Machine Learning: an Introduction

Sergio A. Alvarez
Computer Science Department
Boston College

INTRODUCTION

Machine learning

- Is the study of computational mechanisms that adapt based on experience, improving their performance on a target task over time.

  Mitchell, 1997 (paraphrased)

- Is very useful in data analysis (data mining), to
  - uncover previously unknown patterns
  - build predictive models that generalize well

Bioinformatics applications of ML (Prompramote et al, 2005)

Research Area	Application	Reference
Sequence alignment	BLAST	http://www.ncbi.nlm.nih.gov/BLAST/
	FASTA	http://www.ebi.ac.uk/fasta33/
Multiple sequence alignment	ClustalW	http://www.ebi.ac.uk/clustalw/
	MultAlin	http://prodes.toulouse.inra.fr/multalin/multalin.html
	DiAlign	http://www.genomatix.de/cgi-bin/dialign/dialign.pl
Gene finding	Genscan	http://genes.mit.edu/GENSCAN.html
	GenomeScan	http://genes.mit.edu/genomescan/
	GeneMark	http://www.ebi.ac.uk/genemark/
Protein domain analysis and identification	Pfam	http://www.sanger.ac.uk/Software/Pfam/
	BLOCKS	http://www.blocks.fhcrc.org/
	ProDom	http://prodes.toulouse.inra.fr/prodom/current/html/ home.php
Pattern identification	Gibbs Sampler	http://bayesweb.wadsworth.org/gibbs/gibbs.html
	AlignACE	http://atlas.med.harvard.edu/cgi-bin/alignace.pl
	MEME	http://meme.sdsc.edu/meme/website/intro.html
Protein folding prediction	PredictProtein	http://www.embl-heidelberg.de/predictprotein/ predictprotein.html
	SwissModel	http://www.expasy.org/swissmod/SWISS-MODEL.html

Popularity of ML techniques

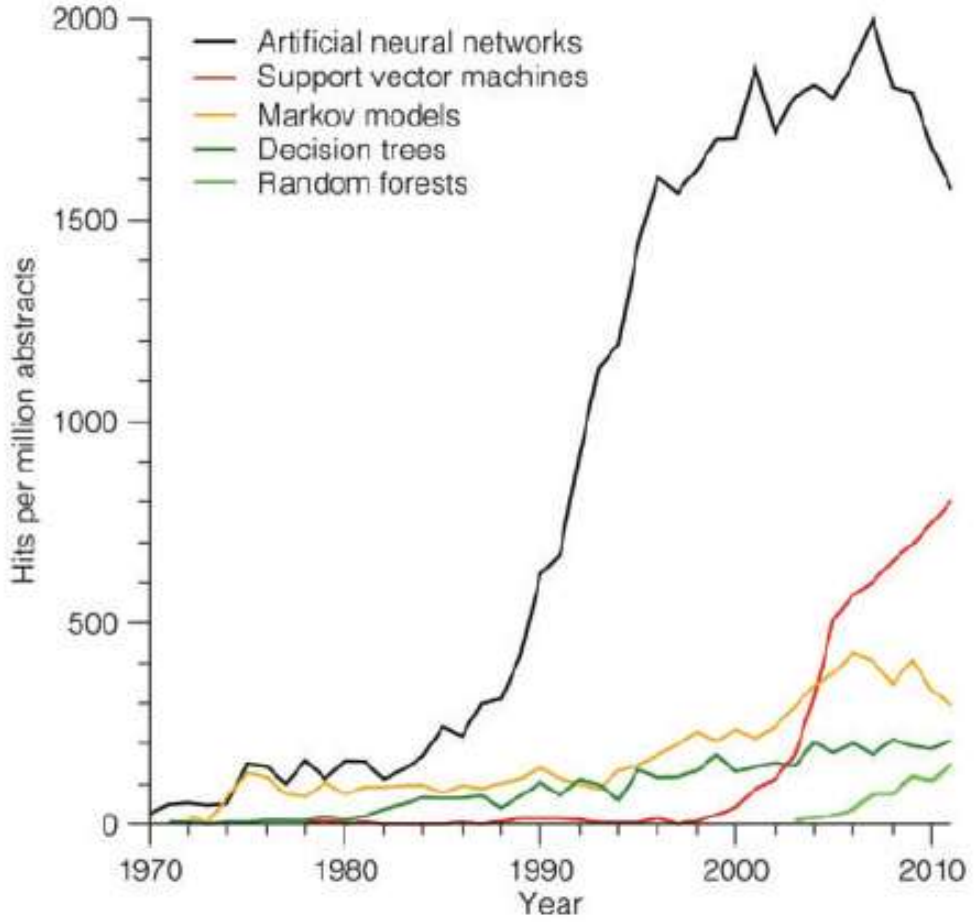


Fig. 1. The growth of supervised machine learning methods in PubMed.

Jensen & Bateman (2011), *Bioinformatics* 27(24): 3331-3332

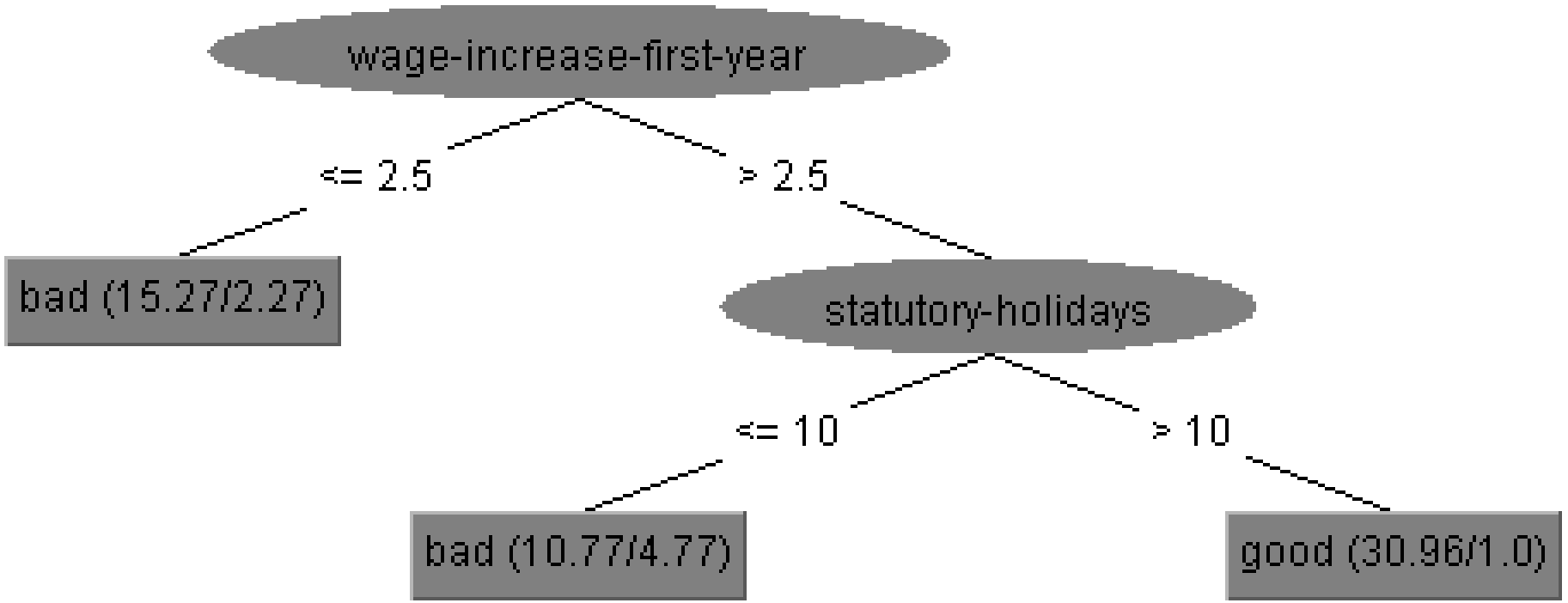
Outline of talk

- Basic concepts
  - Instances and features
  - Input and hypothesis spaces
  - Error and generalization
  - Model vs data complexity
  - Meta-learning
- Preprocessing
  - Attribute selection
  - Feature extraction
- Supervised learning: classification and regression
  - Decision trees
  - Neural networks: gradient descent, hidden representations
  - Instance-based learning
  - Bayesian techniques
  - Support Vector Machines, kernels

BASIC CONCEPTS IN MACHINE LEARNING

Supervised machine learning ("with a teacher")

- Experience for learning takes form of data set of labeled training examples $(x, \hat{y})$



Supervised machine learning tasks

- *Classification* if $\hat{y}$ comes in discrete categories
  - Will patient x get sick ($\hat{y}$ = healthy or $\hat{y}$ = sick)?
  - Is flower of type virginica, versicolor, or setosa?
  - Is nucleotide sequence x coding or non-coding?
- *Regression* if $\hat{y}$ is a continuous target
  - Estimate snowfall $\hat{y}$ based on geographic location x
  - Predict the expression level $\hat{y}$ of x

Unsupervised machine learning ("class discovery")

- Experience consists only of unlabeled training data x (evidence only, no ``answers'')

  Goal is to uncover relationships among examples

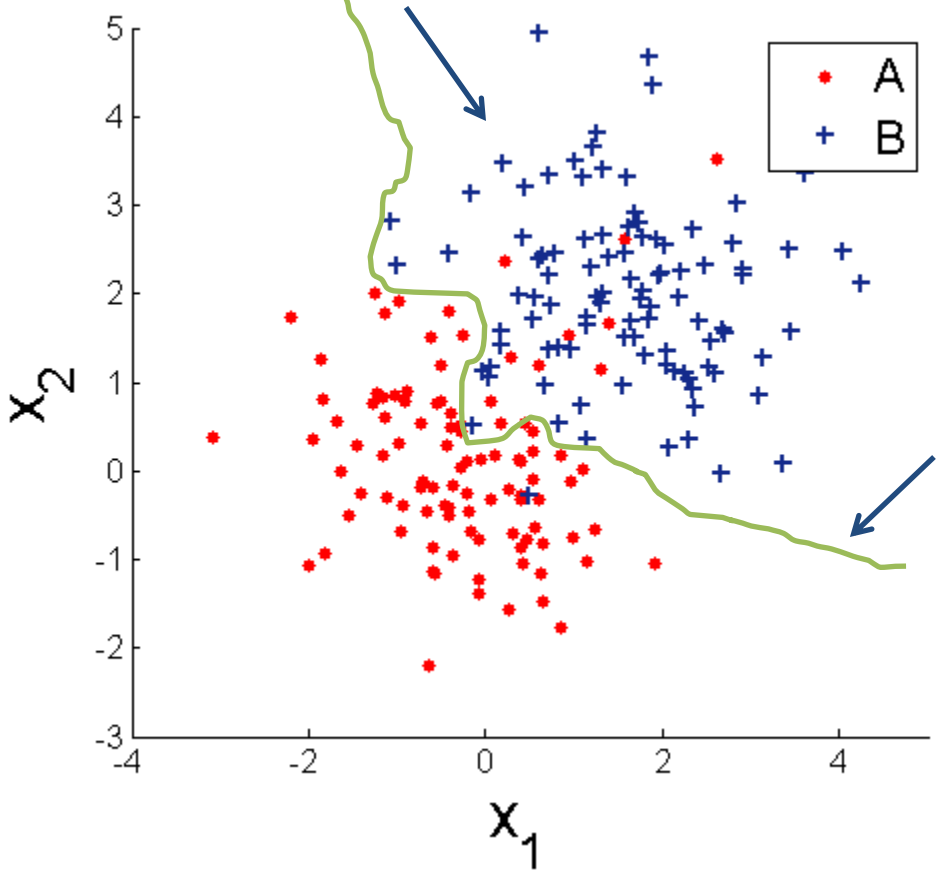
- *Clustering* aims to group examples by similarity
  - Group sequences x by expression pattern

Data, instances, attributes

- Each data example, x, is an *instance*
- Instances x described by *attributes* $x_1, x_2, \ldots, x_n$
  - predictive attributes (evidence)
    - Such as micro-array expression levels
    - Or sepal and petal measurements
  - target attribute is separate (where applicable)
    - Such as type of leukemia
    - Or type of iris flower
- Attribute quality is crucial to ML success

The input space

Examples viewed as points in space



Classifier viewed as decision surface

Simple error metrics

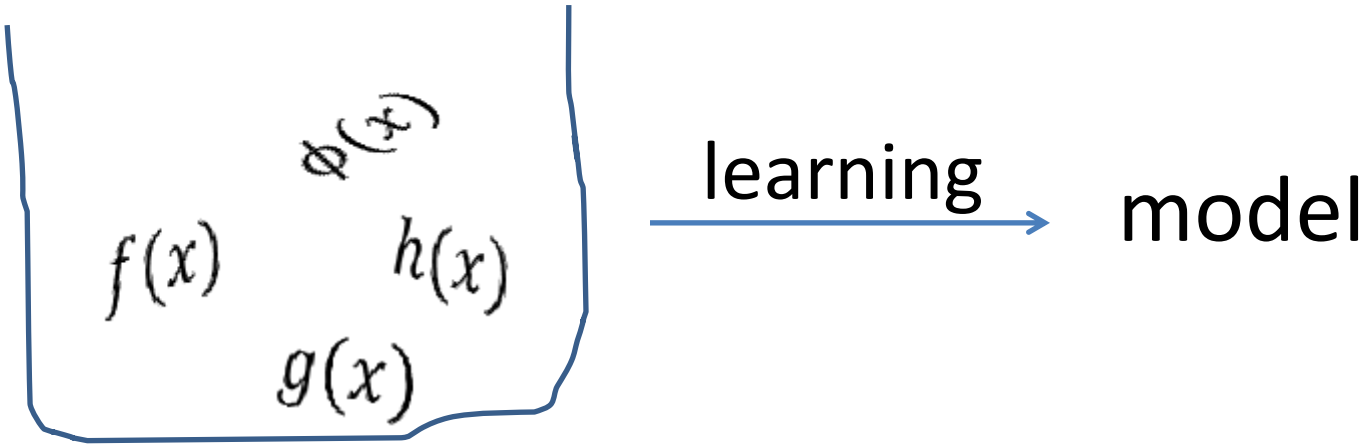
- Classification

$$error\ rate = \frac{\#\ instances\ incorrectly\ classified}{total\ \#\ instances}$$

- Regression

$$mean\ squared\ error = \frac{1}{n}\sum_{\{k=1\}}^{n}(y_k - \hat{y}_k)^2$$

The hypothesis space



Learner type determines hypothesis representation

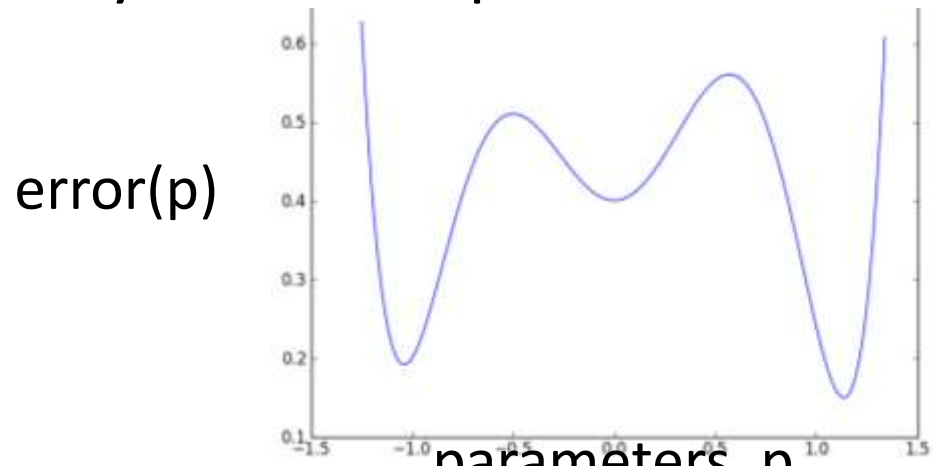
(e.g., rules, decision diagrams, continuous functions)

For given type, hypotheses share a basic structure

but differ in details (e.g., parameter values)

Training a machine learner

- Input: dataset $D = \{(x^1, \widehat{y^1}), \ldots, (x^n, \widehat{y^n})\}$
- Output: element M of hypothesis space, based on D, to be used as a predictive model
- Procedure: depends on learning technique, but seeks deep valleys in model parameter landscape



Testing a predictive model M on labeled data

- Input: predictive model M, test dataset D of labeled instances $(x, \hat{y}(x))$
- Output: predicted labels $y = M(x)$ for all instances x in D, and associated error rate

$$e_M(D) = \frac{\#\ of\ instances\ x\ in\ D\ for\ which\ M(x) \neq \hat{y}(x)}{total\ \#\ instances\ in\ D}$$

Generalization

- Training datasets are just *samples*

- Desire least error across population (unseen)
  - Known as *generalization error*

Estimating the generalization error

- Input: predictive model M, dataset D of labeled instances $(x, \hat{y}(x))$
- Output: estimate of generalization error rate of M

$$g(M) = \lim_{n \to \infty} \frac{\#\ of\ instances\ x\ for\ which\ M(x) \neq \hat{y}(x)\ in\ a\ size\ n\ sample}{n}$$

- Procedure: several approaches (next few slides)

Estimate 1: use the training error

Use error rate on training dataset D as estimate of generalization error

- Pros
  - Easy!
  - Uses all available data (important if data are scarce, reduces random variation due to sample size)
- Cons
  - Underestimates generalization error (serious problem)
  - High risk of overfitting unimportant details of sample

Estimate 2: training-test split

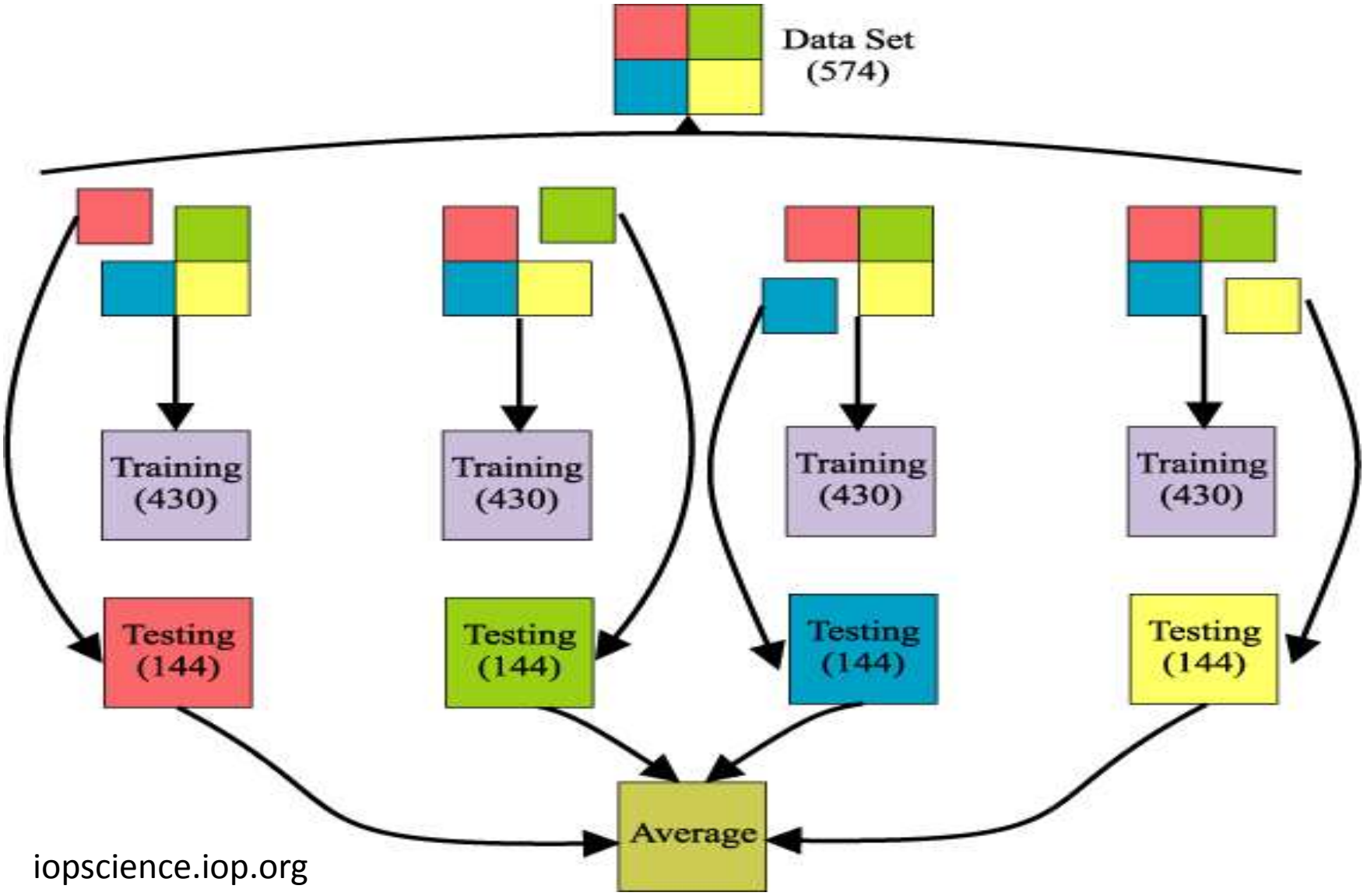
Split D randomly into disjoint portions D_{train} and D_{test}

Train on D_{train} only

Use error rate on D_{test} as generalization estimate

- Pros
  - Better generalization estimate (test on unseen data)
- Cons
  - Reduces size of sample on which model is based, reducing quality of estimate
  - Results may depend sensitively on training / test split

