To appear in *1992 Lectures in Complex Systems*, L. Nadel and D. Stein (Eds.), Addison-Wesley, 1993

COMBINING GENERALIZERS USING PARTITIONS OF THE LEARNING SET

by David H. Wolpert

Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM, 87501, USA, (dhw@santafe.edu)

Abstract: For any real-world generalization problem, there are always many generalizers which could be applied to the problem. This paper discusses some algorithmic techniques for dealing with this multiplicity of possible generalizers. All of these techniques rely on partitioning the provided learning set in two, many different times. The first technique discussed is cross-validation, which is a winner-takes-all strategy (based on the behavior of the generalizers on the partitions of the learning set, it picks one single generalizer from amongst the set of candidate generalizers, and tells you to use that generalizer). The second technique discussed, the one this paper concentrates on, is an extension of cross-validation called stacked generalization. As opposed to cross-validation's winner-takes-all strategy, stacked generalization uses the partitions of the learning set to combine the generalizers, in a non-linear manner, via another generalizer (hence the term "stacked generalization"). After presenting a cursory review of stacked generalization, this paper discusses some possible extensions of stacked generalization.

I. INTRODUCTION

This paper concerns the problem of inferring a function f from a subset of $\mathbf{R}^n$ to a subset of $\mathbf{R}^p$ (the *target* function) given a set of m samples of that function (the *learning set*). The subset of $\mathbf{R}^n$ is the *input space*, labeled X, and the subset of $\mathbf{R}^p$ is the *output space*, labeled Y. A *question* is an input space (vector) value. An algorithm which guesses what the target function is, basing the guess only on a learning set of m $\mathbf{R}^{n+p}$ vectors read off of that target function, is called a *generalizer*. A generalizer guesses an appropriate output for a question via the target function it infers from the learning set. Colloquially, we say that the generalizer is "trained", or "taught", with the learning set, and then "asked" a question.

For any real-world generalization problem there are always many possible generalizers one can use. Accordingly one is *always* implicitly presented with the problem of how to address the multiplicity of possible generalizers. One possible strategy is to simply choose a single generalizer according to subjective criteria. As an alternative, this paper discusses the objective (i.e., algorithmic) technique of "stacking". Before doing so however, to provide some context and nomenclature, this paper presents a cursory discussion of the cross-validation procedure, the traditional method for objectively addressing the multiplicity of possible generalizers.

II. CROSS-VALIDATION

Perhaps the most commonly used "objective technique for addressing the multiplicity of generalizers" is cross-validation[3,7-9,12]. It works as follows:

Let $L = \{(x_i, y_i)\}$ be the learning set of m input-output pairs. The (leave one out) *cross-validation partition set* (CVPS) is a set of m partitions of L. It is written as $\{L_{ij}\}$, $1 \le i \le m$, $1 \le j \le 2$. For fixed i, L_{i2} consists of a single input-output pair from L, and L_{i1} consists of the rest of the pairs from L. The input component of L_{i2} is written as $\text{in}(L_{i2})$, and the output component is written as $\text{out}(L_{i2})$. Varying i varies which input-output pair constitutes L_{i2}; since there are m pairs in L, there are m values of i.

We have a set of generalizers $\{G_j\}$. Indicate by $G_j(L'; q)$ the guess of generalizer G_j when trained on the learning set L' and asked the question q. For each generalizer G_j, use the CVPS to compute the (leave one out) *cross-validation error*, $\Sigma_{i=1}^{m}$ $[G_j(L_{i1}; in(L_{i2})) - out(L_{i2})]^2 / m$. This is the average (squared) error of G_j at guessing one pair of L (L_{i2}) when trained on the rest of L (L_{i1}). If we interpret the cross-validation error for G_j as an estimate of the generalization error of G_j when trained on all of L, then the technique of cross-validation provides a winner-takes-all rule: choose the generalizer which has the lowest cross-validation error on the learning set at hand, and use that generalizer to generalize from the entire learning set.

