PREPRINT. Proc. of the ECML'98 Workshop on Upgrading Learning to the Meta-Level: Model Selection and Data Transformation, pages 54-65, 1998.

Model selection using measure functions

Arne Andersson[1], Paul Davidsson[2], and Johan Lindén[1]

[1] Dept. of Computer Science, Lund University, Box 118, S–221 00 Lund, Sweden.

[2] Dept. of Computer Science, University of Karlskrona/Ronneby, S–372 25 Ronneby, Sweden.

arne@dna.lth.se, pdv@ide.hk-r.se, johan@dna.lth.se

Abstract

The concept of measure functions for generalization performance is suggested. This concept provides an alternative way of selecting and evaluating learned models (classifiers). In addition, it makes it possible to state a learning problem as a computational problem. The the known prior (meta-)knowledge about the problem domain is captured in a measure function that, to each possible combination of a training set and a classifier, assigns a value describing how good the classifier is. The computational problem is then to find a classifier maximizing the measure function. We argue that measure functions are of great value for practical applications. Besides of being a tool for model selection, they: (i) force us to make explicit the relevant prior knowledge about the learning problem at hand, (ii) provide a deeper understanding of existing algorithms, and (iii) help us in the construction of problem-specific algorithms. We illustrate the last point by suggesting a novel algorithm based on incremental search for a classifier that optimizes a given measure function.

1 Introduction

In this work, we suggest a new approach to model selection and evaluation. Today, most methods for evaluating the quality of a learned model (classifier) is based on some kind of cross-validation [6]. However, we argue that it is possible to make evaluations that take into account other important aspects of the model than just classification accuracy on a few instances. Our approach is based on measuring explicit properties of the learned model rather than properties of the algorithm that produced the model. Therefore, in contrast to for example Nakhaeizadeh and Schnabl [9], we pay no attention to properties such as the employed algorithm's time and space complexity.

For each possible combination of a training set and a classifier, a *measure function* assigns a value describing how good the classifier is. Measure functions have a number of favorable properties from both a theoretical and a practical point of view. They provide a complementary tool for selecting models as well as assistance in designing new learning algorithms.

It has been known for a long time, see for example the "no free lunch" theorems (cf. [16, 17]), that the task of computing a good classifier from a data set is not easily defined as a simple computational problem. One purpose of the concept of measure functions is to remedy this situation. As a consequence, we get an explicit distinction between problem formulation, i.e., specifying the measure function, and problem solving, i.e., finding a classifier maximizing the

measure function. By making this distinction, we can isolate the meta-knowledge necessary for model selection from the details of the learning algorithms.

Measure-like criteria have been successfully used for understanding and solving other problems in several scientific areas. One example is model-order selection in linear prediction, where the order or dimension of the predictor, i.e., the number of previous values used to estimate the next value, is often chosen according to a measure-like criterion, such as Akaike's information-theoretic criteria [1], and Parzen's criterion autoregressive transfer [10].

In the next section we introduce the concept of measure functions for generalization performance. This is followed by a discussion of the relation between commonly used heuristics and measure functions. Next, we present a case study where a simple measure function is used for evaluating the models learned by some popular learning algorithms. As a result, we provide clear indications that it is at least as important to spend time on tuning one algorithm as it is to spend time on choosing between different types of algorithms. Finally, we illustrate that measure functions are helpful when designing new algorithms. This is done by presenting a simple algorithm based on incremental search for a model that optimizes a given measure function. Such an algorithm allows much flexibility as well as clear specification of used biases.

2 Measure functions for generalization

For simplicity and to ease comparison with previous work we assume the universe to be a finite set of instances and categories. However, the idea of a measure function is not dependent of a finite universe.

Let U be the set of all possible instances, ff a set of instances ($\mathrm{ff} \subset \mathrm{U}$), and c_{ff} a set of pairs such that each instance in ff is labeled with a category. A generalization task, $\emptyset$, is defined by a pair $\langle\, \mathrm{U}; \mathrm{c}_{\mathrm{ff}} \rangle$. In what follows we will by T refer to the set of all possible generalization tasks (given U and the categories present, K).[1]

A generalization algorithm is an algorithm that given a generalization task produces a classification of the entire universe, c_{U}, i.e., a generalization (classifier). Let C be the set of possible generalizations, $\mathrm{C} = \mathrm{U} \times \mathrm{K}$. Thus, a generalization algorithm computes a function $\mathrm{G} : \mathrm{T} \rightarrow \mathrm{C}$.

