Robust Boosting for Learning from Few Examples

Lior Wolf & Ian Martin
The Center for Biological and Computational Learning
The McGovern institute for brain research
Massachusetts Institute of Technology

Abstract

We present and analyze a novel regularization technique based on enhancing our dataset with corrupted copies of our original data. The motivation is that since the learning algorithm lacks information about which parts of the data are reliable, it has to make more robust classification functions. Using this framework, we propose a simple addition to the gentle boosting algorithm which enables it to work with only a few examples. We test this new algorithm on a variety of datasets and show convincing results.

1. Introduction

Boosting - the iterative combination of classifiers to build a strong classifier - is a popular learning technique. The algorithms based on it, such as AdaBoost and gentleBoost, are easy to implement, work reasonably fast, and in general produce classifiers with good generalization properties for large enough datasets. If the dataset is not large enough and there are many features, these algorithms tend to overfit and perform much worse than the popular Support Vector Machine (SVM) algorithm. SVM can be successfully applied to all datasets, from small to very large.

The major drawback of SVM is that it uses, at run time, when classifying a new example x, all the measurements (features) of x. This poses a problem because, while we would like to cover many promising features during training, computing all the features at run-time might be too costly. This is especially true for object detection problems in vision, where we often need to search the whole image in several scales over thousands of possible locations, each location producing one such vector x. While several approaches for combining feature selection with SVM have been suggested in the past (e.g., [14]), they are rarely used.

The complexity of the feature vector can be controlled more easily by using boosting techniques over weak classifiers based on single features (e.g., regression stumps), such as in the highly successful system of [13]. In this case, the number of features used is bounded by the number of iterations in the boosting process. However, since boosting tends to overfit on small datasets, there is a bit of a dilemma here. An ideal algorithm would enable good control over the total number of features, while being able to learn from only a few examples. Such an algorithm is presented in Sec. 4.

This new algorithm is based on gentleBoost. Within it we implemented a regularization technique based on a simple idea: add corrupted copies of your training dataset to the original one, and the algorithm will not be able to overfit. A general background on fitting and regularization is given in the next Section.

2. Background

We are given a set of n examples $z_i = \{(x_i, y_i)\}_{i=1}^n$, $x \in \mathcal{X}$, $y \in \mathcal{Y}$ drawn from a joint distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$. The ultimate goal of the learning algorithm is to produce a function $f : \mathcal{X} \to \mathcal{Y}$ such that the *expected error* of f given by the expression $\mathsf{E}_{(x,y)\sim\mathcal{P}}(f(x) \neq y)$ is minimized. The boolean expression inside the parentheses evaluates to one if it holds, zero otherwise.

Since we do not know the distribution $\mathcal{P}$ we are tempted to minimize the *empirical error* given by $\sum_{i=1}^n (f(x_i) \neq y_i)$. The problem is that if the space of functions from which the learning algorithm selects f is too large, we are at risk of *overfitting* (learning to deal *only* with the training error). Therefore, while the empirical error is small, the expected error is large. In other words, the *generalization error* (the difference of empirical error from expected error) is large. Overfitting can be avoided by using any one of several *regularization* techniques.

Overfitting is usually the result of allowing too much freedom in the selection of the function f. Thus, the most basic regularization technique is to limit the number of free parameters we use while fitting the function f. For example, in binary classification we may limit ourselves to learning functions of the form $f(x) = (h^\top x > 0)$ (we assume $\mathcal{X} = \Re^n$. h is a vector of free parameters). Using such functions, we reduce the risk of overfitting, but may never optimally learn the *target function* (i.e., the "true function" $f(x) = y$ that is behind the distribution $\mathcal{P}$) of other forms, e.g., we will not be able to learn $f(x) = (x(1)^2 - x(2) > 0)$.

Another regularization technique is to minimize the empirical error subject to constraints on the learned functions.

For example, we can require that the norm of the vector of free parameters h be less than one. A related but different regularization technique is to minimize the empirical error together with a penalty term on the complexity of the function we fit. The most popular penalty term – *Tikhonov regularization* – has a quadratic form. Using the linear model above, an appropriate penalty function would be $||h||_2^2$, and we would minimize $\sum_{i=1}^{n}((h^\top x_i > 0) \neq y_i) + ||h||_2^2$.

