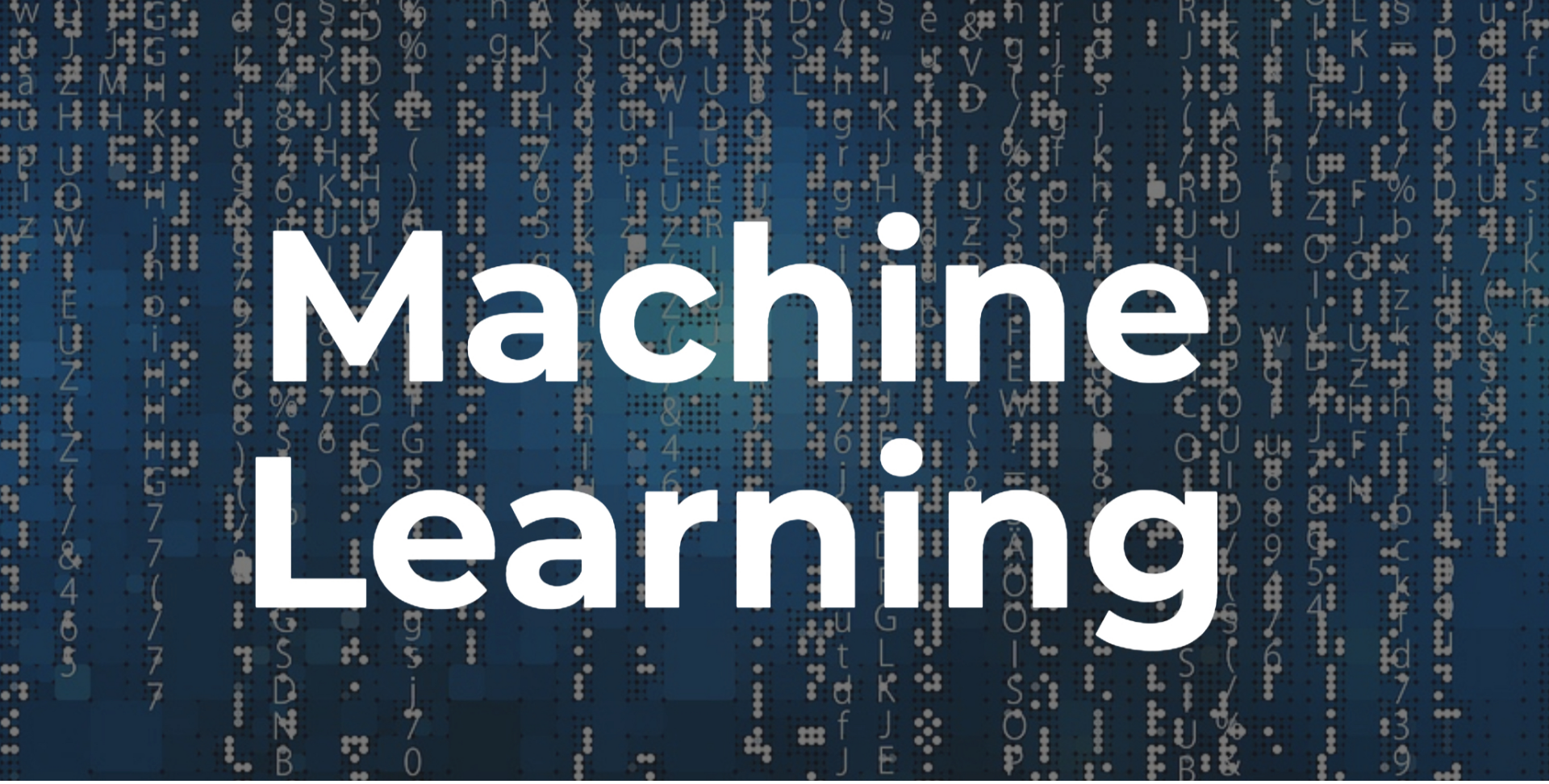


CSCI 5622 Fall 2020

Prof. Claire Monteleoni

Today

- Perceptron convergence
- Decision trees
  - K-d Trees

If time:

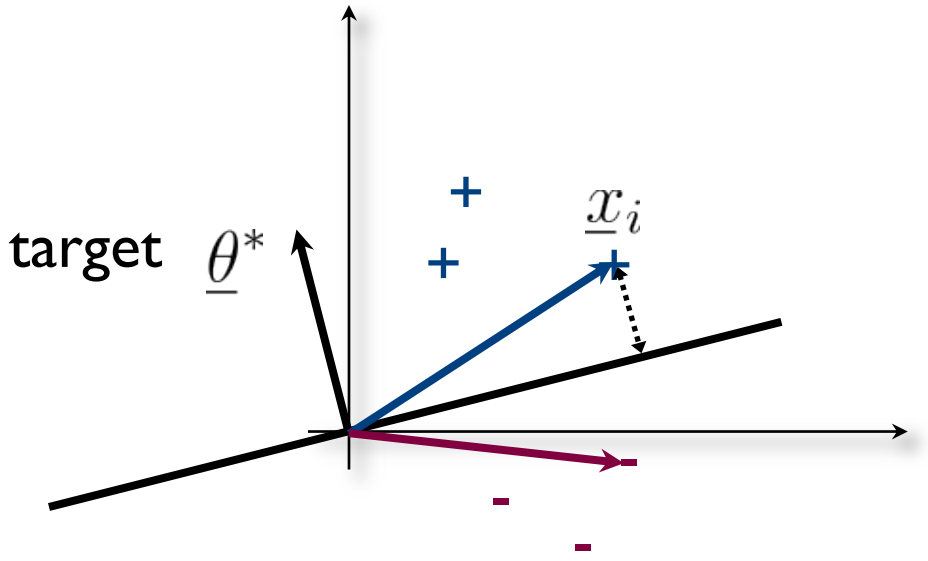
- Overfitting vs. generalization
  - bias-variance tradeoff

- Model evaluation/selection
  - Validation, Cross-Validation

With credit to S. Dasgupta, T. Jaakkola, C. Tan, T. Joachims

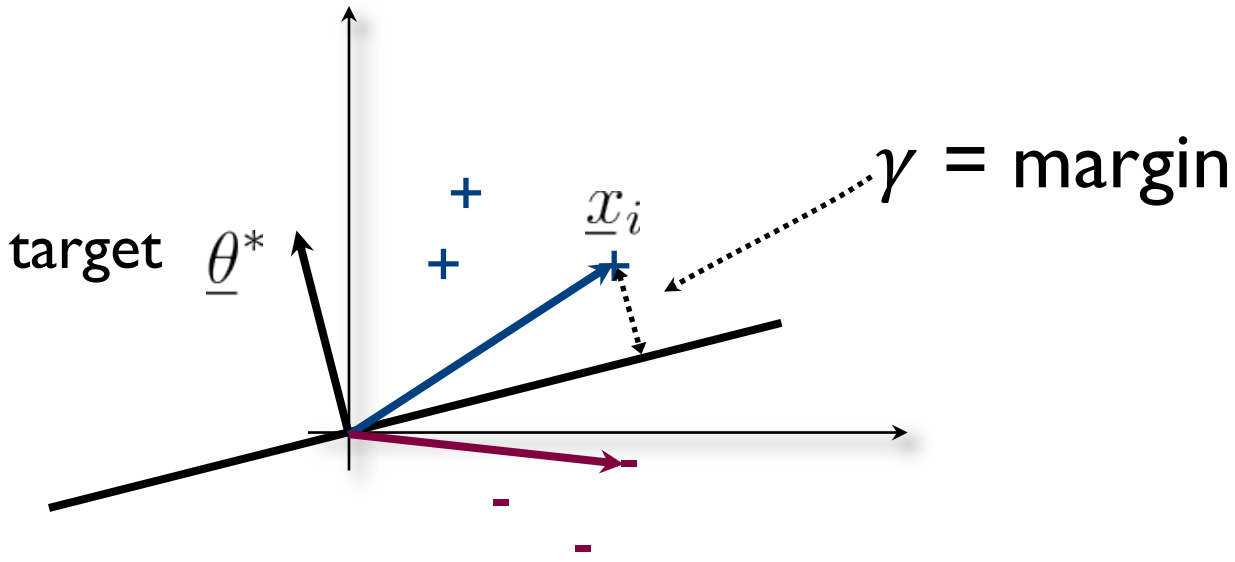
"Margin"

- We can get a handle on convergence by assuming that there exists a target classifier θ^* with good properties
- One such property is margin, i.e., how close the separating boundary is to the points



"Margin"

- We can get a handle on convergence by assuming that there exists a target classifier θ^* with good properties
- One such property is margin, i.e., how close the separating boundary is to the points



Perceptron

Perceptron: If $y_t(v_t \cdot x_t - \tau) < 0$ Filtering rule

$v_{t+1} = v_t + \eta\, y_t\, x_t$ Update step

NOTE: Additive updates. $X=R^d$. Algorithm credited to [Rosenblatt '58].

We will now assume $\eta = 1, \tau = 0$.

Perceptron: If $y_t(v_t \cdot X_t) < 0$ Filtering rule

$v_{t+1} = v_t + y_t\, x_t$ Update step

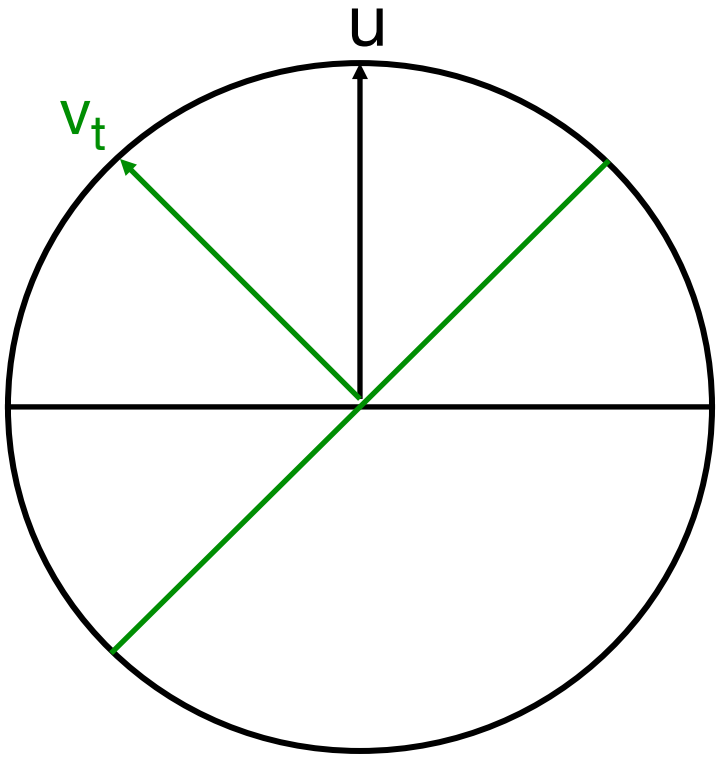
Note: here we use the notation $v = \theta$, $u = \theta^*$

Problem framework

$$x_t \in \mathbb{R}^d,\ \|x\| \le R,\ y_t \in \{-1, +1\}$$

Separability: there exists some perfect classifier, u, such that:

$$u : y_t(u \cdot x_t) > 0 \ \ \forall t, \quad \|u\| = 1$$



Assume u is through origin.

→ Always possible, by increasing dimension by 1.

Current hypothesis: v_t

(Called θ on previous slides)

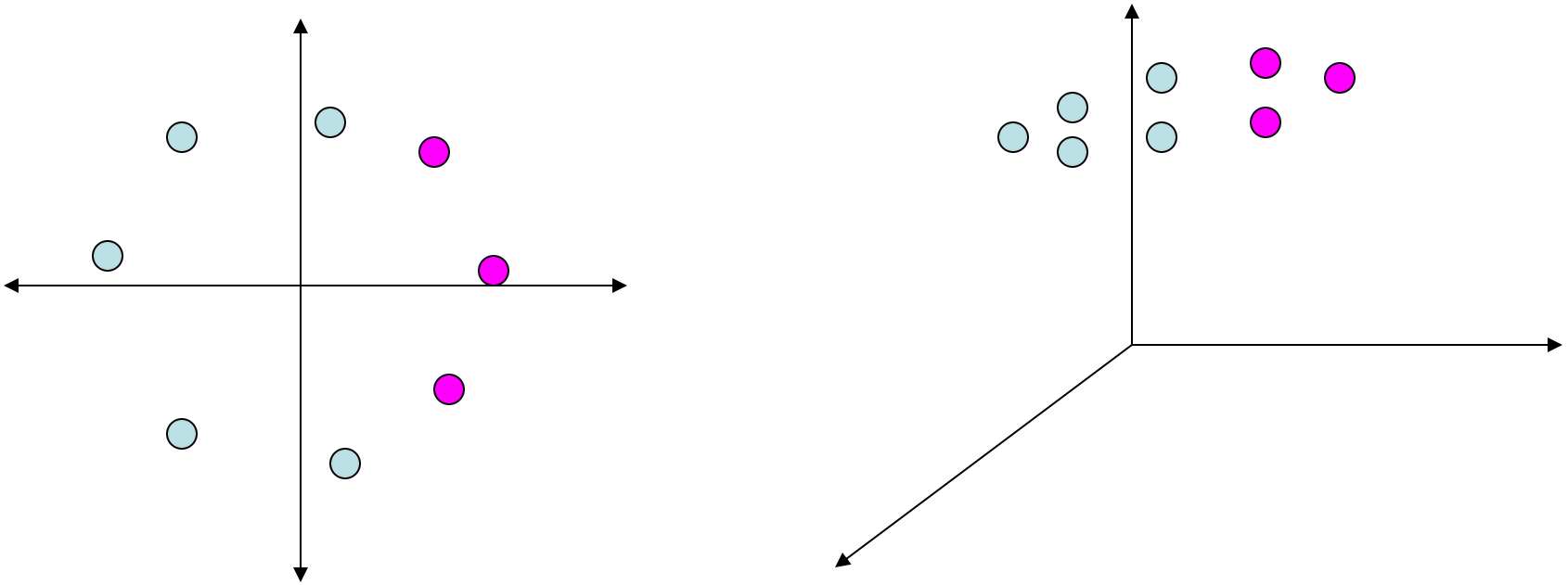
Preprocessing step

Points (x,y), where input space $X = R^d$, label space $Y = \{+1,-1\}$

Add an extra feature to x, and set it to 1:

