Lecture 12

Oct 24th 2008

Review

- Linear SVM seeks to find a linear decision boundary that maximizes the geometric margin
- Using the concept of soft margin, we can achieve tradeoff between maximizing the margin and faithfully fitting all training examples, thus provides better handling of outliers/noisy training examples and (simple) cases that are not linearly separable
- By mapping the data from the original input space into a higher dimensional feature space, we obtain non-linear SVM
  - Let's see a bit more about this

Non-linear SVM

- The basic idea is to map the data onto a new feature space such that the data is linearly separable in this space
- However, we don't need to explicitly compute the mapping. Instead, we use what we call the kernel function (this is referred to as **the kernel trick**)
- What is a kernel function?
  - A kernel function $\boldsymbol{k}(a,b) = \langle \phi(\mathbf{a}) \cdot \phi(\mathbf{b}) \rangle$
  - ϕ is a mapping
- Why using kernel function?
  - The inner product in the mapped feature space $\langle \phi(\mathbf{a}) \cdot \phi(\mathbf{b}) \rangle$ is computed using the kernel function applied to the original input vectors $\boldsymbol{k}(a,b)$
  - This avoids the computational cost incurred by the mapping
- We have shown previously that using kernel in prediction stage is straight forward, but what about learning of **w** (or more directly the α_i's)?
- Does this make any difference in the SVM learning step?
  - Let's look at the optimization problem of the original SVM

Soft margin SVM optimization problem

$$\min_{\mathbf{w},b} \|\mathbf{w}\|^2 + c\sum_{i=1}^{N} \xi_i$$

$$\text{subject to: } y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1-\xi_i, \quad i = 1,\cdots,N$$

$$\xi_i \geq 0, \quad i = 1,\cdots,N$$

- Knowing that $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y^i \mathbf{x}^i$, we can rewrite the optimization problem in terms of inner products

$$\min_{\mathbf{w},b} \sum_{i,j} \alpha_i \alpha_j y^i y^j < \mathbf{x}^i \cdot \mathbf{x}^j > + c\sum_{i=1}^{N} \xi_i$$

$$\text{subject to: } y^i(\sum_{j=1}^{N} \alpha_j y^j < \mathbf{x}^j \cdot \mathbf{x}^i > + b) \geq 1-\xi_i, \quad i = 1,\cdots,N$$

$$\xi_i \geq 0, \quad i = 1,\cdots,N$$

- In fact, learning for SVM is typically carried out using only the inner products of **x**, without using original vector of **x**
- Now we just need to replace the inner products with an appropriate kernel functions

What we have seen so far

- Linear SVM
  - Geometric margin vs. functional margin
  - Problem formulation
- Soft margin SVM
  - Include the slack variable in optimization
  - c controls the trade-off
- Nonlinear SVM using kernel functions
  - Kernel functions

Notes on Applying SVM

- Many SVM implementations are available, and can be found at www.kernel-machine.org/software.html
- Handling multiple class problem with SVM requires transforming a multiclass problem into multiple binary class problems
  - One against rest
  - Pairwise
  - etc

Model selection for SVM

- There are a number of model selection questions when applying SVM
  - Which kernel functions to use?
  - What c parameter to use (soft margin)?
- You can choose to use default options provided by the software, but a more reliable approach is to use cross-validations

Strength vs weakness

- Strengths
  - The solution is globally optimal
  - It scales well with high dimensional data
  - It can handle non-traditional data like strings, trees, instead of the traditional fixed length feature vectors
    - Why? Because as long as we can define a kernel function for such input, we can apply svm
- Weakness
  - Need to specify a good kernel
  - Training time can be long if you use the wrong software package