Sometimes, adding a regularization term to the optimization problem solved by the algorithm is not trivial. In the most extreme case, the algorithm is a black box we cannot alter at all. Still, a simple form of regularization called noise injection can be employed. In the noise injection technique, the training dataset is enriched by multiple copies of each training data point x_i. A zero-mean, low-variance Gaussian noise (independent for each coordinate) is added to each copy, and the original label y_i is preserved. The motivation is that if two data points x, x' are close (i.e., $||x - x'||$ is small), we would like $f(x)$ and $f(x')$ to have similar values. By introducing many examples with similar x values, and identical y values we teach the classifier to have this stability property. Hence, the learned function is encouraged to be smooth (at least around the training points).

The study of the noise injection technique, which blossomed in the mid 90's, established the following results on noise injection: (1) It is an effective way to reduce generalization error. (2) It has a similar effect on shrinkage (the statistical term for regularization) of the parameters in some simple models (e.g., [2]). (3) It is equivalent to Tikhonov regularization [1]. Note that this does not mean that we can always use Tikhonov regularization instead of noise injection, as for some learning algorithms it is not possible to create a regularized version.

The technique we introduce next is similar in spirit to noise injection. However, it is different enough that the results obtained for noise injection will not hold for it. For example, the results of [1] use a Taylor expansion around the original data points. Such an approximation will not hold for our new technique, since the "noise" is too large (i.e., the new datapoint is too different). Other important properties that might not hold are the independence of noise across coordinates, and the zero mean of the noise.

Our regularization technique is based on creating corrupted copies of the dataset. Each new data point is a copy of one original training point, picked at random, where one random coordinate (feature) is replaced with a different value–usually the value of the same coordinate in another random training example. The basic procedure used to generate the new example is illustrated in Fig. 1. We call it the feature knock out (KO) procedure, since one feature value is being altered dramatically. It is repeated many times to create new examples. It can be used with any learning algorithm, and we use it in the analysis presented in Sec. 3.

Input: (x_1,y_1), ..., (x_m,y_m) where $x_i \in \Re^n$, $y_i \in Y$.
Output: one synthesized pair ($\hat{x}, \hat{y}$).

```
1. Select two examples $x_a, x_b$ at random.
2. Select a random feature $k \in [1..n]$.
3. Set $\hat{x} \leftarrow x_a$ and $\hat{y} \leftarrow y_a$.
4. Replace feature $k$ of $\hat{x}$: $\hat{x}(k) \leftarrow x_b(k)$.
```

Figure 1: The Feature Knockout Procedure

However, as we focus our application emphasis on boosting, we use the specialized version in Fig. 2.

The KO regularization technique is especially suited for use when learning from only a few examples. The robustness we demand from the selected classification function is much more than local smoothness around the classification points (c.f. noise injection). This kind of smoothness is easy to achieve when example points are far from one another. Our regularization, however, is less restrictive than demanding uniform smoothness (Tikhonov) or requiring the reduction of as many parameters as possible. Both of these approaches might not be ideal when only a few examples are available because there is nothing to balance a large amount of uniform smoothness, and it is easy to fit a model that uses very few parameters. Instead, we encourage redundancy in the classifier since, in contrast to the shortage of training examples, there is an abundance of features.

3. Analysis

The effect of adding noise to the training data depends on the learning algorithm used, and is highly complex. Even for the case of adding a zero-mean, low-variance Gaussian noise (noise injection) this effect was studied only for simple algorithms (e.g. [2]) or the square loss function [1].

In Sec. 3.1 we study the effect of feature knock-out on the well known linear least square regression problem. We show that it leads to a scaled version of Tikhonov regularization. Compare this to Bishop's result (using a Taylor expansion) that noise injection is equivalent to Tikhonov regularization. Following in Sec. 3.2, we will try to analyze how feature KO affects the variance of the learned classifier.

3.1. Effect of feature KO on linear regression

One of the most basic models we can apply to the data is the linear model. In this model, the input examples $x_i \in \Re^n, i = 1..m$ are organized as the columns of the matrix $A \in \Re^{n \times m}$; the corresponding y_i values are stacked in one vector $y \in \Re^m$. The prediction made by the model is given by $X^\top h$, where h is the vector of free parameters we have to fit to the data. In the common least squares case, $||y - X^\top h||^2$ is minimized.

