

Introductory Applied Machine Learning

Nigel Goddard
School of Informatics

Semester 1

The **primary aim** of the course is to provide the student with a set of practical tools that can be applied to solve real-world problems in machine learning.

Machine learning is the study of computer algorithms that improve automatically through experience [Mitchell, 1997].

In many of today's problems it is
very hard to write a correct program
but very easy to collect examples

Idea behind machine learning:
from the examples, generate the program

Spam Classification



Web page



learning	13
lectures	3
Paris Hilton	0
assignments	7
...	

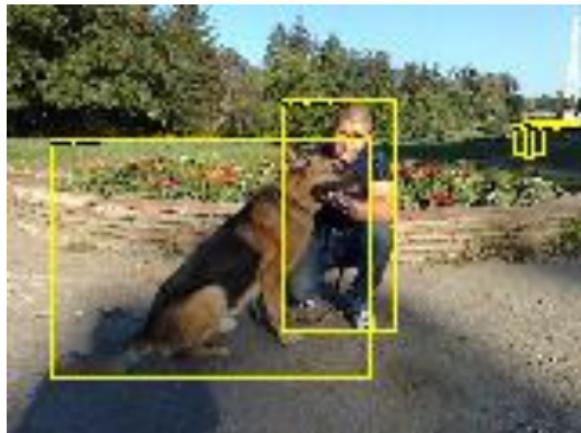
Feature vector



Classifier

SPAM
NONSPAM

Image Processing



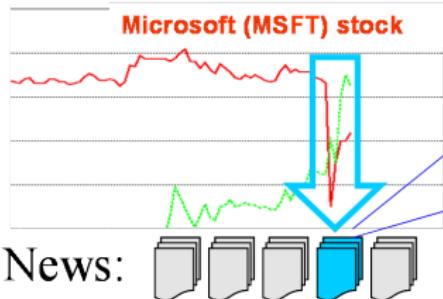
- ▶ Classification: Is there a dog in this image?
- ▶ Localization: If there is a dog in this image, draw its bounding box
- ▶ See: <http://host.robots.ox.ac.uk/pascal/VOC/>

Primate splice-junction gene sequences (DNA)

CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTCGAGCCAGTCTG EI
GAGGTGAAGGACGTCTTCCCCAGGAGCCGGTGAGAAGCGCAGTCGGGGCACGGGATG EI
TAAATTCTTCTGTTAACACCTTCAGACTTATGTGTATGAAGGAGTAGAAGCCAAA IE
AAACTAAAGAATTATTCTTTACATTCAGTTTCTGATCATGAAAACGCCAACAAAA IE
AAAGCAGATCAGCTGTATAAACAGAAAATTATTCGTGGTTCTGTCACTTGTGTATGGT N
TTGCCCTCAGCATCACCATGAACGGAGAGGCCATGCCTGCGCTGAGGGCTGCCAGGCCA N

- ▶ Task is to predict if there is an IE (intron/exon), EI or N (neither) junction in the centre of the string
- ▶ Data from ML repository: <http://archive.ics.uci.edu/ml/>

Financial Modeling



Software giant Microsoft saw its shares dip a few percentage points this morning after U.S. District Judge Thomas Penfield Jackson issued his "findings of fact" in the government's ongoing antitrust case against the Seattle wealth-creation machine...

Words like **Jackson** and **antitrust** are more likely in the stories preceding the plunge.

$$\begin{aligned}P(\text{shares}) &= 0.074 \\P(\text{antitrust}) &= 0.009 \\P(\text{judge}) &= 0.006 \\P(\text{trading}) &= 0.032 \\P(\text{against}) &= 0.025 \\P(\text{Jackson}) &= 0.001\end{aligned}$$

$$\begin{aligned}P(\text{shares} \mid \text{MSFT}\downarrow) &= 0.071 \\P(\text{antitrust} \mid \text{MSFT}\downarrow) &= 0.044 \\P(\text{judge} \mid \text{MSFT}\downarrow) &= 0.039 \\P(\text{trading} \mid \text{MSFT}\downarrow) &= 0.029 \\P(\text{against} \mid \text{MSFT}\downarrow) &= 0.027 \\P(\text{Jackson} \mid \text{MSFT}\downarrow) &= 0.025\end{aligned}$$

$$P(\text{MSFT}\downarrow \mid \text{Jackson}) = P(\text{Jackson} \mid \text{MSFT}\downarrow) P(\text{MSFT}\downarrow) / P(\text{Jackson})$$

[Victor Lavrenko]

Collaborative Filtering

Netflix – Unlimited TV Shows & Movies. How Does It Work?

CiteULike (post) pinboard DME IAML CSlib ECDFStatus

http://www.netflix.com/HowItWorks

Member Sign In

NETFLIX

Start Your 1 Month Free Trial How It Works Browse Selection 1 Month Free Trial Info

Unlimited TV episodes & movies instantly over the Internet plus unlimited DVDs by mail!

On your TV On your computer DVDs by mail

Connect devices like these to your Netflix account to **watch instantly on your TV.**

Wii PlayStation 3 XBOX 360

Learn more > Learn more > Learn more >

Watch as often as you want, anytime you want.

AND

Watch instantly on your computer too!

Laptop

PLAY

Learn more >

PLUS

DVDs by mail

Exchange as often as you want. No late fees - ever!

See other devices that stream instantly from Netflix

FAQs

How does Netflix work?
What is the selection like?
How much does it cost?
How many DVDs can I rent during my Free Trial?
How fast will I get my DVDs?
How long is the free trial?
Can I cancel any time?

How does Netflix work?

Rent what you want
Simply point and click to add movies & TV episodes to your list. Get DVDs by mail plus instantly watch movies (some new releases) & TV episodes (including current season) online on your PC or Mac or streamed instantly from Netflix over the Internet right to your TV via a Netflix ready device.

Start Your 1 Month Free Trial

Free trial offer details.

Email

Confirm Email

Password 4-10 characters

Confirm Password

www.netflix.com/Xbox?nktrk=hiw_xbox

More applications

- ▶ Science (Astronomy, neuroscience, medical imaging, bio-informatics)
- ▶ Environment (energy, climate, weather, resources)
- ▶ Retail (Intelligent stock control, demographic store placement)
- ▶ Manufacturing (Intelligent control, automated monitoring, detection methods)
- ▶ Security (Intelligent smoke alarms, fraud detection)
- ▶ Marketing (targetting promotions, ...)
- ▶ Management (Scheduling, timetabling)
- ▶ Finance (credit scoring, risk analysis...)
- ▶ Web data (information retrieval, information extraction, ...)

Overview

- ▶ What is ML? Who uses it?
- ▶ Course structure / Assessment
- ▶ Relationships between ML courses
- ▶ Overview of Machine Learning
- ▶ Overview of the Course
- ▶ Maths Level
- ▶ Reading: W & F chapter 1

Acknowledgements: Thanks to Amos Storkey, David Barber, Chris Williams, Charles Sutton and Victor Lavrenko for permission to use course material from previous years. Additionally, inspiration has been obtained from Geoff Hinton's slides for CSC 2515 in Toronto

Administration

- ▶ All material in course accessible to 3rd- & 4th-year undergraduates. Postgraduates also welcome.
- ▶ Course materials are on Learn (learn.ed.ac.uk)
- ▶ Lectures: online, with quizzes and reviews.
- ▶ Assessment:
 - ▶ Assignments (4) (25% of mark)
 - ▶ Exam (75% of mark)
- ▶ 4(5?) Tutorials and 4+ Labs
- ▶ Course rep
- ▶ Plagiarism

[http://web.inf.ed.ac.uk/infweb/admin/policies/
guidelines-plagiarism](http://web.inf.ed.ac.uk/infweb/admin/policies/guidelines-plagiarism)

- ▶ Schedule and News

Machine Learning Courses

IAML Basic introductory course on supervised and unsupervised learning.

MLPR More advanced course on machine learning, including coverage of Bayesian methods.

RL Reinforcement Learning.

MLP Real-world ML. This year: Deep Learning.

PMR Probabilistic modelling and reasoning. Focus on learning and inference for probabilistic models, e.g. probabilistic expert systems, latent variable models, Hidden Markov models

- ▶ Basically, IAML: Users of ML; MLPR: Developers of new ML techniques.

Overview of Machine Learning

- ▶ Supervised learning
 - ▶ Predict an output y when given an input \mathbf{x}
 - ▶ For categorical y : *classification*.
 - ▶ For real-valued y : *regression*.
- ▶ Unsupervised learning
 - ▶ Create an internal representation of the input, e.g. clustering, dimensionality
 - ▶ This is important in machine learning as getting labels is often difficult and expensive
- ▶ Other areas of ML
 - ▶ Learning to predict structured objects (e.g., graphs, trees)
 - ▶ Reinforcement learning (learning from “rewards”)
 - ▶ Semi-supervised learning (combines supervised + unsupervised)
 - ▶ We will not cover these at all in the course

Supervised Learning (Classification)



Training data



$$\mathbf{x}_1 = (1, 0, 0, 3, \dots)$$

y₁ = SPAM

Feature
processing



$$\mathbf{x}_2 = (-1, 4, 0, 3, \dots)$$

y₂ = NOTSPAM



Learning algorithm

Classifier

Prediction on new
example

$$\mathbf{x}_{1000} = (1, 0, 1, 2, \dots)$$

y₁₀₀₀ = ???

Supervised Learning (Regression)

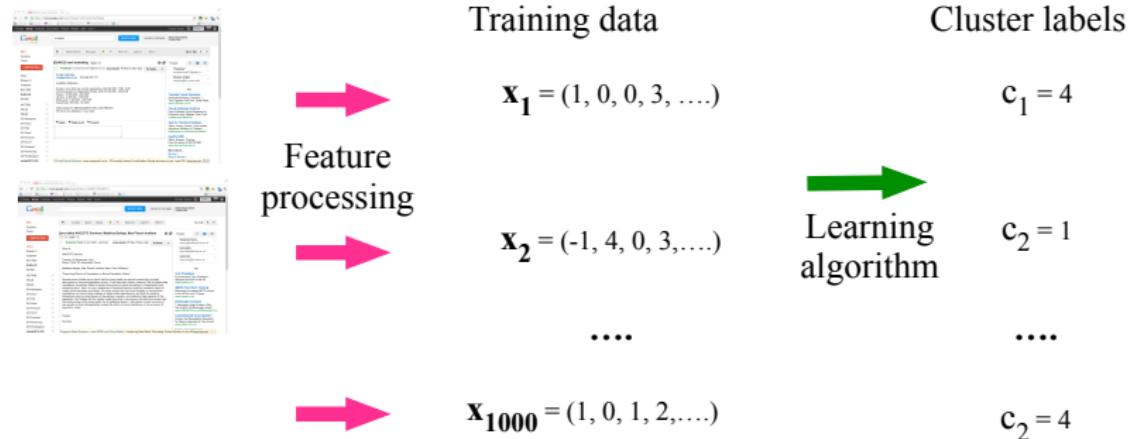
In this course we will talk about linear regression

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \dots + w_D x_D$$

- ▶ $\mathbf{x} = (x_1, \dots, x_D)^T$
- ▶ Here the assumption is that $f(\mathbf{x})$ is a linear function in \mathbf{x}
- ▶ The specific setting of the parameters w_0, w_1, \dots, w_D is done by minimizing a score function
- ▶ Usual score function is $\sum_{i=1}^n (y^i - f(\mathbf{x}^i))^2$ where the sum runs over all training cases
- ▶ We will cover linear regression later in the course

Unsupervised Learning

In this class we will focus on one kind of unsupervised learning, clustering.



General structure of supervised learning algorithms

Hand, Mannila, Smyth (2001)

- ▶ Define the **task**
- ▶ Decide on the **model structure** (choice of inductive bias)
- ▶ Decide on the **score function** (judge quality of fitted model)
- ▶ Decide on **optimization/search method** to optimize the score function

Inductive bias

- ▶ Supervised learning is inductive, i.e. we make generalizations about the form of $f(\mathbf{x})$ based on instances \mathcal{D}
- ▶ Let $f(\mathbf{x}; L, \mathcal{D})$ be the function learned by algorithm L with data \mathcal{D}
- ▶ Learning is impossible without making assumptions about f !!

The futility of bias-free learning



1



0



???

The futility of bias-free learning

- ▶ A learner that makes no *a priori* assumptions regarding the target concept has no rational basis for classifying any unseen examples (Mitchell, 1997, p 42)
- ▶ The *inductive bias* of a learner is the set of prior assumptions that it makes (we will not define this formally)
- ▶ We will consider a number of different supervised learning methods in the IAML; these correspond to different inductive biases

Machine Learning and Statistics

- ▶ A lot of work in machine learning can be seen as a rediscovery of things that were known in statistics; but there are also flows in the other direction
- ▶ The emphasis is rather different. One difference is a focus on *prediction* in machine learning vs *interpretation* of the model in statistics
- ▶ Until recently, machine learning usually referred to tasks associated with artificial intelligence (AI) such as recognition, diagnosis, planning, robot control, prediction, etc. These provide rich and interesting tasks
- ▶ Today interesting machine learning tasks abound.
- ▶ Goals can be autonomous machine performance, or enabling humans to learn from data (data mining).

Provisional Course Outline

- ▶ Introduction (today)
- ▶ Basic probability
- ▶ Thinking about data
- ▶ Naïve Bayes classification
- ▶ Decision trees
- ▶ Linear regression
- ▶ Generalization and Overfitting
- ▶ Linear classification: logistic regression, perceptrons
- ▶ Kernel classifiers: support vector machines
- ▶ Dimensionality reduction (PCA etc)
- ▶ Performance evaluation
- ▶ Clustering (k -means, hierarchical)
- ▶ Neural Networks

Maths Level

- ▶ Machine learning generally involves a significant number of mathematical ideas and a significant amount of mathematical manipulation
- ▶ IAML aims to keep the maths level to a minimum, explaining things more in terms of higher-level concepts, and developing understanding in a procedural way (e.g. how to program an algorithm)
- ▶ For those wanting to pursue research in any of the areas covered you will need courses like PMR, MLPR & MLP

IAML: Basic Maths, Probability and Estimation

Nigel Goddard
School of Informatics

Semester 1

Why Maths?

- ▶ IAML is focused on intuition and algorithms, not theory
- ▶ But sometimes you need maths to express the algorithms
- ▶ e.g., We represent training instances via vectors ($\mathbf{x} \in \mathbb{R}^k$), and linear functions of them as matrices
- ▶ Your first-year courses covered this stuff
 - ▶ But unlike many Informatics courses, we actually use it!

Functions, logarithms and exponentials

- ▶ Defining functions.
- ▶ Variable change in functions.
- ▶ Evaluation of functions.
- ▶ Combination rules for exponentials and logarithms.
- ▶ Properties of exponential and logarithm.

Vectors

- ▶ Scalar (dot) product, transpose.
- ▶ Basis vectors, unit vectors, vector length.
- ▶ Orthogonality, gradient vector, planes and hyper-planes.

Matrices

- ▶ Matrix addition, multiplication
- ▶ Matrix inverse, determinant.
- ▶ Linear transformation of vectors
- ▶ Eigenvalues, eigenvectors, symmetric matrices.

- ▶ General rules for differentiation of standard functions, product rule, function of function rule.
- ▶ Partial differentiation
- ▶ Definition of integration
- ▶ Integration of standard functions.

Probability and Statistics

We will go over these, but useful if you have seen these before.

- ▶ Probability, events
- ▶ Mean, variance, covariance
- ▶ Conditional probability
- ▶ Combination rules for probabilities
- ▶ Independence, conditional independence

Why Probability?

Probability is a branch of mathematics concerned with the analysis of uncertain (random) events

Examples of uncertain events

- ▶ Gambling: Cards, dice, etc.
- ▶ Whether my first grandchild will be a boy or a girl¹
- ▶ The number of children born in the UK last year
- ▶ The title of the next slide

Notice that

- ▶ Uncertainty depends on what you know already
- ▶ Whether something is “uncertain” is a pragmatic decision

¹I have no grandchildren currently, but I do have children

Why Probability in Machine Learning?

The training data is a source of uncertainty.

- ▶ Noise. e.g., Sensor networks, robotics
- ▶ Sampling error. e.g., Choice of training documents from the Web

Many learning algorithms use probabilities explicitly

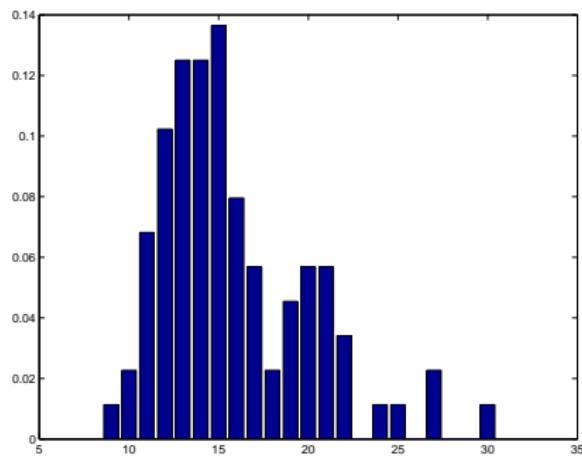
Ones that don't are still often *analyzed* using probabilities.

Random Variables

- ▶ The set of all possible outcomes of an experiment is called the *sample space*, denoted by Ω
- ▶ Events are subsets of Ω (often singletons)
- ▶ A random variable takes on values from a collection of *mutually exclusive* and *collectively exhaustive* states, where each state corresponds to some event
- ▶ A random variable X is a map from the sample space to the set of states
- ▶ Examples of variables
 - ▶ Colour of a car *blue, green, red*
 - ▶ Number of children in a family $0, 1, 2, 3, 4, 5, 6, > 6$
 - ▶ Toss two coins, let $X = (\text{number of heads})^2$. What values can X take?

Discrete Random Variables

Random variables (RVs) can be *discrete* or *continuous*.



- ▶ Use capital letters to denote random variables and lower case letters to denote values that they take, e.g. $p(X = x)$. Often shortened to $p(x)$.
- ▶ $p(x)$ is called a *probability mass function*.
- ▶ For discrete RVs: $\sum_x p(x) = 1$.

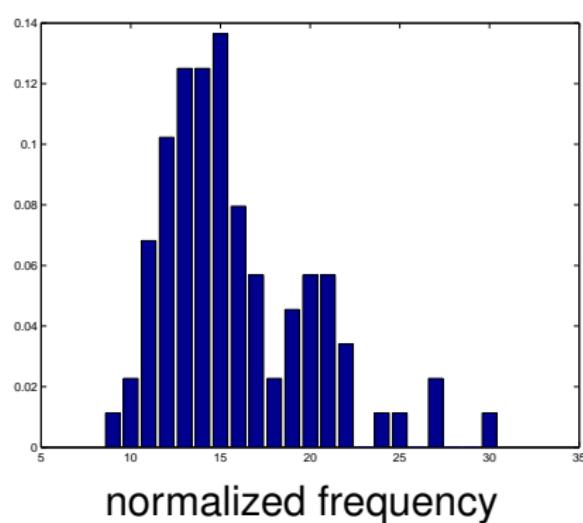
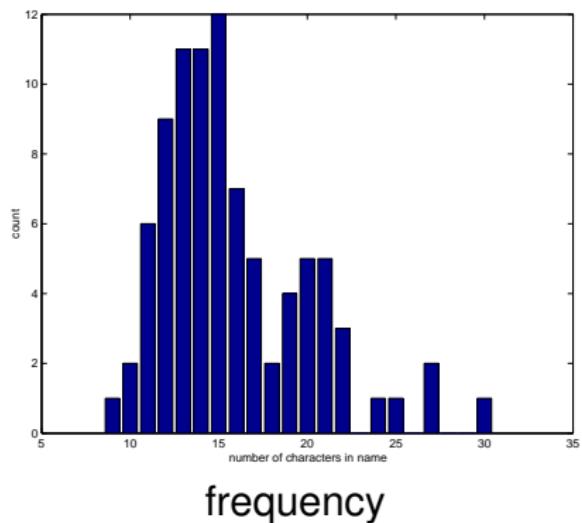
Examples: Discrete Distributions

- ▶ Example 1: Coin toss: 0 or 1
- ▶ Example 2: Have data for the number of characters in names of 88 people submitting tutorial requests:

9	10	10	11	11	11	11	11	11	12	12	12	12	12	12	12	12
12	12	12	13	13	13	13	13	13	13	13	13	13	13	13	13	13
14	14	14	14	14	14	14	14	14	14	14	14	14	15	15	15	15
15	15	15	15	15	15	15	15	15	16	16	16	16	16	16	16	16
16	16	17	17	17	17	17	18	18	19	19	19	19	19	20		
20	20	20	20	21	21	21	21	21	22	22	22	24	25			
27	27	30														

- ▶ Example 3: Third word on this slide.

Frequency



Joint distributions

- ▶ Suppose X and Y are two random variables. X takes on the value *yes* if the word “password” occurs in an email, and *no* if this word is not present. Y takes on the values of *ham* and *spam*
- ▶ This example relates to “spam filtering” for email

		$Y = \text{ham}$	$Y = \text{spam}$
$X = \text{yes}$	0.01	0.25	
	0.49	0.25	

- ▶ Notation

$$p(X = \text{yes}, Y = \text{ham}) = 0.01$$

Marginal Probabilities

The *sum rule*

$$p(X) = \sum_y p(X, Y)$$

e.g. $P(X = \text{yes}) = ?$

Marginal Probabilities

The *sum rule*

$$p(X) = \sum_y p(X, Y)$$

e.g. $P(X = \text{yes}) = ?$

Similarly:

$$p(Y) = \sum_x p(X, Y)$$

e.g. $P(Y = \text{ham}) = ?$

Conditional Probability

- ▶ Let \mathbf{X} and \mathbf{Y} be two disjoint subsets of variables, such that $p(\mathbf{Y} = \mathbf{y}) > 0$. Then the *conditional probability distribution* (CPD) of \mathbf{X} given $\mathbf{Y} = \mathbf{y}$ is given by

$$p(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}) = p(\mathbf{x} | \mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})}$$

- ▶ Gives us the *product rule*

$$p(\mathbf{X}, \mathbf{Y}) = p(\mathbf{Y})p(\mathbf{X} | \mathbf{Y}) = p(\mathbf{X})p(\mathbf{Y} | \mathbf{X})$$

- ▶ **Example:** In the ham/spam example, what is $p(X = \text{yes} | Y = \text{ham})$?
- ▶ $\sum_{\mathbf{x}} p(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}) = 1$ for all \mathbf{y}

Bayes' Rule

- ▶ From the product rule,

$$p(\mathbf{Y}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{Y})p(\mathbf{Y})}{p(\mathbf{X})}$$

- ▶ From the sum rule the denominator is

$$p(\mathbf{X}) = \sum_y p(\mathbf{X}|\mathbf{Y})p(\mathbf{Y})$$

- ▶ Say that \mathbf{Y} denotes a class label, and \mathbf{X} an observation. Then $p(\mathbf{Y})$ is the *prior* distribution for a label, and $p(\mathbf{Y}|\mathbf{X})$ is the *posterior* distribution for \mathbf{Y} given a datapoint \mathbf{x} .