Estimate 3: k-fold cross-validation



iopscience.iop.org

Components of the error

error = bias + variance

- Bias = erro … e to representa … sifier
- Variance = … riation in finite data …

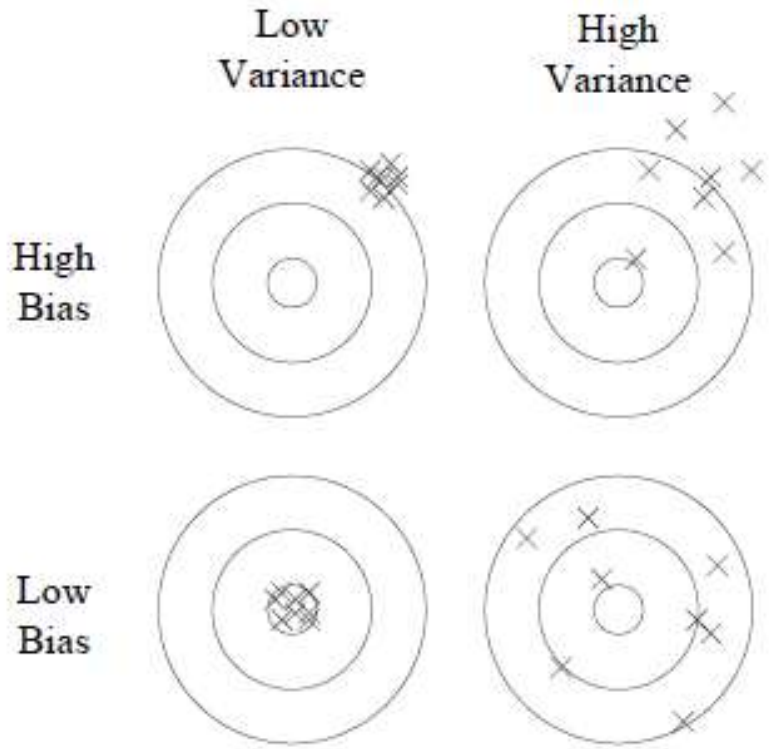
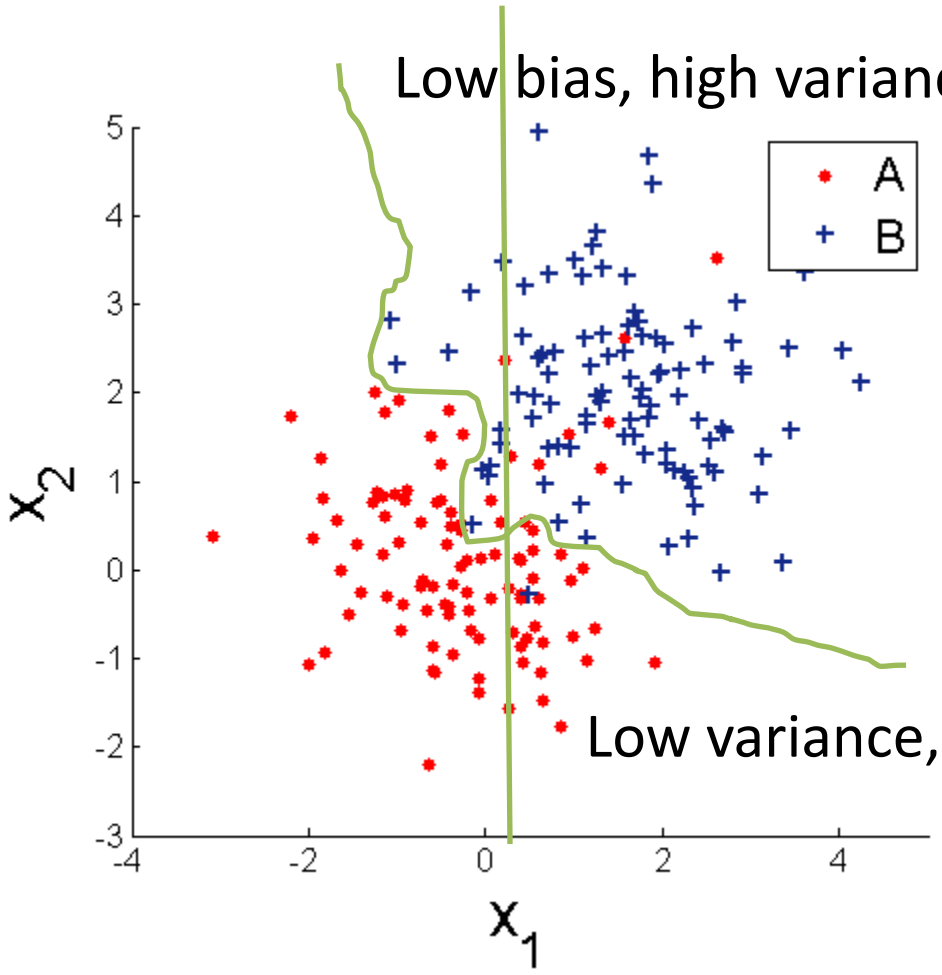


Figure 1: Bias and variance in dart-throwing.

P. Domingos. *Comm. ACM*, 55(10):78-87, 2012.

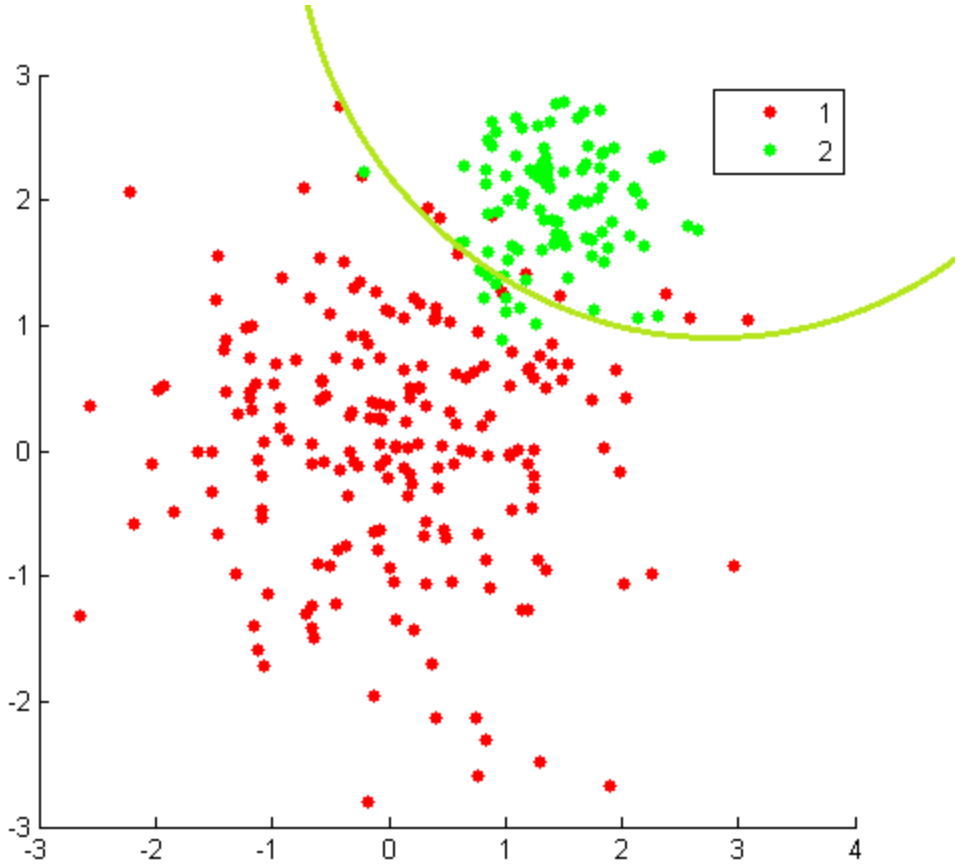
Bias-variance tradeoff



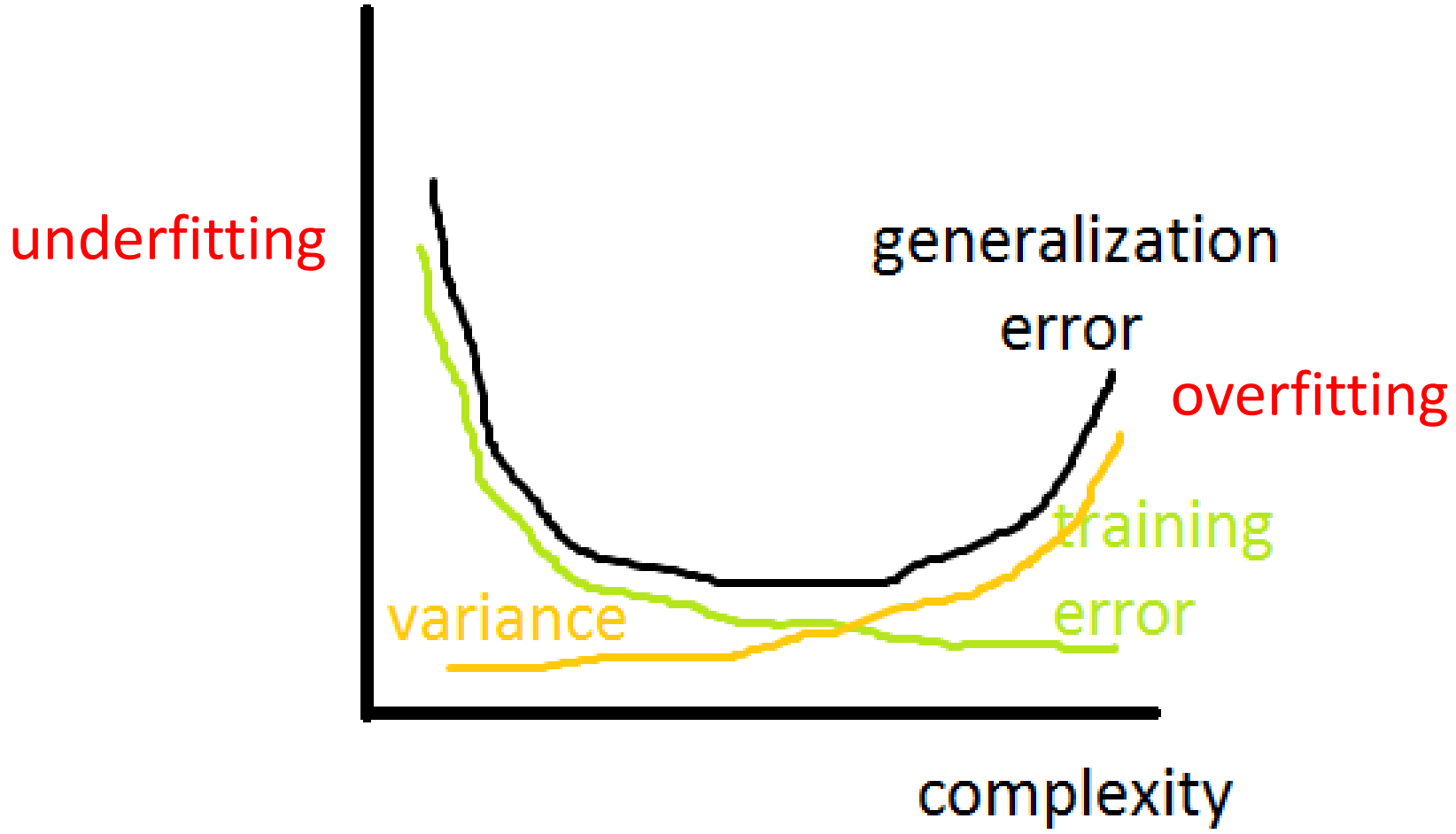
Model complexity and bias-variance tradeoff

- Complexity of predictive model may control a tradeoff between bias and variance
  - High model complexity: low bias, high variance
  - Low complexity: high bias, low variance
  - Best performance often occurs near “crossover” point
    - Good match between model and data complexity

Balancing model complexity and data complexity



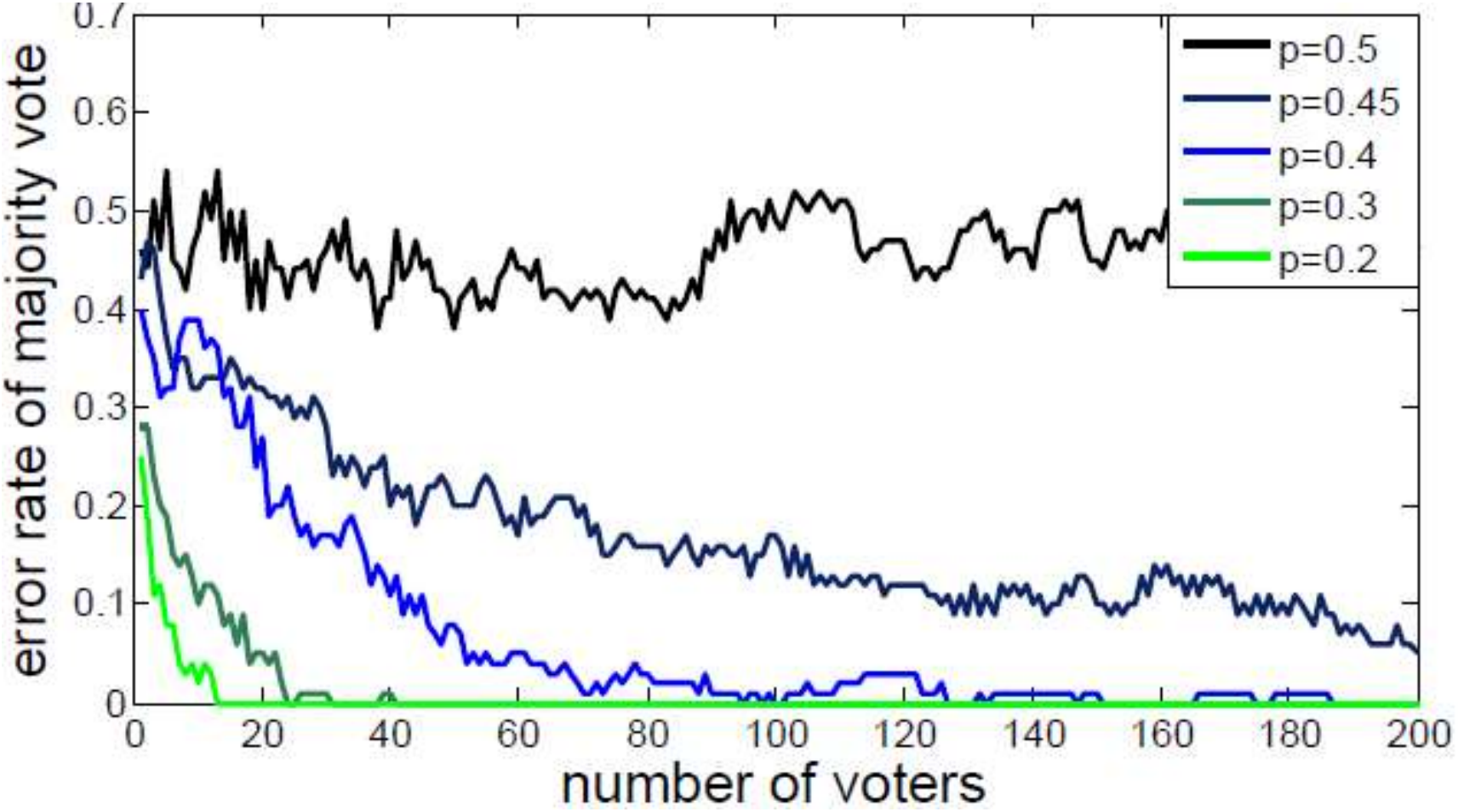
Model complexity and generalization error



Minimum Description Length

- Add model complexity to error metric to estimate generalization error
  - The “description length” is the number of bits needed to encode the dataset given a model:
  - DL = L(model) + L(data | model)
  - First term on right represents model complexity
  - Second term is model fit error (training error)
- Minimum Description Length (MDL) Principle
  - Models with smallest DL should be preferred

The wisdom of crowds



Bagging (Bootstrap AGGregatING)

- Uses repeated sampling with replacement to generate multiple models to be combined
- Aims to reduce variance component of error by simulating a larger data sample

for each i in the range $1 \ldots n$

Generate a bootstrap sample D_i of D of the same size as D by sampling from D with replacement

Train a model M_i ("weak learner") on D_i

predict by plurality vote among the models M_i

Example

- Labor dataset, trees vs bagged decision stumps

Boosting

- Weights instances to focus model search on "harder" cases
- Attempts to address bias component of error
- Input: labeled dataset D, max iterations T, learners $L_1 \ldots L_T$
- Output: a classification model M based on D
- Procedure:

  assign weight of 1 to each instance in D

  for t = 1 to T

    sample D_t from D (w. replacement), likelihood proportional to weight

    train t-th learner L_t on D_t, yielding model M_t; record error e_t

    if e_t is 0 or more than 0.5, break

    for each instance I in D correctly classified by M_t:

      multiply weight(I) by $e/(1-e)$ (less likely to be picked)

Boosting: classification procedure

Given a test instance x:

for each class
initialize weight(class) to 0

for t over valid model indices (some may have been skipped)
let c = class predicted by M_t on input x
add $-\log(\frac{e_t}{1-e_t})$ to weight(c)
(greater amount the more accurate the model)

return $argmax_c$ weight(c)

Example

- AdaBoost on labor dataset

DATA PRE-PROCESSING

Easy pre-processing steps that can help

- Identify and eliminate clear outliers
- Address data instances with missing values
  - Due to measurement error
  - Possibly replace with modal or mean values
- Make attributes comparable
  - Height in m, age in years are on different scales
  - Normalize to range [0,1], or standardize $Z = \frac{X-\mu}{\sigma}$

Discretization

- Some predictive techniques cannot handle continuous values
- Discretization converts continuous attributes to discrete attributes by binning

Unsupervised discretization

- Equal width bins, or equal frequency
- How many bins?

  Bias-variance calculation provides rough estimate

  bias $= O\left(\frac{1}{b}\right)$, variance $= O(\frac{1}{N/b})$

  total $= \frac{1}{b^2} + \frac{b}{N}$ (bias squared to make comparable)

  minimum when $\frac{d\,\text{total}}{db} = 0 \;\Rightarrow\; b = (2N)^{\frac{1}{3}}$

Supervised discretization

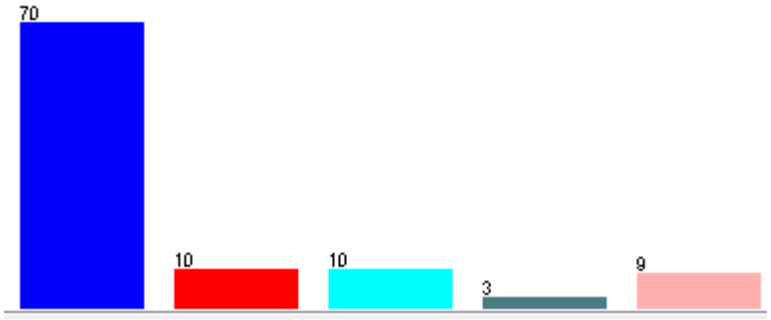
- Supervised approach
  - Class entropy-based discretization maximizes a measure of class homogeneity
  - MDL stopping prevents uncontrolled splitting by penalizing discretization model complexity

Measures of class inhomogeneity: class entropy

- Shannon entropy of probability distribution

$$H = -\sum_k p_k \log p_k$$

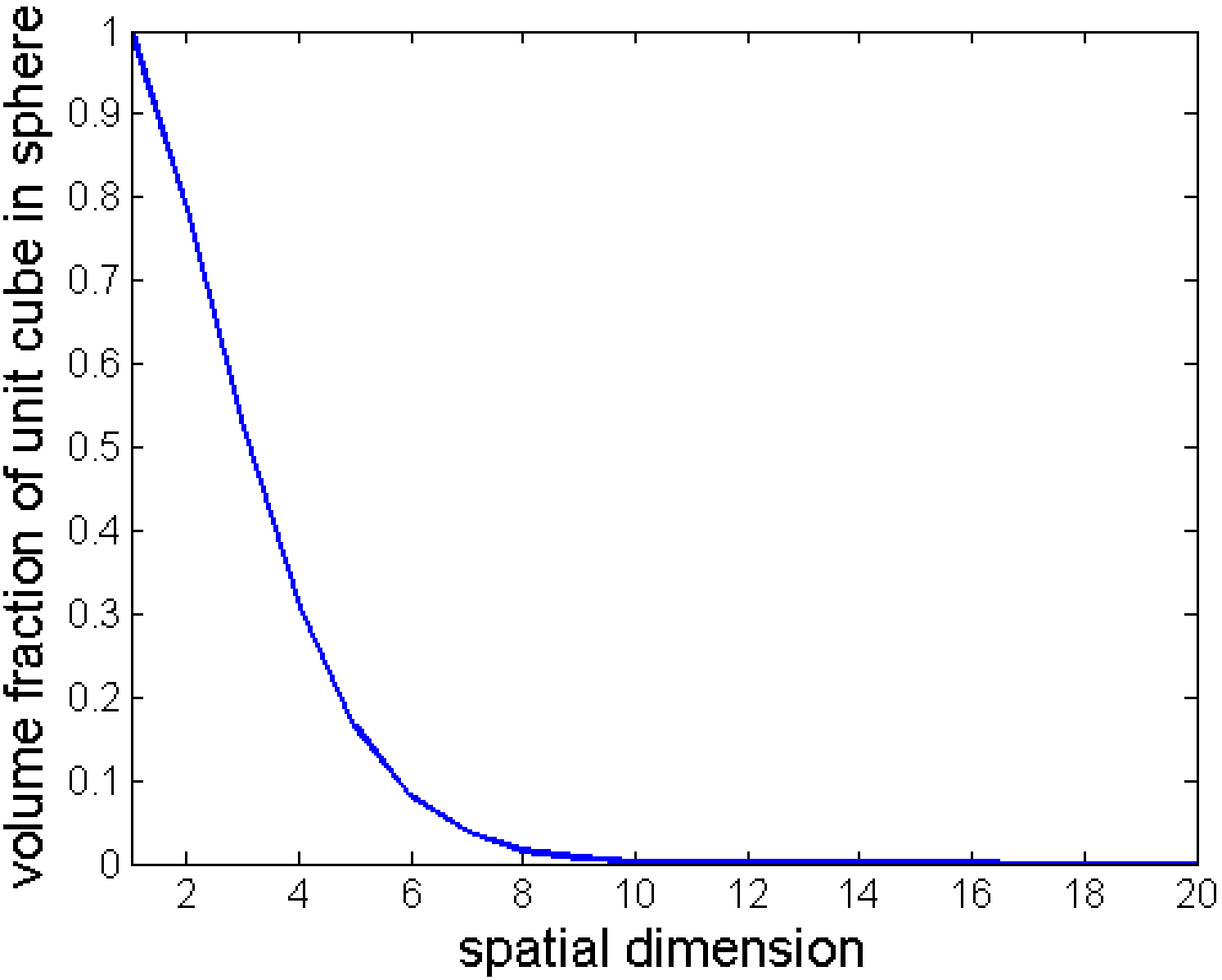
  - H is bits per symbol for optimal lossless code (Shannon, *Bell System Technical Journal* (27), 1948)
  - Use distribution of class labels as p
  - Bins with "greater class purity" have lower class enropy