There are a number of other partition sets one can use besides the leave-one-out cross-validation partition set. Two of the most important are the J-fold cross-validation partition set and the bootstrap partition set. In J-fold cross-validation, i ranges only from 1 to J. For all i, L_{i2} consists of m / J input-output pairs from L, and L_{i1} consists of the rest of L. The input-output pair indices making up L_{i2} are disjoint from those making up L_{j2} for $i \neq j$ (so if no input-output pair is duplicated in L, $L_{i2} \cap L_{j2} = \varnothing$ for $i \neq j$). Accordingly, the set of all L_{i2} covers L, assuming J is a factor of m. Using this partition set, one computes the cross-validation error exactly as in leave-one-out cross-validation, and then chooses the generalizer with the lowest error. Since it only requires the training of a generalizer J times (rather than m times, as in leave-one-out cross-validation), J-fold cross-validation is less computationally expensive than leave-one-out cross-validation. It is also sometimes a more accurate estimator of generalization accuracy.[2]

As another example of an alternative to leave-one-out cross-validation, one can use a *bootstrap* partition set.[3] Such a partition set is found by stochastically creating the L_{i1}: each L_{i1} is formed by sampling L m times in an i.i.d. manner (according to a uniform distribution over the elements of L). There are a number of variations of these basic ideas, e.g., generalized cross-validation[5], stratification, etc. For a discussion of (some) such variations, see reference 10.

Although it isn't based on subjective judgements for addressing the multiplicity of generalizers, cross-validation (and its variations) would be pointless if it didn't work well in the real world. Fortunately, it does work, usually quite well. For example, see reference 12 for an investigation in which cross-validation error has almost perfect correlation with generalization error, for the NETtalk data set.

III. STACKED GENERALIZATION

Cross-validation is a winner-takes-all strategy. As such, it is rather simple-minded; one would prefer to *combine* the generalizers rather than choose just one of them. Interestingly, this goal of combining generalizers can be achieved using the same idea of exploiting partition sets which is employed by cross-validation and its variations. To see how, consider the situation depicted in figure 1. We have a learning set L, and a set of two candidate generalizers, G_1 and G_2. We want to infer (!) an answer to the following question: if G_1 guesses g_1, and G_2 guesses g_2, what is the correct guess?

To answer this question we make the same basic assumption underlying cross-validation: generalizing behavior when trained with proper subsets of the full learning set correlates with generalizing behavior when trained with the full learning set. To exploit this assumption, the first thing one must do is choose a partition set L_{ij} of the full learning set L. For convenience, choose the CVPS.[1] Now pick any partition i from L_{ij}. Train both G_1 and G_2 on L_{i1}, and ask them both the question in(L_{i2}). They will make the pair of guesses g_1 and g_2. In general, since the generalizers weren't trained with the input-output pair L_{i2}, neither g_1 nor g_2 will equal the correct output, out(L_{i2}). Therefore we have just learned something: when G_1 guesses g_1 and G_2 guesses g_2, the correct guess is out(L_{i2}).

From such information we want to be able to infer what the correct guess is when both G_1 and G_2 are trained on the full learning set and asked a question q. The most natural way to carry out such inference is via a generalizer. To do this, first cast the information gleaned from the partition i as an input-output pair in a new space (the new space's input being the guesses of G_1 and G_2, and the new space's output being the correct guess). Repeat this procedure for all partitions in the partition set. Different partitions gives us different input-output pairs in the new input-output space; collect all these input-output pairs, and view them as a learning set in the new space.

This new learning set tells us all we can infer (using the partition set at hand) about the relationship between the guesses of G_1 and G_2 and the correct output. We can now use this new learning set to "generalize how to generalize"; we train a generalizer on this new learning set, and ask it the two-dimensional question $\{G_1(L; q), G_2(L; q)\}$. The resultant guess serves as our final guess for what output corresponds to the question q, given the learning set L. Using this procedure, we have inferred the biases of the generalizers with respect to the provided learning set (loosely speaking), and then collectively corrected for those biases to get a final guess.

Procedures of this sort where one feeds generalizers with information from other generalizers are known as "stacked generalization".[1,14] The original learning set, question, and generalizers are known as the "level 0" learning set, question, and generalizers. The new learning set, new question, and generalizer used for this new learning set and new question are known as the "level 1" learning set, question, and generalizer.

Some important aspects of how best to use stacked generalization become apparent when the learning set is large and the output space is discrete, consisting of only a few (k) values. Assume that we are using the architecture of figure 1 to combine three generalizers, using a bootstrap partition set. Assume further that all $k^3 \times k$ possible combinations of a level 1 input and a level 1 output can occur quite often (i.e., assume the learning set consists of many more than k^4 elements).