We are now ready to introduce the measure function for generalization:

Definition 1 *A measure function for generalization performance,* f*, is a function that to each* $\langle\, \emptyset; \mathrm{c}_{\mathrm{U}} \rangle$ *assigns a value describing how good the generalization is, i.e,* $\mathrm{f} : \mathrm{T} \times \mathrm{C} \rightarrow \mathcal{R}$.

Typically we want to use measure functions for judging how good a generalization is with respect to a certain training set:

Definition 2 *The generalization performance on a particular generalization task* $\emptyset$ *for a generalization algorithm* G *is defined by* $\mathrm{f}.\emptyset; \mathrm{G}.\emptyset\,//$.

We may now describe generalization as a computational problem: Given a generalization task $\emptyset$, and a measure function f, produce a generalization that maximizes f.

One feature of our notion of measure functions is that it helps in simplifying and clarifying the discussion on when generalization is meaningful (cf. [16, 17, 13]). In short, it can be proved that once a non-trivial measure function is defined, some algorithms are better than others, see [3]. By a trivial measure function we mean one that gives constant output regardless of input.

[1] To allow noisy and inconsistent training sets, ff can be made a multi-set. Examples of different magnitude (appearing many times in training) and multiple identical examples with incoherent categories can be represented using a multi-set. This has no effect on our theorems.

3 Analyzing heuristics in terms of measures

Armed with the concept of measure functions, we are in a better position to analyze and compare existing learning algorithms in a way that goes beyond purely representational issues. For example, characterizing algorithms in terms of which measure function they maximize seems to be a plausible way to identify their strengths and weaknesses as well as the regions of expertize for different biases [5].

If we try to describe the implicit measure functions optimized by the most popular algorithms used today, we see that they basically are composed of some, or all, of the following three well-known heuristics:

Subset-fit (on the training set) – the training instances should be classified correctly

Similarity – similar instances should be classified similarly

Simplicity – the partitioning of the universe should be as simple as possible.

As these heuristics typically counteract, a measure function must balance a trade-off between them. However, we begin with discussing them in isolation.

Subset-fit is the currently most used method to *evaluate* learning algorithms. We simply take a number of instances where the correct classification is assumed to be known and let the algorithm try to classify these. The value of the measure function is proportional to the number of correct classifications made by the classifier. In some cases, like when the cost of misclassification is high, we may use different weights on different categories.

It is possible to choose the subset in a number of ways; two basic alternatives are: (i) the training set and (ii) subsets of instances not present in the training set, i.e., a validation set or an off-training set. A special case of a subset-fit measure is the *total-fit* measure whose value is proportional to the number of correct classifications on the entire universe (assuming that a correct classification of the universe exists). In fact, we can see subset-fit measures as an attempt to approximate the total-fit measure. There are a number of other variants of subset-fit measures. For instance, the measure may also be weighted with probabilities. Another variant is the widely used *cross-validation* (CV) method. CV performs a sequence of subset-fit evaluations and then computes the average of these. A disadvantage with subset-fit measure functions, is that they often are of little help in *designing* an algorithm. We can, of course, use subset-fit in an algorithm of a higher order that takes a number of generalization algorithms, compares their generalizations by means of subset-fit on a validation set, and then chooses the generalization that gives the highest grades. Or, similarly, we can use subset-fit measures for tuning the parameters of a single algorithm (cf. VSM [7]).

An intuitive property of good generalization is that "similar" instances should be classified similarly. A problem with similarity is that there is no objective way of measuring it. A distance measure is needed and how distances should be measured varies a lot between applications. An often used heuristic is that, given two (or more) clusters of instances of different categories, the decision border(s) should be centered between the clusters rather than being placed closer to one of the clusters. As a result, query instances that reside in the area between the clusters will be classified as belonging to the category whose instances it is most similar to.

The use of simplicity to define a measure function is an application of Ockham's razor, a principle that has been successful in many scientific fields. An important reason for using simplicity as a heuristic is to reduce over-fitting to the training set. A problem with simplicity, just as with similarity, is that there is no objective method to measure it. Often it is measured with respect to a particular representation scheme, e.g., the size of a decision tree. Note that this

also holds for Kolmogorov complexity, which use Turing machines as its representation scheme. A measure function, on the other hand, should be independent of the hypothesis language of learning algorithms to be useful.