$x^0 = (x,1)$ in R^{d+1}

Then: points (x,y) linearly separable $\Leftrightarrow$ points (x^0, y) linearly separable by a hyperplane through the origin



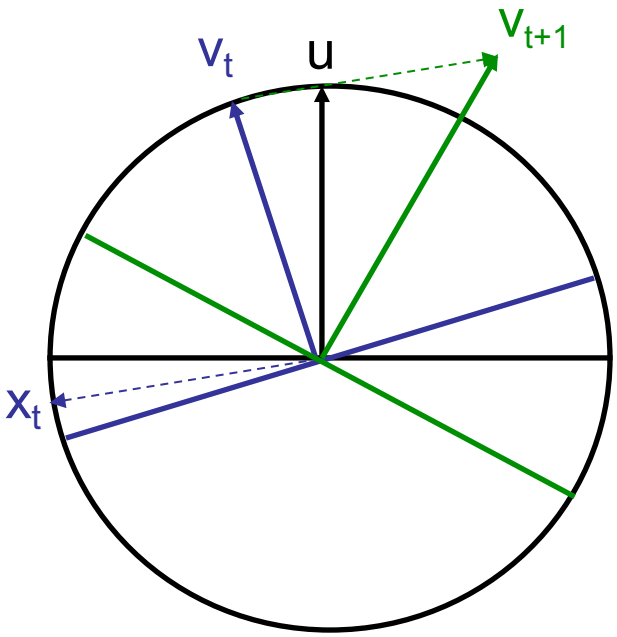
Perceptron

Perceptron: If $y_t(v_t \cdot x_t) < 0$ Filtering rule

$v_{t+1} = v_t + y_t x_t$ Update step

NOTE: Additive updates. Algorithm credited to [Rosenblatt '58].

Example: data is separable by u:



Perceptron

Analyses of standard Perceptron:

Linearly separable (through origin) data, uniform distribution:

- $\tilde{O}(d/\varepsilon^2)$ mistakes (to reach error ε) [Baum '89].
- $\Omega(1/\varepsilon^2)$ mistakes [Dasgupta, Kalai & Monteleoni, '05].
- $\Omega(1/\varepsilon^2)$ labels for active learning [Dasgupta, Kalai & Monteleoni, '05].

Margin assumption: no distribution assumption except linearly separable (through origin), with margin γ

$$y_t(u \cdot x_t) \geq \gamma \text{ for all } t.$$

- $O(1/\gamma^2)$ mistakes to reach zero error [Novikoff '62].

Perceptron analysis with margin

Margin assumption: no distribution assumption except separable (through origin), AND: $y_t(u \cdot x_t) \geq \gamma$ for all t.

- $O(1/\gamma^2)$ mistakes to reach zero error [Novikoff '62].

Proof: Let $\|u\| = 1$. Let (x, y) be a mistake, i.e. $y\,(v_t \cdot x) < 0$, $\|x\| \leq R$.

Lemma 1: $u \cdot v_{t+1} \geq u \cdot v_t + \gamma$.

Proof: $u \cdot v_{t+1} = u \cdot (v_t + y\,x) = u \cdot v_t + y\,(u \cdot x) \geq u \cdot v_t + \gamma$

(by definition of margin, γ).

Lemma 2: $\|v_{t+1}\|^2 \leq \|v_t\|^2 + R^2$

Proof: $\|v_{t+1}\|^2 = \|v_t + y\,x\|^2 = \|v_t\|^2 + 2y(v_t \cdot x) + \|x\|^2$

$\leq \|v_t\|^2 + 2y(v_t \cdot x) + R^2$

$< \|v_t\|^2 + R^2$

because v_t makes a mistake on (x, y), i.e. $2y(v_t \cdot x) < 0$.

Perceptron analysis with margin

Proof continued:

Let $\|u\| = 1$. Let (x, y) be a mistake, i.e. $y\,(v_t \cdot x) < 0$, $\|x\| \le R$.

Lemma 1: $u \cdot v_{t+1} \ge u \cdot v_t + \gamma$.

Lemma 2: $\|v_{t+1}\|^2 \le \|v_t\|^2 + R^2$.

Finally, after M mistakes:

a. $u \cdot v_{M+1} \ge M\,\gamma$, by Lemma 1 (expanding the recurrence).

b. $\|v_{M+1}\|^2 \le M\,R^2$, by Lemma 2. So $\|v_{M+1}\| \le M^{1/2}\,R$.

Since u is a unit vector, $u \cdot v_t \le \|v_t\|$ by Cauchy-Schwartz: $|u \cdot v| \le \|u\|\,\|v\|$

So, $u \cdot v_{M+1} \le \|v_{M+1}\|$.

Using a. and b. for LHS and RHS respectively,

$$M\,\gamma \le u \cdot v_{M+1} \le \|v_{M+1}\| \le M^{1/2}\,R$$

$$M\,\gamma \le M^{1/2}\,R$$

$M^{1/2} \le R / \gamma$, and $M \le (R / \gamma)^2$ □

Fisher's IRIS data

Four features

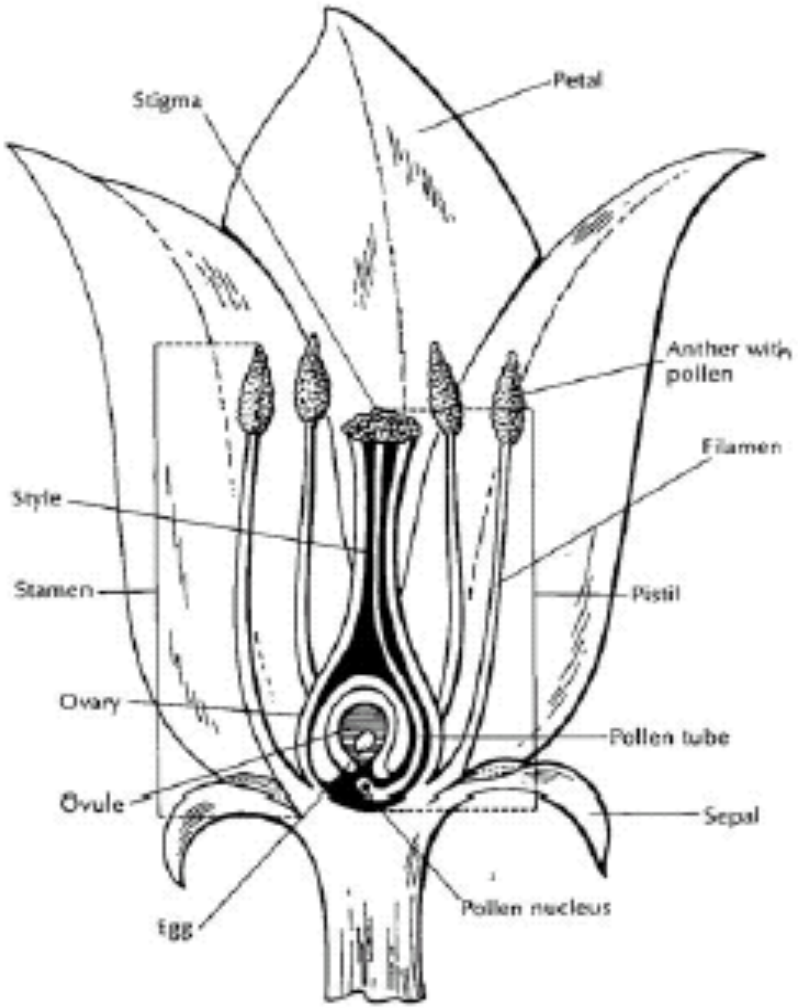
- sepal length
- sepal width
- petal length
- petal width

Three classes (species of iris)

