Robust Learning and Segmentation for Scene Understanding

by

Ian Stefan Martin

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005

Author . . . .
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by . . . .
Tomaso Poggio
Eugene McDermott Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Robust Learning and Segmentation for Scene Understanding

by

Ian Stefan Martin

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis demonstrates methods useful in learning to understand images from only a few examples, but they are by no means limited to this application. Boosting techniques are popular because they learn effective classification functions and identify the most relevant features at the same time. However, in general, they overfit and perform poorly on data sets that contain many features, but few examples. A novel stochastic regularization technique is presented, based on enhancing data sets with corrupted copies of the examples to produce a more robust classifier. This regularization technique enables the gentle boosting algorithm to work well with only a few examples. It is tested on a variety of data sets from various domains, including object recognition and bioinformatics, with convincing results.

In the second part of this work, a novel technique for extracting texture edges is introduced, based on the combination of a patch-based approach, and non-parametric tests of distributions. This technique can reliably detect texture edges using only local information, making it a useful preprocessing step prior to segmentation. Combined with a parametric deformable model, this technique provides smooth boundaries and globally salient structures.

Thesis Supervisor: Tomaso Poggio
Title: Eugene McDermott Professor

Acknowledgments

This thesis is dedicated to
Victor J. Carucci (1928–2003),
engineer, mentor, and forever my friend.

I am very thankful to Tomaso Poggio for his input and supervision, Lior Wolf for his ideas and guidance, and MIT's Center for Biological and Computational Learning (CBCL) for funding this research and providing the necessary resources. I would also like to thank Dimitris Metaxas and Xiaolei Huang of Rutgers for contributing their morphable model to the portion of this work on texture segmentation.

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 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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction: Classification

1.1 Motivation

Classification is one of several primary categories of machine learning problems. In classification, a system is trained to recognize a type of example or differentiate between examples that fall in separate categories. In the case of computer vision, the examples are representations of photographic images and the task of the classifier is to indicate whether or not a specific object or phenomena of interest is present in the image. In order to accomplish this successfully, the classifier must have sufficient prior knowledge about the appearance of the object.

Within the field of computer vision, there is a general consensus that object appearance cannot be captured by a small number of rules or machine instructions. Consider a case where the goal is to identify images containing a chair. A set of rules might include: four vertical legs supporting a horizontal square seat with a perpendicular rectangular back. But chairs may come in many different styles with different numbers of legs or even no legs at all. Characterizing its shape in order to identify unseen chairs using a rule-based system would simply require too many different cases and exceptions, especially considering that a chair could be viewed from any angle. This would easily frustrate the designer of the system since each rule must be constructed manually.

Hence, in vision we resort to an alternative: learning object appearance from examples. Rather than searching for underlying rules or continuity from which to characterize the object, the task is simply to collect images of different instances of the object from various viewing angles and provide them as training examples to a classification algorithm. The training phase produces a classifier that retains some knowledge of the essence of its object of interest in order to correctly classify future examples.

Here, it is important to note that learning is not the same as memorizing. Memorizing every single training example will allow perfect classification of those examples in the future, but it is not clear how to classify unseen examples. Learning is important because it implies the discovery of some general aspect that differs between object categories, thus allowing the classifier to make generalizations about unseen examples.

Training by example does have its own drawback however, since accurate classifiers have traditionally required a large number of properly-labeled examples. Obtaining such training data is usually a laborious task since it involves not only the imaging of many similar but slightly different objects, but also hand-labelling the contents of each image and manually segmenting the images in case they contain more than one class of object. Such work can be time consuming, not to mention mind-numbing.

The need for learning from few training examples is common for many fields in which the collection of annotated data is costly. In computer vision it is motivated by the ability of the primate visual system to learn classes of objects with a lot of interclass variability from only a few examples. While some researchers attribute this ability to the utilization of partly- labeled data (e.g., [4]), it is becoming more evident that this is achievable even without such an auxiliary data set. Most notable is the line of work showing this ability when learning to recognize objects from unsegmented images [20].

These issues are not isolated to the field of computer vision, and indeed may apply to any other field where measured data must be classified automatically and collecting examples is costly. The experiments of Chapter 3 investigate data sets from these related fields in addition to vision data.

1.1.1 The Street Scenes Project

The issues described above are most relevant to Street Scenes, a current project involving real-world still images of street corners. Its goal is to identify all objects in a scene that belong to prescribed categories (cars, trees, buildings, pedestrians, etc.) and indicate where they are located. This is a problem of detecting not only a single class of object, but multiple different types. Systems of the past have only been successful for unobscured images of a specific type of object or a small number of different objects in similar environments.

Unlike the discriminative methods described in this chapter, this problem is not a matter of simply dividing a set of examples into two categories. This is a realm where multiple objects need to be located and identified in the same single image, which has traditionally entailed sliding a neighborhood window over all image positions and computing multiple classifiers at each window position.

To avoid this naïve and time-consuming method, segmentation is sometimes applied, where the image is first divided into regions that seem similar by some metric. Then, each region is classified separately. There are several metrics one can employ in segmentation, e.g. mean shift, but while objects may be multicolored, they often consist of a single texture. Chapter 4 presents the second part of this thesis, examining a method to extract texture edges from natural images that can be used for segmentation. The remainder of this chapter describes popular classification algorithms and regularization methods. Then, Chapter 2 will introduce a new regularization technique.

1.2 Classification Algorithms

Given a set of m examples $z_i = \{(x_i, y_i)\}_{i=1}^{m}$, $x \in \mathcal{X}$, $y \in \mathcal{Y}$ drawn from a joint distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$, the ultimate goal of the learning algorithm is to produce a classifier 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 examples $x_i \in \Re^n$ are vectors of n features, which may equivalently be referred to as measurements or variables. These features can be raw, low-level measurements, such as the intensities of individual pixels in an image, or higher-level representations such as edge-filter outputs or cross-correlation scores. Some feature representations elicit better classification performance than others, but otherwise there is no restriction stating what makes an appropriate feature value.

The two most popular algorithms for classifying data of various types into prescribed categories are *Boosting* and the *Support Vector Machine*. Both perform very well when a large amount of training data is available, but each has a significant drawback, as detailed below.

1.2.1 The Support Vector Machine

The Support Vector Machine (SVM) is an algorithm that locates a decision boundary between the two classes of examples in multidimensional space, such that the margin is maximized. The examples closest to the boundary are called support vectors and their distance from the boundary is the margin. Depending on the type of kernel used, the decision boundary can take on many different shapes, from a n-dimensional hyperplane in the linear case, to a complex, bumpy surface in the case of a Gaussian kernel.

SVM can be successfully applied to all datasets, from small to very large, but its major drawback is that it uses, when classifying a new example x at run time, all the measurements (features) of x. This creates a problem because, while we would like to cover all promising features during training, computing them at run-time might be too costly. This cost is especially important for object detection problems in vision, where it is often necessary to search the whole image in several scales over thousands of possible locations, each location producing one element of x.

SVMs and Feature Selection

Several approaches for combining feature selection with SVM have been suggested in the past [65], but they are rarely used. Recently, a couple of new SVM variants have been suggested, in which a linear SVM is trained multiple times and the relative importance of the features (dimensions) is indicated by their weight in a vector perpendicular to the decision hyperplane. These algorithms are called l_0-norm SVM [28] and SVM-RFE [39]. Another variant, "Wilcoxon SVM," uses the Wilcoxon Mann-Whitney test (see Figure 4-1) to select the most statistically dissimilar features.

1.2.2 Ensemble Methods, Boosting

Ensemble classifiers use voting among a group of base classifiers in order to make a final classification. *Boosting* is a technique that iteratively constructs a strong ensemble classifier from weak base classifiers. A weak classifier is one that performs only slightly better than chance guessing (with an error rate under 50%). The notion that a weighted combination of such classifiers could produce a strong classifier with high accuracy was first shown by Kearns and Valiant [44].

Schapire introduced the first provably polynomial-time boosting algorithm in 1990 and later introduced the first mature boosting algorithm, AdaBoost (short for *Ada*ptive *Boost*ing) [54]. The general study of ensemble classifiers, and variants of AdaBoost in particular, is a very developed field. AdaBoost uses binary weak classifiers in the ensemble. It is a powerful learning algorithm, with a proven record in many applications. It is also very flexible, and easily adapted (e.g.,[60]).

Other variants of the boosting algorithm have been introduced, such as GentleBoost [25], which uses continuous (regression) stumps as the weak learners. GentleBoost is commonly used in machine vision applications because it has good convergence properties and works well in object recognition problems [38]. Our work focuses on boosting over decision stumps [5], simple classifiers obtained by thresholding a single variable. For example, "predict a positive label if the 12th attribute of the input example is larger than 0.7, otherwise predict negative."

AdaBoost and algorithms based on it are easy to implement, work reasonably fast, and in general produce classifiers with good generalization properties for large enough data sets. However, if the data set is not large enough and there are many features, these algorithms tend to perform much worse than the SVM, which can be successfully applied to any data set regardless of size.

1.2.3 Deterministic AdaBoost Variants

In Bioinformatics, a field that at first seems completely unrelated to computer vision, data containing many features but few examples is quite common. For example, with microarray data, the features represent thousands of different genes, while the examples represent human patients and may only be in the tens or hundreds at most. For this reason, a few deterministic variants of AdaBoost have recently been suggested. Following previous work [18] that showed that boosting is not well-suited for expression data, Long and Vega [39] developed a few deterministic modifications of AdaBoost, and showed that these variants perform much better than the traditional AdaBoost on microarray data.

In order to compare our results to theirs, we implemented their two most successful AdaBoost variants: VC and NR. In AdaBoost-VC[1], the empirical weighted error of the classifier chosen at round t is adjusted prior to reweighting. AdaBoost-VC uses $\epsilon_t = \epsilon_t^{emp} + \frac{d}{m}\left(ln\ m + \sqrt{1 + e_t^{emp}/d}\right)$, where m is the number of examples, and d is a free parameter.

The simpler variant, AdaBoost-NR ("No Repeat"), has two modifications over the usual AdaBoost algorithm. First, each gene (variable) is constrained for use in at most one decision stump. Second, a decision stump with an empirical error of zero, is weighted in the final classifier as if it has an empirical error of $0.1/m$.

Since our main method is a modification of GentleBoost (and not AdaBoost) we implemented two variants of GentleBoost that were inspired by Long and Vega. The simpler one is GentleBoost-NR, in which, like AdaBoost-NR, only one stump per gene is included; the VC version of GentleBoost was adapted to accommodate the basic differences between AdaBoost and GentleBoost. For example, AdaBoost uses the global error estimate to update the weights of all the correctly classified training examples, whereas GentleBoost updates each example's weight individually

[1]The initials VC imply that the form of regularization is similar to the one derived using VC bounds on generalization error.

by considering the error of its best base classifier.

We implemented a GentleBoost-VC algorithm in the spirit of AdaBoost-VC. Instead of scaling the weights by $e^{-y \cdot f(x)}$ (as in step (f) of Fig. 2-2), we scale them by $e^{-y \cdot f(x) + sign(y \cdot f(x)) \sqrt{\epsilon_t^{emp}/d}}$, where ϵ_t^{emp} is the squared error of the regression function $f_t(x)$.

1.3 Regularization

Carefully controlling the complexity of the classifier is an important task when training any learning algorithm. If the classifier is not complex enough, it will not be capable of capturing the structure of the training data. On the other hand, an overly complex classifier will focus on the irrelevant noise in the data, leading to overfitting (learning to deal *only* with the training error) and poor generalization ability [6]. Regularization is one way to mediate the complexity of the classifier.

Since we do not know the distribution $\mathcal{P}$, we are tempted to minimize the *empirical error* given by $\sum_{i=1}^{m}(f(x_i) \neq y_i)$. 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. 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 $\mathcal{X} = \Re^m$ and h is a vector of free parameters). Using such functions, the risk of overfitting is reduced, but we 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., it is impossible to learn the polynomial relationship $f(x) = (x(1)^2 - x(2) > 0)$.

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 $\sum_{i=1}^{m}((h^\top x_i > 0) \neq y_i) + ||h||_2^2$.

The complexity of the feature vector can be easily controlled by using boosting techniques over weak classifiers based on single features (e.g., regression stumps), such as in the highly successful system of [64]. In this case, the number of features used is bounded by the number of iterations or rounds in the boosting process. However, since boosting tends to overfit on small data sets, there is a bit of a dilemma here. An ideal algorithm would enable good control over the total number of features used, while being able to learn from only a few examples. The algorithm presented in Section 2.4 demonstrates both these properties.

1.3.1 Enhancing the Data Set

Another method that can be used to regularize the learning process is the creation of virtual examples. 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 that cannot be altered. Enhancing the data set with virtual examples is a universal regularization technique because it can be performed as a preprocessing step prior to training any classification algorithm.

Since we desire the classifier to be able to accurately label examples it has never seen before, we require it to recognize examples similar, but not identical to those it considered during the training process. In other words: no future example is expected to be exactly the same as an example seen in the training process, but its measurements are assumed to be close in value. Hence, regularization using virtual examples simply consists of generating new examples, each based on an existing one, but slightly perturbed in some fashion. Since the learning algorithm sees more variation in the training examples, it is trained to cope with variation in the testing examples.

The downside to regularization through virtual examplesis is that adding more examples to the training set requires more computing time and memory and slows down the training process significantly. It also adds additional parameters to the algorithm (how many virtual examples to add and how much to perturb them) which are not easy to select *a priori*.

1.3.2 Noise Injection

Noise injection is one method of regularizing through enhancement of the data set using virtual examples. The training data is enriched with multiple copies of each 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, the classifier is taught to express this stability property. Hence, the learned function is encouraged to be smooth (at least around the training points). These virtual examples can then be used with any learning algorithm to produce a more robust classifier.

Research efforts of the mid-1990s produced the following results regarding noise injection: (a) It is an effective way to reduce generalization error, e.g., [51]. (b) It has a similar effect on shrinkage (the statistical term for regularization) of the parameters in some simple models (e.g., [10]). (c) It is equivalent to Tikhonov regularization [6]. 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. It is well-known that noise injection leads to an improved generalization ability, but as with any sort of virtual example, it requires more memory space and computing time. It is important to note that noise injection is not suitable for learning from few examples; in this scenario the input examples are probably well separated, and a low variance noise will have little effect.

Chapter 2

Regularization Through Feature Knockout

2.1 Introduction

The technique introduced 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, noise injection used low-variance Gaussian noise, independent to each coordinate. Here, we make use of a very different type of noise: high variance, and dependent between the coordinates. The results of [6] 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 do not hold are the independence of noise across coordinates, and the zero mean of the noise.

This new algorithm (first introduced in [67]) is based on GentleBoost. Within it we implemented a regularization technique based on a simple idea: introduce corrupted copies of the training examples, and the algorithm will not be able to overfit.

This chapter is structured as follows: In Section 2.3, we motivate the use of our method by relating it to well known concepts and to some statistical work. In Section 2.4 we describe our boosting algorithm, and then in Section 2.5 we provide theoretical grounds for the behavior of our methods. Chapter 3 presents experimental

Input: $(x_1, y_1), \ldots, (x_m, y_m)$, 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 $k \in \{1 \ldots n\}$.
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 2-1: The Feature Knockout Procedure

sections demonstrating the performance benefits of our algorithm.

2.2 Feature Knockout

Our regularization technique is based on generating corrupted copies of the data set. Each new data point is a copy of one original training point, selected at random, where one random coordinate (feature) is replaced with a different value–usually the value of the same coordinate in another randomly-selected training example. The basic procedure used to generate the new example is illustrated in Figure 2-1. It is called the Feature Knockout (KO) procedure, since one feature value is being altered dramatically, and it is repeated many times to create new examples. It can be used with any learning algorithm, and is used in the analysis presented in Section 2.5. However, since the application emphasis is on boosting, we use the specialized version presented in Section 2.4.

The KO regularization technique is especially suited for use when learning from only a few examples. The robustness demanded 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. Neither approach is ideal when only a few examples are available, since 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, redundancy is encouraged in the classifier since, in contrast to the shortage of training examples, there might be an abundance of features.

2.3 Notes Regarding Redundancy

Intuition. It is a common belief that simpler is better. *Occam's razor*–"entities should not be multiplied beyond necessity"–is often understood as suggesting the selection of the simplest possible model that fits the data well. Thus, given a data set, one tends to prefer classifier A that uses 70 features over classifier B that uses 85 if they have the same expected error, because the "simpler" classifier is believed to generalize better.

In an apparent contrast to this belief, we know from our daily experience that simpler is not always better. When a good teacher explains something to a class, he will use a lot of repetition. The teacher ensures that the students will understand the idea, even if some of the explanations were unclear. Since the idea could be expressed in a simple form without repetition, his explanation is more complex than necessary.

It is also generally accepted that biological systems use redundancy in their computations. Thus, even if several computational units break down (e.g., when neurons die) the result of the computation is largely unaffected. The learned model is expected to be interpreted with some random error. In these cases, we should not train a single classification function, but instead train to optimize a distribution of such functions.

Redundancy in boosted classifiers. Boosting over a weak classifier increases the weights of those examples it does not classify well. The inclusion of a weak classifier in the strong classifier therefore inhibits future use of similar weak classifiers. In boosting over regression stumps, each weak classifier is based on one feature. Hence, from a group of similar features, one may expect to see no more than a few participating in the strong classifier. Our boosting over regression stumps algorithm, presented in Section 2.4, modulates this effect, by creating a new example for which relying on the selected feature might lead to a mistake. However, it does not change the values of the other features, making the similar features suitable for classifying the new example.

Such a process yields a larger classifier, which uses more features. This effect is clear in our experiments, and might also be interpreted as "a more complex model is needed to deal with more complex data." However, using more features does not necessarily mean that the classifier will lead to a worse generalization error[1].

Koltchinskii and Panchenko have derived bounds on the generalization error of ensemble (voting) classifiers, such as boosting, which take redundancy into consideration [36]. A precise description of their results would require the introduction of more notation, and will not be presented here. Informally speaking, they show that one can refine the measures of complexity used for voting classifiers such that it would encourage ensembles that can be grouped into a small number of compact clusters, each including base ("weak") classifiers that are similar to one another.