4 A case study

We now illustrate how a simple algorithm-independent measure function can be used for selecting which of a number of popular algorithms for a particular application. The measure function we use does not correspond to any known algorithm, rather it is an attempt to capture in an algorithm-independent way how we want an algorithm to behave in this application. Note, however, that this is only one of an infinite number of possible measure functions. Three common classes of algorithms will be compared, but first let us briefly analyze what heuristics they use.[2]

Decision tree induction algorithms *Subset-fit and simplicity:* As an algorithm-specific measure of simplicity it is common to use the size of the decision tree, i.e., the number of nodes. Different decision tree algorithms balance the trade-off between subset-fit on the training set and simplicity differently: ID3 [11] gives priority to subset-fit (i.e., it tries to create the simplest possible tree consistent with the training examples) whereas pruning algorithms such as C4.5 [12] tries to balance the trade-off (i.e., it tries to create an even simpler tree that do not have to be consistent with the training examples).

Similarity: When dealing with numeric features, a similarity criterion is typically taken into consideration when selecting cut-points. The most used approach is to choose cut-points that lie centered between training instances of different categories.

The backpropagation algorithm *Subset-fit:* Roughly speaking, the backpropagation algorithm [15] tries to optimize the subset-fit measure function defined by the training set. The generalization is created incrementally in small steps by an attempt to minimize an error function.

Simplicity: A problem with the plain backpropagation algorithm is that the error function does not provide any penalty for over-fitting. The algorithm will continue to adjust its weights until the error function is minimized, possibly resulting in an unwanted wealth of detail. To avoid over-fitting, it needs to be combined with other strategies such as (i) Decrease the number of neurons in order to get simple partitionings. (ii) Rely on the fact that the backpropagation algorithm slowly creates a more and more complex generalization and stop the process according to some criterion, typically a subset-fit measure defined by some validation set. Note that these strategies are not included in the backpropagation algorithm, they are applied at a meta-level.

Similarity: The sigmoid functions used in the neurons reward not only correct classification of instances, but it does also reward proper distances; a correctly classified instance should be far from the hyperplane represented by the neuron. In this way, the decision borders represented by neurons have a tendency to place themselves in the middle between clusters of different categories.

Nearest neighbor algorithms This class of algorithms [4] is clearly based on similarity. A method to avoid over-fitting is to look at the k nearest neighbors rather than only looking at one neighbor. Larger values of k has a tendency to generate simpler decision borders, the number of isolated regions becomes smaller and sharp corners have a tendency to be smoothed out.

[2] We here concentrate on the main heuristics on which the algorithms are founded, rather than on specific details of the algorithms. Take for instance the ID3 algorithm [11], the intention is to compute the simplest decision tree consistent with the training examples. However, as this is computationally intractable, ID3 employs the concept of information gain to construct a reasonably small tree.

4.1 An abstract measure function

Let us first suggest a very general measure function. We use the same notation as before, i.e., c_{ff} denotes the training set and c_U denotes the classifier produced by the algorithm. A general measure function can be defined as follows:

$$a_0 \frac{|c_{ff} \cap c_U|}{|c_{ff}|} + a_1 \mathrm{simi}(c_{ff}, c_U) + a_2 \mathrm{simp}(c_U) \quad (1)$$

The first term corresponds to subset-fit on the training set, the function simi specifies the similarity aspect, and simp computes simplicity given the partitioning of the instance space. These functions have problem-specific definitions and the trade-off between the three components can be balanced by problem-specific constants (a_0, a_1, and a_2).

Note that, by choosing different functions (simi and simp) and constants, we are able to approximate the measures corresponding to the algorithms discussed in the last section (and many other learning algorithms). Take ID3 (with numerical features) for example, simi should be a function that favors decision border segments that are centered between the closest training instances on each side of the border segment, and simp should favor partitionings that have few rectilinear plane segments.

4.2 An example measure function

Let us now make the measure function (1) more concrete by specifying the functions simi and simp. We would like to stress that this is just an example, invented for a particular application. The point is not to present the ultimate measure function, suitable for all applications. We do not believe that such a measure function exist.