- setosa
- versicolor
- virginica

50 instances of each

Parts of a Flower

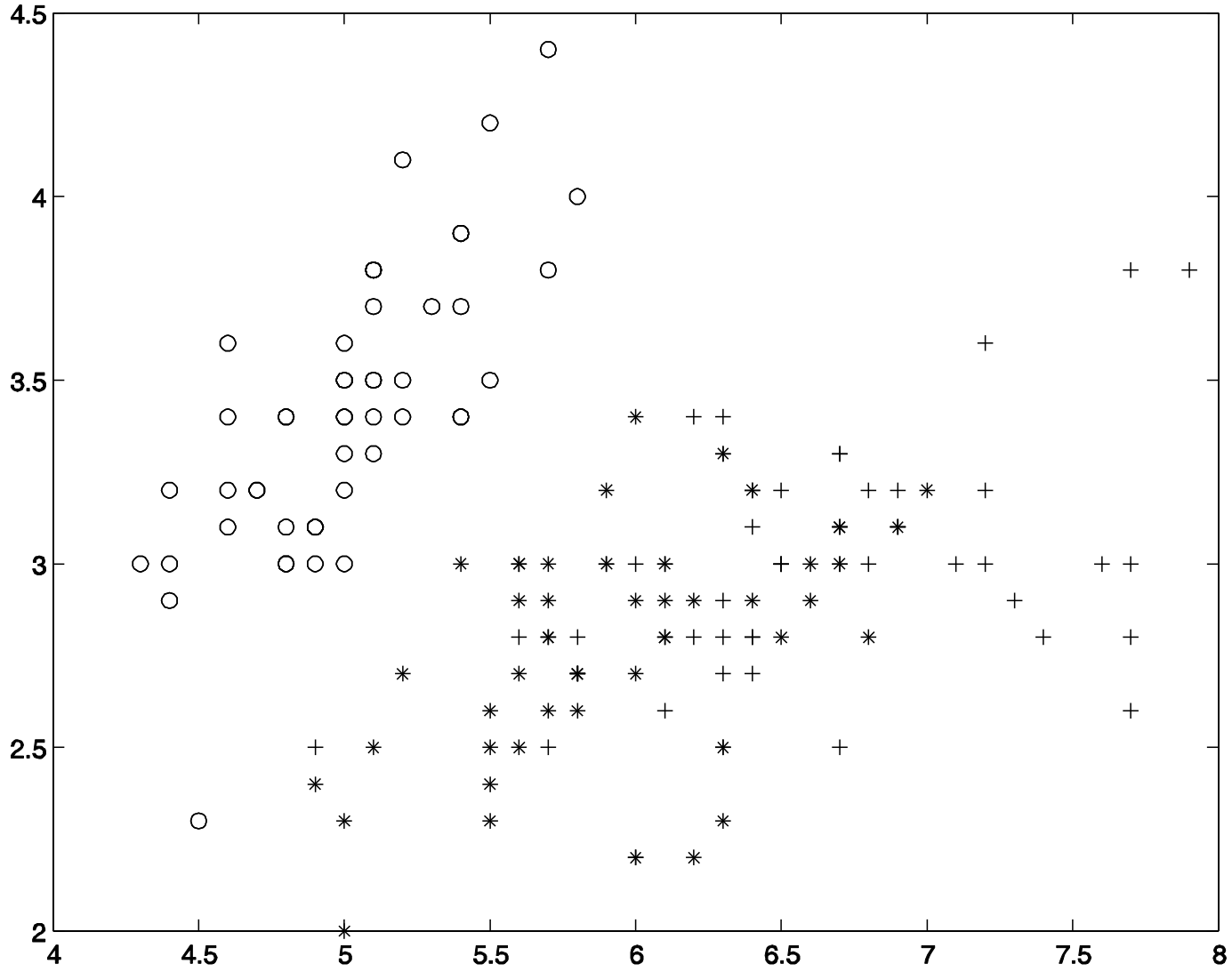


THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS

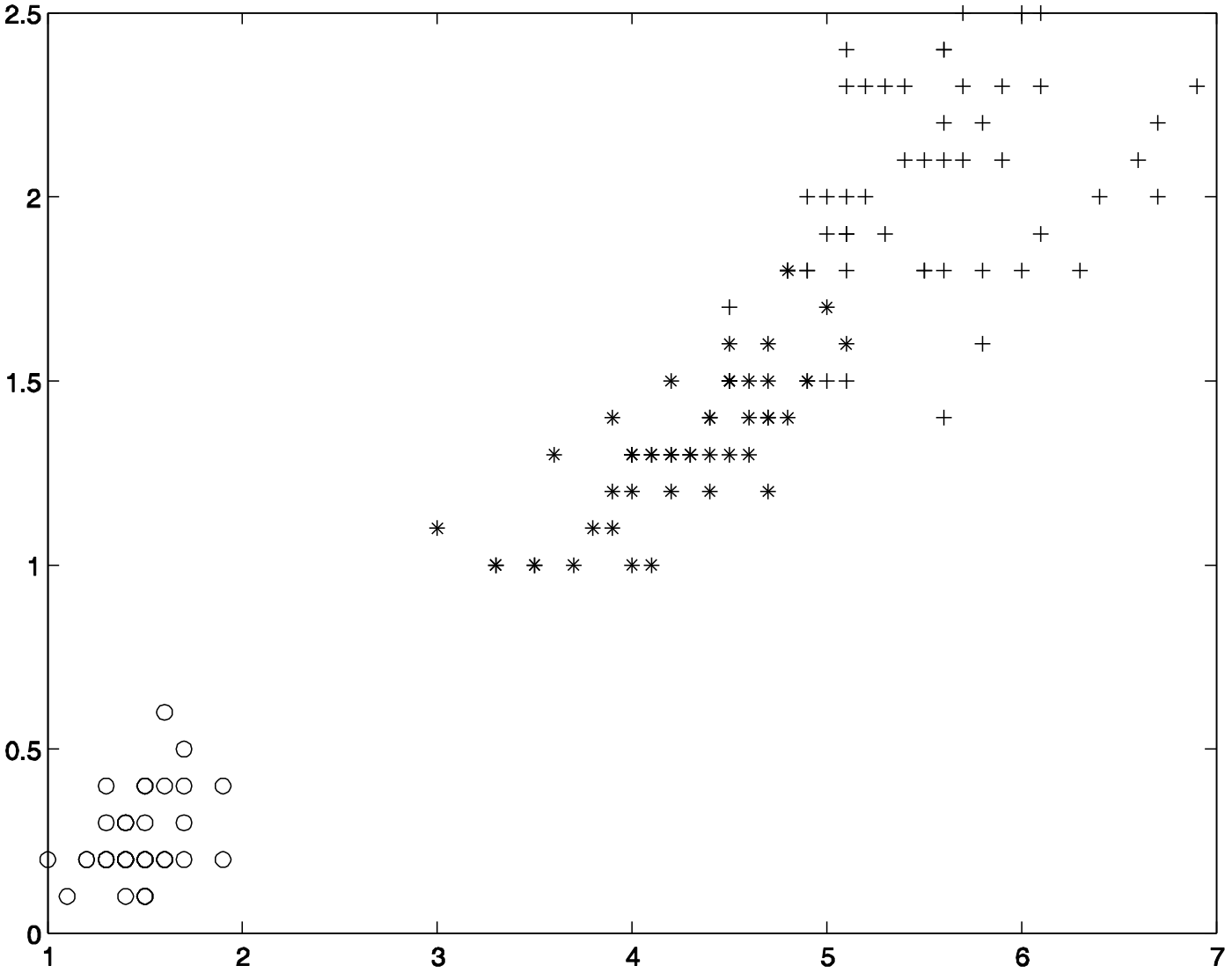
By R. A. FISHER, Sc.D., F.R.S.

I. Discriminant functions

When two or more populations have been measured in several characters, $x_1, \ldots, x_s$, special interest attaches to certain linear functions of the measurements by which the populations are best discriminated. At the author's suggestion use has already been made of this fact in craniometry (*a*) by Mr E. S. Martin, who has applied the principle to the sex differences in measurements of the mandible, and (*b*) by Miss Mildred Barnard, who showed how to obtain from a series of dated series the particular compound of cranial measurements showing most distinctly a progressive or secular trend. In the present paper the application of the same principle will be illustrated on a taxonomic problem; some questions connected with the precision of the processes employed will also be discussed.

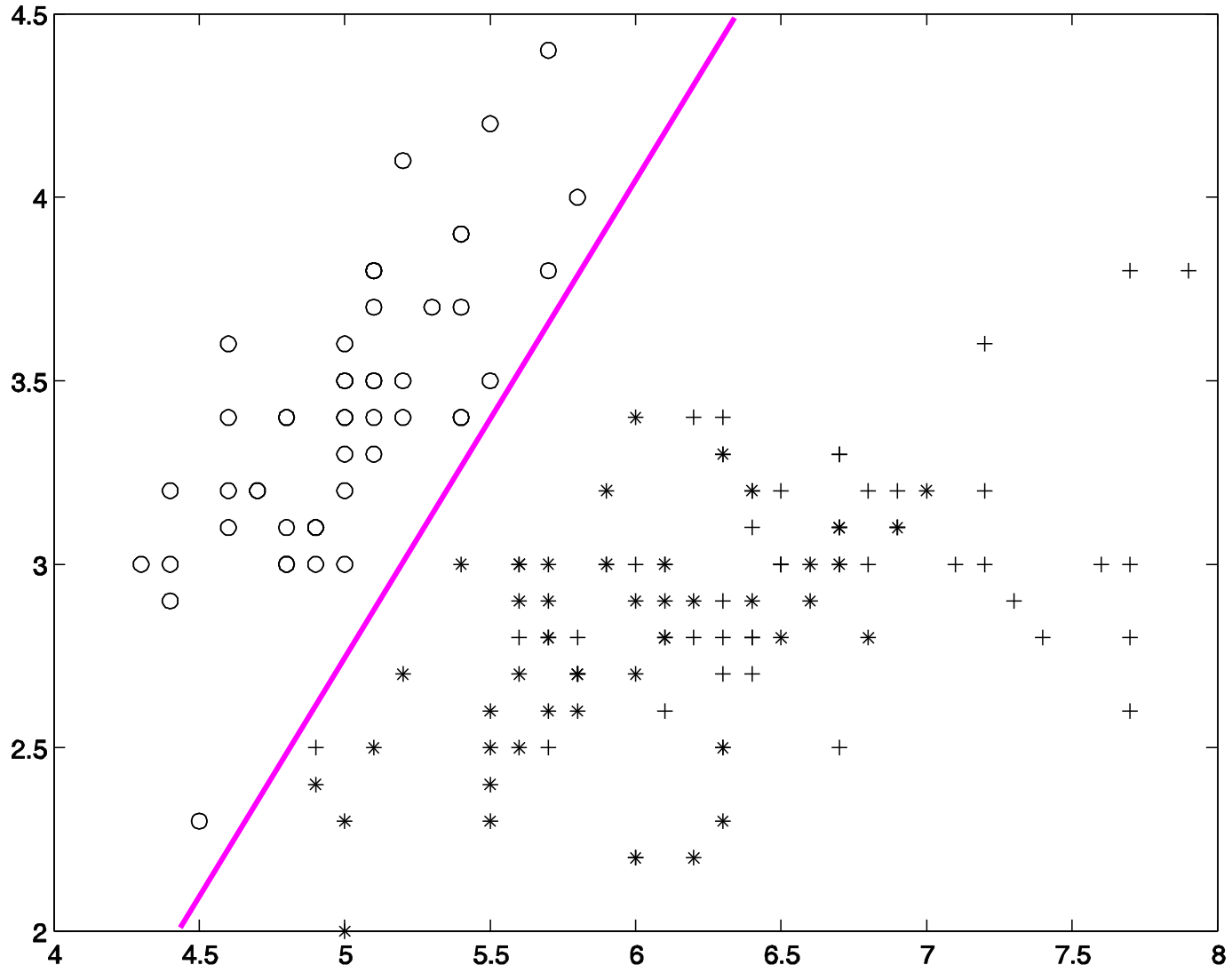


Features 1 and 2 (sepal width/length)



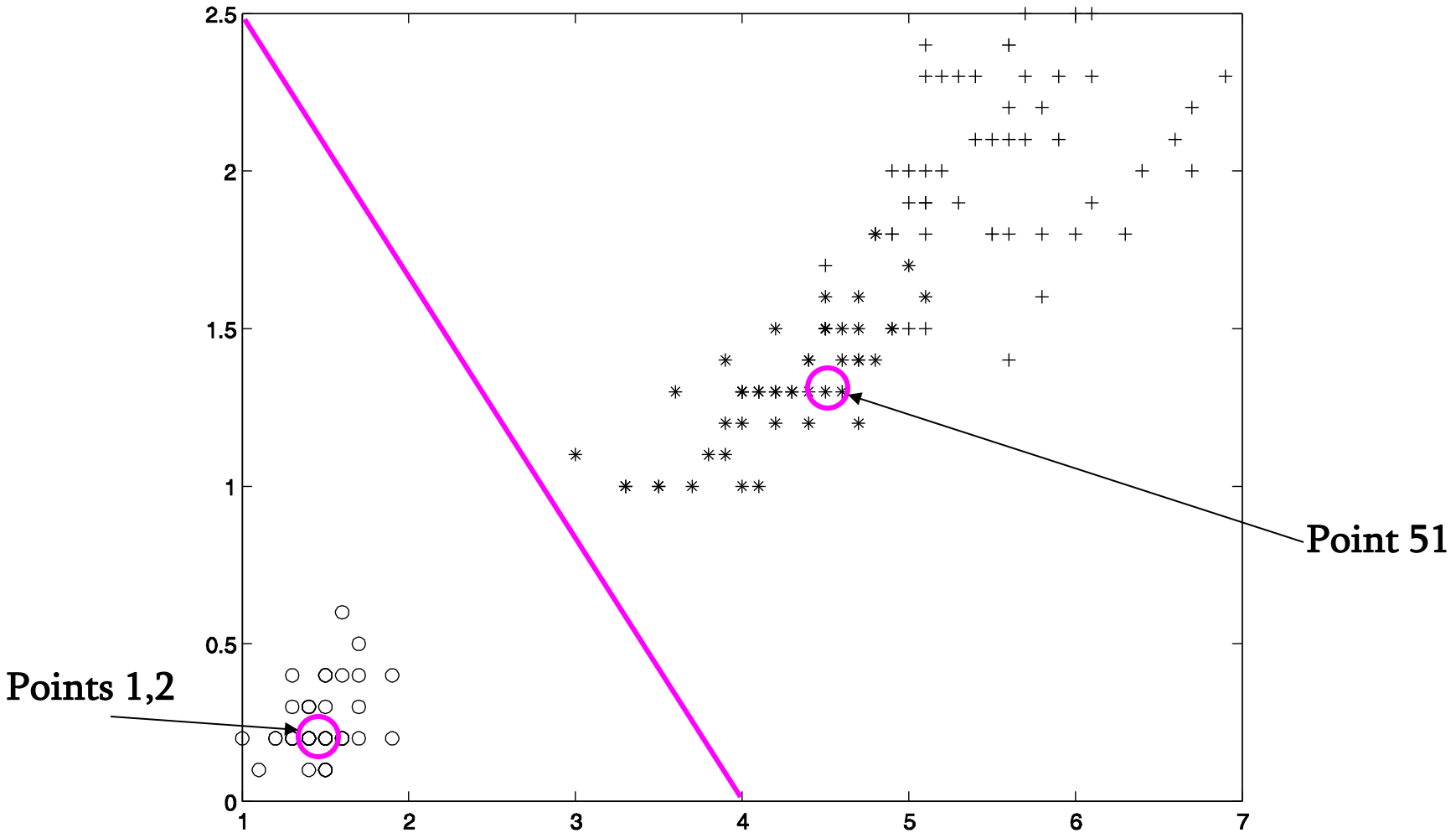
Features 3 and 4 (petal width/length)

Features 1 and 2; goal: separate setosa from other two



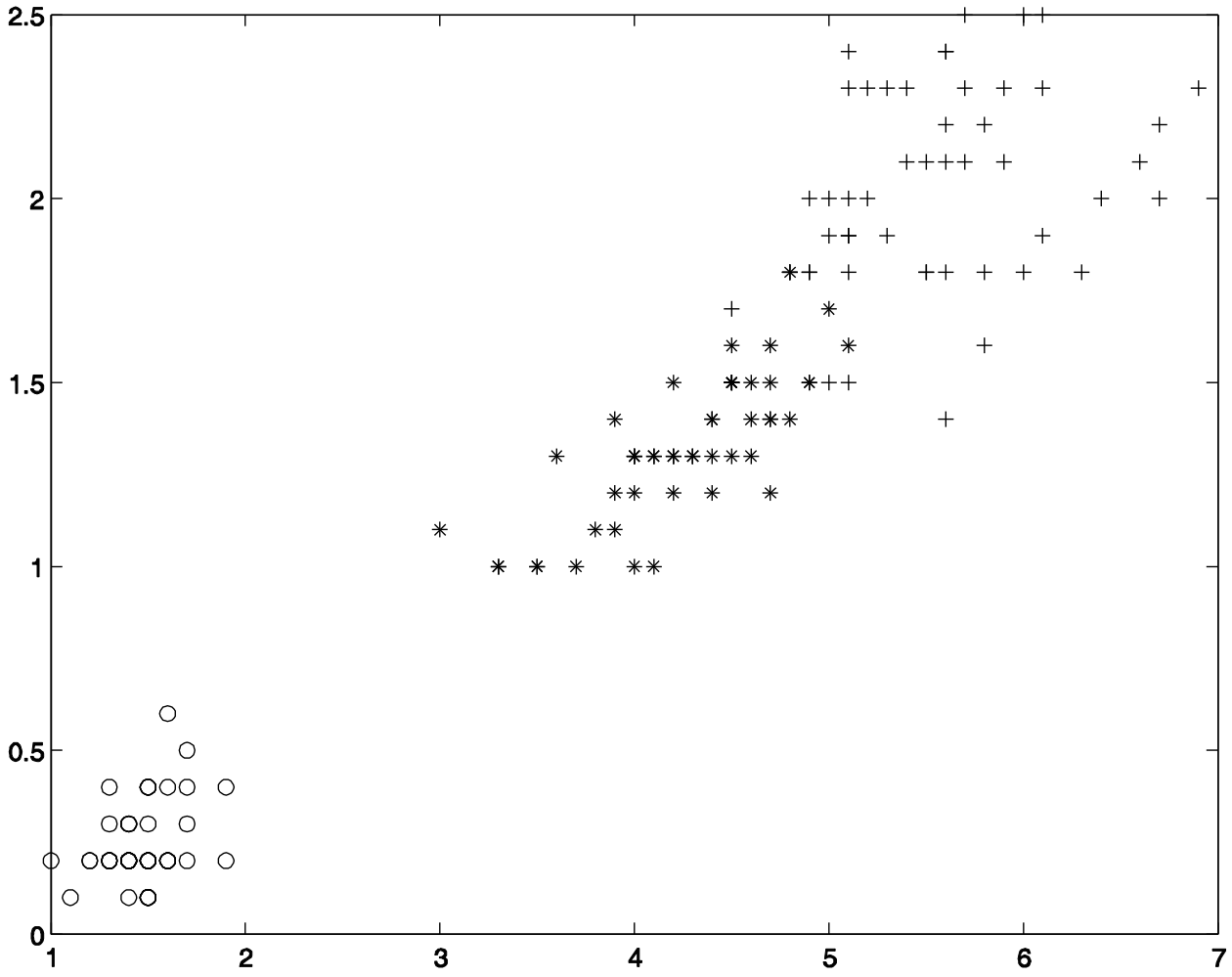
1500 updates (different permutation: 900)