2.4 The GentleBoost-KO Algorithm

While our regularization procedure, presented in Figure 2-1, can be applied, in principle, to any learning algorithm, using it directly when the number of features n is high might be computationally demanding. In other words, 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 generation of many synthesized examples.

The randomized procedure in Figure 2-1 samples this large space of synthesized training examples. Still, if there are many features, the sampling would probably be too sparse. However, for some algorithms our regularization technique can be applied with very little overhead. For boosting over regression stumps, it is sufficient

[1]The terms "simple" and "complex" are not trivial to define, and their definition usually depends on what one tries to claim. The next section presents more rigorous definitions for the cases analyzed.

Input: $(x_1, y_1), \ldots, (x_m, y_m)$, 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 $t = 1, 2, 3, \ldots T$:
    (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 $w_i$, $i = 1 \ldots m + t - 1$.
    (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) Copy the new example weight $w_{m+t}$ from $w_a$:
            $w_{m+t} \leftarrow w_a$
    (f) Update the weights and normalize:
            $w_i \leftarrow w_i e^{-y_i f_t^{(k_{min})}(x_i)}, i = 1 \ldots m + t$
            $w_i \leftarrow w_i / \sum_{i=1}^{m+t} w_i$
3. Output the final classifier $H(x)$.
```

Figure 2-2: The GentleBoost-KO Algorithm. Steps d and e constitute the differences from the original GentleBoost.

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 Figure 2-2. It is a modified version of the GentleBoost algorithm [25]. It is based on GentleBoost because GentleBoost seems to converge faster than AdaBoost, and performs better for object detection problems [38, 60]. At each boosting round, a regression function is fitted (by weighted least-squared error) to each feature in the training set. The experiments use weak linear regression classifiers, fitting parameters a, b and a threshold t so that our regression functions are of the form $f(x) = a(x > t) + b$. The regression function with the least weighted squared error is added to the accumulated ensemble 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 generated using the class y_a 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 the update of the weights of all $m + t$ 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 the experiments of Chapter 3, the boosting processes were stopped after 100 rounds–enough to ensure convergence in all cases.

2.5 Analysis

The effect of adding noise to the training data depends on which learning algorithm is 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. [10]) or the square loss function [6]. Section 2.5.1 studies the effect of Feature Knockout on the well-known linear least squares regression problem and shows that it leads to a scaled version of Tikhonov regularization. Although in the experiments feature knockout is applied to boosting (Figure 2-2), we still gain insight from this simple model.

2.5.1 Effect of Feature Knockout on Linear Regression

The linear model is one of the most basic models that can be applied to the data. In this model, the input examples $x_i \in \Re^n, i = 1 \ldots m$ are organized as the columns of the matrix $A \in \Re^{n \times m}$; the corresponding y_i values comprise a single column vector $y \in \Re^m$. The prediction made by the model is given by $A^\top h$, where h is the vector of free parameters to fit to the data. In the common least squares case, $||y - A^\top h||^2$ 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 textbook 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 the important concern.

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, regularization is applied.

Tikhonov regularization is based on minimizing $||y - A^\top h||^2 + \lambda||h||^2$. 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 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 $A^\top h = y$, we now solve the system $\hat{A}^\top \hat{h} = y$, 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 $D_{kk} = \sqrt{(AA^\top)_{kk}}$ [45]. Next, it will be shown that using the knockout procedure to add many new examples is equivalent to scaled Tikhonov regularization, using the weight matrix above.

The lemma below if simpler to state and prove when the data is normalized such that each variable is centered, i.e., has a mean of zero. The lemma considers the case of infinitely many new training examples, created by the knockout procedure. The case when only a limited number of such examples are created gives rise to results that are much more complex to state.

Lemma 1 *When using the linear model with a least squares fit on centered data, applying the regularization procedure in Figure 2-1 to generate many examples is equivalent to applying scaled Tikhonov regularization, where $D_{kk} = \sqrt{(AA^\top)_{kk}}$.*

Proof 1 *In the case of infinite new examples, the covariance matrix of the new training examples created by the knock-out procedure concentrates around the covariance matrix obtained when all possible knockout examples comprise the training data set. Since this covariance matrix is bounded away from zero, and since there is a finite number of parameters, the same approximation can be assumed for the inverse of the covariance matrix.*

The vector $\hat{h}$, which is fitted by means of a scaled Tikhonov regularization technique, with a parameter λ is given by:

$$\begin{aligned} \hat{h} &= (\hat{A}\hat{A}^\top + \lambda I)^{-1}\hat{A}y \\ &= (D^{-1}AA^\top D^{-1} + \lambda D^{-1}D^2D^{-1})^{-1}D^{-1}Ay \\ &= D(AA^\top + \lambda D^2)^{-1}DD^{-1}Ay \\ &= D(AA^\top + \lambda D^2)^{-1}Ay. \\ \textit{Therefore, } h &= D^{-1}\hat{h} = (AA^\top + \lambda D^2)^{-1}Ay. \end{aligned}$$

Now consider $\widetilde{A}$, the matrix whose columns contain all the possible knockout ex-

amples, together with the original data points. Let $\widetilde{y}$ be the corresponding labels. By applying a least square linear fit to these inputs, we find: $\widetilde{h} = (\widetilde{A}\widetilde{A}^\top)^{-1}\widetilde{A}\widetilde{y}$. There are nm^2 total examples created. Since all features are assumed to have a mean of zero, all the knockout values of each feature cancel out. What remains is $m(n-1)$ exact copies of each variable and $\widetilde{A}\widetilde{y} = m(n-1)Ay$.

Consider the elements of the matrix $\widetilde{A}\widetilde{A}^\top$. The off-diagonal elements represent the dot product between two different variables. Each variable holds either its original value, or a different value, but it may never happen that both contain the knockout values at once. It happens $m(n-2)$ times for each input example that both features hold the original data. The rest of the cases average out to zero, because while one operand of the dot product is fixed, the other operand traverses the whole zero-mean set of feature values. For the diagonal case, because of symmetry, each value appears nm times, making the diagonal nm times the diagonal of the original matrix.

Putting it all together:

$$\begin{aligned}
\widetilde{h} &= (\widetilde{A}\widetilde{A}^\top)^{-1}\widetilde{A}\widetilde{y} \\
&= (m(n-2)AA^\top + 2mD^2)^{-1}m(n-1)Ay \\
&= \frac{n-1}{n-2}(AA^\top + \lambda D^2)Ay, \textit{ where } \lambda = \frac{2}{n-2}.
\end{aligned}$$

The leading fraction does not change the sign of the results, and is close to one. Furthermore, it can be eliminated by scaling the input examples. By ignoring this fraction, the result of a scaled Tikhonov regularization is obtained. The parameter λ can be controlled in this asymptotic case by scaling the new value of the changed feature in step 4 of the knockout procedure (Figure 2-1).

To get a better understanding of the way Feature Knockout works, we study the behavior of scaled Tikhonov regularization. As mentioned in Section 2.3, in the boosting case, the knockout 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 $\check{y}$ on the training data take the form: $\check{y} = Py$.

For example, the unregularized pseudo-inverse case has $\check{y} = A^\top h = A^\top(AA^\top)^{-1}Ay$, and therefore $P = A^\top(AA^\top)^{-1}A$. There is a simple measure of complexity called *the effective degrees of freedom* [29], 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 ($Tr(P) = n$).

Compare this result to Bishop's [6], where he used a Taylor expansion to show that noise injection is equivalent to Tikhonov regularization.

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 2 *This claim is very standard for Tikhonov regularization. Here, a slightly more elaborate proof is presented. Using the same rules, other claims can be proven. For example, the condition number of the matrix inverted using scaled Tikhonov regularization is lower than the one achieved without regularization. A lower condition number is known to lead to better generalization.*

For scaled Tikhonov regularization, we have $\check{y} = Py$, *where*

$$\begin{aligned} Tr(P) &= Tr(A^\top(AA^\top + \lambda D^2)^{-1}A) \\ &= Tr((DD^{-1}AA^\top D^{-1}D + \lambda D^2)^{-1}AA^\top) \\ &= Tr((D(D^{-1}AA^\top D^{-1} + \lambda I)D)^{-1}AA^\top) \\ &= Tr(D^{-1}(D^{-1}AA^\top D^{-1} + \lambda I)^{-1}D^{-1}AA^\top) \\ &= Tr((D^{-1}AA^\top D^{-1} + \lambda I)^{-1}D^{-1}AA^\top D^{-1}) \\ &= Tr((EE^\top + \lambda I)^{-1}EE^\top), \text{ where } E = D^{-1}A. \end{aligned}$$

Let $USV^\top = E$ *be the Singular Value Decomposition of* E, *where* S *is a diagonal matrix, and* U *and* V *are orthonormal matrices. The above trace is exactly* $Tr((US^2U^\top + \lambda I)^{-1}US^2U^\top$. *Let* S^* *be the diagonal matrix with elements* $S^*_{kk} = S^2_{kk} + \lambda$, *then* $(US^2U^\top + \lambda I)^{-1} = (US^*U^\top)^{-1} = US^{*-1}U^\top$. *The above trace* *becomes* $Tr(US^{*-1}U^\top US^2U^\top) = Tr(S^{*-1}S^2U^\top U) = Tr(S^{*-1}S^2) = \sum_k \frac{S_{kk}^2}{S_{kk}^2+\lambda} <$ $rank(E) = rank(A)$*. Compare this value with the effective degrees of freedom of the unregularized least square solution:* $Tr(A^\top(AA^\top)^{-1}A) = Tr((AA^\top)^{-1}AA^\top) = rank(A)$*. The last equality also holds in the case where* A *is not full rank, in which case* $(AA^\top)^{-1}$ *is only defined on the range of* $AA^\top$*.*

2.5.2 Feature Knockout and Boosting Over Regression Stumps

Similar to the work done on noise injection, we examine the effect of feature knockout on a simple regression technique. As shown above, feature knockout resembles the effect of scaled Tikhonov regularization, i.e., high norm features are penalized by the knockout procedure. However, boosting over regression stumps seems to be scale invariant. Multiplying all the values of a feature by some constant does not change the resulting classifier, since the stump thresholds are chosen independently across features. However, a closer look reveals the connection between scaling and the effect of the knockout procedure on boosting.

Boosting over stumps (e.g., [64, 5]) chooses at each round one out of n features, and a threshold for this feature. The thresholds are chosen 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 [54]), which predicts -1 or +1 according to the threshold. Equivalently, one 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.

Consider first the case where the weights over the samples are uniform, i.e., the error due to a misclassification is the same for all examples. Using the intuitions of the linear least squares case, it is desirable 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 [15]). These entropies depend only on the ratio of positive values in each nm-feature–call this ratio p.

Creating new examples using the generic Feature Knockout procedure (as defined in Figure 2-1) does not change the number of possible thresholds. Hence, 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 left unmodified by the knockout procedure. The value for a knocked-out feature (feature k), 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 does occur, 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 p and the probability $2p(1-p)$ behave similarly: both rise monotonically for $0 \leq p \leq \frac{1}{2}$ and then drop symmetrically.

We assume that the error for input example i is weighted by an arbitrary weight w_i and the following result is obtained:

Lemma 3 *Let s be an nm-feature created by combining an input feature k with a threshold t. Let w_i be the weight of the classification error of example $i, 1 \leq i \leq m$. The amount of inhibition s undergoes, as the result of applying feature knockout to create λ new examples, in which k is replaced, is $\lambda p(1-p)\left(\bar{w}^+ + \bar{w}^-\right)$, where $\bar{w}^\pm$ denotes the mean of the weights for which $x_i(k)$ is above or below the threshold.*

Proof 3 *Using the notation of Figure 2-1, here is the complete proof:*

$$
\begin{array}{llcl}
\textit{Let} & k & = & \textit{a given regression stump feature,} \\
& t & = & \textit{a given regression stump threshold,} \\
& S^+ & = & \{i|x_i(k) \geq t\} \\
& S^- & = & \{i|x_i(k) < t\}, \\
& m^\pm & = & \textit{the cardinality (i.e. number of elements) of } S^\pm, \\
& m & = & m^+ + m^- = \textit{the total number of examples } x_i, \\
& p & = & m^+/m, \\
& 1-p & = & m^-/m.
\end{array}
$$

The nm-feature with the lowest weighted expected error is selected by the weak learner. In the absence of Feature Knockout, this expected error is simply the sum of the weights of examples misclassified by feature k and threshold t. Applying feature knockout to the input examples has the same effect of adding an additional term to the expected error representing the amount of inhibition the nm-feature undergoes:

$$\mathsf{E}(error|k,t) = \mathsf{E}(regression\ stump\ error|k,t) + \mathsf{E}(additional\ KO\ error|k,t).$$

The last term can be decomposed as follows: $\mathsf{E}(additional\ KO\ error|k,t) =$

$$\begin{aligned}
& \sum_{i=1}^{m} w_i \cdot P(a=i, x_a(k) < t < x_b(k)\ \ or\ x_a(k) > t > x_b(k)) && (2.1)\\
= \ & P(a=i)\sum_{i=1}^{m} w_i \cdot P(x_a(k) < t < x_b(k) \quad or \quad x_a(k) > t > x_b(k)) && (2.2)\\
= \ & \frac{1}{m}\sum_{i\in S^+} w_i \cdot P(x_b(k) < t) + \frac{1}{m}\sum_{i\in S^-} w_i \cdot P(x_b(k) \geq t) && (2.3)\\
= \ & \frac{1}{m}\sum_{i\in S^+} w_i \cdot (1-p) + \frac{1}{m}\sum_{i\in S^-} w_i \cdot p && (2.4)\\
= \ & \frac{1}{m}\frac{m^+}{m^+}\sum_{i\in S^+} w_i \cdot (1-p) + \frac{1}{m}\frac{m^-}{m^-}\sum_{i\in S^-} w_i \cdot p && (2.5)\\
= \ & p\frac{1}{m^+}\sum_{i\in S^+} w_i \cdot (1-p) + (1-p)\frac{1}{m^-}\sum_{i\in S^-} w_i \cdot p && (2.6)\\
= \ & p(1-p)\left(\frac{1}{m^+}\sum_{i\in S^+} w_i + \frac{1}{m^-}\sum_{i\in S^-} w_i\right) && (2.7)\\
= \ & p(1-p)\left(\bar{w}^+ + \bar{w}^-\right). && (2.8)
\end{aligned}$$

