FeatureBoost: A Meta-Learning Algorithm that Improves Model Robustness

Joseph O'Sullivan JOSULLVN@CS.CMU.EDU
John Langford JCL@CS.CMU.EDU
Rich Caruana CARUANA@CS.CMU.EDU
Avrim Blum AVRIM@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213

Abstract

Most machine learning algorithms are lazy: they extract from the training set the minimum information needed to predict its labels. Unfortunately, this often leads to models that are not robust when features are removed or obscured in future test data. For example, a backprop net trained to steer a car typically learns to recognize the edges of the road, but does not learn to recognize other features such as the stripes painted on the road which could be useful when road edges disappear in tunnels or are obscured by passing trucks. The net learns the minimum necessary to steer on the training set. In contrast, human driving is remarkably robust as features become obscured. Motivated by this, we propose a framework for *robust* learning that biases induction to learn many different models from the same inputs. We present a meta algorithm for robust learning called FeatureBoost, and demonstrate it on several problems using backprop nets, k-nearest neighbor, and decision trees.

1. Motivation

Consider a backprop net learning to steer a car. In the ALVINN system (Pomerleau, 1993) the principal internal features learned by ALVINN nets detect the left and right edges of the road. Typically, ALVINN nets do not learn internal features that detect other road phenomena that could be useful for steering such as road centerlines, roadway signs, trees, other traffic, people, etc. This creates a problem when the left or right edges of the road are obstructed by passing vehicles, or are missing as on bridges and in tunnels. Yet human steering is remarkably robust to the loss of these features. 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 can learn to steer better if they learn to recognize other road features such as centerlines (Caruana, 1997). How can we *force* backprop nets to learn to use a variety of road features when learning to steer?

A related problem arises in health care (Cooper et al., 1997). Basic inputs such as age, gender, and blood pressure are available for most patients before they enter the hospital. Other measurements such as RBC counts, oxygenation, and Albumin become available after patients are hospitalized. As you would expect, models trained to predict patient risk from both the pre and in-hospital features usually outperform models trained to predict risk from only the pre-hospital inputs. But these models perform poorly on patients not yet admitted to the hospital when one marginalizes over the missing in-hospital features. Models that use only the pre-hospital inputs are more accurate for patients not yet admitted to the hospital than marginalized models trained on all the features. How can we force the learning algorithm to learn models that make better predictions when some input features (such as the in-hospital attributes) are missing for some test cases?

If the edges of the road, or the in-hospital features are always available, models learned the usual way perform well. In the ALVINN and health care problems above, the difficulty arises when features are missing or obscured in the test cases. Boosting algorithms such as AdaBoost are one way to make learned models more robust to feature obscuration. If the main features such as the edges of the road are obscured or missing from a few training cases, boosting places more emphasis on these cases because they are predicted poorly. This emphasis forces the learning algorithm to use other features such as road centerlines for these cases. Unfortunately, boosting learns about centerlines by strongly emphasizing the cases that are missing road edges, even though centerlines may be visible in all images. Boosting could learn about other features better if it used all of the training data containing those features to learn about them. How can we make boosting take full advantage of *all* the redundant information in the training set?

This paper introduces a general framework for induction called *robust learning*, which is motivated by our desire to model situations where features may be corrupted or missing in ways not adequately represented in the training set. Guided by the framework, we devise a meta-learning algorithm called FeatureBoost, that trains models to use different subsets of features. Because the final prediction from FeatureBoost combines the predictions of models that depend on different (often overlapping) subsets of features, it is more robust to missing or obscured features.

We develop the paper as follows:

1. Present a general framework for robust learning.
2. Examine a specialization of this framework that suggests one way to improve robustness.
3. Develop a meta-learning algorithm (FeatureBoost) inspired by this model.
4. Test FeatureBoost on a variety of learning problems and machine learning algorithms.

2. A Framework for Robust Learning

Our basic goal is to force an ordinary "base" learning algorithm to extract all the information it can from the training data, in order to learn prediction rules that are robust to the possibility of missing or corrupted features in test cases. To make this more precise, we begin with a theoretical model that, while not perfect, is a useful way of thinking about the problem, and motivates the algorithm given in Section 4.

As in the usual PAC model, we assume that our training examples have n features and are given to us from some fixed distribution D over the input space. We assume there is some target concept c we wish to learn, and to do this we have access to a "base" learning algorithm that chooses hypotheses from some hypothesis class H. For simplicity, we will fix some arbitrary error cutoff ϵ and say a hypothesis $h \in H$ is "good" if its error is $\leq \epsilon$, and is "bad" otherwise.

We begin by formalizing the notion that the training data contains useful redundant information. Specifically, we say that a set of hypotheses H' is *k-robust* if, for any subset of k of the n features, there is some $h \in H'$ that remains good even when those k features are corrupted. For the purposes of this model, we do not need to pin down precisely what "corrupted" means so long as "error" is well-defined, and for any hypothesis h, its error is non-decreasing as additional features become corrupted. (I.e., error tends to increase as more features become corrupted.) If h is a good (low error) hypothesis when no features are corrupted but becomes bad (high error) when a subset S of features are corrupted, we say that *S destroys h*.

Given access to examples from D labeled according to c, the goal of our algorithm will be to produce a k-robust subset $H' \subseteq H$ if one exists. That is, instead of producing a single hypothesis, we want our algorithm to produce a *set* of hypotheses such that no matter what k features are corrupted, at least one of the hypotheses is still good.[1] To achieve our goal, we assume that our base learning algorithm A has the property that if we feed it labeled examples from D with some subset of features corrupted, it will then produce a good $h \in H$ (with respect to the subset of features corrupted) if one exists.

We can say several things about this setup. First, there is a natural brute-force method that achieves our goal by making $\binom{n}{k}$ calls to A: for each set S of k features, feed data into A in which those k features are corrupted, and add the hypothesis produced by A into H'. If A ever fails to output a good hypothesis, we know that H was not k-robust and therefore no k-robust $H' \subseteq H$ exists.

The brute-force algorithm works but is impractical because the powerset of features is exponential in the number of features; ideally we want an algorithm whose running time is polynomial in k. Unfortunately, if H is "perverse", this may not be possible. Consider, for instance, the case that H contains $\binom{n}{k}$ hypotheses, each of which depends on a different subset of $n - k$ features and each changes from good to bad if even just one of those is corrupted. In this case, the only k-robust subset of H is H itself.

On the other hand, there is a natural strategy for "non-perverse" H which can be proved to make only polynomially many calls to A in certain special cases. The strategy is as follows:

1. Initialize $H' = \{\}, S = \{\}$.
2. While not done, do:
    1. Find the smallest set S of features that destroy all $h \in H'$.[2]
    2. Run A on examples from D in which features in S are corrupted, and place the hypothesis found into H'. If A fails, then halt with failure.

We can make several statements about this algorithm.

Theorem 1 *Suppose every good hypothesis $h \in H$ has an associated feature set S_h such that h is good if and only if at least one feature in S_h remains uncorrupted. Then, this algorithm makes at most k calls to A and runs in linear time per iteration.*

Proof: We start with $S = \{\}$ and each time a new h is added to H', we let $S \leftarrow S \bigcup S_h$. By assumption, S will always be the smallest set of features that destroys all of H'. Each iteration increases the size of S by at least 1, so the number of iterations is at most k. ∎

[1] Of course, ideally we would like a single k-robust rule, perhaps a weighted vote among hypotheses in H' — and in fact, this is our goal in the experimental section of the paper. However, this goal appears to be trickier to model theoretically, so we consider here the weaker goal of producing a k-robust set.

[2] Algorithmically, the way we find S would depend on the kind of hypotheses we are considering, but in the worst case we could use brute force to try all S of size 1, then of size 2 and so on.

Theorem 2 *Suppose every good hypothesis $h \in H$ has an associated feature set S_h such that h is good if and only if at least* two *features in S_h remain uncorrupted. Then, the above algorithm makes at most $O(k^2)$ calls to A.*

Proof: Consider a graph G with one node for each of the n features, in which there are initially no edges. At each iteration of the algorithm, put an edge into G between every pair of uncorrupted features in S_h (unless the edge is there already), where h is the new hypothesis produced. Notice that the set S produced in step 2.1 must, by assumption, be a vertex cover (a set of vertices covering all the edges), and every iteration adds at least one new edge into the graph. Therefore, the algorithm is done when there is no longer a vertex cover of size $\leq k$. The theorem follows from the twin facts that (a) no node will reach degree greater than $k+1$ (at that point it must be in the set S), and (b) the size of the minimum vertex cover is at most the size of the maximum matching in the graph. ∎

The algorithm we develop in Section 4 can be thought of as a more practical version of this strategy, for real data and for our real goal of producing a single robust hypothesis.