$H = 1.49$ bits / symbol

Simple ML models can work very well

- 1-R classifier (Holte, 1993)
  - Given a labeled training dataset, D, the 1-R model M determines most discriminating attribute, a^*
  - Predicted label $M(x)$ for any instance x is the most common class among instances in D for which a^* has the same value as $a^*(x)$
- Holte showed that performance of 1-R rivals that of more complex techniques for many problems
- Moral of story: quality of data attributes is crucial


volume fraction of unit cube in sphere
spatial dimension
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
6
8
10
12
14
16
18
20

Attribute selection

- Finds a subset of attributes that is well suited to the predictive task at hand
- Reduces data dimensionality
- Can improve predictive performance
  - Example: ALL-AML leukemia data (Golub et al, 1999)

    7129 DNA micro-array attributes
    - Nearest neighbor classifier, full attribute set, 0.11 error rate
    - Same classifier, 36 attribute subset, 0.05 error rate

Attribute selection approaches

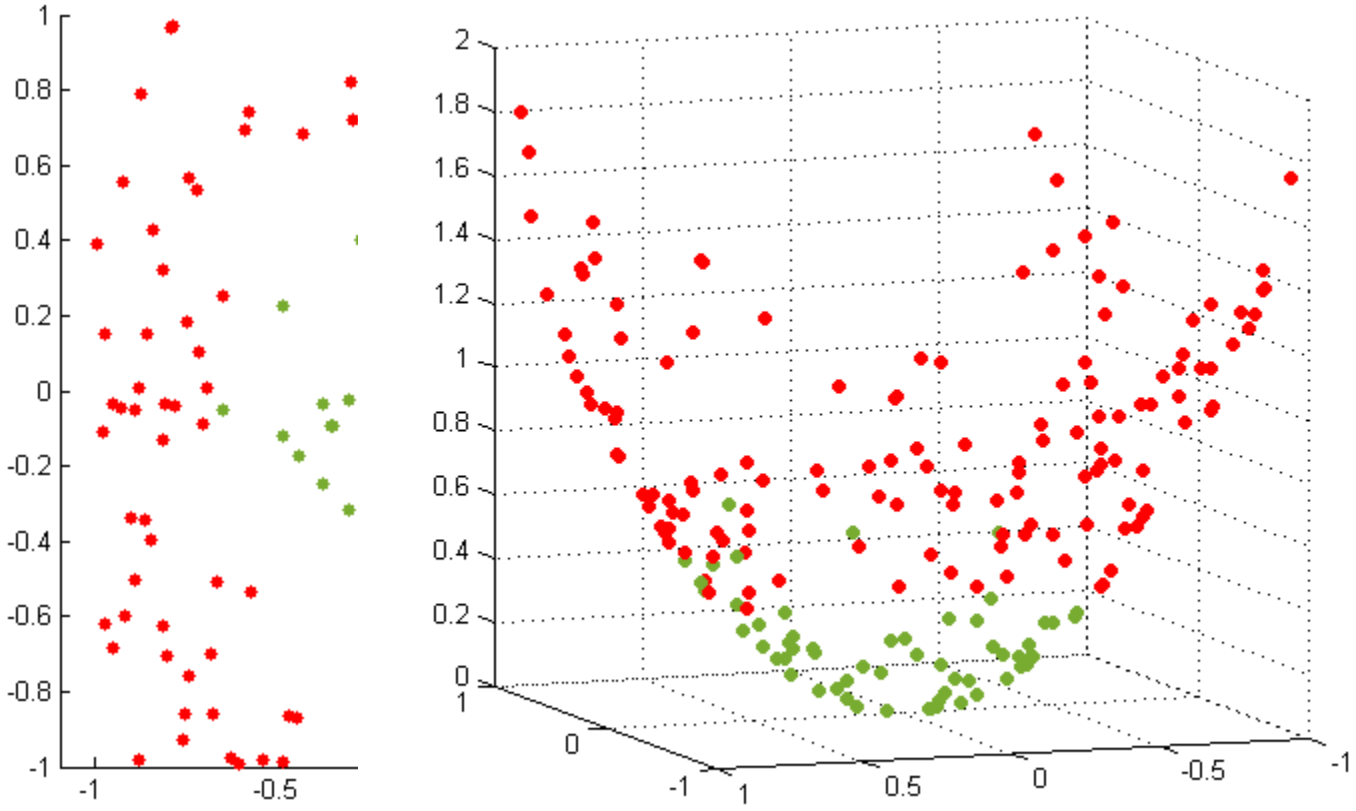
- Domain expertise
  - Priceless, use if available
- Ranking-based (e.g., InfoGain, ReliefF)
  - Grade individual attributes based on class discrimination power
- Subset evaluation (e.g., CFS)
  - Consider sets of attributes collectively
  - Use intrinsic metrics, or wrapper approach (actual classification performance of selected attributes)

Feature extraction

- Involves creation of new attributes from old
  - For example, extract BMI from weight and height
- Can lead to more informative data description
- Can improve predictive performance
- Can reduce dimensionality

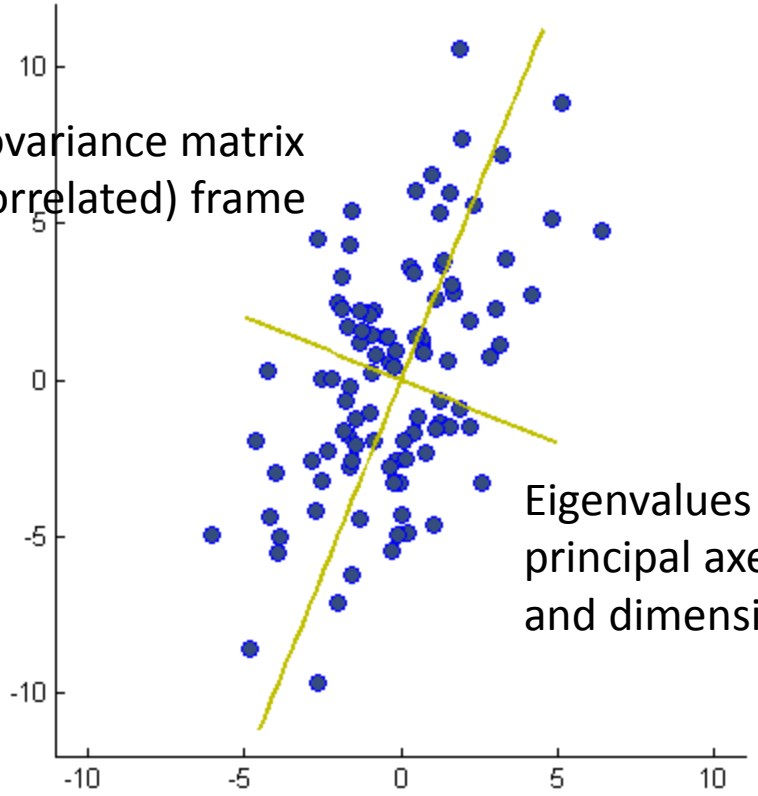
Can these classes be separated by a linear boundary?

Let $z = x^2 + y^2$



Principal Components Analysis (PCA)

Eigenvectors of data covariance matrix form orthogonal (de-correlated) frame



Eigenvalues are variances along principal axes, allowing ranking and dimensionality reduction

Example: digits

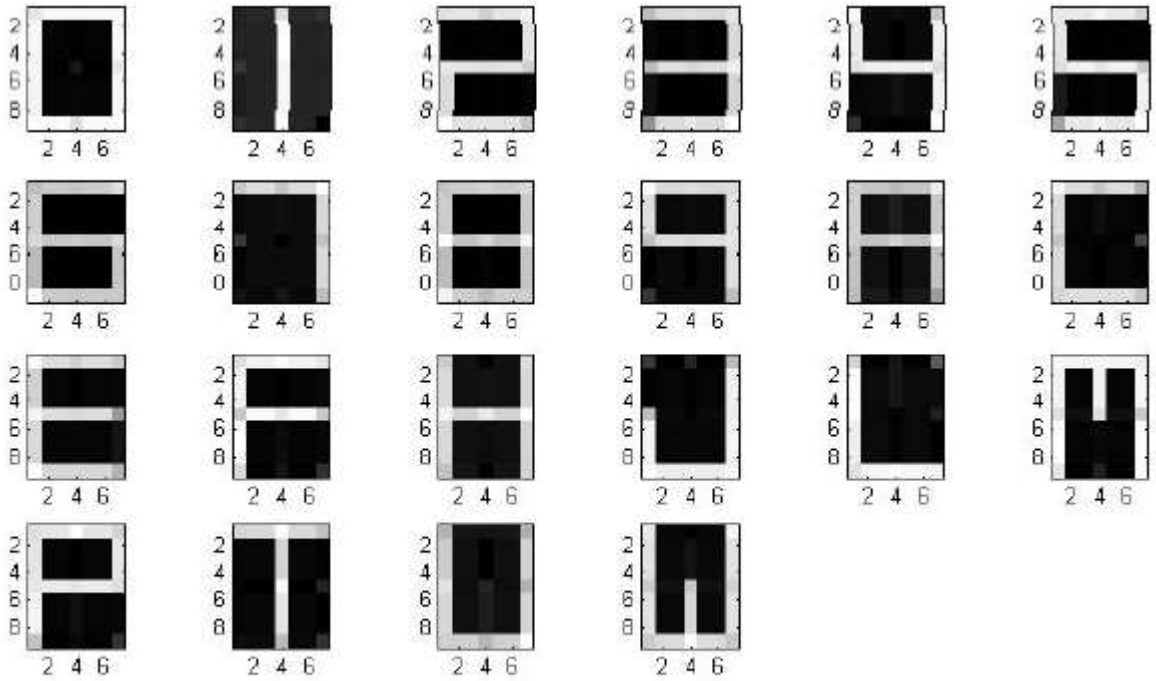
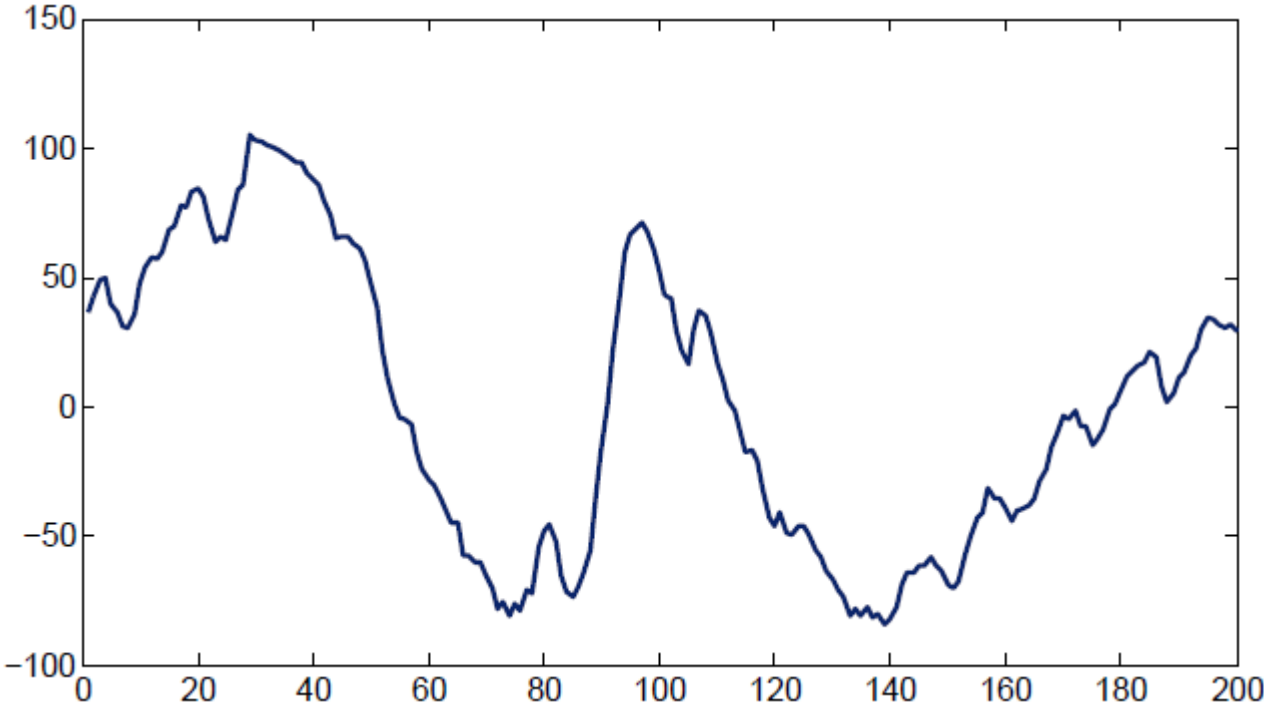
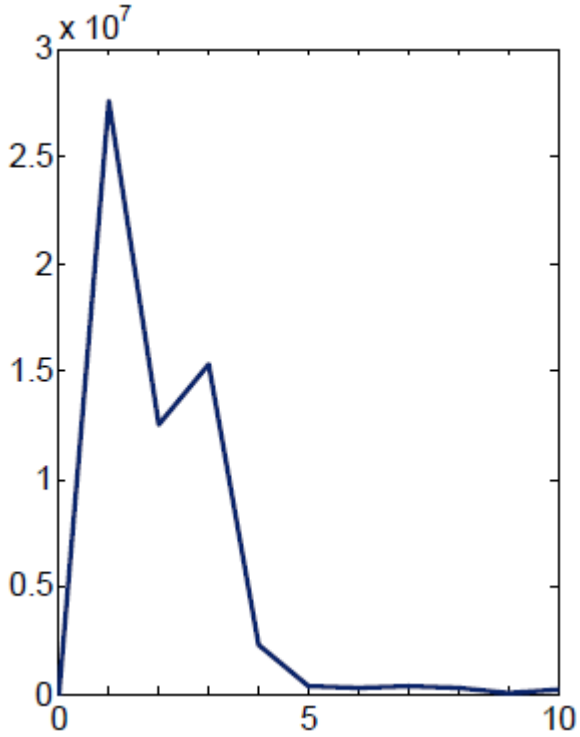


Figure 2: Reconstructions using top 10 eigenvectors

Discrete Fourier Transform (DFT)



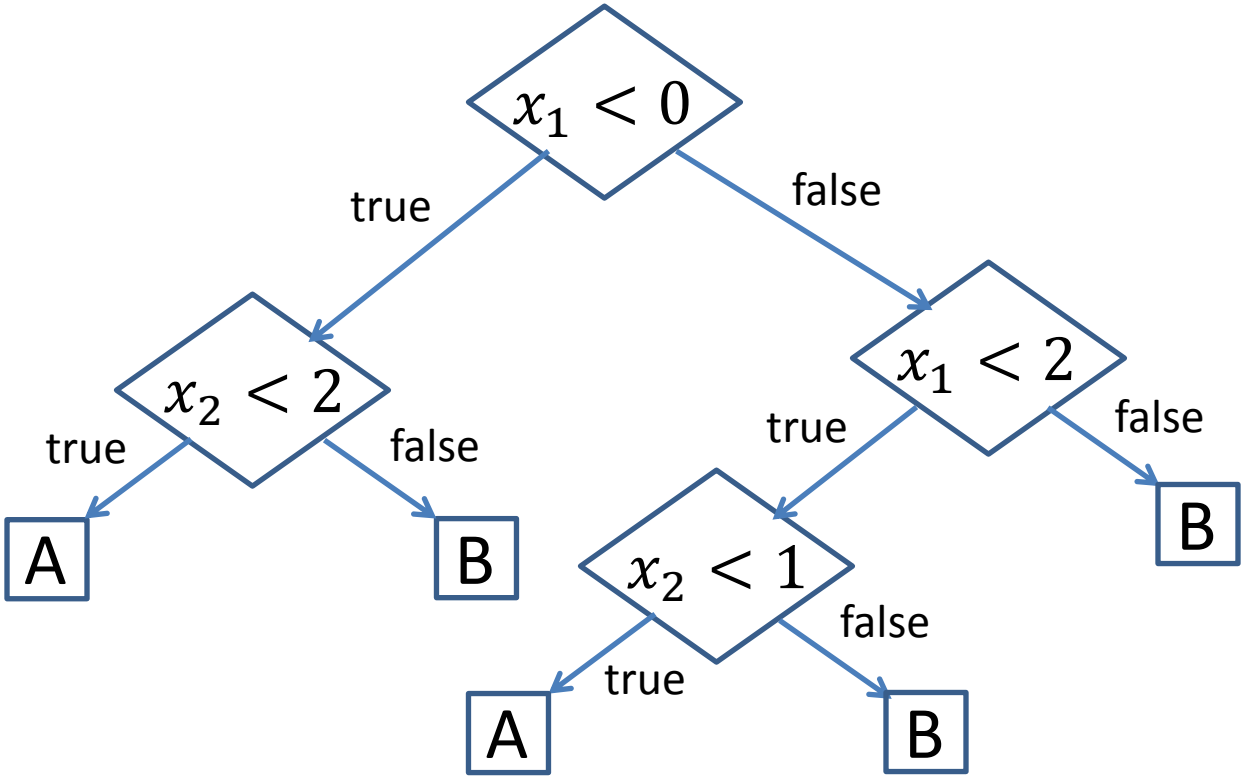
Sleep EEG signal as a function of time



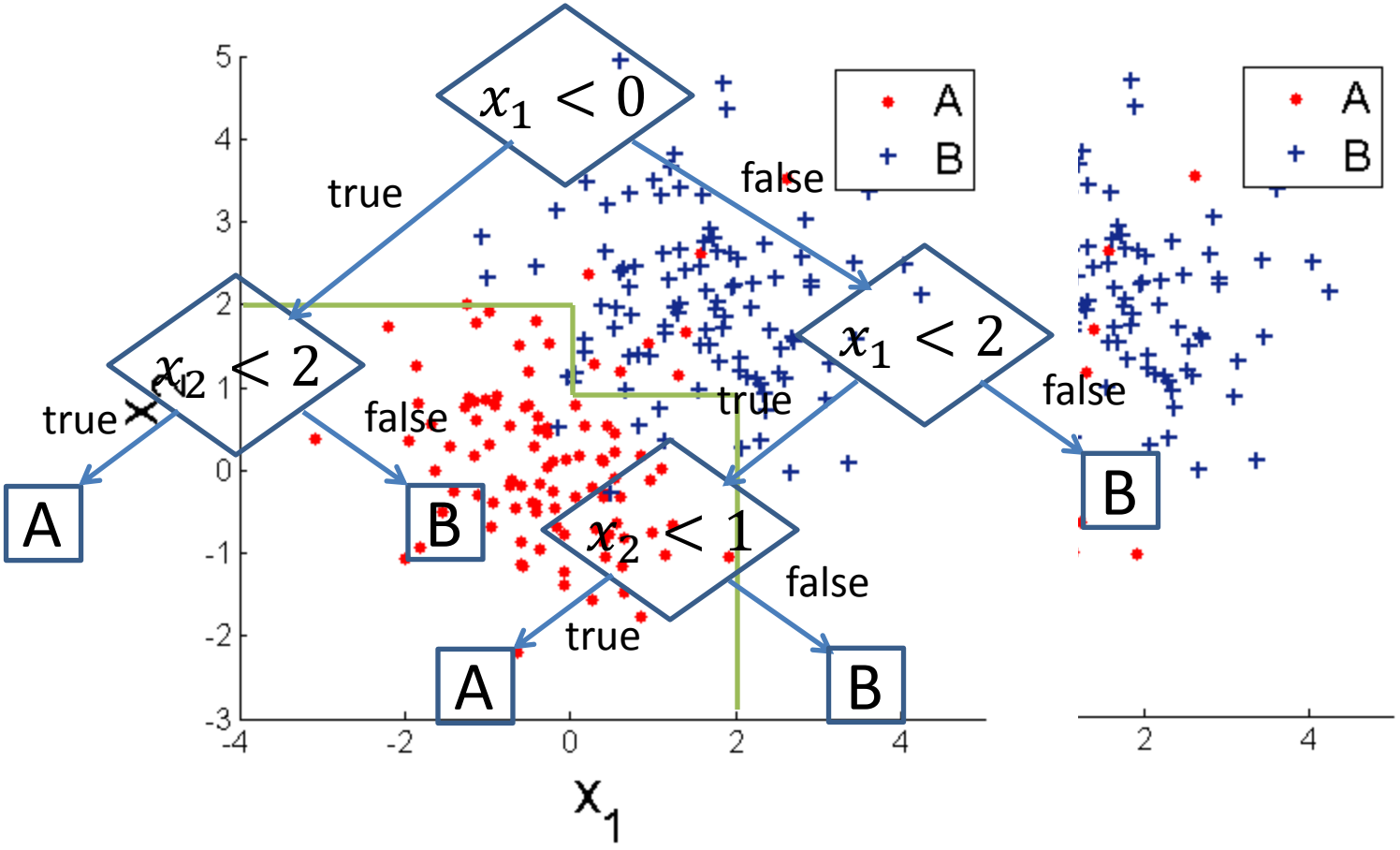
Corresponding frequency content

DECISION TREES

Sample decision tree



Decision tree decision boundaries



Basic decision tree learning algorithm ("iterative dichotomization" (ID), Quinlan)