Features 3 and 4; goal: separate setosa from other two



Iteration 1 [1,51] Iteration 2 [1,2] Iteration 3 [ ]

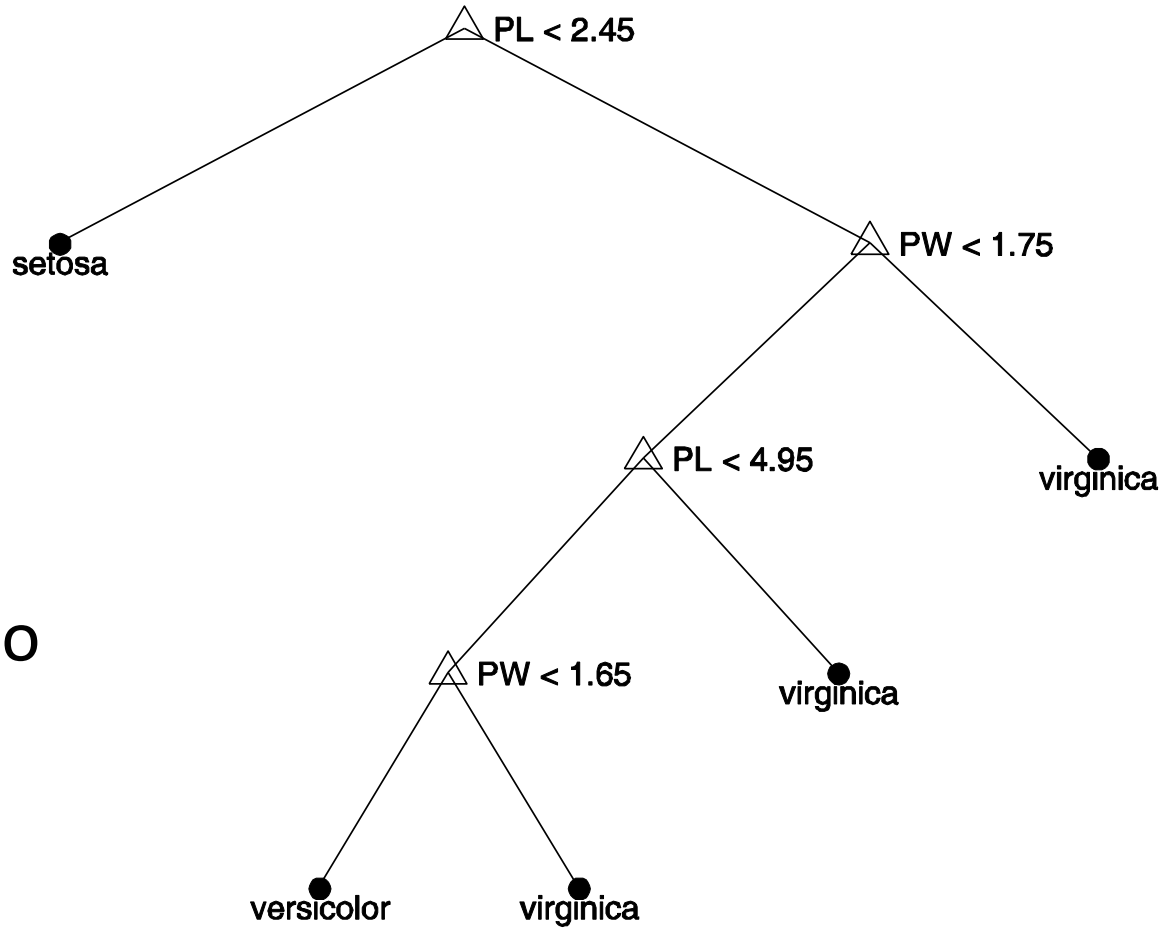
Features 3 and 4; goal: separate versicolor from other two



What if the data is not linearly separable?

Decision trees

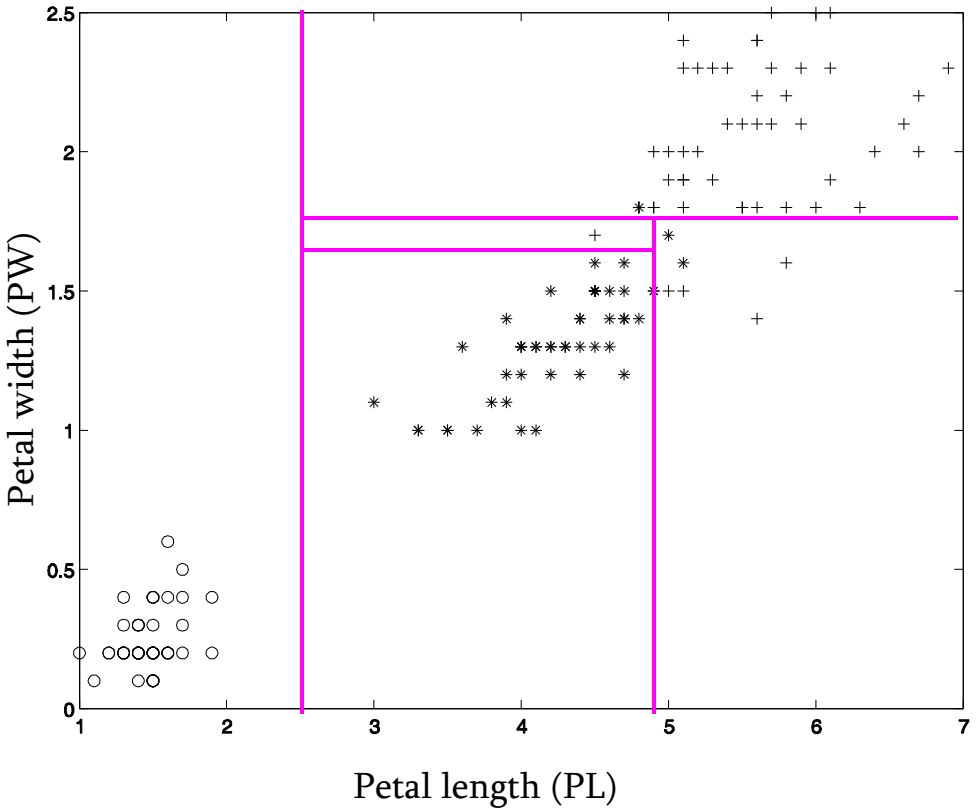
- Simple
- Multiclass
- Not perfect, but close
- Comprehensible to humans

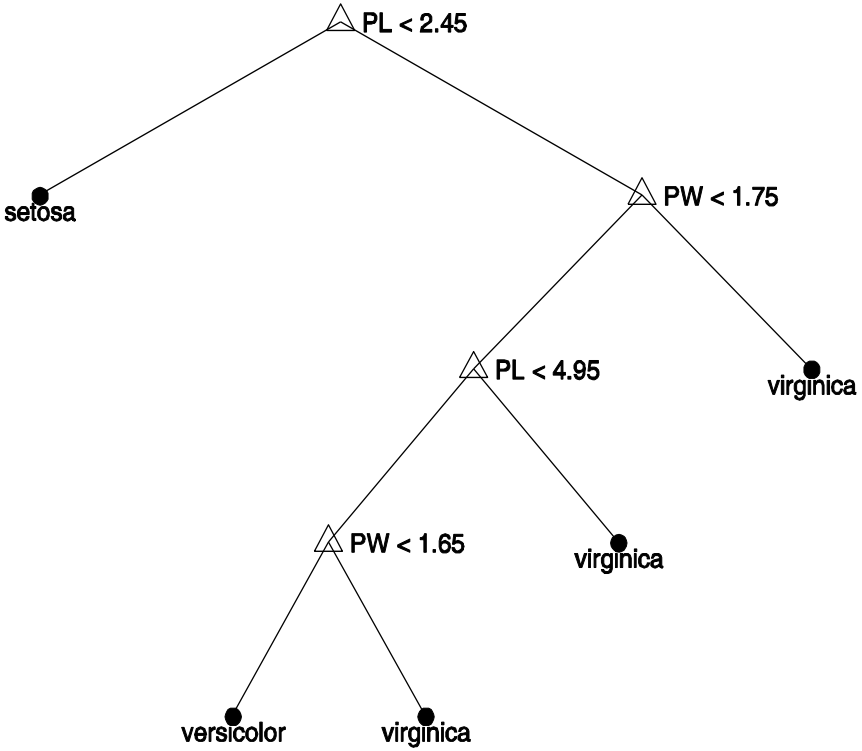


How to build them?

Example

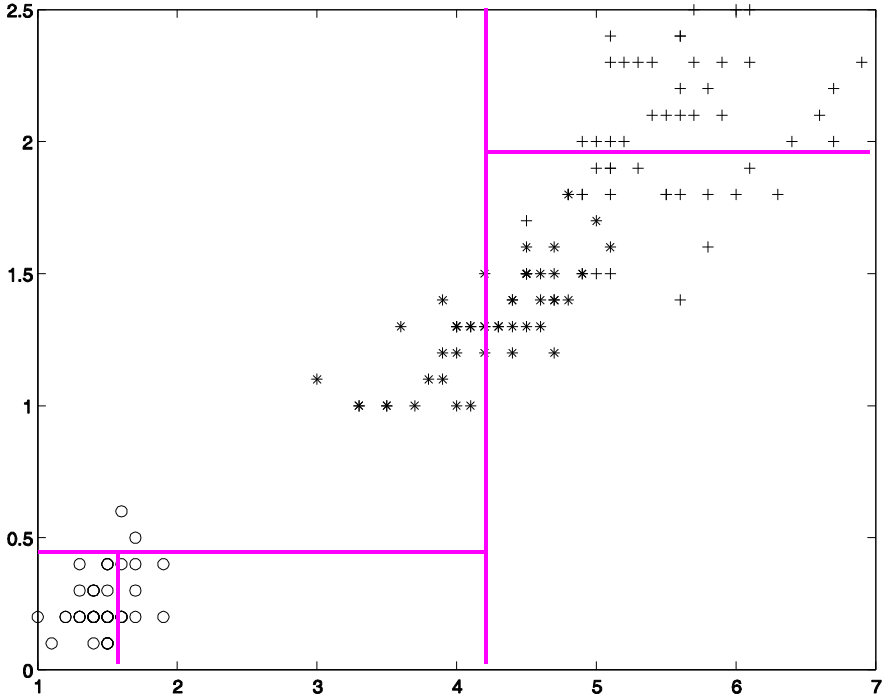
Iris data: 3 classes (setosa, viriginica, versicolor)





K-d trees

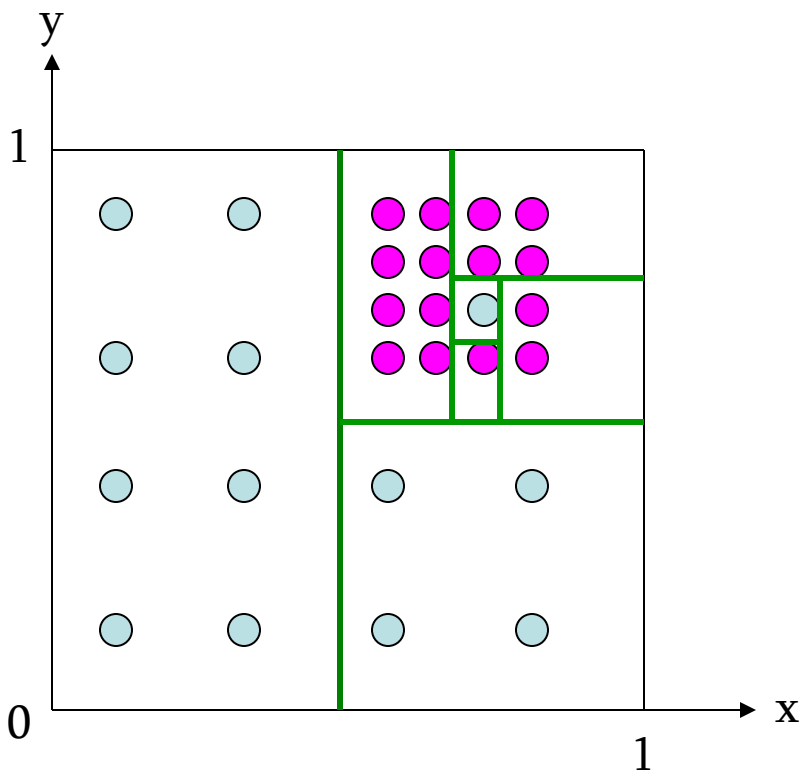
Rapidly partition the space into rectangular regions which contain few points