3. A Special Case of Robust Learning

We now consider a specialization of the robust learning framework presented in the preceding section. In particular, we wish to motivate the idea of using a majority vote to boost robustness.

As in the previous section, assume there exists some unknown underlying distribution, D, from which m examples with n features, $x^m = (x_1, ..., x_n)^m$ 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: $\exists c_1, \ldots, c_q$ such that $y = c_1(x_1, \ldots, x_{i_1}) = c_2(x_{i_1+1}, \ldots, x_{i_2}) = \ldots = c_q(x_{i_{q-1}+1}, \ldots, x_n)$. Co-training makes the same assumption with only 2 feature sets (Blum & Mitchell, 1998).

The learning algorithm is presented with m labeled examples, $(x, y)^m$, and asked to produce a hypothesis, $h : X \rightarrow Y$, used for future predictions. The goal is to minimize the error $e(h) = P_D(h(x) \neq c(x))$.

We wish to understand how the error changes with alteration to 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 marginal distribution of feature set values unchanged by corruption.

There are two extremes 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. In the extreme, if we assume 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 affected $k\%$ of the time. If there are just two labels and the decision tree outputs a random value when it relies on a corrupted feature, this results in error linearly increasing from $e(h)$ to 50% as k increases from 0% to 100%.

The second extreme occurs when the model uses all features. It is 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 tree are independent of errors by all other decision trees. From $k = 50\% - 100\%$, 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 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 algorithm for the second extreme dominates the first one, always doing at least as well under any $k\%$ feature set corruption and often doing significantly better. This motivates us to create a learning algorithm using a mixture of experts with each expert focusing on a different set of features.

4. FeatureBoost

The algorithm we present is a variant of boosting where features are boosted rather than examples. In boosting, a base learning algorithm LEARN is called multiple times. Each time it is presented with a different distribution over the training examples. After each step, this distribution is altered to increase the emphasis on "harder" parts of the space. At the end, the different hypotheses are combined into a single hypothesis. Boosting algorithms alter the distribution by emphasizing particular training examples. FeatureBoost alters the distribution by emphasizing particular features. Think of examples as a matrix where each row is an example. Where boosting emphasizes rows (examples) in the matrix, FeatureBoost emphasizes columns (features) in the matrix instead. Similar work with k nearest neighbors has appeared (Bay, 1998), though the goals and algorithms differ considerably.

The goal of FeatureBoost (see Table 1) is to search for alternate hypotheses amongst the features. A distribution over *features* p_t is updated at each iteration t by conducting a sensitivity analysis on the features used by the model learned in the current iteration. The distribution is used to increase the emphasis on unused features in the next iteration in an attempt to produce different sub-hypotheses.

Table 1. Pseudocode describing the FeatureBoost algorithm.

FEATUREBOOST($(x, y)^M$, LEARN, T)

Input: $(x, y)^M$: M examples with N features
LEARN: Learning algorithm
T: Number of iterations

1 **for** $i \in \{1...N\}$ **do** $w_i^1 \leftarrow \frac{1}{N}$

2 **for** $t \in \{1...T\}$ **do**
3 $\quad p^t \leftarrow w^t / \sum_{i=1}^{N} w_i^t$
4 $\quad (dx, y)^M, \epsilon_{fm} \leftarrow$ DEEMPHASIZE$((x, y)^M, p^t, t, T, 500, \epsilon_{t-1}, h_{t-1})$
5 $\quad h_t \leftarrow$ LEARN$((dx, y)^M)$
6 $\quad I^t \leftarrow$ IMPORTANCE$((x, y)^M, h_t, 500)$
7 $\quad \epsilon_t \leftarrow \sum_{i=1}^{M} \mid h_t(x_i) - y_i \mid$
8 $\quad \beta_t \leftarrow max(\epsilon_t/(1 - \epsilon_t), 0.01)$
9 $\quad$ **for** $n \in \{1...N\}$ **do** $w_n^{t+1} \leftarrow w_n^t + I_n^t * (\epsilon_{fm} - \epsilon_t)$