View the level 1 learning set as a histogram, of (number of occurrences of the various possible) level 1 inputs and associated outputs. Assuming the statistics within the level 0 learning set mirrors the statistics over the whole space, this histogram should approximate well the true joint probability distribution Pr(output, guesses of the 3 generalizers). (In particular, since the learning set is large, finite sample effects should be small.) Therefore if we guess by using that histogram, we will be guessing according to (a good approximation of) the true distribution Pr(output | guesses of the 3 generalizers). In general, such guessing can not give worse behavior than guessing the value of any single one of the generalizers. (Leo Breiman has proven a more formal version of this statement - private communication.) Therefore one should expect that the error of stacking with a "histogram" level 1 generalizer is bounded below by the error of any winner-takes-all technique like cross-validation.

Even when the learning set is large enough so we can ignore finite sample effects, so that the statistics inside L mirrors the statistics across the whole space, etc., it still might be that stacking with a histogram level 1 generalizer won't do better than using one of the level 0 generalizers by itself. This is the case, for example, if the following condition always holds: no matter what the guesses by the three level 0 generalizers, the best guess (i.e., the mode output value of the histogram, for the given level 1 input) is always equal to the guess of the same one of those three level 0 generalizers. For such a scenario, the stacking is simply telling you to always use the guess of that single level 0 generalizer.

A somewhat more illuminating (and surprisingly common) example arises when the three generalizers not only have the same cross-validation error, but also usually make the same guesses when presented with the same L_{i1} and L_{i2}. When the generalizers are synchronized this way (synchronized as far as L is concerned), combining them gains nothing; since the generalizers are acting identically, one might as well run the stacking using only a single one of them, as described in reference 14. (Intuitively, if the generalizers behave identically as far as (partitions of) the data are concerned, then combining them gains one nothing.) This example suggests that one should try to find generalizers which behave very differently from one another, which are in some sense "orthogonal", so that their guesses are not synchronized. Indeed, if for a particular learning set the guesses are synchronized even for generalizers which *are* quite different from one another, the suspicion arises that one is in a data-limited situation; in a certain sense, there is simply nothing more to be milked from the learning set.

For these kinds of reasons, it might be that best results arise if the $\{G_j\}$ are not very sensible as stand-alone generalizers, so long as they exhibit different generalization behavior from one another. (After all, in a very loose sense, the $\{G_j\}$ are being used as extractors of high-level, nonlinear, “features”. The optimal such extractors might not make much sense as stand-alone generalizers.) As a simple example, one might want one of the $\{G_j\}$ to be a shallow decision-tree generalizer, and another one a generalizer which makes very deep decision trees. Neither generalizer is particularly reasonable considered by itself (intermediate depth trees are usually best), but they might operate quite well when used cooperatively in a stacked architecture.

There is now a good deal of evidence that stacking works well in several scenarios. It appears to systematically improve upon both ridge and subset regressions.[1] Moreover, after learning about stacked generalization partitions, Zhang et al. have used it to create the current “champion” at protein folding.[16] See also references 4 and 14.

In addition, it should be possible to use stacked generalization in combination with other schemes designed to augment generalizers. For example, one might be able to use stacking to improve the “boosting” procedure developed recently in the COLT community.[6] The idea would be to train several versions of the same generalizer, exactly as in boosting. However rather than training them with input-output examples chosen from all of L, as in conventional boosting, one trains each of them with examples chosen from one part of a partition set pair (i.e., from an L_{i1}). One then uses the other part of the partition set pair (L_{i2}) to see how to combine the generalizers (rather than just using a fixed majority rule, as in standard boosting). This might give better performance than boosting used by itself. It also naturally extends boosting to situations with non-binary (and even continuous-valued) outputs, in which a simple majority rule makes little sense.

IV. VARIATIONS OF STACKED GENERALIZATION

There are many variations of the basic version of stacked generalization outlined above.

(See reference 14 for a detailed discussion of some of them.) One variation is to have the level 1 input space contain information other than the outputs of the level 0 generalizers. For example, if one thinks that there is a strong correlation between {the guesses of the level 0 generalizers, together with the level 0 question} and {the correct output}, then one might add a dimension to the level 1 input space (several dimensions for a multi-dimensional level 0 input space), for the value of the level 0 question.