We choose to express similarity in terms of distances between training instances and decision borders. We take on the heuristics that correctly classified instances should preferably reside at "safe" distance far from decision borders. For misclassified instances, the reverse should hold; instances should be close to the border of a region containing its own class. Let d_i be the distance between instance x_i and its closest decision border. Assume that we measure distance in such a way that d_i is positive if x_i is correctly classified and negative otherwise. The chosen heuristics can then be expressed by letting x_i's contribution to simi be an increasing function of d_i. Furthermore, it seems reasonable that the area in the close neighborhood of a border is the most critical area; this can be captured by letting the function's derivative be steep near zero. Instances very far from a border should hardly be affected at all if the border is slightly adjusted; this is captured by letting the function be asymptotically constant for large negative and positive values. Hence, a plausible choice would be to use a sigmoid function. However, in some cases where much classification noise is expected, we may want the similarity measure to pay less attention to the misclassified instances. For this reason, we split simi in two parts that can be weighted differently. If x_i is correctly classified, we use $1 - 1/(2^{b \cdot d_i})$ and if x_i is misclassified, we use $1/(2^{b \cdot d_i}) - 1$; the constant b is a parameter to tune distance-dependency. Let R denote the set of correctly classified training instances (i.e., $c_{ff} \cap c_U$). The total simi function will then be

$$\frac{k_1 \sum_{\forall x_i \in R} \left(1 - \frac{1}{2^{b \cdot d_i}}\right) + k_2 \sum_{\forall x_i \in c_{ff} \setminus R} \left(\frac{1}{2^{b \cdot d_i}} - 1\right)}{|c_{ff}|}$$

Then, if we wish to pay less attention to misclassified instances, we can choose $k_2 < k_1$.

Next, we turn to the function simp. Assuming numeric features, we can measure simplicity by measuring the total size of the decision borders, normalized in some suitable way. (For example, if the universe is 3-dimensional we use the total area of the decision borders.)