Table 1. Training time in CPU-seconds

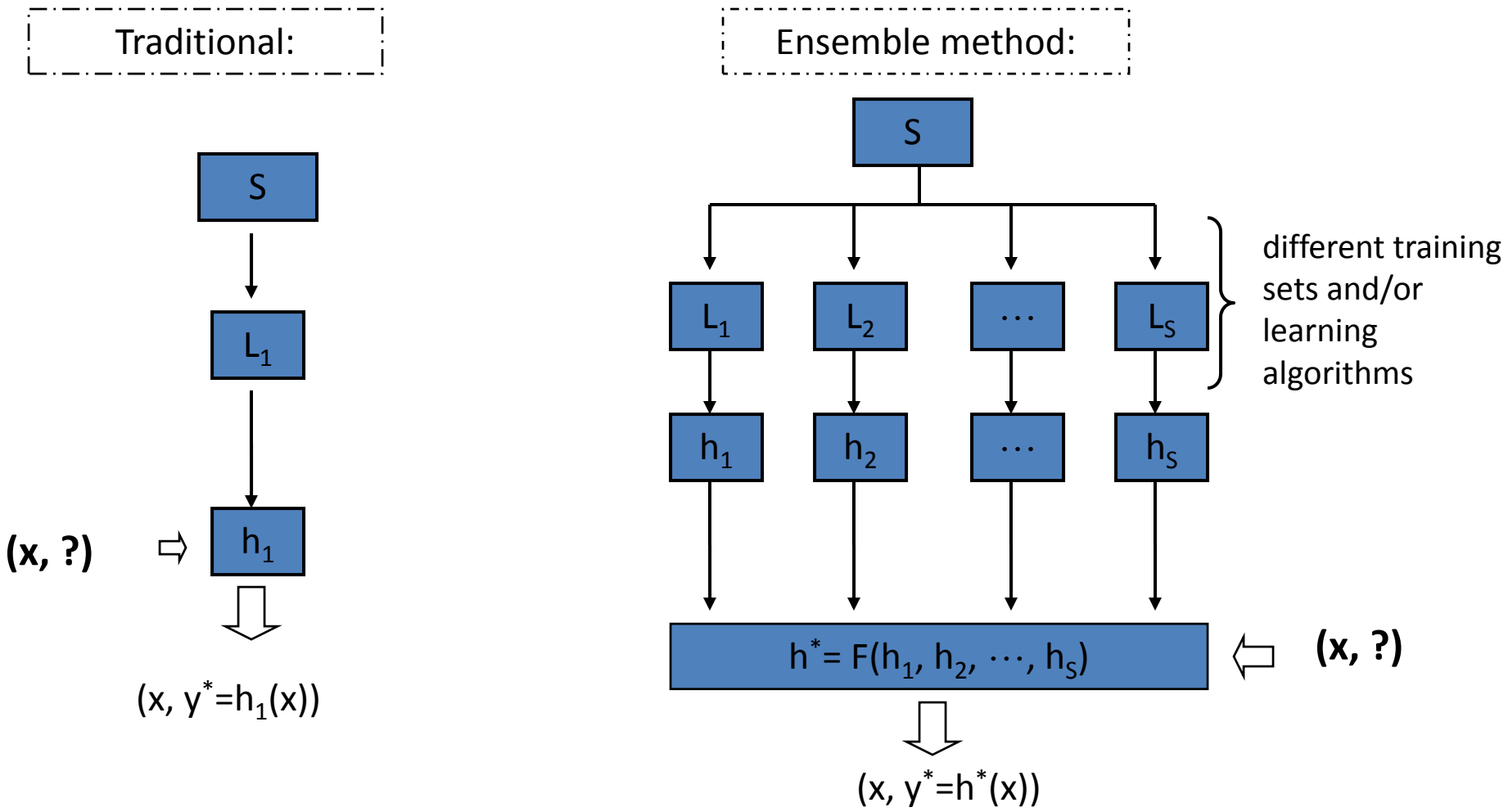
	Pegasos	SVM-Perf	SVM-Light
CCAT	2	77	20,075
Covertype	6	85	25,514
astro-ph	2	5	80
	2007	2006	1998

Ensemble Learning

Ensemble Learning

- So far we have designed learning algorithms that take a training set and output a classifier
- What if we want more accuracy than current algorithms afford?
  - Develop new learning algorithm
  - Improve existing algorithms
- Another approach is to leverage the algorithms we have via ensemble methods
  - Instead of calling an algorithm just once and using its classifier
  - Call algorithm multiple times and combine the multiple classifiers

What is Ensemble Learning



Ensemble Learning

- **INTUITION:** Combining Predictions of multiple classifiers (an ensemble) is more accurate than a single classifier.
- Justification:
  - easy to find quite good "rules of thumb" however hard to find single highly accurate prediction rule.
  - If the training set is small and the hypothesis space is large then there may be many equally accurate classifiers.
  - Hypothesis space does not contain the true function, but it has several good approximations.
  - Exhaustive global search in the hypothesis space is expensive so we can combine the predictions of several locally accurate classifiers.