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

This is repeated, and the sub-hypotheses are combined into a meta-hypothesis that should be more robust. The intuition behind this combination is as for AdaBoost (Freund & Schapire, 1995) (though we are unable to provide strong theoretical guarantees as for AdaBoost). The update factor β_t decreases with ϵ_t, and in turn increases the weight $\ln(1/\beta_t)$ associated with the final hypothesis. Thus, more accurate hypotheses have more influence on the final meta-hypothesis.

We calculate the importance of individual features 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 $(0, 1)$. Pseudocode for the importance calculation is provided in Table 2.

We have experimented with several approaches to the DEEMPHASIZE function for biasing LEARN by the distribution over the features p_t in Step 4 of Table 1. Options range from "hard" (e.g., removing features) to "soft" (e.g., scaling or adding noise to individual features). In this paper we use a "hard" method for progressive feature removal that is applicable to any learning algorithm.

We calculate the current full marginalization error ϵ_{fm} from h_{t-1} before calling LEARN. ϵ_{fm} is defined as the error when every feature is picked randomly. This gives an upper bound on the error of h_{t-1}. We then marginalize features from the training data, weighted by the distribution over the features, until the marginalized performance of h_{t-1} is worse than $((T - t) * \epsilon_{t-1} + t * \epsilon_{fm})/T$. This process seeks a feature set with marginalized performance worse than ϵ_{t-1}, and approaching ϵ_{fm}. In practice, we observe that occasionally it is too easy or too hard to do worse than this threshold, so we also require that at least 15% of the features are removed, and at least 10% of the features remain. The pseudocode for the DEEMPHASIZE algorithm and the helper function MARGINALIZE are provided in Table 4 and Table 3.

Table 2. Pseudocode describing importance calculation algorithm

IMPORTANCE($(x, y)^M, h, T$)

Input: $(x, y)^M$: M examples with N features
h: A hypothesis labeling examples
T: Number of iterations

for $n \in \{1...N\}$ **do** $c_n \leftarrow 0$

for $n \in \{1...N\}$ **do**
$\quad$ **for** $t \in \{1...T\}$ **do**
$\qquad x_1 \leftarrow$ a random example from x^M
$\qquad v \leftarrow h(x_1)$
$\qquad x_2 \leftarrow$ a random example from x^M
$\qquad x_1[n] \leftarrow x_2[n]$
$\qquad$ **if** $h(x_1) \neq v$ **then**
$\qquad\quad c_n \leftarrow c_n + 1$
for $n \in \{1...N\}$ **do** $c_n \leftarrow c_n/T$

Output: c: the "importance" of each feature

Table 3. Pseudocode describing marginalize algorithm

MARGINALIZE($(x, y)^M, p, K, S, h$)

Input: $(x, y)^M$: M examples with N features
p: A distribution over features
k: number of features to marginalize over
S: Total number of statistical iterations
h: previous hypothesis

$\epsilon_m \leftarrow 0$

for $t \in \{1...S\}$ **do**
$\quad x_1 \leftarrow$ random example from x^M
$\quad v \leftarrow h(x_1)$
$\quad$ **for** $n \in \{1...K\}$ **do**
$\qquad f \leftarrow$ draw feature without replacement from p
$\qquad x_2 \leftarrow$ random example from x^M
$\qquad x_1[f] \leftarrow x_2[f]$
$\quad$ **if** $h(x_1) \neq v$ **then**
$\qquad \epsilon_m \leftarrow \epsilon_m + 1$
$\epsilon_m \leftarrow \epsilon_m/S$

Output: ϵ_m: the marginalized error

Table 4. Pseudocode describing deemphasize algorithm

DEEMPHASIZE$((x,y)^M, p, t, T, S, \epsilon, h)$

Input: $(x,y)^M$: M examples with N features
p: A distribution over features
t: iteration number
T: Total number of featureboost iterations
S: Total number of statistical iterations
ϵ: previous hypothesis error
h: previous hypothesis

