker	0.50	0.50	0.51 ○
UCSD FICO	0.56	0.59 ○	0.61 ○
Average	0.75	0.73	**0.77**

Table IV comprises the results for IO-NB. Mean performances are shown to two decimal places.[2] The first point to note from this table is that the performance improvement as a result of using IO-NB instead of NB is quite dramatic: on average the AUROC improves by 0.05 across all datasets.

Secondly, most of these improvements are in the real datasets as opposed to the simple artificial datasets. In fact, only the "Adult", "Localization" and "Poker" datasets fail to record significantly improved performance as a consequence of the IO-NB algorithm being used. The largest improvement from NB to IO-NB is on the "MiniBooNE" dataset that records an increase in mean AUROC from 0.52 to 0.88.

With regard to the artificial datasets, the Naive Bayes learning algorithm by itself clearly performs very well on most of the datasets already, except for "xor". Surprisingly, IO-NB is able to produce a better model for this particular dataset, and performance is improved from 0.52 (i.e. nearly random) to 0.59 – although in fact the random search algorithm RS-NB in this single case actually outperforms IO-NB.

Table V depicts the results for IO-SVM. In general, the effect achieved by replacing SVM with IO-SVM is much less pronounced, and on some datasets, the improvement is actually reversed to become a significant degradation. Comparing IO-SVM's results to those of IO-NB, it is evident that this lesser improvement is mostly a consequence of the SVM classifier without instance optimization (itself a very powerful state-of-the-art classifier) actually performing much better on many of the datasets than Naive Bayes does without instance optimization.

In spite this, there is still an average overall improvement of AUROC from 0.75 for the SVM algorithm to 0.77 for the IO-SVM algorithm.

The final analysis we performed was to compare IO-NB to RS-NB, and IO-SVM to RS-SVM, in order to gauge how many times instance optimization significantly outperformed random search (as opposed to the base classifiers NB and SVM learned on the real datasets).

The results of this analysis are given in Table VI. The first four lines of this table summarize the significance results in Tables IV and V. The final two lines compare the IO-* classifiers against their RS-* variants. The results clearly show that IO-NB and IO-SVM significantly outperform RS-NB and RS-SVM on most (12 or 13) of the datasets, and they are never significantly worse on the remaining datasets.

TABLE VI. COMPARISONS BETWEEN NB, SVM, RS-NB, RS-SVM, IO-NB AND IO-SVM IN TERMS OF STATISTICALLY SIGNIFICANT WINS/LOSSES/DRAWS.

Comparison	Wins/Losses/Draws
IO-NB vs. NB	9/4/3
IO-SVM vs. SVM	7/5/4
RS-NB vs. NB	5/10/1
RS-SVM vs. SVM	4/9/3
IO-NB vs. RS-NB	13/0/3
IO-SVM vs. RS-SVM	12/0/4

An examination of which datasets the IO-* classifiers failed to outperform their corresponding RS-* counterparts revealed that they were all artificial datasets, except for one – namely the "Poker" dataset. The IO-* classifiers improve significantly on random search in every other case. All significance tests were performed with 95% confidence.

VI. EXPERIMENT 2: DE PARAMETER OPTIMIZATION RESULTS

In the next set of experiments, we aimed to determine how sensitive the IO-* algorithms are to DE's parameters.

Although DE has only a small number of parameters, previous research has shown that the algorithm's performance on different problems is often highly dependent on the values for the parameters [5].

For example, on problems where the variables are highly separable (i.e., where each dimension can be optimized individually, to some degree), then low values of CR often perform well. Conversely, in situations where the variables are not separable (or non-linear), higher values of CR are superior.

[2] It should be noted that some differences are significant but small, such as the "threenorm" data in which the difference is at the third decimal place and therefore not shown in the table.

We repeated the experiments from the previous section several times, this time varying the values of F and CR from the very small (0.05) to the very large (1.0 in the case of CR and 1.5 in the case of F). For expediency, we reduced the number of repetitions from 30 in the previous experiments to ten in these ones, but other than that, all other parameter values were the same.

The results for IO-NB and IO-SVM are given in Tables VII and VIII.

As can be observed from the tables, in the case of IO-NB, there is a distinct preference for certain values of F and CR. For both IO-NB and IO-SVM, performance peaks at $CR = 0.5$. This indicates that the problem must be to some degree non-separable, and that therefore DE is an ideal optimization algorithm for this problem. In terms of the F parameter, the optimal value appears to lie in the 0.25-0.5 range.