Independence

- ▶ Independence means that one variable does not affect another, X is (*marginally*) *independent* of Y if

$$p(X|Y) = P(X)$$

- ▶ This is equivalent to saying

$$p(X, Y) = p(X)p(Y)$$

(can show this from definition of conditional probability)

- ▶ X_1 is *conditionally independent* of X_2 given Y if

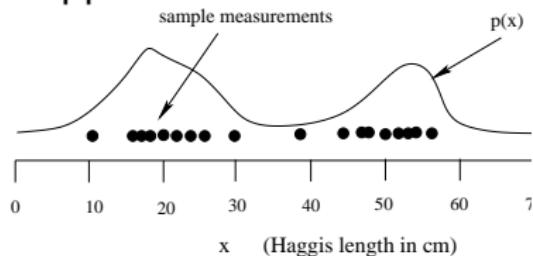
$$p(X_1|X_2, Y) = p(X_1|Y)$$

(i.e., once I know Y , knowing X_2 does not provide additional information about X_1)

- ▶ These are different things. Conditional independence does not imply marginal independence, nor vice versa.

Continuous Random Variables

Suppose we want random values in \mathbb{R} . Example:



- ▶ Formally, a continuous random variable X is a map $X : \Sigma \rightarrow \mathbb{R}$.
- ▶ In continuous case, $p(x)$ is called a *density function*
- ▶ Get the probability $\Pr\{X \in [a, b]\}$ by integration

$$\Pr\{X \in [a, b]\} = \int_a^b p(x)dx$$

- ▶ Always true: $p(x) > 0$ for all x and $\int p(x)dx = 1$ (*cf* discrete case).
- ▶ Bayes' rule, conditional densities, joint densities work exactly as in the discrete case.

Mean, variance

For a continuous RV

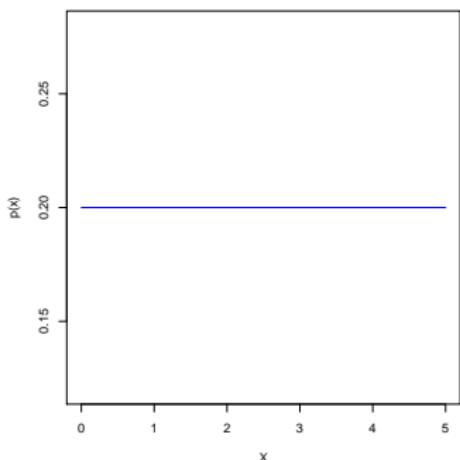
$$\mu = \int xp(x)dx \quad \sigma^2 = \int (x - \mu)^2 p(x)dx$$

- ▶ μ is the *mean*
- ▶ σ^2 is the *variance*
- ▶ For numerical discrete variables, convert integrals to sums
- ▶ Also written: $EX = \int xp(x)dx$ for the mean and
- ▶ $VX = E(X - \mu)^2 = \int (x - \mu)^2 p(x)dx$ for the variance

Example: Uniform Distribution

Let X be a continuous random variable on $[0, N]$ such that “all points are equally likely.”

This is called the uniform distribution on $[0, N]$. Its density is



$$p(x) = \begin{cases} \frac{1}{N} & \text{if } x \in [0, N] \\ 0 & \text{otherwise} \end{cases}$$

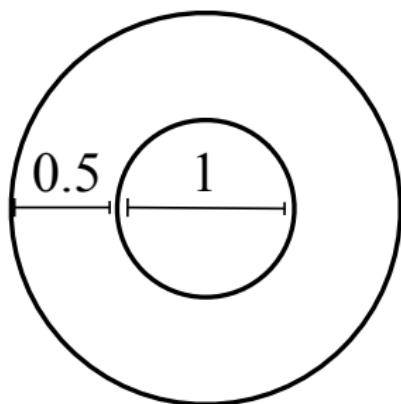
What is EX ? What is VX ?

Quiz Question

- ▶ Let X be a continuous random variable with density p .
- ▶ Need it be true that $p(x) < 1$?

Example: Another Uniform Distribution

Imagine that I am throwing darts on a dartboard.



Let X be the x -position of the dart I throw, and Y be the y position. Assuming that the dart is equally likely to land anywhere on the board:

1. What is the probability it will land in the inner circle?
2. What is the joint density of X and Y ?

Gaussian distribution

- ▶ The most common (and most easily analyzed) distribution for continuous quantities is the Gaussian distribution.
- ▶ Gaussian distribution is often a reasonable model for many quantities due to various central limit theorems
- ▶ Gaussian is also called the normal distribution

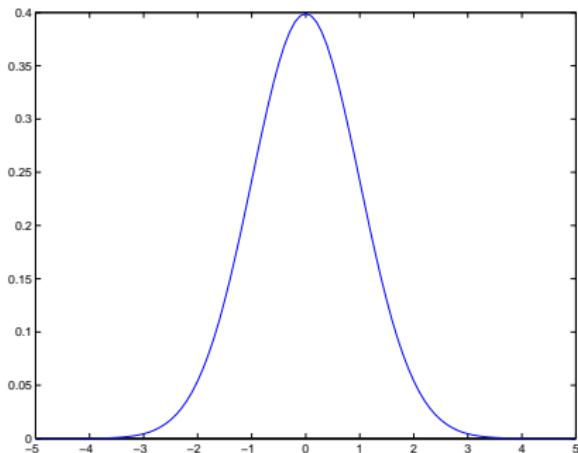
Definition

- ▶ The one-dimensional Gaussian distribution is given by

$$p(x|\mu, \sigma^2) = N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

- ▶ μ is the *mean* of the Gaussian and σ^2 is the *variance*.
- ▶ If $\mu = 0$ and $\sigma^2 = 1$ then $N(x; \mu, \sigma^2)$ is called a *standard Gaussian*.

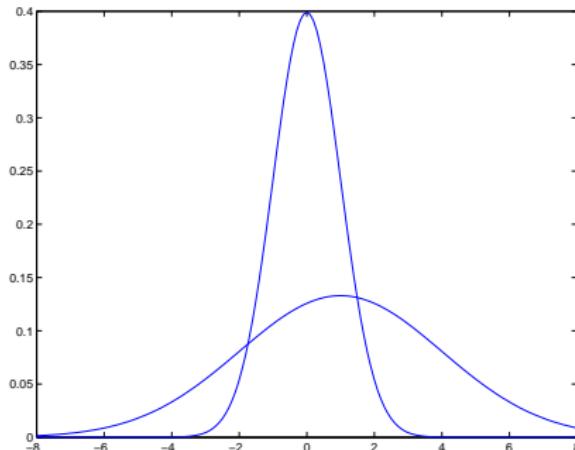
Plot



- ▶ This is a standard one dimensional Gaussian distribution.
- ▶ All Gaussians have a similar shape subject to scaling and displacement.
- ▶ If x is distributed $N(x; \mu, \sigma^2)$, then $y = (x - \mu)/\sigma$ is distributed $N(y; 0, 1)$.

Normalization

- ▶ Remember all distributions must integrate to one. The $\sqrt{2\pi\sigma^2}$ is called a normalization constant - it ensures this is the case.
- ▶ Hence tighter Gaussians have higher peaks:



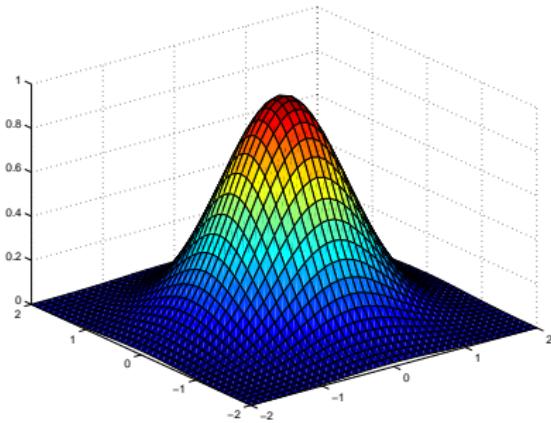
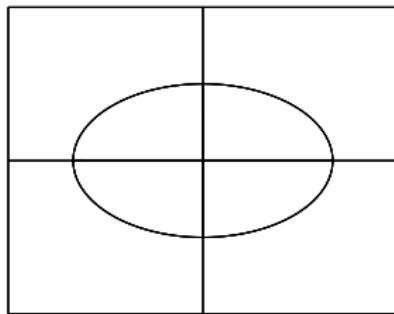
Bivariate Gaussian I

- ▶ Let $X_1 \sim N(\mu_1, \sigma_1^2)$ and $X_2 \sim N(\mu_2, \sigma_2^2)$
- ▶ If X_1 and X_2 are independent

$$p(x_1, x_2) = \frac{1}{2\pi(\sigma_1^2\sigma_2^2)^{1/2}} \exp \left\{ -\frac{1}{2} \left\{ \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} \right\} \right\}$$

- ▶ Let $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$, $\Sigma = \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}$

$$p(\mathbf{x}) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} \left\{ (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\} \right\}$$



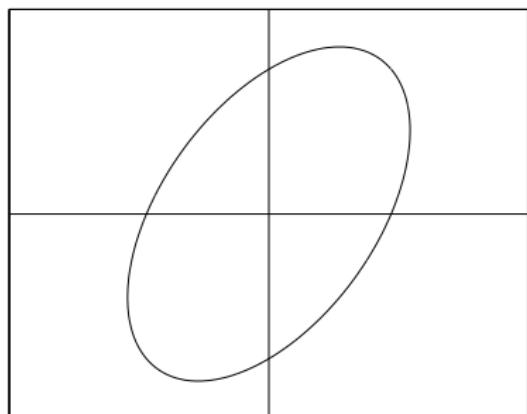
Bivariate Gaussian II

- ▶ Covariance
- ▶ Σ is the covariance matrix

$$\Sigma = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T]$$

$$\Sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

- ▶ Example: plot of weight vs height for a population



Multivariate Gaussian

- ▶ $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}) d\mathbf{x}$
- ▶ Multivariate Gaussian

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$

- ▶ Σ is the covariance matrix

$$\Sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

$$\Sigma = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$$

- ▶ Σ is symmetric
- ▶ Shorthand $\mathbf{x} \sim N(\boldsymbol{\mu}, \Sigma)$
- ▶ For $p(\mathbf{x})$ to be a density, Σ must be positive definite
- ▶ Σ has $d(d + 1)/2$ parameters, the mean has a further d

Inverse Problem: Estimating a Distribution

- ▶ But what if we don't know the underlying distribution?
- ▶ Want to *learn* a good distribution that fits the data we do have
- ▶ How is *goodness* measured?
- ▶ Given some distribution, we can ask how likely it is to have generated the data
- ▶ In other words what is the probability (density) of this particular data set given the distribution
- ▶ A particular distribution explains the data better if the data is more probable under that distribution

Likelihood

- ▶ $p(D|M)$. The probability of the data D given a distribution (or model) M . This is called the likelihood of the model.
- ▶ This is

$$p(D|M) = \prod_{i=1}^N p(\mathbf{x}_i|M)$$

i.e. the product of the probabilities of generating each data point individually.

- ▶ This is a result of the independence assumption.
- ▶ Try different M (different distributions). Pick the M with the highest likelihood → Maximum Likelihood Approach.

Bernoulli distribution

- ▶ Data 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1, total of 20 observations
- ▶ Three hypotheses:
 - ▶ $M = 1$ - Generated from a fair coin. 1=H, 0=T
 - ▶ $M = 2$ - Generated from a die throw 1=1, 0 = 2,3,4,5,6
 - ▶ $M = 3$ - Generated from a double headed coin 1=H, 0=T
- ▶ Likelihood of data. Let c =number of ones:

$$\prod p(x_i|M) = p(1|M)^c p(0|M)^{20-c}$$

- ▶ $M = 1$: Likelihood is $0.5^{20} = 9.5 \times 10^{-7}$
- ▶ $M = 2$: Likelihood is $(1/6)^9 (5/6)^{11} = 1.3 \times 10^{-8}$
- ▶ $M = 3$: Likelihood is $1^9 0^{11} = 0$

Bernoulli distribution

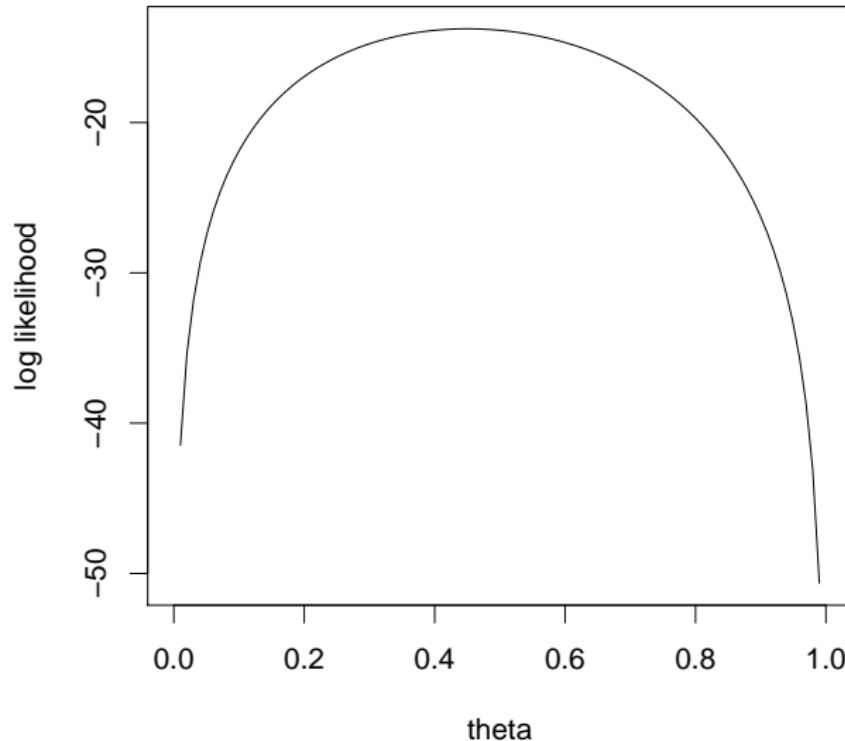
- ▶ Data 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1.
- ▶ Continuous range of hypotheses: $M = \theta$ generated from a Bernoulli distribution with $p(1|M = \theta) = \theta$.
- ▶ Likelihood of data. Let $c =$ number of ones in n tosses

$$\prod p(x_i|M = \theta) = \theta^c(1 - \theta)^{n-c}$$

- ▶ Maximum Likelihood hypothesis? Differentiate w.r.t. θ to find maximum
- ▶ In fact usually easier to differentiate $\log p(D|M)$: log is monotonic

$$\frac{d \log p(D|M)}{d\theta} = \frac{c}{\theta} - \frac{(n - c)}{(1 - \theta)}$$

- ▶ So $c(1 - \theta) - (n - c)\theta = 0$. This gives $\hat{\theta} = c/n$. Maximum likelihood result is intuitive



Notice this depends on the data set ($n = 20, c = 9$). With a different data set, you would get a different function of θ .

Maximum Likelihood Estimation for a Univariate Gaussian

- ▶ Suppose we have data $\{x_i, i = 1, 2, \dots, n\}$
- ▶ Suppose we presume the data was generated from a Gaussian with mean μ and variance σ^2 . Call this the model
- ▶ Then the log probability of the data given the model is

$$\log \prod_i p(x_i | \mu, \sigma^2) = -\frac{1}{2} \sum_i \frac{(x_i - \mu)^2}{\sigma^2} - \frac{n}{2} \log(2\pi\sigma^2)$$

Steps left as exercise: hint $\log \prod = \sum \log$

- ▶ Hence

$$\hat{\mu} = \frac{\sum_i x_i}{n}, \quad \hat{\sigma}^2 = \frac{\sum_i (x_i - \hat{\mu})^2}{n}$$

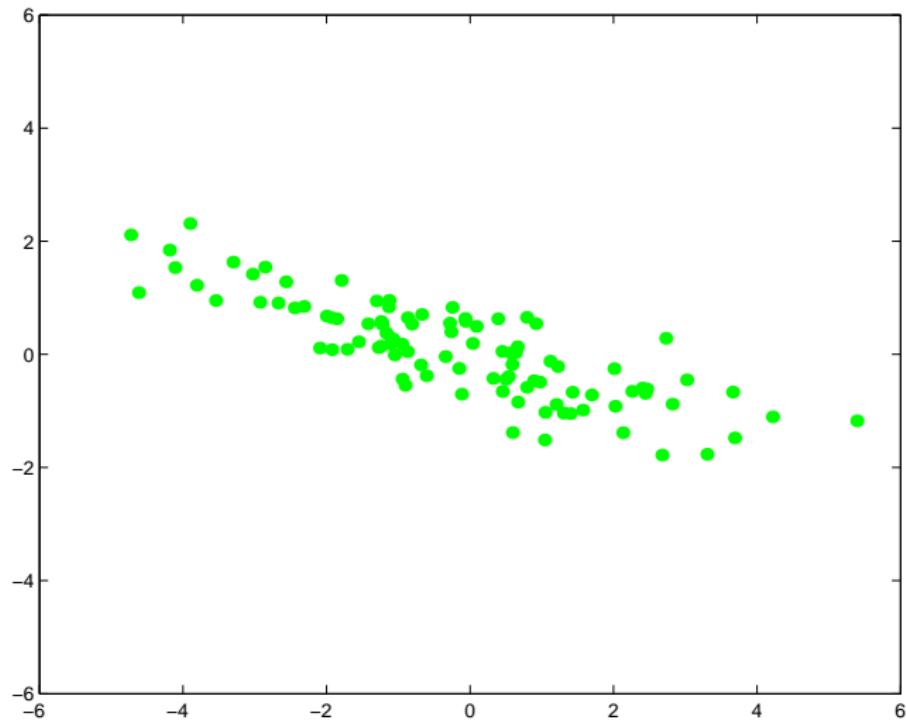
- ▶ (Maximum likelihood estimate of σ^2 is *biased*.)

Multivariate Gaussian: Maximum Likelihood

- ▶ The Maximum Likelihood estimate can be found in the same way
- ▶ $\hat{\mu} = (1/n) \sum_{i=1}^n \mathbf{x}_i$
- ▶ $\hat{\Sigma} = (1/n) \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$

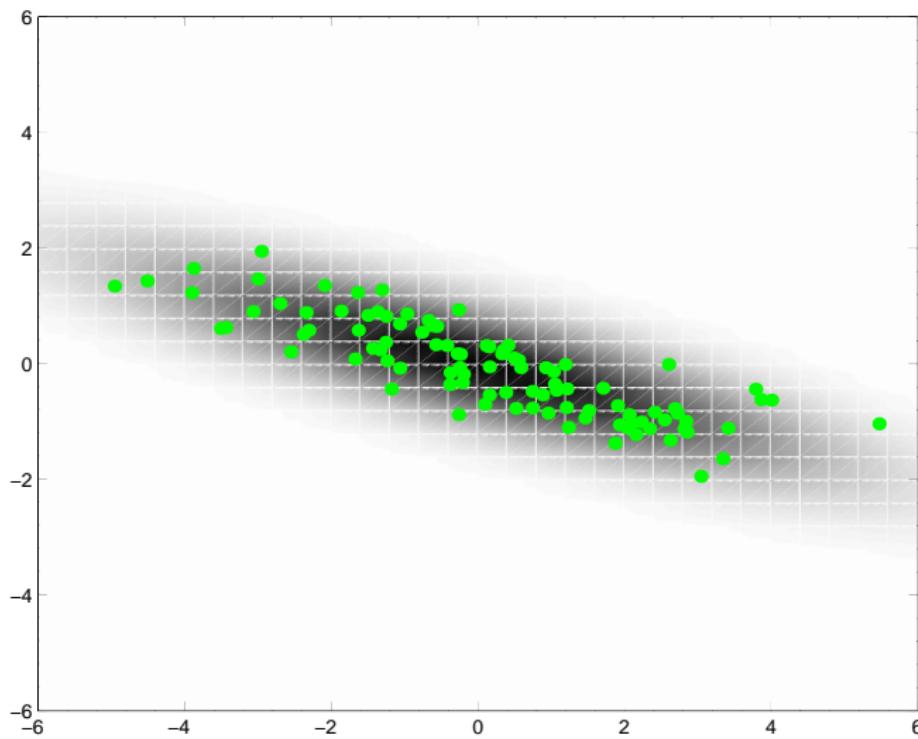
Example

- ▶ The data.



Example

- ▶ The data. The maximum likelihood fit.