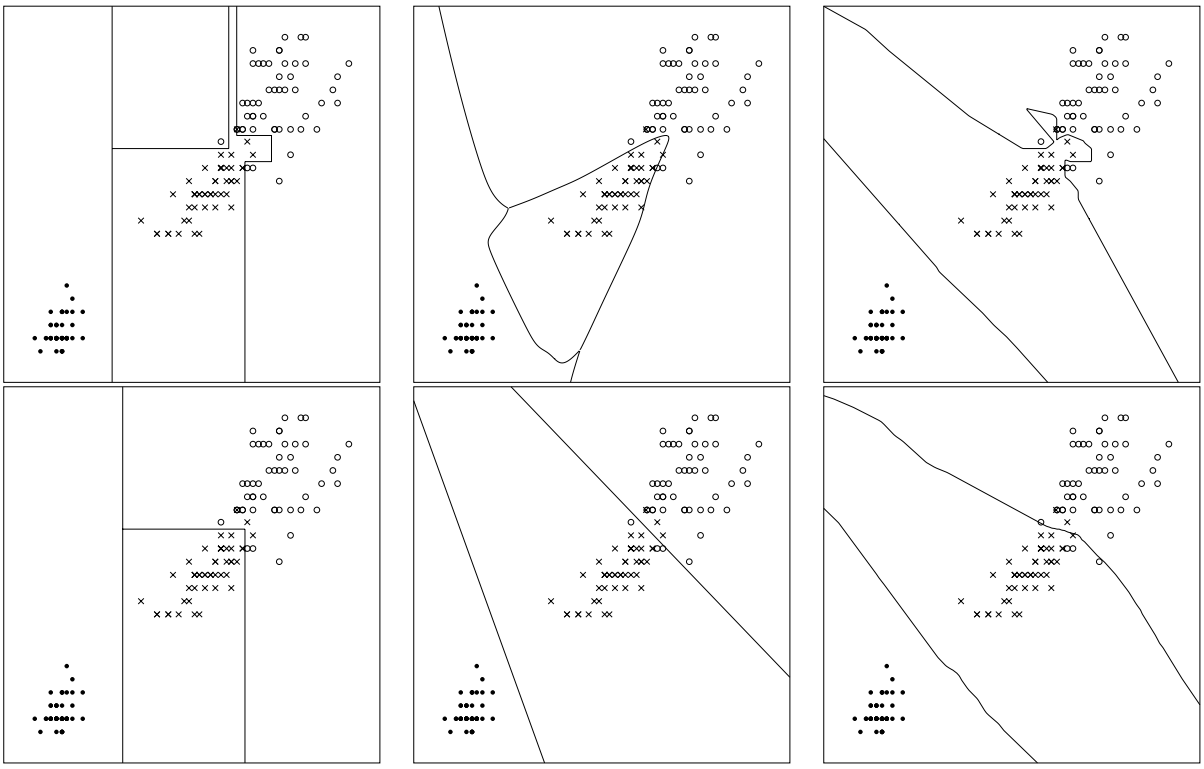

Figure 1: *Left:* Decision tree. The upper version is a tree induced by the ID3 algorithm and the lower version is a pruned tree, taken from [8, page 10]. *Middle:* Backpropagation. The upper version use 30 hidden nodes and 26 500 training epochs; the lower used 2 hidden nodes and 20 000 epochs. *Right:* Nearest neighbor. The upper version is 1-nearest neighbor. The lower version use 10 neighbors.

As the simplicity measure, we use $-L$, where L denotes the total size of the decision borders.

We can now define a measure function as follows:

$$a_0 \frac{|R|}{|c_{ff}|} + a_1 \frac{k_1 \sum_{\forall x_i \in R} \cdot 1 - \frac{1}{2^b - 1} / + k_2 \sum_{\forall x_i \in \cdot c_{ff} \setminus R} / \cdot \frac{1}{2^b - 1} - 1 /}{|c_{ff}|} - a_2 L$$

Here we mainly considered domains with numerical features where it is possible to view measure functions geometrically. However, the definition of measure functions suits nominal or logical features just as well. The measures themselves, on the other hand, must sometimes be defined differently. For instance, rather than being measured by the size of decision borders, simplicity can be measured by the minimal description length [14] of the generalization. Similarity, on the other hand, can be measured in terms of Hamming distances.

4.3 Model evaluation and selection

We have used the well-known Iris data base [2]. For the sake of presentation, we only used two of the four dimensions (petal length and width). In one case, where instances of different categories coincide, we made a small perturbation. (This was made in order for the ID3 algorithm to have consistent input.)

We have made the following choices: (i) We normalize each dimension so that all features in the training set have a value between 0 and 1. (ii) The normalization constant b for the simi

	$\frac{\|R\|}{\|cff\|}$	$\frac{\sum_{\forall x_i \in R} 1 - \frac{1}{2} d_i}{\|cff\|}$	$\frac{\sum_{\forall x_i \in cff \setminus R} 1 - \frac{1}{2} d_i}{\|cff\|}$	L	measure
ID3	1.000	0.586	0	3.406	1.208
pruned tree	0.980	0.682	0.005	2.325	1.260
backprop, 30 nodes	0.987	0.599	0.002	2.802	1.215
backprop, 2 nodes	0.960	0.703	0.008	2.516	1.245
1-nearest neighbor	1.000	0.679	0	3.330	1.256
10-nearest neighbor	0.967	0.740	0.009	2.697	1.265

Table 1: Subset-fit, similarity, and simplicity, as well as the resulting measure for our choice of parameters.

function is chosen as $\sqrt{150}$. The intuition behind this is that there are 150 training instances and if these instances were spread out evenly on a grid, the distance between two instances would be $1/\sqrt{150}$. Using this distance as a unit distance gives a reasonable sigmoid-like behavior of the simi function. (iii) The total length of the decision borders is measured in a window containing the bounding box of the training set and a surrounding 10% margin.

We apply our measure to the three classes of algorithms discussed above. For each class we have two versions: one that tends to over-fit the training set, and one that includes a non-over-fitting criterion. The induced generalizations are depicted in Figure 1.