$\epsilon_{fm} \leftarrow$ MARGINALIZE$((x,y)^M, p, N, S, h)$
$\epsilon_m \leftarrow \epsilon$
$K \leftarrow 0$

while $\epsilon_m < (T-t) * \epsilon + t * \epsilon_{fm}$ **and** $\frac{K}{N} < 0.90$ **or** $\frac{K}{N} < 0.15$ **do**
    $K \leftarrow K + 1$
    $\epsilon_m \leftarrow$ MARGINALIZE$((x,y)^M, p, K, S, h)$
$(dx,y)^M \leftarrow (x,y)^M$ with K features removed using p

Output: $(dx,y)^M, \epsilon_{fm}$

5. Empirical Results

We now demonstrate FeatureBoosting of artificial neural nets (ANN), k-nearest neighbor (KNN), and decision trees (DT). For DT we used IND (Buntine, 1992). For ANN we use three layer backprop nets with 5 hidden units, conjugate gradient descent, and early stopping with hold-out sets. For KNN we use unweighted Euclidean distance with $k = 2$. We will contrast FeatureBoost with the meta-learning algorithms MIXTURE (a simple mixture of experts) and ADABOOST (Freund & Schapire, 1995).

We test each meta-learning algorithm on three domains: the UCI Vote domain, a real pneumonia problem, and a synthetic problem we created to demonstrate FeatureBoosting. The Vote 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 prehospital and 35 in-hospital features (Cooper et al., 1997). The label is the risk of death. In each trial 1000 examples were used for training, 2000 for testing. In the synthetic domain, THRESHOLD, a threshold function is drawn uniformly from the interval $[25, 75]$. Examples are drawn uniformly from $[0, 100]$ and example features are encoded in several ways: with a Gray code, a "peaks" code, a binary code, a unary code, and the value divided by 100. Gray 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 (Caruana, 1997). Unary code ("thermometer code") outputs as many 1's as the example value, then fills in 0's to reach 100 features. 100 examples are used for training and 300 for testing.

Our experiments investigate how robust each learning algorithm is to random uniform feature corruption of the test data. For each case in the test set, a random n percent of the features are corrupted. The features corrupted are choosen independently from case to case.

The results in Figure 1 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 typically uses more features than LEARN. The improvement is less than FeatureBoost, however, because a simple mixture of experts does not focus learning to use different sets of features. The graphs also suggest that the benefit 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.

The synthetic domain (THRESHOLD) allows us to corrupt features in different ways. Here the input is encoded in several redundant ways: with a Gray code, a "peaks" code, a binary code, a unary code, and the value divided by 100. Thus there are multiple disjoint subsets of features that can predict the target. Figure 2 shows the performance of FeatureBoost on THRESHOLD using decision trees. The graphs compare the error of (DT) to that of the voting algorithms, and show how robust each method is to corrupted test data. However, the test sets are now corrupted two different ways; The left hand graph is as before, when each example has percentage n features selected at random for corruption. In the right hand graph, each example has some percentage n of the feature *sets* (e.g. grey code, binary code, etc) corrupted. This 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. Both the AdaBoost and Mixture voting methods are hit hard by such grouped mutilation — FeatureBoost, while weakened, still demonstrates its robustness.

6. Discussion

FeatureBoost addresses the problem of how to benefit from redundancy in input features. Most machine learning methods are lazy and learn "abridged" models. Where there is redundancy, a learning algorithm should be capable of learning accurate models using any of a number of different subsets of features. But most do not. Instead, they learn the simplest models that are accurate on the training data. Because simple models often depend on fewer features, and fail to exploit redundancy in the input features, they are not very robust when when some features are missing, oc-

