Using Errors to Create Piecewise Learnable Partitions

Oded Maron
M.I.T. Artificial Intelligence Lab
545 Technology Square, #755
Cambridge, MA 02139
oded@ai.mit.edu

After a learning system has been trained, the usual procedure is to average the testing errors in order to obtain an estimate of how well the system has learned. However, that is tossing away a lot of potentially useful information. We present an algorithm which exploits the distribution of errors in order to find where the algorithm performs badly and partition the space into parts which can be learned easily. We will show a simple example which gives the intuition of the algorithm, and then a more complex one which brings forth some of the details of the algorithm. Let us suppose that we are trying to learn the absolute value function. Almost all learning algorithms perform well along the arms of the function, but do badly around the cusp. If we notice th 'hill' of errors around $x = 0$, then we can partition the space which we are trying to learn into two parts which fall on either side of the hill. Those two partitions have the property of not only being linear, but of being learnable. Each partition can be trained separately, and when tested separately gives a better answer since irrelevant and misleading training points from other partitions have not been included.

Now let us take a more complex example of trying to learn the function shown in Figure 1. It is made up of constant parts for simplicity's sake, but in fact the parts can be anything learnable. The discontinuities cannot be learned easily by the particular learning algorithm which we are using (local weighted regression which looks at 10% of the nearest points). Figure 2 shows the error distribution gotten by cross validation. The problem of finding 1-d 'hills' of errors for one dimensional functions now becomes a problem of finding multi-dimensional 'ridges'. This is equivalent to the vision problem of edge detection. While techniques exist for two or three-dimensional edge detection, it is a difficult problem for arbitrary dimensions. Therefore, we make the assumption that the ridges occur in axis-parallel hyperplanes. While this is a big assumption, it makes the problem tractable, and this assuption is used in other well known partitioning algorithms such as CART. The algorithm now proceeds as follows: for each dimension, we scan across that dimension, looking for ridges which are parallel to our scan line. If we find such a ridge, then all points before that are put in one partition, and we continue scanning. After going through all of the dimensions this way, we have partitioned the space into pieces which are independently learnable.

The question which remains is: how do we detect these hills? The hill must be high enough so that noisy data will not cause over-partitioning. In addition, the hill must be wide enough so that a few outliers will not cause over-segmentation of the training set. Unfortunately, there is no rigorous answer to this question. We used the heuristics of detecting hills which are higher than the average error and wider than $\sqrt{n}$ if there are n points in the current partition. Once the partitions are created, we need to assign a learner to each one. The choice of how to do this is up to the particular implementation. The same learning architecture can be used for every partition, or we can try to find the best learner for each piece of the domain. Each partition also has a hyper-rectangle associated with it which encompasses the points in that partition. For a new query point, we find which hyper-rectangle in which it falls and use the learner associated with that partition to answer the query. The main differences between this algorithm and other decision-tree algorithms is that the partitions are not necessarily constant or linear — they are learnable, and that ridges of error are used to decide where to break the space.

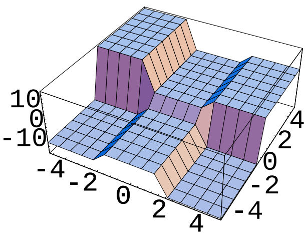

Figure 1:

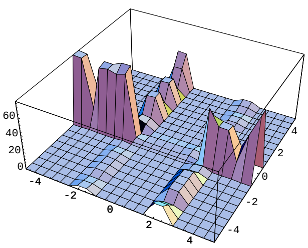

Figure 2: