Robust Learning with FeatureBoost

Joseph O'Sullivan, John Langford, Rich Caruana, Avrim Blum
School of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213
{*josullvn,jcl,caruana,blum*}*@cs.cmu.edu*

Abstract

PAC learning typically assumes that the training and test sets are drawn from the same distributions. This assumption often is violated in practice. How can machine learning algorithms be biased to output hypotheses that are robust to alterations in the test distribution? We create a framework for learning in environments where the test and training distributions differ because features in the test sets are likely to be missing or corrupted. Motivated by this framework, we present a new meta-learning algorithm, FeatureBoost. We demonstrate FeatureBoost on three learning problems using backprop nets, k-nearest neighbor, and decision trees.

1 Motivation

Consider a backprop net learning to steer a car in Pomerleau's ALVINN system [7]. The principle internal features learned by ALVINN nets detect the left and right edges of the road. ALVINN nets do not learn internal features that detect road centerlines. This creates a problem when the left or right edges of the road are obstructed by passing vehicles, or are missing as on bridges. Yet human steering is remarkably robust to the loss of these features because human drivers can fall back on a number of alternate features as different subsets of road features come in and out of view. Backprop nets do learn to steer better if they are *forced* to learn to recognize centerlines[4]. Why do ALVINN nets not automatically learn to use a variety of road features (such as centerlines) when learning to steer?

A related problem arises in pneumonia risk prediction[8]. Here there are a number of basic inputs available for patients before they enter the hospital (e.g., age, gender, blood pressure), as well as a number of lab features that become available after patients enter the hospital (e.g., RBC counts, oxygenation, Albumin). Models trained to predict risk from both the pre and in-hospital features are more accurate than models trained from only the pre-hospital inputs, but perform poorly on patients not yet admitted to the hospital (marginalizing over the missing in-hospital features). Models trained to predict risk from only the pre-hospital inputs are more accurate on patients not yet admitted to the hospital than marginalized models trained on all the features. If we can train models to make good predictions from only the pre-hospital inputs, why don't models trained with all the features work this well when some of the features are missing?

In ALVINN and pneumonia above, problems arise when the training and test set distributions differ because features are missing or obscured in the test set. In the real world, it is common for training sets to be drawn from different distributions than the test sets on which the models must perform well. The distributions differ for a variety of reasons:

1. Temporal drift in the underlying distributions with the test set drawn later.
2. The training set is 'cleaned' more than future test sets.
3. The training sample is too small to capture all the phenomena that will occur in future test sets.
4. Unanticipated uses of the models (e.g. use upon partial examples)

This paper introduces FeatureBoost, a meta-learning algorithm that trains models to use different sets of features. Because the final prediction will combine predictions from models depending on different (often overlapping) sets of features, it will be more robust to missing or obscured features. We develop the paper as follows:

1. Present a simple model that suggests how robustness can be improved.
2. Develop a meta-learning algorithm inspired by the model (FeatureBoost).
3. Test FeatureBoost on a variety of learning problems and algorithms.

2 A Robust Learning Model

The model we present is similar to PAC models in machine learning theory. Assume there exists some unknown underlying distribution, D, from which n examples with m features, $x^n = (x_1, ..., x_m)^n$ are drawn. These examples are labeled according to an unknown target function, $y = c(x)$.

Further assume that q disjoint subsets of the features can predict the label, $y = c_1(x_1, ..., x_i) = c_2(x_{i+1}, ..., x_j) = ... = c_q(x_{k+1}, ..., x_m)$. Co-training [2] makes a similar assumption.

The learning algorithm is presented with n labeled examples, $(x, y)^n$, and asked to produce a hypothesis, $h : X \rightarrow Y$, used for future predictions. The goal is to minimize the disagreement between $h(x)$ and $c(x)$ according to some loss function such as squared error or hamming loss. We focus on the hamming loss function, $E_h = P_D(h(x) \neq c(x))$.

Assume that an unbiased estimate of the error under the unknown distribution D can be found. We wish to understand how this error may change with an alteration in the test distribution. Specifically, consider a test distribution in which some percentage of the disjoint feature sets are corrupted uniform randomly. Corruption of a feature set in an example, x, is accomplished by picking a second example, z, from D and substituting the values of the feature set from z into x. This approach leaves the distribution of the feature unchanged by corruption.

There are two extreme cases to consider. The first is an Occam's razor motivated learner such as a decision tree, which attempts to find a small set of features that can predict the label. If we assume that a decision tree focuses on a particular 1 of the q feature sets, then when $k\%$ of the feature sets are corrupted, the decision tree is effected $k\%$ of the time. If there are just two labels and the decision tree outputs a random value when it relies on a corrupted feature set, this results in error linearly increasing from E_h to 50% as k increases from 0% to 100%.

The second extreme case occurs when the model uses all feature sets to predict the label. It's difficult to state how this will effect accuracy without making assumptions about the learning algorithm. Assume that a decision tree is used on each disjoint feature set with the resulting tree outputs passed through a majority voting function and that errors by one decision are independent of errors by all other decision trees. From $k = 0\% - 100\%$, the error will increase from E_h to 50% at a rate dependent on the value of q, the number of feature sets. Each decision tree error can be viewed as flipping a random coin with some bias. If enough random coins come up with the wrong label, they will overwhelm the good predictors in the vote. The probability of error is distributed as the cumulative distribution of a binomial tail.

The second learning algorithm dominates the first one, always doing at least as well under any $k\%$ feature set corruption and often doing significantly better. We are motivated to create a learning algorithm using a mixture of experts with each expert focusing on a different set of features.

3 FeatureBoost

The algorithm we present is a variant of boosting where features are boosted rather than examples. Think of examples as a matrix of inputs with each row a particular example. Whereas AdaBoost and other boosting algorithms emphasize particular rows (examples) in the matrix, FeatureBoost emphasizes particular columns (features) in the matrix instead. This is similar to Bay's work with k nearest neighbors [1], though the goals and algorithms differ considerably.

The goal of FeatureBoost is to search for alternative explanations for the target amongst the features. This is accomplished by conducting a sensitivity analysis on the features used by previously learned models, and then biasing future learning away from the features used most. This process is repeated and a weighted mixture of experts is output (see Table 1).

Table 1: Pseudocode describing the FeatureBoost algorithm.

FEATUREBOOST($e, D,$ WEAKLEARN$, T$)

Input: e: N examples each containing M features
D: Distribution over features (say, uniform)
WEAKLEARN: Weak Learning algorithm
T: Number of iterations

for $i \in \{1..M\}$ **do** $w_i^1 \leftarrow D(i)$
for $t \in \{1..T\}$ **do**
    $p^t \leftarrow w^t / \sum_{i=1}^{M} w_i^t$
    $h_t \leftarrow$ hypothesis of WEAKLEARN under p^t
    $I^t \leftarrow$ importance of each of M features to WEAKLEARN
    $\beta_t \leftarrow max(\epsilon_t/(1-\epsilon_t), 0.01)$
    $\epsilon_t \leftarrow \sum_{i=1}^{N} \mid h_t(i) - c(i) \mid$
    for $i \in \{1..M\}$ **do** $w_i^{t+1} \leftarrow w_i^t + I_i^i * (0.5 - \epsilon_t)$

Output: $h_f(i) = \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) h_t(i) \geq \frac{1}{2} \sum_{t=1}^{T} \log \frac{1}{\beta_t}$

It is interesting to contrast FeatureBoost with AdaBoost. To make this simpler, we used the same variable names as in [5]. In FeatureBoost, the distribution D

is over *features* and WEAKLEARN must accept a bias in the form of a probability distribution p^t over *features*. In AdaBoost, the distribution D is over examples, WEAKLEARN must accept a bias in the form of a probability distribution p^t over *examples*, there is no need to calculate the importance of features as in line 5, and the weights in line 7 are updated by:

$$\textbf{for } i \in \{1..N\} \textbf{ do } w_i^{t+1} \leftarrow w_i^t \beta_t^{1-|h_t(i)-c(i)|}.$$

It is not obvious how best to bias WEAKLEARN by a probability distribution over features. We have considered several approaches, including scaling individual features and adding noise to individual features. Here we use a progressive feature masking procedure for $t > 1$. Before calling WEAKLEARN, we calculate the current full marginalization error, ϵ_{fm} from $t-1$. We then marginalize the most important features from the training data, until the marginalized performance of WEAKLEARN is worse than $((T-t)*0.1+t*\epsilon_{fm})/T$.[1] In practice, we observe that occasionally it is too easy or too hard to do worse than this bound, so we also require that at least 15% of the features are removed, and at least 10% of the features remain.