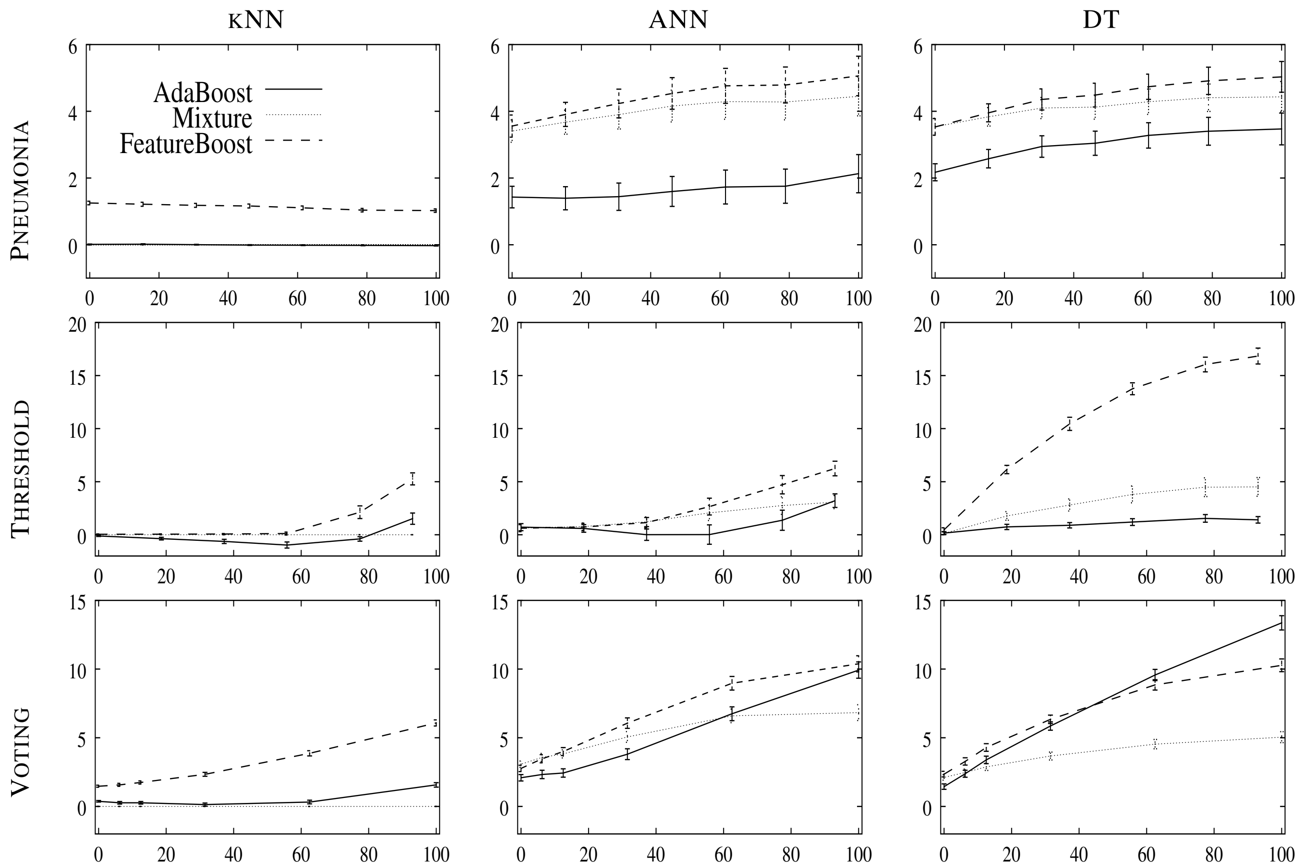

Figure 1. Percent improvement over learning in three domains (PNEUMONIA, THRESHOLD, and VOTING) for three learners (kNN, ANN and DT) and for three ways of generating experts. Results are presented with 95% confidence intervals from 100 or more trial using 20 experts. The x axis is the percent random corruption of the individual features. The y axis is the percent difference between the error of LEARN and the error of the other methods.

cluded, or corrupted.

FeatureBoost is a meta-learning algorithm that makes the underlying 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 features, and combine these models weighted by their estimated accuracy. This approach is impractical, of course, for problems having more than a few attributes as the power set is too large.

Despite the success of FeatureBoost on the test 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 label. 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 this separation is rarely so clean and it may be best to use overlapping subsets of features.

We presented a "hard" version of DEEMPHASIZE in FeatureBoost. We can also conceive of "soft" versions. All the learning algorithms we examined allow other ways of deemphasizing features.

1. For Neural Nets, multiply the value of a feature by it's emphasis. While it is theoretically possible that this alteration does not change the learned hypothesis, in practice the hypothesis does vary.
2. For K nearest neighbor, use a weighted inner product.
3. For decision trees, multiply the information gain of a feature by that features weight before comparing it to other features.