In the case that the matrix A is full rank and overdetermined, it is well known that the optimal solution is $h = A^{+}y$, where $A^{+} = (AA^{\top})^{-1}A$ is known as the pseudo inverse of the transpose of A (our definition of A is the transpose of the common text book definition). If A is not full rank, the matrix inverse $(AA^{\top})^{-1}$ is not well defined. However, as an operator in the range of A it is well defined, and the above expression still holds, i.e., even if there is an ambiguity in selecting the inverse matrix, there is no ambiguity in the operation of all possible matrices on the range of the columns of A, which is what we care about.

Even so, if the covariance matrix $(AA^{\top})$ has a large condition number (i.e., it is close to being singular), small perturbations of the data result in large changes to h, and the system is unstable. The solution fits the data A well, but does not fit data which is very close to A, hence there is overfitting. To stabilize the system, we apply regularization.

Tikhonov regularization is based on minimizing $||y - X^{\top}h||^2 + \lambda||h||^2$. This is equivalent to using a regularized pseudo inverse: $A_{\lambda}^{+} = (AA^{\top} + \lambda I)^{-1}A$, where I is the identity $n \times n$ matrix, and λ is the regularization parameter.

In many applications, the linear system we need to solve is badly scaled, e.g., one variable is much larger in magnitude than the other variables. In order to rectify this, we may apply a transformation to the data that weights each variable differently, or equivalently weight the vector h by applying a diagonal matrix D, such that h becomes $\hat{h} = Dh$.

Instead of solving the original system $Ah = y$, we now solve the system $\hat{A}\hat{h} = y$, where $\hat{A} = D^{-1}A$. Solving this system using Tikhonov regularization is termed "scaled Tikhonov regularization." If D is unknown, a natural choice is the diagonal matrix with the entries $D_{kk} = \sqrt{(AA^{\top})_{kk}}$ [9]. We will now show that using the knock-out procedure to add many new examples is equivalent to scaled Tikhonov regularization, using the weight matrix above.

Lemma 1 *When using the linear model with a least squares fit, applying the knock out procedure in Fig. 1 to generate many examples is equivalent to applying scaled Tikhonov regularization where $D_{kk} = \sqrt{(AA^{\top})_{kk}}$.*

Proof see [15]

To get a better understanding of the way feature knock out works, we study the behavior of scaled Tikhonov regularization. In the boosting case, the knock out procedure is expected to produce solutions which make use of more features. Are these models more complex? This is hard to define in the general case, but easy to answer in the linear least square case study.

In linear models, the predictions $\breve{y}$ on the training data take the form: $\breve{y} = Py$. For example, in the unregularized pseudo inverse case we have $\breve{y} = A^{\top}h = A^{\top}(AA^{\top})^{-1}Ay$, and therefore $P = A^{\top}(AA^{\top})^{-1}A$. There is a simple measure of complexity called *the effective degrees of freedom* [7], which is just $Tr(P)$ for linear models. A model with $P = I$ (the identity matrix) has zero training error, but may overfit. In the full rank case, it has as many effective degrees of freedom as the number of features ($Tr(P) = n$).

Lemma 2 *The linear model obtained using scaled Tikhonov regularization has a lower effective degree of freedom than the linear model obtained using unregularized least squares.*

Proof see [15].

Similar to the work done on noise injection, we examined the effect of our procedure on a simple regression technique. We saw that feature knock out resembles the effect of scaled Tikhonov regularization, i.e., high norm features are penalized by the knock out procedure. However, boosting over regressions stumps seems to be scale invariant. Multiplying all the values of a feature by some constant does not change the resulting classifier, since the process that fits the regression stumps (see Sec. 4) uses the values of each feature to determine the thresholds that it uses. However, a closer look reveals the connection between scaling and the effect of the knock out procedure on boosting. *Boosting over stumps* (e.g., [13]) chooses at each round one out of n features, and one threshold for this feature. The thresholds are picked from the m possible values that exist in between every two sorted feature values. The feature and the threshold define a "weak classifier" (the basic building blocks of the ensemble classifier built by the boosting procedure [10]), which predicts -1 or +1 according to the threshold. Equivalently, we can say that boosting over stumps chooses from a set of nm binary features – these features are exactly the values returned by the weak classifiers. These nm features have different norms, and are not scale invariant. Let us call each such feature an nm-feature.

