Algorithm independant machine learning

Ganesh, Owen, Zach

No Best Classifier

Thus we measure the "match" and the "alignment" of the learning algorithm.

Bias is used to measure how well the learning algorithm matches the problem.

Variance is used to measure the precision of the match.

High bias means poor match.

High variance means weak match.

Bias and Varaince seen through Regression

let $F(x)$ be the true function with noise that is unknown

let $F(\cdot)$ be the estimate based on n samples in a set D generated by $F(x)$

let $g(x; D)$ be the regression function

How dependent is the regression function $g(x; D)$ on the training set D ?

Mean Square Error

In general the mean square error gives a non-negative value to how much the estimator differs from the function it is try to estimate.

Now lets take the average over all training sets D of a fixed size n, then

$$\varepsilon_D[(g(x;D)-F(x))^2] = (\varepsilon_D[g(x;D)-F(x)])^2 + \varepsilon_D[(g(x;D)-\varepsilon_D[g(x;D)]]^2$$

where

$$\text{Bias}^2 = (\varepsilon_D[g(x;D)-F(x)])^2$$

$$\text{Variance} = \varepsilon_D[(g(x;D)-\varepsilon_D[g(x;D)]]^2$$

Bias and Variance

The Bias is the difference between the expected value and the true value. Low bias implies that the estimate is a good match to F from D.

The Variance shows how much the estimate changes as the training set varies. Low variance means that the estimate F does not change much as we very the training set varies.

The equation as the sum of the bias and variance is similar to a physics conservation law for classifiers.

