Why Classifying Search Algorithms is Essential

John R. Woodward, Jerry Swan
School of Computer Science
University of Nottingham
emails: jrw@cs.nott.ac.uk, Jerry.Swan@nottingham.ac.uk

***Abstract*—In chemistry, the periodic table of elements was a huge leap of progress. It allowed elements to be placed in a table allowing their classification. More importantly, it allowed predictions to be made about their properties, and in some cases predictions about elements which had not even been discovered at the time the periodic table was proposed. We argue that the current state of search methodologies is analogous to the state of chemistry *before* the arrival of the periodic table, and such a classification system is well overdue.**

Many machine learning research papers are currently published on the premise that the proposed algorithm does better on a particular data set than another algorithm. This approach, while it does produce better results on the given data set, does not produce an understanding of why a particular algorithm performs well on a particular problem. The No Free Lunch Theorem, while stating this is impossible over all data sets, also provides a possible framework. We state why the performance table associated with No Free Lunch (which has rows and columns similar to the periodic table), which is exactly what we are looking for, is unworkable, as problems cannot be indexed or enumerated in practice. We believe the classification of algorithms and problems is the biggest issue facing machine learning today.

Progress in science is often brought about by asking the right questions, before finding the answers, and it is this question we address in this paper. Therefore, the contribution of this paper is raising the profile of the challenge of classifying algorithms and problems. An underlying aim is to reduce the number of papers with titles of the form "Algorithm X Applied to Problem Y", or simply "A New Algorithm", as new algorithms should only be introduced with an intended class of problem instances in mind.

Keywords- search; no free lunch theorems; bias; induction; machine learning; optimization; generalization.

I. Introduction and Outline

Machine learning attempts to find an underlying rule that accounts for the observed data [1]. This has two purposes; to identify a pattern in the observed data (memorization), and to make predictions about unobserved data (generalization). While humans and machines seem to be very good at taking raw data and inducing a rule, what we have yet to achieve is to do this at the meta level of algorithms and problems themselves [14]. That is, predict which algorithms will perform well on which problems (i.e. classify problems and algorithms). We consider this to be one of the most important open, yet poorly addressed, questions in machine learning today.

The classification of algorithms and problems is synonymous with choosing the best algorithm in order to produce the best solution on the given problem. As algorithms themselves are performing a prediction task, we are just transferring the problem from the base level (concerning raw data) to the meta level (concerning problems and algorithms) [11]. Note that functions and problems are taken to mean the same, and search algorithm and algorithm are also taken to mean the same.

It is important to state the motive for wanting to classify algorithms and problems. We are interested in classifying algorithms in terms of their performance on problems in order to identify the best algorithm for a problem (this is much of the focus of machine learning, i.e. obtaining better results and comparing the performance of algorithms).

The performance table associated with the No Free Lunch theorems (NFL) [3,4,7,8] in a sense solves the problem of classification. The performance table tells us all we need to know about algorithms and problems, as it lists all algorithms against all problems in a table, and the cell at the intersection contains the result of applying that algorithm to that problem. We can make predictions about any algorithm, and make precise statements about its performance on any given problem. However there are reasons why this is not a suitable approach.

Knowing which algorithm to apply to which problems has become something of a black-art in machine learning. Assemble ten computer scientists in a room and ask them what algorithm to apply to which problem and you will most likely receive ten different responses. We do not yet have an over-arcing method which matches problems and algorithms.

The outline of the remainder of the paper is as follows. In section II we examine the impact of classification on empirical sciences. In section III we consider some issues of naively using brute force to attempt to achieve classification by blindly executing algorithms on problems, without an overall framework. In section IV we examine the use of a population table associated with NFL as a way forward. In section V we discuss a number of issues arising from attempting the goal of classification. In section VI we conclude the paper.

II. The Importance of Classification

In the three major scientific disciplines (physics, chemistry and biology), classification has played a central role in the progress of each field. The reason we go into detail here is really to emphasize the point that classification is fundamental to the progression of the sciences, and a new classification system often signals a paradigm shift.