TABLE VII. AUROC RESULTS OF PARAMETER OPTIMIZATION EXPERIMENTS FOR IO-NB USING DIFFERENT VALUES OF F AND CR, ALL OTHER PARAMETERS REMAINING THE SAME.

	CR=0.05	CR=0.10	CR=0.50	CR=1.0
F=0.05	0.8090	0.8170	0.8093	0.7473
F=0.10	0.8117	0.8183	0.8102	0.7547
F=0.25	0.8125	0.8140	0.8150	0.7785
F=0.50	0.8109	0.8103	**0.8206**	0.7913
F=1.00	0.8086	0.8072	0.7983	0.7906
F=1.50	0.8084	0.8021	0.7899	0.7883

TABLE VIII. AUROC RESULTS OF PARAMETER OPTIMIZATION EXPERIMENTS FOR IO-SVM USING DIFFERENT VALUES OF F AND CR, ALL OTHER PARAMETERS REMAINING THE SAME.

	CR=0.05	CR=0.10	CR=0.50	CR=1.0
F=0.05	0.7613	0.7641	0.7659	0.7041
F=0.10	0.7590	0.7642	0.7655	0.7077
F=0.25	0.7634	0.7674	**0.7754**	0.7266
F=0.50	0.7625	0.7647	0.7717	0.7453
F=1.00	0.7616	0.7577	0.7503	0.7536
F=1.50	0.7609	0.7555	0.7489	0.7562

VII. ANALYSIS OF ARTIFICIAL DATASETS

In order to understand more fully the nature of the artificial datasets that the Instance Optimization algorithms produce, we selected three datasets – specifically, "circle", "Churn" and "MAGIC", all of which IO-NB was able to significantly improve Naive Bayes on – and we executed the IO-NB algorithm using 100% of the dataset for training data (as opposed to 20%, which was used in previous experiments). All other parameters were kept the same as specified in Table I.

Unlike the previous experiments, this time we retained the artificial dataset used to train the best Naive Bayes model for further analysis.

We computed two statistics of interest from the artificial datasets, and compared these values to the same statistics computed from the real datasets. The two statistics were (i) the mean absolute correlation between predictive features and the class label, and (ii) the mean entropy per column of the datasets.

The mean absolute correlation between features/class is defined in Equation 4, where C is a correlation matrix computed from a dataset, k is the index of the class label, and n is the number of columns in the dataset.

$$MAC(C,k,n) = \sum_{i=1,i\neq k}^{n} \frac{|C_{i,k}|}{n-1} \quad (4)$$

Intuitively, this statistic measures the strength of the relationship (positive or negative) between each feature in a dataset and the class. Higher values should indicate that the features are better able to individually predict the class label; conversely low values near zero indicate that the features are less able to predict the class.

For the entropy calculation, we used a standard entropy formula available in most statistics textbooks. Entropy is calculated from frequencies, and to convert our datasets into this form, we took each column, divided the range of values for the current column into ten bins, and converted the raw data for the column into the form of a ten-bin frequency histogram. The entropy for each column was computed from this histogram, and then averaged over all the columns in the dataset.

The results of our analysis are given in Tables IX and X .

TABLE IX. MEAN ABSOLUTE CORRELATION BETWEEN FEATURES AND CLASS FOR REAL AND ARTIFICIAL DATASETS.

Dataset	Real	Artificial
circle	0.02	0.27
MAGIC	0.16	0.30
Churn	0.01	0.29

TABLE X. MEAN COLUMN ENTROPY FOR REAL AND ARTIFICIAL DATASETS.

Dataset	Real	Artificial
circle	1.76	1.60
MAGIC	1.37	1.64
Churn	0.14	0.31

We can make two interesting observations from this analysis. Firstly, the correlation between the features and the class labels in the artificial datasets is considerably higher than it is in the original datasets.

This suggests that in the artificial data, the features may be more separable than they are in the real data. In turn this may explain why Naive Bayes is frequently much improved by IO-NB: the Naive Bayes assumption that features are independent given the class appears to be more true in the artificial data than it is in the real data. This is very interesting given that in most of the datasets, there is a clear and large degree of non-separability between features. For example in the "circle" dataset, the class label (indicating whether a point is inside or outside a unit circle) is obviously a non-linear function of the two features.

The second observation is that the entropy per-column in the artificial datasets is, with the exception of the smallest dataset, increased in the artificial data. This suggests that the process of evolving the artificial data is also one of reducing the amount of redundancy in the data.