Introductory Applied Machine Learning

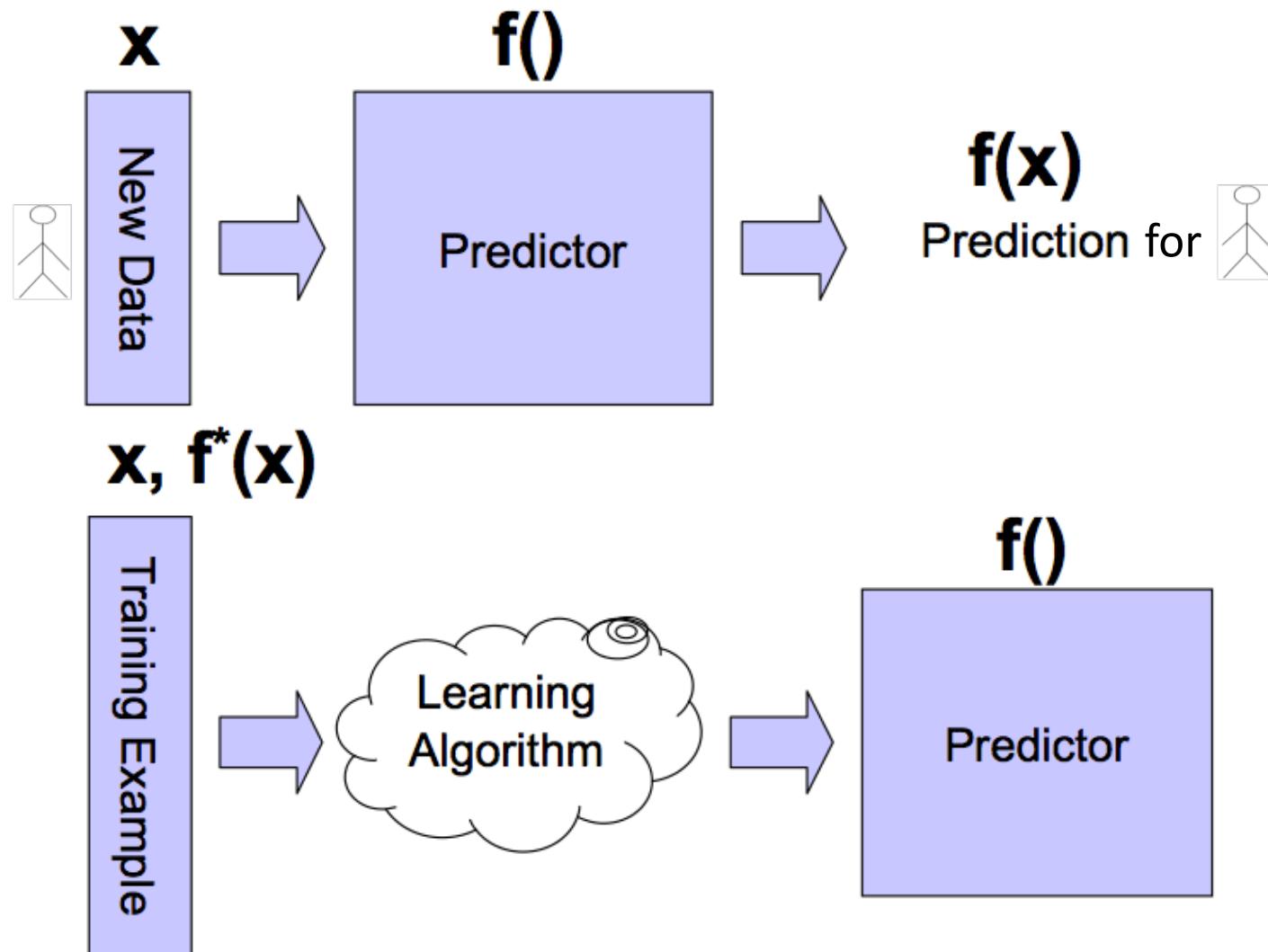
Thinking about Data

Victor Lavrenko and Nigel Goddard
School of Informatics
University of Edinburgh

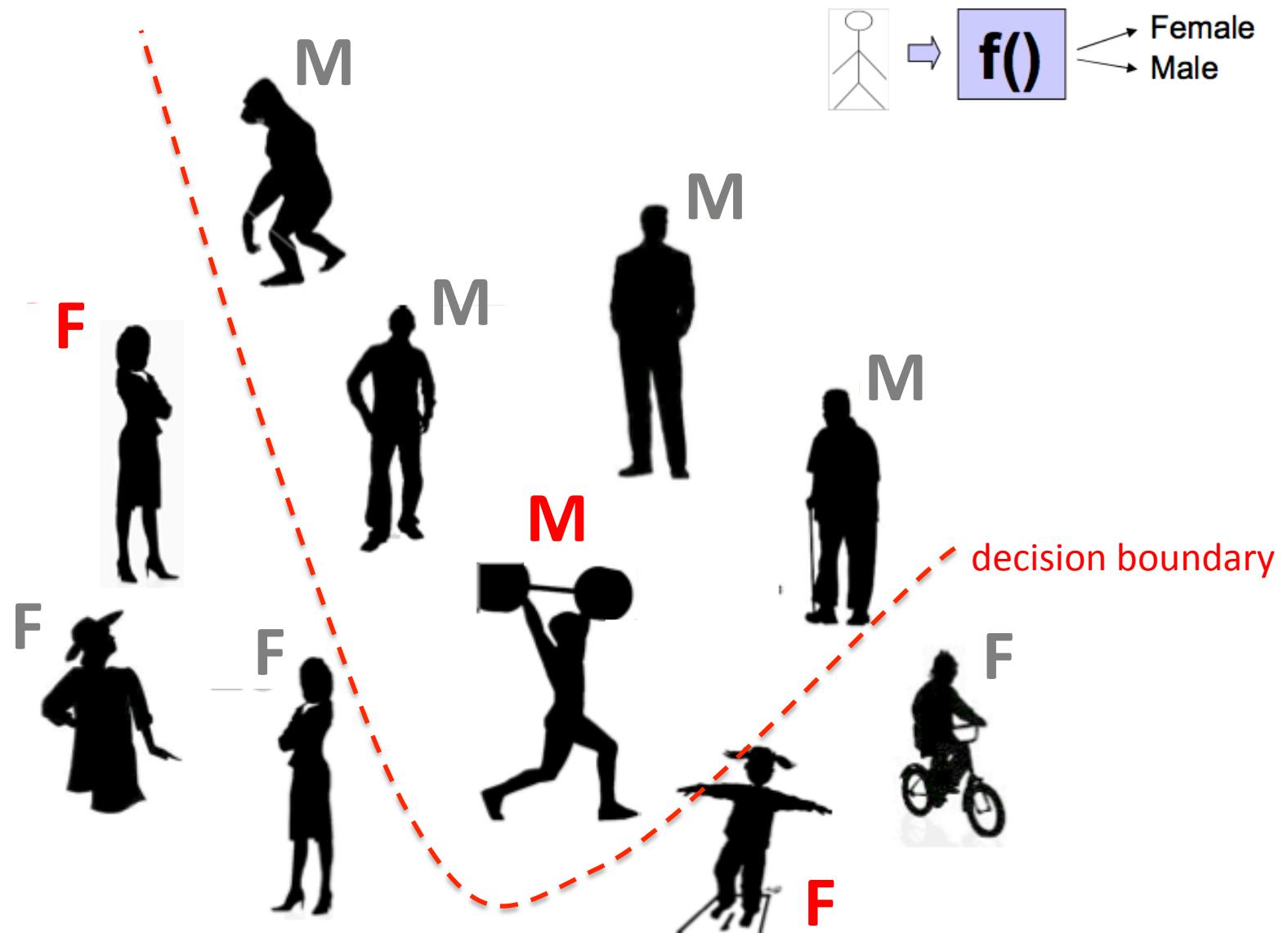
Overview

- What is machine learning?
 - examples: classification, regression, clustering
- Attribute-value pairs
 - bag-of-features representation
 - categorical attributes
 - ordinal attributes
 - numeric attributes, issues
- Examples of real data

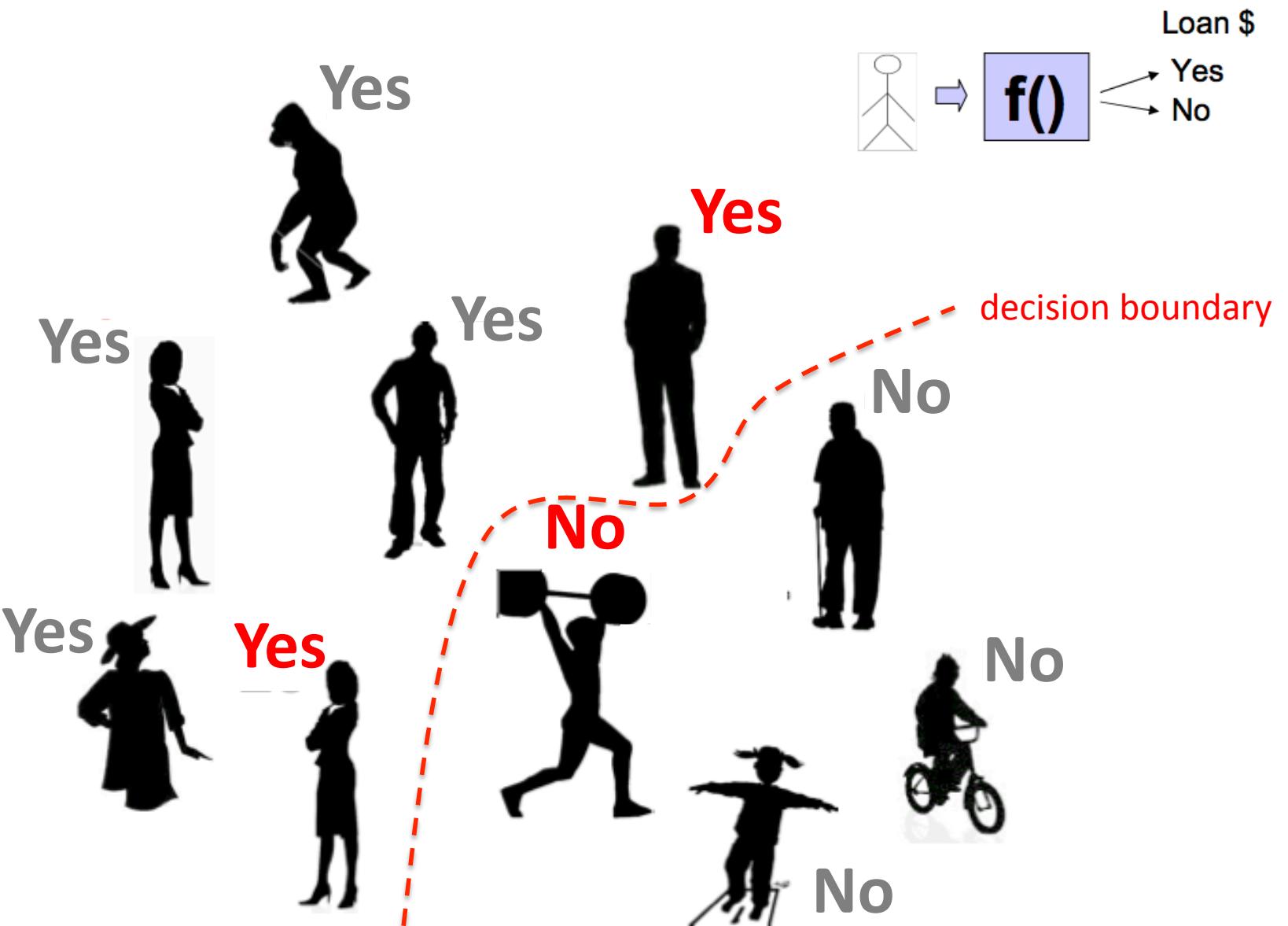
Learning from Examples



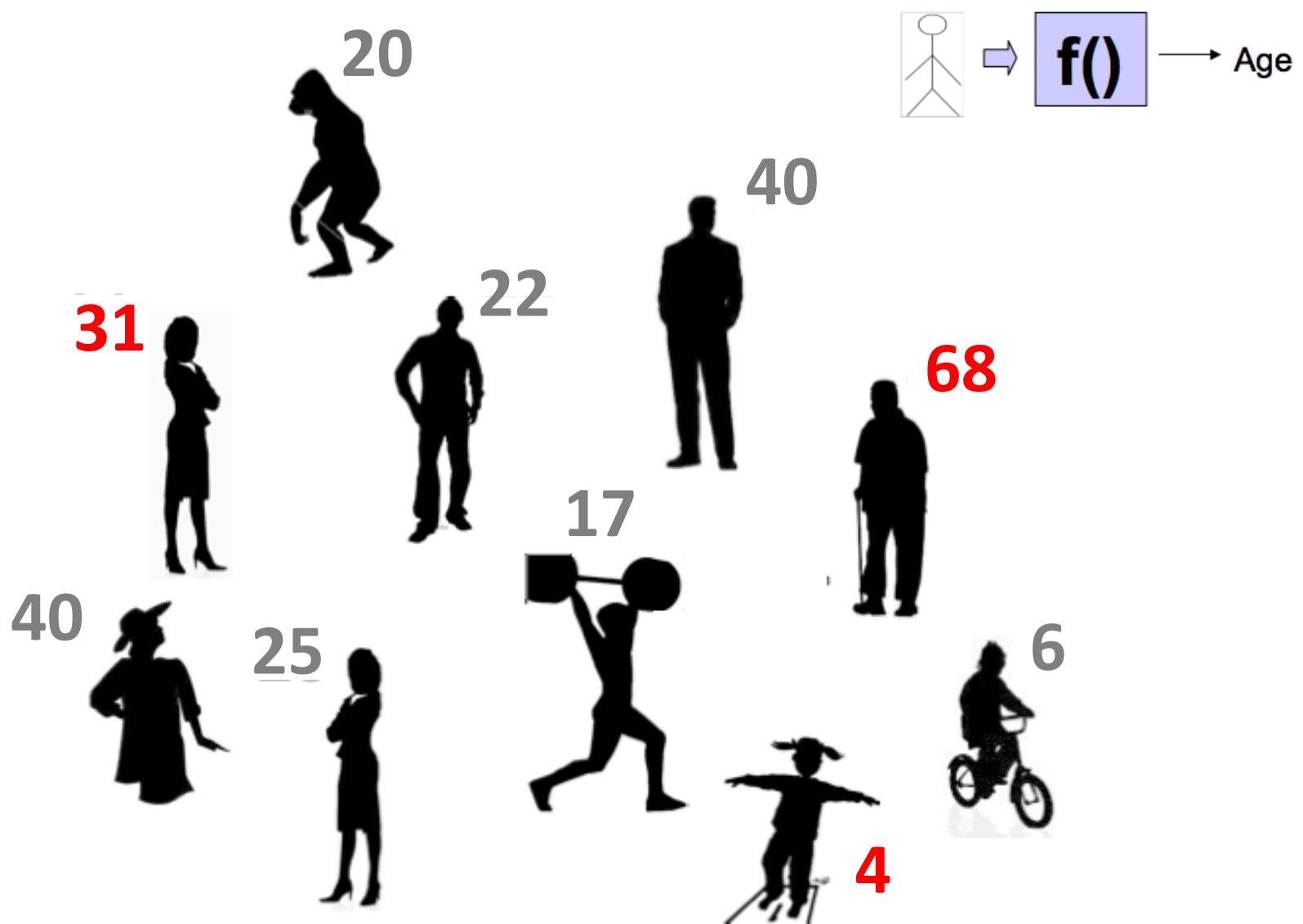
Classification (supervised learning)



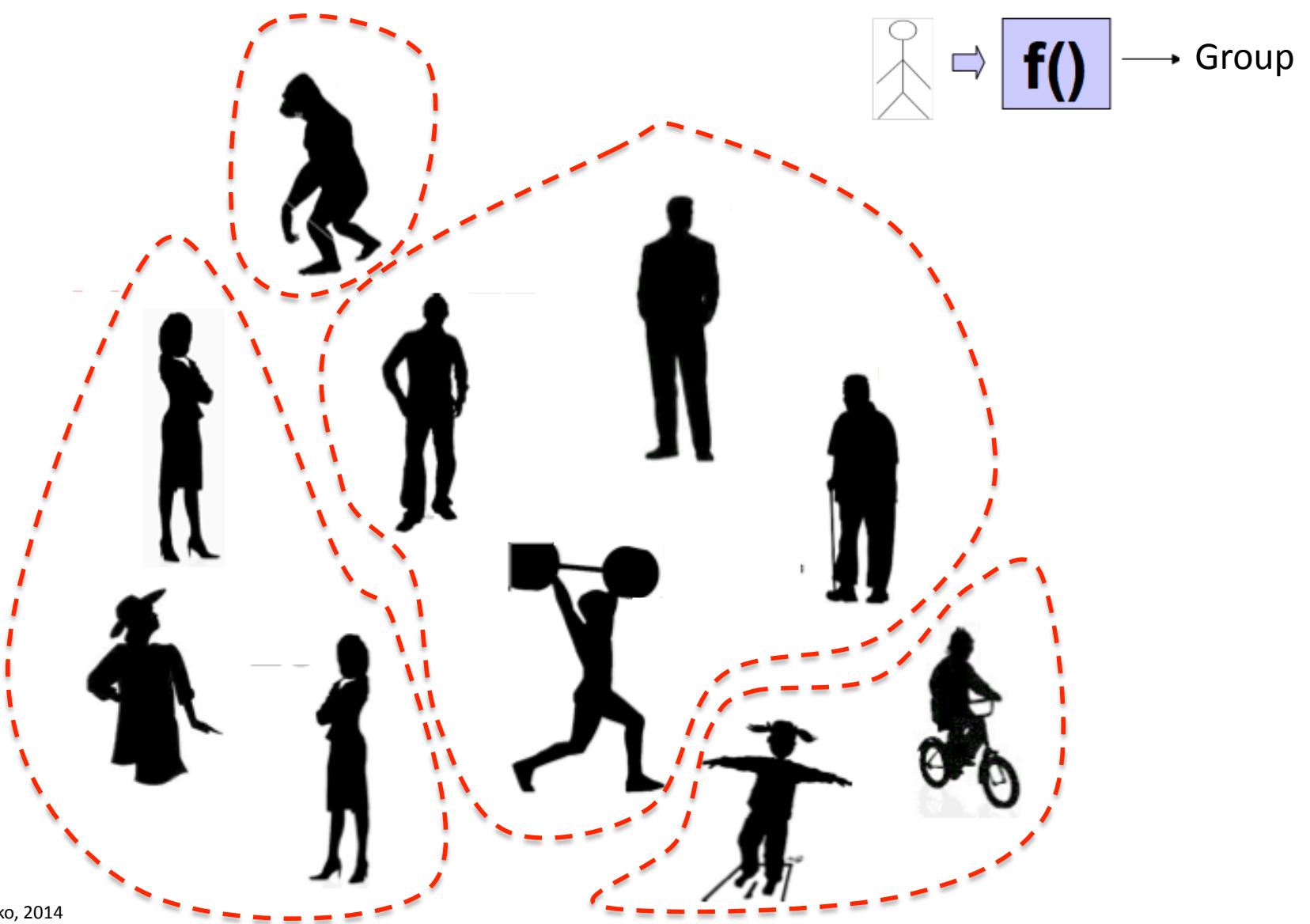
Classification (supervised learning)



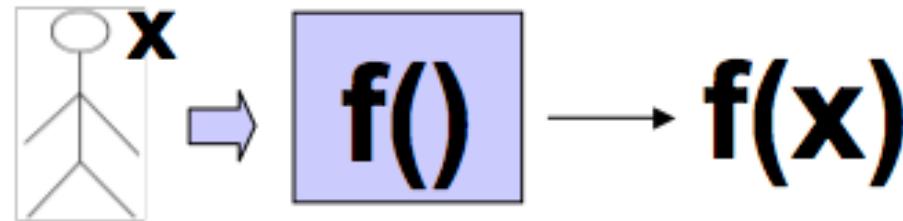
Regression (supervised learning)



Clustering (unsupervised learning)



Representing Data



- How do we represent mathematically?
- Depends on what we're trying to do:
 - deciding to loan money?
 - predicting gender?
- Represent as a set of attribute-value pairs
 - example: $x = \{\text{height}=\text{180cm}, \text{eyes}=\text{"blue"}, \text{job}=\text{"student"}\}$

Attribute-value pairs

- $x = \{\text{height}=\textcolor{blue}{180cm}, \text{eyes}=\textcolor{blue}{\text{"blue"}}, \text{job}=\textcolor{blue}{\text{"student"}}\}$
- un-ordered “bag-of-features”
 - if structure is essential – embed it in the attributes
- Have to convert any dataset to this form
- Generally three types of attributes:
 - categorical: *red, blue, brown, yellow*
 - ordinal: *poor, satisfactory, good, excellent*
 - numeric: *-3.14, 6E23, 0, 1*

Categorical attributes

- Each instance falls into one of a set of categories
 - **genre**: $\{\text{classical}, \text{jazz}, \text{rock}, \text{techno}\}$
 - categories are mutually exclusive
- Categories usually encoded as numbers
 - no natural ordering to categories
 - only equality testing ($=, \neq$) is meaningful
- Synonymy a major challenge for real datasets:
 - e.g. social tags: $\text{country} == \text{folk}$? $\text{house} == \text{techno}$?

Ordinal attributes

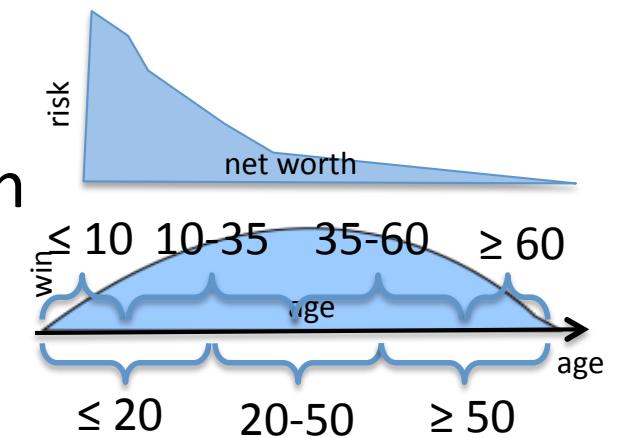
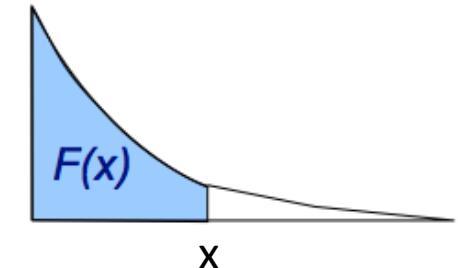
- Instance falls into one of a set of categories
- There is a natural ordering to categories
 - **education level:** $\{\text{none}, \text{school}, \text{university}, \text{post-graduate}\}$
 - **Likert scale:** $\{\text{disagree}, \text{neutral}, \text{agree}, \text{strongly agree}\}$
- Encoded as numbers to preserve ordering
 - meaningful to compare values: $(<, =, >)$
 - should not add / multiply / measure “distance”
- Sometimes hard to differentiate from categorical:
 - does $\{\text{single}, \text{married}, \text{divorced}\}$ have a natural ordering?

Numeric attributes

- Integers or real numbers
 - meaningful to add, multiply, compute mean / variance
 - integers not always the same as real numbers
- Usually want to normalize values (why?)
 - zero mean, unit variance: $x' = (x - \text{mean}) / \text{st.dev}$
 - sometimes want [0,1]: $x' = (x - \min) / (\max - \min)$
- Sensitive to extreme (unusually large/small) values
 - e.g. height: {165,167,171,175,176,181,183,1820}[cm]
 - must handle this before normalization

Numeric attributes: issues

- Skewed distributions
 - systematic extreme values
 - affects regression, kNN, NB; but not DTs
 - simple fix: $\log(x)$ or $\text{atan}(x)$, then normalize
 - cumulative distribution function: $x' = F(x) = P(X \leq x)$
- Non-monotonic effect of attributes
 - affects regression, NB, DTs(gain); less important for kNN
 - monotonic: net worth and lending risk
 - higher net worth \rightarrow lower lending risk
 - non-monotonic: age \rightarrow win a marathon
 - sweet spot: not too young, not too old
 - simple fix: quantization
 - can be unsupervised, overlapping



Overview

- Attribute-value pairs
- Examples of real data
 - credit scoring
 - handwritten digits
 - object recognition
 - text classification
- Issues to consider

Example: credit scoring

- Numeric attributes:
 - loan amount (e.g. *\$1000*)
 - installment / disposable income (e.g. *35%*)
- Ordinal:
 - savings: *{none, <100, 100..500, 500..1000, >1000}*
 - employed: *{unemployed, <1yr, 1..4yrs, 4..7yrs, >7yrs}*
- Categorical:
 - purpose: *{car, appliance, repairs, education, business}*
 - personal: *{single, married, divorced, separated}*
 - housing: *{for free, rents, owns}* ← perhaps ordinal?