The knockout procedure causes a classification error when it selects an example on one side of the threshold and replaces a feature value with one taken from an example on the other side of the threshold. The joint probability of this occurrence, scaled by the weights w_i and summed over all examples (as shown in Eq. 2.1), constitutes the additional expected error due to the knockout effect. Since the knockout procedure selects examples randomly with repetition (i.e. it is possible that $x_a = x_b$ and an exact copy is produced), the joint probability is independent and can be factored into the *product of the uniform probability that any given example is chosen for knockout and the probability that the second randomly-chosen example x_b is on the other side of the threshold t, as shown in Eq. 2.2.*

Separating the summation over all i into two distinct summations over examples that lie on each side of the threshold allows these probabilities to be expressed in terms of p, as shown in Eqs. 2.3 and 2.4. The factor $\frac{1}{m}$ can also be expressed in terms of p, as shown in Eqs. 2.5 and 2.6. Factoring out the quantity $p(1-p)$ in Eq. 2.6 reveals the sum of two means, the mean of the weights of the examples that fall above the threshold and the mean of the weights of those that fall below it, yielding the final expression for the additional expected error corresponding to the effect of Feature Knockout.

Creating an additional knockout example increases the probability of a resulting error by a factor of two. Likewise, the additional expected error increases proportionally to the number of knockout examples created. Therefore, the effect of generating λ knockout examples, is an additional error term with an expectation of $\lambda p(1-p)\left(\bar{w}^{+}+\bar{w}^{-}\right)$.

Hence, similar to the scaling in the linear case, the knockout procedure inhibits high magnitude features (here the magnitude is measured by the entropy). Note that in the algorithm presented in Section 2.4, a feature is used for knockout only after it was selected to be a part of the output classifier. Still, knockout inhibits more weak classifiers based on these high-entropy features, making them less likely to be selected again. It is possible to perform this inhibition directly on all features, therefore mimicking the full knockout procedure. An implementation of this is described and evaluated in Section 3.2.1. As it turns out, this direct inhibition, (which is based on the expected inhibition) does not perform nearly as well as the stochastic knockout procedure.

The following section analyzes how the Feature Knockout procedure affects the variance of the learned classifier. This analysis provides further insights on the way the procedure works.

2.5.3 Bias/Variance Decomposition

Bias/variance decomposition is a technique used to analyze the expected performance of a certain class of learning algorithms. The bias is a measure of how closely the average model matches the target distribution. The variance reveals the amount that the estimates vary for different training sets. In selecting a model, there is usually a tradeoff between bias and variance.

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 knockout 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.

In the case of the square loss function $L(f(x), y) = (y - f(x))^2$, the cost function can be decomposed (similarly to [27]) into bias and variance, respectively:

$$\sum_{i=1}^{n} \mathsf{E}_{\hat{x}\sim C_X(x_i)} L(f(\hat{x}), y_i) = \sum_{i=1}^{n} L(\mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}), y_i) + \sum_{i=1}^{n} \mathsf{E}_{\hat{w}\sim C_X(x_i)} L(\hat{w}, \mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x})).$$

Consider the *related* optimization problem which minimizes $\sum_{i=1}^{n} L(\mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}), y_i)$ subject to $\sum_{i=1}^{n} \mathsf{E}_{\hat{w}\sim C_X(x_i)} L(\hat{w}, \mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}))$ being bounded [2].

The bound on the term $\sum_{i=1}^{n} \mathsf{E}_{\hat{w}\sim C_X(x_i)} L(\hat{w}, \mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}))$ means that the learning algorithm has a *bounded differences property* [41] with regards to selection of features. i.e., by removing one of the features, the value of the learned function f will not change by more than a bounded amount. Consider a situation (which exists in our object recognition experiments) where our features are pulled independently

[2]The relation between the problems is that since one solves the second problem (the one with the bound) by applying Lagrange multipliers, every solution of the second minimization problem is also a solution to a minimization problem of the form $\sum_{i=1}^{n} \mathsf{E}_{\hat{x}\sim C_X(x_i)} L(f(\hat{x}), y_i) = \sum_{i=1}^{n} L(\mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}), y_i) + \lambda \sum_{i=1}^{n} \mathsf{E}_{\hat{w}\sim C_X(x_i)} L(\hat{w}, \mathsf{E}_{\hat{x}\sim C_X(x_i)} f(\hat{x}))$ for some λ.

from a pool of many possible features. The bounded difference property guarantees that with high probability the observed testing error is close to the expectation of the testing error with regards to selecting another set of random features of the same size.

An interesting future direction would be to consider situations where the features extracted are a random subset of all possible features. This is a common scenario in recent computer vision algorithms, where the features are often elaborate templates extracted from random locations in a set of training or natural images. Learning using random projections of the data is another such example. A desirable property of the learning algorithm would be stability with regard to this random choice of features in addition to the (assumed) random selection of the training set.

Let us now turn our attention to another "bias-variance" decomposition. We consider the one based on the 0/1 loss function, as analyzed in [16]. We follow the terminology of [16, 62] 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 data set 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 Knockout, 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 Knockout 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[3]. In the experiment presented in Tab. A we measure the net variance, and show that it is actually reduced by applying Feature Knockout. An interesting application that is not explored in this chapter is the exploitation of net variance to derive confidence bars *at a point* (i.e., a measure of certainty in our prediction for a specific input example). Since Feature Knockout emphasizes these differences, it might yield narrower confidence bars.

2.6 Summary and Conclusions

Boosting algorithms continue to gain popularity since they select only the important features of a data set. Unfortunately, they perform poorly on small training sets due to overfitting. SVM performs well regardless of the size of the data set, but uses all features regardless of their relevance. We introduced Feature Knockout, a simple generic regularization technique, and discovered that it is related to deterministic scaled regularization techniques. When applied to GentleBoost, feature knockout promotes redundancy in the output classifier and prevents overfitting on small data sets, yielding performance comparable to SVM for data sets of any size. Hence, it provides the best of both worlds: feature selection and robust learning from few examples.

Feature Knockout is well-suited for machine vision applications because it is not unusual for image data to be partially obscured (i.e. containing incorrect features) and objects in the same conceptual classification may exhibit differences in only a few features (a car is still a car even if a tire is missing, for example). Boosting selects these important features, while Feature Knockout prevents it from relying too heavily

[3]We omit the formal discussion on the relation between variance on training examples and variance on testing examples.

on any particular feature.

Feature knockout, by itself, is similar in spirit to work done a decade ago on noise injection. Back then it was used in combination with bagging, to allow simple classifiers to gain better generalization properties. Here, a different kind of noise is used to prevent modern classifiers from eliciting an oversimplified classification rule.

Chapter 3

Experiments

This chapter presents the results of experiments using the GentleBoost-KO algorithm of Section 2.4 on several different types of data sets, demonstrating that it performs as well or better than the other proposed deterministic boosting variants. In contrast to most boosting techniques, which overlook the high variance each base classifier's performance has on small data sets, our method encourages redundancy in the constructed classifier. This is done not in a deterministic way, by estimating a bound on the expected performance, but by creating new random examples to be used for training.

3.1 General-Purpose Data: UCI Repository

The feature knockout method was evaluated on 10 UCI Repository[7] data sets that were made suitable for binary classification, by either thresholding the value of the target function (e.g. the price in the housing data set) at its median, or by picking a label to be the positive class. These 10 data sets were: arrhythmia, dermatology, e.coli, glass, heart, housing, letters, segmentation, wine, and yeast.

Each data set was split randomly into 10% training, 90% testing, and each of the following classifiers were run: original, KO, NR, and VC variants of AdaBoost and GentleBoost, Linear and Gaussian SVM, as well as SVM-RFE which was allowed to select half of the features. Linear SVM was also run on a data set that contained

100 examples generated in accordance with the knockout procedure of Figure 2-1. In addition, results are reported for GentleBoost combined with noise injection (the algorithm that adds gaussian noise to the examples) with the best noise variance found. For each training example, ten noisy virtual examples were created. By selecting parameters according to the performance on the testing data, the noise injection results were biased, and should only be taken as an upper bound for the performance of noise injection.

Table A shows the mean error, the standard deviation of the error, and the number of features used by the classifiers (SVM always uses the maximal number of features). The variance over a distribution of knockout examples for correct classifications (unbiased variance), and incorrect classifications (biased variance) (see Section 2.5.3 below) was also measured in the following manner: for each *testing* example, 50 knockout examples were generated in accordace with Figure 2-1, and the classification variance was computed over these 50 examples. The variance was averaged over all biased and unbiased testing examples separately. A good classifier produces more variance for incorrectly classified examples, and only a little variance for correctly classified ones. The net variance, i.e. the difference between the unbiased and biased variance, is expected to be higher for better classifiers.

It is apparent from the results that:

1. In general the knockout procedure helps GentleBoost, raising it to the same level of performance as Linear SVM.

2. Knockout seems to help AdaBoost as well, but not always, and sometimes helps SVM.

3. Knockout seems to help increase the net variance (which is good, see Section 2.5.3).

4. As expected, knockout produces classifiers that tend to use more features.

5. Knockout shows different, sometimes better, performance than noise injection (GentleBoost-NI).

6. The feature selection of SVM-RFE seems to hurt more than it helps.

7. The VC variants seem to perform poorly on these datasets.

8. Knockout beats the NoRepeat (NR) variants most of the time.

3.2 Experiments to Understand Feature Knockout

Feature knockout makes it more likely for training errors to occur, and when they do, the effects are profound. In order to better understand these effects, it is useful to investigate how a knockout example affects the training process by simulating its behavior and examining the results. In the following two experiments, we first simulate the outcome of feature knockout using a technique we call *direct inhibition*, and then examine the influence of individual knockout examples on the ensemble classifier.

3.2.1 Direct Inhibition

As demonstrated in the proof of Lemma 3, the effect of feature knockout on the expected error of a regression stump can be simulated, without generating any new examples, by adding an additional error term to the expected error. A modified version of the standard GentleBoost algorithm was created to take into account this additional expected error in the selection of the best regression stump for each boosting round. This direct inhibition therefore simulates the effect of feature knockout on GentleBoost without generating any new examples. Keep in mind that direct inhibition does not recreate the full effect of feature knockout, since it only deterministically estimates the expected error of the randomized knockout process.

There are two different methods of applying direct inhibition to the selection of the best regression stump. The inhibition error can either be added to select features only, or applied uniformly over all features. Select inhibition is used to simulate the effect of GentleBoost-KO, in which knockout is performed only on the features that actually participate in the classification. Alternatively, uniform inhibition simulates the effect

Table 3.1: Comparison of the performance effects of both uniform and direct inhibition on the UCI data sets. Experiments included ten independent runs using random splits of 10% training and 90% testing examples.

Data Set	GentleBoost	GentleBoost-KO	Select Direct Inhibit.	Uniform Direct Inhibit.
Arrhythmia	37.6±2.7	35.6±1.8	37.9±2.6	39.7±6.0
Dermatology	4.6±3.5	1.0±1.2	3.7±3.0	3.8±3.0
E.coli	11.0±6.6	10.2±5.0	10.4±6.5	12.3±6.0
Glass	35.5±8.8	30.3±6.6	35.0±8.0	37.6±9.4
Heart	28.3±9.4	23.6±3.0	28.0±9.7	28.6±9.2
Housing	18.3±2.7	17.0±1.5	20.7±7.7	21.3±7.8
Letters	3.9±0.6	3.3±0.2	4.3±0.6	4.2±0.6
Segmentation	7.4±4.4	6.8±3.2	7.2±4.5	7.5±4.4
Wine	14.3±4.0	7.7±2.4	14.8±4.5	14.2±3.9
Yeast	33.0±2.6	32.3±1.5	35.2±4.9	35.0±5.0

of applying the knockout procedure to all features, a method that would otherwise require an excessive amount of memory if the knockout procedure of Figure 2-1 were used.

Refer to Table 3.1 for a comparison of the performance of direct inhibition and feature knockout. The results show the mean error for ten independent runs of the modified versions of GentleBoost on each UCI data set, using a random split of 10% training and 90% testing examples. While direct inhibition sometimes performs better than the original GentleBoost, it does not reach the performance level of GentleBoost-KO.