Using the intuitions of the linear least squares case, we would like to inhibit features of high magnitude. All nm-features have the same norm ($\sqrt{(m)}$), but different entropies (a measure which is highly related to norm). These entropies depend only on the ratio of positive values in each nm-feature - call this ratio p.

Creating new examples using the feature knock-out procedure does not change the number of possible thresholds, and therefore the number of features remains the same. The values of the new example in the nm feature space will be the same for all features originating from the $n - 1$ features that were not changed in the knockout procedure. The value for a knocked-out feature (feature k in Fig. 1), will change if the new value is on the other side of the threshold as compared to the old value. This will happen with probability $2p(1 - p)$. If this sign flip happens then the feature is inhibited because it gives two different classifications to two examples with the same label (KO leaves labels unchanged). Note that the entropy of a feature with a positive ratio of p and the probability $2p(1-p)$ behave similarly: both rise monotonically for $0 \leq p \leq 1/2$ and then drop symmetrically. Hence, We obtain the following result:

Lemma 3 *Let t be a single nm-feature created by combining a single input feature with a threshold. The amount of inhibition t undergoes, as the result of applying feature knockout, grows monotonically with the entropy of p.*

Hence, similarly to the scaling in the linear case, the knock out procedure inhibits high magnitude features (here the magnitude is measured by the entropy). Note that in the algorithm presented in Sec. 4, a feature is used for knock-out only after it was selected to be a part of the output classifier. Still, KO inhibits more weak classifiers based on these features with higher entropies, making them less likely to get picked again. It is possible to perform this higher-entropy preferential inhibition directly on all features, therefore simulating the full knock-out procedure. The implementation of this is left for future experiments.

3.2. Bias/variance decompositions

Many training algorithms can be interpreted as trying to minimize a cost function of the form $\sum_{i=1}^{n} L(f(x_i), y_i)$, where L is a loss function. For example, in the $0/1$ loss function $L(f(x), y) = (f(x) \neq y)$, we pay 1 if the labels are different, 0 otherwise. By applying the knock-out procedure to generate more training data, an algorithm that minimizes such a cost function will actually minimize: $\sum_{i=1}^{n} \mathsf{E}_{\hat{x} \sim C_X(x_i)} L(f(\hat{x}), y_i)$, where $C_X(x)$ represents the distribution of all knocked-out examples created from x.

Consider a bias-variance decomposition based on the $0/1$ loss function, as analyzed in [3]. We follow the terminology of [3] with a somewhat different derivation, and for the presentation below we include a simplified version. Assume for simplicity that each training example occurs in our dataset with only one label, i.e., if $x_i = x_j$ then $y_i = y_j$. Define the *optimal prediction* f_* to be the "true" label $f_*(x_i) = y_i$. Define the *main prediction* of a function f to be just the prediction $f(x)$. The *bias* is defined to be the loss between the optimal and main predictions: $B(x) = (f(x) \neq f_*(x))$. The *variance* $V(x)$ is defined to be the expected loss of the prediction with regard to the main prediction: $V(x) = \mathsf{E}_{\hat{x} \sim C_X(x)}(f(x) \neq f(\hat{x}))$. These definitions allow us to present the following observation:

Observation 1 *Let $B0$ be the set of all training- example-indices for which the bias $B(x_i)$ is zero (the unbiased set). Let $B1$ be the set for which $B(x_1) = 1$ (the biased set). Then, $\sum_{i=1}^{n} \mathsf{E}_{\hat{x} \sim C_X(x_i)}(f(\hat{x}) \neq y_i) = \sum_{i=i}^{m} B(x_i) + \sum_{i \in B0} V(x_i) - \sum_{i \in B1} V(x_i)$*

In the unbiased case ($B(x) = 0$), the variance ($V(x)$) increases the training error. In the biased case ($B(x) = 1$), the variance at point x decreases the error. A function f, which minimizes the training cost function that was obtained using feature knock-out, has to deal with these two types of variance directly while training. Define the net variance to be the difference of the biased variance from the unbiased; a function trained using the feature knock-out procedure is then expected to have a higher net variance than a function trained without this procedure. If we assume our corruption process C_X is a reasonable model of the robustness expected from our classifier, a good classifier would have a high net variance on the testing data. The net variance measured in our experiments [15] shows the effect of the feature knockout approach.