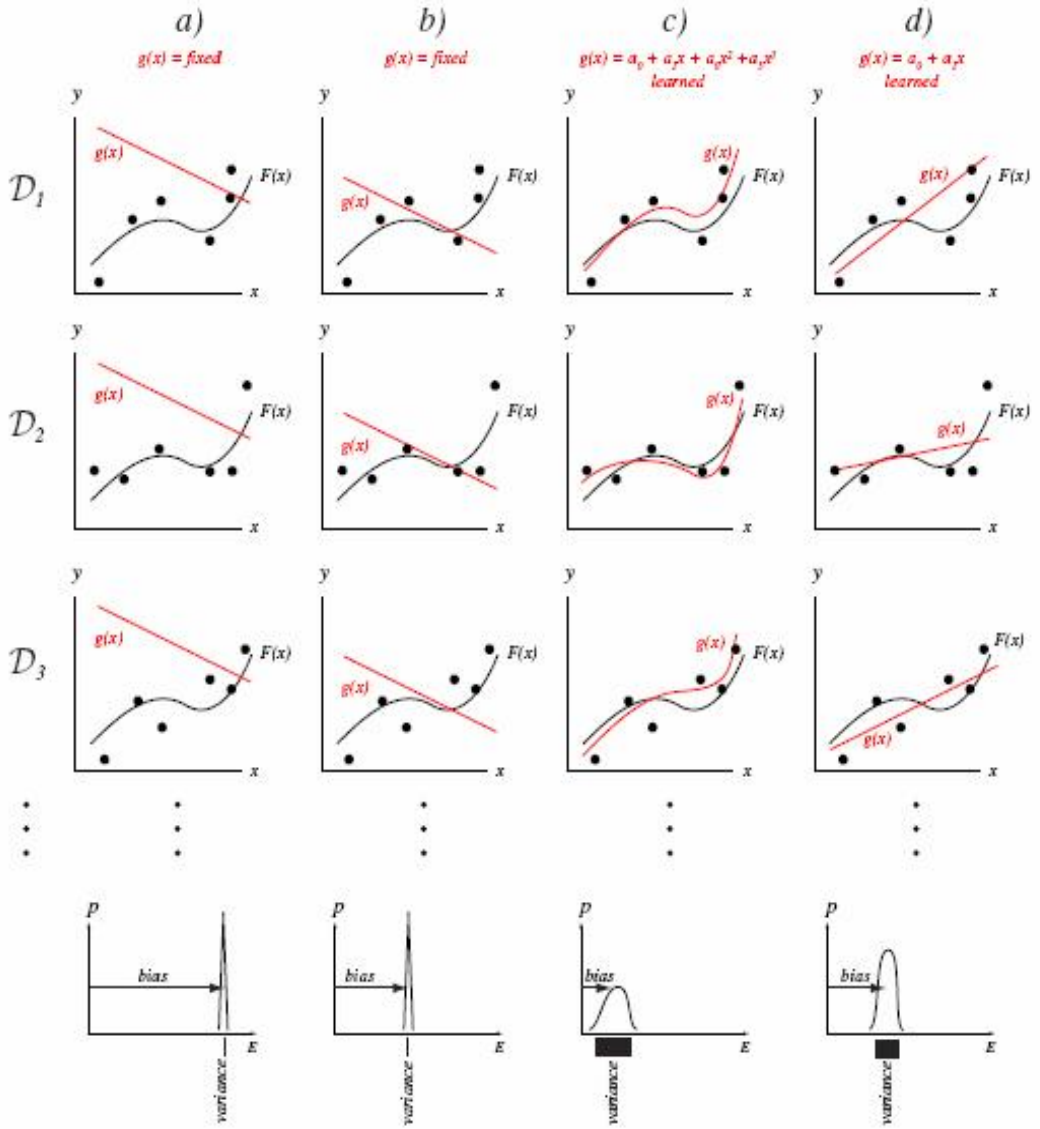

a)
g(x) = fixed
b)
g(x) = fixed
c)
g(x) = a0 + a1x + a2x2 +a3x3
learned
d)
g(x) = a0 + a1x
learned
D1
D2
D3
y
x
g(x)
F(x)
p
bias
variance
E

FIGURE 9.4. The bias-variance dilemma can be illustrated in the domain of regression. Each column represents a different model, and each row represents a different set of $n = 6$ training points, $\mathcal{D}_i$, randomly sampled from the true function $F(x)$ with noise. Probability functions of the mean-square error of $E = \mathcal{E}_{\mathcal{D}}[(g(x) - F(x))^2]$ of Eq. 11 are shown at the bottom. Column a shows a very poor model: a linear $g(x)$ whose parameters are held fixed, *independent* of the training data. This model has high bias and zero variance. Column b shows a somewhat better model, though it too is held fixed, independent of the training data. It has a lower bias than in column a and has the same zero variance. Column c shows a cubic model, where the parameters are trained to best fit the training samples in a mean-square-error sense. This model has low bias and a moderate variance. Column d shows a linear model that is adjusted to fit each training set; this model has intermediate bias and variance. If these models were instead trained with a very large number $n \to \infty$ of points, the bias in column c would approach a small value (which depends upon the noise), while the bias in column d would not; the variance of all models would approach zero. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

Column a is a poor estimate. The spike represents a very high error. This leads to the very high bias.

Column b is a bit better, but still not good.

Column c is much better. It has a lower bias but a larger variance. This is because of the conservation law between the two, but overall this is the best match of the four.

Column d is ok, it is a linear match to a cubic which is not correct, but it is better than a and b.

Bias and Variance for Classification

The slides previous were to understand why and how bias and variance can be used to give a value to well and estimates what it is estimating. Now though, we turn to using bias and variance in classification.

Consider the two category case. We can then let the discriminant function have values 0 or +1.

$$F(x) = Pr[y = 1|x] = 1 - Pr[y = 0|x]$$

It is possible with this case to have a high classification with a poor mean-square error fit. This is because we are considering only values 0 and 1. The posterior selected could possibly be only slight larger than the rest. To avoid this we recast the discriminant function.

let that recast be

$$y = F(x) + \epsilon$$

where ϵ is a zero mean, random value, and to make it simple assume it be a centered binomial distribution with variance

$$Var[\epsilon|x] = F(x)(1 - F(x))$$

Then we can express the function to be estimated by

$$F(x) = \varepsilon[y|x]$$

The goal now is to find an estimate $g(x; D)$ that minimizes the mean square error.

If we assume equal priors $P(w_1) = P(w_2) = 0.5$ then the Bayes discriminant y_b has threshold $1/2$ and the Bayes decision boundary is the set of points for which $F(x) = 1/2$

For a given training set D if the classification error $Pr[g(x;D) = y]$ averaged over predictions at x agree with the Bayes Discriminant then we have the lowest possible error.

$$Pr[g(x;D) = y] = Pr[y_B(x) \neq y] = min[F(x), 1 - F(x)]$$

If it does not agree, then we have an increased error

$$Pr[g(x;D) = y] = max[F(x), 1 - F(x)]$$

$$Pr[g(x;D) \neq y_B] =$$

$$\Phi[sgn[F(x)-1/2][\varepsilon_D[g(x;D)]-1/2]Var[g(x;D)]^{-1/2}]$$

The product of the first and second terms is called boundary bias.

The third term is the variance.

It is important to take away that this is a non-linear relationship.

This means that the value of the variance depends on the boundary bias and small changes lead to very different outcomes.