- Input: labeled training dataset, D
- Output: decision tree classifier
- Procedure

  if D contains single class c, or stopping condition met

  return a single decision node that predicts class c

  a = attribute that splits D most class-homogeneously

  $D_1, D_2, \ldots, D_k$ = resulting partition of D according to a

  $T_1, T_2, \ldots, T_k$ = decision trees corresponding to D_i (recursion)

  $r(a) =$ internal node that tests attribute a

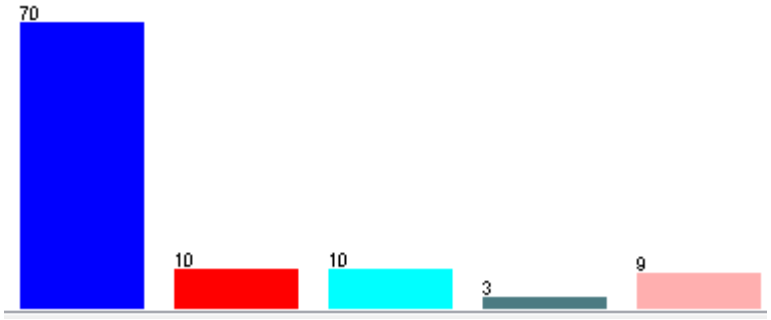
  return decision tree with root $r(a)$ and children $T_1, T_2, \ldots, T_k$

Measures of class inhomogeneity: class entropy

- Shannon entropy of probability distribution

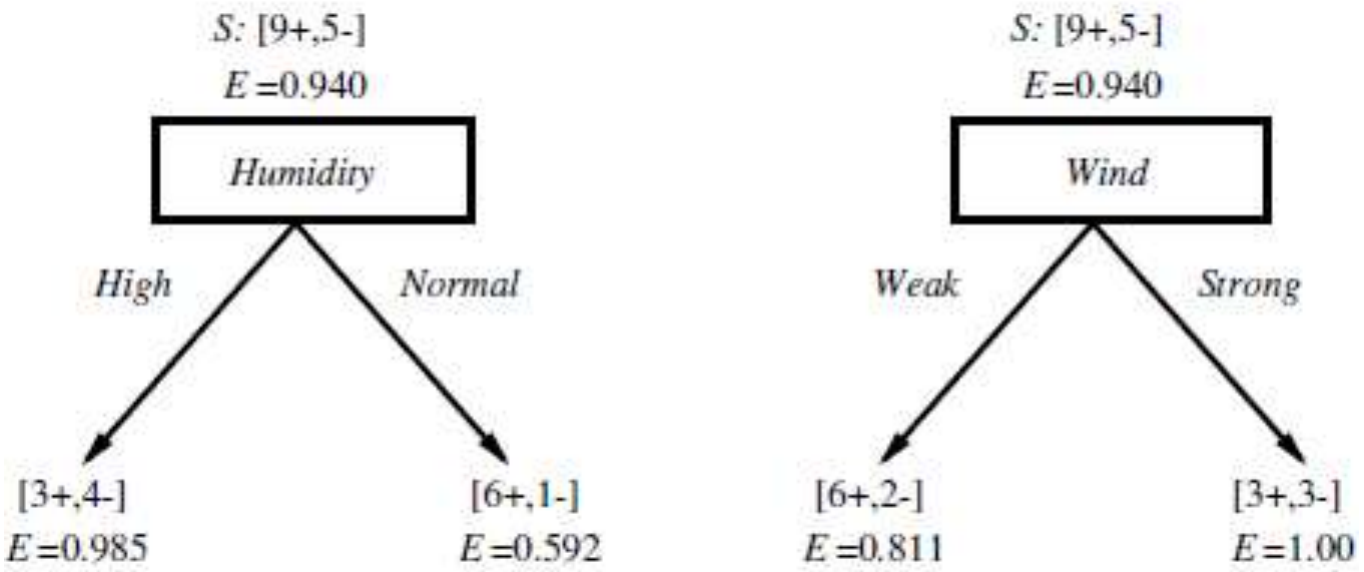
$$H = -\sum_k p_k \log p_k$$

  - H is bits per symbol for optimal lossless code (Shannon, *Bell System Technical Journal* (27), 1948)
  - Use distribution of class labels as p



$H = 1.49$ bits / symbol

Class entropy-based splitting



T. Mitchell

When to stop splitting?

- Decision tree learning is prone to overfitting
  - Early stopping helps (e.g., monitor DL)

- Overfitting can be reduced by pruning
  - use statistical error bounds or MDL principle

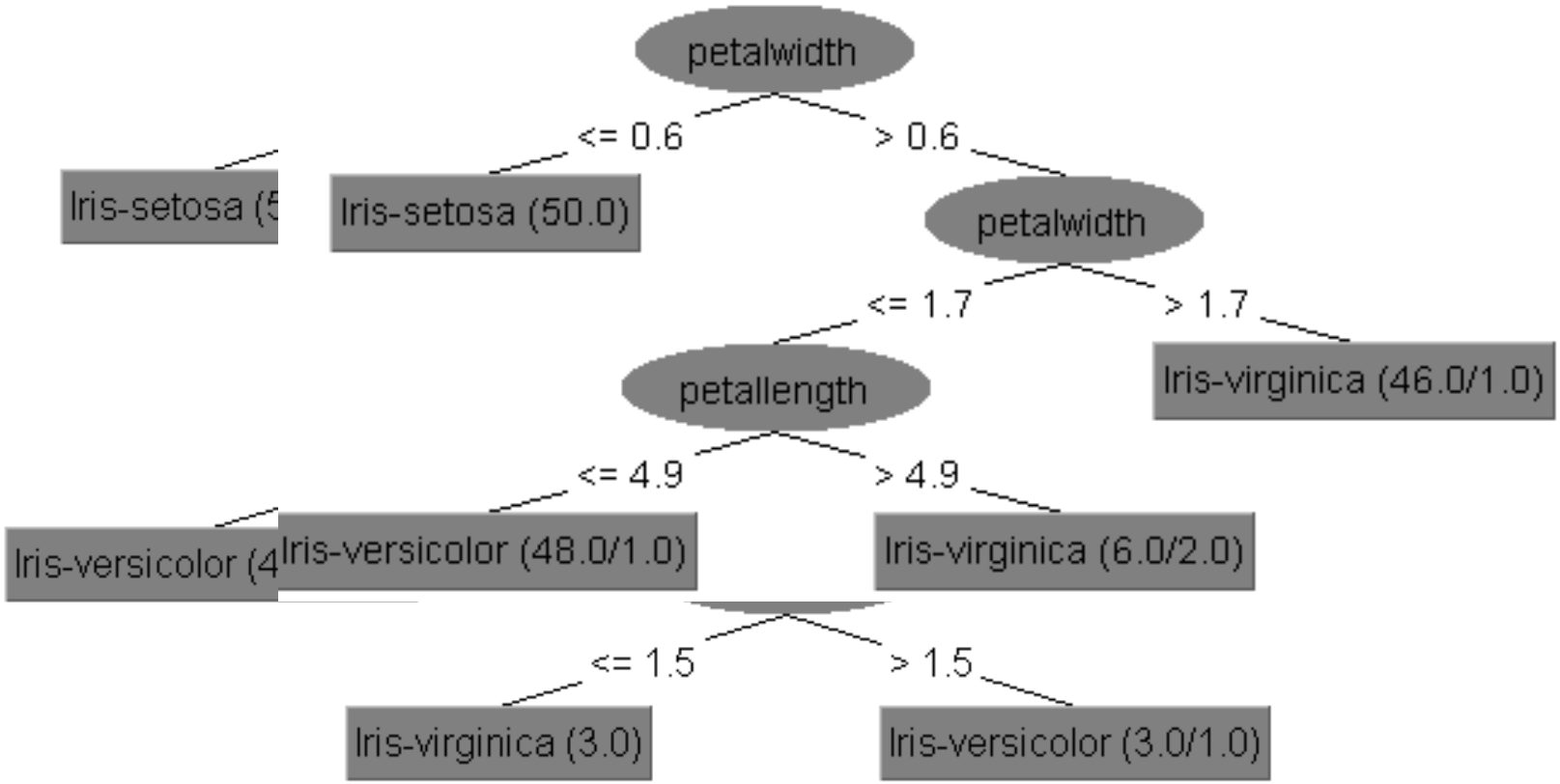
Tree pruning

- Can be based on binomial upper bound for generalization error in terms of training error, e

$$\frac{e + \frac{z^2}{2n} + z\sqrt{\frac{e(1-e)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where z is Gauss critical value for desired confidence

Decision tree for Fisher Iris dataset in Weka



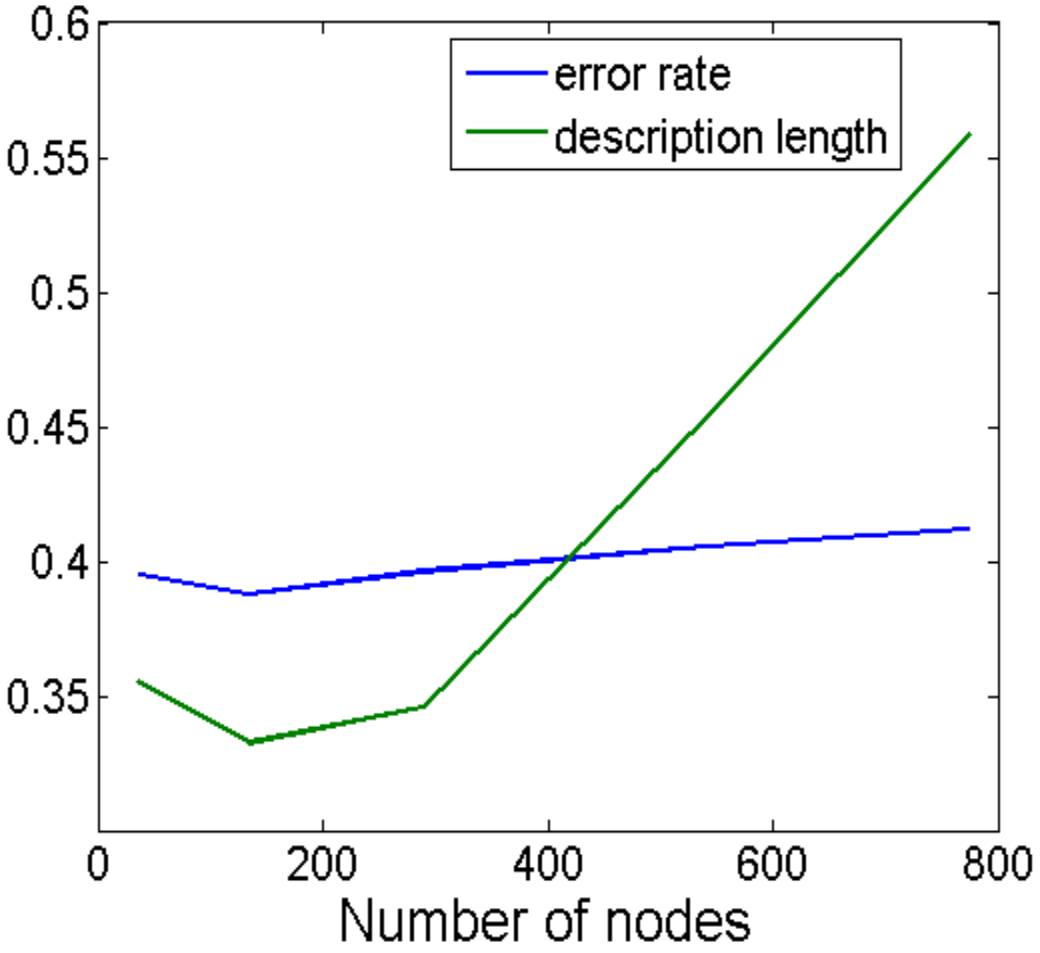
Random forests

- Bagging with trees as individual models, using a random subset of attributes for each tree
  - Randomization of features aims to reduce correlation among individual models in ensemble

Description length for decision trees

- Assume c classes, a attributes, n nodes
- Description length of model only
  - To code a decision node (leaf)
    - Just specify what class, need $\log c$ bits
  - To code an attribute test node (internal)
    - Specify attribute in $\log a$ bits, location in $\log n$ bits
- Description length of training data given model
  - To code a training error
    - Specify instance in $\log I$ bits, class in $\log c$ bits

Decision tree size and performance



MORE ABOUT EVALUATION OF SUPERVISED MACHINE LEARNING

Basic evaluation metrics

- Classification

$$error\ rate = \frac{\#\ instances\ incorrectly\ classified}{total\ \#\ instances}$$

$$accuracy = \frac{\#\ instances\ correctly\ classified}{total\ \#\ instances} = 1\ - error\ rate$$

- Regression

$$mean\ square\ error = \frac{1}{n}\sum_{\{k=1\}}^{n}(y_k - \hat{y}_k)^2$$

Confusion matrices

- Provide a breakdown of errors by class label

A	B	← Classified as
25	8	Actual A
3	20	Actual B

Error rate / accuracy can be misleading

- Is a classification error rate of 1% good?
  - Correctly predicting gender 99% of the time in a general population is excellent
  - Correctly predicting one of the blood type labels {AB-, other} 99% of the time is less impressive (prevalence of blood type AB- is 0.6% in the US)

Alternative error metrics (two-class case)

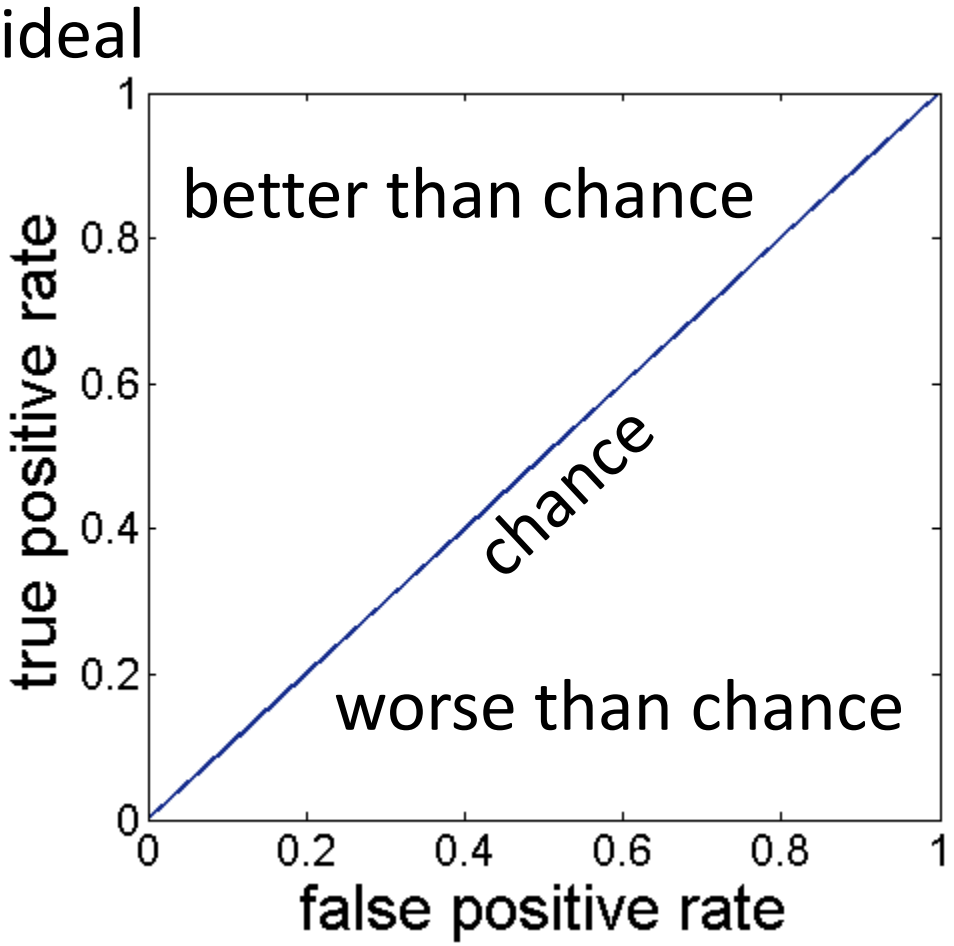
- False positive and negative rates (lower is better)

$$\text{fpr} = P(predict + \mid actual -)$$
$$\text{fnr} = P(predict - \mid actual +)$$

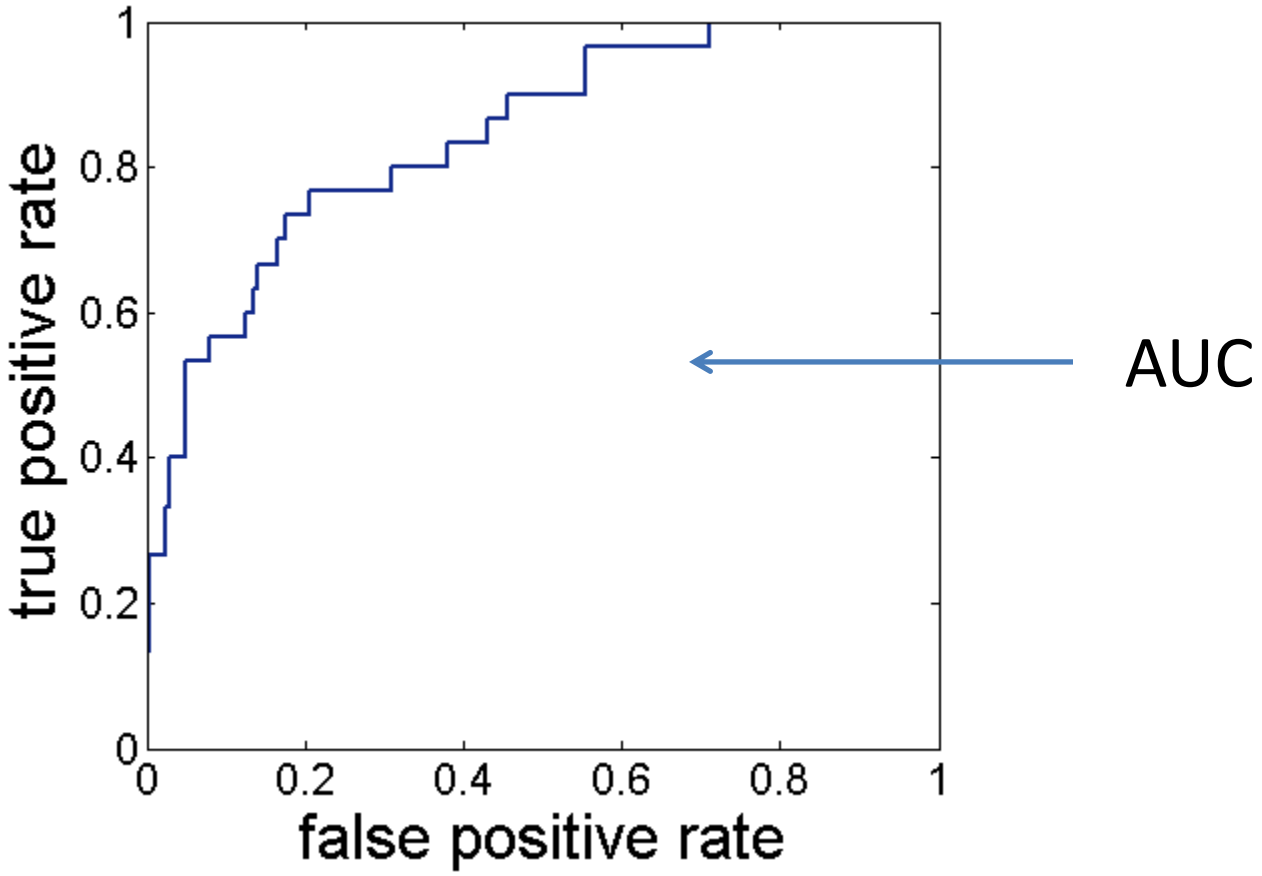
- Information retrieval metrics (higher is better)

$$\text{Precision} = P(actual + \mid predict +)$$
$$\text{Recall} = P(predict + \mid actual +)$$
$$F = \frac{2\ \text{Precision Recall}}{\text{Precision} + \text{Recall}}$$

ROC (Receiver Operating Characteristic)



ROC (Receiver Operating Characteristic)