How to generate ensemble?

- There are a variety of methods developed
- We will look at two of them:
  - Bagging
  - Boosting (Adaboost: adaptive boosting)
- Both of these methods takes a single learning algorithm (we will call this ***the base learner***) and use it multiple times to generate multiple classifiers

Bagging: Bootstrap Aggregation
(Breiman, 1996)

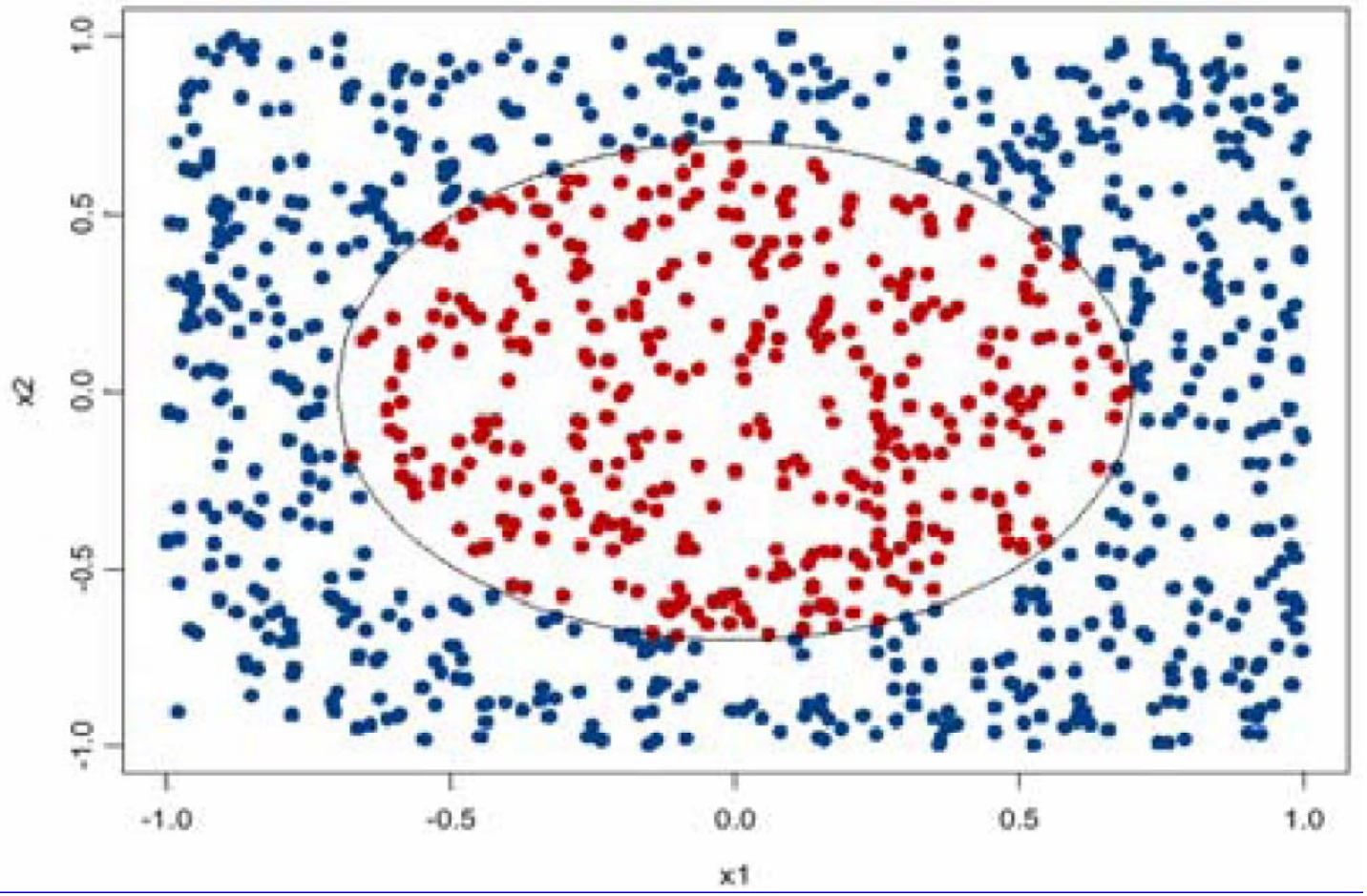
- Generate a **random sample** from training set **S** by a random re-sampling technique called **bootstrapping**
- Repeat this sampling procedure, getting a sequence of **T training sets: S_1,S_2,…,S_T**
- Learn a sequence of classifiers h_1,h_2,…,h_T for each of these training sets, using the same base learner
- To classify an unknown point X, let each classifier predict
  - $h_1(X) = 1$ $h_2(X) = 1$, $h_3(X) = 0$, …, $h_T(X) = 1$,
- Take simple **majority vote** to make the final prediction
  - Predict the class that gets the most vote from all the learned classifers