The periodic table is a method of categorizing the chemical elements. If elements are ordered by their atomic mass one sees a periodicity of the properties as a function of atomic mass. In fact more regularity can be observed if elements are plotted by their atomic number rather than their atomic mass. The layout of the periodic table demonstrates the recurring chemical properties. At the time, predictions could be made about the existence and properties of a few undiscovered elements which would later occupy the empty locations in the table with the predicted properties.

Biological classification is the method biologists use to categorize organisms in a tree data structure. Originally, it was based on physical characteristics (phenotypes), but has been revised to reflect Darwin's principle of evolutionary ancestry (genotypes). In this case, a whale is classified as a mammal rather than as a fish, which reflects the fact that it is a mammal that has returned to the water. One may also notice that whale meat tastes more like beef than fish and a whale's tail moves vertically rather than horizontally like a fish. In principle, it may be possible to state what the properties of a hypothetical species would be, before it has even evolved, in a similar fashion to predicting properties of elements in the periodic table.

In particle physics, during the 1950s and 60s, a large number of subatomic particles were discovered, and the need for a classification system arose. The current state of classification of these elementary particles is the Standard Model. It predicted the existence of the W and Z bosons, the gluon, the top and charm quark before their existence was confirmed experimentally. It also predicts the Higgs Boson, which is the only particle predicted by the model which has not (yet?) been observed. However the model is incomplete, as it fails to take gravity into account, but can be credited with some success after making many correct predictions.

These examples were chosen as they are familiar to most readers, but also because they emphasize the underlying importance of classification. In summary, classification really is central to the empirical sciences, not only in terms of organization, but more importantly in terms of prediction.

III. Considering Classification

In this section we provide a classification table of algorithms and problems. We then draw an analogy between the process of search and multiplication and conclude this section with a thought experiment, where the performances of an algorithm are entered in a classification table automatically.

In table I we illustrate what a classification scheme may look like. The columns are different algorithms, and the rows are different problems. The idea driving classification is that the performance of one algorithm on one problem can be obtained from the table, without actually having to execute the search process. Thus when deciding between two algorithms, we can choose the better performing one in order to obtain the solution to our problem.

While there may be some benefit in applying a new algorithm to a problem, we gain little insight into the search process itself. Let us draw an analogy between trying to understand the search process, and the process of multiplication. If we have an interest in the actual result of the computation (search or multiplication), then the computation is a worthwhile investment in time, otherwise it is not.

TABLE I
A classification table showing algorithms in columns and problems rows, and the performance of that algorithm on that problem in the intersecting cell.

	algorithm-1	algorithm-2	...	algorithm-m
problem-1	1.0	2.3	...	3.6
problem-2	1.6	6.5	...	7.6
...	...	...	...	...
problem-n	3.5	8.6	...	6.5

It is very likely that someone before us has multiplied together the numbers 45 and 52, as they are relatively small numbers. (I *am* interested in this computation as it is my salary i.e. pay per week multipled by the number of weeks in the year). However, as far as I am aware, no one has multiplied the following two numbers; 37854556 and 2347337367. (I *am not* interested in this computation as both numbers were typed randomly). Would publishing a mathematical paper in a well respected peer-reviewed journal, claiming to be the first to calculate this number be a worthy article? Unquestionably the answer is no. Similarly, what do we learn after applying say an ant colony optimization algorithm to a scheduling problem (apart from knowing how well that algorithm performs on that problem)? So why do we adopt a similar attitude with research into algorithms?

Generating new algorithms, and then applying them to new problems in order to understand the working of the search process is futile without the context of a framework in which to place the results. A framework is required in which to state the relationships between algorithms, problems and performance.

Imagine the following thought experiment. We write a master-algorithm which generates a set of algorithms and a set of problems, and then applies each algorithm to each problem, completing the entries, row by row and column by column, of a population table I. But would the algorithm learn to apply which algorithm to which problem in general? In other words, would it be able to tell us what is the best algorithm for a new problem, without actually executing it beforehand? A human observer may uncover some patterns, but the master-algorithm lacks an introspective meta-level where it can draw conclusions about the base layer (algorithms and problems). It is only learning at the base-level and not at the meta-level.