One might suppose that direct inhibition performs worse than feature knockout because it inhibits the initial selection of a feature to classify upon, in addition to inhibiting the continued reliance upon the same feature. In other words, features that would otherwise be selected first by GentleBoost would be inhibited by direct inhibition even before they are selected. While this may be true for uniform inhibition, it should not affect the select inhibition, which is applied only to features that have already been selected. The fact that neither method performed as good as feature knockout indicates that the effect of feature knockout is more complex than something that can be captured in a simple error expectation.

Unless the knockout procedure shifts a feature value from one side of a regression stump threshold to the other, it has no effect on the training process. While many knockout examples do not exercise any affect on the selection of regression stumps, a handful of them or even a single one will inhibit the reliance on a small number of features in successive selections of the weak classifiers, thus leading to better generalization. It seems that the stochastic nature of feature knockout allows it to produce these rare influential examples, while direct inhibition fails to do so. Perhaps direct inhibition underestimates the importance of a single knockout example. To further investigate this supposition, the next section examines the effect of individual knockout examples on the performance of GentleBoost.

3.2.2 One Example KO

To understand if it is the rare events that contribute to the generalization properties of GentleBoost-KO, we study the effect of a single knockout example on the trained classifier. Several experiments were run in which 100 knockout examples created during a typical run of GentleBoost-KO were then appended individually to the training data of independent runs of the standard GentleBoost algorithm. To clarify: for each data set and split, we ran GentleBoost-KO for 100 rounds, resulting in the creation of 100 knockout examples. We then ran GentleBoost 100 additional times, each time on the original training set plus one of the artificial knockout examples.

Since the examples were created by GentleBoost-KO, we can be sure that the knocked-out features are ones that are important to classification, and thus should make a difference in the ensemble classifier. By independently appending each of these examples to the original training set and successively running GentleBoost for each example, we can determine the portion of knockout examples that actually helped improve performance in the output classifier, one at a time. Histograms of the performance change due to 100 individual knockout examples (as well as the original performance level of GentleBoost and GentleBoost-KO for comparison) are shown in Figure B-2. The solid line shows the performance of the traditional GentleBoost algorithm, while the dashed line indicates that of GentleBoost-KO. Bars indicate the number of KO examples that caused GentleBoost to attain specific levels of performance.

In most cases, some of the individual knockout examples improved GentleBoost and some worsened its performance. The histograms for E.coli and Glass demonstrate cases where a mere handful of knockout examples can lead to significant performance gains. Segmentation was the only data set for which all knockout examples worsened the performance of GentleBoost, signifying that, while knockout usually leads to a better classifier, it does not work in all cases.

The overall conclusion from this section's experiments is that the Feature Knockout procedure cannot be understood by considering only the expected inhibition each weak classifier undergoes. Rather, it is individual knockout examples that make GentleBoost-KO preferable over GentleBoost in many cases. This result cannot be used in practice though, since it is impossible to identify the influential KO examples *a priori*.

3.3 Vision Experiments

3.3.1 Visual Recognition Using Caltech Data Sets

We also tested the boosting variants and SVM with feature selection on several Caltech object recognition data sets (presented in [21]). In each experiment the classifier had to perform binary classification, i.e. distinguish between images containing an object and background images that lack the object. The data sets: Airplanes, Cars, Faces, Leaves and Motorbikes, as well as the background images are available at `http://www.vision.caltech.edu/html-files/archive.html`. Example images are shown in Figure 3-1. Any colored images were converted to grayscale before use. Note that this task is challenging since the images contain clutter, i.e., the objects appear in front of arbitrary backgrounds. For some classes, the images are not normalized in terms of their location in the image, suggesting some advantage to the use of translation invariant features.

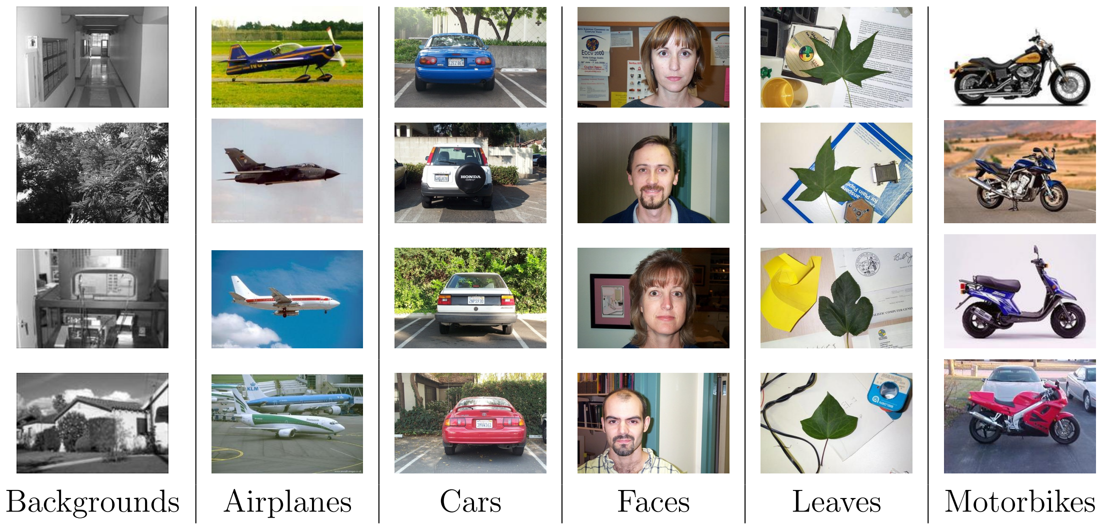


Figure 3-1: Sample Images from the Caltech Object Recognition Database

In the experiments, the predefined splits of the positive class were used[1]. From the positive training examples, a random subset was sampled to serve as the positive training set. This way, although each experiment trains on a different number of positive examples, the test sets are fixed and allow direct comparison.

Since discriminative methods were used, a negative training set was necessary. Toward this end, examples were removed randomly from the negative set (the background images) and used for training. Twice as many negative examples were used as positive ones. Note that extra negative training examples are used, because for visual recognition, negative training images are easy to collect. For each data set and training size, the experiment was repeated 10 times, each time drawing a new random subset of positive training examples, and re-splitting the negative training and testing set.

Where parametric classifiers were employed, results are shown for the best parameters found. Since cross-validation is problematic for small training sets (especially with less than six positive examples), the hyperparameters were hand-selected to minimize the testing error (i.e., the parametric classifiers were given an unfair advantage and their performance should likewise be considered an upper-bound). We considered

[1]No predefined split was available for the Leaves dataset, so it was split randomly into 50% training and 50% testing examples.

Gaussian SVM kernels with widths $\sigma \in \{0.01, 0.1, 1, 10, 100\}$. The parameter d was tested with values 1,2, and 3 in the VC boosting variants. In GentleBoost-NI (GentleBoost with noise injection), ten noisy examples were generated for each training example, and noise variances $\sigma \in \{1\%, 5\%, 10\%, 20\%, 30\%\}$ were considered. Similarly to the 100-round boosting experiments, 100 knockout examples were generated for Linear SVM-KO.

To convert each image into feature-vectors, 1000 C2 features [56] were used. 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 those reported in [20]. The results are shown in Figure B-1. To facilitate comparison with previous work, the error at the equilibrium-point between false and true positives was used as the error-measure. It is clear that for a few dozen examples, all algorithms attain the same performance level. However, for only a few training examples, GentleBoost does not perform as well as SVM, while GentleBoost-KO performs significantly better. The true victors in this experiment were AdaBoost-KO and AdaBoost-NR, which consistently challenged Linear SVM, even for very few examples.

For clarity, not all classifiers were included in Figure B-1. Table A lists the complete equilibrium error results for all classifiers and training set sizes tested. The various SVMs failed to demonstrate a marked difference in performance from the simple linear kernel and were thus omitted. A linear kernel also maintains a low VC-dimension, which helps to avoid overfitting, and is therefore important when dealing with a small number of training examples.

Note that we ran the boosting procedures for 100 rounds, which means they used fewer than 100 features. Compared to the 1000 features used by the SVM classifier, the boosting classifiers offer a considerable savings in run time.

Experiments Using SIFT Features

We also tried to apply Lowe's SIFT features [40] to the same data sets, although these features were designed for a different task. For each image, we used Lowe's binaries to compute the SIFT description of each key point. We then sampled from the training

Error Measure	Algorithm	Air-planes	Cars	Faces	Leaves	Motor-bikes
Mean Error	Linear SVM	0.104	0.019	0.107	0.118	0.033
	GentleBoost	0.118	0.036	0.168	0.137	0.026
	GentleBoost-KO	0.100	0.033	0.119	0.114	0.023
Error at Equilibrium	Linear SVM	0.108	0.018	0.111	0.126	0.007
	GentleBoost	0.120	0.037	0.166	0.132	0.003
	GentleBoost-KO	0.111	0.030	0.136	0.120	0.008

Table 3.2: Mean testing error, and error at equilibrium for Lowe's SIFT features, applied to the Caltech data sets. In this table the features were the minimum distance from the keypoints in the image to the keypoints collected during training.

set 1000 random keypoints $k_1, \ldots, k_{1000}$. 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: $[v^I(1) \ldots v^I(1000)]$, such that $v^I(j) = min_i ||k_j - k_i^I||$. Note that in [40] 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 [21] we produced the results in Table 3.2. Although we tried several kernels for SVM, they did not outperform than the linear kernel. GentleBoost seems only slightly worse, and GentleBoost-KO succeeds in outperforming it. Again, note that the boosting classifiers use less than one tenth of the features. Unfortunately, all algorithms performed poorly on these features when few training examples were used. These figures are omitted since a difference of several percentage points is insignificant when the algorithms only achieve 70% accuracy.

Since SIFT features were not created for the purpose of classifying images of generic objects, we tried another way to apply them to the Caltech data sets. In the following experiment we implemented the method of [14], which is based on a text analysis approach. The SIFT descriptors (128 dimensional vectors) were clustered into 1000 "vocabulary terms" using k-means. The SIFT keypoints in a new image are distributed between the clusters using the least Euclidean distance. The frequencies

Error Measure	Algorithm	Air-planes	Cars	Faces	Leaves	Motor-bikes
Mean Error	Linear SVM	0.111	0.019	0.135	0.173	0.075
	GentleBoost	0.436	0.173	0.250	0.378	0.326
	GentleBoost-KO	0.386	0.063	0.173	0.237	0.165
Error at Equilibrium	Linear SVM	0.107	0.018	0.132	0.172	0.076
	GentleBoost	0.357	0.169	0.249	0.398	0.335
	GentleBoost-KO	0.331	0.061	0.199	0.260	0.173

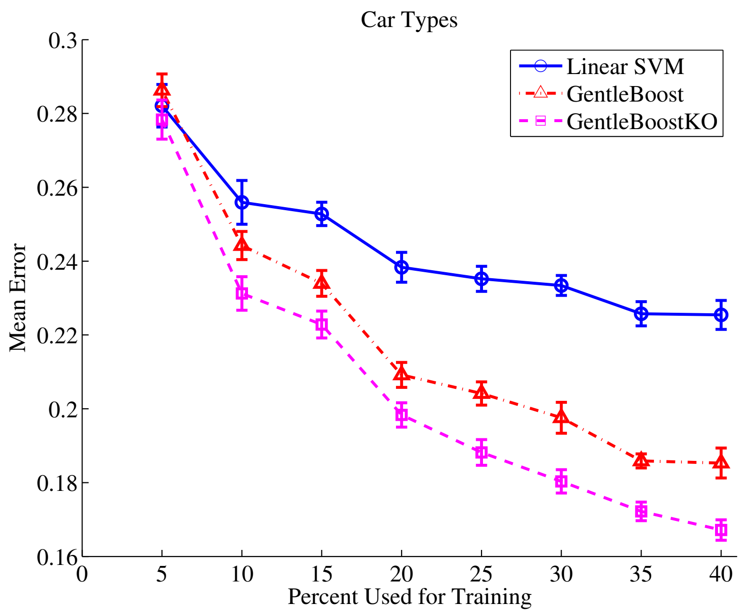
Table 3.3: Mean testing error, and error at equilibrium for Lowe's SIFT features with vocabulary terms, applied to the Caltech data sets. Here a vocabulary of 1000-keypoint clusters was created using the k-means clustering algorithm. The frequencies of vocabulary terms in the keypoints of the new image were used as features.

of the terms, i.e., the number of keypoints detected in the new image which belong to each cluster constitutes the vector passed to the classifier. In a similar manner to [14] we used linear SVM in our experiments, and compared the performance with GentleBoost and GentleBoost-KO. The results are reported in Table 3.3. They seem comparable to the results above using the SIFT features directly.

3.3.2 Car Type Identification

The Car Type data set 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 from a video stream generated by a Mobileye car detector mounted on the windshield 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 in Figure 3-2. 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, 30 repetitions each). The rest of the examples were used for testing. It is evident that for this specific data set GentleBoost outperforms SVM. Still, GentleBoost-KO does even better.

Presented in Tab. 3.3.2 are further results obtained using nonlinear SVMs, and



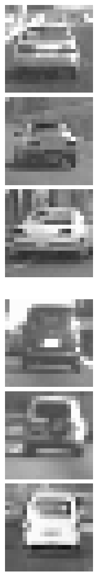


Figure 3-2: The results for the car types data set, together with example images. Points indicate the mean and standard error of 30 independent experiments versus percentile of training images. Note that for this image the vertical axis does not start at zero.

Table 3.4: Results omitted from the Car Types data set plot (Figure 3-2), including the best performing polynomial and Gaussian SVM, as well as the Random Forest method.

Classifier	Train/Test Split		
	10/90	25/75	40/60
Best Gaussian SVM	29.1	28.2	27.9
Best Polynomial SVM	24.4	9.6	17.9
Random Forest	27.0	21.9	18.8

using the Random Forest (RF) Algorithm [11]. We compare with RF since it is a recent development in ensemble classifiers, with a lot of experimental support. Note, however, that RF was not designed to perform feature selection on small data sets[2]. Results are also shown for the best Gaussian and polynomial SVM kernels, but again none outperformed linear SVM.

[2]RF may not be appropriate for small data sets, because of its out-of-bag method and reliance on deep CART trees with many branches. Nor does it select relevant features *per se*.

3.4 Bioinformatics Experiments

3.4.1 Data Sets of Long and Vega

In order to compare to the results of Long and Vega [39], we recreated their experiments using the five published datasets: ALL-AML, ER, Colon, LN, and Brain, all available at `http://www.cs.columbia.edu/~plong/boost_microarray/`. For each experiment, the biological datasets were randomly split using two-thirds of the examples for training and the remaining third for testing. The experiments were repeated 100 times independently and the mean testing errors were computed for two cases: one in which all classifiers were limited to using 10 features and another in which they were limited to 100 features. The resulting mean error rates are shown in Table 3.5. The number following VC indicates the setting of the parameter d. The GentleBoost-KO-VC rows use GentleBoost with both KO and VC modifications. The original results of Long and Vega are reproduced for comparison in Table 3.6.

It is evident from these results that:

1. GentleBoost has a slight advantage over AdaBoost on these data sets.
2. GentleBoost-KO usually outperformed GentleBoost and GentleBoost-NR.
3. GentleBoost-VC does not beat regular GentleBoost, but the combined GentleBoost-KO-VC does, and
4. GentleBoost-KO-VC is consistently among the best performing algorithms for each dataset, but at the cost of an additional parameter.