Another useful variation is to have the level 1 output space be an estimate for the error of the guess of one of the generalizers rather than a direct estimate for the correct guess. In this version of stacked generalization the level 1 learning set has its outputs set to the error of one of the generalizers rather than to the correct output. When the level 1 generalizer is trained on this learning set and makes its guess, that guess is interpreted as an error estimate; that guess is subtracted from the guess of the appropriate level 0 generalizer to get the final guess for what output corresponds to the question.

There are some particularly nice features of this error-estimating version of stacked generalization. One is that it can be used not as a way to improve a guess, but rather simply to get a confidence estimate for that guess. Another nice feature of having the level 1 output be an error estimate is that this estimate can be multiplied by a real-valued constant before being subtracted from the appropriate level 0 generalizer's guess (when one is trying to improve that guess). When this constant is 0, the guessing of the entire system reduces to simply using that level 0 generalizer by itself. As the constant grows, the guessing of the entire system becomes less and less the guess of that level 0 generalizer by itself, and more and more the guess of a full stacked generalization architecture; that constant provides us with a knob determining how conservative we wish to be in our use of stacked generalization. Yet another nice feature of having the level 1 output space be an error estimate for one of the level 0 generalizers is that with such an architecture the guess of that level 0 generalizer often no longer needs to be in the level 1 input space, since its information is already being incorporated into the final guess automatically (when one does the subtraction). In this way one can reduce the dimensionality of the level 1 input space by one.

Another variation of stacked generalization is suggested by the distinction between "strong" cross-validation and "weak" cross-validation.[11,15] The conventional form of cross-validation discussed so far is "weak" cross-validation. Using it to judge amongst generalizers is equivalent to saying, "Given L, I will pick the G_j which best guesses one part of L when trained on another part of it". In strong cross-validation, one instead says, "Given a target function f, I will pick the G_j which best guesses one part of f when trained on (samples from) another part of it". Intuitively, with strong cross-validation we are saying that we don't want to rely too much on the learning set at hand, but rather want to concern ourselves with behavior related to the target function from which the learning was (presumably randomly) sampled. We want to take into account what would have happened if we had had a different learning set chosen from the same target function.

There are a number of ways to use strong cross-validation in practice. One can be viewed as using decision-directed learning to create guesses for f, and then measuring strong cross-validation over those guessed f. This procedure starts by training all the generalizers on the entire learning set L. Let the resultant guesses for the input-output function be written as $\{h_j\}$ (the index of a generalizer G_j and of its guess h_j have the same value). For each h_j, one 1) randomly samples h_j according to the distribution $\pi(x)$ to create a "learning set" and then a separate "testing set"; 2) trains the corresponding generalizer G_j on that learning set; and then 3) sees how well the trained G_j predicts the elements of the testing set. One does this many times, and tallies the average error. One does this for all j. The j which gives the smallest average error for this procedure is the one picked. (I.e., one generalizes from L with h_j, where j is the index giving the smallest average error.) One variation involves observing the behavior of generalizers G_j when they're trained on learning sets constructed from functions $h_{i \neq j}$. Another variation involves using a different $\pi(x)$ for creating learning sets and testing sets (e.g., one might want to ensure that the elements in the new learning sets lie very close to the elements in the original L). Note that the whole procedure can then be iterated: one uses the original L to create h's which are used to create new L's, then use those new L's to create new h's, and so on.

Strong cross-validation makes most sense if one is in a noise-free scenario. It also makes most sense if one doesn't view cross-validation (directly) in terms of generalization error estimation and how to use L to perform such estimation. The idea is to instead view it as an *a priori reasonable* criterion for choosing amongst a set of $\{G_j\}$: choose that G_j which is, loosely speaking, most self-consistent with respect to the learning set. In other words, take the generalizers at their word. If a generalizer guesses h_j, let's call its bluff, and then see what the ramifications are, what kind of cross-validation errors we would have gotten if the target function were indeed h_j and we sampled it to get a different learning set from L.

Such a viewpoint notwithstanding, when using strong cross-validation in practice it often makes sense to bias the distribution $\pi(x)$ used for sampling the h_j - perhaps strongly - in favor of the points in the original learning set. In the extreme, when the (level 0 input space) sampling only runs over the input values in the original L, strong cross-validation essentially reduces to the bootstrap procedure (assuming the sampling is done without any noise, and that each G_j acts apropriately and perfectly reproduces the elements of the learning set on which it's trained).