4. The gentleBoostKO algorithm

While our regularization procedure can be applied, in principle, to any learning algorithm, using it directly when the number of features n is high might be computationally demanding. This is because for each one of the m training examples, as many as $n(m-1)$ new examples can be created. Covering even a small portion of this space might require the creation of many synthesized examples.

However, for some algorithms our regularization technique can be applied with very little overhead. For boosting over regression stumps, it is sufficient to modify those features that participate in the trained ensemble (i.e., those features that actually participate in the classification).

The basic algorithm used in our experiments is specified in Fig. 2. It is a modified version of the gentleBoost algorithm [6]. gentleBoost seems to converge faster than AdaBoost, and performs better for object detection problems [12]. At each boosting round, a regression function is fitted (by weighted least-squared error) to each feature in the training set. We used linear regression for our experiments, fitting parameters a, b and th so that our regression functions are of the form $f(x) = a(x > th) + b$. The regression function with the least weighted squared error is added to the total classifier $H(x)$ and its associated feature (k_{min}) is used for Feature Knockout (step d).

In the Feature Knockout step, a new example is created using the class of a randomly selected example x_a and all of its feature values except for the value at k_{min}. The value for this feature is taken from a second randomly-selected example x_b. The new example x_{m+t} is then appended to the training set. In order to quantify the importance of the new example in the boosting process, a weight has to be assigned to it. The weight w_{m+t} of the new example is estimated by copying the weight of the example from which most of the features are taken (x_a). Alternatively, a more precise weight can be determined by applying the total classifier $H(x)$ to the new example.

As with any boosting procedure, each iteration ends with

Input: (x_1,y_1), ..., (x_m,y_m) where $x_i \in \Re^n$, $y_i \in Y = \pm 1$.
Output: Composite classifier $H(x)$.

```
1. Initialize weights $w_i \leftarrow 1/m$.
2. for $t = 1, 2, 3, ...T$.
    (a) For each feature $k$, fit a regression function $f_t^{(k)}(x)$ by
        weighted least squares on $y_i$ to $x_i$ with weights $w_i$, $i = 1..m+t-1$.
    (b) Let $k_{min}$ be the index of the feature with the minimal associated
        weighted least square error.
    (c) Update the classifier $H(x) \leftarrow H(x) + f_t^{(k_{min})}$
    (d) Use Feature KO to create a new example $x_{m+t}$:
            Select two random indices $1 \leq a, b \leq m$
            $x_{m+t} \leftarrow x_a$
            $x_{m+t}(k_{min}) \leftarrow x_b(k_{min})$
            $y_{m+t} \leftarrow y_a$
    (e) Set new example weight $w_{m+t}$ to that of its source:
            $w_{m+t} \leftarrow w_a$
    (f) Update the weights and normalize:
            $w_i \leftarrow w_i e^{-y_i f_t^{(k_{min})}(x_i)}, \; i = 1..m+t$
            $w_i \leftarrow w_i / \sum_{i=1}^{m+t} w_i$
3. Output the final classifier $H(x)$
```

Figure 2: The GentleBoostKO Algorithm

the update of the weights of all examples (including the new one), and a new round of boosting begins. This iterative process finishes when the weights of the examples converge, or after a fixed number of iterations. In our experiments, we stopped the boosting after 100 rounds–enough to ensure convergence in all cases.

5. Experiments

Visual recognition using the Caltech datasets. We tested our gentleBoostKO algorithm on several Caltech object recognition datasets that were presented in [5]. In each experiment we had to distinguish between images containing an object and background images that do not contain the object. The datasets: Airplanes, Cars, Faces, Leafs and Motorbikes, as well as the background images were downloaded from `http://www.vision.caltech.edu/`.For the experiments we used the predefined splits (available to all the datasets but the Leafs dataset). For leafs, we used a random split of 50% training and 50% testing. Note that since our methods are discriminative, we needed a negative training set. For this end, we removed 30 random examples from the negative testing set, and used them for training.