The calculation of the importance of individual features is performed by repeatedly picking a random training example and assigning a random value to the feature according to the distribution of values for that feature in the training set. The error of the hypothesis on this example is then calculated. After many iterations (we use 500), the change in the average error of the hypothesis is detected. These error changes yield an importance vector over features that is scaled to have entries in $(0,1)$. For some weak learners the importance of features can be directly calculated.

4 Results

Figure 1 shows the performance of FeatureBoost on a synthetic domain (THRESHOLD) using boosted decision trees. In this domain, a threshold function is drawn uniformly from the interval $[25,75]$. Examples are drawn uniformly from $[0,100]$ and example values are coded in several ways: with a grey code, a "peaks" code, a binary code, a unary code, and the value divided by 100. Grey code is a similar to binary code except that only one feature changes at a time when counting from 0 to 100. Peaks code consists of 100 features with values calculated by a Gaussian centered on the example value[4]. Unary code ("thermometer code") outputs as many 1's as the example value, then fills in 0's to reach 100 features. Coding the inputs these different ways yields data that satisfies the assumption that there are multiple disjoint subsets of the features which can predict the target. Each trial trains on 100 examples and tests on 300.

The graphs in Figure 1 compare the error of the decision tree "weak" learner (DT) to that of several voting algorithms as corruption increases: MIXTURE (a simple mixture of experts), ADABOOST [5], and FEATUREBOOST. The results show how robust each method is to random corruption of the test data. The left graph in Figure 1 shows results for uniform feature corruption: where each feature is corrupted individually (feature sets contain only one feature). The right graph shows results for uniform feature set corruption: entire feature sets (the grey, binary, or thermometer coding) are corrupted at one time. Feature set corruption simulates clusters of features tending to be occluded or corrupted together, as when parts of an image are occluded, or when an event such as not being admitted to the hospital causes an entire set of features to be missing.

[1] We believe that using ϵ_{t-1}, the error of WEAKLEARN from $t-1$, instead of the constant 0.1, is more appropriate, but have not yet tested this.

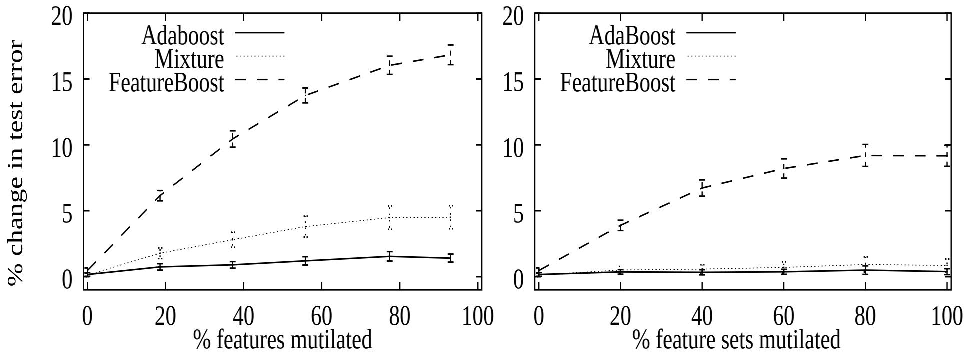

Figure 1: The improvement from boosting DT by FeatureBoost, Mixture Models, and AdaBoost on THRESHOLD. On the left, individual features are randomly corrupted. On the right, corruption happens to entire *sets* of features. Results are presented with 95% confidence intervals from 100 trials using 20 experts.

We now demonstrate FeatureBoost applied to artificial neural nets (ANN), k-nearest neighbor (kNN), and decision trees (DT). For DT we used Buntine's IND. For ANN we use three layer backprop nets with 5 hidden units, conjugate gradient descent, and early stopping using hold-out sets. For kNN we use unweighted Euclidean distance with $k = 2$.