ROC operating point optimization

- Two types of errors

$$fpr = P(predict + \mid actual -)$$
$$fnr = P(predict - \mid actual +)$$

- Contribute differently to overall error rate

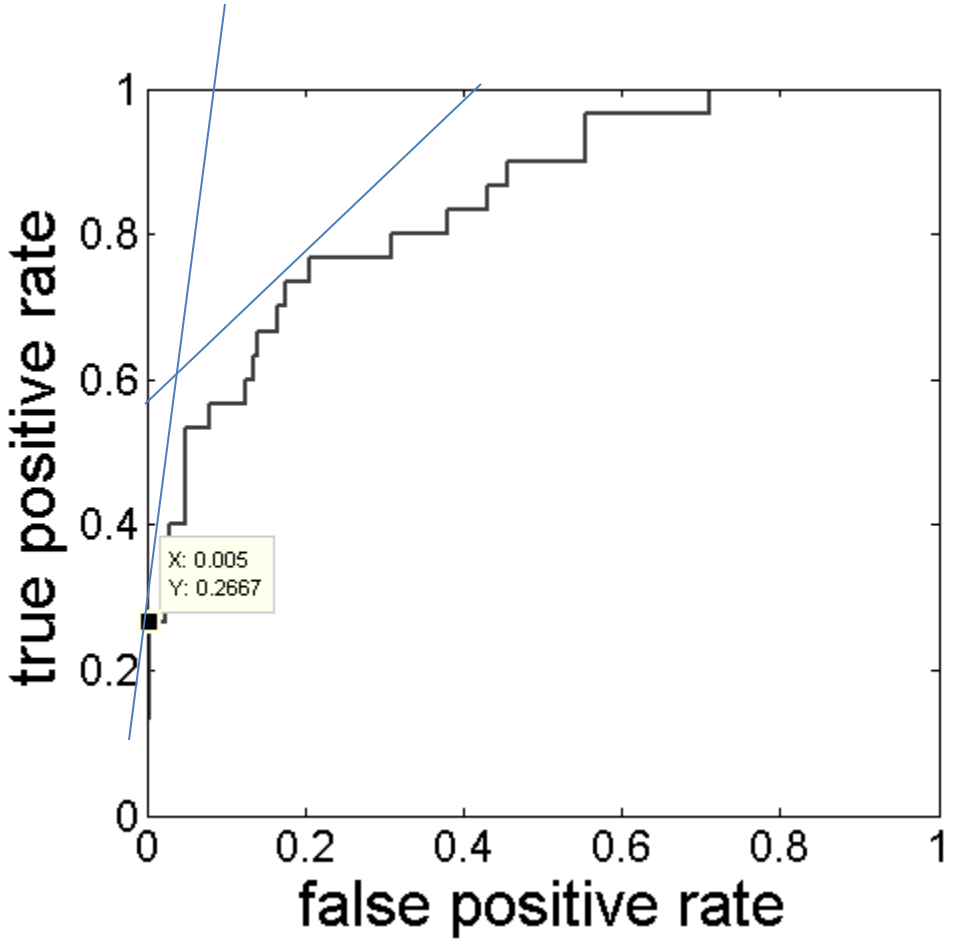
$$P(error) = P(+)fnr + P(-)fpr$$

- Write in terms of ROC axis variables

$$P(error) = P(+)(1 - tpr) + P(-)fpr$$

error is constant along lines of slope $P(-)/P(+)$

ROC operating point optimization



ARTIFICIAL NEURAL NETWORKS

Perceptrons

Output activated (out = +1) if net stimulus exceeds threshold

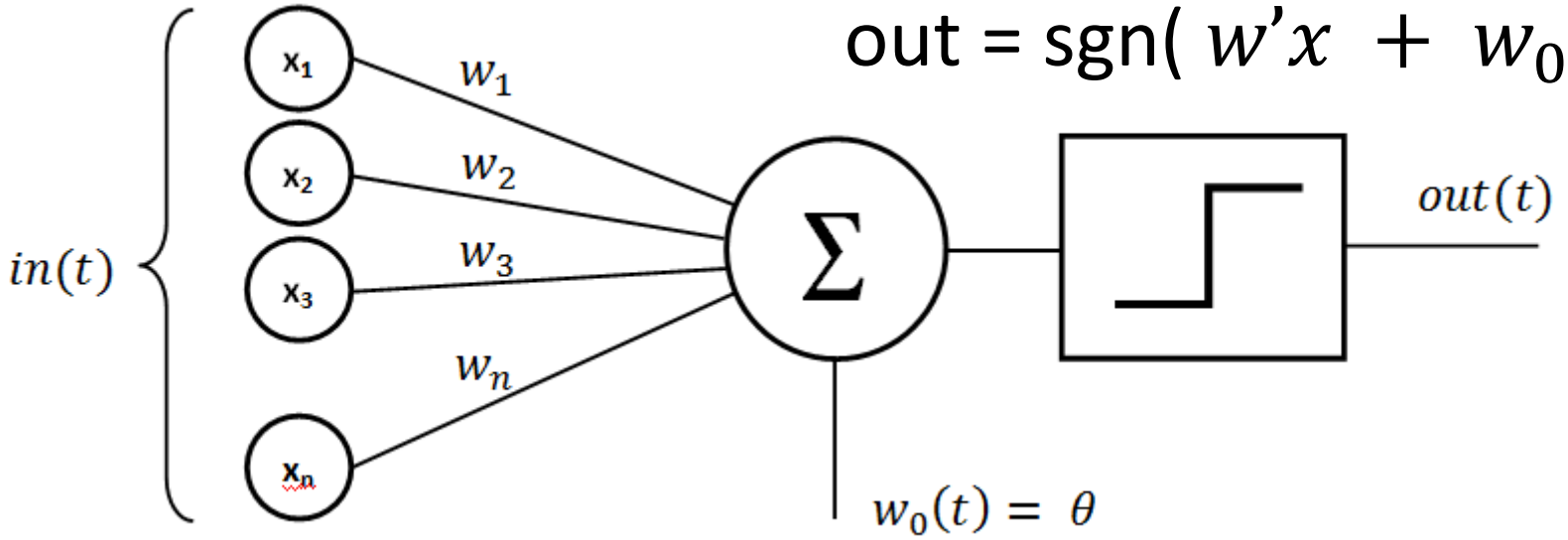
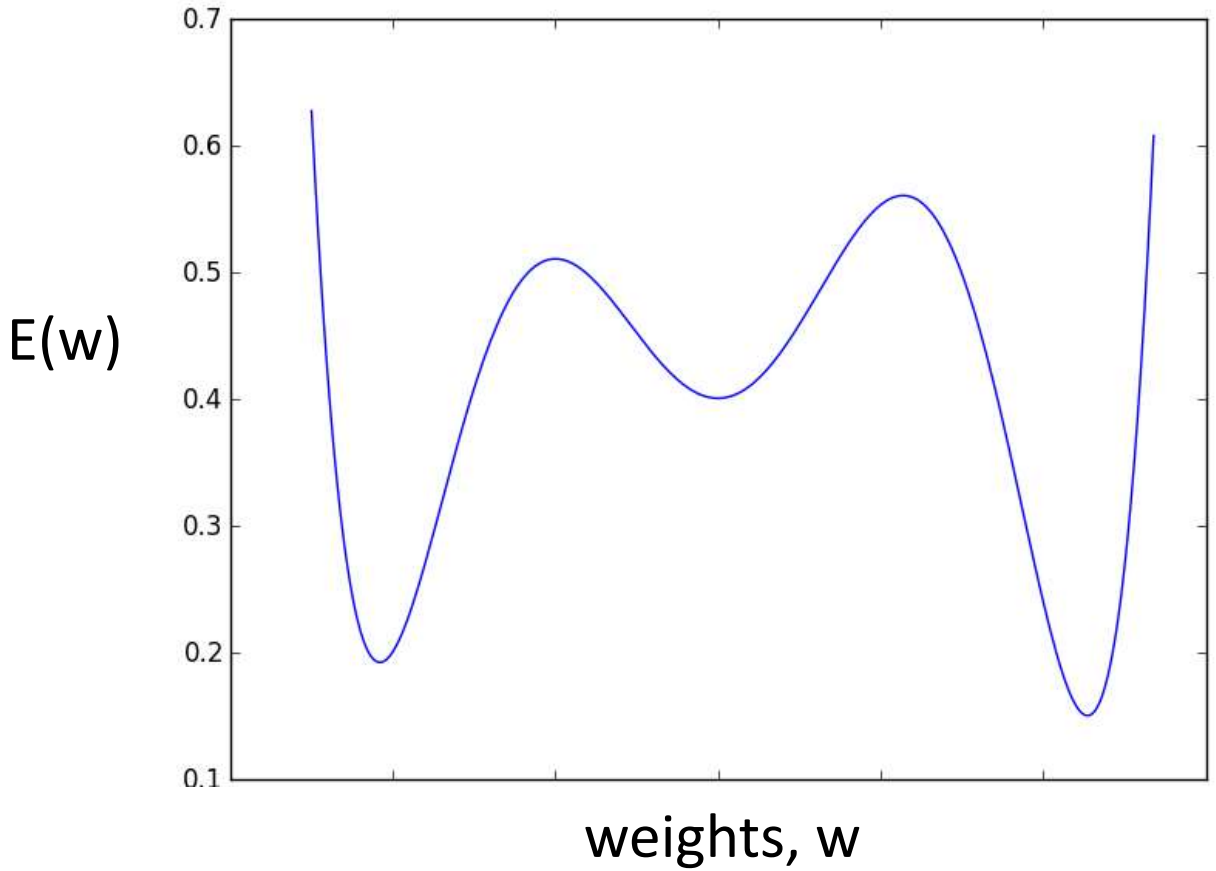


image: wikimedia.org

Learning is about the weights

- "Synaptic plasticity"
- Input-output behavior of the perceptron is completely determined by the weight vector w (with threshold)
- How to pick the weight values?

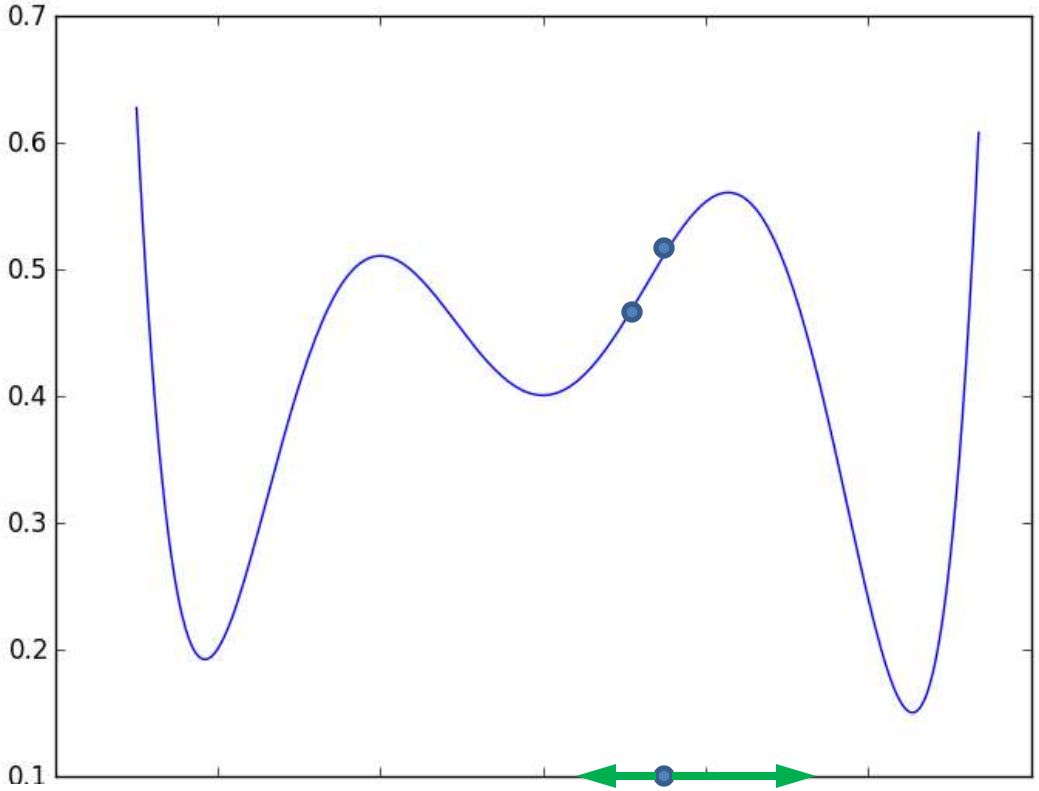
Error landscape



Gradient descent learning

Converges to local minimum if learning rate small

Easy to get stuck in local minimum if learning rate too large



E(w)

weights = w

$$\Delta w = -\eta \frac{\partial E}{\partial w}$$

Perceptron learning algorithm (gradient descent)

$$error = \frac{1}{2n} \sum_{\{k=1\}}^{n} (y_k - \hat{y}_k)^2, where\ y_k = \sum_{\{j=1\}}^{m} w_j\ x_{k,j}$$

$$\frac{\partial error}{\partial w_j} = \frac{1}{n} \sum_{\{k=1\}}^{n} (y_k - \hat{y}_k)\ \ x_{k,j}$$

"stochastic" weight updates:

$$\Delta w_j = \eta\ \hat{y}_k\ x_{k,j} \quad \text{if}\ \ y_k \neq \hat{y}_k$$

converges if zero-error solution exists (Rosenblatt, 1950s)

Limitations of single perceptrons

- Decision surface is linear in input space

$$y = 0 \text{ if and only if } \mathrm{w}'\mathrm{x} + \mathrm{w}_0 = 0$$

- But data may not be linearly separable at all (e.g., a positive island in a sea of negatives)
  - Perceptrons are representationally challenged

Multilayer neural networks

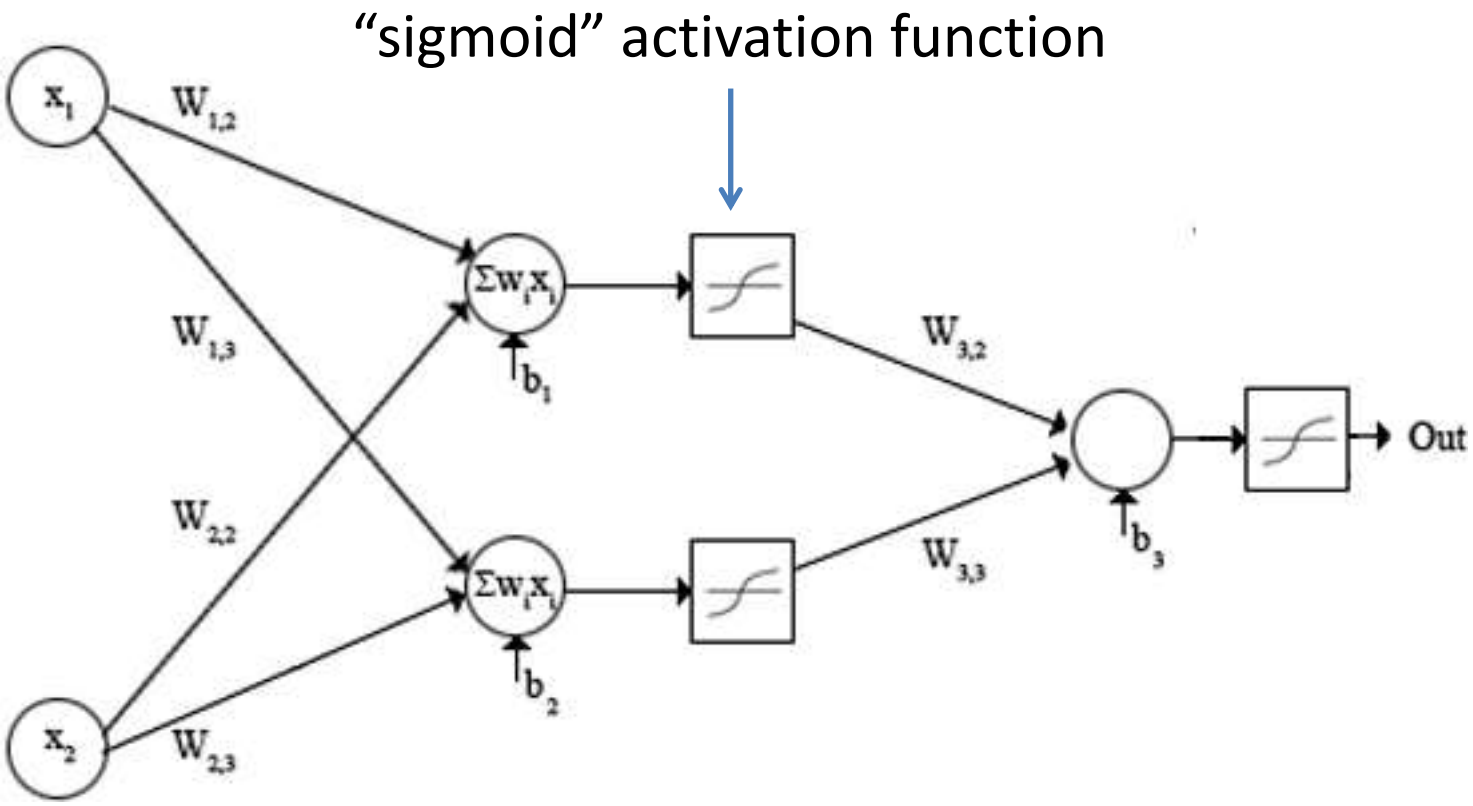


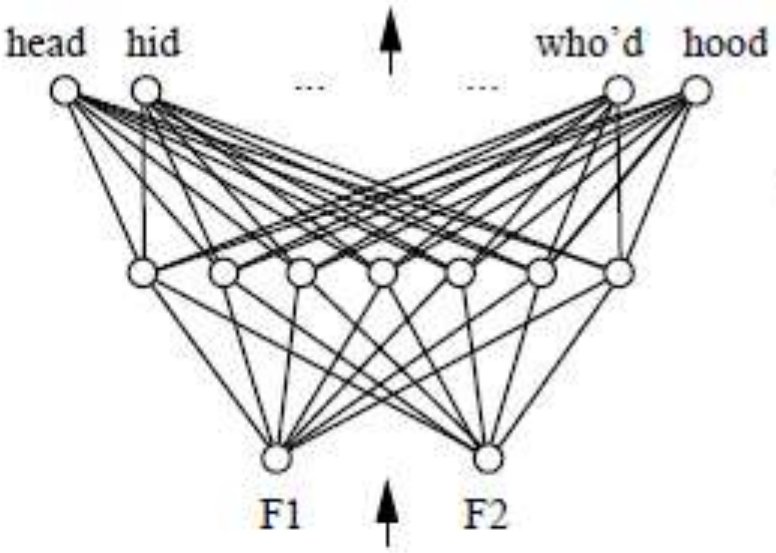
Image: matlabgeeks.com

Universal approximation property

- (Cybenko, 1989) Artificial neural networks with a single hidden layer of sigmoid units can uniformly approximate any continuous input-output function arbitrarily closely.

- In particular, classes that can be separated by a continuous boundary can be separated with arbitrarily small error by an ANN classifier.

Nonlinear ANN decision boundary



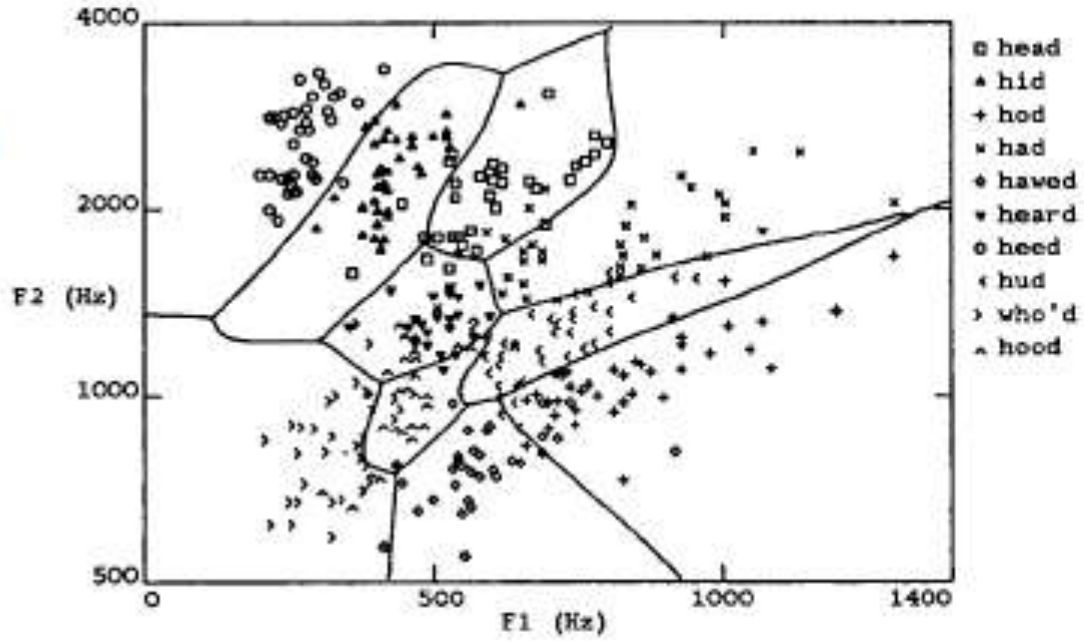


Diagram: T. Mitchell

Training ANN: the error back-propagation (EBP) algorithm

- Gradient descent in error landscape (as for perceptrons), rewritten in a recursive form
  - Repeat until convergence or stopping condition met
    - For each training instance $(x_1, x_2, \ldots x_n, \hat{y})$

      Present x_i to network inputs, propagate through network, yielding hidden activations h_j and outputs y_k

      Compute error at each output unit: $\delta_k = y_k(1 - y_k)(\hat{y}_k - y_k)$

      Propagate errors back through network, computing δ at each unit

      $\delta_h = y_h(1 - y_h) \sum_k \delta_k w_{k,h}$ ($w_{k,h}$ is weight from hidden h to output k)

      $\delta_i = y_i(1 - y_i) \sum_h \delta_h w_{h,i}$ ($w_{h,i}$ is weight from input i to hidden h)

      Update network weights:

      $w_{a,b} = w_{a,b} + \eta y_b \delta_a$ ($w_{a,b}$ is weight from unit b to unit a)