To turn each image into feature-vectors we used 500 C2 features [11]. These extremely successful features allow us to learn to recognize objects using few training images, and the results seem to be comparable or better than the results reported in [4]. The results are shown in Fig. 3. To compare with previous work, we used the error at the equilibrium-point between false and true positives as our error-measure. It is clear that for a few dozen examples, SVM, gentleBoost and gentleBoostKO have the same performance level. However, for only a few training examples, gentleBoost does not perform as well as SVM, while gentleBoostKO achieves the same level of performance.

We also tried to apply Lowe's SIFT features [8] to the same datasets, although these features were designed for a different task. For each image, we used Lowe's binaries to comute the SIFT description of each key point. We then sampled from the training set 1000 random keypoints $k_1, ..., k_{1000}$. Let $\{k_i^I\}$ be the set of all keypoints associated with image I. We represented each training and testing image I by a vector of 1000 elements: $[v^I(1)...v^I(1000)]$, such that $v^I(j) = min_i ||k_j - k_i^I||$. Note that in [8] the use of the ratio of distances between the closest and the next closest points were encouraged (and not just the minimum distance). For our application, which disregards all geometric information, we found that using the minimum gives much better results. For the testing and training splits reported in [5] we got the following results (ME=mean error, EqE=error at equilibrium):

Algorithm	Planes	Cars	Faces	Leaves	Motor.
Lin. SVM ME	0.104	0.019	0.107	0.118	0.033
gentleB ME	0.118	0.036	0.168	0.137	0.026
gentleBKO ME	0.100	0.033	0.119	0.114	0.023
Lin. SVM EqE	0.108	0.018	0.111	0.126	0.007
gentleB EqE	0.120	0.037	0.166	0.132	0.003
gentleBKO EqE	0.111	0.030	0.136	0.120	0.008

Car type identification. This dataset consists of 480 images of private cars, and 248 images of mid sized vehicles (such as SUV's). All images are 20×20 pixels, and were collected using Mobileye's car detector, on a video stream taken from the front window of a moving car. The task is to learn to identify private cars from mid sized vehicles, which has some safety applications. Taking into account the low resolution and the variability in the two classes, this is a difficult task. The results are shown on the bottom right corner of Fig. 3. Each point of the graph shows the mean error when applying the algorithms to training sets of different size (between 5 and 40 percent of the data). The rest of the examples were used for testing. It is evident that for this dataset gentleBoost outperforms SVM. Still, gentleBoostKO does even better.

Acknowledgments

This report describes research done at the Center for Biological & Computational Learning, which is in the McGovern Institute for Brain Research at MIT, as well as in the Dept. of Brain & Cognitive Sciences, and which is affiliated with the Computer