IV. No Free Lunch Theorems

In this section we introduce the population table [3,4], and then explain the difficulties using it for classification. NFL states that all algorithms perform equally over the set of all problems so it is not surprising that it can be associated with a framework which can be used for classification. That is, it offers a framework in which all problems and all search algorithms can be considered.

A. The No Free Lunch Framework

Let X and Y be finite sets and $f : X \to Y$, where $y_i \equiv f(x_i)$. The size of X is $|X|$ and the size of Y is $|Y|$. There are $|Y|^{|X|}$ possible functions. We can represent functions as

TABLE II
THE POPULATION TABLE FOR ALL FUNCTIONS WITH A DOMAIN $\{x_1, x_2\}$ AND RANGE $\{y_1, y_2\}$. THE COLUMNS ARE LABELED WITH ALL POSSIBLE FUNCTIONS (4 IN THIS CASE). THE ROWS ARE LABELED WITH ALL POSSIBLE SEARCH OPERATORS (2 IN THIS CASE). ENTRIES IN THIS TABLE CORRESPOND TO THE PERFORMANCE VECTOR FOR THE GIVEN FUNCTION AND SEARCH OPERATOR.

	$f_{<y_1,y_2>}$	$f_{<y_1,y_1>}$	$f_{<y_2,y_2>}$	$f_{<y_2,y_1>}$
$< x_1, x_2 >$	$< y_1, y_2 >$	$< y_1, y_1 >$	$< y_2, y_2 >$	$< y_2, y_1 >$
$< x_2, x_1 >$	$< y_2, y_1 >$	$< y_1, y_1 >$	$< y_2, y_2 >$	$< y_1, y_2 >$

sequence of ordered pairs $(x_i, f(x_i))$, and if we define an order on the items in X, it is only necessary to list the function values in order to define the funtion. e.g. the function with $f(x_1) = y_2$ and $f(x_2) = y_1$ (i.e. $f_{<y_2,y_1>}$), can be written as the list $< y_2, y_1 >$. An *algorithm V,* is an ordered sequence of points in X, $V \equiv \langle x_a, x_b, \ldots, x_i, \ldots, x_n \rangle$ that lists all points in X and has length $|X|$. It is assumed that no point is revisited. The ith element corresponds to the ith point visited in X. There are $|X|!$ distinct algorithms. A given algorithm and function will produce a corresponding path in Y. Let us call this sequence of points in Y a *performance vector*.

A population table has rows which are labeled with all $|X|!$ distinct algorithms, and columns are labeled by all $|Y|^{|X|}$ possible functions [3,4]. Each element in the table contains the performance vector generated by the corresponding search vector (row) and function (column). For example, the function $f_{<y_2,y_1>}$ (4th column) and search vector $< x_2, x_1 >$ (2nd row) generated the performance vector $< y_1, y_2 >$ (see table IV-A).

In this framework, each problem and algorithm can be enumerated (indexed). Given that we can do this, it is easy to make predictions about performance of a particular algorithm on a particular problem, without actually executing the algorithm on the problem. We just look it up in the cell with the corresponding algorithm and problem. The performance table is exactly what we require.

Unfortunately, the picture is far from as simple as this. Firstly, in order to index a problem precisely, we would have to know its value at every point, which is impractical. Secondly, an algorithm is viewed as a permutation of items from the function's domain. Therefore indexing an algorithm would require actually executing it. Thus in many cases indexing problems and algorithms is out of reach.

What is apparent from this framework is that it does not make sense to say a function is "hard to search", or that a particular algorithm is "good at search". We must talk about an algorithm in the context of a problem. One may ask which task is easier, inserting a nail or a screw into a plank of wood? This (ill-defined) question depends on the context. If a hammer is available, then the task of inserting the nail is easy. However if a screwdriver is available the converse is true. As English [6] comments *"It is much as though the community is insisting that tools be (all purpose) Swiss army knives instead of hammers and screwdrivers"*.