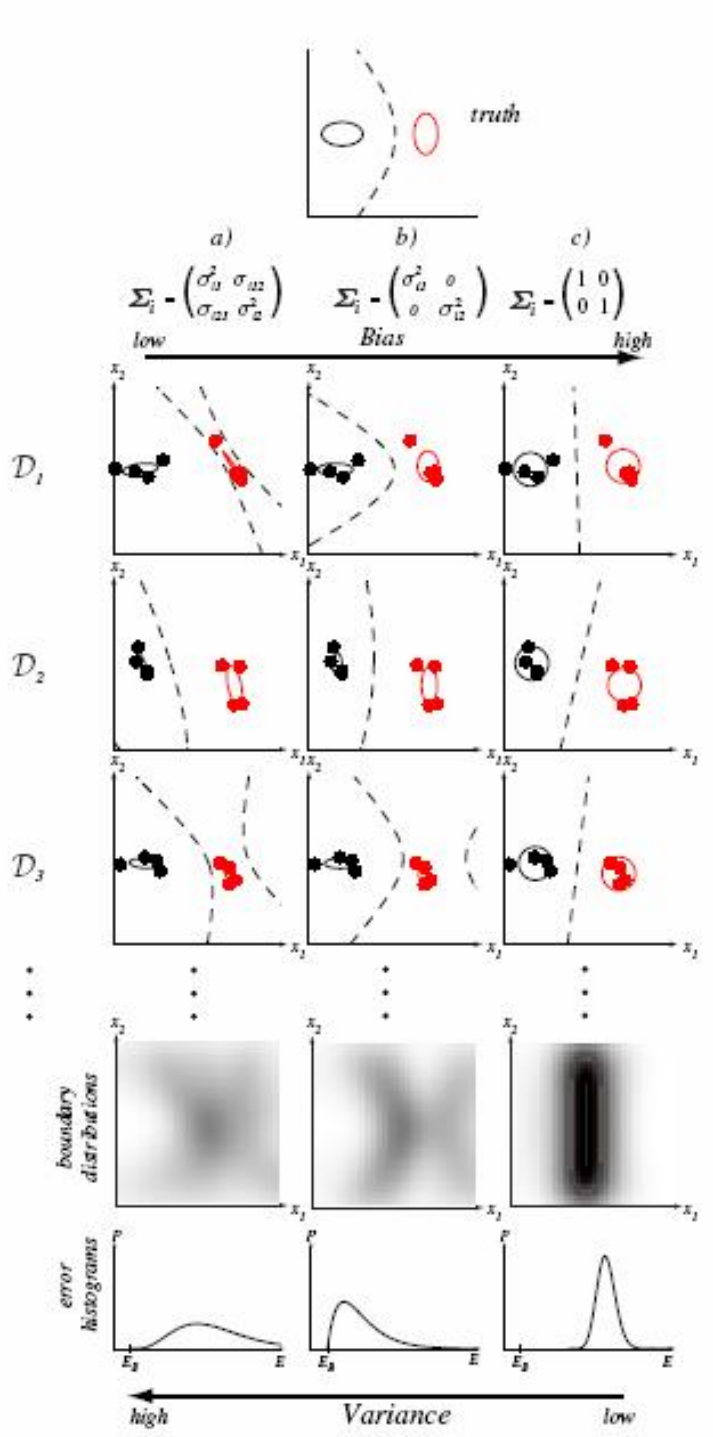

truth
a)
b)
c)
$\Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & \sigma_{i12} \\ \sigma_{i21} & \sigma_{i2}^2 \end{pmatrix}$
$\Sigma_i = \begin{pmatrix} \sigma_{i1}^2 & 0 \\ 0 & \sigma_{i2}^2 \end{pmatrix}$
$\Sigma_i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
low
Bias
high
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_3$
boundary distributions
error histograms
high
Variance
low

FIGURE 9.5. The (boundary) bias-variance trade-off in classification is illustrated with a two-dimensional Gaussian problem. The figure at the top shows the true distributions and the Bayes decision boundary. The nine figures in the middle show different learned decision boundaries. Each row corresponds to a different training set of $n = 8$ points selected randomly from the true distributions and labeled according to the true decision boundary. Column a shows case of a Gaussian model with fully general covariance matrices trained by maximum-likelihood. The learned boundaries differ significantly from one data set to the next; this learning algorithm has high variance. Column b shows the decision boundaries resulting from fitting a Gaussian model with diagonal covariances; in this case the decision boundaries vary less from one row to another. This learning algorithm has a lower variance than the one at the left. Finally, column c shows decision boundaries learning by fitting a Gaussian model with unit covariances (i.e., a linear model); notice that the decision boundaries are nearly identical from one data set to the next. This algorithm has low variance. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

Jackknifing consists of taking the average of a series of versions of the set with a sample removed in order to estimate some information about the distribution of the set.
The mean function we are used to:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Now we will modify it to use jackknifing. We will begin by calculating the mean while leaving one value out:
NOTE: I'm using j as the value to be left out, and leaving i as the value used within the sum because it is less confusing (in my opinion); the book has most of these functions the other way around, so watch that.

$$\mu_{(j)} = \frac{1}{n-1} \sum_{i \neq j}^{n} x_i$$

As you can see, it is possible to use jackknifing on any technique you would ordinarily use on a set of numbers.