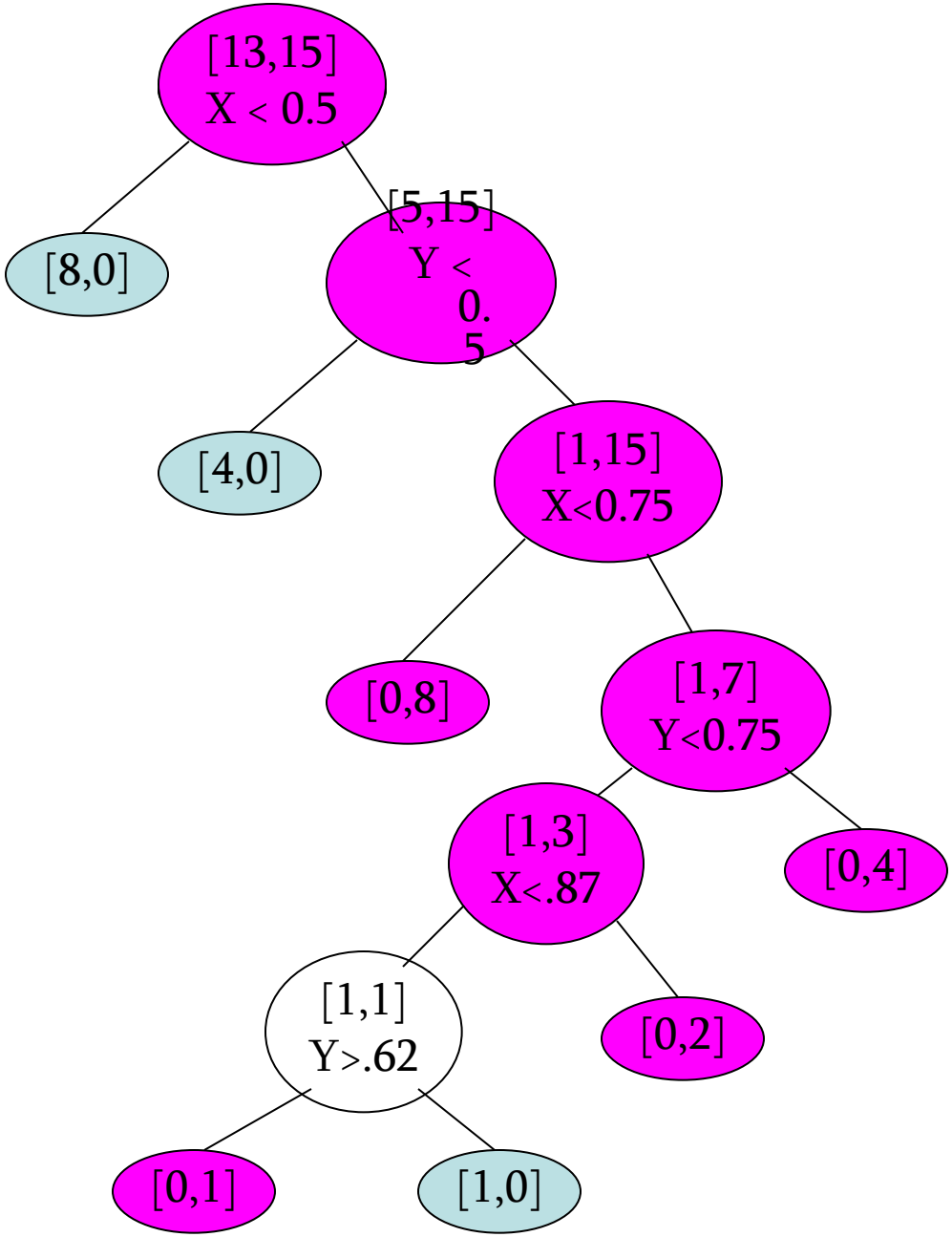


- Cycle through dimensions.
- Split at median.
- Recurse on each side (along next dimension).

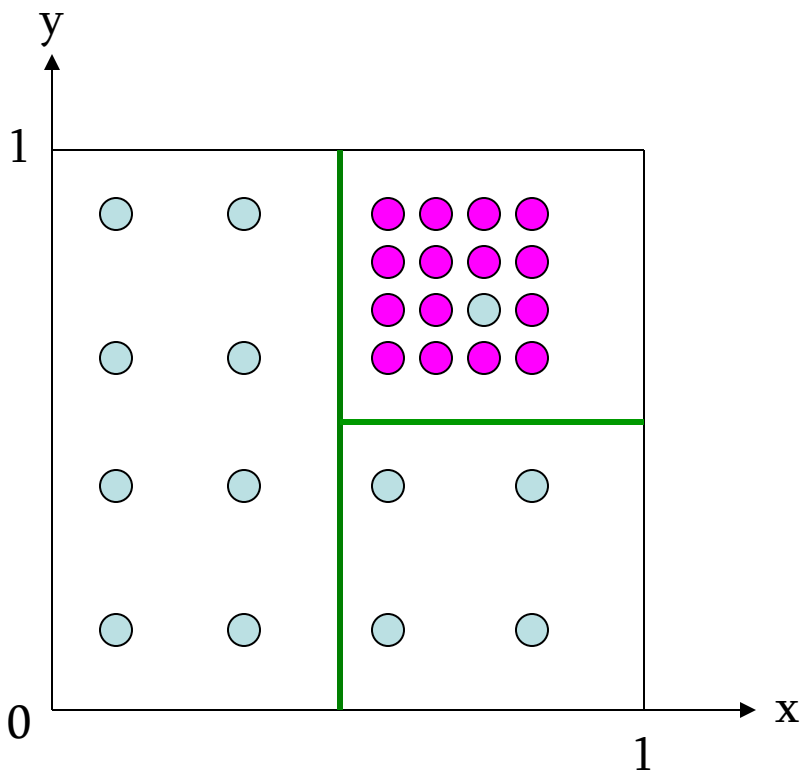
NOTE: Also useful for creating a nearest-neighbor data structure.

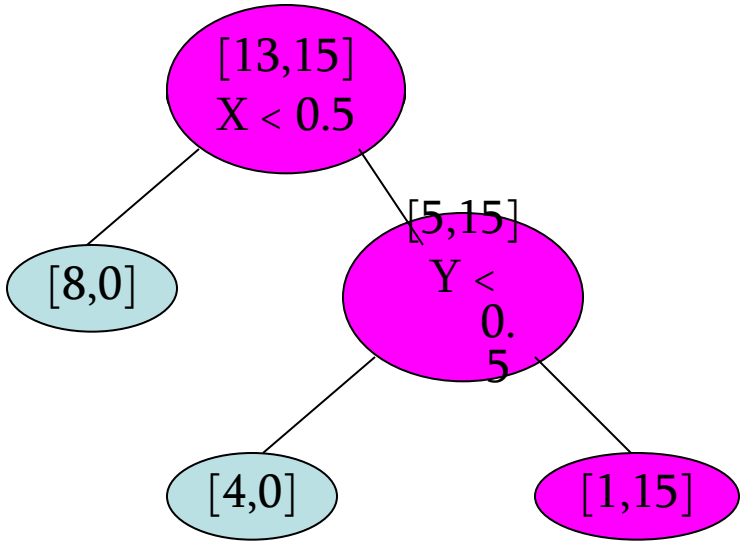
But for the current problem, we don't care how many points each region contains: we just want them to consist (almost) exclusively of points from one class…

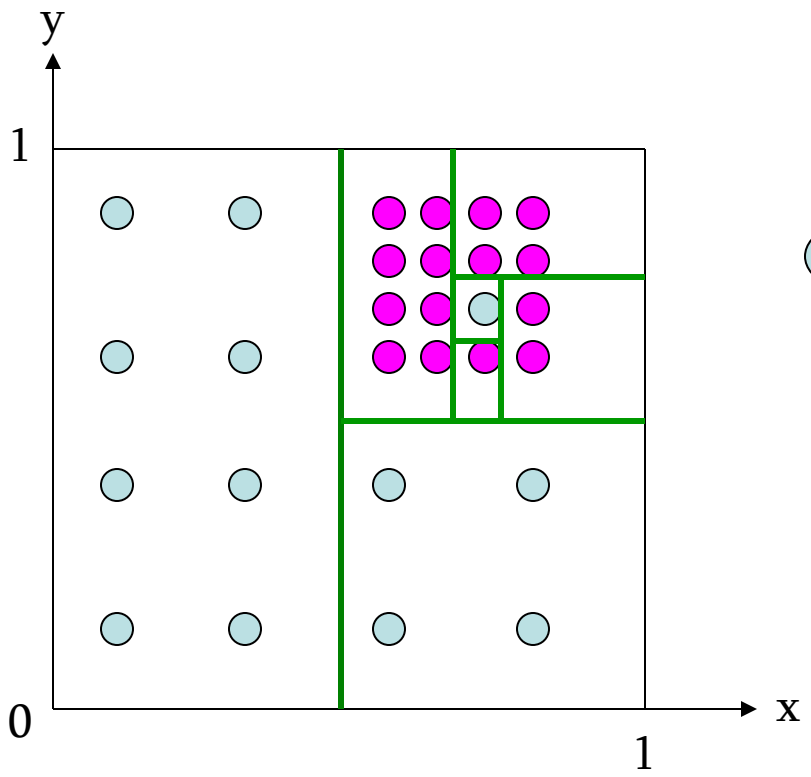

y
1
0
x
1

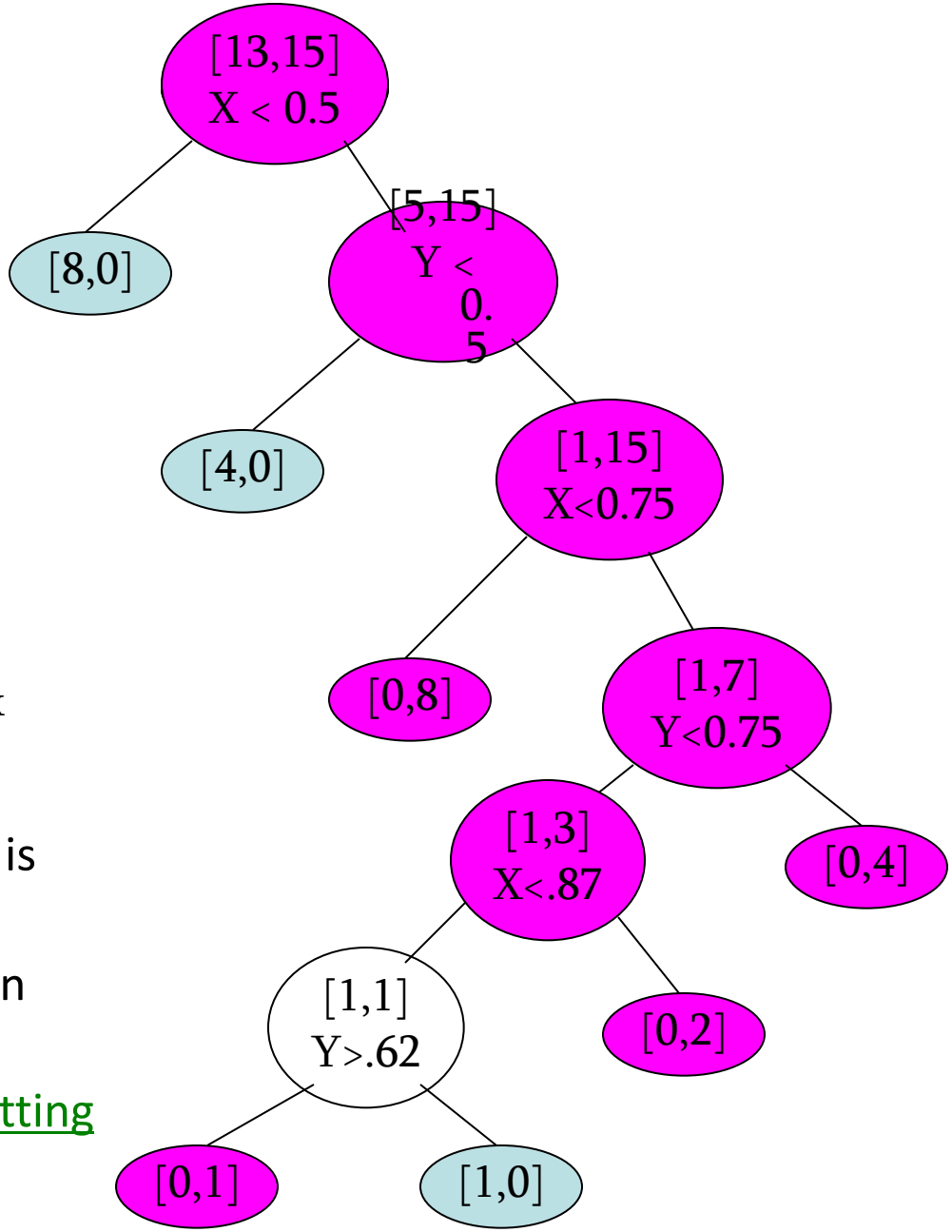

[13,15]
X < 0.5
[8,0]
[5,15]
Y < 0.5
[4,0]
[1,15]
X<0.75
[0,8]
[1,7]
Y<0.75
[1,3]
X<.87
[0,4]
[1,1]
Y>.62
[0,2]
[0,1]
[1,0]

Retracing a few steps…









- This tree does slightly better – but is much more complex.
- And that one point was probably an outlier anyway.
- We have probably ended up overfitting the data.

Decision tree issues

Very expressive family of classifiers:

Any type of data can be accommodated:
real, Boolean, categorical, …

Can perfectly fit any self-consistent training set
- no example pairs (x, y), (x, y')

But this also means that there is serious danger of overfitting.