5. GentleBoost-NR performed about the same as AdaBoost-NR.
6. SVM-RFE and l_0-norm SVM performed comparably (worse than Linear SVM, but that is to be expected).
7. Our SVM-RFE implementation performed much better than Long and Vega's, suggesting that there might be a flaw in their results.

Table 3.5: Results for GentleBoost Variants on the Long-Vega Data Sets

Classifier	Gene Limit	ALL-AML	ER	Colon	LN	Brain
GentleBoost	10	7.3	19.7	20.2	41.2	39.7
GentleBoost-KO	10	4.4	18.8	21.9	41.9	38.2
GentleBoost-NR	10	3.1	20.9	23.3	45.9	41.2
GentleBoost-VC1	10	6.5	20.0	23.1	41.6	43.8
GentleBoost-VC2	10	6.0	21.2	21.0	41.2	38.2
GentleBoost-VC3	10	6.0	20.3	21.2	40.3	38.2
GentleBoost-KO-VC1	10	4.0	18.8	19.0	39.4	39.3
GentleBoost-KO-VC2	10	3.5	19.4	19.3	38.4	39.0
GentleBoost-KO-VC3	10	4.4	18.4	19.5	38.1	35.5
Linear SVM	10	3.7	13.1	18.8	41.9	35.8
l_0-norm SVM	10	6.2	20.3	24.5	38.4	39.2
SVM-RFE	10	6.5	25.0	21.0	38.8	40.5
GentleBoost	100	4.8	14.7	22.1	37.2	34.8
GentleBoost-KO	100	2.1	12.5	19.3	36.6	35.8
GentleBoost-NR	100	2.7	12.2	16.9	41.9	35.2
GentleBoost-VC1	100	5.0	12.5	47.9	35.9	49.5
GentleBoost-VC2	100	4.4	14.1	20.7	36.6	38.5
GentleBoost-VC3	100	4.4	13.4	20.7	38.4	39.5
GentleBoost-KO-VC1	100	2.1	13.4	26.9	37.2	46.2
GentleBoost-KO-VC2	100	2.1	12.8	20.2	37.2	38.3
GentleBoost-KO-VC3	100	1.7	13.1	19.5	36.2	36.8
Linear SVM	100	2.5	14.1	17.6	36.6	36.0
l_0-norm SVM	100	3.1	20.0	18.1	35.9	35.2
SVM-RFE	100	2.5	17.5	17.4	38.1	34.8

3.4.2 Microarray Data

Data sets. We evaluated the performance of the knockout method for the problem of gene expression classification on four data sets containing treatment outcome or status studies. The first was a study of the treatment outcome of patients with diffuse large cell lymphoma (DLCL), here referred to as "lymphoma" [57]. The second data set came from a study of the treatment outcome of patients with childhood medulloblastomas [48], here referred to as "brain". For both sets, positive examples indicate a successful outcome. The third was a study of the metastasis status of patients with breast tumors [63], referred to as "breast met", where positive samples

Table 3.6: Original Results of Long and Vega

Classifier	Gene Limit	ALL-AML	ER	Colon	LN	Brain
Adaboost	10	6.2	19.9	25.3	40.4	42.3
Adaboost-VC	10	3.9	18.1	24.4	43.8	41.1
Adaboost-NR	10	3.5	19.5	25.1	42.7	41.2
Adaboost-PL	10	7.0	20.6	23.4	36.5	41.9
Arc-x4-RW	10	6.5	19.8	25.0	39.1	41.4
Arc-x4-RW-NR	10	3.3	17.8	24.7	42.1	40.7
SVM-RFE	10	13.4	20.9	19.2	48.4	39.2
Wilcoxon/SVM	10	6.4	23.2	24.3	35.4	39.3
Adaboost	100	5.2	16.1	23.4	35.4	38.2
Adaboost-VC	100	2.8	13.8	22.6	42.8	38.2
Adaboost-NR	100	2.7	13.2	21.9	40.6	36.5
Adaboost-PL	100	5.0	17.2	23.2	36.2	38.6
Arc-x4-RW	100	5.4	16.6	23.7	36.9	38.0
Arc-x4-RW-NR	100	2.6	12.8	21.6	41.1	36.1
SVM-RFE	100	6.5	12.6	20.7	48.1	35.7
Wilcoxon/SVM	100	3.3	17.5	23.6	40.4	37.8

Table 3.7: Microarray Data Set Details

Data Set	Dimensions	Positive Examples	Negative Examples
Lymphoma	7129	32	26
Brain	7129	39	21
Breast Metastasis	24624	44	34
Lymph Status	12600	47	43

indicate the patients were disease-free for 5 years after the onset and the negative examples indicate metastasis within that time period. The fourth is an unpublished study of breast tumors [50] for which corresponding lymph nodes were either cancerous or not, referred to as “lymph status”. The sizes and dimensionality of these datasets are summarized in Table 3.7.

We tested many algorithms on these datasets, including the original and variants of SVM, AdaBoost, and GentleBoost. Feature selection was added to Linear SVM using the Wilcoxon Mann-Whitney test to choose 100 differentially-expressed

Table 3.8: Mean classification error and standard deviation for 20 independent runs on randomized (80% training, 20% testing) splits of the microarray data sets. Except for Linear SVM and Linear SVM-KO, all classifiers were limited to using 100 features.

Classifier	Lymph Status	Brain	Breast Metastasis	Lymphoma
Linear SVM	37.8±11.7	36.3±13.0	43.3± 11.3	42.5± 16.2
Linear SVM-KO	37.8±11.7	36.3±13.0	43.1± 11.5	42.5± 16.4
Wilcoxon SVM	35.6±12.7	41.7±12.1	48.3± 13.3	47.5± 11.2
l_0-norm SVM	40.9±10.4	31.7±11.7	46.7± 11.3	47.9± 15.7
SVM-RFE	36.6± 9.8	31.7±12.9	43.1± 9.5	43.7± 18.1
GentleBoost	36.2±12.1	36.2±12.5	48.6± 10.3	46.2± 16.6
GentleBoost-KO	35.9± 9.0	34.2±10.4	46.7± 9.9	44.6± 15.6
GentleBoost-NR	33.8±10.2	36.2±11.6	47.8± 12.0	46.2± 17.2
GentleBoost-VC1	43.8±11.5	49.2±14.3	48.9± 8.2	41.7± 15.3
GentleBoost-VC2	40.0±11.0	34.2± 8.5	43.3± 9.6	38.7± 11.6
GentleBoost-VC3	37.5±10.3	32.9± 9.5	46.4± 12.1	41.7± 16.7
GentleBoost-KO-VC1	40.9± 9.6	50.4±13.4	43.6± 13.5	39.6± 14.0
GentleBoost-KO-VC2	40.0± 9.2	37.1± 8.8	45.3± 13.4	39.2± 14.1
GentleBoost-KO-VC3	35.6±10.4	34.2±11.8	45.3± 9.4	42.9± 14.9
AdaBoost	37.2±11.9	36.2± 9.9	47.8± 11.9	49.2± 14.5
AdaBoost-KO	37.2±11.9	36.2± 9.9	45.8± 15.6	42.9± 15.1
AdaBoost-NR	33.8±11.2	35.8± 9.8	45.0± 8.8	42.1± 15.9
AdaBoost-VC1	38.4±11.7	36.3± 8.7	51.9± 8.9	41.2± 16.8
AdaBoost-VC2	39.7±10.8	27.1±12.9	49.4± 8.2	53.3± 17.2
AdaBoost-VC3	55.6±10.3	72.5±13.8	48.6± 8.6	57.9± 13.1

genes. GentleBoost and AdaBoost were run with the previously suggested NR and VC variants (see Section 1.2.3). The number following VC indicates the setting of the parameter d. The GentleBoost-KO-VC rows use GentleBoost with both KO and VC modifications. Since microarray data is very difficult to classify, 80% of the data was used for training, leaving 20% for testing.

Results. For these difficult data sets, there is no clear winner. See Table 3.8 for the mean error measurements. It is apparent from the results that GentleBoost-KO outperforms the standard GentleBoost and Linear SVM. In general, knockout helped boosting but had little effect on Linear SVM. SVM-RFE is easily the best feature selecting SVM on these data sets, with performance very close to Linear SVM. The NoRepeat (NR) and knockout (KO) variants were close in performance; AdaBoost-NR was slightly better than AdaBoost-KO, but GentleBoost-KO slightly beat GentleBoost-NR.

3.4.3 NMR Spectroscopy

Since biological data sets are hard to create, learning from few examples is a real issue with biological data. This section was motivated by the need to create a learning algorithm that can be run and evaluated on the 12 samples available for each experiment in the following NMR data set.

A group of researchers wished to classify mice according to the number of days that passed since they received some dosage of a drug. The experiment lasted several days and urine samples were collected from each mouse twice daily. The drugs were administered just after the first PM sample was taken at day 0. The spectrum of the resulting NMR test contains 198 frequencies, each giving rise to one feature vector.

Our task was to find a good classifier and to locate some bio-markers that can be used to design a simple urine test. For each dose there were only 6 mice, resulting in a total of 12 examples for each experiment. The boosting variants were run for 100 rounds or the selection of 20 distinct features, whichever came first, with results summarized in Table 3.9. For SVM, GentleBoost and GentleBoost-KO we measure the leave-one-out (LOO) classification error. While LOO has a large variance, we were forced to use it due to the low number of examples.

It is no surprise that Linear SVM performed the best on these few examples. SVM-RFE and l_0-norm SVM both managed to perform as well as the standard Linear SVM while using less features. The Wilcoxon SVM performed poorly, however. The results show a clear advantage of the GentleBoost-KO algorithm over GentleBoost, but the best boosted classifier is clearly AdaBoost-KO with a mere six LOO errors. Knockout does not help SVM in this case, but definitely improves GentleBoost-VC, which seems unstable without it.

Table 3.9: LOO errors (out of a possible 12) for three classifiers on the NMR data set. The positive examples were taken on Day I, and the negative on Day II. After one day, a low dose is easy to detect. At day two, the body is almost back to normal; day three is *very* difficult. The effect of a high dose is maximal only after several days.

	Dose	Low			Medium			High		
	Day I	0	0	0	0	0	0	0	0	0
Algorithm	Day II	1	2	3	1	2	3	1	2	3
Linear SVM		0	2	1	0	0	0	0	0	0
Linear SVM-KO		1	5	1	0	0	1	1	0	0
l_0-norm SVM		1	1	1	0	0	0	0	0	0
SVM-RFE		0	2	1	0	0	0	0	0	0
Wilcoxon SVM		0	5	6	0	0	0	1	0	0
AdaBoost		0	7	5	4	8	3	4	2	0
AdaBoost-KO		0	1	2	0	0	3	0	0	0
AdaBoost-NR		0	5	2	0	0	3	0	0	0
AdaBoost-VC1		0	7	5	4	8	3	4	2	0
AdaBoost-VC2		12	4	5	8	4	9	8	10	12
AdaBoost-VC3		12	6	8	8	4	9	8	10	12
GentleBoost		0	7	3	3	7	4	4	1	0
GentleBoost-KO		0	3	4	0	1	0	0	0	0
GentleBoost-NR		0	3	3	0	0	2	0	0	1
GentleBoost-VC1		0	6	3	3	6	4	4	1	0
GentleBoost-VC2		0	5	3	3	6	4	4	1	0
GentleBoost-VC3		0	5	3	3	7	4	4	1	0
GentleBoost-KO-VC1		0	5	4	0	1	0	0	0	0
GentleBoost-KO-VC2		0	4	4	0	1	0	0	0	0
GentleBoost-KO-VC3		0	5	4	0	1	0	0	0	0

3.4.4 High-Throughput Docking

In high-throughput docking (HTD) classification, one tries to predict the activity level of compounds against a protein target of interest in an attempt to identify novel compounds that elicit a desired biological response. A typical data set contains millions of compounds, and the number of features (describing the chemical and geometrical properties of the compounds) is in the thousands.

The initial experiments we present are on a data set recommended to us by [35]. It contains a subset of 2305 compounds that were pre-filtered using commercial software. Of these components, 230 are HIV-1 protease inhibitors, and the rest are inactive. Since the actual verification of the results is time-consuming, it is desirable to reduce

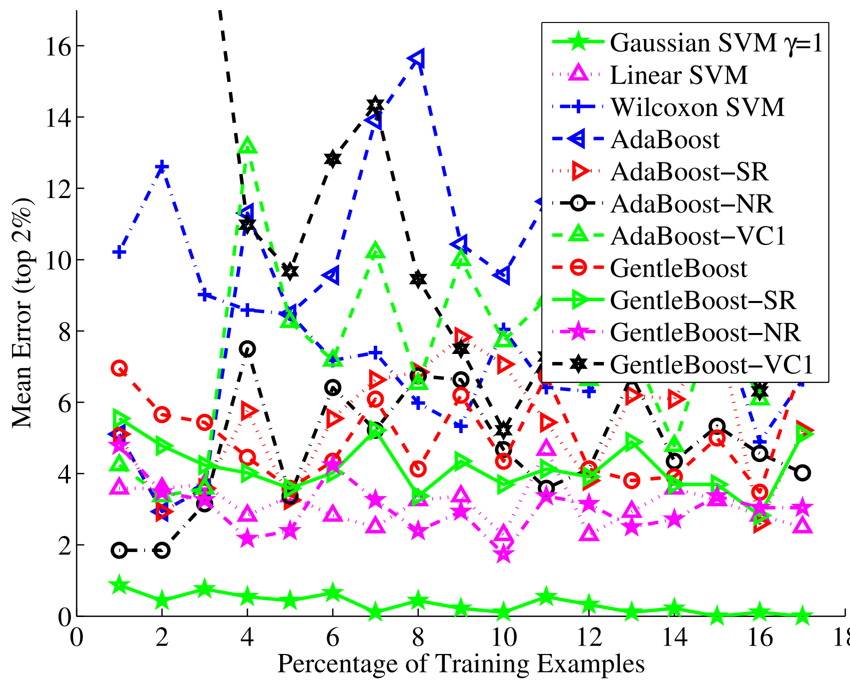


Figure 3-3: Classifier Performance on the HTD Data

the number of components returned even further. Hence, an appropriate error measure is the percentile of true components recovered out of those predicted with the highest scores.

Results. See Figure 3.4.4 for a plot of mean error vs. percentage of examples used for training for each classifier considered. Since Linear SVM and Linear SVM-KO performed indistinguishably, only Linear SVM is shown. The Gaussian SVM used the best kernel width from $\gamma \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$, and the VC lines represent the best performing classifier using parameter $d \in \{1, 2, 3\}$.

It is clear that this type of data is best characterized by the Gaussian SVM. The best-performing boosted classifier was GentleBoost-NR, followed closely by GentleBoost-KO. They both show performance comparable to Linear SVM, even with few examples. AdaBoost does not work nearly as well as GentleBoost on this data, but knockout still helps it significantly. Once again, the VC variants appear to be very unstable and exhibit very poor performance for few examples. The Wilcoxon feature selection for SVM also performed poorly.

3.4.5 Conclusion

It is evident that feature knockout usually helps the algorithms to which it is applied. NR and KO variants seem to perform better than other boosting variants through encouraging redundancy in the resulting classifier. Where traditional boosting algorithms tend to overfit, GentleBoost-KO works well regardless of the amount of training examples. The VC variants of both AdaBoost and GentleBoost did not perform as well on the datasets considered. Even though GentleBoost-KO-VC demonstrated some of the best performance in a few cases, the VC variants in general are suspiciously unstable for few examples, and are recommended for use with only medium or large size datasets as a result.

Our experimental evidence shows that KO is difficult to imitate in a deterministic manner. Noise injection was used to regularize unstable algorithms a decade ago; in its new form (KO) it can be used to prevent the overfitting of state-of-the-art algorithms. It enables these classifiers to be used on Bioinformatics data sets that are noisy and contain very few examples.

Chapter 4

Patch-Based Texture Edges

4.1 Introduction

The detection of image edges has been one of the most explored domains in computer vision. While most of the effort was aimed at the detection of intensity edges, the study of color edges and the study of texture edges are well developed fields as well.

The dominant approach in texture edge analysis is to construct a description of the local neighborhood around each pixel, and then to compare this descriptor to the descriptors of nearby points. This approach is often referred to as “patch-based” since a fragment surrounding each pixel is used to compute its filter output. In this work, however, the term “patch-based” is quite distinguishable: It means that the gray values of the patch are used as is, and that the basic operation on patches is the comparison of two patches using image correlations measures, such as normalized cross correlation between the gray values, or their Euclidean distance.

What makes this approach novel for texture edge detection is that since texture is a stochastic property, this kind of descriptor would be traditionally considered unfit. In other words, since the gray values of two neighboring patches from the same texture could be very different, most researchers search for more elaborate descriptors. This is in contrast to the dominant trend in current texture synthesis research, where patches of the original texture are stitched together in order to generate a new texture image, a trend that seems to be much more successful than the best descriptor based methods.

The main idea of this work is simple to grasp: if a point lies on the left-hand side of a texture edge, the distribution of similarities of the patch centered at this point to the patches on its left is different from the distribution of similarities to the patches on its right. Detection of the texture edges can therefore be achieved by examining these differences in similarity distributions.

As we will show in this chapter, sampling from the distributions of similarities can be done very efficiently. In order to estimate whether the distributions are the same, we use a non-parametric test called the Wilcoxon Mann-Whitney Test [66]. It is similar to the t-test but performs well even for small sample sizes with unknown distributions. This test was used previously for SVM feature selection in Section 3.4.2.

In contrast to intensity edges, which have many uses in computer vision, texture edges have been used primarily for image segmentation. In order to make this work complete, we couple it together with a segmentation scheme. Since texture edges are often gapped, we use a hybrid deformable model to capture the image contour. This type of deformable model borrows the best features from traditional parametric deformable models [34, 59] and geometric level-set based deformable models [13, 43], and enjoys the advantage of bridging over gaps in contours, topology freedom during evolution, and fast convergence. In particular, the model shape is implicitly represented in a higher dimensional space of distance transforms as a distance map "image," and model deformations are efficiently parameterized using a space warping technique: the Free Form Deformations (FFD) [1, 3] based on cubic B-splines.