Now we will modify it to use jackknifing. We will begin by calculating the mean while leaving one value out:
NOTE: I'm using j as the value to be left out, and leaving i as the value used within the sum because it is less confusing (in my opinion); the book has most of these functions the other way around, so watch that.

$$\mu_{(j)} = \frac{1}{n-1} \sum_{i \neq j}^{n} x_i$$

As you can see, it is possible to use jackknifing on any technique you would ordinarily use on a set of numbers.

Bias is the distance from the target that the error is; in other words, Bias is accuracy. Variance on the other hand is a measure of the range of the results; in other words, Variance is precision.
To begin with, it is possible to generalize the leave-one-out operation. Instead of using $\mu_{(j)}$, we can define $\theta_{(j)}$ as being $\hat{\theta}$on all values of x except j; we can then define $\theta_{(.)}$ as the mean of all values of θ_j. We can now use these generalized forms of the jackknifing process for pattern recognition functions.
it's possible to use jackknifing in order to identify the bias of a function.

$$bias = \theta - \varepsilon[\hat{\theta}]$$

using the jackknife method on theta we get:

$$bias_{(.)} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta})$$

rearranging this we get the estimate of the jackknife method on $\hat{\theta}$:

$$\tilde{\theta} = \hat{\theta} - bias_{(.)} = n\hat{\theta} - (n-1)\hat{\theta_{(.)}}$$

It is also possible to use $\theta_{(i)}$in order to compute an estimate of the variance of the function, as we did before:

$$\sigma^2_{(.)}(\hat{\theta}) = \frac{(n-1)}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2$$

Now I will show an example, it diverges slightly from the book, in order to make the set easier to understand, but can be contrasted with the books example; in this case, we are generating the mode of the following dataset:

$$\{0, 0, 0, 10, 10\}, n = 5$$

$$\hat{\theta}_{(.)} = \frac{1}{5}(5 + 5 + 5 + 0 + 0) = 3$$

these both use the same method for deciding ties, when there are ties for the mode it averaged the two for the mode of that set. Now we'll go back to my example, which has to use the same principle from earlier to calculate bias:

$$bias_{(.)} = 4(3 - 0) = 12$$

Next we calculate the variance, which is where my example is a little simpler:

$$\sigma^2_{(.)}(\hat{\theta}) = \frac{4}{5}[3((5 - 3)^2) + 2((0 - 3)^2)] = 24$$

the example from the book went as follows:

$$\{0, 10, 10, 10, 20, 20\}, n = 6$$

$$\hat{\theta}_{(.)} = \frac{1}{6}(10 + 15 + 15 + 15 + 10 + 10) = 12.5$$

as you can see, we get a bias of 12 and a variance of ~4.9, which can be compared to the books examples of 12.5 and ~5.6 for their values.

The Jackknife technique does give us a nice estimate for the mean and variance of a function, but it is extremely time consuming; which is where the bootstrap method comes in. The bootstrap method takes n random samples from the data set (where n is the length of the set), and repeats it T times where T is chosen by the user. It then averages the T estimates (represented by $\hat{\theta}^{*(i)}$) to get $\hat{\theta}^{*(.)}$(or the bootstrap estimate):

$$\hat{\theta}^{*(.)} = \frac{1}{T}\sum_{i=1}^{T}\hat{\theta}^{*(i)}$$

This estimate can be scaled based upon the level of computing power available to you, as the higher the T, the better the estimate; as $T \rightarrow \infty$(the number of samples becomes unreasonably large), $\hat{\theta}^{*(.)} = \hat{\theta}$.
Now with $\hat{\theta}^{*(.)}$, which corresponds with $\hat{\theta}_{(.)}$, we can create the bootstrap bias and variance

$$bias^{*(.)} = \hat{\theta}^{*(.)} - \hat{\theta}$$

$$\sigma^{2*(.)}[\theta] = \frac{1}{T}\sum_{i=1}^{T}(\hat{\theta}^{*(i)} - \hat{\theta}^{*(.)})^2$$

Arcing, Bagging, and Boosting

Arcing (Adaptive reweighting and combining): reusing/reselecting data to improve classification.

Example arcing procedure: Bagging (boostrap aggregation)

- Use multiple versions of training set, where each set $\mathcal{D}_i$ is drawn from a random $n_i < n$ patterns in the original set $\mathcal{D}$.
- Each of the $\mathcal{D}_i$ bootstrapped sample sets is used to train a component classifier, C_i.
- Classification is based on a simple vote of the i classifiers.

Why Perform Bagging?

Unstable classifiers are those for which a small change in the training set produces a large change in accuracy.

