Measure-based Performance Evaluation

Arne Andersson [a], Paul Davidsson [b], Johan Lindén [c]

[a] *Computing Science Dept., Uppsala University, Box 311, S–751 05 Uppsala, Sweden. E-mail: arnea@csd.uu.se*

[b] *Dept. of Computer Science, University of Karlskrona/Ronneby, S–372 25 Ronneby, Sweden. E-mail: pdv@ipd.hk-r.se*

[c] *Dept. of Computer Science, Lund University, Box 118, S–221 00 Lund, Sweden. E-mail: johan@cs.lth.se*

Abstract

The concept of measure functions for generalization performance is suggested. This concept provides an alternative way of selecting and evaluating learned classifiers, and it allows us to define the learning problem as a computational problem.

Key words: Classifier performance evaluation, cross-validation, generalization.

1 Introduction

In this work, we suggest a new approach to evaluation of classification performance. Today, most methods for evaluating the quality of a learned classifier are based on some kind of cross-validation (Kohavi, 1995). However, we argue that it is possible to make evaluations that take into account other important aspects of the classifier than just classification accuracy on a few instances.

It has been known for a long time, see for example the "no free lunch" theorems (cf. (Schaffer, 1994,Wolpert, 1995)), 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 classifier selection from the details of the learning algorithms.

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 classifiers learned by some popular learning algorithms. Finally, we illustrate that measure functions are helpful when designing new algorithms, by presenting a simple algorithm 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 each possible combination of a data set and a classifier, a *measure function* assigns a value describing how good the classifier is at generalizing from the data set. 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 (Akaike, 1974), and Parzen's criterion autoregressive transfer (Parzen, 1977).

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 on a finite universe.

Let U be the set of all possible instances, α a set of instances ($\alpha \subset U$), and c_α a set of pairs such that each instance in α is labeled with a category. A generalization task, τ, is defined by a pair $\langle U, c_\alpha \rangle$. In what follows we will by T refer to the set of all possible generalization tasks (given U and the categories present, K).

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, $C = U \times K$. Thus, a generalization algorithm computes a function $G : T \to C$.

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

Definition 2 *The generalization performance on a particular generalization task τ for a generalization algorithm G is defined by $f(\tau, G(\tau))$.*

With these definitions, we may describe generalization as a computational problem: Given a generalization task τ, 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. (Schaffer, 1994,Wolpert, 1995,Rao, Gordon, Spears, 1995)). In short, it can be proved that once a non-trivial measure function is defined, some algorithms are better than others (see (Andersson, Davidsson, Lindén, 1998)). By a trivial measure function we mean one that gives constant output regardless of input.

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 expertise for different biases (Gordon, des Jardins, 1995).

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 – known 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 most popular scheme for measuring subset-fit is the *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. On the other hand, they help in tuning or choosing algorithms.

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. 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, a query instance that resides in the area between the clusters will be classified as belonging to the category whose instances it is most similar to.

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 uses 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 to use 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. Three common classes of algorithms will be compared, but first let us briefly analyze what heuristics they use.

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 (Quinlan, 1986) 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 (Quinlan, 1993) trades accuracy for simplicity.

Similarity: When dealing with numeric features, a similarity criterion is typically taken into consideration by choosing cut-points that lie centered between training instances of different categories.

The backpropagation algorithm *Subset-fit:* Roughly speaking, the backpropagation algorithm (Rumelhart, Hinton, Williams, 1986) 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. Therefore, it needs to be combined with other strategies such as decreasing the number of neurons or stopping the training early.

Similarity: The sigmoid functions used in the neurons have a tendency to place neurons in the middle between clusters of different categories.

Nearest neighbor algorithms This class of algorithms (Dasarathy, 1990) is clearly based on similarity. A method to avoid over-fitting is to look at more than one nearest neighbor, this has a tendency to generate simpler decision borders.

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_α denotes the training set and c_U denotes the classifier produced by the algorithm. A general measure function can be defined as follows:

$$a_0 \frac{|c_\alpha \cap c_U|}{|c_\alpha|} + a_1 simi(c_\alpha, c_U) + a_2 simp(c_U) \tag{1}$$

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).

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 and 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 consider domains with numerical features where it is possible to view measure functions geometrically. We would like to stress that this is just an example; we do not claim that our measure is the best choice.

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" distances 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 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., $c_\alpha \cap c_U$). The total *simi* function will then be

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

Then, if we wish to pay less attention to misclassified instances, we can choose $k_2 < k_1$. We denote the two parts $simi_{(R)}$ and $simi_{(c_\alpha \setminus R)}$.

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.)