4.2 Related Work

Below we discuss traditional texture segmentation approaches, the emerging patch-based techniques, and explain the background of the statistical tests we employ.

4.2.1 Feature-Based Texture Edge Detection and Segmentation

Traditional methods for texture analysis are often grouped into three major categories: statistical, structural and spectral. In the statistical approach, texture statistics (e.g., moments of the gray-value historgram, or co-occurrence matrices) serve as texture descriptors. In structural approaches, the structure is analyzed by constructing a set of rules that generates the texture. In spectral approaches, the texture is analyzed in the frequency domain.

In contrast to the wealth of approaches suggested in the past, the last decade was dominated by the filter bank approach, to which we will suggest an alternative.

> "There is an emerging consensus that for texture analysis, an image should first be convolved with a bank of filters tuned to various orientations and spatial frequencies."[23]

Of the many contributions that employ banks of filters, the most common set of filters used seems to be the Gabor filters [22, 31, 32, 23, 42, 53]. We would like to especially mention the work of [53] which, like our work, emphasizes the detection of texture edges, and not texture segmentation. In relation to our work, we would also like to point out that non-parametric tests have been used in the past for texture segmentation, [31, 32], where nearby blocks of the image were grouped together if the distributions of filter outputs in those blocks were not statistically distinguishable. Similarly to our work, the statistical distinguishability was measured by using non parametric tests: [31] used the the Kolmogorov-Smirnov distance and [32] used the χ^2 statistic. On a more abstract level we find relation to the work of [26] in which characteristics of small segments in the image are used as part of the texture description in addition to filter banks. These segments are localized as part of a bottom-up approach. We conjecture that, similar to the move in object recognition from semantic-object-parts to patches at random locations [61], patches from textured areas may prove to be similar in strength to identified sub-segments for texture segmentation (of course, we cannot use shape descriptors, since the shape is not given).

4.2.2 Patch-Based Methods

The filter bank approach was popular in the field of texture synthesis as well (e.g., [30, 49]), up until the advent of the patch based methods. In the few years since its publication [19, 37], the patch-based method has dominated the field of texture synthesis.

The basic use of the patch for texture synthesis consists of stitching together small overlapping patches of the input texture, such that their boundaries overlap (i.e., the gray value differences at the boundaries are minimal). This results in a new texture image, which seems to match the original texture in appearance, and has similar statistical properties. A similar approach was used for super-resolution [24] and for class-based edge detection [8]. The success of the patch-based methods has been extended to image completion [17] and to image denoising [2].

Patch-based methods were also recently shown to be extremely successful in object recognition [61, 60]. Similarities between patches taken from training images, and patches of the image to be classified, seem to be extremely powerful in discriminating between the object classes.

4.2.3 Non-Parametric Statistical Tests

Non-parametric statistical tests are preferred over their parametric counterparts when certain assumptions about the data cannot be made. For example, the two sample t-test assumes that the difference between the two independent samples it is applied to is normally distributed, while its non-parametric analog, the Wilcoxon Mann-Whitney test [66, 58], does not.

The Wilcoxon Mann-Whitney Test is one of the most powerful of the non-parametric tests for comparing two samples. It is used to test the null hypothesis that two samples have identical distribution functions against the alternative hypothesis that the two distribution functions differ only with respect to location (median), if at all. This

Given two vectors of samples v_a and v_b, of lengths n_a and n_b we wish to find a measure for the similarity of the underlying distributions.

1. Combine the samples into one vector of length $n_a + n_b$ and sort this vector.
2. Each observation in the combined vector has a rank. The first observation has a rank of 1, the second has a rank of 2, etc.
3. Let w_a be the sum of all of the ranks of elements originating from the vector v_a, and let w_b be a similar sum for v_b.
4. Use the statistic $w = min(w_a, w_b)$ to determine if the two distributions are different. Very low values of w suggest they are.

Figure 4-1: The Wilcoxon Mann-Whitney Test

test can also be applied when the observations are ranks, that is, ordinal data rather than direct measurements.

This test has several advantages that make it especially suitable for out application. First, it is valid for data from any distribution and is robust to outliers. Second, it reacts to differences both in the location of the distributions (i.e., to the difference of their median), and to the shape of the distributions. The test is well known, but since it is uncommon in computer vision circles, we include a description of it in Figure 4-1.

4.3 Patch-Based Texture Edge Detection

Our extremely simple method is illustrated in Figure 4-2. In essence, it tests whether a point in the image (x, y) is near a texture edge. Assume a situation where the point (x, y) is not near a texture edge. In this case the similarities between the patch surrounding (x, y) and the nearby patches to its left and right are drawn from the same distribution. In our experiments we measure similarities (or rather dissimilarities) by simply computing the Euclidean distance between the patch at (x, y) and the nearby

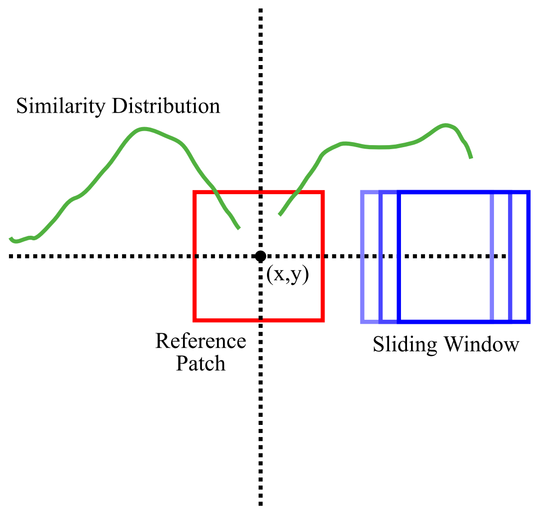


Figure 4-2: An illustration of the patches near the center patch that are used in order to compute the similarity distributions. Four distributions are sampled: D_{up}, D_{down}, D_{left} and D_{right}. The pixel at the center would be considered to lie on a texture edge if, according to the Wilcoxon Mann Whitney test, the distribution D_{up} is determined to be different from the distribution D_{down}, or if D_{left} is determined to be different from D_{right}.

patches. Our use of the actual image patch as a template, instead of a predefined filter bank, has the potential to be very sensitive to changes in the local texture.

Let D_{right},D_{left} be the distributions of similarities between the patch around (x, y) and the nearby patches. If there is a texture edge on the left side of (x, y), it is natural to expect the distributions D_{right} and D_{left} to be different. For example, it might be reasonable to assume larger similarities within D_{right}.

In order to find whether the two distributions are the same, we sample patches slightly to the left and to the right of the point (x, y). In the experiments we used a maximum distance of 15 pixels, and sampled at each pixel. We therefore sampled 15 similarities from each distribution. This small sample size, and the unexpected nature of the probability distribution of the similarities, make parametric tests inappropriate.

As mentioned above, we use the Wilcoxon Mann-Whitney test, which excels for samples small in size, and assumes very little about the nature of the distributions. The horizontal edge points are those points for which the test determines that the two distributions D_{right} and D_{left} are different. The same process is then applied vertically, and two similar distributions D_{up} and D_{down} are compared. For our application we

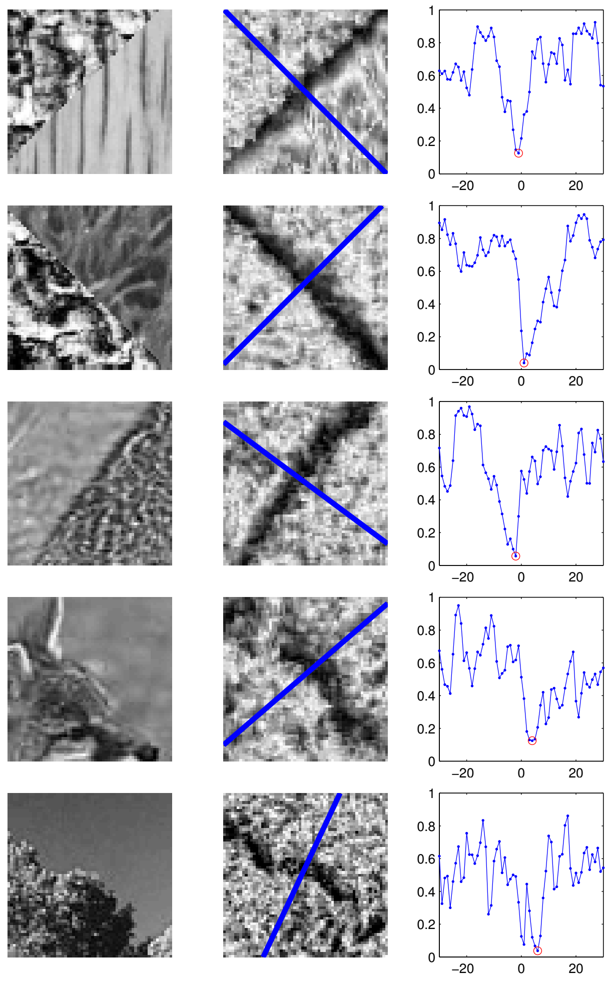


Figure 4-3: Profile of the edges obtained using our method. Left: the original part of the image. Middle: the texture edge we get. Right: the profile as a 2D plot. Note that the profile has a double edge effect, but it is rather minimal.

combine the two edge directions by taking the minimum value returned for the two tests.

Note, that since measurements from patches as far as 15 pixels away affect the distribution, we can expect the test score to change gradually. Moreover, since when (x, y) lies exactly on a texture edge, the patch around it is a hybrid patch, composed of two textures, we expect the difference between the distributions to be lower exactly at the edge. It turns out that for the small patch size we used in the experiments (3×3 pixels), these concerns did not affect the texture edges dramatically. This is demonstrated in Figure 4-3 with plots of several edge profiles.

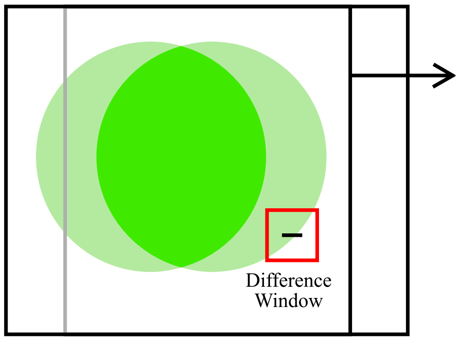


Figure 4-4: An illustration of the efficient method to sample the four distributions using vector operations. To sample all patch similarities at once (for all of the patches which are Δx pixels to the right or to the left), a copy of the image is translated by Δx pixels, and then subtracted from the original image. The difference is squared, and then summed at each patch in the image, which is a separable operation.

4.3.1 Efficient Computation

Every pixel in the image contributes to many patches, which are in turn compared with many overlapping patches. A naïve implementation would compute the difference of the same two pixels multiple times. Also, in some programming environments or hardware configurations (e.g., Matlab, designated graphics hardware), vector computations are done more efficiently than the repeated index-by-index computation.

The implementation we suggest is illustrated in Figure 4-4, and is based on computing all of the patch comparisons to patches at a distance of k in either the vertical or horizontal direction at the same time. In order to do so, one only needs to translate the image k pixels in either the horizontal or vertical direction, and subtract the resulting image from the original image. Since we are interested in the Euclidean distance, we square each value in the difference image, we then sum across all patches in the difference image. Since the summing operation can be performed as a separable convolution (i.e., can be done first in the horizontal direction, then vertically), the procedure can be made extremely efficient.

4.3.2 Flux

Below, we describe a method which was designed to circumvent concerns regarding double-edge detection. The method is based on the idea of the gradient flux [9],

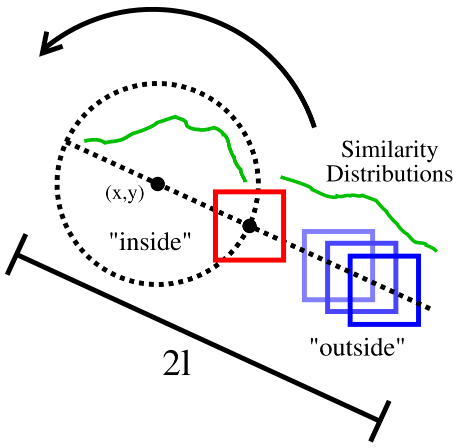


Figure 4-5: An alternative architecture using the flux idea. A pixel would not be on a texture edge if the similarity of points along a circle around it are as likely to be similar to points inside the circle, as they are to points outside the circle. For each point on the circle of radius r, the similarity of the patch around it is compared to patches along the line of length $2l$ which passes through the center point and the point on the circle. $l = 2r$ in the figure, but this does not have to be the case. We then keep a record of whether the closest patch was inside or outside the circle. For points that do not lie on a texture edge, the number of inside votes should be close to the number of outside votes.

where a medial axis transform is found by computing the distance transform of a shape, and finding the points for which the gradient flux (the sum of all gradients at the boundary of a small circle surrounding a point) is zero. Since on the medial axis the distances to the boundaries on both sides are equal, the gradients along the boundary of the circle will cancel each other.

Imagine the patches at a circle around (x, y) as being pulled to the most similar patch which is up to some distance away from them. The following process is illustrated in Figure 4-5. A circle of patches is drawn around the points (x, y), at a radius r. Each patch around a point on the circle's boundary is compared with patches along a line that connects the point on the circle and the point (x, y) (the circle's center). More concretely, the patch on the circle's boundary is compared with patches along that line that are at a distance up to l from the boundary, either inside or outside the circle.

If the circle is well within a uniform texture region, we can expect that the patch which most closely resembles the patch at the boundary will be either inside or outside the circle, with equal probabilities. We keep a record for each point along the boundary that states whether the closest patch to it was inside or outside the circle.

Sampling such points, in their natural order along the boundary of the circle, results in a sequence of the form $[in, out, in, in, out, in, \ldots]$. In order to check whether this sequence is random or not, we can apply any ordinal statistical score of randomness (a method which relies on the order of elements). An example for such a score is the Wilcoxon score, on top of which the Wilcoxon Mann-Whitney test is designed. This solution is not optimal though, since it depends on where we started to sample the circle (the direction is irrelevant in the ordinal tests we are aware of). We therefore take a maximum of the score over all possible starting points. A much simpler alternative is to just count whether the number of "in" equals the number of "out."

The flux-based method described here, though able to better handle the double edge problem, and perhaps more correct than the "grid" method we described first (Figure 4-2), is too computationally expensive. The increased accuracy we observed in our experiments did not justify the extra computational time spent. Therefore, in our experiments, we present results using the grid method which is much faster.

4.4 The Free-Form Deformable Model

The detected texture edges can be coupled with a hybrid deformable model that moves in the manner of free form deformations to achieve segmentation over the entire image domain.

The Euclidean distance transform is used to implicitly embed an evolving model as the zero level set of a higher dimensional distance function [46]. If we denote the model as $\mathcal{M}$, and the implicit model representation as a distance map $\Phi_{\mathcal{M}}$, then the shape defines a partition of the image domain: the region enclosed by $\mathcal{M}$, $[\mathcal{R}_{\mathcal{M}}]$; the background region $[\Omega - \mathcal{R}_{\mathcal{M}}]$; and the model itself, $[\partial\mathcal{R}_{\mathcal{M}}]$, which corresponds to the zero level set. Such model shape representation provides a feature space in which objective functions that are optimized using a gradient descent method are stable enough to use.