One might use the idea behind strong cross-validation to construct a kind of "strong" stacked generalization. For example, one might start by forming the $\{h_j\}$, and then forming learning and testing sets by sampling the $\{h_j\}$, just as in strong cross-validation. For each such learning and testing set, one forms new level 1 input-output pairs which are added to the level 1 learning set (i.e., one treats the newly created learning set as an L_{i1} and the newly created testing set as an L_{i2}). As with strong cross-validation, in practice one should probably have the sampling used to create the new learning and testing sets be heavily weighted towards the points in the original learning set. In the extreme where the sampling only runs over the input values in the original L, one essentially recovers stacked generalization with a bootstrap partition set (assuming the sampling is done without any noise, and that each G_j acts appropriately and perfectly reproduces the elements of the learning set on which it's trained).

Other variations of stacked generalization are based on using the level 1 learning set differently from the way it's used in figure 1. As an example, one might never train the level 0 generalizers on all of L, but rather only use the guesses of the generalizers when trained on the L_{i1}, the level 0 learning sets directly addressed by the level 1 learning set. (The idea is that the level 1 learning set only *directly* tells us how to guess when the training is on subsets of L, so perhaps we should try to use it only in concert with such subsets.) To make a guess for what output goes with a question q, one uses a "cloud" consisting of a set of points in the level 1 input space. Each element of the cloud is fixed by a partition set index i, and is given by the level 1 input vector $\{G_j(L_{i1}; q)\}$ (j indexing the components of the vector). In other words, each such point is the vector of guesses made by the generalizers when trained on L_{i1}. The "cloud" of such points is formed by running over all i, i.e., over all elements of the partition set. Given this cloud, there are a number of ways to use the level 1 learning set to make the final guess. For example, one might make a guess for each level 1 input value in the cloud, by using the level 1 generalizer and the level 1 learning set. One could then average these guesses over the elements of the cloud, where each guess is weighted according to how close (according to a suitable metric) the associated level 1 input is to an element of the level 1 learning set. (In other words, we would weight a guess of the level 1 generalizer more if we have more confidence in it, based on how close the associated level 1 question is to elements of the level 1 learning set.)

There are a number of possible schemes for automatically optimizing the choice of generalizers and/or stacking architecture. It is easiest to consider them in the context of the architecture of figure 1. One of the most straight-forward of these schemes is to use minimal cross-validation error of the entire stacked generalization structure as an optimality criterion, perhaps together with genetic algorithms as a search strategy.[2] It should be noted that besides minimal cross-validation error of the entire system, there are many other possible optimality criteria, some of which are much more computationally efficient. One of the simplest is the mean-squared error of a a least-mean-squared (LMS) fit of a hyperplane to the level 1 learning set. In a similar vein, one might use as criterion the degree to which the guesses being fed up to the level 1 input space are not synchronized (see above), or more generally the degree to which the level 1 learning set is both single-valued and spread out in the level 1 input space. (This last criterion is quite similar to the idea behind error-correcting output codes, except that rather than changing the way an output vector is coded, here we are changing the level 1 input space.)

The basic idea of searching over stacking structures is not restricted to changes in discrete quantities like network topologies or choices of generalizers. For example, consider the case where each level 0 generalizer is parameterized by a constant specifying the degree of regularization (or depth of a decision tree, or some such). One could keep the topology and choice of generalizers constant, and vary the level 0 generalizers' parameterizing constants (so as to maximize the value of an optimality measure). This might result in level 0 generalizers with very different behavior from one another, a property which, as mentioned above, is often desirable.

As a practical note, if one uses schemes like those just mentioned with cross-validation as one's optimality criterion, it's important to bear in mind that one can "over-cross-validate" just as one can "over-train".[13] Accordingly, one might either try to stop the search process early, or "regularize" the cross-validation error somehow (e.g., penalize use of those level 0 generalizers which have many degrees of freedom which the search-over-cross-validation-errors can vary). Similar considerations often apply to other optimality criteria besides cross-validation.