As our measure is parameterized, the choice of parameters will govern which classifier is "best". We only give one example, where the parameters are set according to some simple rules of thumb and with a particular application in mind. These rules are chosen so that each of the three components has approximately the same effect on the measure. We chose $a_0 = 1$, $a_1 = 1$, $k_1 = k_2 = 0{:}5$ and $a_2 = 0{:}025$. The motivation is as follows: (i) We set $a_0 = 1$. (ii) A reasonable choice is to give the same weight to similarity and subset-fit. The similarity measure lies somewhere between -1 and 1. Thus, if we set $a_0 = 1$, we should choose half the weight for the two components of the the simi function, i.e. $a_1 = 1$, $k_1 = k_2 = 0{:}5$. (iii) In order to find a suitable value of a_2, we compare a reasonable tolerance in subset-fit with a reasonable variation in length. In this application, the variance in the subset-fit measure should be expected to be small, say that a reasonable algorithm should have a subset-fit measure between 0.95 and 1. To determine a reasonable variation in length, we note that the total length of two vertical lines separating the three classes would be $2{:}4$. The major part of the decision borders passes through "uninteresting areas". The length of the borders in these areas dominates the simplicity measure, this indicates that we should be quite tolerant to long borders; say that we expect a reasonable algorithm to have border length between 2 and 4. Hence, a variance of 0.05 for subset-fit should match a variance of 2 for simplicity. This gives $a_2 = 0{:}025$. Although we here try to argue that the values of a_0-a_2 are reasonable, we do not claim to prove this. Ideally, the values should be determined through careful analysis of the characteristics of the application at hand. In Table 1, we give the three terms subset-fit, similarity, and simplicity, as well as the resulting measure for our choice of parameters.

4.4 Observations from the experiment

Our measure is algorithm-independent, yet the table indicates that it captures quite well the properties that algorithm designers strive for when applying heuristics to find a proper trade-off between learning the training set and over-fitting. In this particular case, the 10-nearest neighbor

algorithm gave the highest score. Now, can we draw the conclusion that it is the best algorithm? No, of course not. But given the computational problem defined by the measure function and the training set, it provides the best solution of the six algorithms. For other applications, corresponding to other measure functions (and training sets), we expect different results. The main point here is that by using measure functions rather than just cross-validation, we are able to make more sophisticated evaluations, taking into account other important aspects than just classification accuracy on a few instances.

However, more interestingly the experiment indicates that, although the three types of algorithms tested come from different traditions, the difference between their generalizations is smaller than the difference between the generalizations computed by different versions of the same algorithm. This provides a good illustration of the fact that *it is at least as important to spend computational power on tuning one algorithm as it is to spend the power on choosing between different types of algorithms.* Thus, one should spend time to try to find the appropriate degree of pruning in decision tree algorithms, number of nodes in a neural network, or k in k-nearest neighbor.

It is worth mentioning that by adding one more node to the decision tree induced by ID3, namely a node that "cuts off" the thin vertical segment in the picture, the measure would increase significantly, from 1.208 to 1.242. The resulting partition would also "look" simpler, although the tree would be larger and hence more "complex".

5 Measure-based algorithms

The concept of measure functions immediately suggests new classes of generalization algorithms. As an example, consider the following approach: Start with some initial decision space and make incremental changes. using the measure function as evaluation criterion. In this way, we can implement an optimization strategy, such as hill climbing. A major advantage with this kind of algorithms is that by choosing an appropriate measure function we can explicitly specify the generalization behavior we are striving for.

To illustrate this, we have implemented a very simple measure-based algorithm that, given a data set and a measure function, searches for a good partitioning of the instance space:

1. Let the entire universe be classified as one of the categories.
2. Pick at random some geometrical object which is placed in the space and "painted" by a randomly chosen category. This gives a new generalization.
3. Compute the measure for the new generalization. If the measure was improved over the previous one, keep the new generalization, otherwise retain the previous one.
4. Go back to 2. Stop when no improvement has been made during several iterations.
5. Redo the whole process a number of times and keep the result with the best measure.

In Step 1, we could have used some other algorithm to generate the initial partitioning. This would probably speed up the process. The last step is not necessary; it is just a simple way to avoid ending up at a bad local maximum. In our example implementation, we let the randomly chosen geometrical objects be just rectangles of varying size, either axis-parallel or rotated 45 degrees. To make the algorithm run faster, some simple heuristics is applied; after a new geometrical object has been successfully added, the algorithm tries adding similar objects at approximately the same position.

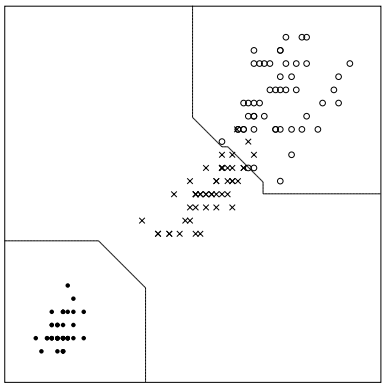

Figure 2: The generalization produced by a simple measure-based algorithm.