Building a decision tree

Greedy algorithm: build the tree top-down

At each stage:

- Look at all current leaves and all possible splits
- Choose the split that most decreases the uncertainty

We need a measure of uncertainty…

Uncertainty

e.g.: **+** p fraction of the points

- 1-p fraction

How uncertain is this?

(i) Entropy

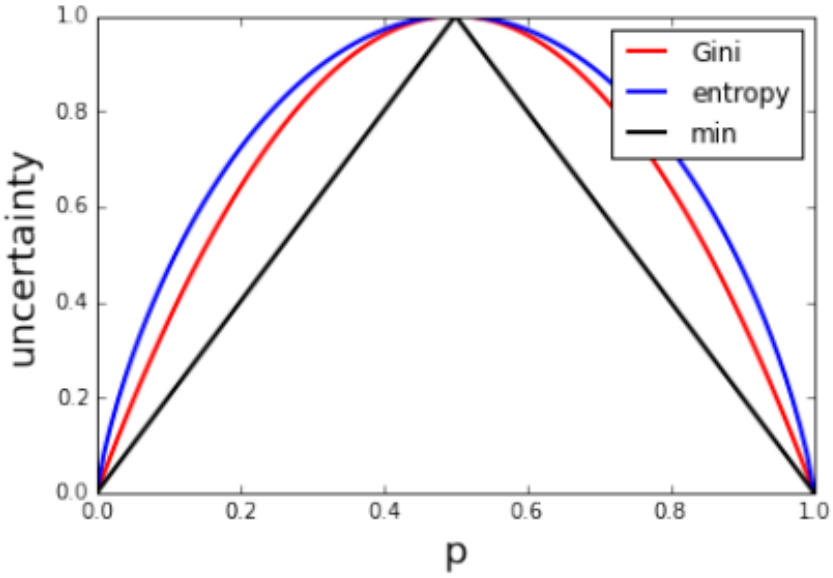
$$p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$$

(ii) Gini index

$$2p(1-p)$$

(iii) Min

$$\min\{p, 1-p\}$$

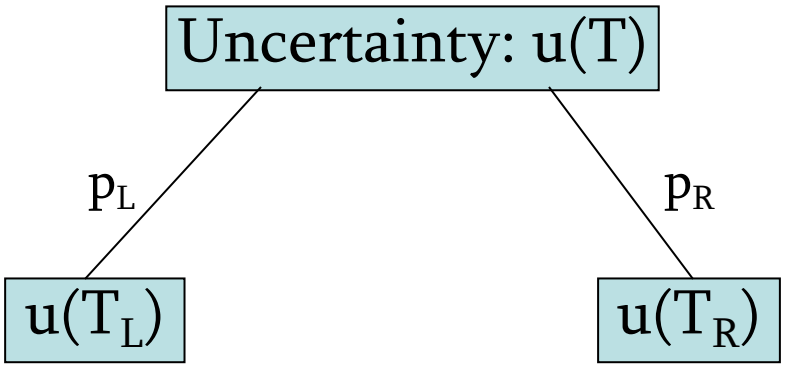


Uncertainty

Generalize to k classes: p_1, p_2, …, p_k fraction of points

	k = 2	General k
Entropy	$p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$	$\sum_{i=1}^{k} p_i \log \frac{1}{p_i}$
Gini index	$2p(1-p)$	$\sum_{i=1}^{k} p_i(1-p_i) = 1 - \|p\|^2$
Min	$\min\{p, 1-p\}$	$\min_i p_i$

The benefit of a split



Of the points in T:

p_L fraction go to T_L

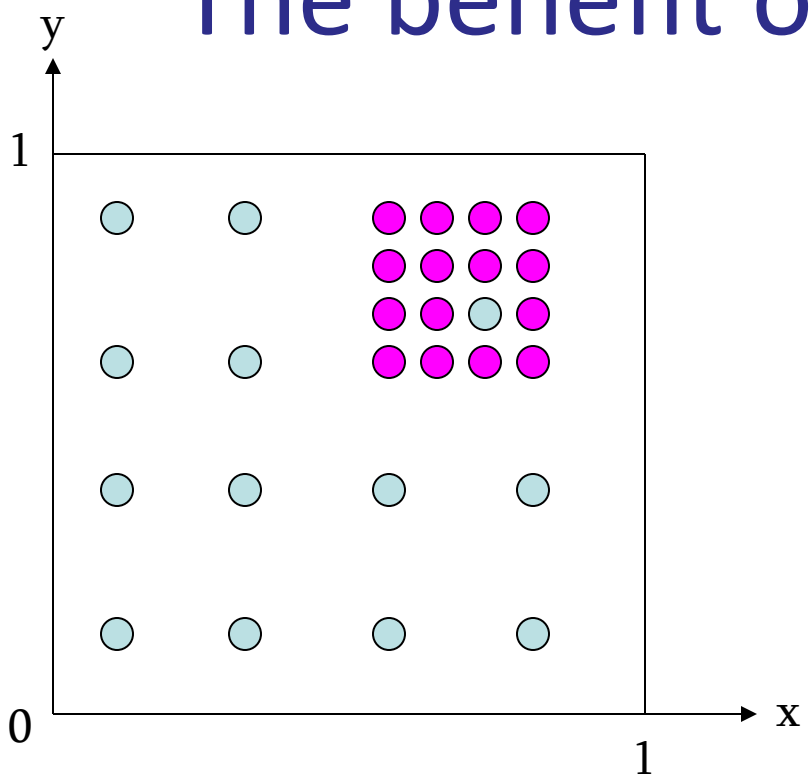
p_R fraction go to T_R

$$\text{Benefit of split} = (u(T) - \{p_L\, u(T_L) + p_R\, u(T_R)\}) \times |T|$$

Expected uncertainty after split

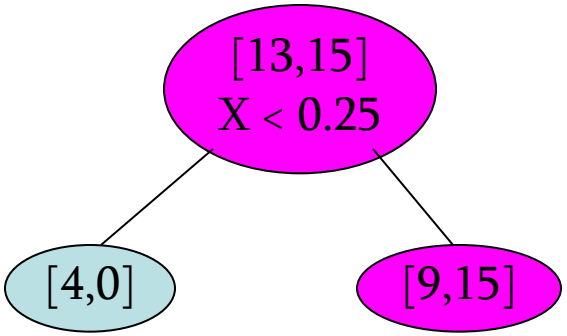
points in T

The benefit of a split: example



Initial uncertainty (Gini):

$$u = 2 \cdot \frac{13}{28} \cdot \frac{15}{28}$$



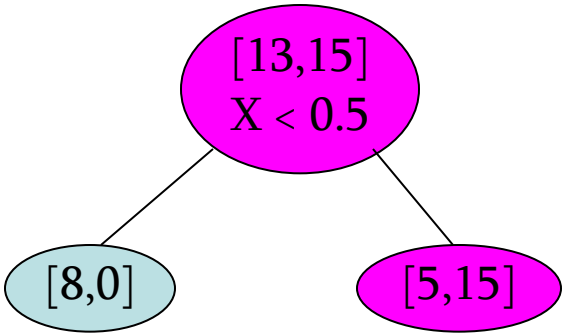
$p_L = 4/28$

$u_L = 0$

$p_R = 24/28$

$u_R = 2 \times 9/24 \times 15/24$

Gain = $u - p_L u_L - p_R u_R = u - 22.5/56$



Gain = $u - p_L u_L - p_R u_R = u - 15/56$

Greedy decision tree building

Start with all points in a single node

Repeat

    Pick the split with greatest benefit

Until ???

When to stop?

(i) When each leaf is pure?

(ii) When the tree is already pretty big?

(iii) When each leaf has uncertainty < some threshold?

Common strategy: keep going until leaves are pure (recall: this didn't work too well for us earlier…)

Then, shorten the tree by pruning, to correct the overfitting problem.

What is overfitting?

Data comes from some true underlying distribution D on X x Y.

[X = input space, Y = label space]

All we ever see are samples from D: training set, test set, etc.

When we choose a classifier h: X → Y, we can talk about its error on the training set (x_1, y_1), …, (x_m, y_m):

$$\hat{\epsilon}(h) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}(h(x_i) \neq y_i)$$

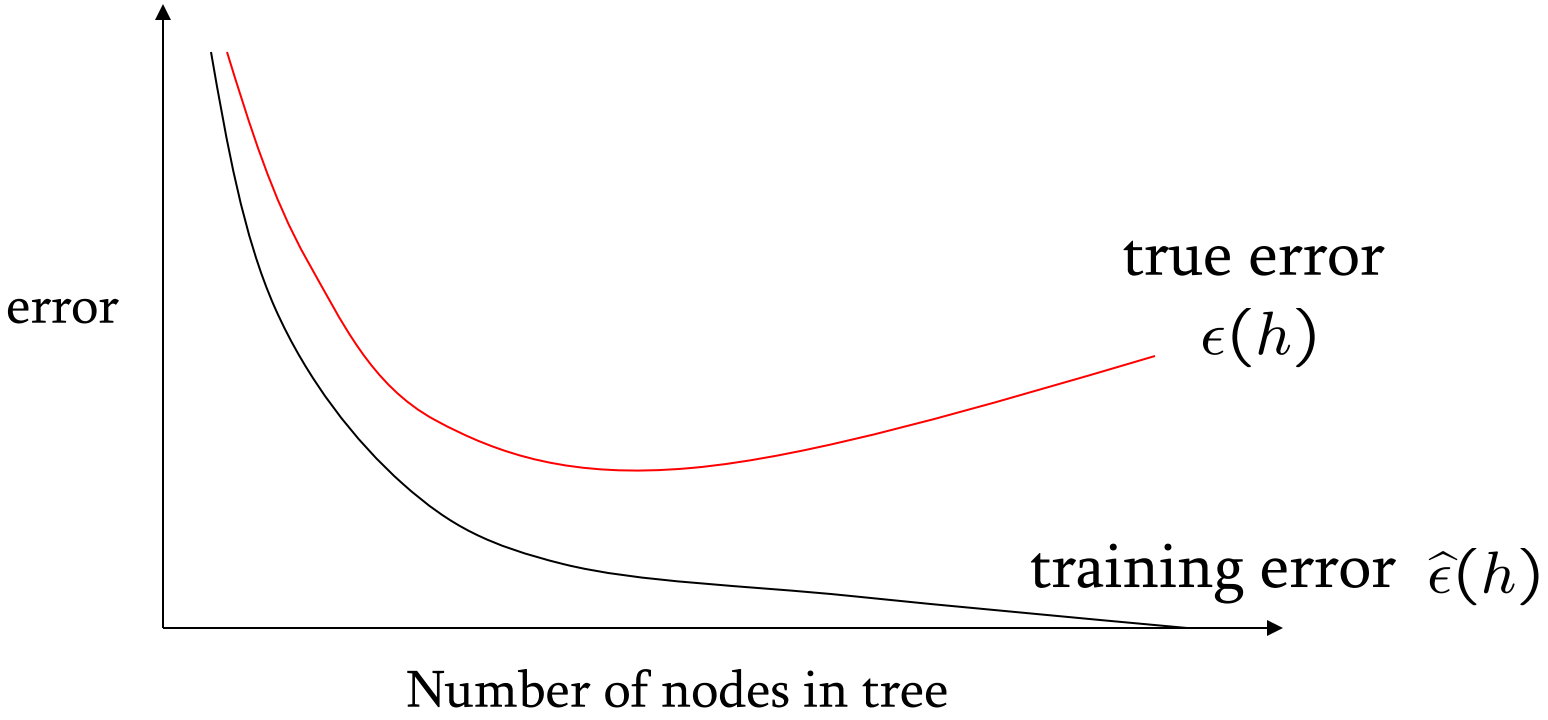
But we can also talk about its true error:

$$\epsilon(h) = \mathbf{P}_{(x,y)\sim D}(h(x) \neq y)$$

How are these two quantities related?

Overfitting: picture

E.g., building a decision tree



As we make our tree more and more complicated:

- The training error keeps going down
- but the true error stops improving and may even get worse!

Overfitting: one perspective

1. The true underlying distribution D is the one whose structure we would like to capture
2. The training data reflects the structure of D, so it helps us.
3. But it also has **chance** structure of its own – we must try to avoid modeling this.

For instance: D = uniform distribution over {1,2,3,…,100}

Pick three training points: eg. 6, 12, 98.

They all happen to be even: but this is just chance structure.

It would be bad to build this into a classifier.

Overfitting: another perspective

"Fit a line to a point": absurd

"Fit a plane to two points": likewise

Moral: It is not good to use a model which is so complex that there isn't enough data to reliably estimate its parameters.

Decision tree pruning

[1] Split the training data S_{full} into two parts:

- A smaller training set S
- A validation set V (a model of reality, a surrogate test set)

[2] Build a full decision tree T using S

[3] Then prune T using V:

Repeat

if there a node u in T such that removing the subtree rooted at u decreases the error on V:

T = T – {subtree rooted at u}

Of course, V has chance structure too, but its chance structure is unlikely to coincide with that of S.