The deformations that a model can undergo are defined using a space warping technique: the Free Form Deformations (FFD) [55]. In essence, FFD deforms an object by manipulating a regular control lattice F overlaid on its volumetric embedding space. In the Incremental Free Form Deformations (IFFD) formulation used in [33], the deformation parameters $\mathbf{q}$ are the deformations of the control points in both x and y directions:

$$\mathbf{q} = \{(\delta F^x_{m,n}, \delta F^y_{m,n})\};\ (m,n) \in [1,M] \times [1,N],$$

where the control lattice is of size $M \times N$. The deformed position of a pixel $\mathbf{x} = (x,y)$ is given by $D(\mathbf{q};\mathbf{x}) = \mathbf{x} + \delta D(\mathbf{q};\mathbf{x})$. Given the deformation of the control lattice from F^0 to F, it is defined in terms of a tensor product of Cubic B-spline polynomials:

$$D(\mathbf{q};\mathbf{x}) = \sum_{k=0}^{3}\sum_{l=0}^{3} B_k(u)B_l(v)(F^0_{i+k,j+l} + \delta F_{i+k,j+l}), \tag{4.1}$$

$$\text{where } i = \lfloor \tfrac{x}{X} \cdot (M-1) \rfloor + 1,\ j = \lfloor \tfrac{y}{Y} \cdot (N-1) \rfloor + 1.$$

To find texture region boundaries given a simple-shape model initialized around a seed point, the dynamics of the free-form deformable model are derived from edge energy terms. Instead of intensity edges which fail to separate textured regions, we use the texture edges computed using our patch-based method above. Since true edges between different texture regions correspond to low values on our texture edge image I_t, we define a boundary data term that encourages model deformations that map the model boundary to pixel locations with smallest values on I_t. This energy term E_b is defined as follows:

$$E_b = \frac{1}{V(\partial\mathcal{R}_\mathcal{M})} \iint_{\partial\mathcal{R}_\mathcal{M}} \big(I_t(D(\mathbf{q};\mathbf{x}))\big)^2 d\mathbf{x},$$

where $V(\mathcal{R})$ represents the volume of a region $\mathcal{R}$.

The above boundary term E_b can help the model to converge to the exact edge location where the difference between two neighboring texture patches is maximized. However, it may cause the model to get stuck in local minima when the model is initialized far-away from the true boundary. To address this problem, we compute a binary edge map by thresholding on the texture edge image I_t. We encode this edge information by computing the un-signed distance transform of the edge map. The resulting distance map image is denoted by Φ_e. We then define another data term E_e which aims to minimize the sum-of-squared-differences between the implicit shape representation values both on the model and inside the model and the underlying distance values on Φ_e at corresponding deformed positions. This can be written as:

$$E_e = \frac{1}{V(\mathcal{R})} \iint_{\mathcal{R}} \big(\Phi_{\mathcal{M}}(\mathbf{x}) - \Phi_e(D(\mathbf{q};\mathbf{x}))\big)^2 d\mathbf{x},$$

where $\mathcal{R} = \mathcal{R}_{\mathcal{M}} \cup \partial\mathcal{R}_{\mathcal{M}}$. During optimization, when the model is still far-away from the true edges, this term serves as a two-way ballooning force that expands or shrinks the model along the gradient direction of Φ_e. At an edge with small gaps, this term also constrains the model to follow the "geodesic" path (i.e., with the shortest smooth path connecting the two open ends of a gap).

Combining the two data terms – the boundary term E_b and the thresholded edge term E_e, the overall energy functional is: $E = E_b + kE_e$, where k is a constant balancing the contributions from the two terms. We are able to omit an explicit model smoothness term here because of the strong implicit smoothness constraints imposed by FFD.

Both terms are differentiable with respect to the free-form-deformation parameters $\mathbf{q}$, and a gradient-descent based method is used to derive the model evolution equation for each element $\mathbf{q}_i$ in $\mathbf{q}$:

$$\frac{\partial E}{\partial \mathbf{q}_i} = \frac{\partial E_b}{\partial \mathbf{q}_i} + k\frac{\partial E_e}{\partial \mathbf{q}_i}, \tag{4.2}$$

where

$$\frac{\partial E_b}{\partial \mathbf{q}_i} = \frac{1}{V(\partial\mathcal{R}_{\mathcal{M}})} \iint_{\partial\mathcal{R}_{\mathcal{M}}} 2I_t(D(\mathbf{q};\mathbf{x})) \cdot \left(\nabla I_t(D(\mathbf{q};\mathbf{x})) \cdot \frac{\partial}{\partial \mathbf{q}_i} D(\mathbf{q};\mathbf{x})\right) d\mathbf{x}$$

$$\frac{\partial E_e}{\partial \mathbf{q}_i} = \frac{1}{V(\mathcal{R}_{\mathcal{M}} \cup \partial\mathcal{R}_{\mathcal{M}})} \iint_{\mathcal{R}_{\mathcal{M}} \cup \partial\mathcal{R}_{\mathcal{M}}} 2\big(\Phi_{\mathcal{M}}(\mathbf{x}) - \Phi_e(D(\mathbf{q};\mathbf{x}))\big) \cdot \left(-\nabla \Phi_e(D(\mathbf{q};\mathbf{x})) \cdot \frac{\partial}{\partial \mathbf{q}_i} D(\mathbf{q};\mathbf{x})\right) d\mathbf{x}$$

In the above formulas, the partial derivatives $\frac{\partial}{\partial \mathbf{q}_i} D(\mathbf{q};\mathbf{x})$ can be easily derived from the model deformation formula in Eq. 4.1.

The whole image is processed in the following manner: the first region is segmented by starting a deformable model at the center of the image. Another point well outside the first region is then used to initialize a second model, and a second region is segmented. The process continues until almost all of the points in the image are segmented. In the case where a new region grows into an old region, the two regions are joined together.

4.5 Experiments

Below we present our experiments. We would like to stress that all the results were obtained by using texture edges alone. We did not use intensity edges or color information. While these could be easily incorporated into our FFD framework by adding terms to the energy function, we avoided this in order to demonstrate the power of our texture analysis method.

4.5.1 Comparing methods for texture edge detection

The main purpose of these experiments is to demonstrate that Gabor based filter bank methods cannot be easily altered to local methods of deriving texture edges. Indeed, in [31, 32] a global clustering method was used to combine regions based on the filter bank descriptors; in [53] a method based on anisotropic diffusion in the direction of the global principle direction was suggested; in [26] the filter bank output was integrated along a region and was modified with statistics on the shape of small segments. One can also refer to the text of [53, 26], where the limitations of the local filter bank measurements are discussed.

In Figure 4-6, we compare our method, the Canny edge detector, and a method based on [32], where for each pixel we plot the maximum difference (using the original parameters and distance measure) of the block around it to the nearest four blocks. The results are similar if the Wilcoxon Mann-Whitney test is used instead of χ^2. As can be seen, this “alternative” is not doing well at all. Further evidence can be found in Figure 4-a of [53].

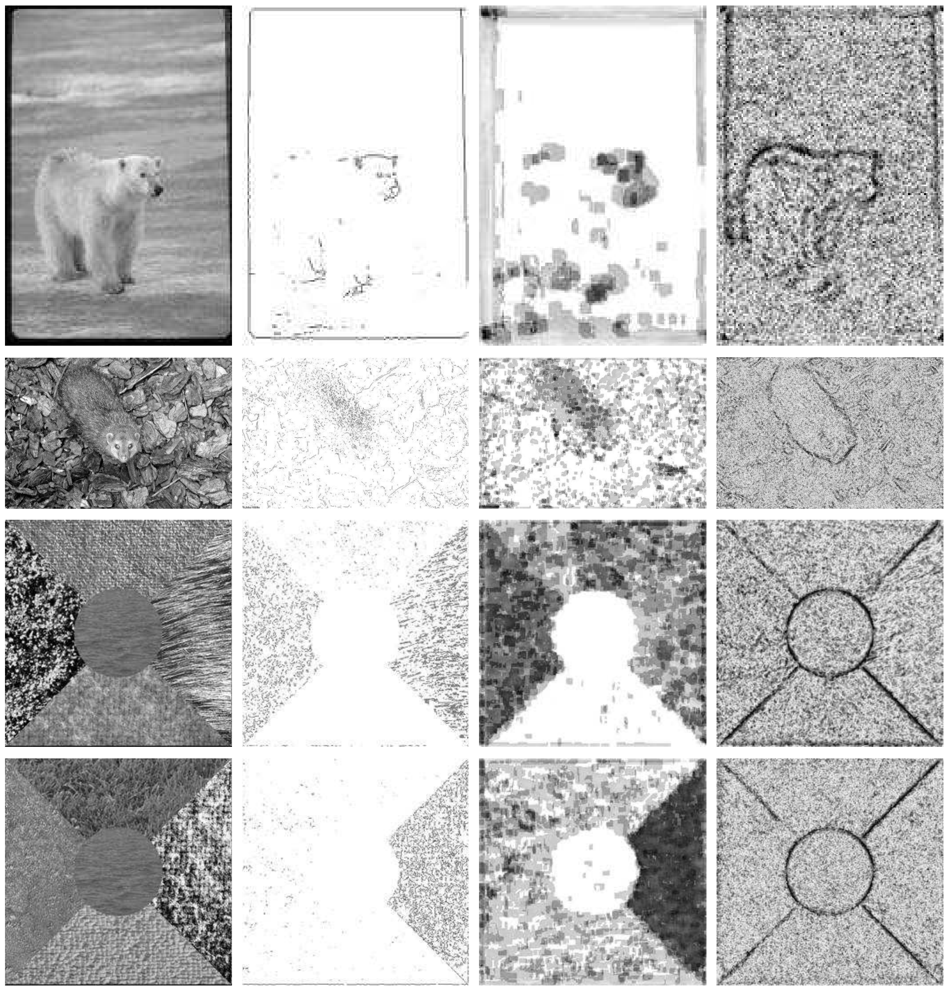

(a) (b) (c) (d)

Figure 4-6: Comparison of edge detection performed on the original gray images (a), using the Canny edge detector (b), filter bank dissimilarity based on [32](c), and our method (d).

4.5.2 Experiments on Texture Mosaics

Next, we show results on the texture mosaic images constructed by the authors of [32], which are available online at `http://www-dbv.cs.uni-bonn.de/image/mixture.tar.gz`. This data set contains mosaics generated from a set of 86 micro-patterns from the Brodatz texture album [12]. Each image contains 5 random textures out of this set, and is of size 512×512.

As mentioned above, the parameters were fixed to a patch size of 3×3 and to a sample size of 15 in each direction. The process of retrieving the edges is very

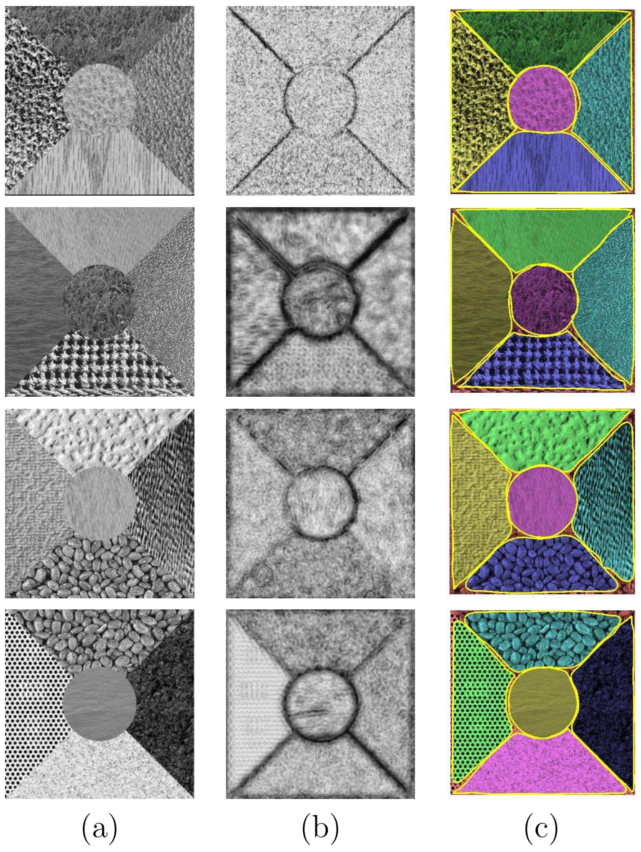


Figure 4-7: Results of our edge detection and texture segmentation methods on several mosaics constructed by the authors of [32]. (a) the original images. (b) the recovered texture edges. (c) the resulting segmentation.

efficient, and our Matlab implementation can detect edges for such an image in under five seconds.

The results on the several challenging (with regard to their lack of intensity edges) images in this data set are presented in Figure 4-7.

Real image experiments In Figure B, we present experiments on images taken from the first 25 grayscale testing images of the *Berkeley Segmentation Dataset*[1]. The figure illustrates the original images, the recovered texture edges, and the resulting segmentation. The dark area below the wolf in the top-right image is due to an artificial texture created by the image being uniformly saturated (a maximum intensity of 255) in that region. Since the neighboring distributions are homogeneous, the

[1]`http://www.cs.berkeley.edu/projects/vision/grouping/segbench/`

Wilcoxon test fails to mix them during sorting and judges the whole region to be a texture boundary. This can be avoided by first randomly sorting the distributions before applying the Wilcoxon test. The images in the lower right demonstrate the detection of texture edges that also constitute intensity edges. We did not use any intensity edges, but as can be seen in the first image, edges between regions of uniform but different intensities are detected by our method.

4.6 Summary and Conclusions

The patch based technologies, which are based on local gray value representations and correlations between gray values, have proven to be successful in many computer vision domains, and suggest an appealing alternative to filter bank approaches. While there is no doubt that their recent proliferation is partly due to the increasing computational power available, the representation itself seems inherently powerful.

We used patches in order to compute texture edges. The edge representation (as opposed to a representation of regions using some form of descriptor) is powerful in that it can be readily combined with global optimization based-segmentation (e.g. “snakes”). Most energy-based methods do not deal with texture edges. Attempts that have been made in the past to incorporate texture into these methods used simple texture descriptors such as mean intensity of a region or the variance of the intensity in that region [47, 52], and were computationally expensive.

By using our patch-based texture edge detection technique, combined with Free-Form Deformations, we are able to suggest a tractable solution, which enjoys both rich texture information, and the advantages of a global solution. These advantages include the detection of a smooth boundary, which is globally salient. We focused mainly on texture edges, but one can easily add the traditional energy terms for intensity edges and color edges to the framework described, making it complete for image segmentation. This completeness was available in the affinity based approaches, but not in the energy-based methods.

Appendix A

Tables

Table A.1: Performance Comparison on UCI Repository Data Sets

Set	Classifier	Mean Error	Variance Bias.–Unb.= Net	Feat. Used	Set	Mean Error	Variance Bias.–Unb.= Net	Feat. Used
Arrhythmia	Linear SVM	37.0± 6.0	0.05– 0.03= 0.02	279.0	Dermatology	0.8± 1.0	0.32– 0.01= 0.31	34.0
	Linear SVM-KO	36.6± 5.4	0.05– 0.03= 0.01	279.0		1.1± 1.5	0.26– 0.01= 0.25	34.0
	SVM-RFE	36.0± 3.6	0.08– 0.05= 0.03	140.0		15.1±13.3	0.20– 0.07= 0.13	17.0
	Gaussian SVM	48.9± 4.4	0.00– 0.00= 0.00	45.0		13.3± 4.3	0.09– 0.04= 0.06	34.0
	AdaBoost	42.1± 5.7	0.26– 0.22= 0.03	14.4		2.4± 1.5	0.76– 0.65= 0.11	2.2
	AdaBoost-KO	37.9± 3.9	0.16– 0.12= 0.04	41.4		1.7± 1.7	0.40– 0.05= 0.36	12.2
	AdaBoost-NR	39.1± 3.4	0.08– 0.06= 0.03	100.0		1.1± 1.3	0.17– 0.01= 0.16	34.0
	AdaBoost-VC	46.9± 6.2	0.62– 0.59= 0.03	1.1		2.5± 1.7	0.71– 0.65= 0.06	2.1
	GentleBoost	36.7± 3.6	0.15– 0.10= 0.05	43.4		2.3± 1.5	0.58– 0.49= 0.09	5.6
	GentleBoost-KO	34.5± 2.5	0.07– 0.04= 0.02	127.3		0.8± 1.1	0.19– 0.01= 0.19	27.3
	GentleBoost-NR	36.8± 3.5	0.10– 0.06= 0.04	100.0		2.6± 2.7	0.29– 0.03= 0.27	34.0
	GentleBoost-VC	48.0± 5.0	0.30– 0.30=-0.00	6.9		9.1±20.9	0.15– 0.49=-0.34	6.9
	GentleBoost-KO-VC	34.3± 2.6	0.06– 0.04= 0.02	93.3		0.5± 0.4	0.35– 0.01= 0.34	26.3
	GentleBoost-NI	33.4± 3.9	0.14– 0.09= 0.05	47.0		1.2± 1.3	0.57– 0.21= 0.37	9.8

Set	Classifier	Mean Error	Variance Bias.–Unb.= Net	Feat. Used	Set	Mean Error	Variance Bias.–Unb.= Net	Feat. Used
E.coli	Linear SVM	10.4±12.2	0.34– 0.22= 0.12	7.0	Glass	45.8± 6.0	0.23– 0.22= 0.01	8.0
	Linear SVM-KO	6.9± 3.8	0.54– 0.23= 0.31	7.0		42.4± 5.9	0.31– 0.28= 0.03	8.0
	SVM-RFE	7.7± 4.4	0.73– 0.43= 0.30	4.0		48.3± 4.6	0.24– 0.24= 0.00	4.0
	Gaussian SVM	4.9± 2.1	0.53– 0.25= 0.28	7.0		39.7± 8.0	0.17– 0.23=-0.06	8.0
	AdaBoost	9.9± 2.6	0.64– 0.45= 0.19	3.9		40.6± 8.9	0.44– 0.38= 0.07	5.2
	AdaBoost-KO	9.3± 2.2	0.59– 0.27= 0.32	5.8		42.5± 5.9	0.35– 0.27= 0.08	7.9
	AdaBoost-NR	12.8± 3.8	0.44– 0.22= 0.22	7.0		38.6± 7.0	0.30– 0.28= 0.02	8.0
	AdaBoost-VC	14.3± 4.0	0.74– 0.71= 0.04	1.9		46.4±13.9	0.67– 0.69=-0.02	1.4
	GentleBoost	8.0± 2.4	0.61– 0.40= 0.21	4.6		39.9± 7.6	0.34– 0.32= 0.03	6.1
	GentleBoost-KO	6.2± 1.7	0.57– 0.23= 0.35	6.1		34.9± 6.6	0.34– 0.26= 0.08	8.0
	GentleBoost-NR	12.6± 6.9	0.44– 0.21= 0.24	7.0		37.7± 6.0	0.33– 0.28= 0.05	8.0
	GentleBoost-VC	29.7±18.6	0.14– 0.26=-0.13	4.3		42.3± 9.2	0.31– 0.31=-0.00	5.4
	GentleBoost-KO-VC	28.1±20.8	0.41– 0.28= 0.12	6.1		38.3± 5.3	0.26– 0.26=-0.00	8.0
	GentleBoost-NI	5.7± 1.7	0.49– 0.24= 0.25	7.0		32.7± 6.8	0.39– 0.36= 0.04	6.3

Set	Classifier	Mean Error	Variance Bias.–Unb.= Net	Feat. Used	Set	Mean Error	Variance Bias.–Unb.= Net	Feat. Used
Heart	Linear SVM	26.6± 4.2	0.36– 0.18= 0.18	13.0	Housing	17.5± 2.2	0.46– 0.27= 0.19	13.0
	Linear SVM-KO	24.9± 4.1	0.33– 0.19= 0.14	13.0		17.0± 2.6	0.39– 0.22= 0.17	13.0
	SVM-RFE	24.7± 4.2	0.41– 0.27= 0.14	7.0		19.6± 2.7	0.52– 0.34= 0.18	7.0
	Gaussian SVM	48.7± 3.6	0.05– 0.06=-0.00	13.0		37.6± 5.8	0.12– 0.15=-0.03	13.0
	AdaBoost	26.2± 4.0	0.46– 0.32= 0.15	6.5		17.2± 2.1	0.54– 0.34= 0.20	5.5
	AdaBoost-KO	24.2± 3.5	0.33– 0.18= 0.15	10.9		17.3± 2.8	0.37– 0.19= 0.18	9.9
	AdaBoost-NR	24.3± 2.6	0.28– 0.15= 0.13	13.0		18.9± 3.7	0.27– 0.13= 0.14	13.0
	AdaBoost-VC	36.0±14.5	0.76– 0.72= 0.05	2.1		20.3± 4.6	0.65– 0.53= 0.12	3.0
	GentleBoost	28.8± 7.3	0.32– 0.24= 0.08	7.3		18.8± 3.0	0.40– 0.22= 0.18	10.7
	GentleBoost-KO	24.1± 2.3	0.33– 0.16= 0.17	13.0		17.5± 1.8	0.38– 0.15= 0.24	13.0
	GentleBoost-NR	25.4± 4.0	0.32– 0.16= 0.17	13.0		19.0± 4.2	0.34– 0.14= 0.20	13.0
	GentleBoost-VC	37.9±10.6	0.10– 0.12=-0.02	6.7		42.4± 9.7	0.14– 0.21=-0.07	4.6
	GentleBoost-KO-VC	39.2± 8.2	0.16– 0.15= 0.01	11.5		43.2±14.0	0.12– 0.14=-0.02	12.5
	GentleBoost-NI	23.9± 3.5	0.29– 0.17= 0.12	12.2		17.6± 1.8	0.39– 0.19= 0.21	12.6