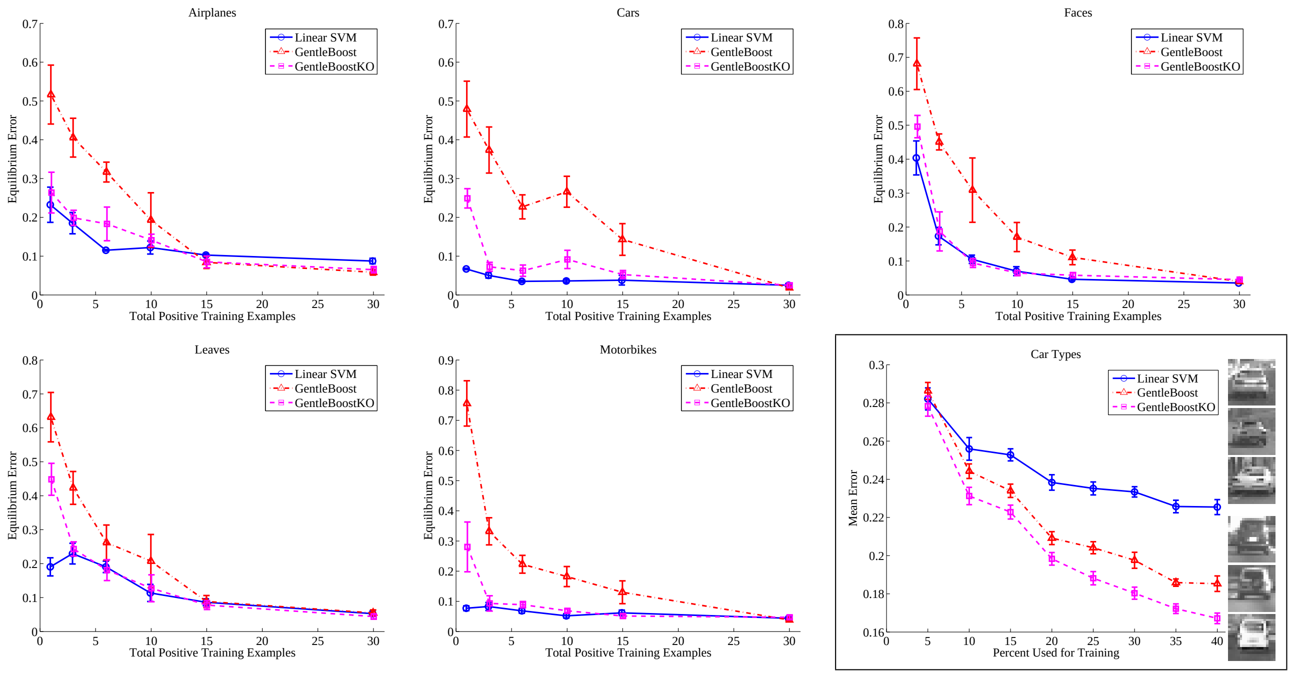

Figure 3: A comparison using the C2 features between gentleBoost, gentleBoostKO (Fig. 2), and linear SVM on the five Caltech datasets: Airplanes, Cars, Faces, Leafs and Motorbikes. The graphs show the the equilibrium error rate vs. the number of training examples used from the class we want to detect. In each experiment, the test set was fixed to be the same as those described in [5]. **Lower right corner:** The results of applying the three algorithms to the car types dataset, together with example images. The results shown are mean and standard error of 30 independent experiments versus percentile of training images.

Sciences & Artificial Intelligence Laboratory (CSAIL). This research was sponsored by grants from: Office of Naval Research (DARPA) Contract No. MDA972-04-1-0037, Office of Naval Research (DARPA) Contract No. N00014-02-1-0915, National Science Foundation (ITR/SYS) Contract No. IIS-0112991, National Science Foundation (ITR) Contract No. IIS-0209289, National Science Foundation-NIH (CRCNS) Contract No. EIA-0218693, National Science Foundation-NIH (CRCNS) Contract No. EIA-0218506, and National Institutes of Health (Conte) Contract No. 1 P20 MH66239-01A1. Additional support was provided by: Central Research Institute of Electric Power Industry (CRIEPI), Daimler-Chrysler AG, Compaq/Digital Equipment Corporation, Eastman Kodak Company, Honda R&D Co., Ltd., Industrial Technology Research Institute (ITRI), Komatsu Ltd., Eugene McDermott Foundation, Merrill-Lynch, NEC Fund, Oxygen, Siemens Corporate Research, Inc., Sony, Sumitomo Metal Industries, and Toyota Motor Corporation.

References

[1] C.M Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 1995.

[2] L. Breiman. Heuristics of Instability and Stabilization in Model Selection. *Ann. Statist.*, 1996.

[3] P. Domingos. A Unifies Bias-Variance Decomposition for Zero-One and Squared Loss. *Proc. Int. Conf. AI*, 2000.

[4] L. Fei-Fei, R. Fergus, & P. Perona. A Bayesian approach to unsupervised 1-Shot learning of Object categories. *ICCV*03.

[5] R. Fergus, P. Perona, and A. Zisserman. Object Class Recognition by Unsupervised Scale-Invariant Learning. *CVPR* 2003

[6] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Ann. Statist.*, 2000.

[7] T. Hastie, R. Tibshirani, J. H. Friedman The Elements of Statistical Learning Springer, 2001.

[8] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[9] A. Neumaier. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review* 1998.

[10] R.E. Schapire. A brief introduction to boosting. Proc. Int. Joint Conf. AI, 1999.

[11] T. Serre, L. Wolf and T. Poggio. A New Biologically Motivated Framework for Robust Object Recognition. *CVPR*, 2005.

[12] A. Torralba, K.P. Murphy and W.T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. *CVPR*, 2004.

[13] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 2001.

[14] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio and V. Vapnik. Feature Selection for SVMs. *NIPS*, 2001.

[15] L. Wolf and I. Martin. Regularization Through Feature Knock Out. MIT CSAIL TR: CBCL-242, 2004.