Example: SPAM data set

4601 points, each corresponding to an email message

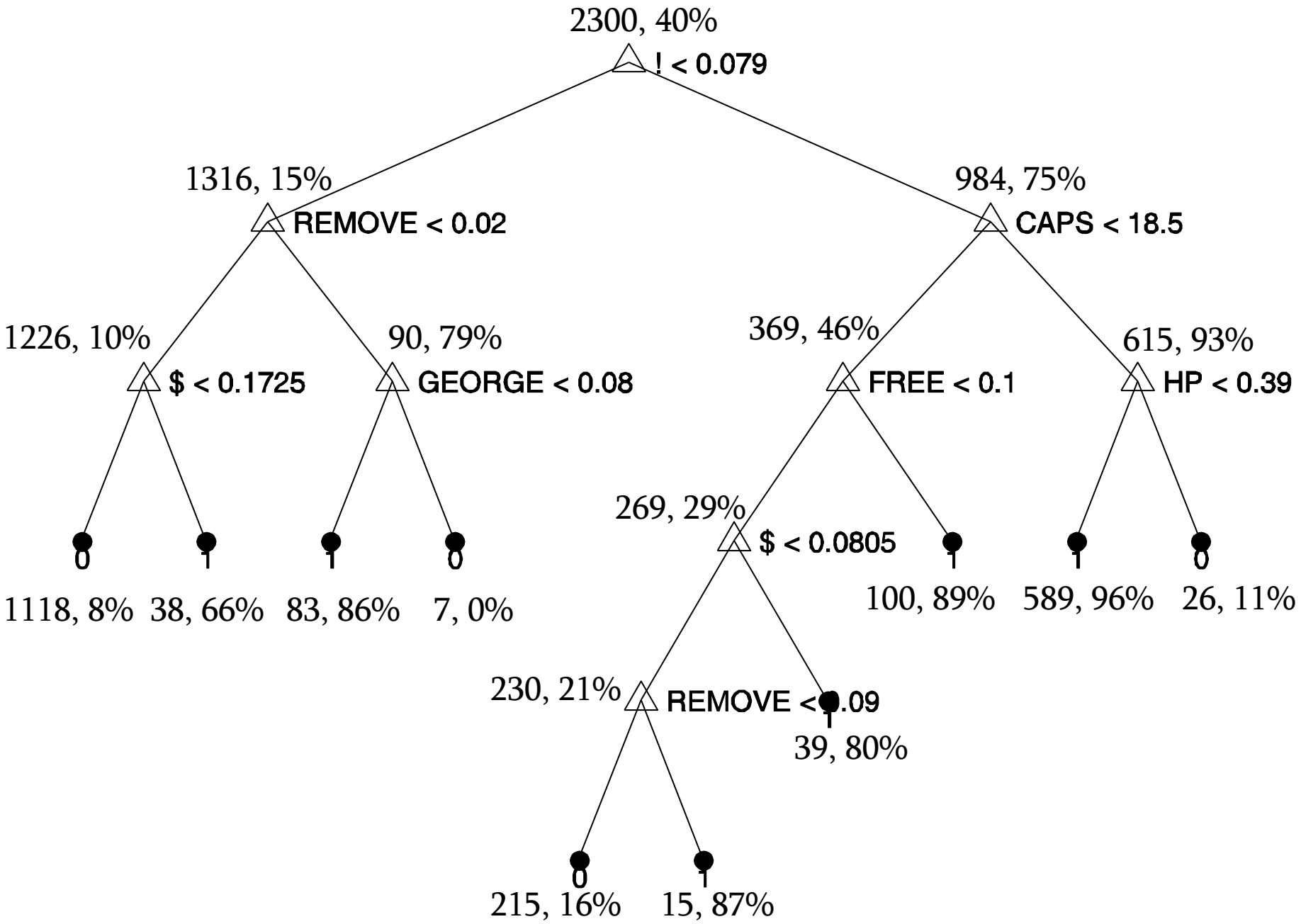
39.4% are SPAM

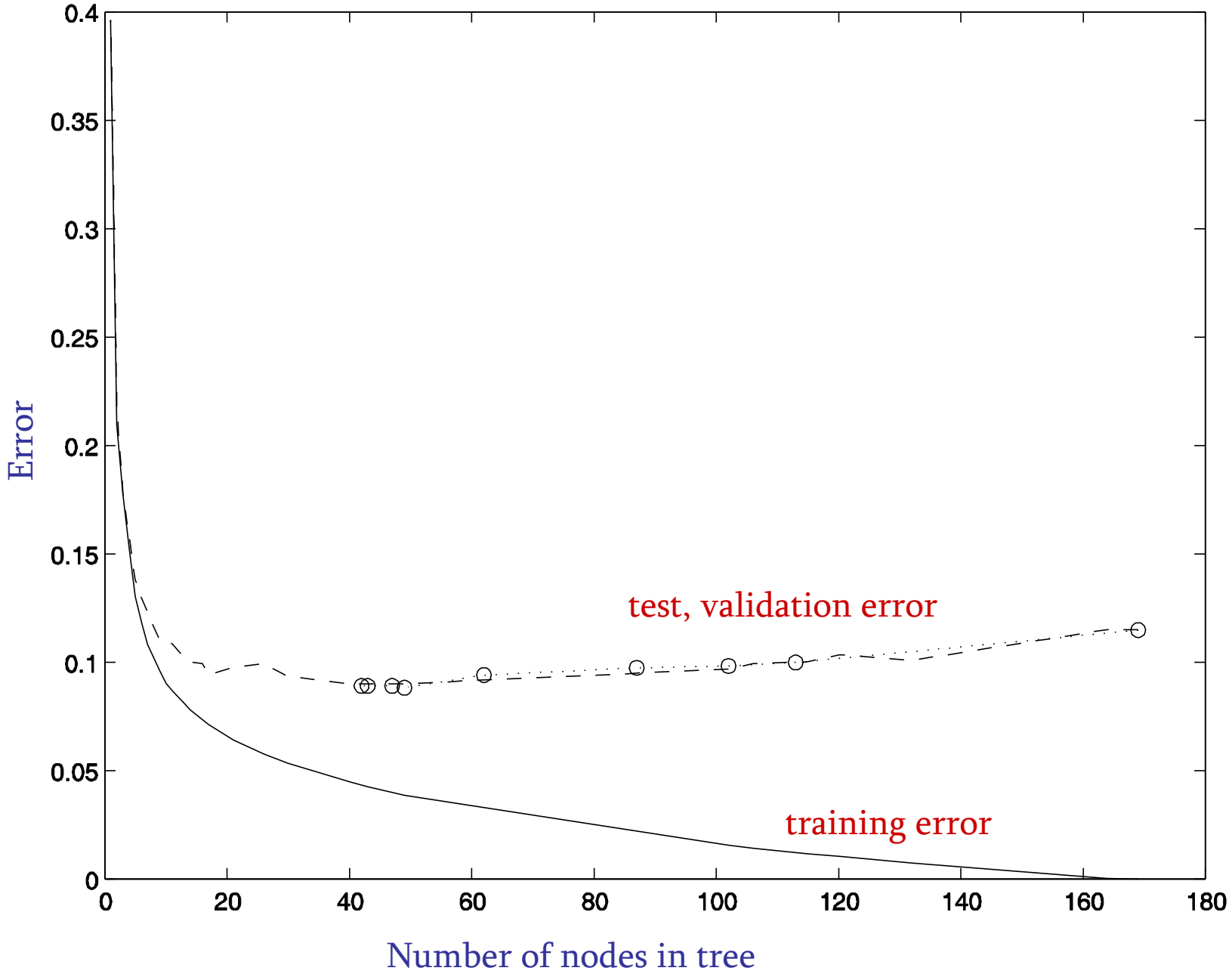
Each point has 57 features:

- 48 check for specific words, eg. FREE
- 6 check for specific characters, eg. !
- 3 others, eg. longest run of capitals

Randomly divide into three parts:

50%	training data
25%	validation
25%	testing


2300, 40%
! < 0.079
1316, 15%
REMOVE < 0.02
984, 75%
CAPS < 18.5
1226, 10%
$ < 0.1725
90, 79%
GEORGE < 0.08
369, 46%
FREE < 0.1
615, 93%
HP < 0.39
269, 29%
$ < 0.0805
230, 21%
REMOVE < 0.09
0
1
1
0
1
1
1
0
1118, 8%
38, 66%
83, 86%
7, 0%
100, 89%
589, 96%
26, 11%
39, 80%
0
1
215, 16%
15, 87%


0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Error
0
20
40
60
80
100
120
140
160
180
Number of nodes in tree
test, validation error
training error

How accurate is the validation set?

How accurate are error estimates based on the validation set?

For any classifier h and underlying distribution D on X x Y:

"true error" $\quad \text{err}(h) = \mathbf{P}_{(x,y)\sim D}(h(x) \neq y)$

"error on set S" $\quad \text{err}(h, S) = \frac{1}{|S|} \sum_{(x,y)\in S} \mathbf{1}(h(x) \neq y)$

Suppose S is chosen i.i.d. (independent, identically distributed) from D. Then (over the random choices of S),

$$\mathbf{E}[\text{err}(h, S)] = \text{err}(h)$$

And the standard deviation of err(h,S) is about $1/\sqrt{|S|}$.

How accurate is the validation set?

Fix any h. Suppose S is chosen i.i.d. (independent, identically distributed) from D. Then (over the random choices of S),

$$\mathbf{E}[\text{err}(h, S)] = \text{err}(h)$$

And the standard deviation of err(h,S) is about $1/\sqrt{|S|}$

(i) In this scenario, S is used to assess the accuracy of a single, pre-specified classifier h.

Can the same S be used to check many classifiers simultaneously?

Answer: VC theory

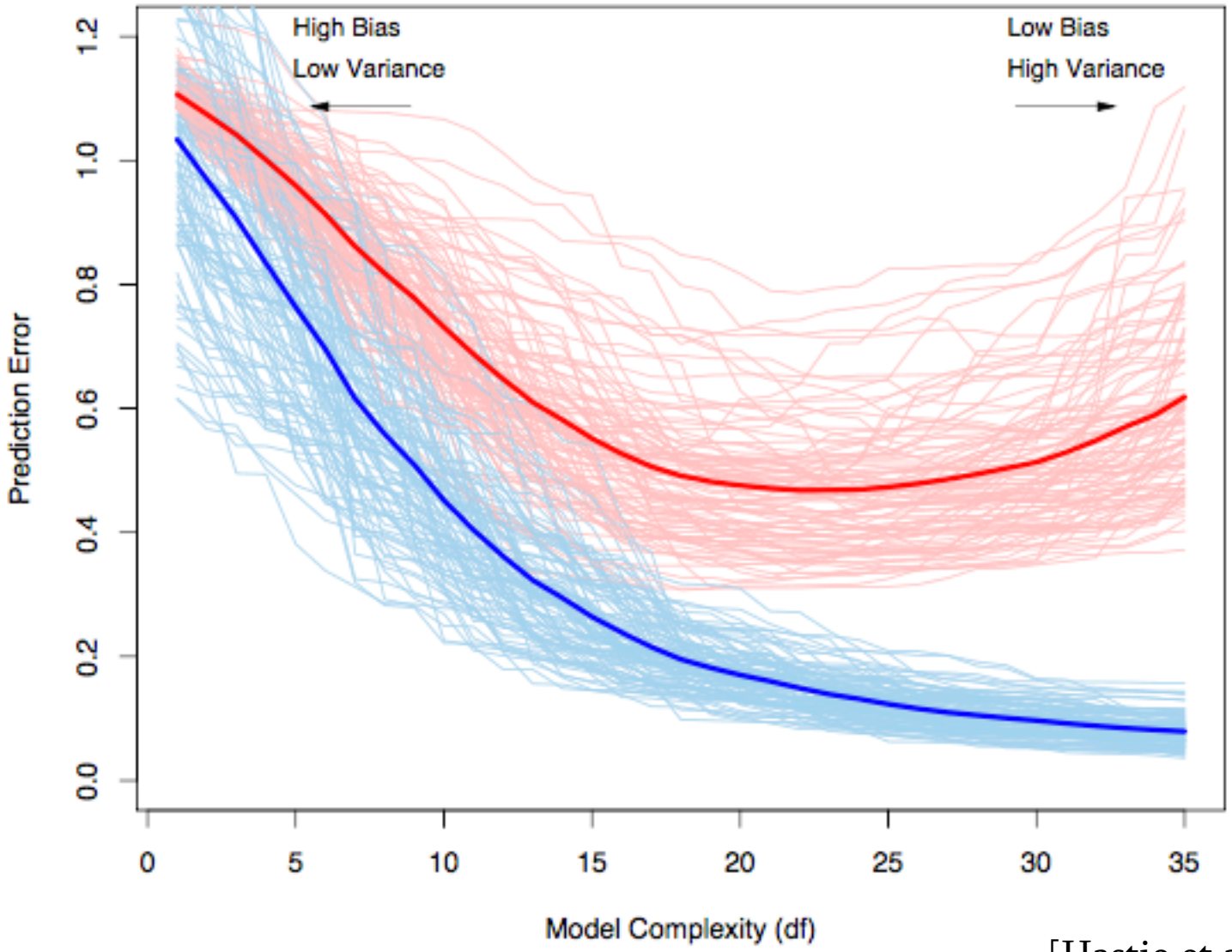
(ii) In particular, if h was created using S as a training set, then the above scenario does not apply. In such situations, err(h,S) might be a very poor estimate of err(h).

Example

Predict flu trends using search data

- X: search data, Y: fraction of population with flu
- S_{train} = all data before 2012
- S_{test} = all data in 2012

Example



Test error

Training error

[Hastie et al., 2001. Fig. 7.1]

Bias-Variance Tradeoff

Allowing the model (hypothesis class) to be more and more complex, the resulting classifier, *h*, will have a better and better fit to the training data, thus the bias of *h* reduces.

But as complexity increases, *h* will increasingly overfit to the chance structure of the particular training data set. So there will be higher variance of the true error of classifiers, *h'*, learned over different training sets.