Picking attributes

- Previous example: obvious attributes
 - not always the case (e.g. images)
- How do we pick a representation?
- Think about what we're trying to accomplish:
 - we're learning a predictor: $f(x) \rightarrow y$
 - x should encode some information relevant to y
 - idea: “similar” representations iff x_1, x_2 in the same class:
 - similar values for attributes if x_1, x_2 in the same class
 - dissimilar values if not
 - “similar” not always a straightforward concept
 - a good intuition for thinking about representations



Example: digit recognition

- Recognize handwritten digits
 - application: automatic postal code processing
- Offline process
 - input: bitmap image
 - no pen stroke data
- Challenges:
 - varying style, slant pressure, pen type

A grid of handwritten digits from 0 to 9, arranged in 10 rows and 10 columns. The digits are written in various styles, sizes, and orientations, illustrating the challenge of offline digit recognition.

3	6	8	1	7	9	6	6	9	1
6	7	5	7	8	6	3	4	8	5
2	1	7	9	7	1	2	8	4	5
4	8	1	9	0	1	8	8	9	4
7	6	1	8	6	4	1	5	6	0
7	5	9	2	6	5	8	1	9	7
1	2	2	2	2	3	4	4	8	0
0	2	3	8	0	7	3	8	5	7
0	1	4	6	4	6	0	2	4	3
7	1	2	8	7	6	9	8	6	1

Handwritten digits: attributes

- Represent each pixel as a separate attribute
 - 400 attributes (20x20 bitmap)
 - each attribute is a real number
 - degree of “blackness” of a pixel
 - could represent as binary (0,1)
 - 0 (white) if $x_i < t$, else 1 (black)
 - natural, space/CPU-efficient
 - thinking in terms of similarity
 - (0,1) will increase mismatches
 - may want to do the opposite: “blur” the image

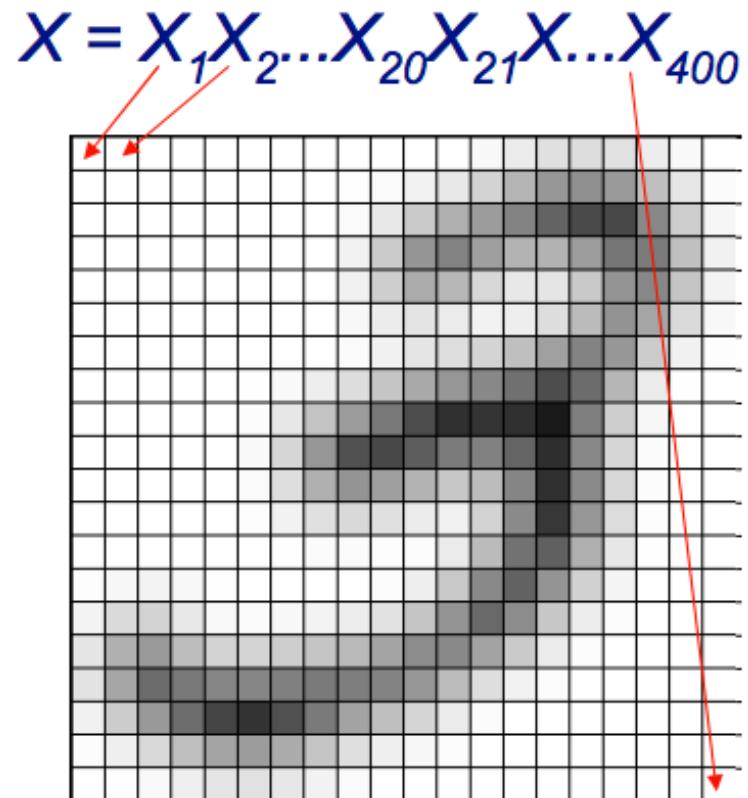
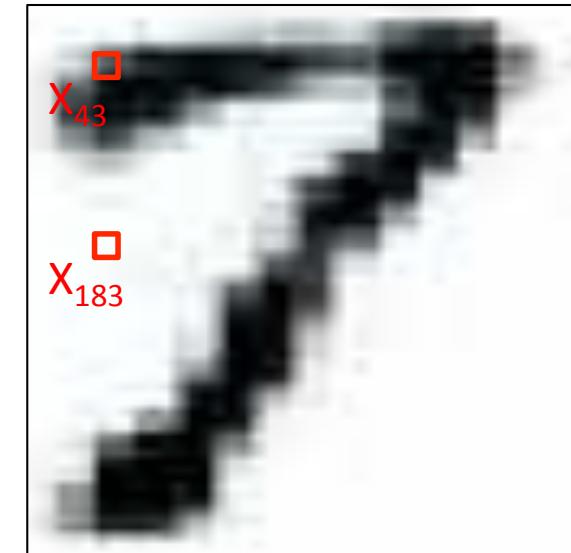
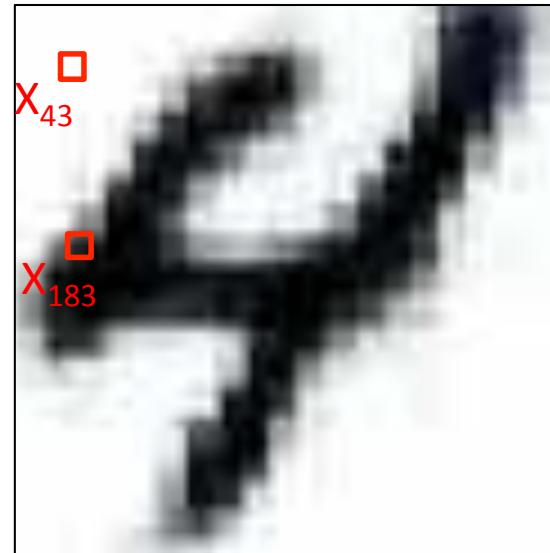
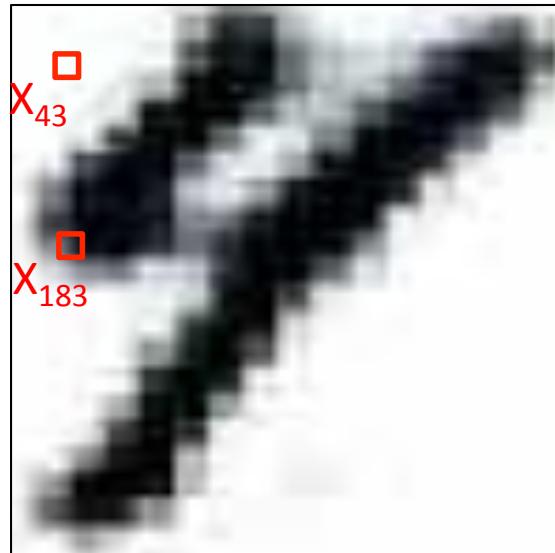
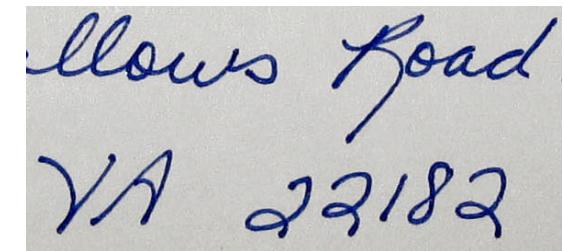


Image pixels as attributes

- works when same pixel = same meaning
 - X_{43} ... stroke in the upper-left corner



- true because we can isolate each digit, rescale, de-slant

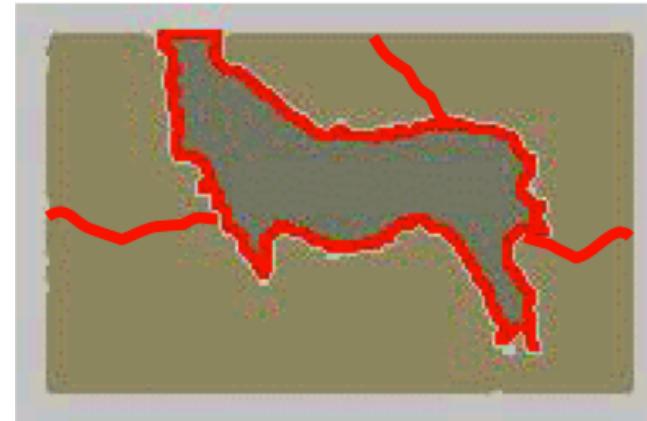


Example: object recognition

- Recognize object in an image
 - animals, faces, military targets
- Input is a photograph (bitmap)
- Challenges:
 - position in a photo, orientation, scale
 - lighting differences, obstructions
- Using pixels as attributes will not work
 - want something that makes different zebras “similar”
 - many ways to achieve this, will outline one possibility



Object recognition: attributes



position (x,y)
relative area
circumference
convexity
orientation
color freq.
texture filters

- Segment the image into “regions”
 - algorithms: BlobWorld, Normalized Cuts
- Compute features describing the region
- Segmentation will make errors
 - hope these errors are systematic (same for all zebras)
 - sometimes can get away with simple rectangular grid

Example: text classification

- Assign class label to a text document
 - detect spam, identify topics/genres, predict events
 - input: string of characters
 - idea: words carry meaning
- Naïve way: words as values

$X_1 = \text{this}$
 $X_2 = \text{proposition}$
 $X_3 = \text{on}$
 $X_4 = \text{behalf}$
 $X_5 = \text{of}$
 $X_6 = \text{mr}$
 $X_7 = \text{lee}$
 $X_8 = \text{kun}$



$X_1 = \text{this}$
 $X_2 = \text{investment}$
 $X_3 = \text{proposition}$
 $X_4 = \text{on}$
 $X_5 = \text{behalf}$
 $X_6 = \text{of}$
 $X_7 = \text{mr}$
 $X_8 = \text{lee}$



Dear Sir

This ~~investment~~ proposition on behalf of Mr Lee Kun Hee (former chairman of Samsung Electronics). He requires an experienced business person or company that can profitably invest monies in excess of Fifty Two million US Dollars (US\$52m) only, outside Asia. The sum of money will be paid from African Development Bank Group, South Africa...

Text classification: attributes

- Better way: words as numeric attributes
 - one attribute for **every possible word** in the language
 - value: 1 if word was observed in email, 0 otherwise
 - may use frequencies or tf-idf weights
 - note: 10^5 - 10^6 attributes, 99.99% zeros

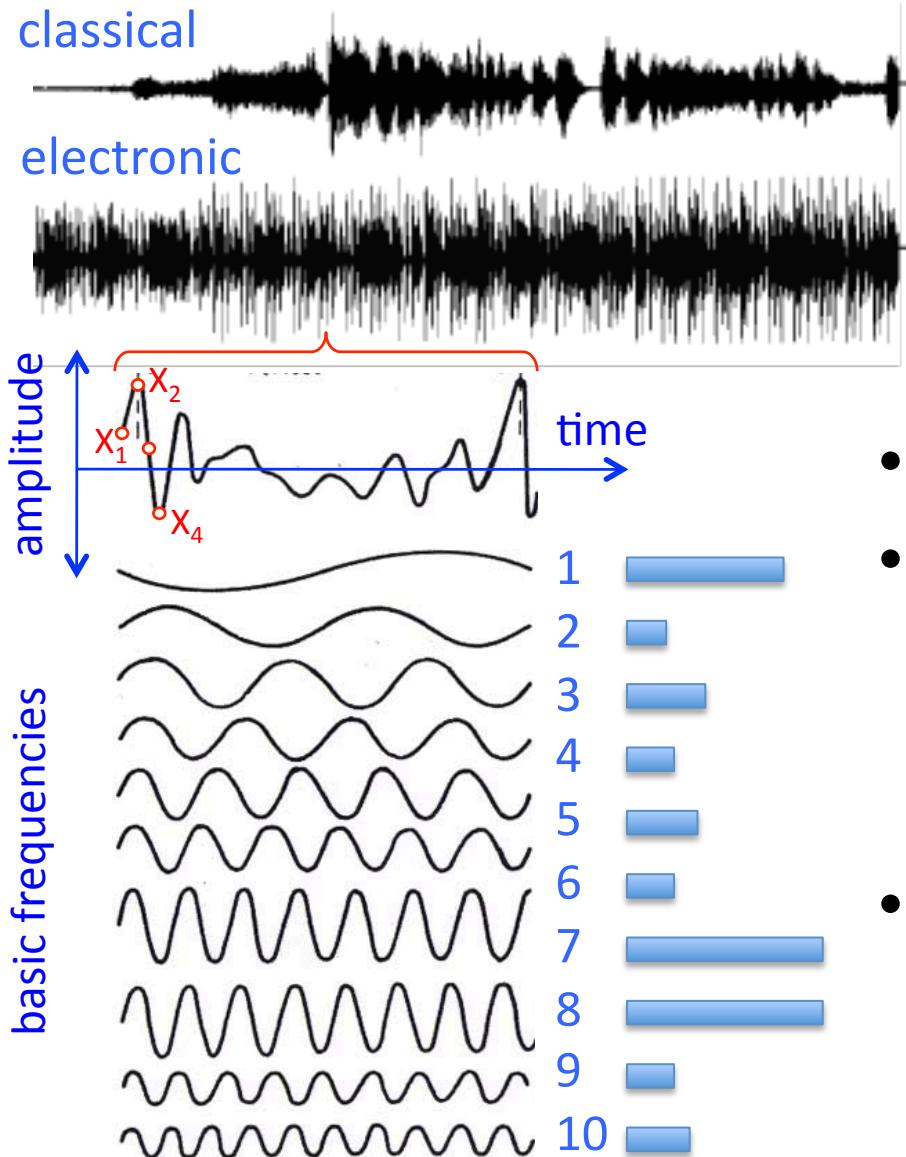
Dear Sir

This investment proposition on behalf of Mr Lee Kun Hee (former chairman of Samsung Electronics). He requires an experienced business person or company that can profitably invest monies in excess of Fifty Two million US Dollars (US\$52m) only, outside Asia. The sum of money will be paid from African Development Bank Group, South Africa...



0	aardvark
0	apple
1	africa
1	bank
0	bear
1	business
0	cat
1	funds
0	gorilla
1	investment
0	zebra
0	zoo

Example: music classification



- Music = time series
- Naive representation:
 - sample at regular intervals
 - X_t = amplitude at time t
- Periodic series = \sum sine waves
- Fourier transform:
 - decompose music into base frequencies f
 - find “weight” of each f
- Representation:
 - X_f = weight of frequency f
 - insensitive to shift, volume

Issues in Machine Learning

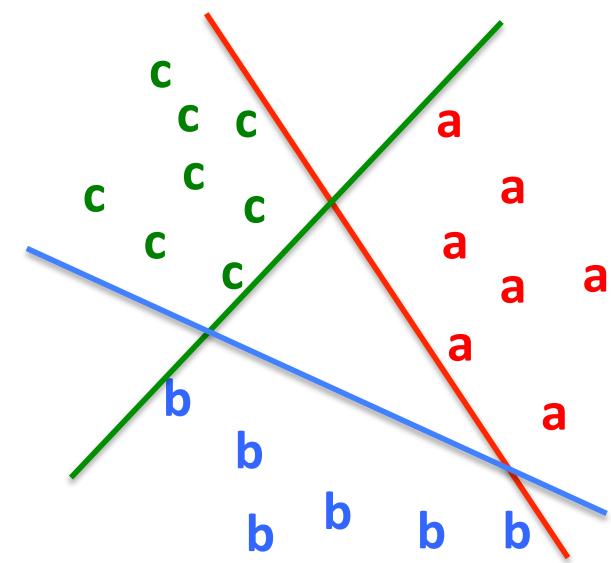
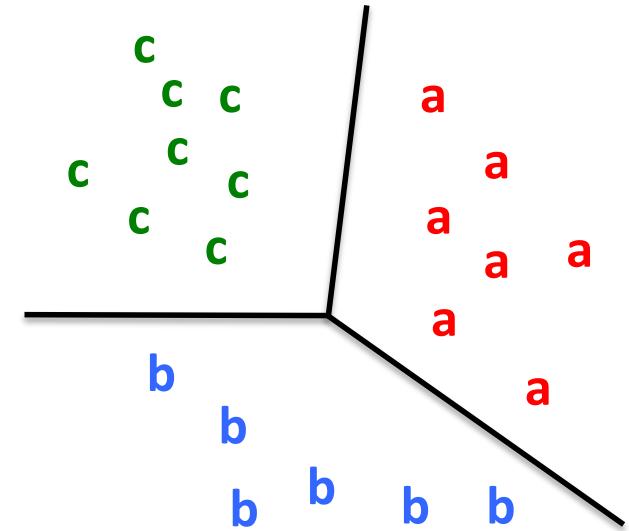
- Supervised vs. unsupervised
- What are we predicting?
- Outliers in the data
- Missing data
- Generative vs. discriminative
- Dimensionality

Supervised vs. Unsupervised

- Supervised learning:
 - trying to predict a specific quantity
 - have training examples with labels
 - can measure accuracy directly
- Unsupervised learning:
 - trying to “understand” the data
 - looking for structure or unusual patterns
 - not looking for something specific (supervised)
 - does not require labeled data
 - evaluation usually indirect or qualitative
- Semi-supervised:
 - using unsupervised methods to improve supervised algs.
 - usually few labeled examples + lots of unlabeled

Multi-class vs. Binary classification

- Multi-class:
 - classes mutually exclusive:
 - instance is either a or b or c
 - even if it's an outlier
 - NB, kNN, DT, logistic
- Binary:
 - one-vs-rest:
 - $\{a\}$ vs $\{\text{not } a\}$, $\{b\}$ vs $\{\text{not } b\}$
 - classes may overlap
 - instance can be both a and b
 - can be in none of the classes
 - SVM, logistic, perceptron

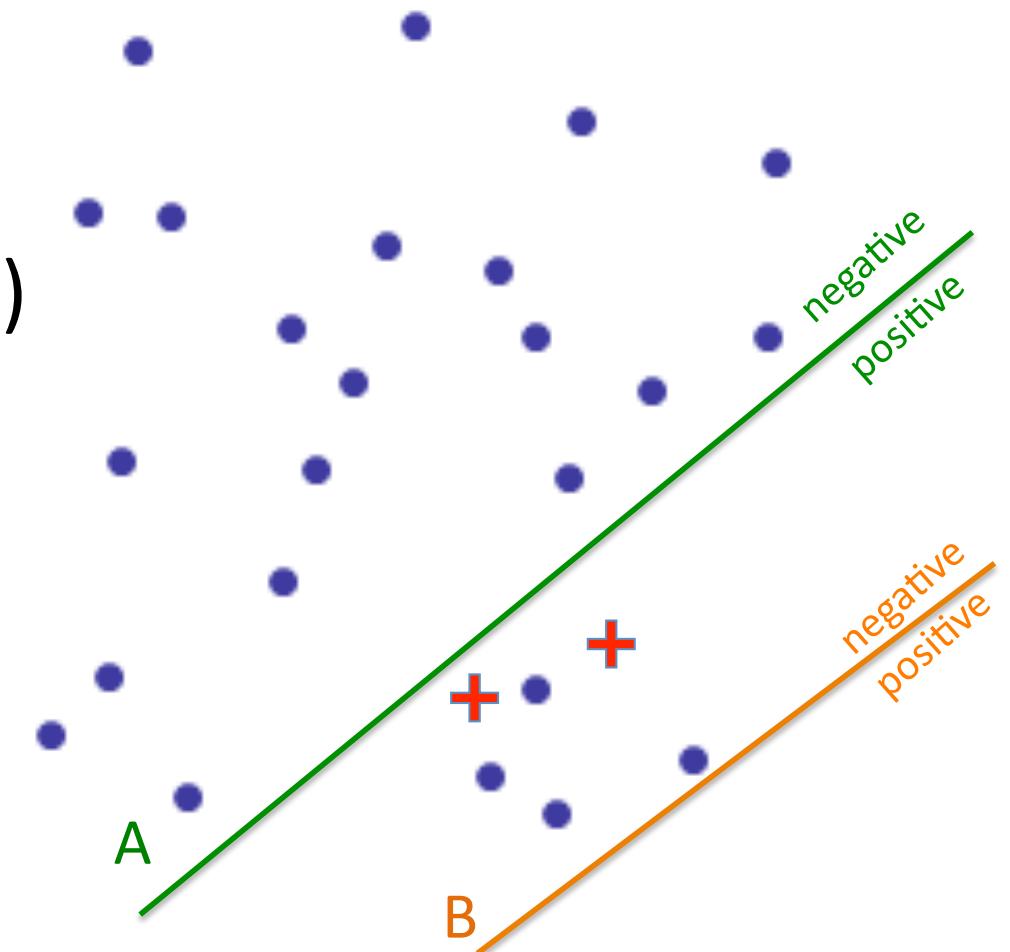


What are we predicting?