We test each FeatureBoosted algorithm on three domains: the UCI Vote domain, a real pneumonia problem, and a synthetic problem we created to demonstrate FeatureBoosting. The Voting domain consists of a congressman's voting record on 16 votes. The label is the party, democrat or republican, which the congressman belongs to. In each trial, 100 examples were used for training, 335 for testing. The Pneumonia domain uses real patient data consisting of 30 pre-hospital and 35 in-hospital features. The label is the risk of death. In each trial 1000 examples were used for training, 2000 for testing.

The results in Figure 2 suggest that FeatureBoost dominates AdaBoost and simple mixtures of experts when features in the test set are corrupted. Simple mixtures of experts improve robustness because a mixture of experts is not an Occam's razor learner, but not as much as FeatureBoost because a simple mixture of experts does not focus learning to use different sets of features. The graphs also suggest that the effect of FeatureBoost is most pronounced for DT and least pronounced for kNN. This is expected because DT is biased to use few features, whereas kNN with unweighted Euclidean distance uses all features.

5 Discussion

The basic problem FeatureBoost addresses is how to benefit from redundancy in the input features. Most machine learning methods are what we might call "abridged" learners. They learn the simplest models that are accurate on the training data. Because simple models often depend on fewer features, these models are not very robust when applied to test sets in which important features are missing, occluded, or corrupted.

FeatureBoost makes an learning algorithm less abridged by learning multiple models that use different (possibly overlapping) sets of features. The ultimate unabridged learning procedure would be to train separate models on the power set of the input

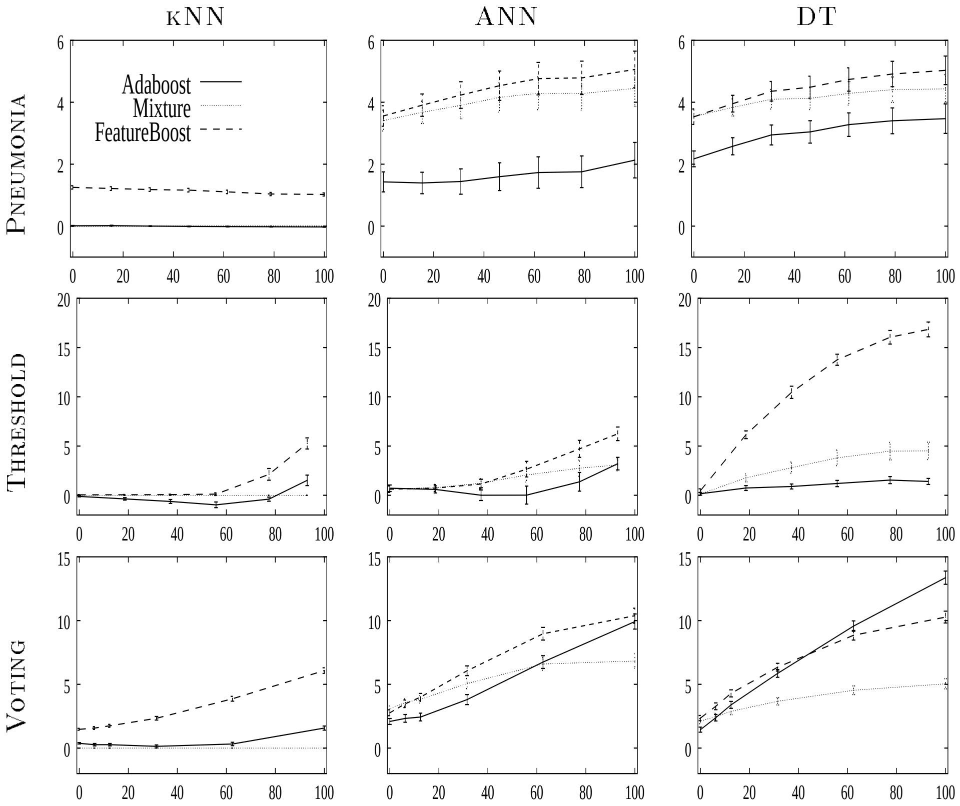

Figure 2: Percent improvement over weak learning in three domains (Pneumonia, Threshold, and Voting) for three weak learners (kNN, ANN and DT) and for three ways of generating experts. Results are presented with 95% confidence intervals from 100 or more trial using 20 experts.

features, and combine these models weighted by their estimated accuracy. This approach is impractical, of course, for problems having more than a few attributes.