Bias-Variance Error Decomposition

If we assume all data points (x,y) obey: $y = f(x) + \epsilon$

where: $\mathrm{E}[\epsilon] = 0$

$\mathrm{Var}(\epsilon) = \sigma^2$

Then, $\mathrm{Err_h}(x) = \mathrm{E}[(y - h(x))^2 | X = x]$

$= \sigma^2 + \mathrm{Var}[h] + (\mathrm{Bias}[h])^2$

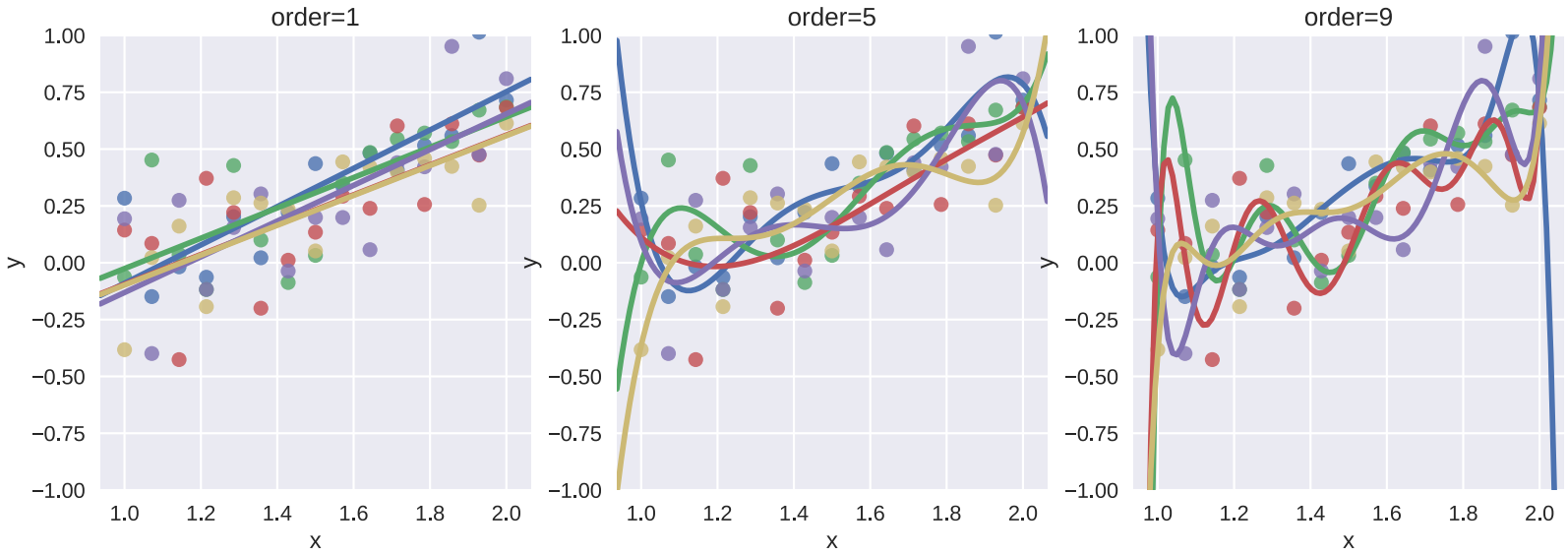
Irreducible error (can't be changed by choice of h)

where: $\mathrm{Var}[h] = \mathrm{E}[h(x)^2] - \mathrm{E}[h(x)]^2$

$\mathrm{Bias}[h] = (f(x) - \mathrm{E}[h(x)])$

NOTE: All expectations are w.r.t. the random training set *h* is learned from, NOT *x*.

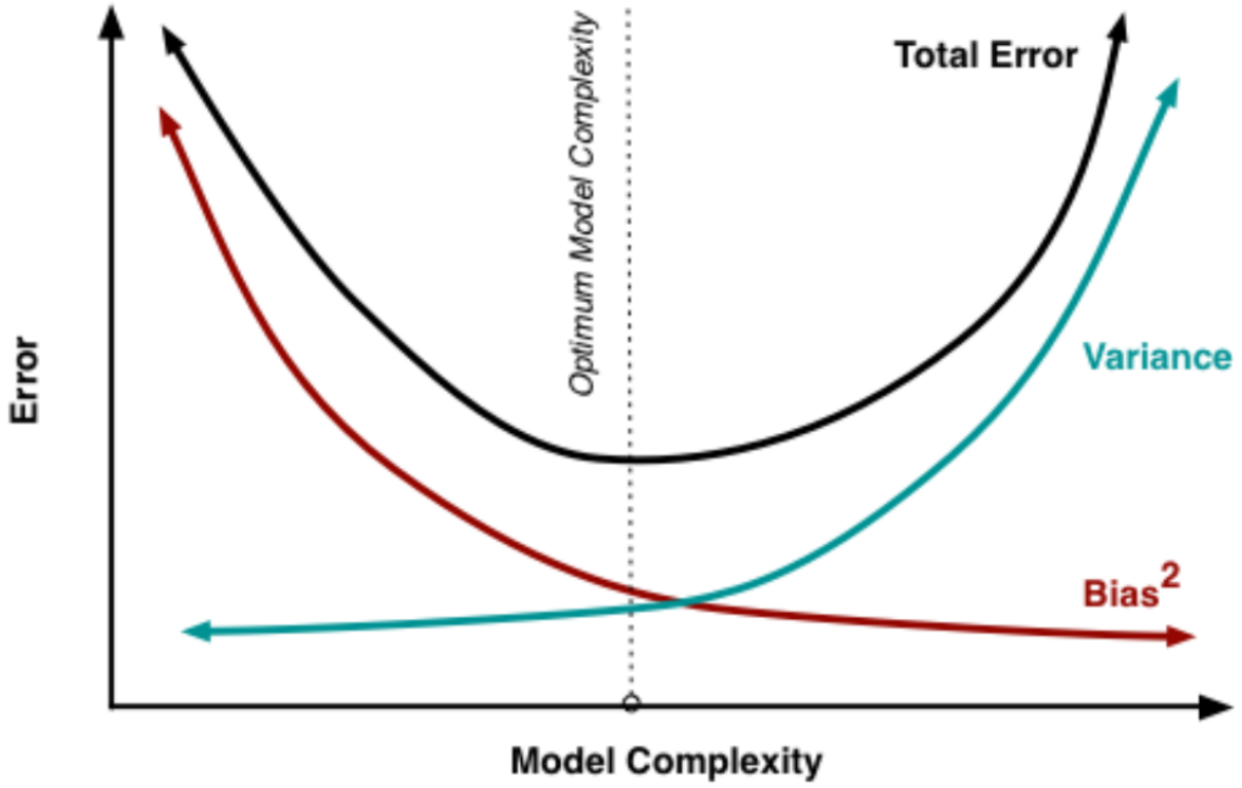
Example



Increasing the order of the polynomial model used to fit the data, increases model complexity eventually leading to overfitting.

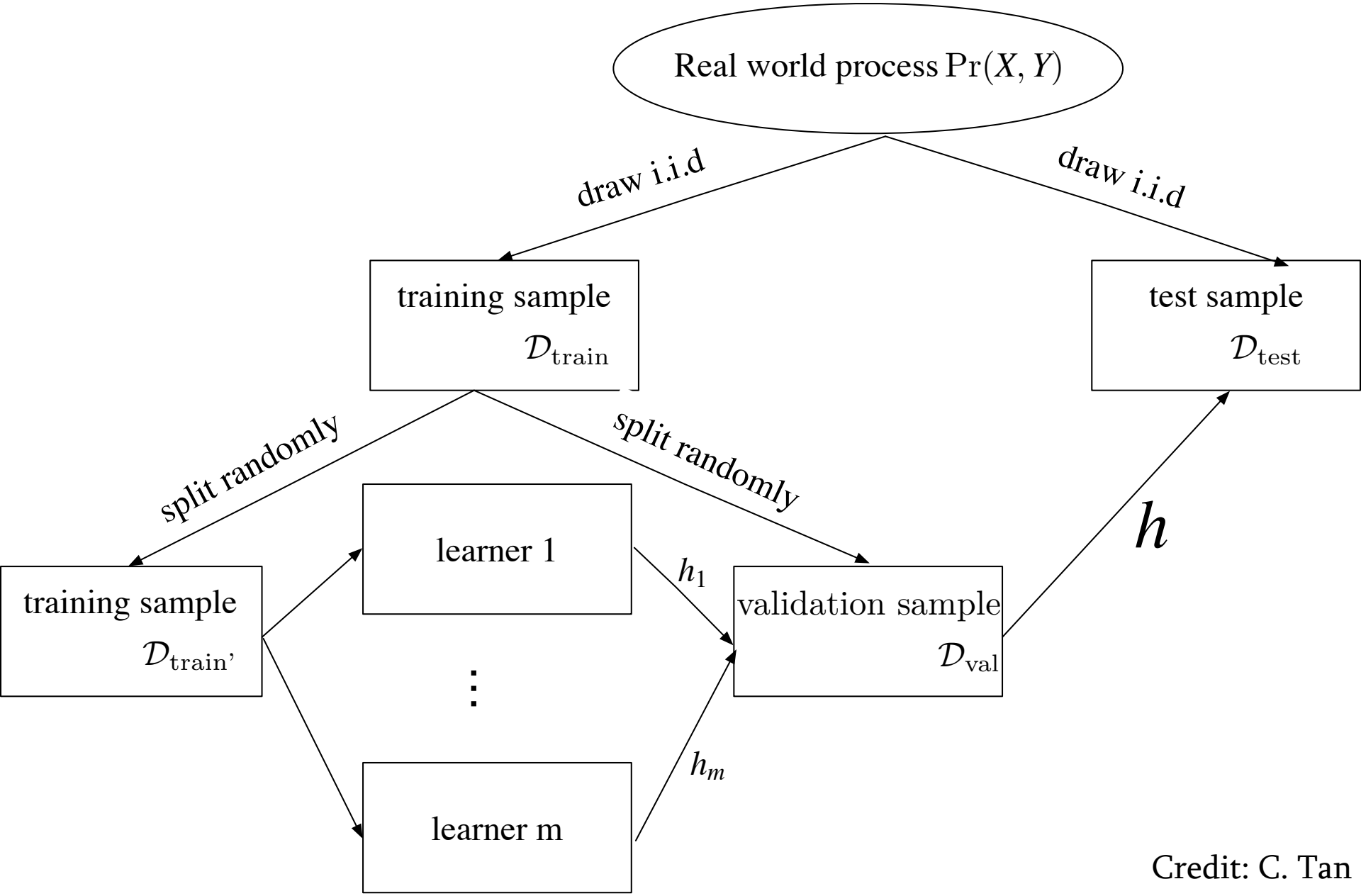
Credit: C. Tan

Bias-Variance Tradeoff



Credit:
http://scott.fortmann-roe.com/docs/BiasVariance.html

Model Selection



Credit: C. Tan

Managing data sets: Train v. Test

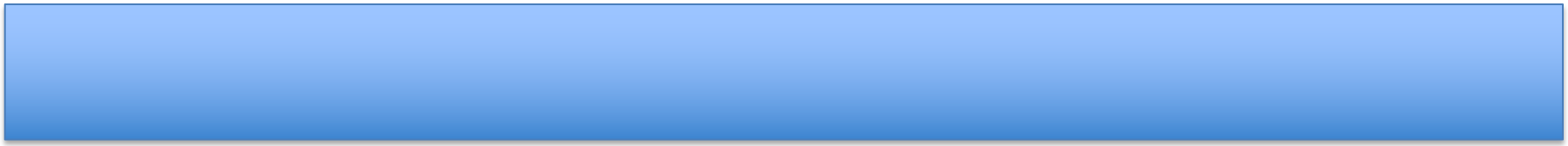
- In general, you should have a separate training set and test set.
- Ideally they will be drawn from the same distribution.
- E.g. randomly permute your entire input data set, then fix T, and use all (x_t,y_t) for $t < T$ for Train, and the remaining data for Test.

Managing data sets: Train v. Test

- First create the holdout data set, the Test set, for computing the final test error. For best performance it will be a (labeled) dataset from the same distribution as the Training set.
  - E.g. technique on previous slide.

- With the remaining data set, you can either divide it into two parts for Train and Validation, as specified on the previous slide, or you can run Cross-Validation on it.
  - This is for model selection / parameter tuning

- Whole data set:



- Separate the test set, and hold it out:

- From the remaining data, use part for Validation:

- Or run cross validation to train and fit parameters:

Cross validation: motivation

- To get a more robust estimate of the error rate, we should run more experiments, and average the error rates over the experiments.

- If we have a huge amount of labeled data, we can run several disjoint experiments (e.g. further split Train and Test into more data sets) and average the test error over the results. This is usually not the case.
  - Instead we do Cross-Validation.

N-fold Cross-validation

- Cross validation data set:

- Fold 1:

- Fold 2:

- …
- Fold n

N-fold cross-validation

- Fix N, for example 5 or 10.
- Run N experiments. In each experiment, 1/N fraction of the data is used for Test and the remaining data is used for Train. The Test sets over the N experiments are disjoint, and the Train sets are overlapping.
- Average the Test error over the N experiments.
- Note: different classifiers will likely be learned in each experiment, as the Train sets differ slightly.

Leave-one-out cross-validation

- N is set to the size of the data set. For each experiment, Test consists of the i-th point, and Train consists of all points but the i-th point.
- Average the test error over the N experiments.

Model selection / Parameter tuning

- For each meta-parameter, e.g. *k* in *k*-NN, explore a range of parameter values. Log-search can be performed by doubling, or halving the value of the parameter.

- On the tuning data set (a.k.a. Validation set, or CV on the training data), try to find the "knee" in the curve: the parameter setting such that the tuning test error is minimized, before increasing again.