Avoiding overfitting in ANN

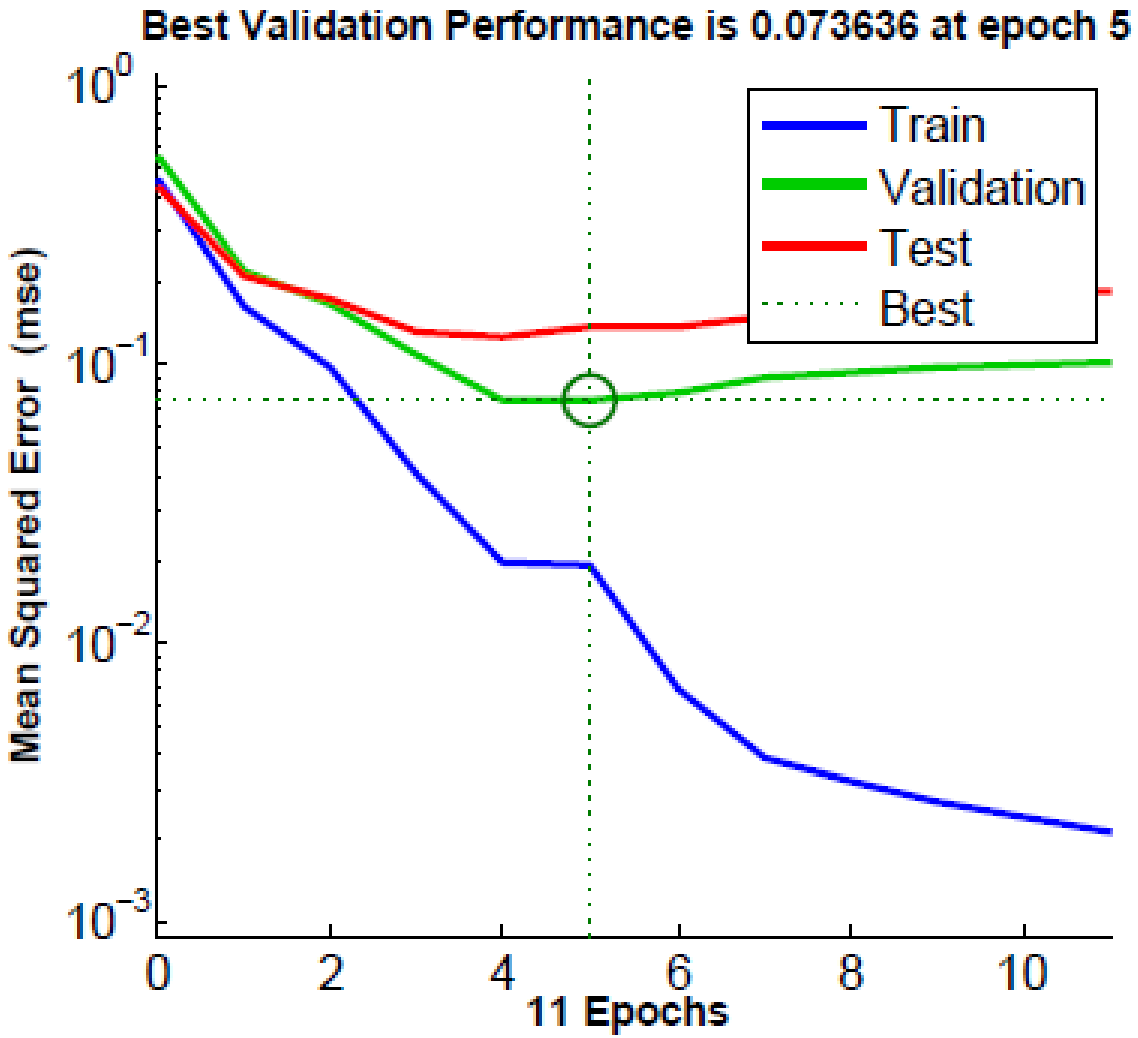
- Universal approximation property is appealing
  - but is a double-edged sword
  - easy to overfit training dataset

- Validation set approach
  - Given labeled training dataset, D

    Split D into disjoint portions $D_{train}, D_{validate}, D_{test}$

    Train network iteratively on D_{train}, test periodically on D_{train}

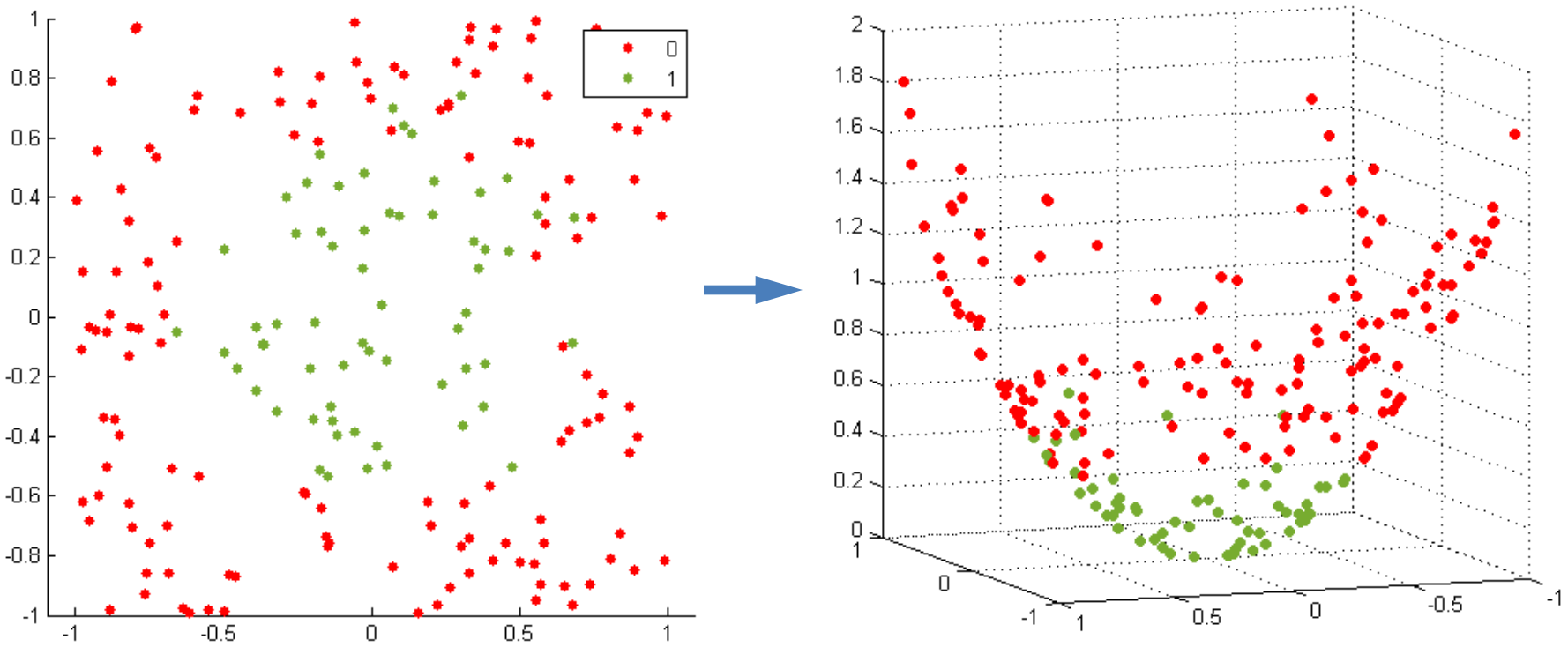
    Stop when validation error increases consistently


Best Validation Performance is 0.073636 at epoch 5
Mean Squared Error (mse)
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
2
4
6
8
10
11 Epochs
Train
Validation
Test
Best

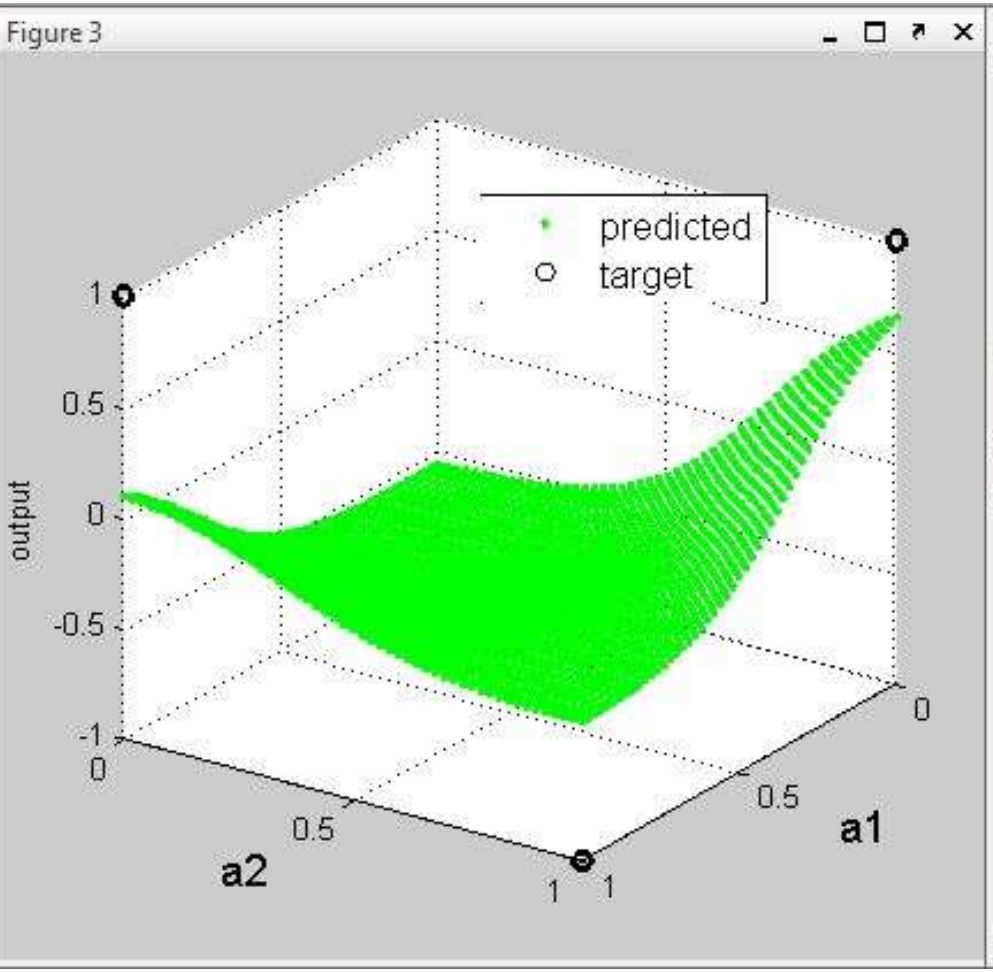
Alternatives to error back-propagation

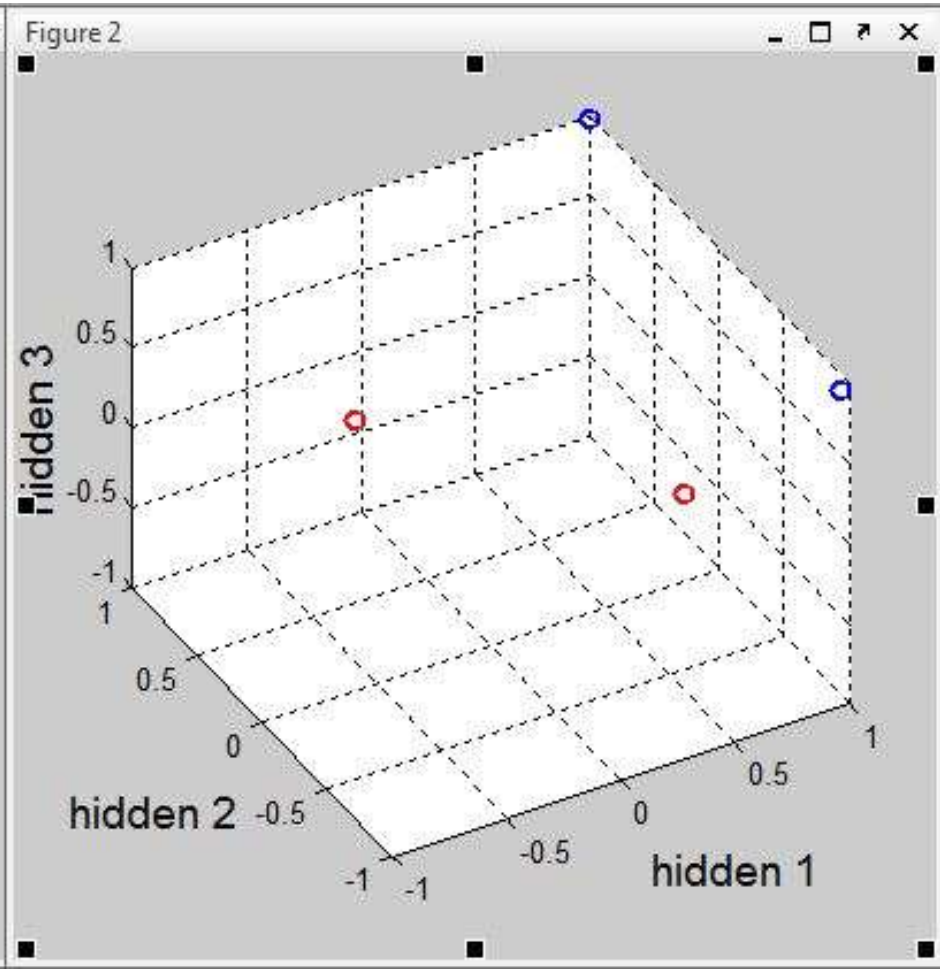
- EBP can be very slow to converge
  - Gradient vanishes at local minima of error
- Many variations
  - Adaptive learning rate
  - Conjugate gradient methods
  - …

ANN hidden layer representations

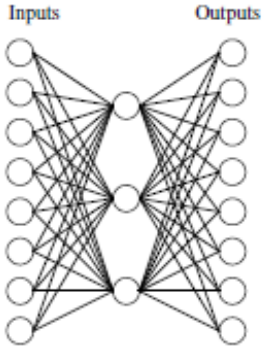


ANN hidden layer representations





ANN hidden layer representations



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Diagram: T. Mitchell

Deep learning

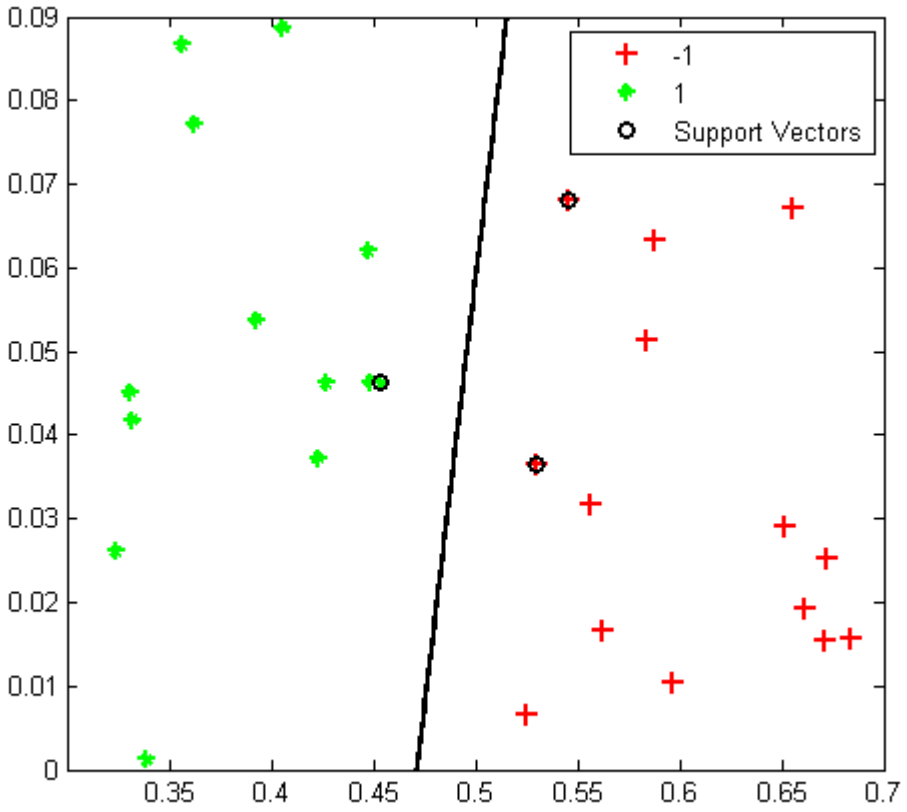
- Recent work revisits multi-layer networks as classifiers and as generative models
  - Training uses Markov chain Monte Carlo ideas
  - Hierarchical hidden representations develop
  - See work by G. Hinton and others

    digits generation
- and the talk by Prof. Baldi on Wed morning

SUPPORT VECTOR MACHINES (SVM)

Making perceptrons more robust

- Many diff ries
  - Which o



SVM: optimization formulation

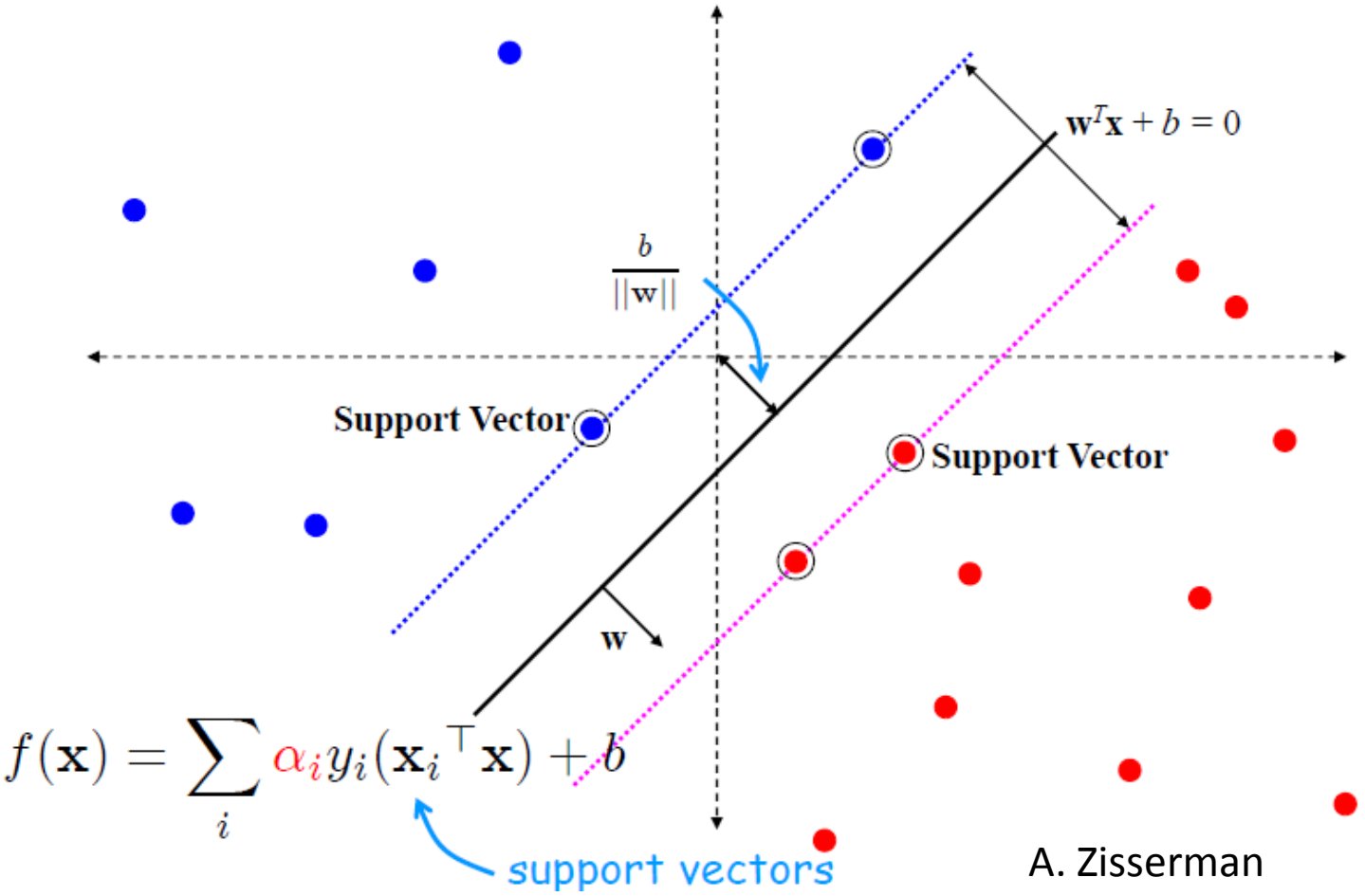
- Seek weight vector w that maximizes margin among w that classify all examples correctly

$$\max_{w} \frac{b}{||w||} \quad \text{subject to } y^i(w'x + b) \geq 1 \ \ \forall i$$

- Solution is a linear combination of the data points x_i at the margins ("support vectors")