Despite the success of FeatureBoost on these problems, there are difficulties using it. The optimal schedule for biasing feature use depends on the learning algorithm and target function. FeatureBoost can be considered a heuristic search through the space of "ideal" subsets of features. The goal of this search is to find diverse sets of explanations for the output. If, as with the synthetic threshold function, features are encoded with multiple redundant disjoint subsets of inputs, we would not want use the same feature in two different models. In real world problems, however, this separation is rarely so clean and it may be best to user overlapping subsets of features.

We compare FeatureBoost to AdaBoost because AdaBoost should improve robustness in some circumstances. Suppose when training ALVINN nets to steer, most of the training data contains unobstructed views of the left and right sides of the road, and only a few training cases have obstructed views of the sides of the road. The first models trained with AdaBoost will use the sides of the road as their main features. AdaBoost, however, will then focus it's effort on training cases where these models fail. In these training cases, the centerlines may be the next best features. AdaBoost probably will no learn to use centerlines as well as FeatureBoost, however, because

AdaBoost will learn about centerlines from the small fraction of training cases in which edges are not visible. If centerlines are visible in most images, including those with unobstructed views of road edges, FeatureBoost will learn about centerlines using the full training set. Combining AdaBoost with FeatureBoost may yield the best of both methods. We do not yet have theoretical guarantees for FeatureBoost comparable to those for AdaBoost.

The difference between corrupted features and missing features is important. If values are missing, a technique similar to "sleeping experts"[6] may be more effective. In this setting, there are many sub-hypotheses that combine to form an unabridged hypothesis using weighted majority (only hypotheses with no missing values vote). A softer form of this where sub-hypotheses vote with strength proportional to their confidence may work well in real world settings.

Corrupted test data is not the only model for Robust learning. Robust learning can also be thought of as PAC learning with a loss function that varies with the examples. When learning to drive a car, you don't care so much about how perfectly the car drives on open highway, but you care very much about how well it avoids crashes in more difficult situations. A PAC learner that is told which examples are riskier might also create more robust hypotheses.

There are many approaches one might use to make machine learning algorithms more unabridged. This paper presents just one. Another approach we are testing is to use feature selection to train models on compact sets of features, then remove the features selected (hard or soft removal), and train additional models on the remaining features. Just as there are a number of algorithms for boosting examples, there may be several good algorithms for boosting features.

FeatureBoost is one way of creating more robust learning. It should be most effective when applied to Occam's razor-style learners such as decision trees. Our results show that it is also effective with neural nets and kNN.

References

[1] Bay, S. (1998) Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets. *Proceedings of the Fifteenth International Conference on Machine Learning (ICML98)* pp. 37-45. Morgan Kaufmann.

[2] Blum, A. & Mitchell, T. (1998) Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pp. 92-100.

[3] Buntine, W. (1992) Learning Classification Trees. *Statistics and Computing* **2**:63-73.

[4] Caruana, R. (1997) Multitask Learning. Ph.D. Thesis, School of Computer Science, CMU.

[5] Freund, Y. & Schapire, R. (1995) A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting. *Proceedings of the Second European Conference on Computational Learning Theory.*

[6] Freund, Y. & Schapire, R. & Singer, Y. & Warmuth, M. (1997) Using and Combining Predictors that Specialize. *STOC97*.

[7] Pomerleau, D. (1993) *Neural Network Perception for Mobile Robot Guidance.* Kluwer Academic Publishers.

[8] Cooper, G. F., Aliferis, C. F., Ambrosino, R., Aronis, J., Buchanan, B. G., Caruana, R., Fine, M. J., Glymour, C., Gordon, G., Hanusa, B. H., Janosky, J. E., Meek, C., Mitchell, T., Richardson, T., and Spirtes, P. (1997) An Evaluation of Machine Learning Methods for Predicting Pneumonia Mortality. *Artificial Intelligence in Medicine 9*, pp. 107-138.