VIII. CONCLUSIONS

To conclude, we have proposed a new DE and artificial data-based meta learner for machine learning called Instance Optimization (IO). DE is used to construct an artificial dataset that is in turn used to train a classifier. In this paper the

classifiers are either Naive Bayes or a linear Support Vector Machine.

This paper has outlined the technical details involved in using DE to learn artificial datasets conforming to specific schema. A significant number of experimental results show that IO frequently outperforms equivalent classifiers trained on original training data in a standard way.

The classifiers that we tested were Naive Bayes and linear Support Vector Machines. While both of these classifiers make a number of simplifying assumptions about their training data (which may or may not be true), the result is that these algorithms produce straightforward and interpretable models.

We have also attempted to analyze the performance of IO in terms of both the algorithm's sensitivity to parameters, and the statistical characteristics of the artificial data that the algorithm produces.

It needs to be noted that in terms of runtime, IO is certainly not competitive. This is obviously because our approach requires multiple (4,000 in this paper) train/test iterations whereas the corresponding base classifiers without IO require only a single train/test iteration. We see the advantage of this method primarily in boosting the quality of simple interpretable machine learning models where this is desired.

Future work in this area could look at further analysis of and enhancements to IO. For example, our analysis in Section VII shows that artificial datasets have a significantly higher mean correlation between features and the predictive class than real datasets do. This is a very interesting observation, and it would be a fascinating experiment to include the MAC (see Equation 4) of a dataset explicitly in the cost function along with the AUROC in the hope that high quality optima will be more readily discoverable.

We are also interested in applying this method of learning classifiers to other interpretable models such as decision trees. Exploring the ease or difficulty of learning much larger datasets, for example datasets with 100 or more examples, is yet another avenue for future research.

Finally, other applications of this approach remain to be explored. For example, classifier recommendation is a problem in which a machine learning system must suggest (for a naive user and a given particular dataset) the k best classifiers for the user to try on the dataset, in order from best to worst. One problem is how to train these classifier recommendation systems, as example datasets covering all possible classifier ranking permutations may not be available. Instance Optimization could help in this case, as it should be possible to evolve artificial datasets such that the cost of a dataset corresponds to a particular ranking of a set of classifiers obtained when machine learning experiments are performed on the dataset.

In summary, Instance Optimization appears to be a promising new approach in evolutionary algorithm-based machine learning and we plan to pursue approach in the future.

REFERENCES

[1] Deng, H. and Runger, G. Feature Selection via Regularized Trees. In *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2012.

[2] Cano, J.R. and Herrera, F. and Lozano, M. Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *IEEE Transactions on Evolutionary Computation* 2003, v. 7(6), pp. 561-575.

[3] Venables W.N. and Ripley B.D. *Modern Applied Statistics with S*, Springer-Verlag, 2002.

[4] Storn R. and Price K. Differential Evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization* v. 11, pp. 341-359, 1997.

[5] Das, S. and Suganthan, P.N. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* v. 15(1), 2011, pp. 4-31.

[6] Tasoulis D.K., Pavlidis N.G., Plagianakos V.P. and Vrahatis M.N. Parallel Differential Evolution. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, 2004.

[7] John G.H., Langley P. Estimating Continuous Distributions in Bayesian Classifiers. *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, pp. 338-345, 1995.

[8] Platt J. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, 1998.

[9] Keerthi S.S., Shevade S.K., Bhattacharyya C., Murthy K.R.K. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, v. 13(3), pp. 637-649, 2001.

[10] Spackman, K. Signal detection theory: Valuable tools for evaluating inductive learning. *Proc. 6th International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann. pp. 160163, 1989.

[11] Agrawal R., Imielinski T and Swami A. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering.* v. 5(6) pp. 914-925, 1993.

[12] Triguero I., Derrac J., Garcia S and Herrera F. A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification. *IEEE Trans. on Systems, Man and Cybernetics – Part C: Applications and Reviews.* v. 42(1) pp. 86-100, 2011.

[13] Leisch F. and Dimitriadou E. *MLbench: Machine Learning Benchmark Problems.* R package version 2.1-1, 2010.

[14] Breiman, L. *Bias, variance, and arcing classifiers.* Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA, 1996.

[15] Sun, Q. and Pfahringer, B. Bagging ensemble selection. In *AI 2011: Advances in Artificial Intelligence*, pp. 251-260, Springer, 2011.