Bootstrapping

```
S' = {}
For i=1, …, N (N is the total number of points in S)
        draw a random point from S and add it to S'
End
Return S'
```

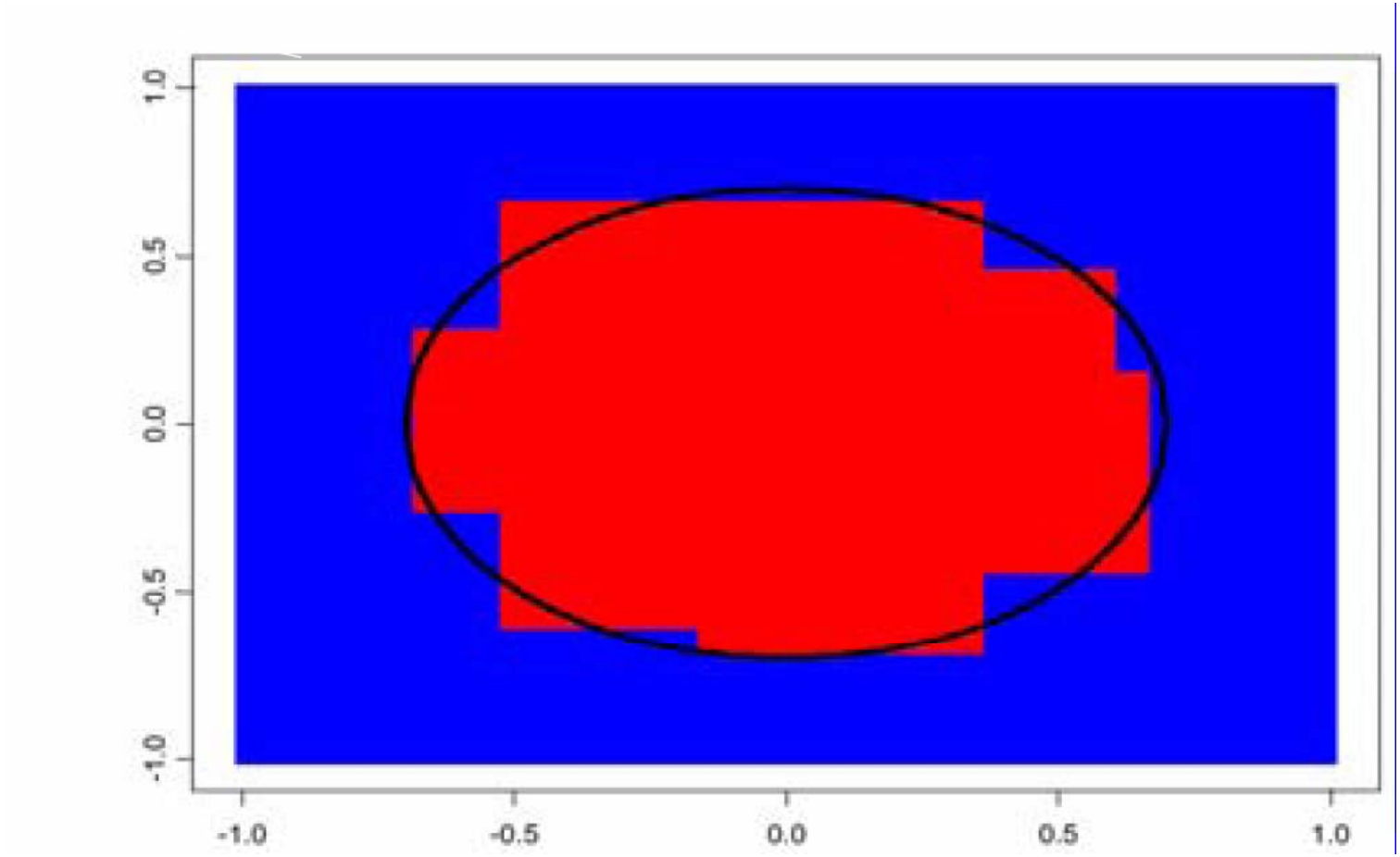
- This is a sampling procedure that samples with replacement
  - Each time a point is drawn, it will not be removed
  - This means that we can have multiple copies of the same data point in my sample
  - New training set contains the same number of points (may contain repeats) as the original training set
  - On average, 66.7% of the original points will appear in a random sample