As a final point, note that it might be possible to use stacking profitably for purposes other than generalization. For example, let's say we're combining a set of generalizers, as in figure 1, and that each of those generalizers is a Bayesian generalizer. The differences between the generalizers is in their choice of prior. In other words, we implicitly have a hyperparameter α, and rather than a known prior $P(f)$ we have a conditional prior, $P(f \mid \alpha)$ (α indexes the different generalizers). We might want to know $P(\alpha)$.

How to find $P(\alpha)$? One possibility is to be a pure Bayesian, and (try to) derive $P(\alpha)$ using first-principles reasoning. Another possibility is empirical Bayes: loosely speaking, one "cheats" (i.e., uses less than fully rigorous reasoning), and sets priors using frequency count (i.e., maximum likelihood) estimates based on past experience. A third possibility is to cheat a different way, by using stacked generalization. One version of this idea is to use a level 1 generalizer which is an

LMS fit of a hyperplane to the level 1 learning set, with all the coefficients of the hyperplane restricted to be non-negative. Given such a fit, one might, as a crude heuristic, take the resultant guessing of the full stacked structure to give the maximum a posteriori $P(f \mid L)$. Since $P(f \mid L) = \Sigma_\alpha [P(f \mid L, \alpha) \times P(\alpha \mid L)]$, and since the guesses of the level 0 generalizers give $P(f \mid L, \alpha)$, one might estimate $P(\alpha \mid L)$ as the coefficients of the hyperplane fit, suitably normalized. (Loosely speaking, for large enough L and a leave-one-out CVPS, one might presume that $P(\alpha \mid L_{i1}) \cong P(\alpha \mid L)$.) One might then estimate $P(\alpha)$ from $P(\alpha \mid L)$, using the (assumed known) likelihood $P(L \mid f)$ and conditional prior $P(f \mid \alpha)$. The idea would be that all the Bayesian generalizers use the same likelihood, so the difference in their utility (as measured by the hyperplane coefficients) must reflect differences in α. Just as with empirical Bayes, one is setting "priors" by means of the data.[3]

FOOTNOTES

[1] In general we will want to pick a partition set in which L_{i1} is a proper subset of L, lest we "generalize how to learn" rather than "generalize how to generalize". (See reference 15.) Indeed, one could make a case for the opposite extreme, in which one picks the partition set so that as few as possible of the actual input-output pairs in L_{i2}'s are also found in L_{i1}'s.

[2] As an aside, it's interesting to view such a use of a genetic algorithm from an artificial life perspective. Both the constituent level 0 generalizers and the full stacked structure perform the same task (generalization). However whereas the full structure will be "fit" (have low cross-validation error), the level 0 generalizers in general need not be. The situation is somewhat analogous to a fit eukaryote being formed out of not-very-fit prokaryotes.

[3] This idea had its genesis in a discussion I had with Peter Cheeseman and Leo Breiman in Au-

gust 1992. Of course, any flaws in the idea as presented here are wholly a reflection of my elaboration of it.

ACKNOWLEDGMENTS

This article was supported by the Santa Fe Institute and by NLM grant F37 LM00011.

REFERENCES

1. Breiman, L. “Stacked regressions.” TR 367, Department of Statistics, University of California at Berkeley, 1992.

2. Breiman, L. and Spector, P. “Submodel selection and evaluation - X random case.” *International Statistical Review*, in press, 1992.

3. Efron, B. “Computers and the theory of statistics: thinking the unthinkable.” *SIAM REVIEW* **21** (1979): 460-480.

4. Gustafson, S., Little, G., and Simon, D. “Neural network for interpolation and extrapolation.” *Report number 1294-40, The University of Dayton Research Institute, Dayton, Ohio,* 1990.

5. Ker-Chau Li. “From Stein’s unbiased risk estimates to the method of generalized cross-validation.” *The Annals of Statistics* **13** (1985): 1352-1377.

6. Schapire, R. E. “The strength of weak learnability”. *Symposium on foundations of Computer Science* (1989): 28-33.

7. Stone, M. “An asymptotic equivalence of choice of model by cross-validation and Akaike’s criterion.” *J. Roy. Statistic. Soc. B* **39** (1977): 44-47.

8. Stone, M. Asymptotics for and against cross-validation.” *Biometrika* **64** (1977): 29-35.

9. Stone, M. “Cross-validatory choice and assessment of statistical predictions.” *J. Roy. Statistic. Soc. B* **36** (1974): 111-120.