- Are there dominating classes?
 - does it affect anything?
- Example:
 - *Predict if scientific publication will lead to a Nobel prize*
 - claim: have a classifier that will be at least 99.99% accurate
- What is the appropriate error metric?
 - relative cost of false positives / false negatives
 - medical diagnosis vs. investment opportunities

Accuracy and un-balanced classes

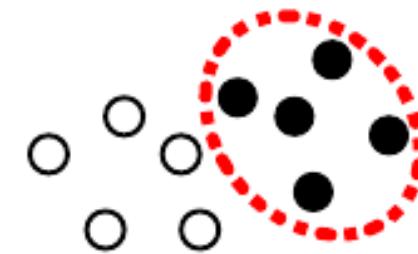
- You're predicting Nobel prize (+) vs. not (•)
- Would you prefer classifier A or B?
- Is accuracy (% correct) higher for A or B?
- Accuracy / error rate poor metric here
- Want:
 - cost (Miss) > cost (FA)



Generative vs. Discriminative

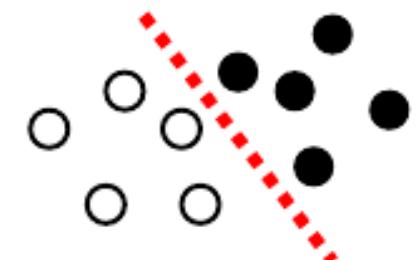
- Generative:

- probabilistic “model” of each class
- decision boundary:
 - where one model becomes more likely
 - natural use of unlabeled data



- Discriminative:

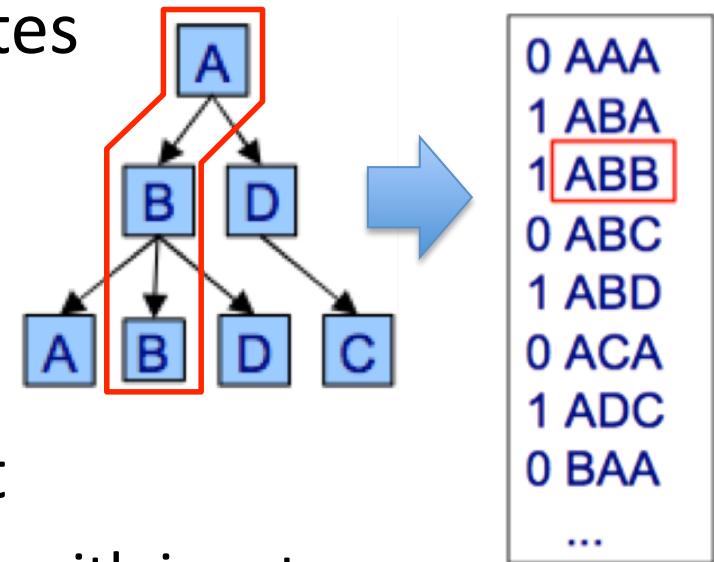
- focus on the decision boundary
- more powerful with lots of examples
- not designed to use unlabeled data
- only supervised tasks



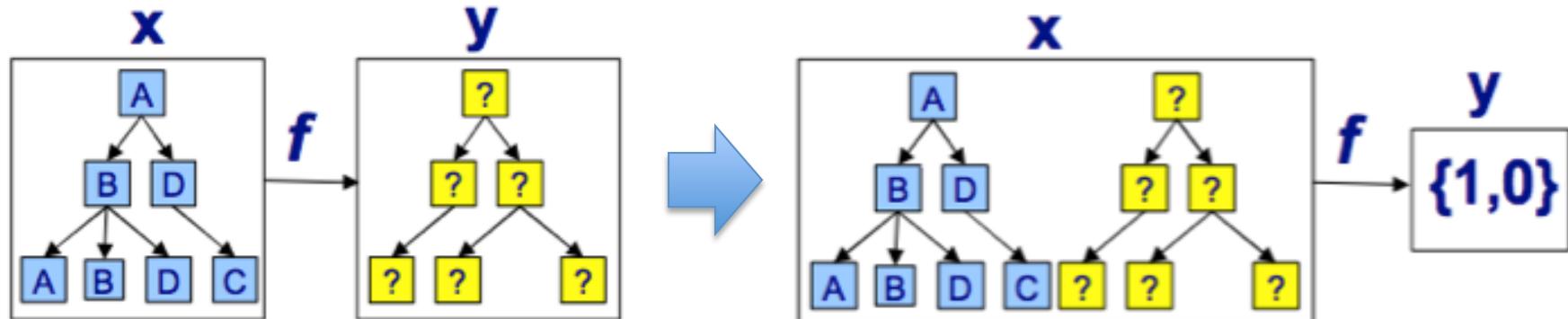
Dealing with structure

- Structured input: embed in attributes

- e.g. tree w. free branching, labels
 - meaning of “A” depends on level
 - one possible representation:
 - attributes = root-to-leaf paths

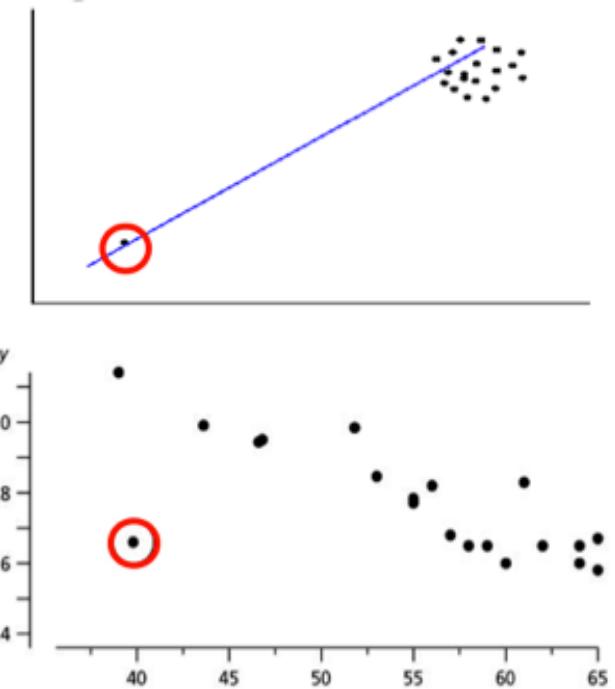


- Structured output: embed in input
 - predict 1/0: output does / doesn't go with input
 - search over possible outputs becomes main focus



Outliers in the data

- Isolated instances of a class that are unlike any other instance of that class
 - affect all learning methods to various degrees
- Extreme attribute values:
 - detect: confidence interval
 - remove or threshold
- Dissimilar to other instances
 - remove or try to fix (mis-labeled?)
- Always try to visualize the data
 - helps detect many irregularities



Introductory Applied Machine Learning

Naïve Bayes

Victor Lavrenko and Nigel Goddard
School of Informatics

Overview

- Naïve Bayes classifier
 - components and their function
 - independence assumption
 - dealing with missing data
- Continuous example
- Discrete example
- Pros and cons

Bayesian classification

- Goal: learning function $f(x) \rightarrow y$
 - y ... one of k classes (e.g. spam/ham, digit 0-9)
 - $x = x_1 \dots x_n$ – values of attributes (numeric or categorical)
- Probabilistic classification:
 - most probable class given observation: $\hat{y} = \arg \max_y P(y|x)$
- Bayesian probability of a class:

$$P(y|x) = \frac{\underbrace{P(x|y)P(y)}_{\text{class model prior}}}{\underbrace{\sum_{y'} P(x|y')P(y')}_{\text{normalizer } P(x)}}$$

Bayesian classification: components

$$P(y|x) = \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')}$$

Example:
y ... patient has Avian flu
x ... observed symptoms

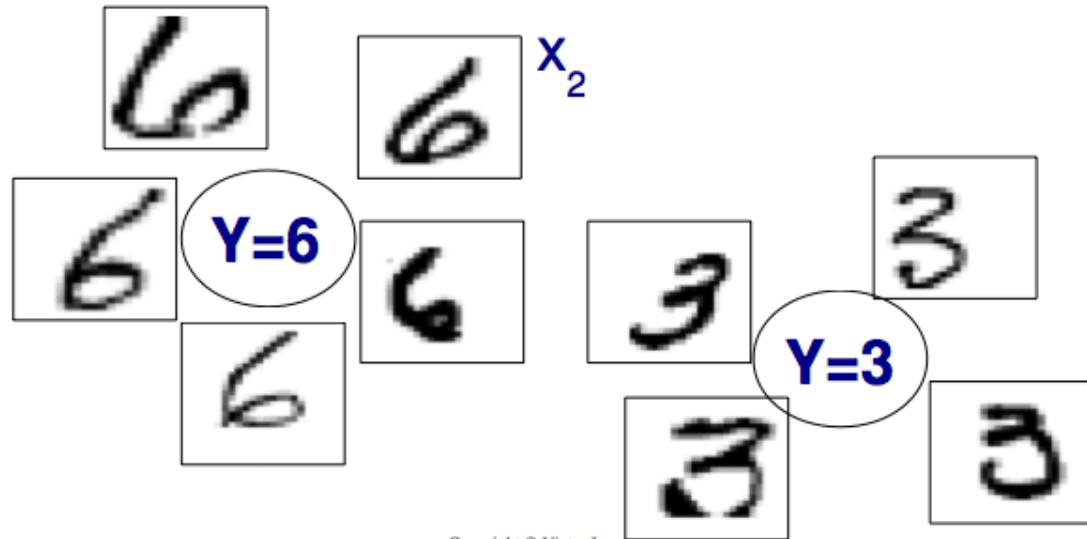
- $P(y)$: prior probability of each class
 - encodes how which classes are common, which are rare
 - apriori much more likely to have common cold than Avian flu
- $P(x|y)$: class-conditional model
 - describes how likely to see observation x for class y
 - assuming it's Avian flu, do the symptoms look plausible?
- $P(x)$: normalize probabilities across observations
 - does not affect which class is most likely (**arg max**)

Bayesian classification: normalization

$$\text{Normalizer: } P(x) = \sum_{y'} P(x|y') P(y')$$

- an “outlier” has a low probability under every class

$$P(X=x_1 | Y=3) < P(X=x_2 | Y=3)$$

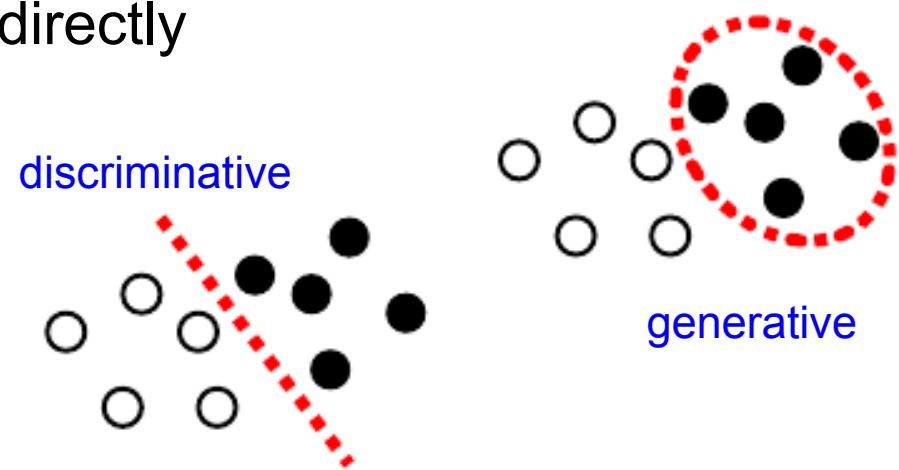


normalizer makes
 $P(Y=3|X=x_1)$
comparable
to non-outliers

Naïve Bayes: a generative model

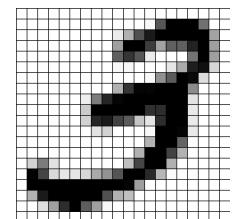
- A complete probability distribution for each class
 - defines likelihood for any point x
 - $P(\text{class})$ via $P(\text{observation})$
$$P(y|x) \propto P(x|y) P(y)$$
 - can “generate” synthetic observations
 - will share many properties of the original data
- Not all probabilistic classifiers do this
 - possible to estimate $P(y|x)$ directly
 - e.g. logistic regression:

$$P(y|x) = \frac{1}{z_y} \exp\left(\sum_i \lambda_i g_i(y, x)\right)$$



Independence assumption

- Compute $P(x_1 \dots x_n | y)$ for every observation $x_1 \dots x_n$
 - class-conditional “counts”, based on training data
 - problem: may not have seen every $x_1 \dots x_n$ for every y
 - digits: 2^{400} possible black/white patterns (20x20)
 - spam: every possible combination of words: $2^{10,000}$
 - often have observations for individual x_i for every class
- idea: assume $x_1 \dots x_n$ conditionally independent given y



$$P(x_1 \dots x_d | y) = \prod_{i=1}^d P(x_i | x_1 \dots x_{i-1}, y) = \prod_{i=1}^d P(x_i | y)$$

chain rule (exact) independence

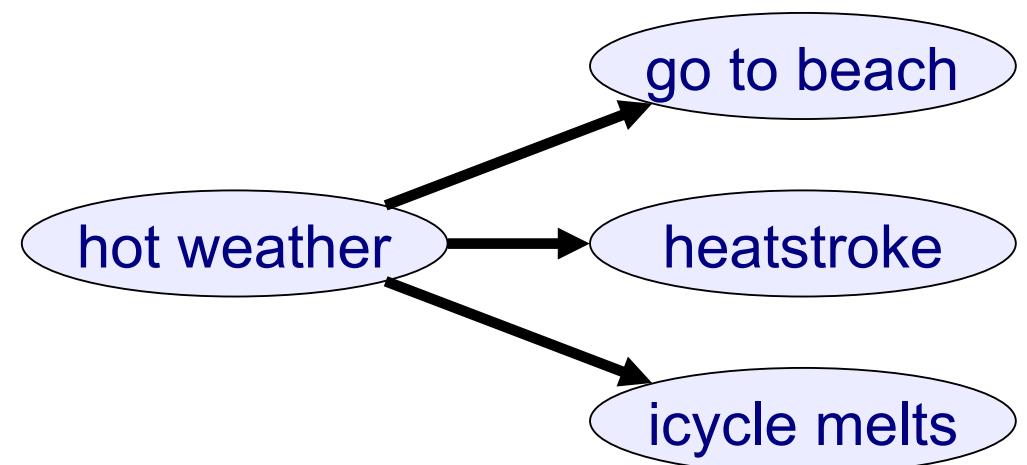
Conditional independence

- Probabilities of going to the beach and getting a heat stroke are not independent: $P(B,S) > P(B) P(S)$
- May be independent if we know the weather is hot

$$P(B,S|H) = P(B|H) P(S|H)$$

- Hot weather “explains” all the dependence between beach and heatstroke
- In classification:

- class value explains all the dependence between attributes



Continuous example

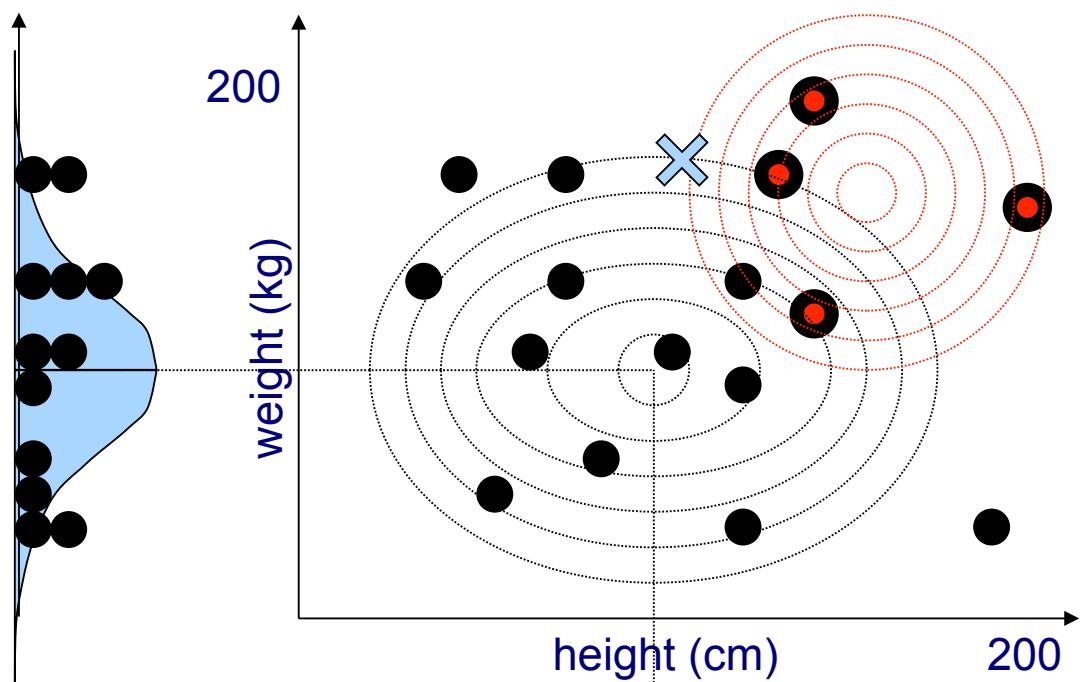
- Distinguish children from adults based on size
 - classes: {a,c}, attributes: height [cm], weight [kg]
 - training examples: $\{h_i, w_i, y_i\}$, 4 adults, 12 children
- Class probabilities: $P(a) = \frac{4}{4+12} = 0.25 ; P(c) = 0.75$
- Model for adults:
 - height ~ Gaussian with mean, variance
 - weight ~ Gaussian $(\mu_{w,a}, \sigma^2_{w,a})$
 - assume height and weight independent
- Model for children: same, using $(\mu_{h,c}, \sigma^2_{h,c}), (\mu_{w,c}, \sigma^2_{w,c})$

$$\begin{cases} \mu_{h,a} = \frac{1}{4} \sum_{i:y_i=a} h_i \\ \sigma^2_{h,a} = \frac{1}{4} \sum_{i:y_i=a} (h_i - \mu_{h,a})^2 \end{cases}$$

Continuous example

$$P(a) = \frac{4}{4+12} = 0.25 ; P(c) = 0.75$$

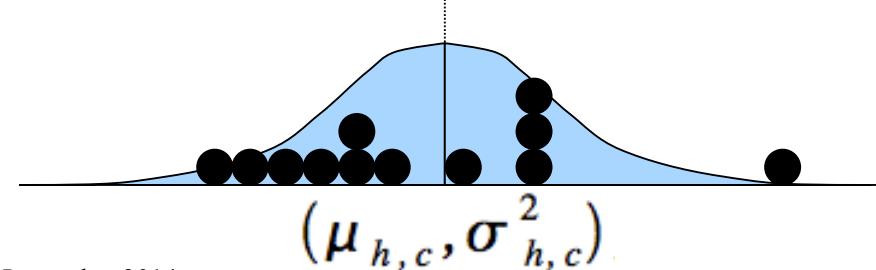
$$p(h_x|c) = \frac{1}{\sqrt{2\pi \sigma_{h,c}^2}} \exp -\frac{1}{2} \left(\frac{(h_x - \mu_{h,c})^2}{\sigma_{h,c}^2} \right)$$



$$p(w_x|c) = \frac{1}{\sqrt{2\pi \sigma_{w,c}^2}} \exp -\frac{1}{2} \left(\frac{(w_x - \mu_{w,c})^2}{\sigma_{w,c}^2} \right)$$

$$p(h_x|a) = \frac{1}{\sqrt{2\pi \sigma_{h,a}^2}} \exp -\frac{1}{2} \left(\frac{(h_x - \mu_{h,a})^2}{\sigma_{h,a}^2} \right)$$

$$p(w_x|a) = \frac{1}{\sqrt{2\pi \sigma_{w,a}^2}} \exp -\frac{1}{2} \left(\frac{(w_x - \mu_{w,a})^2}{\sigma_{w,a}^2} \right)$$

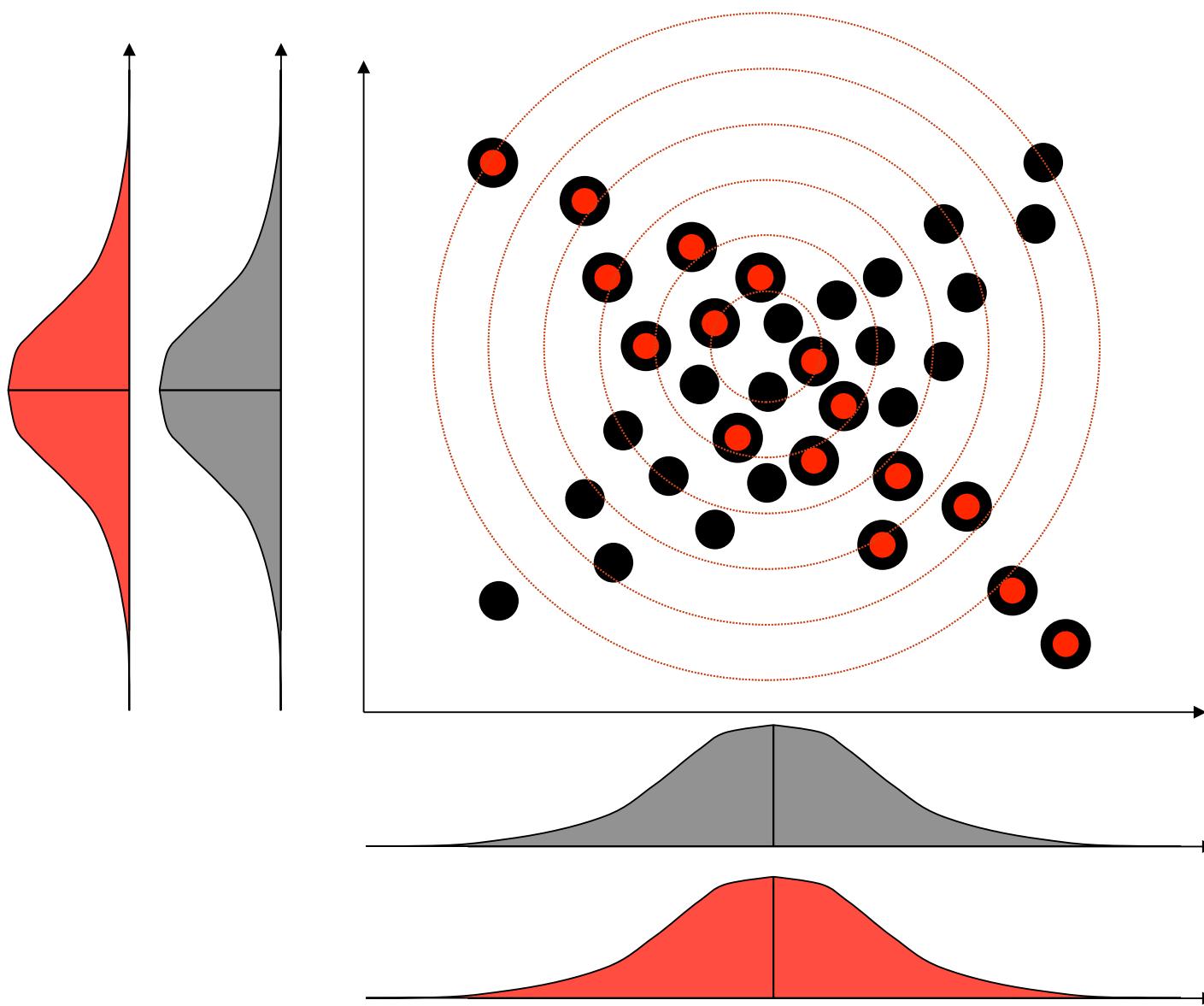


$$P(x|a) = p(h_x|a)p(w_x|a)$$

$$P(x|c) = p(h_x|c)p(w_x|c)$$

$$P(a|x) = \frac{P(x|a)P(a)}{P(x|a)P(a)+P(x|c)P(c)}$$

Problems with Naïve Bayes



Discrete example: spam

- Separate spam from valid email, attributes = words

D1: "send us your password"

spam

D2: "send us your review"

ham

D3: "review your password"

ham

D4: "review us"

spam

D5: "send your password"

spam

D6: "send us your account"

spam

new email: "review us now"

		P (spam) = 4/6	P (ham) = 2/6
spam	ham		
2/4	1/2	password	
1/4	2/2	review	
3/4	1/2	send	
3/4	1/2	us	
3/4	1/2	your	
1/4	0/2	account	

$$P(\text{review us} | \text{spam}) = P(0,1,0,1,0,0 | \text{spam}) = \left(1 - \frac{2}{4}\right)\left(\frac{1}{4}\right)\left(1 - \frac{3}{4}\right)\left(\frac{3}{4}\right)\left(1 - \frac{3}{4}\right)\left(1 - \frac{1}{4}\right)$$

$$P(\text{review us} | \text{ham}) = P(0,1,0,1,0,0 | \text{ham}) = \left(1 - \frac{1}{2}\right)\left(\frac{2}{2}\right)\left(1 - \frac{1}{2}\right)\left(\frac{1}{2}\right)\left(1 - \frac{1}{2}\right)\left(1 - \frac{0}{2}\right)$$

$$P(\text{ham} | \text{review us}) = \frac{0.0625 \times 2/6}{0.0625 \times 2/6 + 0.0044 \times 4/6} = 0.87 \text{ (note identical example)}$$