Bagging Example



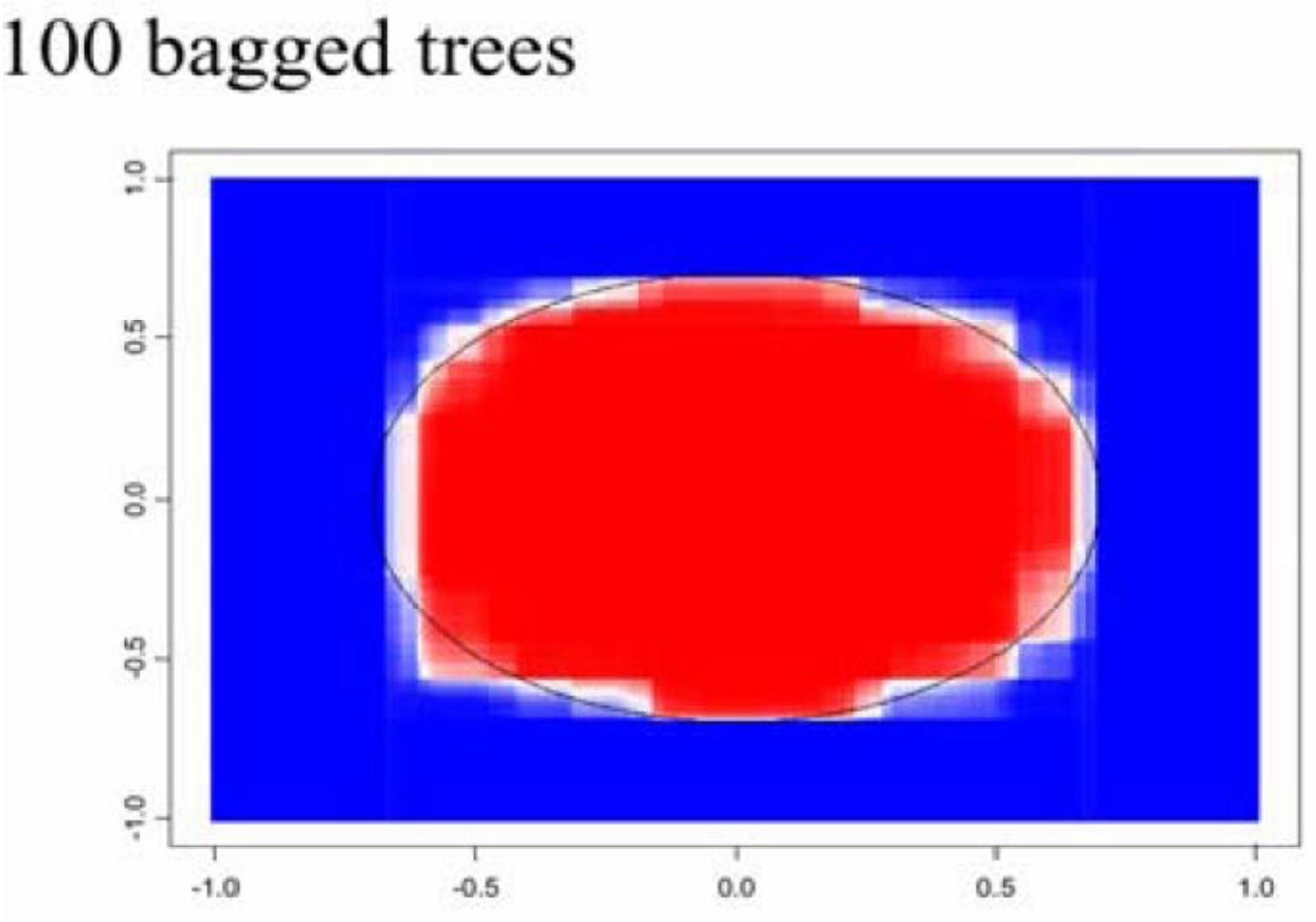
The true decision boundary

Decision Boundary by the CART Decision Tree Algorithm



Note that the decision tree has trouble representing this decision boundary

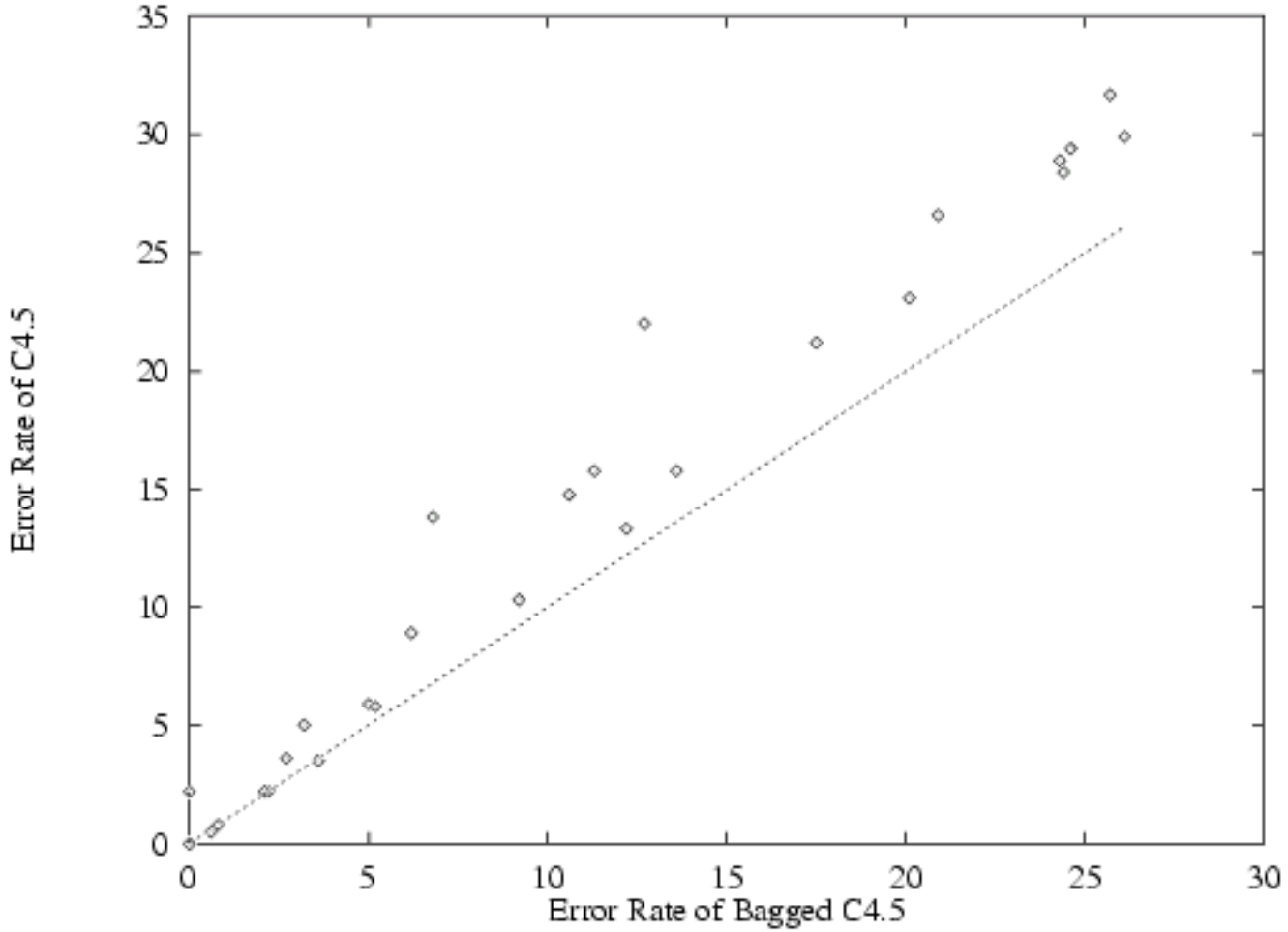
100 bagged trees



By averaging 100 trees, we achieve better approximation of the boundary, together with information regarding how confidence we are about our prediction.

Empirical Results for Bagging Decision Trees

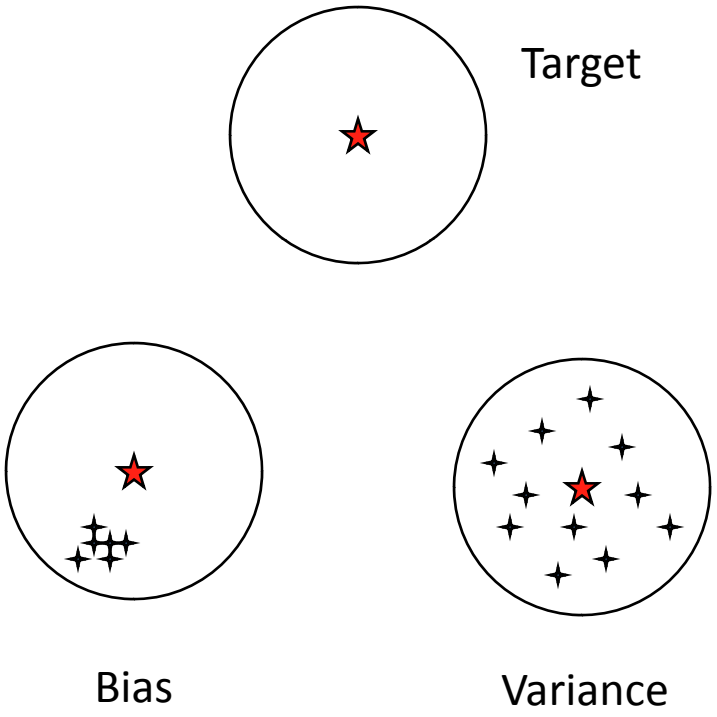
(Freund & Schapire)



Each point represents the results of one data set

Why can bagging improve the classification accuracy?

The Concept of Bias and Variance



Bias/Variance for classifiers

- Bias arises when the classifier cannot represent the true function – that is, the classifier underfits the data
- Variance arises when the classifier overfits the data – minor variations in training set cause the classifier to overfit differently
- Clearly you would like to have a low bias and low variance classifier!
  - Typically, low bias classifiers (overfitting) have high variance
  - high bias classifiers (underfitting) have low variance
  - We have a trade-off

Effect of Algorithm Parameters on Bias and Variance

- k-nearest neighbor: increasing k typically increases bias and reduces variance
- decision trees of depth D: increasing D typically increases variance and reduces bias

Why does bagging work?

- Bagging takes the average of multiple models --- reduces the variance
- This suggests that bagging works the best with low bias and high variance classifiers