The difference between corrupted features and missing features is important. If values are missing, a technique similar to "sleeping experts (Freund et al., 1997)" may be more effective. In this setting, there are many sub-hypotheses that taken together are combined to form a more robust hypoth-

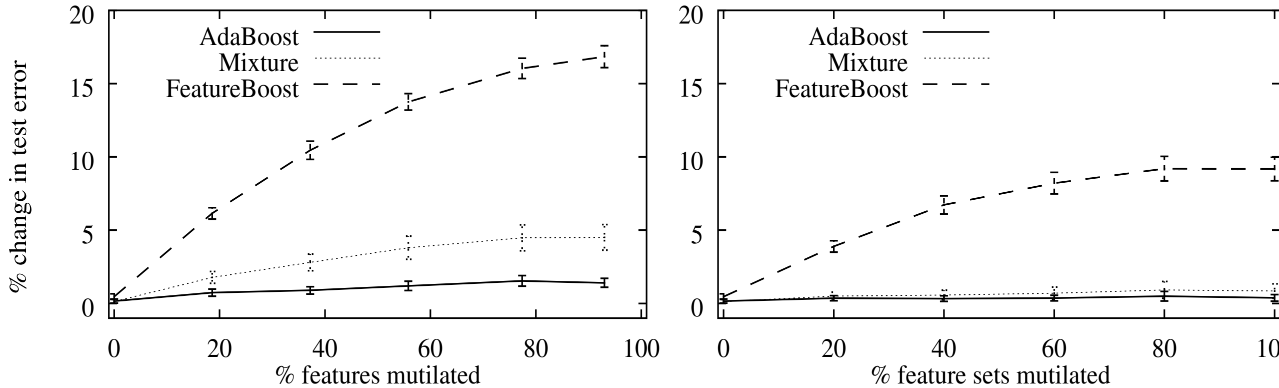

Figure 2. The improvement from boosting DT by FeatureBoost, Mixture Models, and AdaBoost on THRESHOLD. On the left, individual features are randomly corrupted. On the right, corruption happens to entire *sets* of features. Results are presented with 95% confidence intervals from 100 trials using 20 experts. The y axis is the percent difference between the error of LEARN and the error of other methods (higher is better).

esis using weighted majority where 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.

Just as there are several algorithms for boosting examples, we believe there will be several algorithms for boosting features. In fact, we can augment AdaBoost to make it more robust. If we corrupt *training* examples before using AdaBoost, AdaBoost eventually places emphasis on examples where the main features have been corrupted, and thus learns models that depend on other features. We find that AdaBoosting corrupted training sets performs well when the corruption in the training set is identical to the corruption in the test set, but can perform poorly when the test set is corrupted differently than the training set. Combining AdaBoost with FeatureBoost may yield the best of both methods. Note that while we have begun to develop a theoretical framework for robust learning, we do not yet have theoretical guarantees for FeatureBoost comparable to those for AdaBoost.

7. Related Work

Pomerleau noted that ALVINN nets learned models that failed when unexpected new features arose (guardrails) or expected features were obscured (road edges):

> "The appearance or disappearance of irrelevant features can disrupt a network's driving when the network's training did not demonstrate their irrelevance…While they may obscure or replace features which appear on a 'normal' image, there remain enough features in the image to at least in theory make driving using the same network possible…The reason that this type of transitory disturbance causes trouble is that each network is trained over a relatively short stretch of road (< 2 miles). As a result, during training the network is not exposed to all possible driving situations it might encounter when driving autonomously" (Pomerleau, 1993).

Clearly our motivation for robust learning is very similar to Pomerleau's. He devised a method called "structured noise" that bears some resemblance to the process of corrupting selected features with noise in FeatureBoost. His structured noise method added or subtracted (occluded) localized 2D regions in road images in the training set as a way of biasing the network to be more robust to transitory feature inclusion and occlusion. Unlike the FeatureBoost meta-algorithm, his goal was to train one network to be more robust, not to train multiple models to use different subsets of features. We suspect that FeatureBoost could benefit from Pomerleau's method of corrupting localized 2D regions in images when applying FeatureBoost to image recognition problems.