Although the employed measure (i.e., the one discussed above) is quite unsophisticated and the method for generating candidate generalizations is very rough, this algorithm quickly finds a reasonable generalization for the Iris data set (see Figure 2). With respect to the measure, it also gives a "better" generalization than the algorithms evaluated above. The one shown in the figure gets the value 1.288 (the four terms are 0.973, 0.731, 0.007, and 1.891). With a more sophisticated algorithm, a higher value would have been achieved.

It is important to note that measure-based generalization algorithms of the kind described here are very general in the sense that by choosing an appropriate measure function, we can achieve almost any generalization behavior. In principle, we are able to imitate the behavior of known algorithms, and more interestingly, imitate behavior of unknown algorithms. There is much more to be said about measure-based algorithms, but this is not the place for describing the behavior of particular algorithms in detail. Full evaluation of measure-based algorithms will be presented elsewhere. We have just illustrated the possibility with this example.

6 Concluding remarks

6.1 Measure functions vs. cross-validation

Measures functions and statistical methods, such as cross-validation, are complementary. They can be used separately or in combination.

When using cross-validation, an algorithm is assumed to give high accuracy on the entire universe if it shows good statistical behavior on available data. In this way, cross-validation performs a *quantitative* analysis. The result of such analysis is often sound, but it is easy to generate counter-examples where cross-validation will fail. Hence, cross-validation in itself gives no guarantee that the output of the chosen algorithm is "sound".

On the other hand, using a measure function we can guarantee certain *qualitative* properties of the produced generalization(s). Therefore, we have a mechanism to ensure that the outcome of an algorithm will never be disastrous.

Cross-validation and measure functions can be combined in several ways. For example, in algorithm selection we can apply both methods and pick the algorithm which shows the best combined quality.

6.2 Final comments

The application and selection of classification algorithms is today often guided by rules of thumb, educated guesses, and hearsay. At best, the problem to be solved is analyzed and if necessary a problem-specific algorithm is constructed. However, the most common approach is to compare a number of algorithms, e.g., by means of cross-validation and select the best one for the task. Although the latter approaches are on the right track, we believe that there is a strong need for more robust methods for telling what classifier should be applied to which problem. We propose solving a problem by establishing a measure function suitable for the problem at hand and then choosing an algorithm that aims to implement this measure. The difference between choosing a suitable algorithm and choosing a suitable measure may seem subtle, but we believe it is important—an abstraction from 'ad hoc' solutions to thoroughly motivated applications of suitable classifiers.

As we have seen, the concept of measure functions suggests new classes of generalization algorithms. One such class consists of algorithms that incrementally searches the space of generalizations in order to find one that maximizes the measure function. We expect that future work along this line will be very fruitful. Another future line of research is to investigate the possibilities of automatically determining an appropriate measure function given the available data set, for instance, by computing appropriate values for the parameters of a given measure function.

References

[1] H. Akaike. A new look at statistical model identification. *IEEE Transactions on Automatic control*, 19:716–723, 1974.

[2] E. Anderson. The Irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[3] A. Andersson, P. Davidsson, and J. Lindén. Measuring generalization quality. Technical Report LU–CS–TR: 98–202, Department of Computer Science, Lund University, Lund, Sweden, 1998.

[4] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, 1990.

[5] D.F. Gordon and M. des Jardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1/2):5–22, 1995.

[6] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI-95*, Morgan Kaufmann, 1995.

[7] D.G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[8] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[9] G. Nakhaeizadeh and A. Schnabl. Development of multi-criteria metrics for evaluation of of data mining algorithms. In *KDD'97*, 1997.

[10] E. Parzen. Multiple time series modeling: determining the order of approximating autoregressive schemes. In *Multivariate Analysis, IV*, pages 283–295. North-Holland, 1977.

[11] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[12] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[13] R.B. Rao, D. Gordon, and W. Spears. For every generalization action, is there really an equal and opposite reaction? Analysis of the conservation law for generalization performance. In *Twelfth International Conference on Machine Learning*, pages 471–479. Morgan Kaufmann, 1995.

[14] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[15] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol.1: Foundations*. MIT Press, 1986.

[16] C. Schaffer. A conservation law for generalization performance. In *Eleventh International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann, 1994.

[17] D.H. Wolpert. Off-training set error and a priori distinctions between learning algorithms. Technical Report SFI TR: 95–01–003, Santa Fe Institute, Santa Fe NM, USA, 1995.