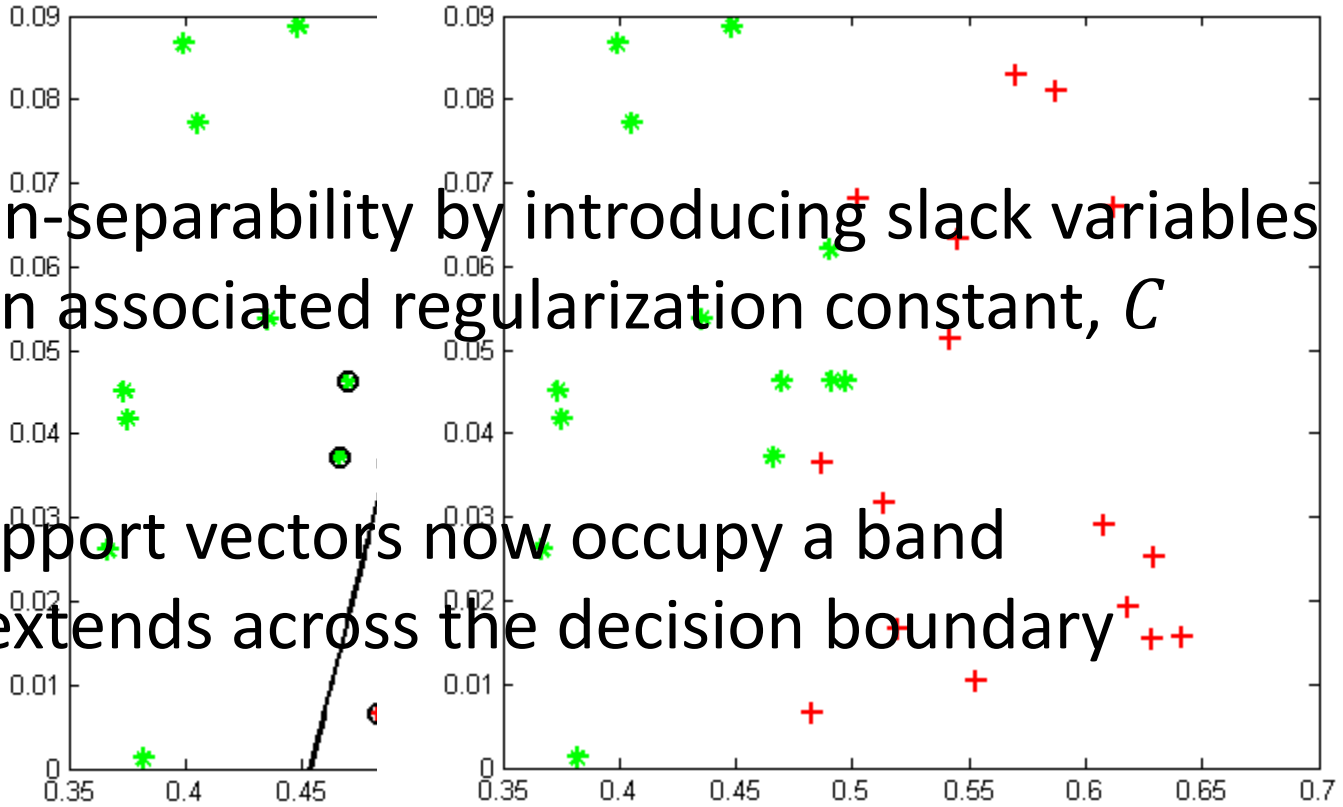
$$w^* = \sum_i \alpha_i \, y_i x_i$$

SVM: dual formulation



A. Zisserman

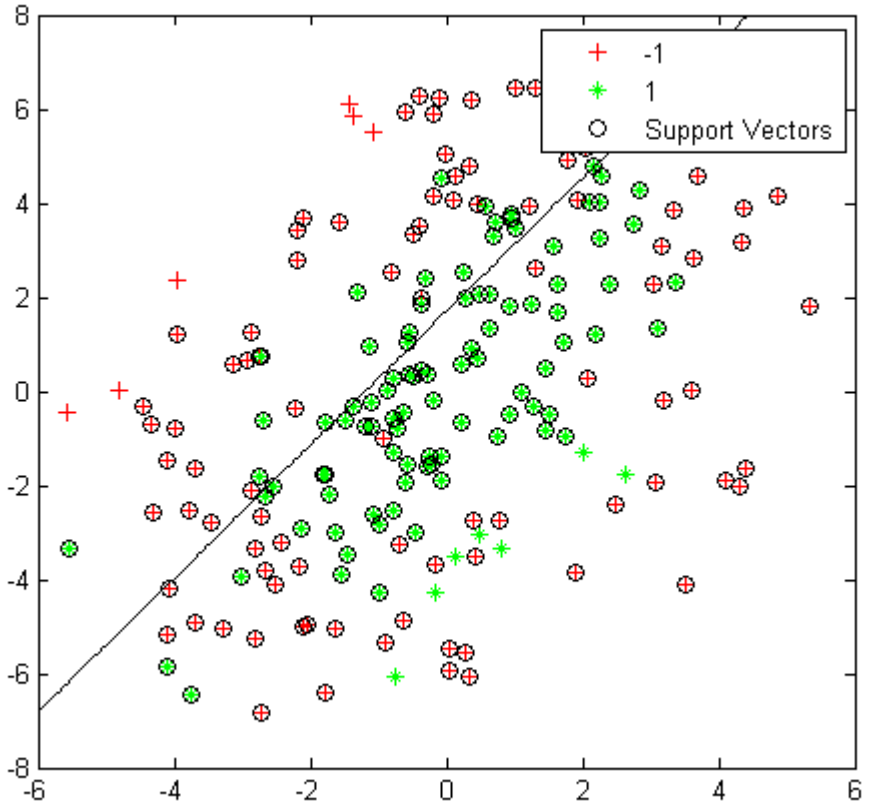
SVM: linearly non-separable case



Address non-separability by introducing slack variables and an associated regularization constant, C

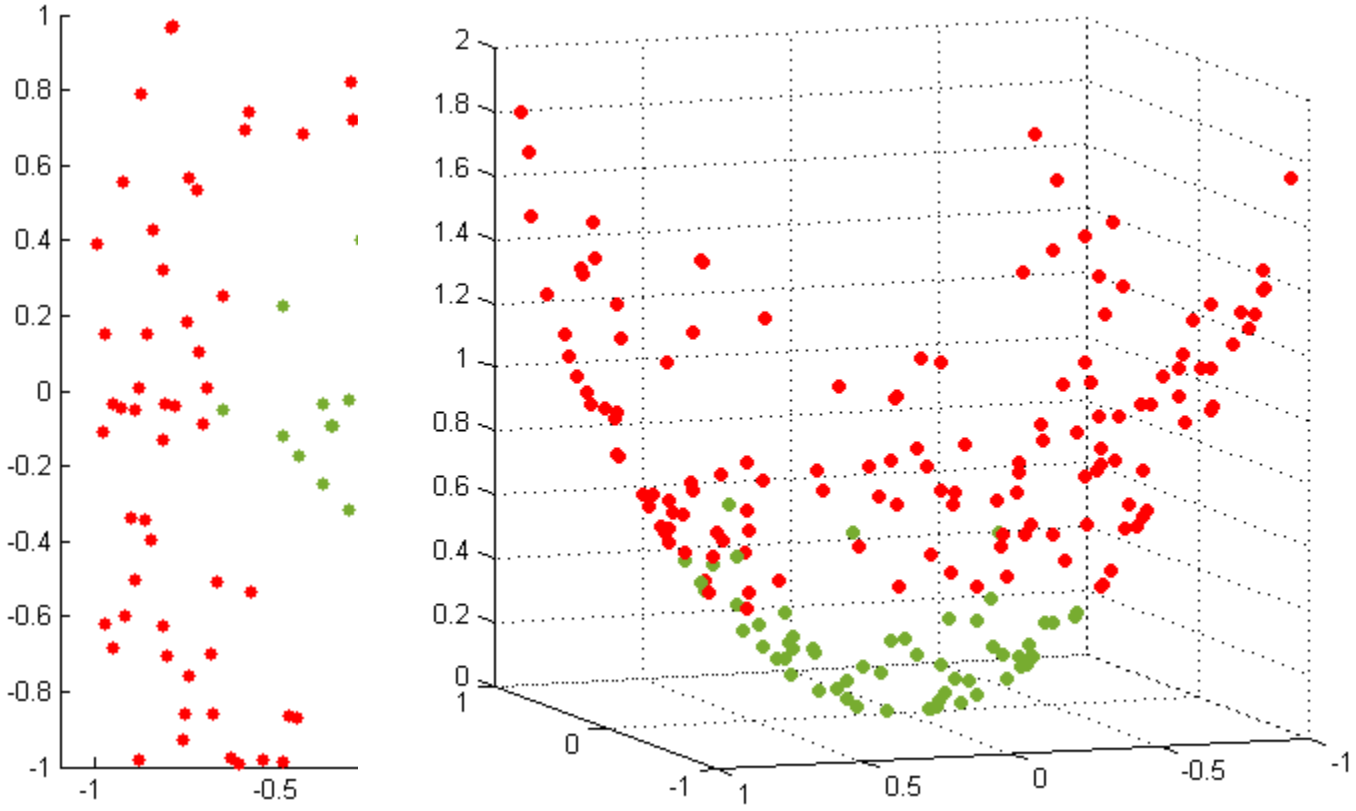
Support vectors now occupy a band that extends across the decision boundary

SVM: linearly non-separable case



Recall solution by feature extraction

Let $z = x^2 + y^2$



Nonlinear SVM

- Decision boundary in previous slide is linear in the features $f_1 = x^2$ and $f_2 = y^2$
- It can therefore be found using a linear SVM in the extended feature space (x, y, f_1, f_2)
- Consider the mapping Φ: $(x, y) \mapsto (x, y, x^2, y^2)'$
- SVM solution only depends on the dot products $\Phi(x_i)'\Phi(x_j)$ of the training examples (Gram matrix)
  - This can be shown using the dual optimization formulation

More general nonlinear boundaries

- Any conic section (parabola, hyperbola, ellipse) is a level curve of a quadratic function in 2D
  - a linear combination of the features $1, x, y, xy, x^2, y^2$
- An SVM in this 6D space will be able to find a quadratic decision boundary if one exists
- Higher-order boundaries addressed analogously
  - Order n requires a feature space of dimension $O(n^2)$

Kernels

- High-D feature space computations are inefficient
- Shortcut possible using notion of kernel

  Similarity function $K(x, y)$ on the input space S such that:

  $K(x, y)$ is symmetric: $K(x, y) = K(y, x)\ \forall x, y$

  $K(x, y)$ is continuous

  $K(x, y)$ is positive semi-definite:

  $$\sum_{i,j} c_i K(x_i, x_j) c_j \geq 0$$

  for all finite point sequences $x_1, x_2, \ldots, x_n$ in S

  and all real number sequences $c_1, c_2, \ldots, c_n$.

Mercer's theorem

(Mercer, 1908) Let $K(x, y)$ be any continuous, symmetric, positive semi-definite kernel on the input space S.

There exists a mapping $\phi: S \rightarrow F$, where F is a (usually higher-dimensional) inner product space, so that $K(x, y) = < \phi(x), \phi(y) >_F$ for all x, y in S.

The “kernel trick”

- Since the dual formulation of SVM only depends on the Gram matrix $\Phi(x_i)'\Phi(x_j)$, and since $\Phi(x_i)'\Phi(x_j) = K(x_i, x_j)$, the kernel function suffices in order to solve for all nonlinear decision boundaries that are linear in the high-D feature space F.

- You don’t even have to explicitly know the precise feature space or feature mapping

Popular kernel families

- Polynomial

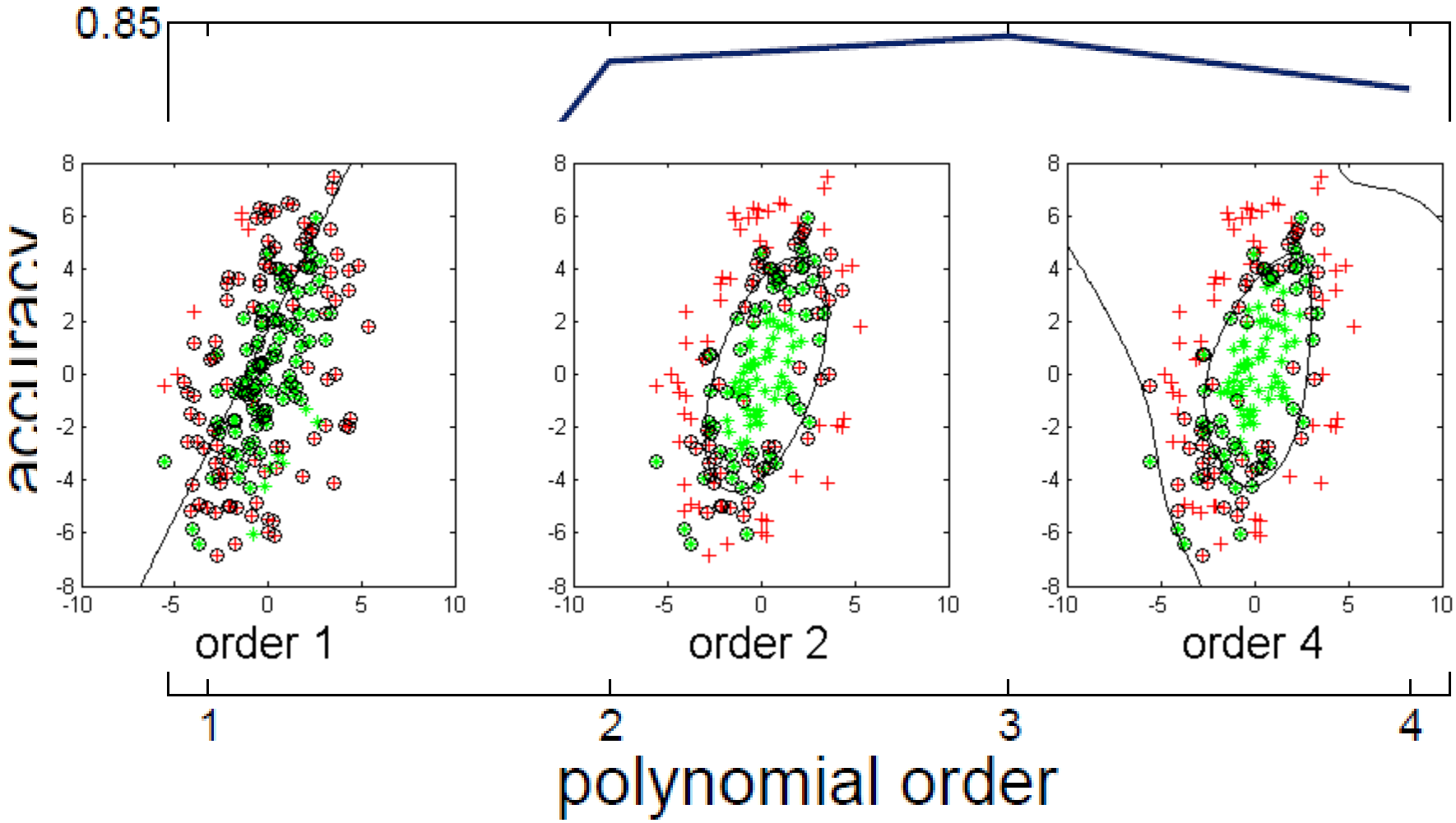
$$K(x, y) = (1 + x \cdot y)^d$$

- Gaussian (radial basis function)

$$K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

- String (Lodhi et al., 2002)

Bias-variance tradeoff in SVM



INSTANCE-BASED MACHINE LEARNING

The data-driven viewpoint

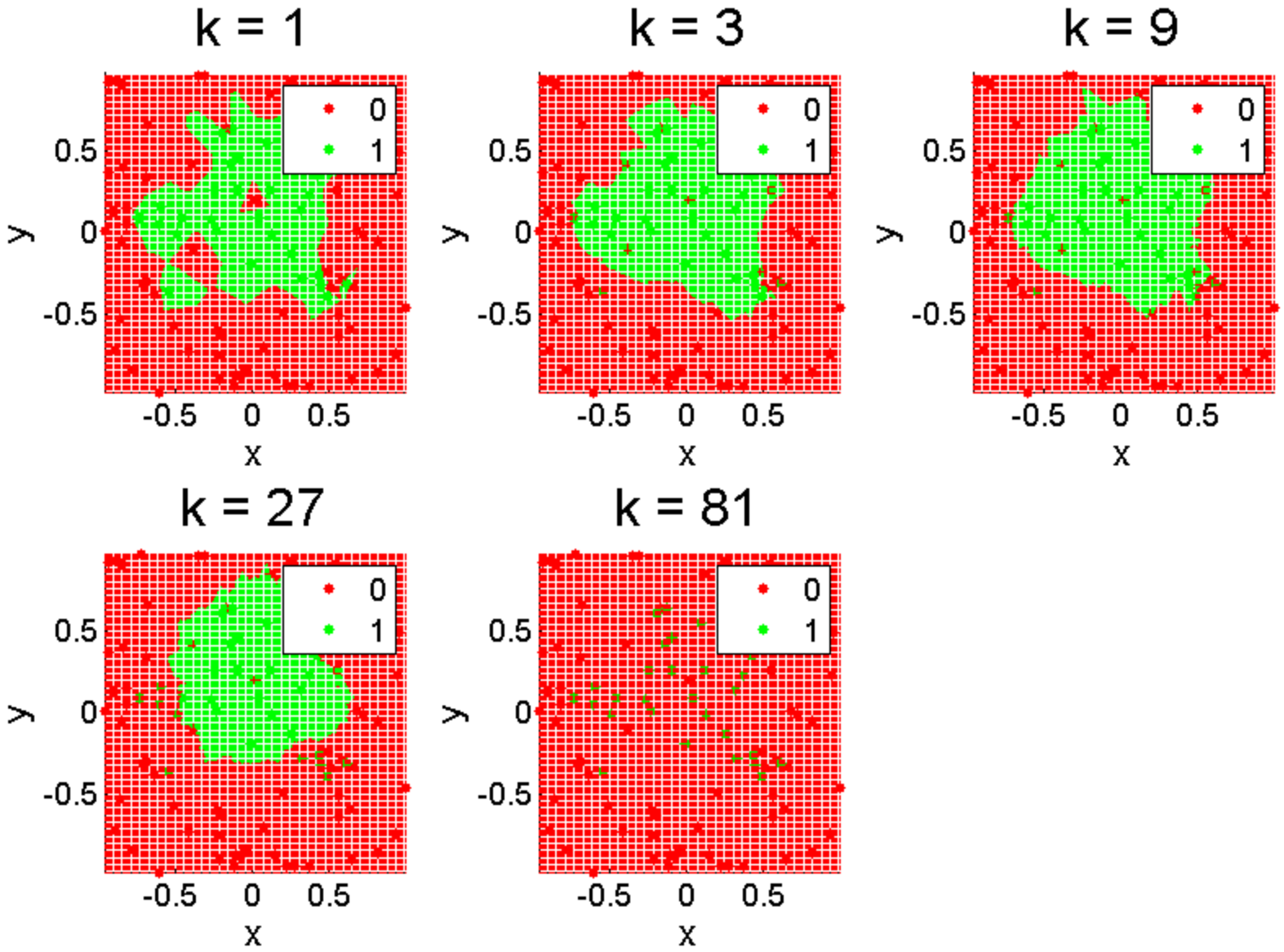
- Why have models at all? Let data tell the story
- Instance-based learning algorithm ("lazy learning")
  - Input: dataset $D = \{(x^1, \hat{y}^1), \ldots, (x^n, \hat{y}^n)\}$
  - Output: predictive model M based on D
  - Procedure:

    return D
- Instance-based prediction algorithm
  - Input: unlabeled data instance, x
  - Output: predicted label for x
  - Procedure:

    return label of most similar instances to x in D

Nearest-neighbor prediction

- Define similarity in terms of a distance metric $d(x, y)$ on the space of data instances
- For prediction on an instance, x
  - Find the k nearest neighbors of x in training set D
  - Classification: predict modal class among the k nearest neighbors
  - Regression: predict weighted mean of the target values of k nearest neighbors


k = 1
k = 3
k = 9
k = 27
k = 81
0
1
0.5
0
-0.5
y
-0.5
0
0.5
x

How many neighbors, k?

- Simple analysis yields order of magnitude of optimal k
- Optimal k where bias and variance effects "cross"

  Standard deviation of mean of k neighbors = $O\left(\frac{1}{\sqrt{k}}\right)$

  Bias = O(diameter of k random points)

  - k points occupy $\frac{k}{N}$ of total volume, so distance in unit cube = $\left(\frac{k}{N}\right)^{\frac{1}{d}}$

  Bias = stdev iff $\frac{1}{\sqrt{k}} = \left(\frac{k}{N}\right)^{\frac{1}{d}}$ iff $\boxed{k = N^{\frac{1}{1+\frac{d}{2}}}}$

- For d=2, above predicts

  $k = \sqrt{N}$, so $k = 10$ in case $N = 100$

Instance-based learning pros and cons

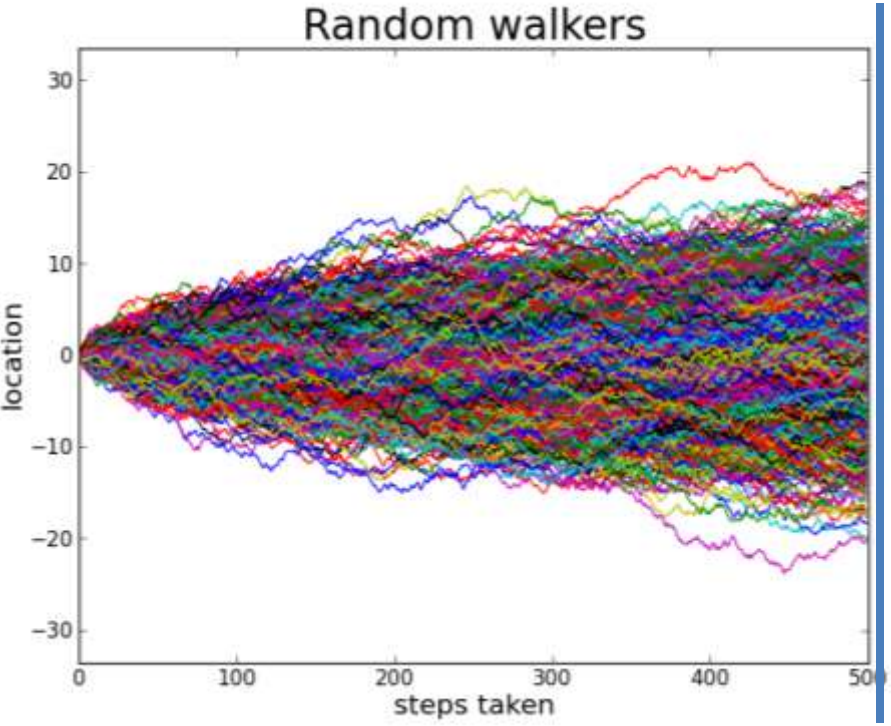
- Pros
  - Training is not needed
  - Does not rely on any parametric assumptions
- Cons
  - Requires a lot of memory
  - Prediction is computationally intensive due to similarity search

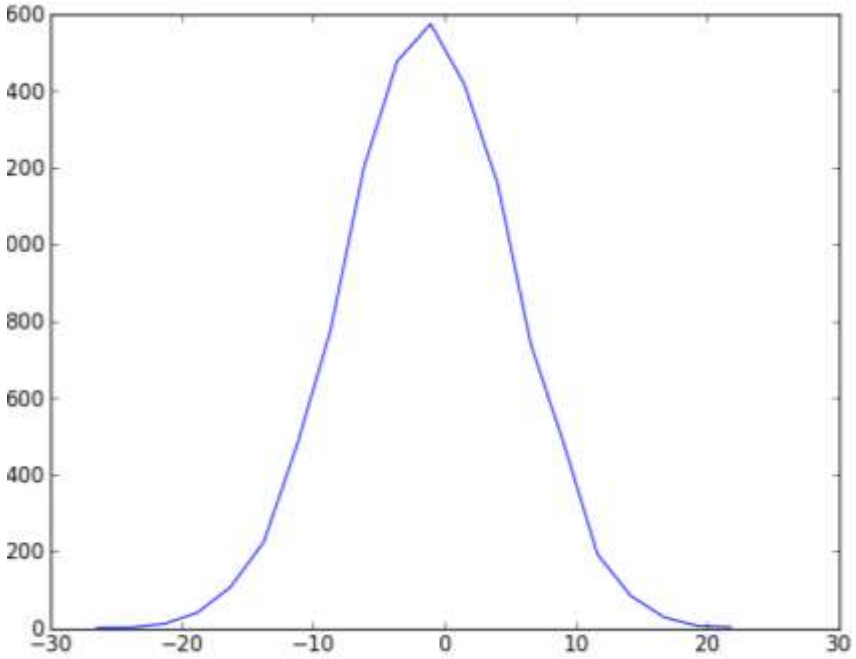
PROBABILISTIC MACHINE LEARNING

Randomness unpredictable (mostly)

Random event means occurrence unpredictable

However, randomness allows some structure "in the large" except for relative frequency





Probabilistic classification

- Classes $c_1, c_2, \ldots, c_k$ with "prior" probabilities $P(c_i)$
- Attributes $x_1, x_2, \ldots, x_m$ with joint class-conditional distribution $P(x_1, x_2, \ldots, x_m | \, c_i)$

- Given a data observation $x_1, x_2, \ldots, x_m$,

  what class should be predicted?