10. Weiss, S. M., and C. A. Kulikowski. *Computer Systems that Learn*. San Mateo, California: Morgan Kaufmann Publishers, 1991.

11. Wolpert, D. “A mathematical theory of generalization: part II.” *Complex Systems* **4** (1990): 201-249. Cross-validation is a special case of the technique of “self-guessing” discussed here.

12. Wolpert, D. “Constructing a generalizer superior to NETtalk via a mathematical theory of generalization”, *Neural Networks* **3** (1990): 445-452.

13. Wolpert, D. “On the connection between in-sample testing and generalization error.” *Complex Systems* **6** (1992): 47-94.

14. Wolpert, D. “Stacked generalization.” *Neural Networks* **5** (1992): 241-259.

15. Wolpert, D. “How to deal with multiple possible generalizers.” In *Fast Learning and Invariant object recognition*, edited by B. Soucek, 61-80. New York: J. Wiley & Sons, 1992.

16. Zhang, X., Mesirov, J.P., and Waltz, D.L. “A Hybrid System for Protein Secondary Structure Prediction.” *Journal of Molecular Biol*ogy **225** (1992): 1049-1063.

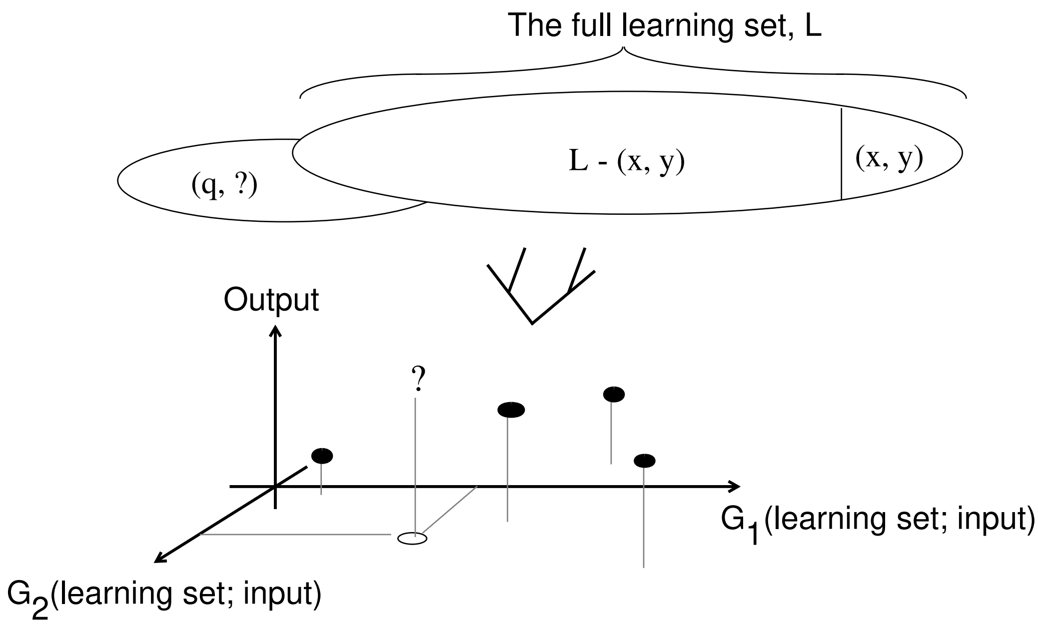

Figure 1. A stylized depiction of how to combine the two generalizers G_1 and G_2 via stacked generalization. A learning set L is symbolically depicted by the full ellipse. We want to guess what output corresponds to the question q. To do this we create a CVPS of L; one of these partitions is shown, splitting L into {(x, y)} and {L - (x, y)}. By training both G_1 and G_2 on {L - (x, y)}, asking both of them the question x, and then comparing their guesses to the correct guess y, we construct a single input-output pair (indicated by one of the small solid ellipses) of a *new* learning set L'. This input-output pair gives us information about how to go from guesses made by the two generalizers to a correct output. The remaining partitions of L give us more of such information; they give us the remaining elements of L'. We now train a generalizer on L' and ask it the two-dimensional question $\{G_1(L; q), G_2(L; q)\}$. The answer is our final guess for what output corresponds to q.