Set	Classifier	Mean Error	Variance Bias.–Unb.= Net	Feat. Used	Set	Mean Error	Variance Bias.–Unb.= Net	Feat. Used
Letters	Linear SVM	17.5± 2.2	0.46– 0.27= 0.19	13.0	Segmentation	12.0±11.0	0.34– 0.16= 0.18	19.0
	Linear SVM-KO	17.0± 2.6	0.39– 0.22= 0.17	13.0		10.4±10.0	0.33– 0.15= 0.17	19.0
	SVM-RFE	19.6± 2.7	0.52– 0.34= 0.18	7.0		16.6±12.1	0.43– 0.20= 0.24	10.0
	Gaussian SVM	37.6± 5.8	0.12– 0.15=-0.03	13.0		14.3± 1.0	0.00– 0.00= 0.00	19.0
	AdaBoost	17.2± 2.1	0.54– 0.34= 0.20	5.5		5.9± 4.5	0.37– 0.23= 0.14	4.3
	AdaBoost-KO	17.3± 2.8	0.37– 0.19= 0.18	9.9		6.8± 3.8	0.24– 0.08= 0.16	11.4
	AdaBoost-NR	18.9± 3.7	0.27– 0.13= 0.14	13.0		9.0± 3.3	0.19– 0.05= 0.14	19.0
	AdaBoost-VC	20.3± 4.6	0.65– 0.53= 0.12	3.0		6.7± 3.3	0.27– 0.31=-0.04	1.1
	GentleBoost	18.8± 3.0	0.40– 0.22= 0.18	10.7		6.6± 3.6	0.31– 0.18= 0.13	5.5
	GentleBoost-KO	17.5± 1.8	0.38– 0.15= 0.24	13.0		6.1± 4.4	0.10– 0.03= 0.08	18.0
	GentleBoost-NR	19.0± 4.2	0.34– 0.14= 0.20	13.0		10.0± 3.8	0.18– 0.04= 0.15	19.0
	GentleBoost-VC	42.4± 9.7	0.14– 0.21=-0.07	4.6		9.5± 5.3	0.42– 0.28= 0.14	5.8
	GentleBoost-KO-VC	43.2±14.0	0.12– 0.14=-0.02	12.5		9.6± 4.7	0.22– 0.07= 0.15	17.9
	GentleBoost-NI	17.6± 1.8	0.39– 0.19= 0.21	12.6		6.3± 3.5	0.31– 0.16= 0.15	5.7

Set	Classifier	Mean Error	Variance Bias.–Unb.= Net	Feat. Used	Set	Mean Error	Variance Bias.–Unb.=Net	Feat. Used
Wine	Linear SVM	11.4± 4.0	0.38– 0.17= 0.21	13.0	Yeast	31.1± 0.3	0.00– 0.00=0.00	8.0
	Linear SVM-KO	9.7± 4.0	0.36– 0.14= 0.22	13.0		31.0± 0.3	0.00– 0.00=0.00	8.0
	SVM-RFE	16.6± 9.3	0.46– 0.29= 0.17	7.0		32.6± 2.2	0.58– 0.40=0.18	4.0
	Gaussian SVM	36.8± 2.7	0.05– 0.04= 0.01	13.0		28.6± 0.7	0.36– 0.22=0.14	8.0
	AdaBoost	11.8± 5.1	0.76– 0.56= 0.20	4.0		38.9± 9.4	0.56– 0.38=0.18	1.7
	AdaBoost-KO	7.8± 2.8	0.40– 0.13= 0.27	11.0		31.2± 1.2	0.08– 0.05=0.03	4.1
	AdaBoost-NR	10.1± 1.6	0.33– 0.13= 0.20	13.0		31.9± 1.4	0.04– 0.02=0.02	8.0
	AdaBoost-VC	13.8± 5.4	0.74– 0.75=-0.01	1.7		39.5±10.1	0.57– 0.38=0.19	1.4
	GentleBoost	11.1± 5.4	0.76– 0.51= 0.25	5.7		34.2± 2.6	0.57– 0.42=0.15	5.3
	GentleBoost-KO	6.8± 1.6	0.40– 0.10= 0.30	12.9		32.2± 1.7	0.46– 0.31=0.15	8.0
	GentleBoost-NR	10.5± 3.0	0.41– 0.12= 0.30	13.0		31.6± 1.8	0.32– 0.22=0.10	8.0
	GentleBoost-VC	11.9± 4.2	0.74– 0.56= 0.18	4.7		43.6±10.3	0.42– 0.36=0.06	3.8
	GentleBoost-KO-VC	5.2± 0.9	0.43– 0.10= 0.33	13.0		40.4± 9.3	0.23– 0.17=0.05	6.9
	GentleBoost-NI	8.3± 2.9	0.38– 0.18= 0.20	11.0		31.9± 2.1	0.49– 0.33=0.16	7.1

Table A.2: Performance on Caltech Objects for Few Examples

	Classifier	Total Positive Examples 1	3	6	10	15	30
Airplanes	Linear SVM	26.1	16.8	12.6	11.5	10.7	8.1
	Linear SVM KO	26.3	16.8	12.4	11.6	10.9	8.5
	Poly SVM (order 2)	27.8	16.5	12.4	11.1	10.1	7.6
	Gaussian SVM $\gamma = 10^{-4}$	26.9	18.4	12.8	12.7	12.0	10.9
	l_0-norm SVM	27.2	16.1	12.7	13.4	13.3	11.0
	SVM-RFE	27.4	15.9	12.4	11.7	10.4	8.5
	AdaBoost	47.6	41.4	32.4	18.8	11.3	5.8
	AdaBoostKO	17.4	15.8	12.7	10.3	8.7	5.8
	AdaBoost-VC3	47.6	41.4	87.0	27.3	14.4	6.8
	GentleBoost	55.1	41.4	30.5	18.1	11.2	5.6
	GentleBoostKO	43.4	22.9	16.8	13.0	10.8	6.9
	GentleBoost-NR	23.4	14.7	10.3	9.0	7.4	5.1
	GentleBoost-VC3	55.1	41.4	30.7	18.2	11.3	5.7
	GentleBoostKO-VC3	41.8	23.2	16.7	12.9	10.6	7.5
	GentleBoostNI0.1	56.2	32.0	24.5	13.1	9.4	6.6

	Classifier	Total Positive Examples 1	3	6	10	15	30
Cars	Linear SVM	7.2	6.2	5.5	4.2	3.2	1.9
	Linear SVM KO	7.2	6.2	5.5	4.1	3.2	1.9
	Poly. SVM (order 2)	7.5	7.1	6.0	4.4	3.2	2.0
	Gaussian SVM $\gamma = 10^{-4}$	7.0	6.9	7.0	6.8	6.9	6.8
	l_0-norm SVM	5.9	5.6	5.1	5.4	4.8	5.0
	SVM-RFE	5.9	5.4	4.8	3.6	3.3	1.9
	AdaBoost	60.3	40.0	34.7	20.4	9.9	3.1
	AdaBoostKO	8.6	6.6	5.9	4.6	4.1	2.5
	AdaBoost-VC3	60.3	40.0	91.9	20.8	10.6	3.6
	GentleBoost	68.0	37.3	30.5	20.3	9.7	3.2
	GentleBoostKO	28.3	14.4	12.0	7.3	4.3	3.1
	GentleBoost-NR	8.9	6.0	5.6	3.9	3.2	2.0
	GentleBoost-VC3	68.0	37.3	30.5	20.3	9.9	3.0
	GentleBoostKO-VC3	29.5	13.7	12.2	7.3	4.6	3.0
	GentleBoostNI0.1	44.7	29.0	18.8	9.6	5.4	2.5

	Classifier	Faces: Total Positive Examples 1	3	6	10	15	30	Leaves: Total Positive Examples 1	3	6	10	15	30	Motorbikes: Total Positive Examples 1	3	6	10	15	30
	Linear SVM	34.7	21.2	12.5	6.7	6.9	3.9	31.3	19.4	12.9	8.7	8.0	6.0	13.0	7.8	7.4	5.7	5.3	3.6
	Linear SVM KO	35.0	20.6	12.0	6.9	6.7	4.2	30.9	19.7	12.7	9.1	8.1	5.9	13.2	7.9	7.6	5.7	5.3	3.7
	Poly. SVM (order 2)	33.5	21.3	11.5	7.3	6.0	4.0	33.3	20.1	13.1	9.2	8.7	6.7	14.5	8.8	7.8	5.7	5.6	3.8
	Gaussian SVM 10^{-4}	33.3	24.2	15.3	13.2	10.4	8.0	31.2	20.5	15.3	12.8	12.4	11.9	14.8	7.7	6.9	6.0	5.9	5.3
	l_0-norm SVM	35.4	23.5	13.4	10.2	10.1	9.6	32.3	20.9	14.5	12.6	11.9	9.9	11.3	9.0	7.5	6.9	6.6	7.1
	SVM-RFE	35.5	23.6	13.5	9.6	7.5	5.0	32.6	20.7	14.2	11.4	9.6	7.2	11.6	8.8	7.9	6.2	5.6	4.4
	AdaBoost	59.3	39.4	26.3	15.5	11.8	4.0	50.8	42.5	33.2	12.9	8.9	6.2	56.8	36.3	23.3	22.8	13.3	4.0
	AdaBoostKO	28.6	12.1	7.5	6.1	5.4	3.9	21.2	16.2	12.6	9.1	7.6	4.1	11.6	8.0	6.1	4.8	4.4	3.8
	AdaBoost-VC3	59.3	39.4	90.5	26.6	12.8	4.4	50.8	42.5	85.7	32.6	12.3	7.1	56.8	36.3	92.0	23.0	14.4	4.1
	GentleBoost	61.7	48.7	26.3	15.2	11.1	3.9	63.7	39.6	32.4	12.3	8.8	5.6	63.5	35.7	22.1	22.8	13.2	4.1
	GentleBoostKO	53.3	19.5	10.1	8.0	6.4	4.4	43.4	29.6	15.5	10.6	8.6	5.1	31.5	10.6	7.7	6.1	5.5	4.5
	GentleBoost-NR	33.2	11.3	5.5	4.7	3.8	2.7	26.0	15.0	10.5	8.2	7.5	4.8	15.5	8.5	6.2	4.3	4.1	3.4
	GentleBoost-VC3	61.7	48.7	26.3	15.5	11.6	3.9	63.7	39.6	32.6	12.6	9.3	5.4	63.5	35.7	22.1	22.8	13.2	4.0
	GentleBoostKO-VC3	52.0	18.8	9.9	8.0	6.3	4.4	42.8	29.9	15.2	10.3	8.4	5.3	31.7	10.6	7.8	6.0	5.6	4.5
	GentleBoostNI0.1	60.1	33.0	16.3	7.8	5.4	3.4	59.4	39.5	12.4	8.2	7.1	4.9	44.9	21.7	23.7	14.1	7.2	4.0

Appendix B

Figures

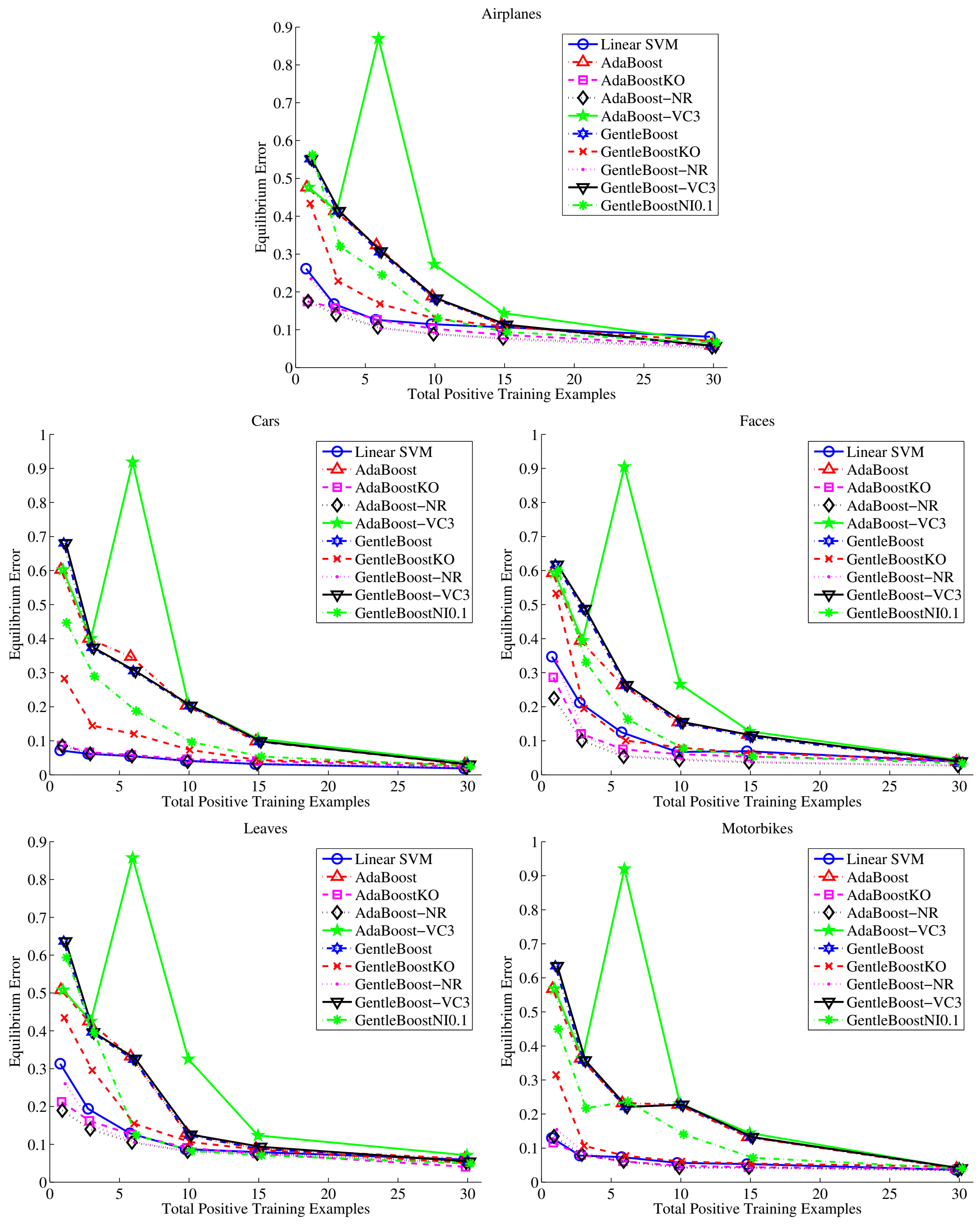


Figure B-1: A comparison between boosting variants using the C2 features on the five Caltech data sets: Airplanes, Cars, Faces, Leaves and Motorbikes. The graphs show the the equilibrium error rate vs. the number of training examples used from the class we want to detect. In each experiment, the test set was fixed to be the same as those described in [21], except for the random examples from the background set that were put aside for training. Each experiment was repeated 10 times.

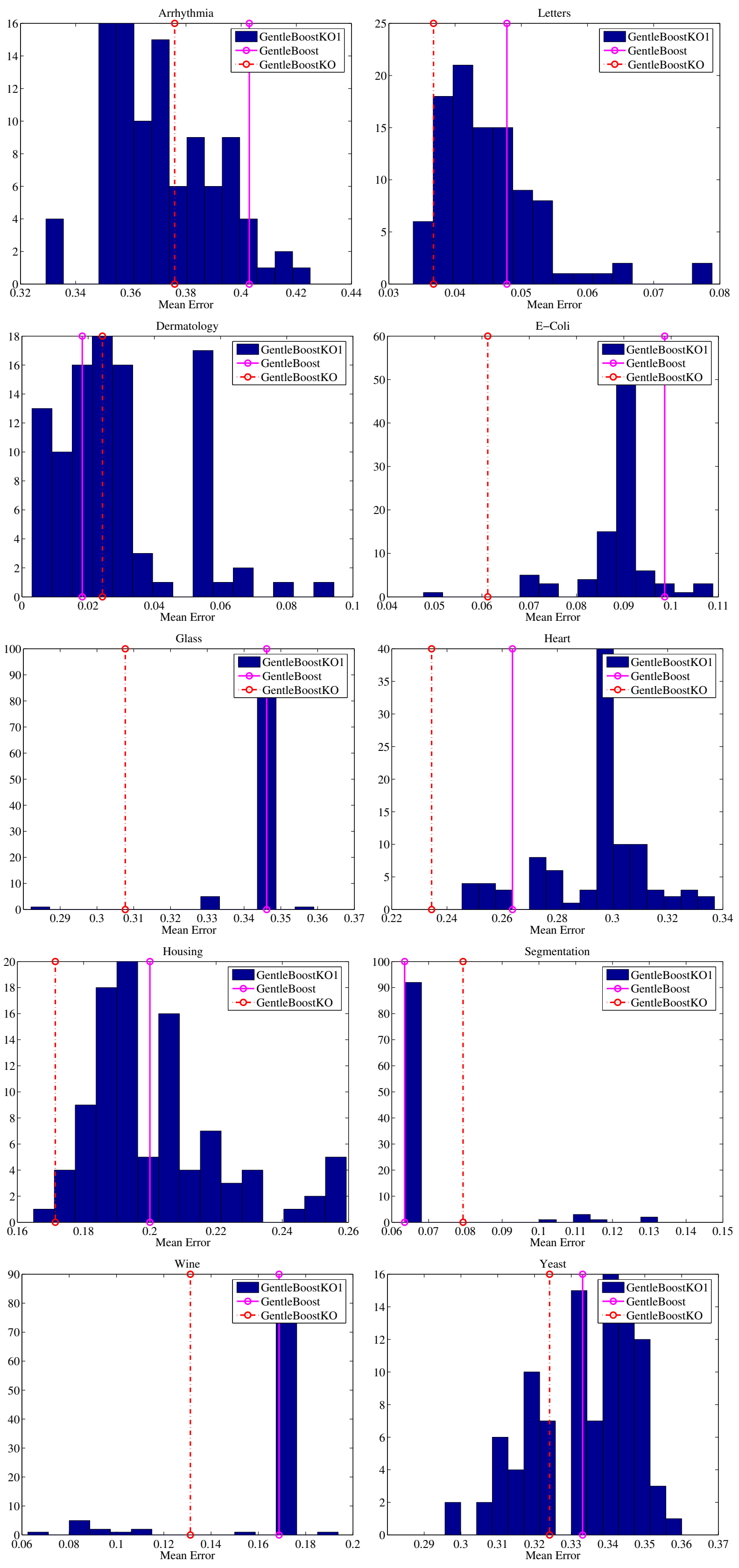


Figure B-2: Performance histograms of the effect of a single knockout example on GentleBoost.

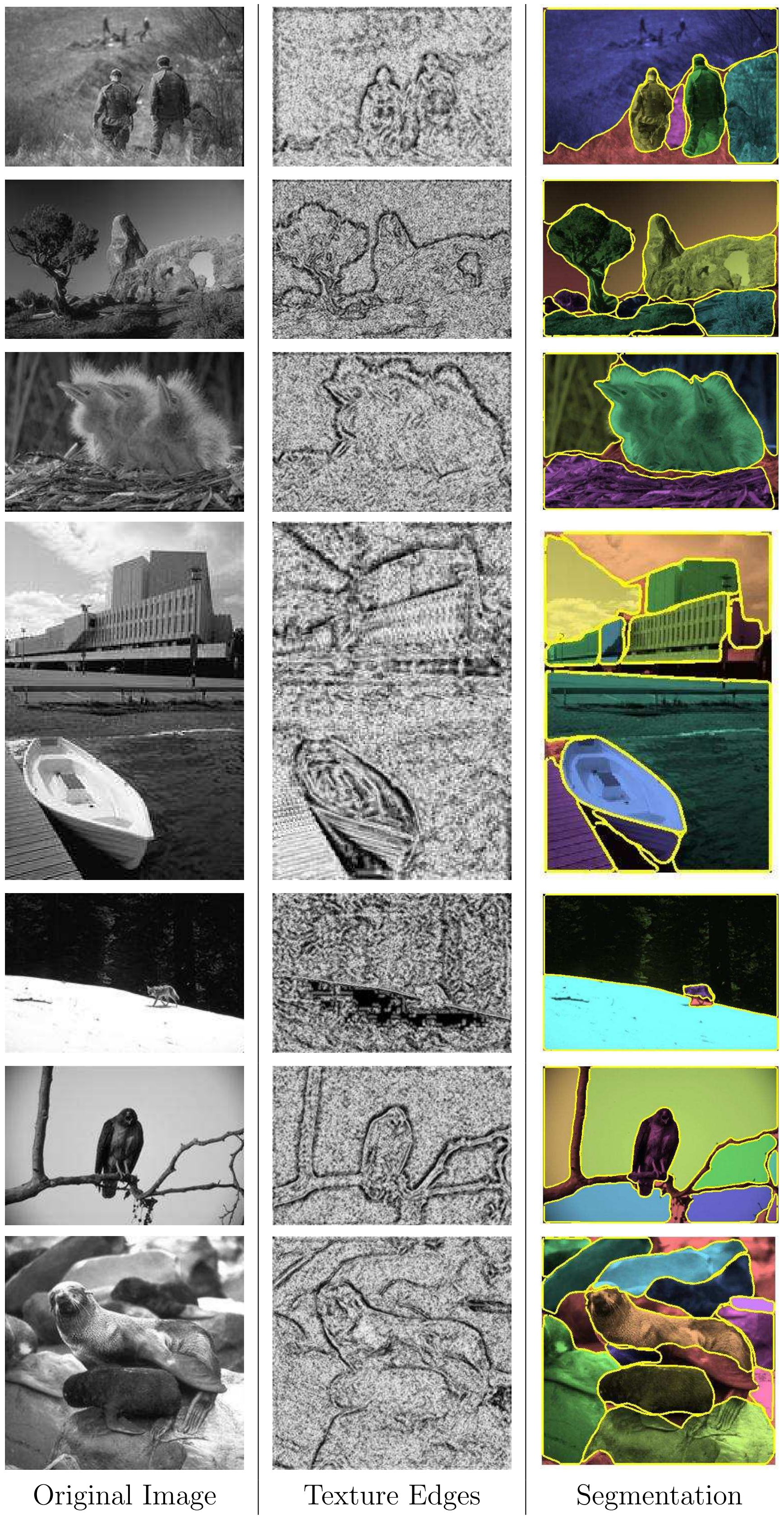


Figure B-3: Segmentations of Berkeley Database Images

Bibliography

[1] A. A. Amini, Y. Chen, M. Elayyadi, and P. Radeva. Tag surface reconstruction and tracking of myocardial beads from spamm-mri with parametric b-spline surfaces. *IEEE Transactions on Medical Imaging*, 20(2):94–103, 2001.

[2] S. Awate and R. Whitaker. Image denoising with unsupervised, information-theoretic, adaptive filtering. Technical Report UUCS-04-013, The University of Utah, 2004.

[3] E. Bardinet, L. D. Cohen, and N. Ayache. A parametric deformable model to fit unstructured 3d data. *Computer Vision and Image Understanding*, 71(1):39–54, 1998.

[4] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: a geometric framework for learning from examples. Technical Report TR-2004-06, CS Technical Report.

[5] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schlummer, and Z. Yakhini. Tissue classification with gene expression profiles. *Journal of Computational Biology*, 7:559–584, 2000.

[6] C. M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[7] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[8] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. *European Conference on Computer Vision*, pages 109–124, 2002.

[9] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. *Medical Image Analysis*, 2004.

[10] L. Breiman. Heuristics of instability and stabilization in model selection. *Ann. Statist.*, 24(6):2350–2383, 1996.

[11] L. Breiman. Random forests. *Biologically Motivated Computer Vision*, 45(1):5–32, 2001.

[12] P. Brodatz. *Textures: A Photographic Album for Artists and Designers.* Dover Publications, 1966.

[13] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Conference on Computer Vision*, pages 694–699, 1995.

[14] G. Csurka, C. Bray, C. Dance, and L. Fan. Visual categorization with bags of keypoints. In *European Conference on Computer Vision*, 2004.

[15] L. Devroye. *A Course in Density Estimation.* Birkhauser, 1987.

[16] P. Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *International Conference on Artificial Intelligence*, 2000.

[17] I. Drori, D. Cohen-Or, and H. Yeshurun. Fragment-based image completion. *SIGGRAPH*, 2003.

[18] S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97(457):77–87, 2002.

[19] A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *SIGGRAPH*, pages 341–346, 2001.

[20] L. Fei-Fei, R. Fergus, and P. Perona. A bayesian approach to unsupervised one-shot learning of object categories. *International Conference on Computer Vision*, 2003.

[21] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. *Computer Vision and Pattern Recognition*, 2003.

[22] I. Fogel and D. Sagi. Gabor filters as texture discriminator. *Biological Cybernetics*, 61:103–113, 1989.

[23] C. Fowlkes, D. Martin, and J. Malik. Glearning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. *Computer Vision and Pattern Recognition*, 2003.

[24] W. T. Freeman, T. Jones, and E. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, pages 56–65, 2002.

[25] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Ann. Statist.*, 28(2):337–407, 2000.

[26] M. Galun, E. Sharon, R. Basri, and A. Brandt. Texture segmentation by multiscale aggregation of filter responses and shape elements. *International Conference on Computer Vision*, pages 716–723, 2003.

[27] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 1992.

[28] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Journal of Machine Learning*, 46(1-3):389–422, 2002.

[29] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[30] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. *SIGGRAPH*, pages 229–238, 1995.

[31] T. Hofmann, J. Puzicha, and J. M. Buhmann. Unsupervised segmentation of textured images by pairwise data clustering. *ICIP*, 1996.

[32] T. Hofmann, J. Puzicha, and J. M. Buhmann. An optimization approach to unsupervised hierarchical texture segmentation. *ICIP*, 1997.

[33] X. Huang, D. Metaxas, and T. Chen. Metamorphs: Deformable shape and texture models. *Computer Vision and Pattern Recognition*, 2004.

[34] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–331, 1987.

[35] A. E. Klon, M. Glick, and J. W. Davies. Application of machine learning to improve the results of high-throughput docking against the hiv-1 protease. *J. Chem Inf. Comput. Sci.*, 44:2216–2224, 2004.

[36] V. Koltchinskii and D. Panchenko. Complexities of convex combinations and bounding the generalization error in classification. *Ann. Statist.*, 2003.

[37] L. Liang, C. Liu, Y. Xu, B. Guo, and H. Y. Shum. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Research, 2001.

[38] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. *Deutsche Arbeitsgemeinschaft für Mustererkennung*, 2003.

[39] P. M. Long and V. B. Vega. Boosting and microarray data. *Machine Learning*, 52:31–44, 2003.

[40] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[41] G. Lugosi. Concentration-of-measure inequalities. 2003.

[42] J. Malik and P. Perona. Preattentive texture discrimination with early mechanisms. *J. Optical Soc. Am. A*, 7(5):923–932, May 1990.

[43] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modelling with front propagation: a level set approach. In *TPAMI*, volume 17, pages 158–175, 1995.

[44] R. Meir and G. Rätsch. An introduction to boosting and leveraging. *Advanced lectures on machine learning*, pages 118–183, 2003.

[45] A. Neumaier. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review*, 40(3):636–666, 1998.

[46] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed : Algorithms based on the hamilton-jacobi formulation. *Journal of Comp. Physics*, 79(12):12–49, 1988.

[47] N. Paragios and R. Deriche. Geodesic active regions and level set methods for supervised texture segmentation. *International Journal of Computer Vision*, 2002.

[48] S. L. Pomeroy, P. Tamayo, L. Gaasenbeek, M. amd Sturla, M. Angelo, M. McLaughlin, J. Kim, L. Goumnerova, P. Black, C. Lau, J. Allen, D. Zagzag, J. Olson, T. Curran, C. Wetmore, J. Biegel, T. Poggio, S. Mukherjee, R. Rifkin, A. Califano, G. Stolovitzky, D. Louis, J. Mesirov, E. S. Lander, and T. Golub. Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, 415(24):436–442, 2002.

[49] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1), 2000.

[50] S. Ramaswamy. Personal communication. 2004.

[51] Y. Raviv and N. Intrator. Bootstrapping with noise: An effective regularization technique. *Connection Science*, 8(3):355–372, 1996.

[52] M. Rousson and R. Deriche. A variational framework for active and adaptive segmentation of vector valued images. In *Workshop on Motion and Video Computing*, 2002.

[53] Y. Rubner and C. Tomasi. Coalescing texture descriptors. In *ARPA Image Understanding Workshop*, 1996.

[54] R. E. Schapire. A brief introduction to boosting. In *International Joint Conference on Artificial Intelligence*, volume 8, pages 1401–1406, 1999.

[55] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Annual Conference on Computer Graphics*, 1986.

[56] T. Serre, M. Riesenhuber, J. Louie, and T. Poggio. On the role of object-specific features for real world object recognition in biological vision. *Biologically Motivated Computer Vision*, 2002.

[57] M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. Aguiar, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus, T. S. Ray, M. A. Koval, K. W. Last, A. Norton, T. A. Lister, J. Mesirov, D. S. Neuberg, E. S. Lander, J. C. Aster, and T. R. Golub. Diffuse large b-cell lymphoma outcome prediction by gene expression profiling and supervised machine learning. *Nature Medicine*, 8(1):68–74, 2002.

[58] S. Siegel and N. J. Castellan. *Nonparametric Statistics for the Behavioural Sciences*. McGraw-Hill, 1988.

[59] L. H. Staib and J. S. Duncan. Boundary finding with parametrically deformable models. *TPAMI*, 14(11), 1992.

[60] A. Torralba, K. P. Murphy, and W. T. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. *Computer Vision and Pattern Recognition*, 2004.

[61] S. Ullman and E. Sali. Object classification using a fragment-based representation. *Biologically Motivated Computer Vision, First IEEE International Workshop, BMVC 2000*, 1811:73–87, 2000.

[62] G. Valentini and T. Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. 2004.

[63] L. J. van't Veer, H. Dai, M. J. van de Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend. Expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536, 2002.

[64] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition*, 2001.

[65] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. In *NIPS*, 2001.

[66] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[67] L. Wolf and I. Martin. Robust boosting for learning from few examples. *Computer Vision and Pattern Recognition*, 2005.