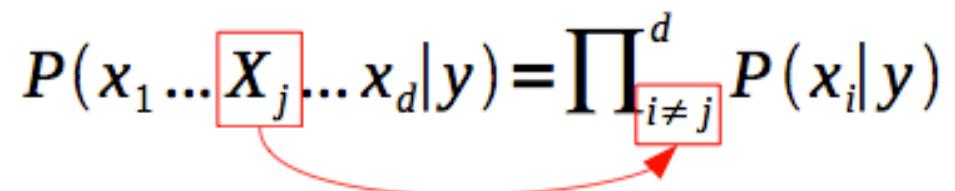
Problems with Naïve Bayes

- Zero-frequency problem
 - any mail containing “account” is spam: $P(\text{account}|\text{ham}) = 0/2$
 - solution: never allow zero probabilities
 - Laplace smoothing: add a small positive number to all counts:
 - may use global statistics in place of ϵ : $\text{num}(w) / \text{num}$
 - very common problem (Zipf's law: 50% words occur once)
- Assumes word independence
 - every word contributes independently to $P(\text{spam}|\text{email})$
 - fooling NB: add lots of “hammy” words into spam email

$$P(w|c) = \frac{\text{num}(w, c) + \epsilon}{\text{num}(c) + 2\epsilon}$$

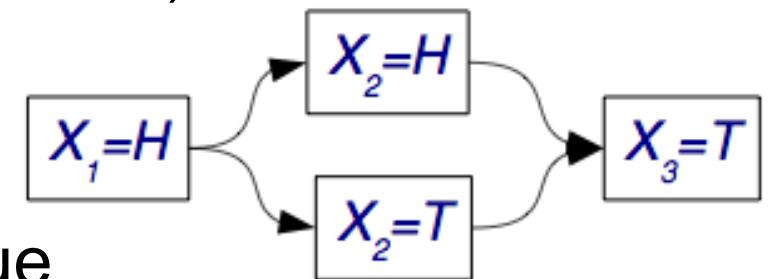
Missing data

- Suppose don't have value for some attribute X_i
 - applicant's credit history unknown
 - some medical test not performed on patient
 - how to compute $P(X_1=x_1 \dots X_j=? \dots X_d=x_d | y)$
- Easy with Naïve Bayes
 - ignore attribute in instance where its value is missing
 - compute likelihood based on observed attributes
 - no need to "fill in" or explicitly model missing values
 - based on conditional independence between attributes

$$P(x_1 \dots X_j \dots x_d | y) = \prod_{i \neq j}^d P(x_i | y)$$


Missing data (2)

- Ex: three coin tosses: Event = $\{X_1=H, X_2=?, X_3=T\}$
 - event = head, unknown (either head or tail), tail
 - event = $\{H,H,T\} + \{H,T,T\}$
 - $P(\text{event}) = P(H,H,T) + P(H,T,T)$
- General case: X_j has missing value



$$P(x_1 \dots \boxed{x_j} \dots x_d | y) = P(x_1 | y) \cdots \boxed{P(x_j | y)} \cdots P(x_d | y)$$

$$\sum_{x_j} P(x_1 \dots \boxed{x_j} \dots x_d | y) = \sum_{x_j} P(x_1 | y) \cdots \boxed{P(x_j | y)} \cdots P(x_d | y)$$

$$= P(x_1 | y) \cdots \left[\sum_{x_j} P(x_j | y) \right] \cdots P(x_d | y)$$

$$= P(x_1 | y) \cdots [1] \cdots P(x_d | y)$$

Introductory Applied Machine Learning

Decision Trees

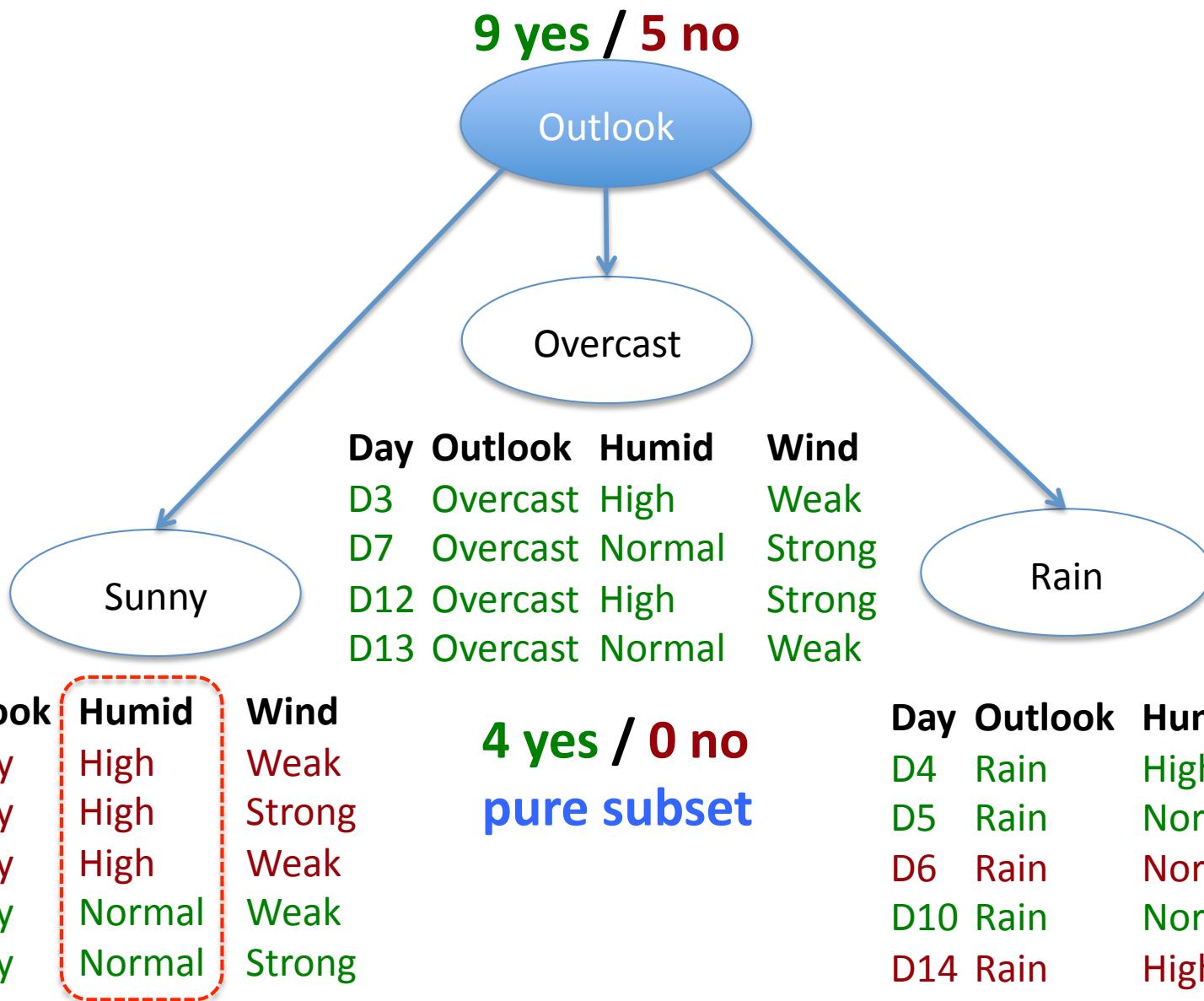
Victor Lavrenko and Nigel Goddard
School of Informatics

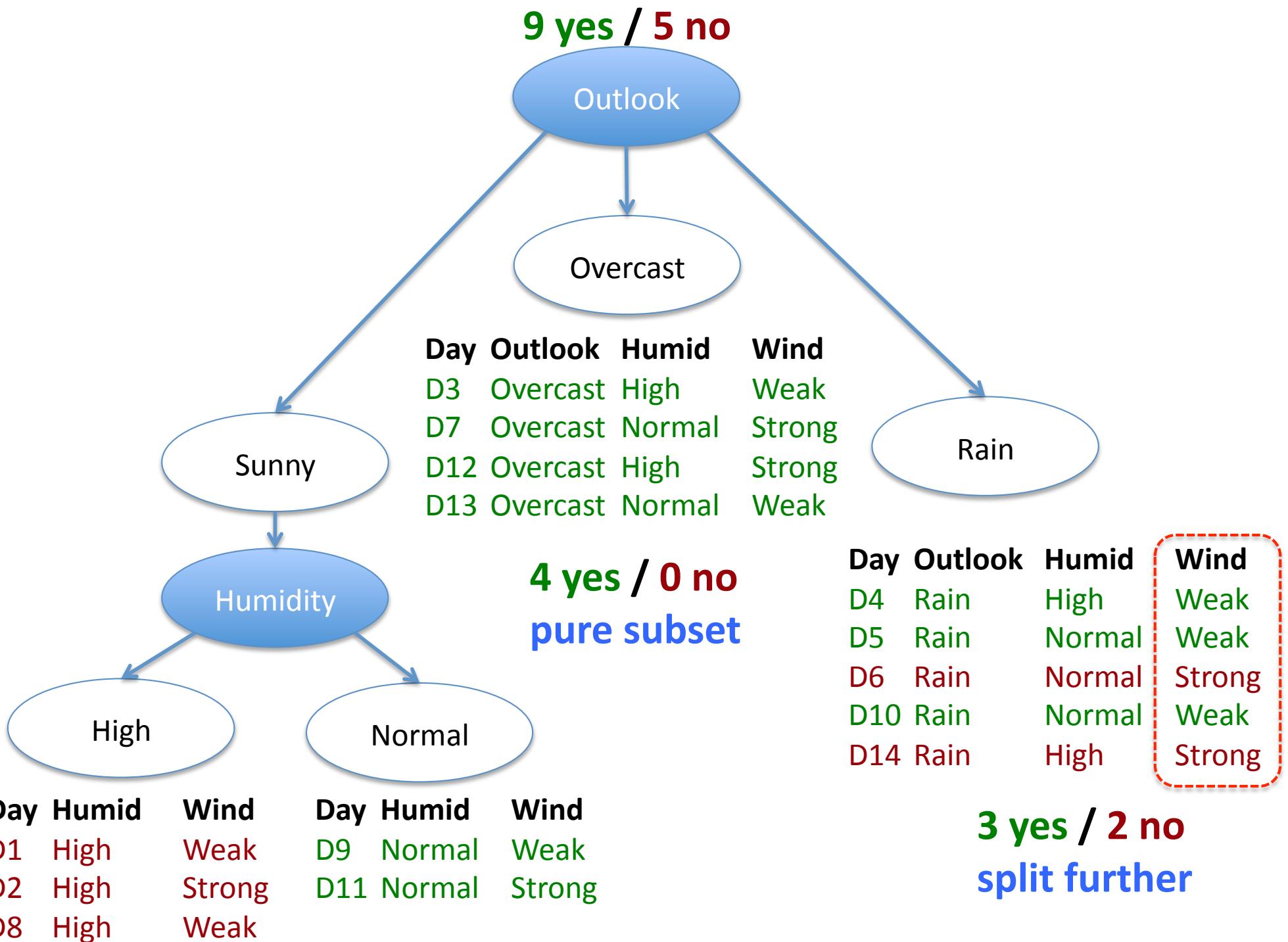
Predict if John will play tennis

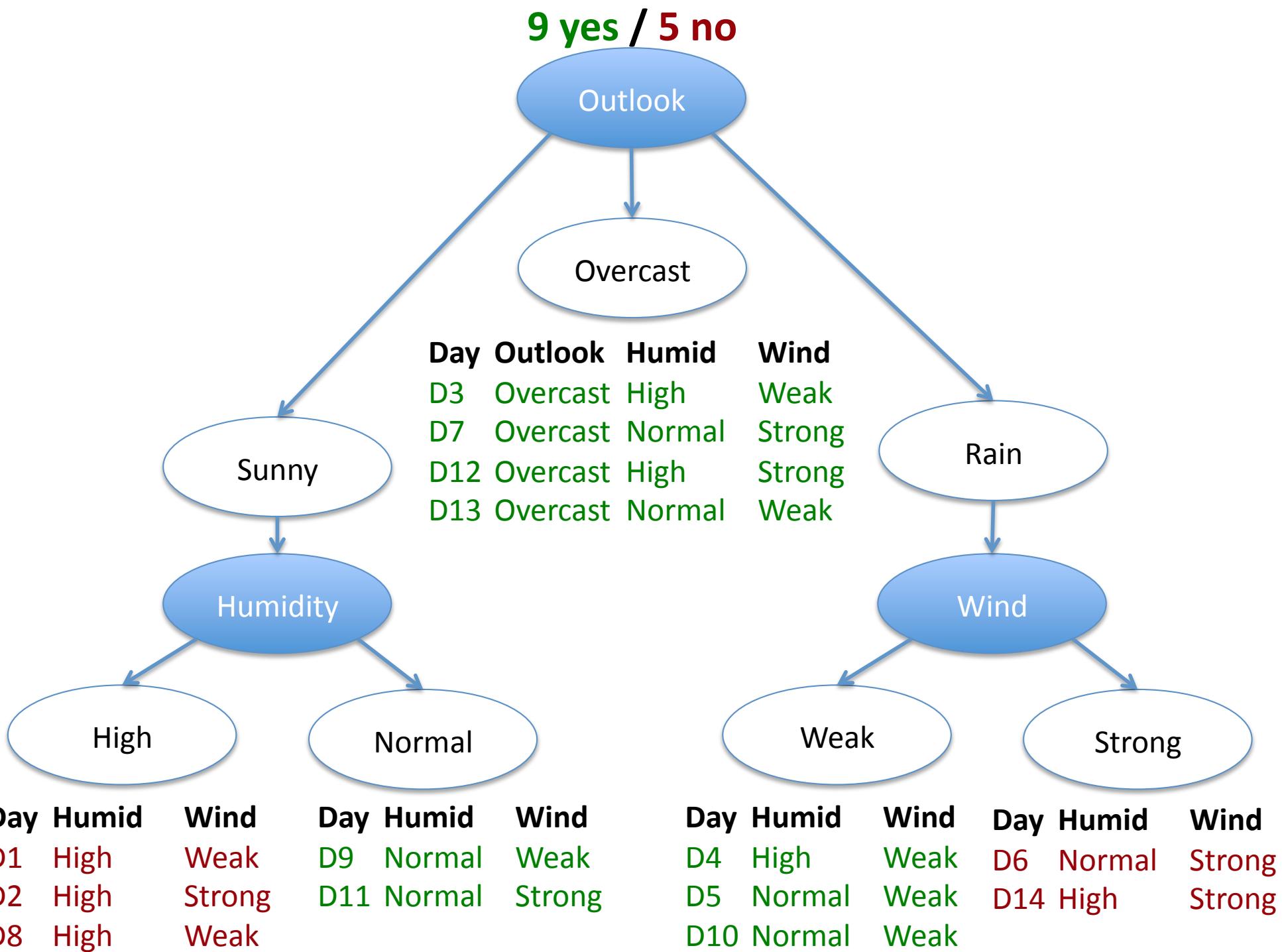
- Hard to guess
- Try to *understand* when John plays
- Divide & conquer:
 - split into subsets
 - are they pure?
(all yes or all no)
 - if yes: stop
 - if not: repeat
- See which subset new data falls into

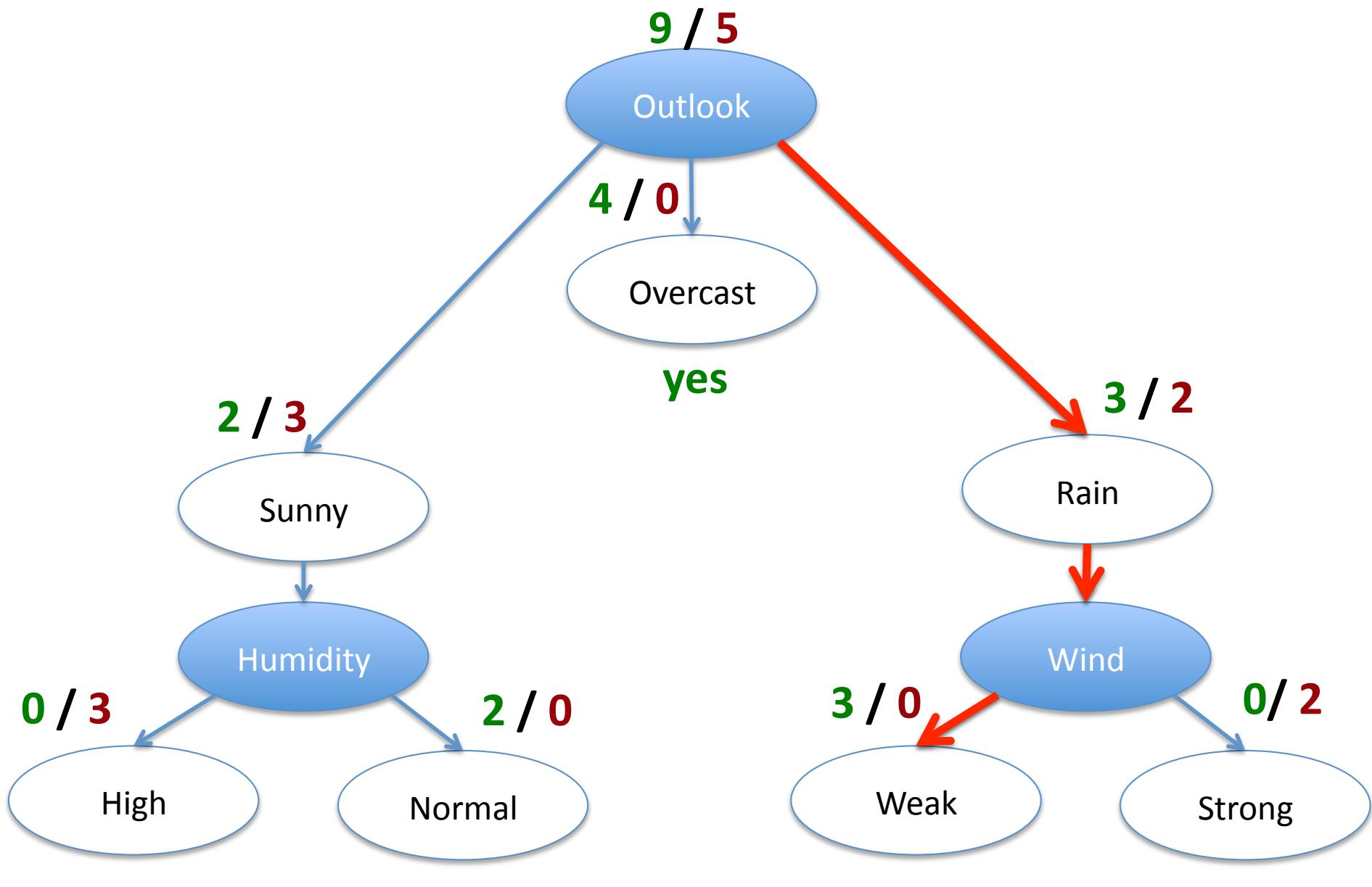
Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No
New data:				
D15	Rain	High	Weak	?









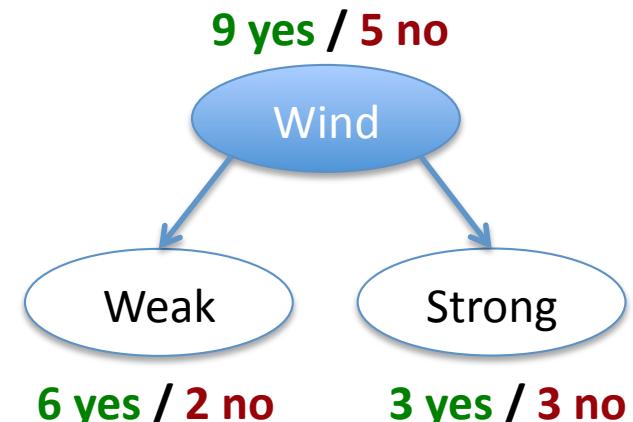
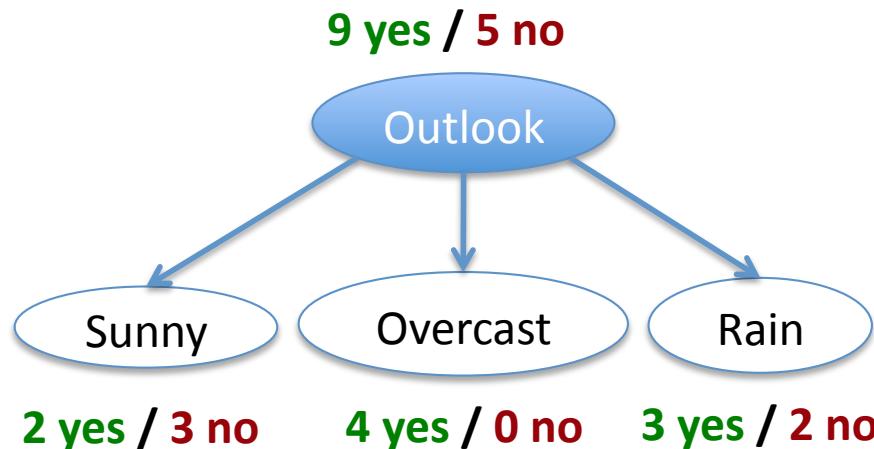
New data:

Day	Outlook	Humid	Wind	
D15	Rain	High	Weak	→ Yes

ID3 algorithm

- Split (node, {examples}):
 1. $A \leftarrow$ the best attribute for splitting the {examples}
 2. Decision attribute for this node $\leftarrow A$
 3. For each value of A , create new child node
 4. Split training {examples} to child nodes
 5. For each child node / subset:
 - if subset is pure: STOP
 - else: Split (child_node, {subset})
- Ross Quinlan (ID3: 1986), (C4.5: 1993)
- Breimanetal (CaRT: 1984) from statistics

Which attribute to split on?