Bagging typically improves unstable classifiers.

Intuitively, bagging averages over differences in the training set.

However, there's no formal proof of this.

Boosting

Boosting - use the most informative subset of the training set $\mathcal{D}$ to train classifiers.

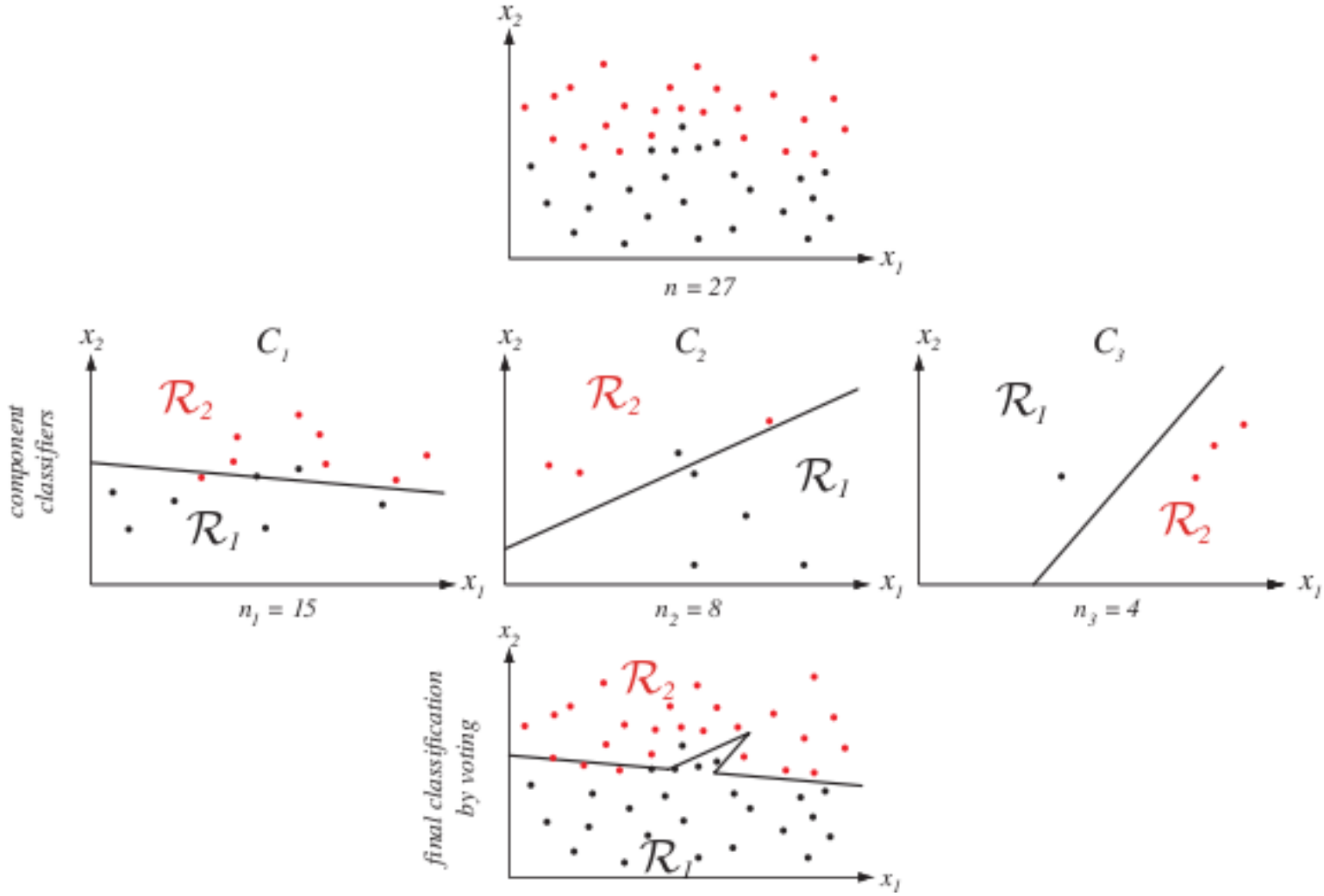
Example of boosting: train 3 classifiers for a 2 class problem using training set $\mathcal{D}$

- Select $n_1 < n$ patterns randomly from $\mathcal{D}$ to create $\mathcal{D}_i$.
- Train a classifier C_1 using $\mathcal{D}_1$. C_1 needs to be at worst a weak learner (at least as good as random chance).
- Select $\mathcal{D}_2$, made up of the most informative patterns of $\mathcal{D}$ given C_1. Half of the patterns of $\mathcal{D}_2$ should be correctly classified by C_1 and the other half should be misclassified by C_1.