Robust learning is related to the UAV (unspecified attribute value) (Goldman et al., 1997) and RFA (restricted focus-of-attention) (Ben-David & Dichterman, 1998) models of learning studied in the Computational Learning Theory literature. The UAV model is a query model in which both the training and test data may have missing features, and given a partially-specified example, the goal is to answer whether or not all completions of it have the same label (and to output the correct label if they do). In RFA learning, the learning algorithm is only allowed to examine a small number of features in each training example, and its goal is to produce a hypothesis that has low PAC-style error over fully specified data. Our framework is similar to the RFA setup, except the roles of training and test data are reversed, and we do not assume that the learning algorithm can choose which features are missing.

8. Future Work

The framework we present for robust learning does not encompass all ways in which input features might differ in the train and test distributions. FeatureBoost assumes that this difference can be modelled as random corruption and elimination of selected features. While this is an effective means of focussing the learner's attention *away* from some features, it probably is not an accurate model of how features get corrupted in the real world. For example, features in images tend to be corrupted in connected regions. In medicine, procedures may return results for dozens of measurements at a time. In both examples, features come or go in *clumps*. The way in which corrupted features clump depends on the domain. It may be useful to devise specialized versions of robust learning where the data corruption model better fits the processes that affect features in each domain.

An alternate approach to robust learning that we are examining uses feature selection to train models on compact sets of features. It then removes the selected features, and trains additional models on the remaining features. This approach may better model domains such as medicine where features being missing is more common than features being corrupted or occluded.

FeatureBoost, which is based on one model of robust learning, has limitations. One limitation is that the current version of FeatureBoost is restricted to classification problems. This is why we were unable to test FeatureBoost on the autonomous vehicle steering problem that initially motivated our investigation. We are currently developing an extension to FeatureBoost that can handle regression problems.

We are not yet able to provide theoretical guarantees for FeatureBoost similar to those available for boosting algorithms such as AdaBoost. Part of the difficulty stems from the fact that robust learning is most useful when the train and test distribution differ in ways that may be difficult to characterize. One of our goals is to find restricted (though possibly less realistic) models of robust learning where we will be able to make strong theoretical guarantees.

9. Summary

Most machine learning algorithms extract the minimum information from the training set they need to accurately predict the labels. Once they learn a model that performs well on the training data, they stop learning. This *laziness* makes the models they learn less robust on future test cases that have missing or occluded features. Often there is redundancy in the inputs that could have been exploited to learn models more robust to the loss or corruption of some input features. A learning algorithm that was less lazy would try to learn as many different models from the available inputs as possible.

Motivated by this, we introduced a general learning framework called *robust learning*. We then presented a meta-learning algorithm called FeatureBoost that makes the machine learning algorithm it is applied to more robust. We demonstrated FeatureBoost with three different learning methods: backprop nets, k-nearest neighbor, and decision trees. The results suggest FeatureBoost can use these methods to learn models that are more robust to the loss of important features in the test data.

References

Bay, S. D. (1998). Combining nearest neighbor classifiers through multiple feature subsets. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 37–45). San Francisco: Morgan Kaufmann.

Ben-David, S., & Dichterman, E. (1998). Learning with restricted focus of attention. *Journal of Computer and System Science*, *56*, 277–298.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the 11th Annual Conference on Computational Learning Theory* (pp. 92–100). New York, NY: ACM Press.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, *2*, 63–73.

Caruana, R. (1997). *Multitask learning*. Doctoral dissertation, School of Computer Science, Carnegie Mellon University.

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., & Spirtes, P. (1997). An evaluation of machine learning methods for predicting pneumonia mortality. *Artificial Intelligence in Medicine 9*, 107–138.

Freund, Y., & Schapire, R. E. (1995). A decision theoretic generalization of on-line learning and an application to boosting. *Proceedings of the Second European Conference on Computational Learning Theory* (pp. 23–37). Springer-Verlag.

Freund, Y., Schapire, R. E., Singer, Y., & Warmuth, M. K. (1997). Using and combining predictors that specialize. *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing* (pp. 334–343).

Goldman, S., Kwek, S., & Scott, S. (1997). Learning from examples with unspecified attribute values. *Proceedings of the 10th Annual Conference on Computational Learning Theory* (pp. 231–242). New York, NY: ACM Press.

Pomerleau, D. A. (1993). *Neural network perception for mobile robot guidance*. Boston: Kluwer Academic Publishers.