V. DISCUSSION

While we currently use machine learning algorithms to make predictions about unseen data on a given problem, what we are not currently doing is using machine learning to make predictions about which algorithm performs best on a given problem. This is the same problem (i.e. classification) but at the meta-level instead of the base-level. Just as we have been using machine learning techniques to identify hidden patterns in raw data, it may now be employed in the introspective way of looking for patterns in the performance on problems of the algorithms themselves. This is also related to the issue regarding the *necessity* of meta-bias in search algorithms [14].

To re-emphasized this point, supervised learning is a two stage process consisting of training and testing. We train an algorithm on a set of data points drawn from a *problem*, and then test in on a new set of data points drawn from the *same problem*. This occurs at the base level, but the same principle applies at the meta level. We should also train an algorithm on a set of problems drawn from a *problem class*, and test it on problems drawn from the *same problem class*. It does not make sense to test an algorithm on a random collection of problems as many research papers do.

We can reiterate this point in terms of benchmarking, the practice of selecting a set of problems on which to demonstrate the prowess of an algorithm [10]. Benchmarks need to define a probability distribution over a set of problems (either implicitly or explicitly), and should not contain just a single problem (which is called the one-shot-scenario in [9]). Researchers should not test their algorithms on a single problem, or a general broad range of problems, but on a specific narrow set of problems. In light of the fact that a problem should always have a context (i.e. a probability distribution over problems to which the problem belongs), this renders the UCI Machine Learning Repository data sets (http://archive.ics.uci.edu/ml/), often used to benchmarking, as almost useless, as this is an unconnected collection of problems. If a set of problems is to be useful, the individual problems should be drawn from the same probability distribution.

Algorithms are becoming more sophisticated and the problems tackled are becoming more challenging, making it more difficult to decide which algorithm to use on which problem. It has been said that industry uses simple methods to solve complex problems, while academia uses complex methods to solve simple problems. Perhaps classification is the bottleneck in this situation, where we can learn to apply the appropriate algorithm to the appropriate problem. A related point is that just as no one person understands all the components of a large piece of complex software (constructed by millions of man hours), it may be the case that no one person knows all the intricacies of a given algorithm (e.g. selection criteria, ranking scores, mutation processes, parallelization techniques, decomposition mechanisms, ...). In other words, algorithms and problems are complex pieces of software and need to be managed as such. So now may be a good time to hand over the management (i.e. classification) of organizing these algorithms to an automated heuristic book-keeping system.

Earlier we used a table to illustrate classification (table II). Ultimately we do not know what data structure will work best to facilitate the classification of search algorithms and problems. It may be that a table which approximates to a performance table is promising. A tree data structure may be useful, showing the ancestry of different algorithms (e.g. a hill climber splits into a simulated annealing and the great deluge algorithms). Or a directed acyclic graph may best reflect the fusing of algorithms of different ancestry into a hybrid variants. Finally a graph data structure may be appropriate for the classification of algorithms and problems, as both algorithms and problems can be expressed in terms of graphs.

Possible approaches to the classification issue involve using a machine learning methodology to predict which algorithms preform well on which problems. Random sampling of a problem may give some information about the class of problem it belongs to, but unbiased sampling does not necessarily reflect the biased sampling of an algorithm. Landmarking overcomes this to some extend by sending out a set of scouting algorithms on a reconnaissance mission [12]. Some sort of feature selection may also be appropriate. It may be that data mining may be employed in order to discover patterns in the data and decide what class a problem belongs to. It may be that simple historical labeling of problems according to which algorithms have performed well in the past is a fruitful direction [13], or a case based reasoning system. However there is the difficulty of defining a suitable metric between algorithms. Classification is similar to learning to learn [11], and many combinations of learners have been used at the base and meta-levels.

Instead of deciding which algorithm is suitable for a problem class, an alternative approach is to evolve an algorithm for the problem class. In [5] on-line bin packing algorithms are evolved, for different problem classes. The problem classes are defined by probability distributions over the sizes of items to be packed, defining a hierarchy of problem classes, giving rise to a hierarchy of bin packing heuristics. They find general heuristics perform robustly well over all problem classes, while more specialized heuristics do better on the more specialized problem classes which they were intended for, and poorly on specialized classes they were not intended for (failing catastrophically in some case). In other words, the narrower the problem class, the more specialized the evolved heuristic. Thus the heuristics show a range of behavior from general to specific, as one would expect. They are in the strong position of being able to make precise claims about the performance of their heuristics on new problems (or problem classes) not seen before.