Boosting II

- Train C_2 using $\mathcal{D}_2$.
- Select $\mathcal{D}_3$, made up of those patterns that C_1 and C_2 disagree on.
- Train C_3 using $\mathcal{D}_3$.
- Classify a test pattern $\mathbf{x}$ as $h_1(\mathbf{x})$ if $h_1(\mathbf{x}) = h_2(\mathbf{x})$, otherwise, classify as $h_3(\mathbf{x})$.

Boosting III



Boosting IV - Choosing n

How do we choose n_1? Our goal is $n_1 \approx n_2 \approx n_3 \approx \frac{n}{3}$.

Problem: if classification is easy, then C_1 will do very well and there won't be a lot of patterns left for n_2. The opposite holds if classification is hard.

Solution: Without prior knowledge, attempt boosting several times until we pick an acceptable n_1.

We can also apply boosting recursively on the component classifiers; for our example, applying boosting i times gives 3^i component classifiers.

AdaBoost I

AdaBoost (adaptive boosting) trains k_{max} classifiers, where k_{max} is set by the user.

Basic idea: focus on using most informative/hard patterns for training classifiers.

We give each sample $\mathbf{x}_i \in \mathcal{D}$ a probability for each iteration of the k iterations of the algorithm: $W_k(i)$. $W_1(i) = \frac{1}{n}$ for all i.

At each iteration of the algorithm, we train a classifier C_k using a $\mathcal{D}_k \subset \mathcal{D}$, where patterns are selected according to $W_k(i)$.

For every $\mathbf{x}_i$, we update $W_k(i)$, increasing it if C_k misclassifies $\mathbf{x}_i$ and decreasing it otherwise.

AdaBoost II - Algorithm

ADA-BOOST $(\mathcal{D} = \{\mathbf{x}^1, y_1, ...\mathbf{x}^n, y_n\},\ k_{max},\ W_1(i) = \frac{1}{n})$

for $k = 1..k_{max}$

train C_k using $\mathcal{D}$ sampled according to $W_k(i)$

let $E_k =$ the training error of C_k measured on $\mathcal{D}$

let $\alpha_k = \frac{1}{2}\ln[(1 - E_k)/E_k)]$

for $i = 1..n$

let $W_{k+1}(i) = \frac{W_k(i)}{Z_k} e^{\pm\alpha_k}$

where we use $-\alpha_k$ if $h_k(\mathbf{x}^i) = y_i$, $+\alpha_k$ otherwise

return C_k and α_k for $k = 1..k_{max}$

AdaBoost III - Classification/Error Rate

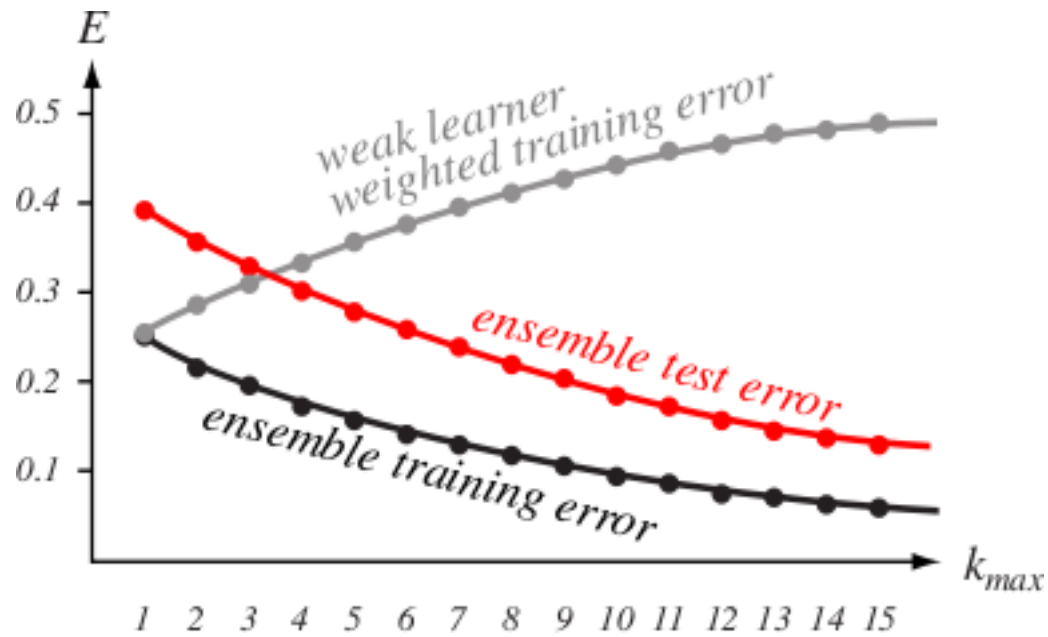
Classification decision: we create a discriminant function, $g(\mathbf{x}) = \sum_{k=1}^{k_{max}} \alpha_k h_k(\mathbf{x})$. Our decision is merely $\text{sgn}(g(\mathbf{x}))$.