- Want to measure “purity” of the split
 - more certain about Yes/No after the split
 - pure set (4 yes / 0 no) => completely certain (100%)
 - impure (3 yes / 3 no) => completely uncertain (50%)
 - can't use $P(\text{"yes"} \mid \text{set})$:
 - must be symmetric: 4 yes / 0 no as pure as 0 yes / 4 no

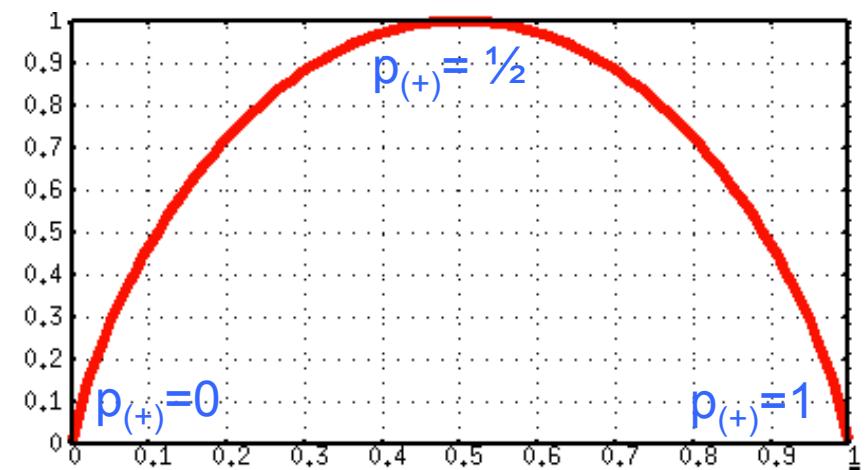
Entropy

- Entropy: $H(S) = - p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)}$ bits
 - S ... subset of training examples
 - $p_{(+)} / p_{(-}$... % of positive / negative examples in S
- Interpretation: assume item X belongs to S
 - how many bits need to tell if X positive or negative
- impure (3 yes / 3 no):

$$H(S) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \frac{3}{6} = 1 \text{ bits}$$

- pure set (4 yes / 0 no):

$$H(S) = -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0 \text{ bits}$$



Information Gain

- Want many items in pure sets
- Expected drop in entropy after split:

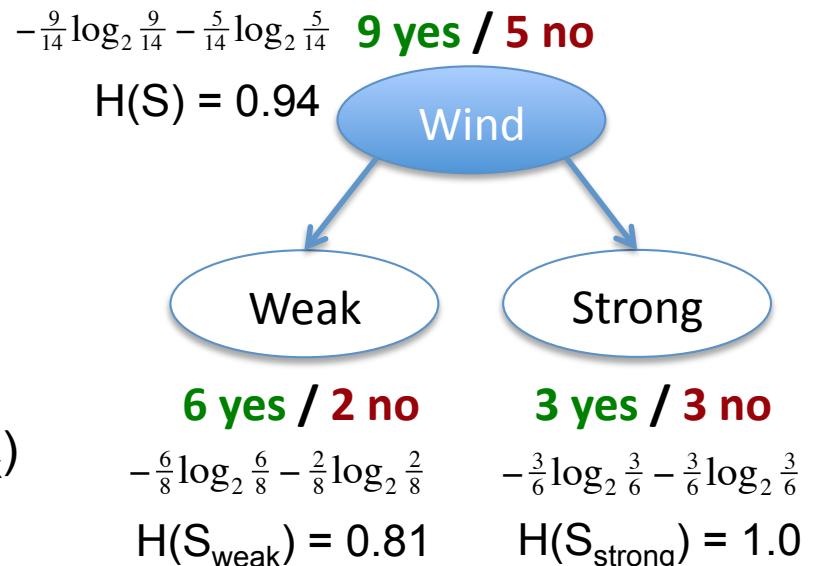
$$Gain(S, A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V)$$

V ... possible values of A
S ... set of examples {X}
S_v ... subset where X_A = V

- Mutual Information
 - between attribute A and class labels of S

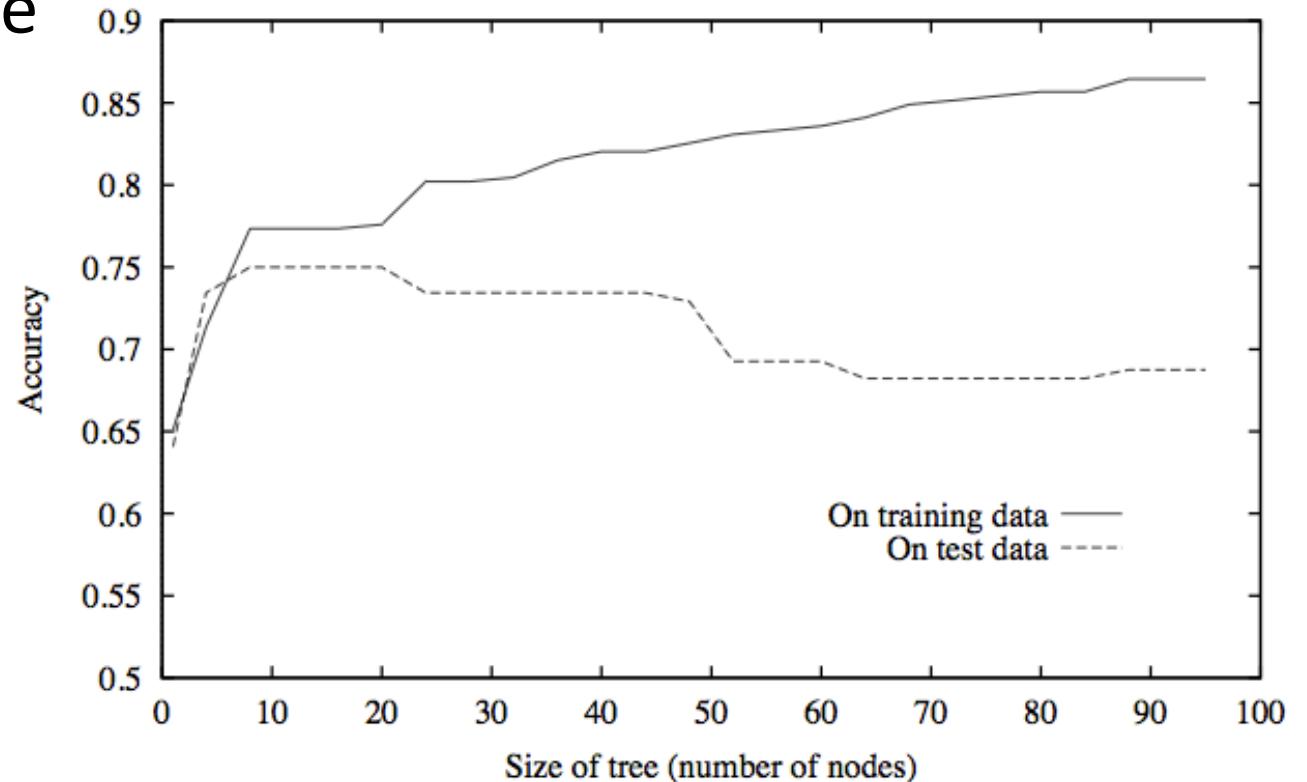
Gain (S, Wind)

$$\begin{aligned} &= H(S) - \frac{8}{14} H(S_{\text{weak}}) - \frac{6}{14} H(S_{\text{strong}}) \\ &= 0.94 - \frac{8}{14} * 0.81 - \frac{6}{14} * 1.0 \\ &= 0.049 \end{aligned}$$



Overfitting in Decision Trees

- Can always classify training examples perfectly
 - keep splitting until each node contains 1 example
 - singleton = pure
- Doesn't work on new data



Avoid overfitting

- Stop splitting when not statistically significant
- Grow, then post-prune
 - based on validation set
- Sub-tree replacement pruning (WF 6.1)
 - for each node:
 - pretend remove node + all children from the tree
 - measure performance on validation set
 - remove node that results in greatest improvement
 - repeat until further pruning is harmful

General Structure

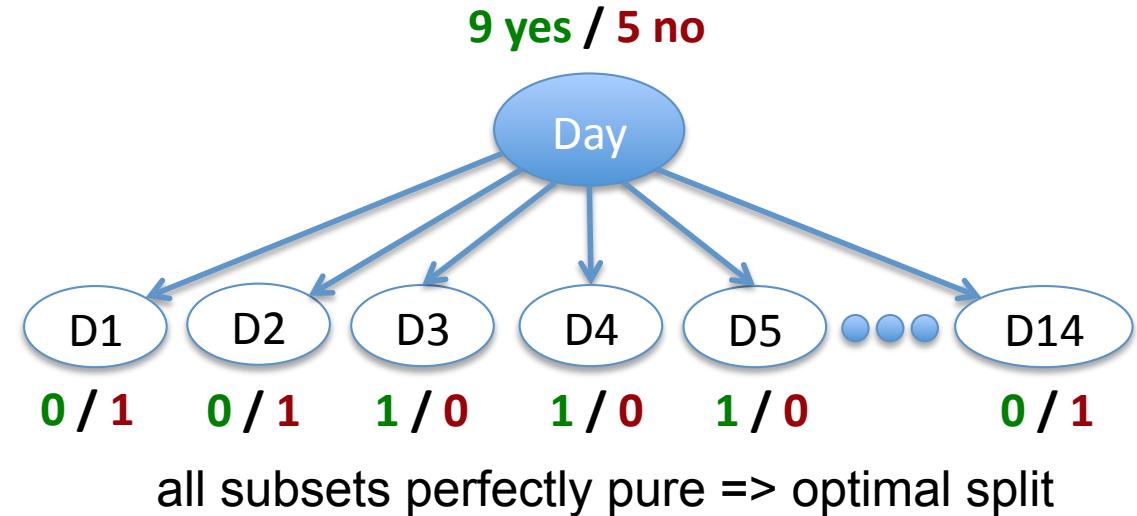
- **Task:** classification, discriminative
- **Model structure:** decision tree
- **Score function**
 - information gain at each node
 - preference for short trees
 - preference for high-gain attributes near the root
- **Optimization / search method**
 - greedy search from simple to complex
 - guided by information gain
- Book: sections 3.2, 3.3, 4.3

Problems with Information Gain

- Biased towards attributes with many values
- Won't work for new data: D15 Rain High Weak
- Use GainRatio:

$$SplitEntropy(S, A) = - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} \log \frac{|S_V|}{|S|}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitEntropy(S, A)}$$

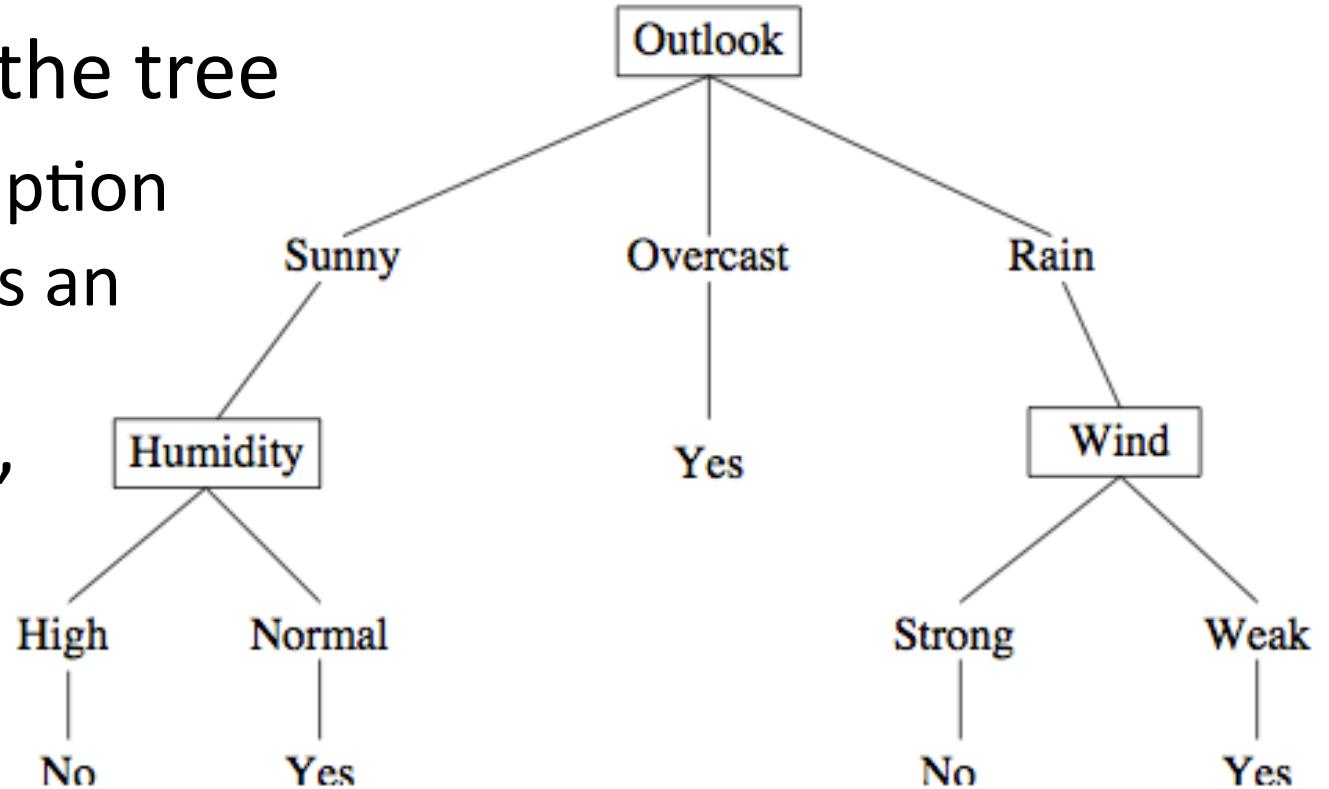


A ... candidate attribute
V ... possible values of A
S ... set of examples {X}
S_v ... subset where X_A = V

penalizes attributes with many values

Trees are interpretable

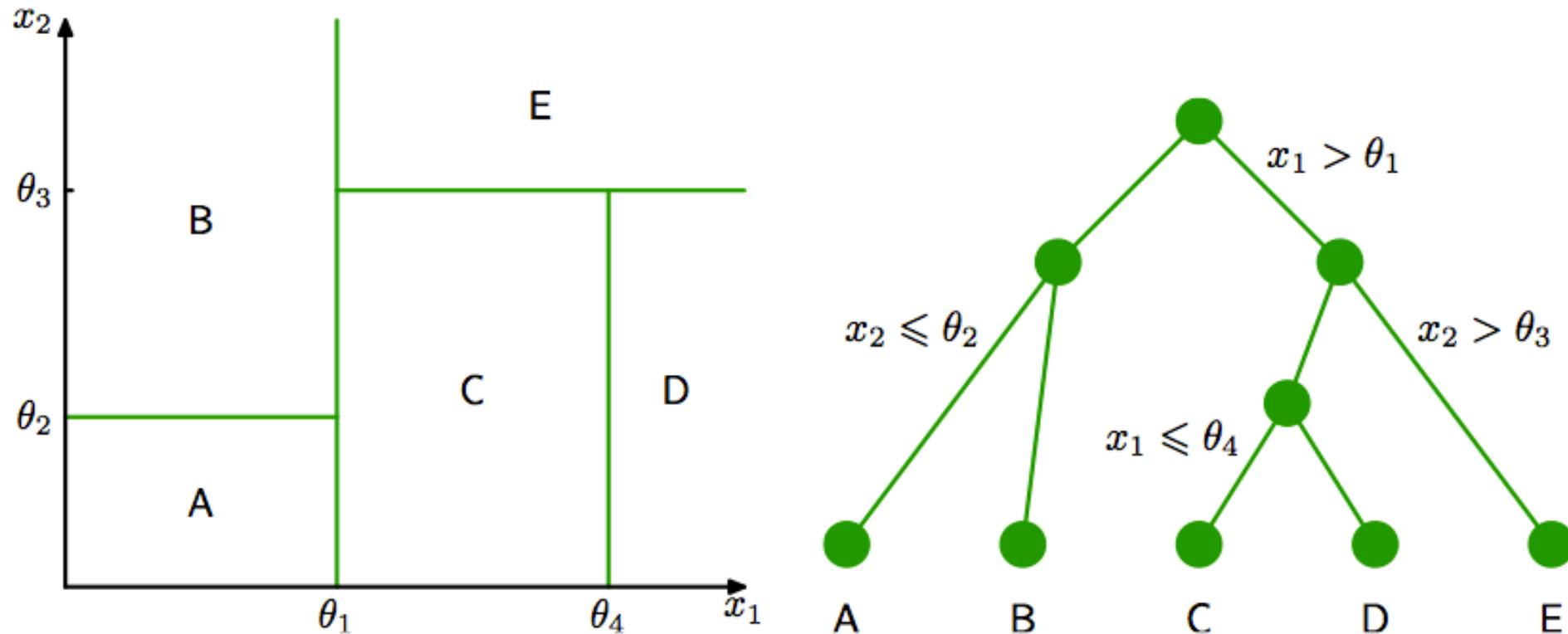
- Read rules off the tree
 - concise description of what makes an item positive
- No “black box”
 - important for users



Rule:
$$\left. \begin{array}{l} (\text{Outlook} = \text{Overcast}) \vee \\ (\text{Outlook} = \text{Rain} \wedge \text{Wind} = \text{Weak}) \vee \\ (\text{Outlook} = \text{Sunny} \wedge \text{Humidity} = \text{Normal}) \end{array} \right\}$$
 logical formula in DNF
(disjunctive normal form)

Continuous Attributes

- Dealing with continuous-valued attributes:
 - create a split: (Temperature > 72.3) = True, False
- Threshold can be optimized (WF 6.1)

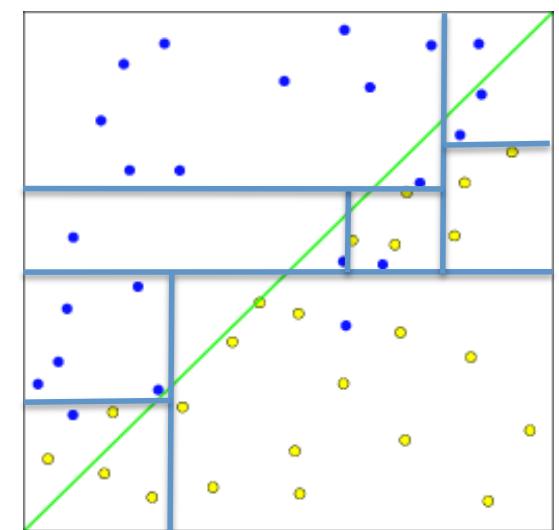


Multi-class and Regression

- Multi-class classification:
 - predict most frequent class in the subset
 - entropy: $H(S) = - \sum_c p_{(c)} \log_2 p_{(c)}$
 - $p_{(c)}$... % of examples of class c in S
- Regression:
 - predicted output = average of the training examples in the subset
 - requires a different definition of entropy
 - can use linear regression at the leaves (WF 6.5)

Pros and Cons

- Pros:
 - interpretable: humans can understand decisions
 - easily handles irrelevant attributes ($\text{Gain} = 0$)
 - can handle missing data (WF 6.1)
 - very compact: $\#\text{nodes} \ll D$ after pruning
 - very fast at testing time: $O(\text{depth})$
- Cons:
 - only axis-aligned splits of data
 - greedy (may not find best tree)
 - exponentially many possible trees



Random Decision Forest

- Grow K different decision trees:
 - pick a random subset S_r of training examples
 - grow a full ID3 tree T_r (no pruning):
 - when splitting: pick from $d \ll D$ random attributes
 - compute gain based on S_r instead of full set
 - repeat for $r = 1 \dots K$
- Given a new data point X :
 - classify X using each of the trees $T_1 \dots T_K$
 - use majority vote: class predicted most often
- State-of-the-art performance in many domains

Summary

- ID3: grows decision tree from the root down
 - greedily selects next best attribute (using Gain)
 - entropy: how uncertain we are of Yes/No in a set
 - Gain: reduction in uncertainty following a split
- Searches a complete hypothesis space
 - prefers smaller trees, high gain at the root
- Overfitting addressed by post-pruning
 - prune nodes, while accuracy \uparrow on validation set
- Fast, compact, interpretable

Introductory Applied Machine Learning

Generalization, Overfitting, Evaluation

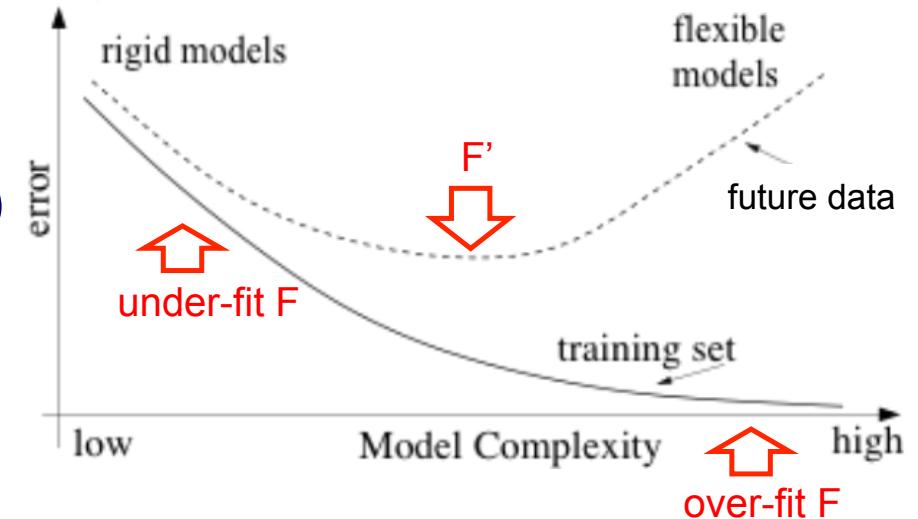
Victor Lavrenko and Nigel Goddard
School of Informatics

Generalization

- Training data: $\{x_i, y_i\}$
 - examples that we used to train our predictor
 - e.g. all emails that our users labelled ham / spam
- Future data: $\{x_i, ?\}$
 - examples that our classifier has never seen before
 - e.g. emails that will arrive tomorrow
- Want to do well on future data, not training
 - not very useful: we already know y_i
 - easy to be perfect on training data (DT, kNN, kernels)
 - does not mean you will do well on future data
 - can over-fit to idiosyncrasies of our training data

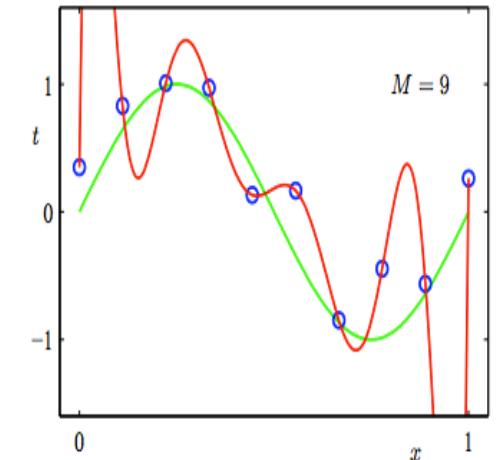
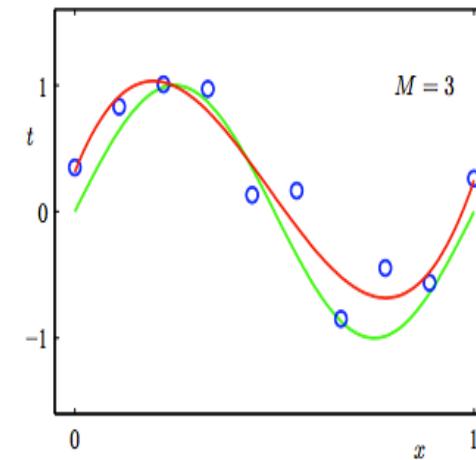
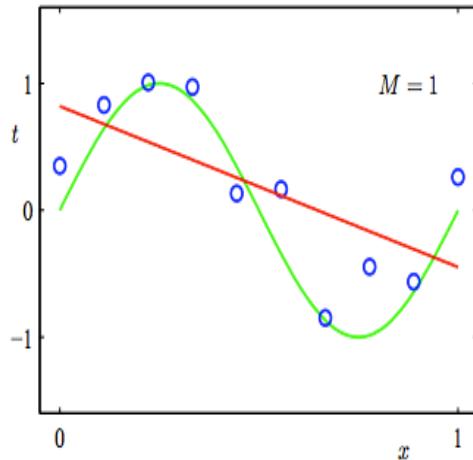
Under- and Over-fitting