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:

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

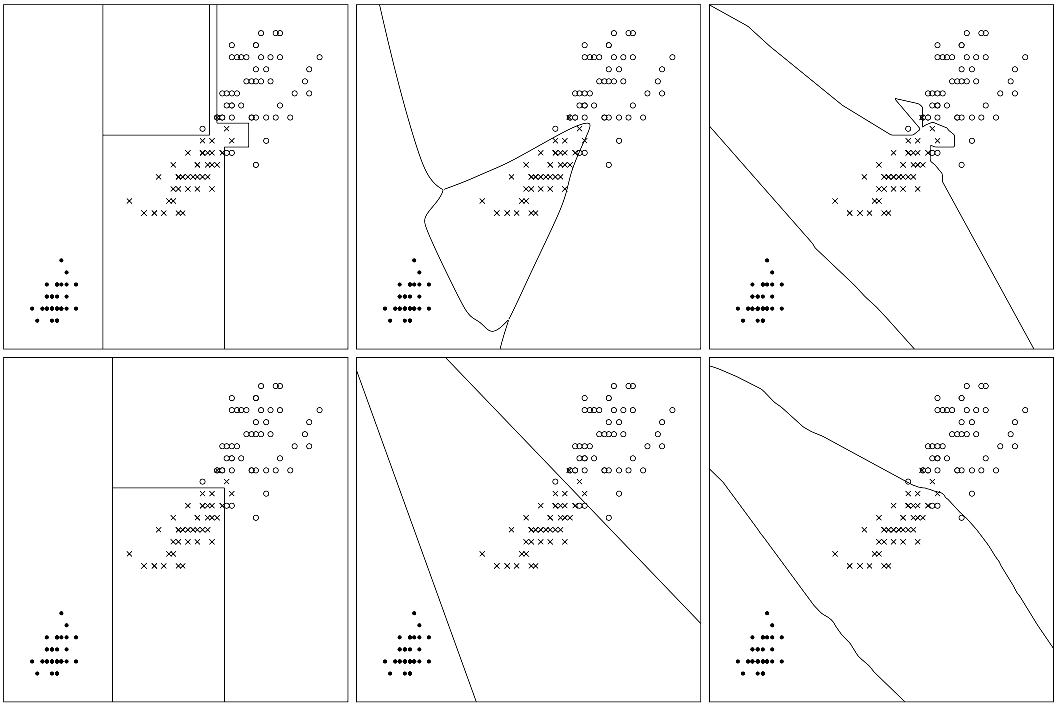

Fig. 1. *Left:* Decision tree. The upper version is a tree induced by the ID3 algorithm and the lower version is a pruned tree, taken from (Michie, Spiegelhalter, Taylor, 1994, p. 10). *Middle:* Backpropagation. The upper version use 30 hidden nodes and 26 500 training epochs; the lower use 2 hidden nodes and 20 000 epochs. *Right:* Nearest neighbor. The upper version is 1-nearest neighbor. The lower version use 10 neighbors.

4.3 Classifier evaluation and selection

We have used the well-known Iris data base (Anderson, 1935). For the sake of presentation, we only used two of the four dimensions (petal length and width).

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* 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

	$\frac{\lvert R\rvert}{\lvert c_\alpha\rvert}$	$simi_{(R)}$	$simi_{(c_\alpha \setminus R)}$	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 (in two parts), and simplicity, as well as the resulting measure for our choice of parameters.

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 so that each of the three components has approximately the same influence 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 *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.

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. From this, however, we cannot draw the conclusion that it is the best algorithm. 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.

Also, 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.

In the example we used a two-dimensional feature space for the sake of presentation. However, most practical applications deal with feature spaces with higher dimensionality. The complexity of computing simplicity and similarity grows with the number of dimensions, therefore we need appropriate approximations of these components in order to keep the approach tractable. To estimate simplicity, we may, for example, use the average number of decision borders crossed by random lines through feature space. The similarity component may also be approximated—for each instance, find the closest border by searching along each dimensional axis separately, or search along lines connecting examples of different categories.

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 an 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

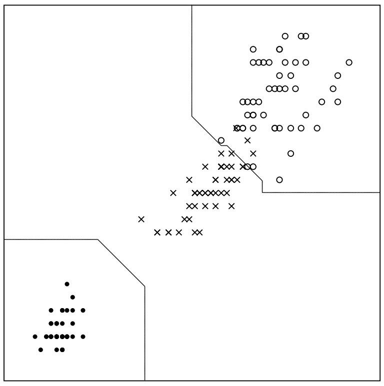

Fig. 2. The generalization produced by a simple measure-based algorithm.

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 in essentially the following way:

(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.

In our example implementation, we let the randomly chosen geometrical objects be just rectangles of varying size, either axis-parallel or rotated 45 degrees.

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 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 Figure 2 gets the value 1.288 (the four terms are 0.973, 0.731, 0.007, and 1.891). There is much more to be said about measure-based algorithms, 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 CV, are complementary. They can be used separately or in combination.

When using CV, an algorithm is assumed to give high accuracy on the entire universe if it shows good statistical behavior on available data. In this way, CV performs a *quantitative* analysis. The result of such analysis is often sound, but it is easy to generate counter-examples where CV will fail. Hence, CV 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.

CV 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 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

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

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

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

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

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

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

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

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

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

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

Rao, R.B., D. Gordon, W. Spears, 1995. 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.

Rumelhart, D.E., G.E. Hinton, R.J. Williams, 1986. 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.

Schaffer, C. 1994. A conservation law for generalization performance. in: *Eleventh International Conference on Machine Learning* (Morgan Kaufmann, 1994), pages 259–265.

Wolpert, D.H. 1995. 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.*