The training error rate can be made arbitrarily low by choosing a sufficiently large k_{max}, as long as each C_k is a weak learner. A weak learner has an error rate of less than $\frac{1}{2}$, so we can write our error rate of C_k, $E_k = \frac{1}{2} - G_k$ for some $\frac{1}{2} \geq G_k > 0$.

Then our ensemble error rate,

$$E = \prod_{k=1}^{k_{max}} 2\sqrt{E_k(1-E_k)} = \prod_{k=1}^{k_{max}} \sqrt{1-4G_k^2} \leq \exp(-2\sum_{k=1}^{k_{max}} G_k^2)$$

AdaBoost IV - Error Rate



AdaBoost V - Observations

Note that we are minimizing training error using AdaBoost, not our error rate in general.

Fortunately, choosing a large k_{max} does not typically lead to overfitting.

Boosting only works if our classifiers are weak learners.

Queries and the Oracle

Suppose our training set is unlabeled, but we have an oracle that we can give query patterns to and receive the correct label. However, this process may be expensive.

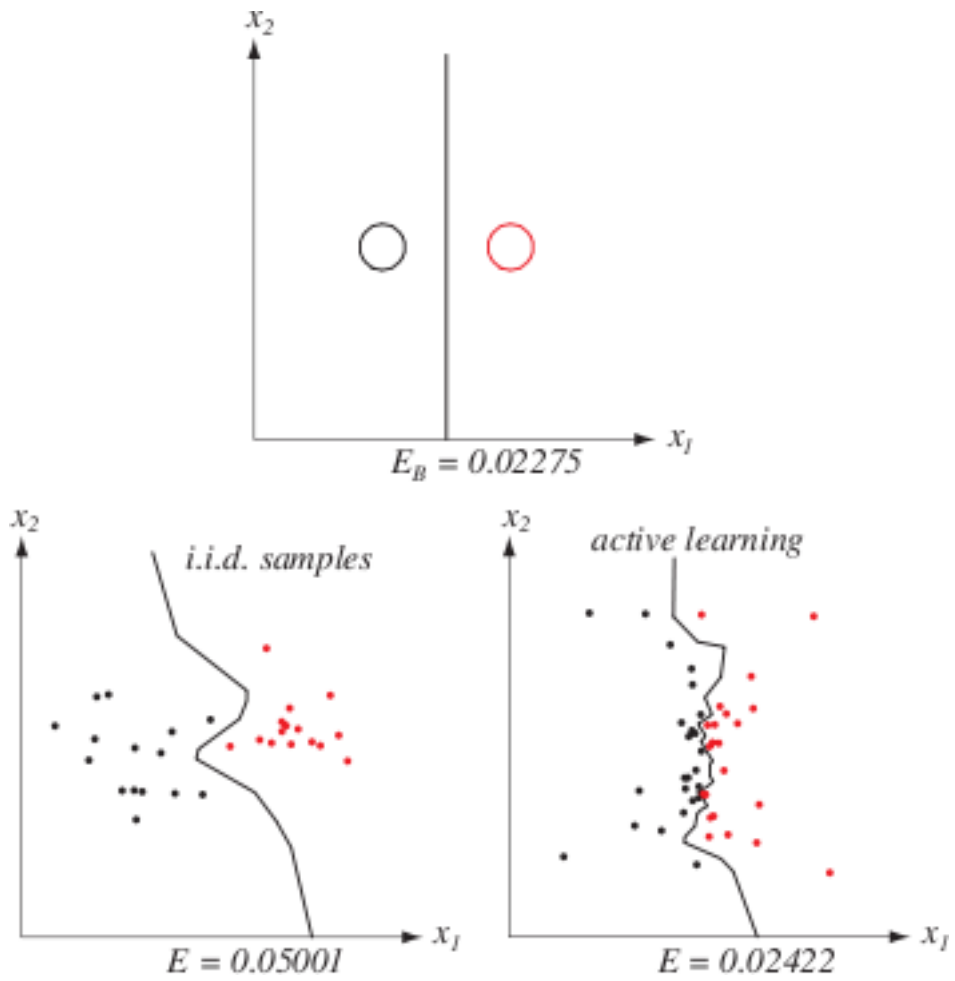
The idea is the same as before: we want to focus on the most informative training patterns.

We start with a weak classifier C_0 that's been trained on a small, random subset of patterns labeled by the oracle.

- Confidence based: if $g_i(\mathbf{x}) \approx g_j(\mathbf{x})$, then $\mathbf{x}$ is informative.