VI. Conclusions

Classification is vitally important in the empirical sciences, and machine learning, which is also inherently empirical, is no exception. It is hard to invent an effective classification scheme for algorithms and problems, and would represent a major milestone in computer science as it has done in the other sciences. This difficulty is perhaps why researchers have been shy of such a monumental task. Flooding the literature with new algorithms will not help us understand how existing algorithms work, and therefore we should resist the temptation of generating new algorithms (which is easy) and concentrate on the more daunting task of classification (which is hard). We therefore emphasize that the goal of classification in machine learning is of paramount importance to the success of the field.

The problems we are typically interested in tackling are NP-hard and the classification of problems is possibly an NP-hard problem itself. Real-world problems cannot be indexed precisely for the purposes of a classification table. In addition the number of algorithms available is also undergoing a combinatorial explosion, as more novel algorithms saturate the literature. Thus we will be forced to resort to employing heuristic based methods to manage the task of classification.

Classification, meta-learning and benchmarking are intrinsically linked. Benchmarking should involve selecting a probability distribution over a set of problems, defining the problem class. In order to learn about a problem class a learner must have a meta-level in order to alter its bias [11]. (i.e. shift its own bias towards that defined by the probability distribution of the problem class). As classification is ultimately concerned with deciding which algorithm does well which problem class, classification is therefore connected with benchmarking. A problem class corresponds to a representative benchmark set. If an algorithm is intended to be used with more than a single problem (as almost all algorithms are), then it *must* have a meta layer where it can alter its own bias to match that of the problem class.

It does not make sense to talk about an algorithm in absence of the context of a class of problems. These problems must be related in some way if an algorithm is to demonstrate superior performance across a class of problem. Thus a problem class defines a niche in which a search algorithm fits, and therefore goes hand in hand with classification.

References

[1] T. M. Mitchell. Machine Learning. McGraw-Hill 1997.

[2] T. M. Mitchell, The need for biases in learning generalizations (Rutgers University, Department of Computer Science, Technical Report CBM-TR-117, 1980).

[3] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In proceedings of the Genetic and Evolutionary Computation Conference, 565-570, California, USA, 7-11 July 2001. Morgan Kaufmann.

[4] C. Schumacher. Fundamental Limitations of Search. PhD thesis, University of Tennessee, Department of Computer Sciences, Knoxville, TN, 2000.

[5] E. K. Burke, J. Woodward, M. Hyde, G. Kendall, Automatic heuristic generation with genetic programming: Evolving a Jack of all trades or a master of one. Genetic and Evolutionary Computation Conference, GECCO 2007.

[6] T. M. English, Evaluation of Evolutionary and Genetic Optimizers: No Free Lunch 1996 MIT Press

[7] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, 1995

[8] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67-82, April 1997.

[9] S. Droste, T. Jansen, I. Wegener. Perhaps Not a Free Lunch But At Least a Free Appetizer (1998)

[10] J. Brownlee. A Note on Research Methodology and Benchmarking Optimization Algorithms [Technical Report]. Victoria, Australia: Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology; 2007 Jan; Technical Report ID: 070125.

[11] S. Thrun and L. Pratt, Learning To Learn, S. Thrun and L. Pratt, ed., Kluwer Academic Publishers, 1998, 354 pages.

[12] B. Pfahringer, H. Bensusan, C. Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. Proceedings of the Seventeenth International Conference on Machine Learning 2000

[13] J. Vanschoren, B. Pfahringer, G. Holmes, Learning from the Past with Experiment Databases. Lecture notes in artificial intelligence vol:5351 pages:485-496

[14] John R. Woodward, The Necessity of Meta Bias in Search Algorithms, 2010 International Conference on Computational Intelligence and Software Engineering (CiSE 2010) http://www.ciseng.org/2010 Wuhan, China December 10-12, 2010