Maximum likelihood classification

- Predict class $\text{argmax}_i P(x_1, x_2, \ldots, x_m \mid c_i)$
- Example
  - P(positive lab result | sick) = 0.9
  - P(positive lab result | healthy) = 0.2
  - Therefore, given a positive lab result as evidence, predict that the patient is sick

Bayesian classification

- Bayes' rule accounts for prior and class-conditional probs
  - For hypotheses h and evidence e

$$P(h \mid e) = P(h)P(e \mid h)/P(e)$$

- Bayesian classifier
  - Predict class $\text{argmax}_{\text{i}}\ P(c_i \mid x_1, x_2, \ldots, x_m)$

- Example
  - P(sick = 0.01), P(+ lab | sick) = 0.9, P(+ lab | healthy) = 0.2
  - By Bayes' rule,
  - P(sick | + lab) = 0.01 0.9 / D, P(healthy | + lab) = 0.99 0.2 / D
  - Therefore, given a positive lab result as evidence,
    predict that the patient is healthy (about 20 times as likely)

Naïve Bayes approach

- Bayesian approach often impractical
  - Can't estimate details of class-conditional distributions
    - 10 ternary attributes allow 59000 combinations
    - How to estimate these probabilities using 100 instances?
- Assume class-conditional independence (CCI)

$$P(x_1, x_2, \ldots, x_m \mid c_i) = \prod_j P(x_j | c_i)$$

Predict class $\text{argmax}_\text{i}\, P(c_i) \prod_j P(x_j | c_i)$

  - Estimation of individual attribute values $\prod_j P(x_j | c_i)$ is feasible
    - 10 ternary attributes only require 10(3) = 30 probabilities
- Naïve Bayes can work well even if CCI assumption fails

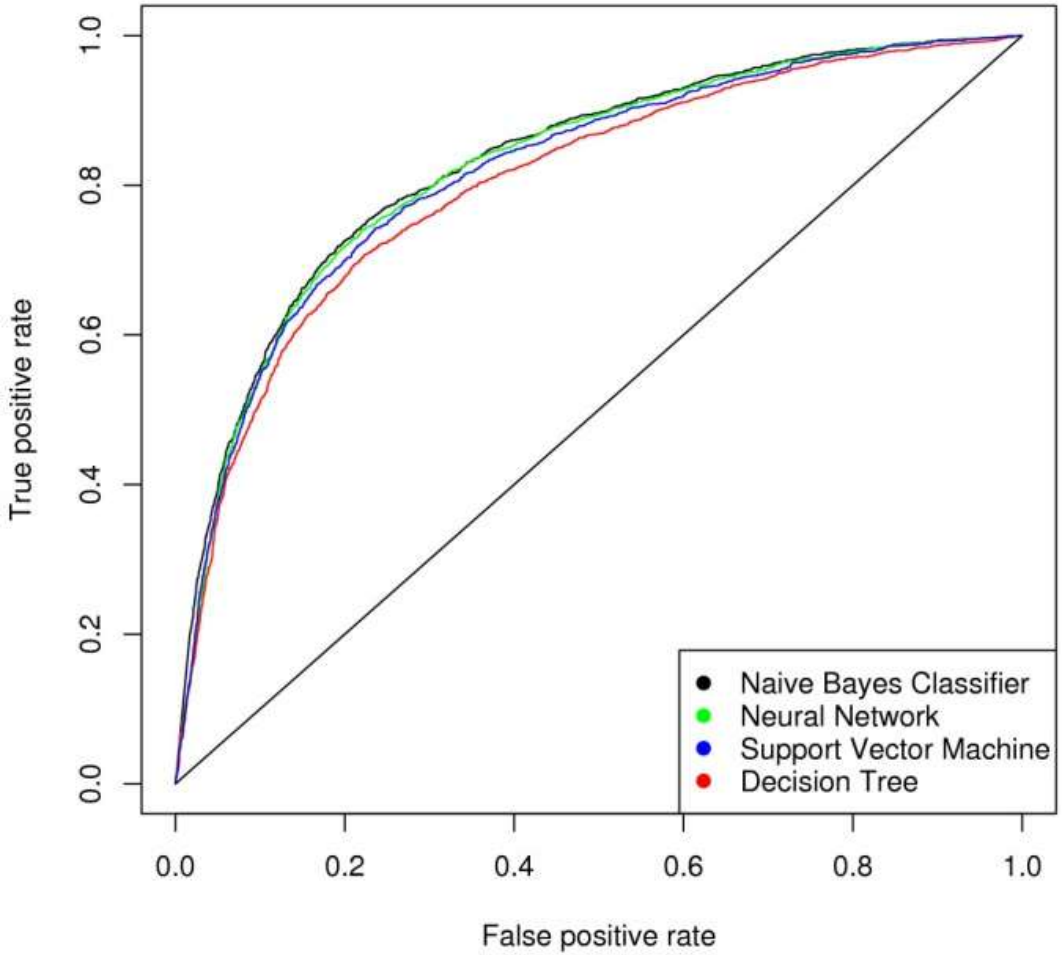
Naïve Bayes for text categorization

- Bag of words representation of document, d
  - Define dictionary of allowed words, W
  - Bernoulli version
    - View d as vector $d: W \rightarrow \{0,1\}$ of word occurrences
  - Multinomial version
    - View d as vector $d: W \rightarrow Z^{+} \cup \{0\}$ of word counts

- Use naïve Bayes word independence assumption

$$P(d \mid class) = \prod_{w \in W} P(d(w) \mid class)$$

Naïve Bayes for text categorization

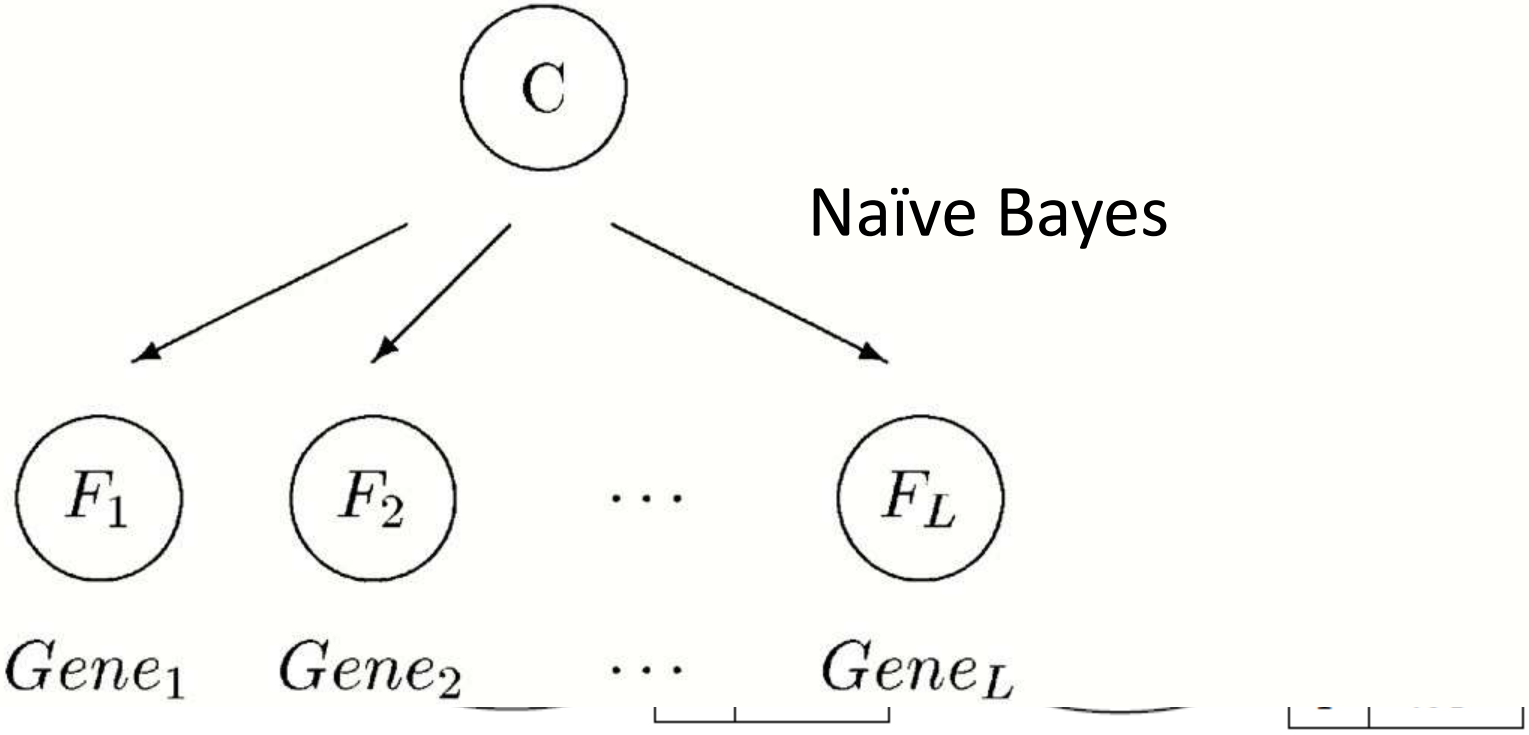


Wang et al, *BMC Bioinformatics* 8:269, 2007

Bayesian networks

Each node conditionally independent of its nondescendants, given its parents

Graphical models of conditional dependence relationships



Naïve Bayes

Koller and Sahami, 1996

Russell and Norvig, *Artificial Intelligence* (AI), 2000

Expectation-maximization
(Dempster, Laird, Rubin 1977, but basic idea is earlier)

- Iterative general approach to estimation of "hidden parameters" in probability models
  - Including Bayesian networks

- Repeat to convergence or stopping condition
  - E step
    - Calculate expected value of generative log likelihood of data given current model parameters
  - M step
    - Adjust parameter values to maximize expected log likelihood

Inference in Bayesian networks

- Exact inference is computationally complex

- Approximate inference techniques
  - Markov chain Monte Carlo samples from equilibrium distribution of Markov chain

Unsupervised clustering using E-M

- Model unlabeled data using mixture of known parametric distributions, say Gaussians
  - Hypothesize k populations, each described by different parameter values (e.g., prior probability of that population, and its mean and covariance)
- Use E-M to estimate parameters for given k
- Compare different k using DL ideas (combine generative log likelihood with model complexity)
  - Bayes Information Criterion (BIC)
  - Akaike Information Criterion (AIC)

Example

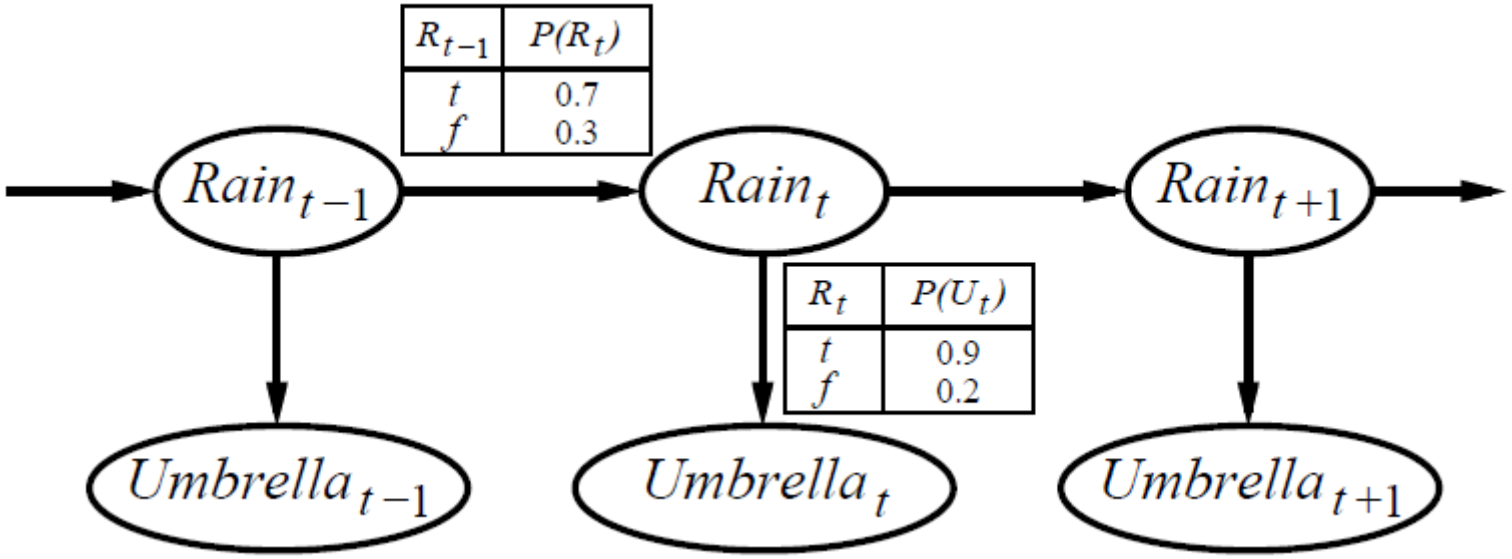
- Mixture of Gaussians E-M clustering

Probabilistic sequence models

- Model random effects over time or space
  - Speech
  - Sequence generation
- Use Bayesian networks, plus time slices
  - The future is conditionally independent of the past, given the present
- Typically include hidden variables

Hidden Markov models (HMM)

States X_t at top not observable, only evidence e_t at bottom



Russell and Norvig, *Artificial Intelligence*

HMM state estimation (forward algorithm)

- Given evidence $e_1, e_2, \ldots, e_t, e_t$, how to estimate of actual final state X_t?
- Seek conditional distribution $P(X_t \mid e_{1\ldots t})$

  Assume distribution $P(X_{t-1} \mid e_{1\ldots t-1})$ known (at $t-1$)
- By Bayes' rule, $P(X_t \mid e_{1\ldots t}) = c\, P(e_{1\ldots t} \mid X_t)$

  $= c\, P(e_t \mid X_t)\, P(X_t \mid X_{t-1})\, \sum P(X_{t-1} \mid e_{1\ldots t-1})$

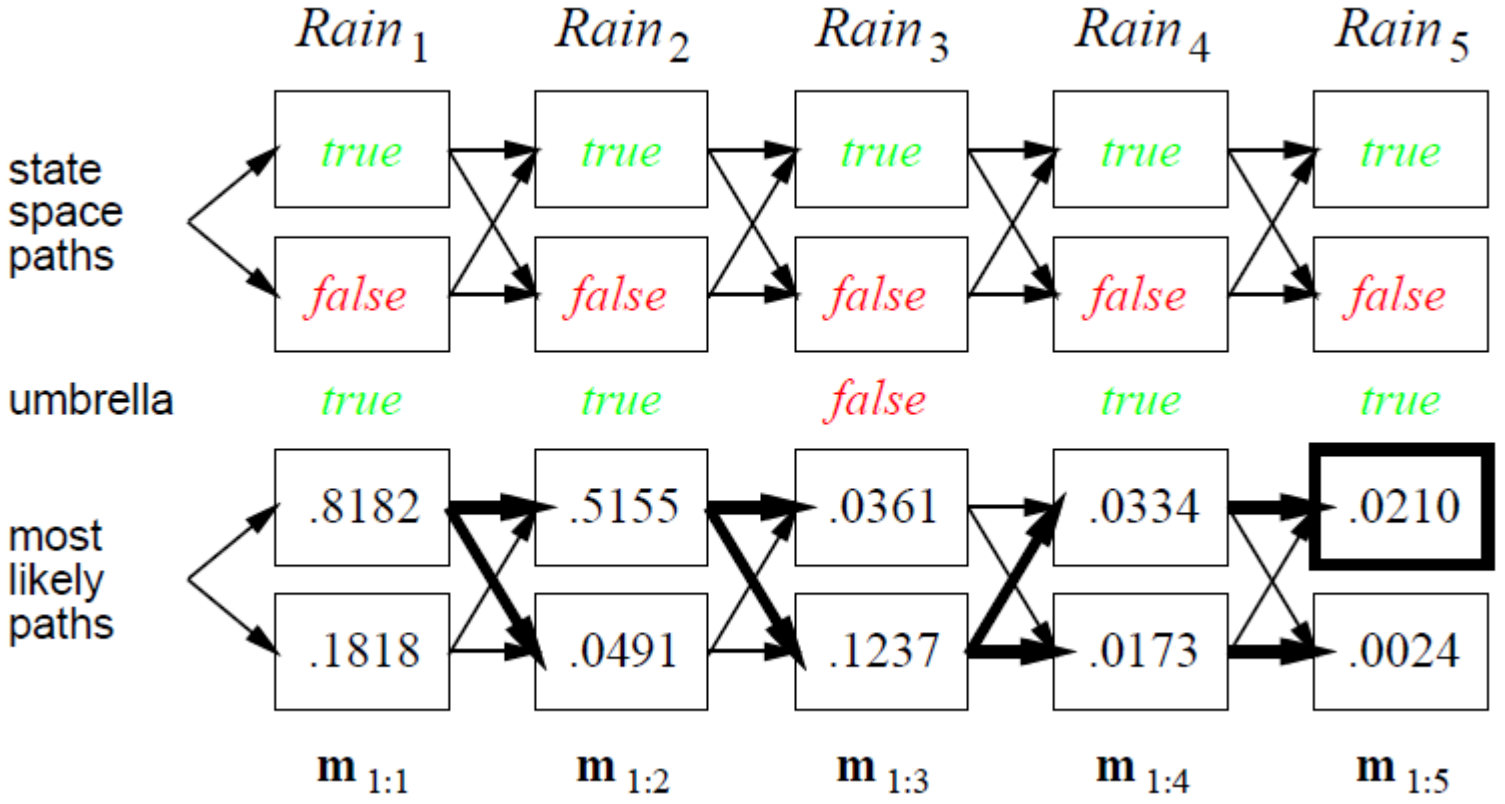
  which is a sum over the conditional distribution at $t-1$ (solved by recursion)

Example

- Berkeley PacMan

HMM most likely state sequence (Viterbi algorithm)

- Dynamic programming approach



Russell and Norvig, *Artificial Intelligence*

Learning HMM parameters

- Use Expectation-Maximization
  - Known as Baum-Welch algorithm in HMM context

TO LEARN MORE

Free machine learning software (APIs and interactive)

- Python
  - scikit-learn
  - mlpy
  - pyML
- Java
  - Weka
  - Apache Mahout
- C / C++
  - SHOGUN (includes some of SVM-Light)
  - mlpack
- R
  - kernlab

Other machine learning resources

- ML on Coursera
  - https://www.coursera.org/course/ml
- ML course materials by Thorsten Joachims
  - http://svmlight.joachims.org/

Introductory ML references

- Tom Mitchell. *Machine Learning*, McGraw-Hill, 1997
- Christopher M. Bishop. *Pattern Recognition and Machine Learning*, Springer, 2006
- Richard Duda, Peter Hart and David Stork. *Pattern Classification*, 2nd ed., John Wiley & Sons, 2001
- Ian H. Witten, Eibe Frank, Mark A. Hall. Data Mining: *Practical Machine Learning Tools and Techniques*, 3rd ed., Morgan Kaufmann, 2011
- Stuart Russell and Peter Norvig. *Artificial Intelligence: a Modern Approach*, 3rd ed., Prentice Hall, 2009

THANK YOU