- Voting based: if we have multiple classifiers and there is disagreement between them on the class of $\mathbf{x}$, then $\mathbf{x}$ is informative.

Queries and the Oracle Example



Customer Churn

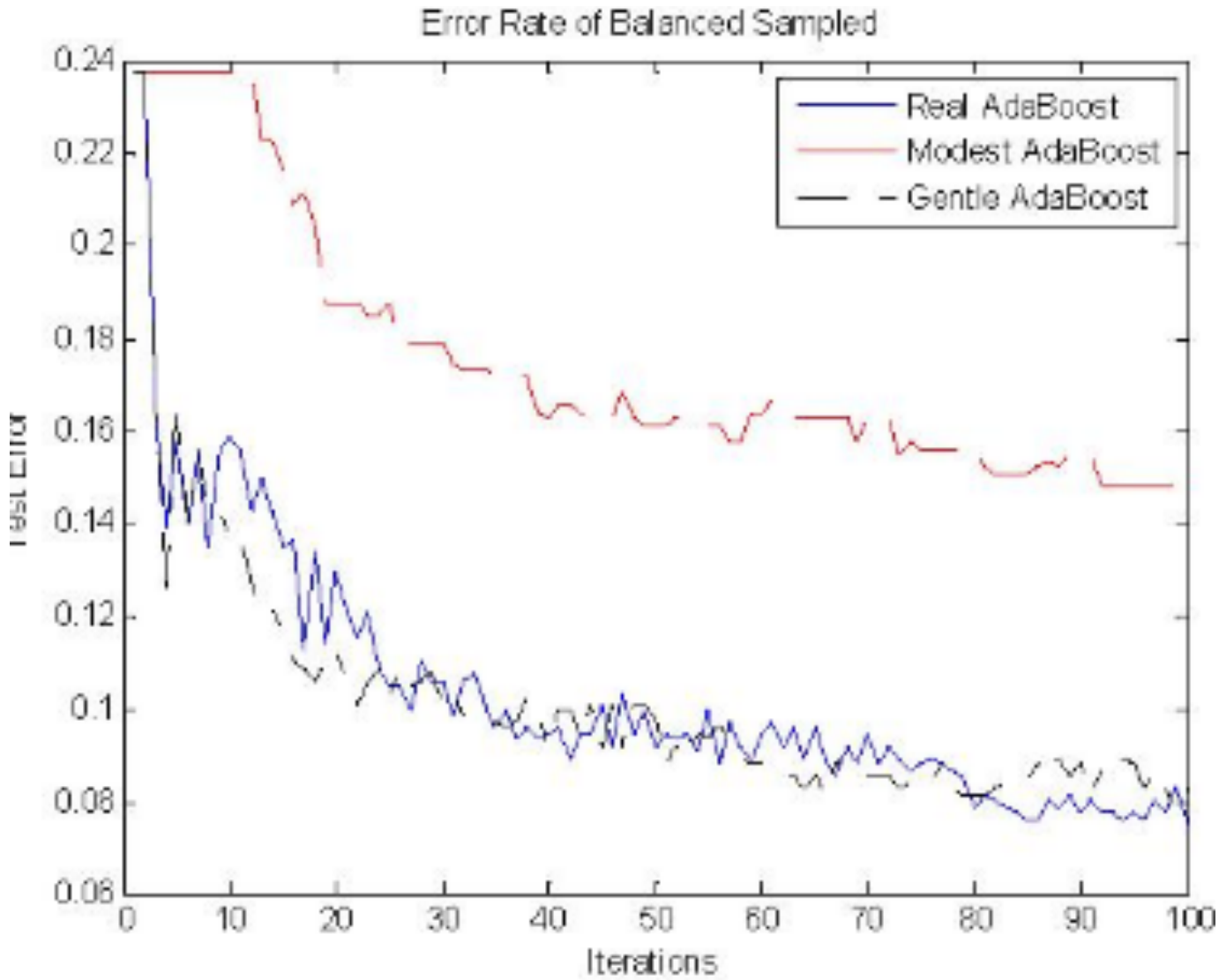
Used an ensemble of stump classifiers

Trained using three variations on AdaBoost, each with slightly different update rules.

- Real AdaBoost - AdaBoost presented previously
- Gentle AdaBoost - Better performance on noisy data
- Modest AdaBoost - Resistent to overfitting, but slower decreases in training error

$k_{max} = 100$

Customer Churn Error Rates



References

Duda, Richard, Hart, Peter, and Stork, David. Pattern Classification. New York: John Wily & Sons, 2001.

Shao Jinbo, Li Xiu, Liu Wenhuang. **The Application of AdaBoost in Customer Churn Prediction. In** *Service Systems and Service Management, 2007 International Conference on,* pages 1-6, June 2007.