- Over-fitting:
 - predictor too complex (flexible)
 - fits “noise” in the training data
 - patterns that will not re-appear
 - predictor F over-fits the data if:
 - we can find another predictor F'
 - which makes more mistakes on training data: $E_{train}(F') > E_{train}(F)$
 - but fewer mistakes on unseen future data : $E_{gen}(F') < E_{gen}(F)$
- Under-fitting:
 - predictor too simplistic (too rigid)
 - not powerful enough to capture salient patterns in data
 - can find another predictor F' with smaller E_{train} and E_{gen}



Under- and Over-fitting examples

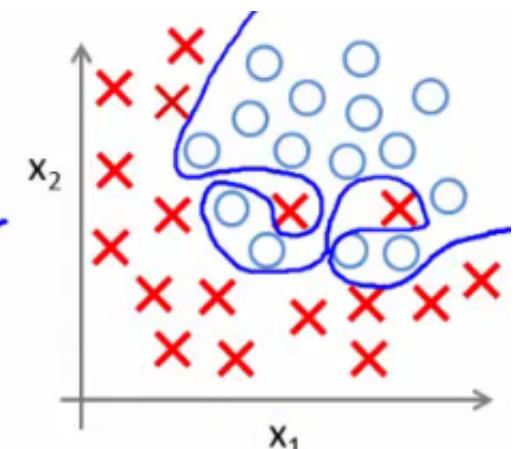
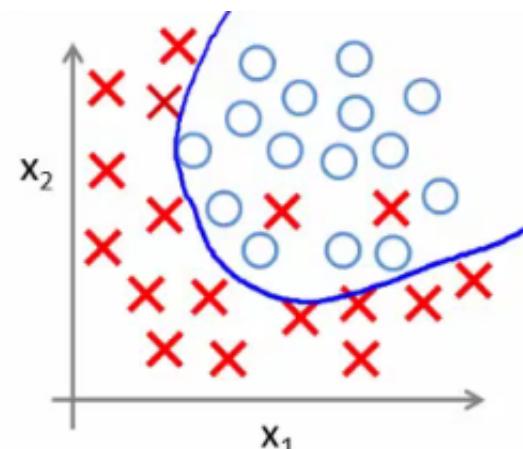
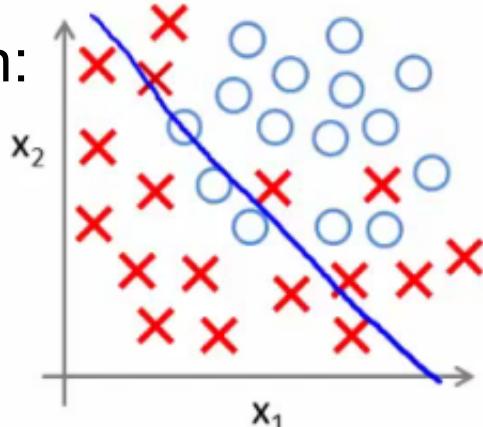
Regression:



predictor too inflexible:
cannot capture pattern

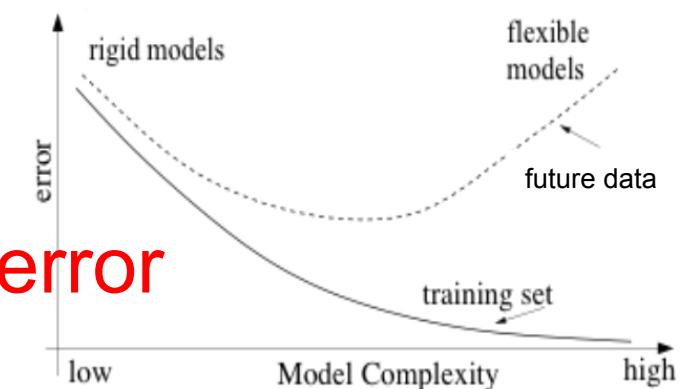
predictor too flexible:
fits noise in the data

Classification:



Flexible vs. inflexible predictors

- Each dataset needs different level of “flexibility”
 - depends on task complexity + available data
 - want a “knob” to get rigid / flexible predictors
- Most learning algorithms have such knobs:
 - regression: order of the polynomial
 - NB: number of attributes, limits on σ^2 , ε
 - DT: #nodes in the tree / pruning confidence
 - kNN: number of nearest neighbors
 - SVM: kernel type, cost parameter
- Tune to minimize **generalization error**



Training vs. Generalization Error

- Training error:
- Generalization error:

- how well we will do on future data
- don't know what future data x_i will be
- don't know what labels y_i it will have
- but know the “range” of all possible $\{x, y\}$
 - x : all possible 20x20 black/white bitmaps
 - y : $\{0, 1, \dots, 9\}$ (digits)

Can never compute generalisation error

$$E_{train} = \frac{1}{n} \sum_{i=1}^n \underbrace{\text{error}(f_D(\mathbf{x}_i), y_i)}_{\substack{\text{value we} \\ \text{predicted}}} \quad \begin{array}{l} \text{same? different by how much?} \\ \text{true value} \end{array}$$

$$E_{gen} = \int \underbrace{\text{error}(f_D(\mathbf{x}), y)}_{\substack{\text{error as before}}} p(y, \mathbf{x}) d\mathbf{x} \quad \begin{array}{l} \text{Usually} \\ E_{train} \leq E_{gen} \\ \text{how often we expect} \\ \text{to see such } \mathbf{x} \text{ and } y \end{array}$$

Estimating Generalization Error

- Testing error:

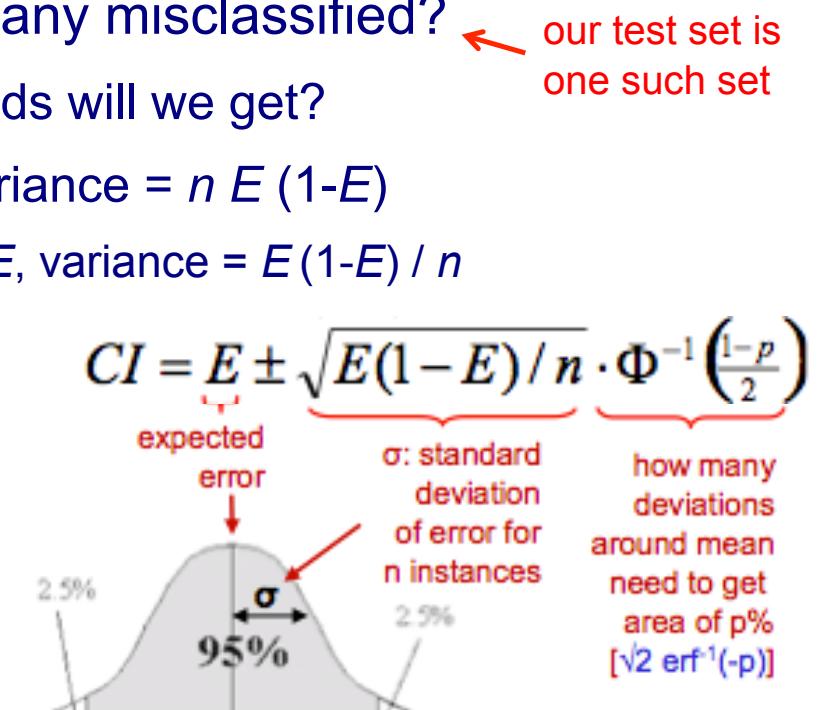
$$E_{test} = \frac{1}{n} \sum_{i=1}^n \text{error}(f_D(\mathbf{x}_i), y_i)$$

over testing set

- set aside part of training data (testing set)
- learn a predictor without using any of this data
- predict values for testing set, compute error
- gives an **estimate** of true generalization error
 - if testing set is **unbiased** sample from $p(x,y)$: $\lim_{n \rightarrow \infty} E_{test} = E_{gen}$
 - how close? depends on n
- Ex: binary classification, 100 instances
 - assume: 75 classified correctly, 25 incorrectly
 - $E_{test} = 0.25$, E_{gen} around 0.25, but how close?

Confidence Interval for Future Error

- What range of errors can we expect for **future** test sets?
 - $E_{test} \pm \Delta E$ such that 95% of future test sets fall within that interval
- E_{test} is an unbiased estimate of $E = \text{true error rate}$
 - E = probability our system will misclassify a random instance
 - take a random set of n instances, how many misclassified?
 - flip E -biased coin n times, how many heads will we get?
 - Binomial distribution with mean = $n E$, variance = $n E (1-E)$
 - $E_{future} = \#\text{misclassified} / n$, \sim Gaussian, mean E , variance = $E (1-E) / n$
 - 2/3 future test sets will have error in $E \pm \sqrt{E(1-E)/n}$
 - p% confidence interval for future error:
 - for $n=100$ examples, $p=0.95$ and $E = 0.25$
 - $\sigma = \sqrt{(0.25 \cdot 0.75/100)} = .043$
 - $CI = 0.25 \pm 1.96 \cdot \sigma = \mathbf{0.25 \pm 0.08}$
 - $n=100$, $p=0.99 \rightarrow CI = \mathbf{0.25 \pm 0.11}$
 - $n=10000$, $p=0.95 \rightarrow CI = \mathbf{0.25 \pm 0.008}$

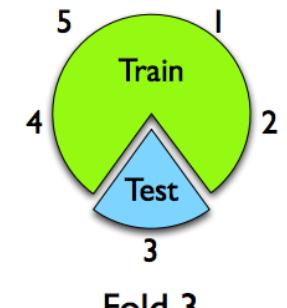
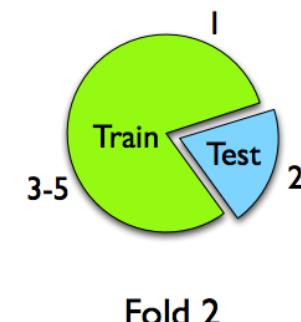
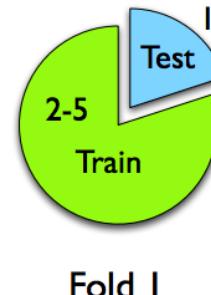


Training, Validation, Testing sets

- Training set: construct classifier
 - NB: count frequencies, DT: pick attributes to split on
- Validation set: pick algorithm + knob settings
 - pick best-performing algorithm (NB vs. DT vs. ...)
 - fine-tune knobs (tree depth, k in kNN, c in SVM ...)
- Testing set: estimate future error rate
 - never report best of many runs
 - run only once, or report results of every run
- Split **randomly** to avoid bias

Cross-validation

- Conflicting priorities when splitting the dataset
 - estimate future error as accurately as possible
 - large testing set: big n_{test} → tight confidence interval
 - learn classifier as accurately as possible
 - large training set: big n_{train} → better estimates
 - training and testing cannot overlap: $n_{\text{train}} + n_{\text{test}} = \text{const}$
- Idea: evaluate Train → Test, then Test → Train, average results
 - **every** point is both training and testing, never at the same time
 - reduces chances of getting an unusual (biased) testing set
 - 5-fold cross-validation
 - randomly split the data into 5 sets
 - test on each in turn (train on 4 others)
 - average the results over 5 folds
 - more common: 10-fold



Leave-one-out

- n-fold cross-validation ($n = \text{total number of instances}$)
 - predict each instance, training on all $(n-1)$ other instances
- Pros and cons:
 - best possible classifier learned: $n-1$ training examples
 - high computational cost: re-learn everything n times
 - not an issue for instance-based methods like kNN
 - there are tricks to make such learning faster
 - classes not balanced in training / testing sets
 - random data, 2 equi-probable classes \rightarrow wrong 100% of the time
 - testing balance: {1 of A, 0 of B} vs. training: { $n/2$ of B, $n/2-1$ of A}
 - duplicated data \rightarrow nothing can beat 1NN (0% error)
 - wouldn't happen with 10-fold cross-validation

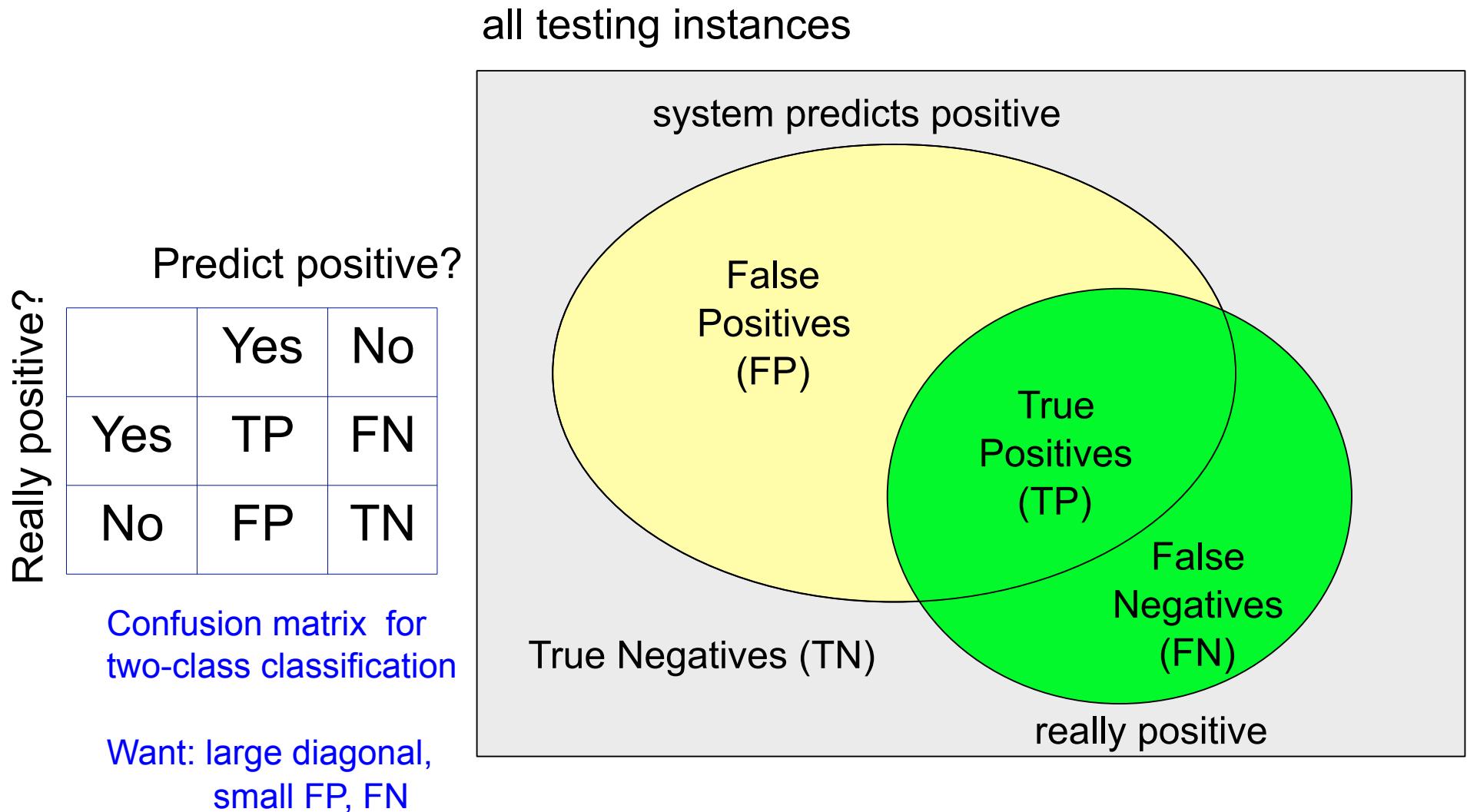
Stratification

- Problems with leave-one-out:
 - training / testing sets have classes in different proportions
 - not limited to leave-one-out:
 - K-fold cross-validation: random splits → imbalance
- Stratification
 - keep class labels balanced across training / testing sets
 - simple way to guard against unlucky splits
 - recipe:
 - randomly split each class into K parts
 - assemble i^{th} part from all classes to make the i^{th} fold

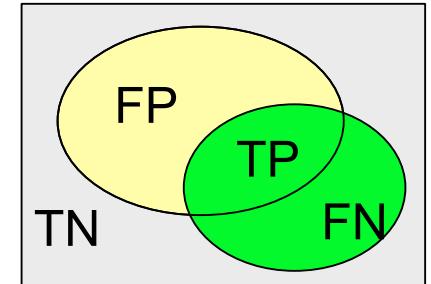
Evaluation measures

- Are we doing well? Is system A better than B?
- A measure of how (in)accurate a system is on a task
 - in many cases Error (Accuracy / PC) is not the best measure
 - using the appropriate measure will help select best algorithm
- Classification
 - how often we classify something right / wrong
- Regression
 - how close are we to what we're trying to predict
- Unsupervised
 - how well do we describe our data
 - in general – really hard

Classification measures: basics



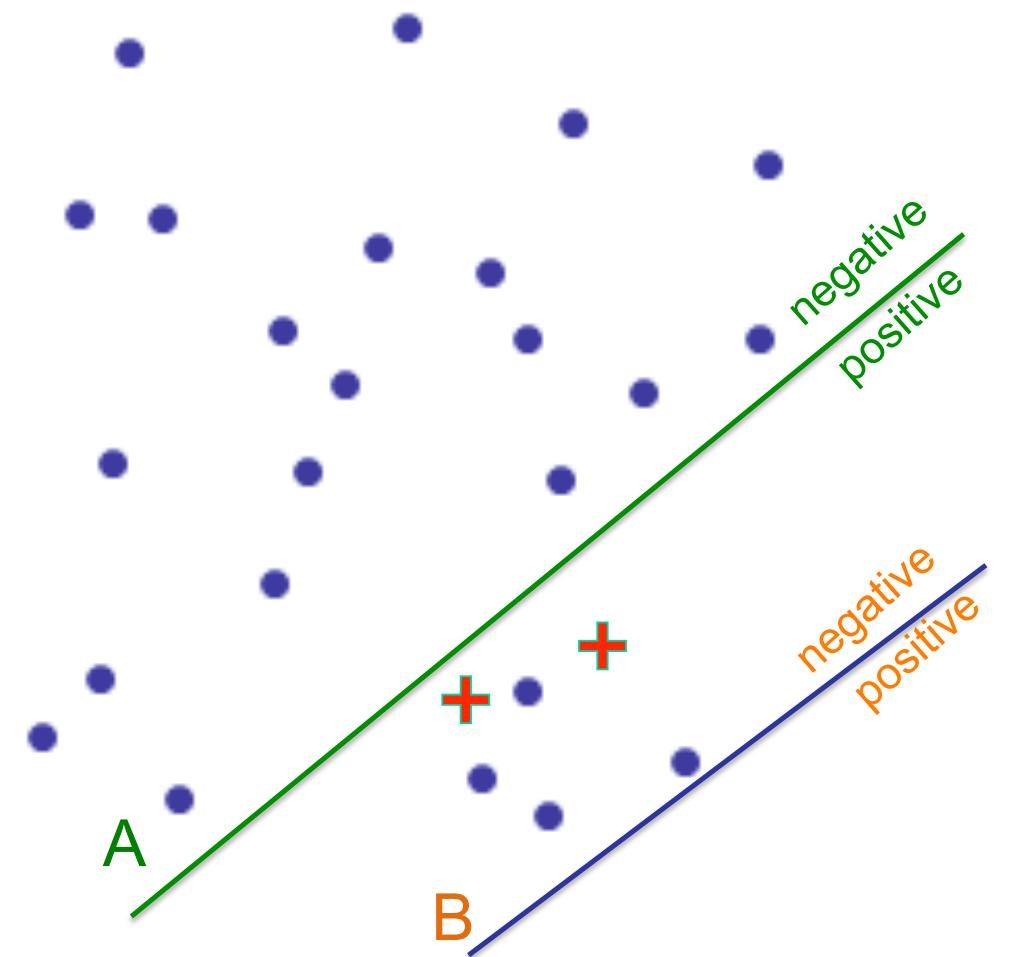
Classification Error



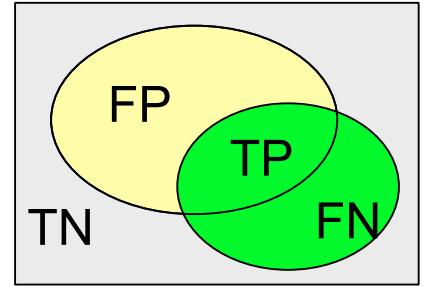
- Classification error = $\frac{\text{errors}}{\text{total}} = \frac{FP + FN}{TP + TN + FP + FN}$
- Accuracy = $(1 - \text{error}) = \frac{\text{correct}}{\text{total}} = \frac{TP + TN}{TP + TN + FP + FN}$
- Basic measure of “goodness” of a classifier
- Problem: cannot handle unbalanced classes
 - ex1: predict whether an earthquake is about to happen
 - happen very rarely, very good accuracy if always predict “No”
 - solution: make FNs much more “costly” than FPs
 - ex2: web search: decide if a webpage is relevant to user
 - 99.9999% of pages not relevant to any query → retrieve nothing
 - solution: use measures that don’t involve TN (recall / precision)

Accuracy and un-balanced classes

- You're predicting Nobel prize (+) vs. not (•)
- Human would prefer classifier A.
- Accuracy will prefer classifier B (fewer errors)
- Accuracy poor metric here



Misses and False Alarms



- False Alarm rate = False Positive rate = $FP / (FP+TN)$
 - % of negatives we misclassified as positive
- Miss rate = False Negative rate = $FN / (TP+FN)$
 - % of positives we misclassified as negative
- Recall = True Positive rate = $TP / (TP+FN)$
 - % of positives we classified correctly ($1 - \text{Miss rate}$)
- Precision = $TP / (TP + FP)$
 - % positive out of what we predicted was positive
- Meaningless to report just one of these
 - trivial to get 100% recall or 0% false alarm
 - typical: recall/precision or Miss / FA rate or TP/FP rate

Evaluation (recap)

- Predicting class C (e.g. spam)
 - classifier can make two types of mistakes:
 - FP: false positives – non-spam emails mistakenly classified as spam
 - FN: false negatives – spam emails mistakenly classified as non-spam
 - TP/TN: true positives/negatives – correctly classified spam/non-spam
 - common error/accuracy measures:
 - Classification Error: $\frac{errors}{total} = \frac{FP+FN}{TP+TN+FP+FN}$
 - Accuracy = 1-Error: $\frac{correct}{total} = \frac{TP+TN}{TP+TN+FP+FN}$
 - False Alarm = False Positive rate = $FP / (FP+TN)$
 - Miss = False Negative rate = $FN / (TP+FN)$
 - Recall = True Positive rate = $TP / (TP+FN)$
 - Precision = $TP / (TP+FP)$

		Predicted C?	
		Yes	No
Really C?	Yes	TP	FN
	No	FP	TN

Utility and Cost

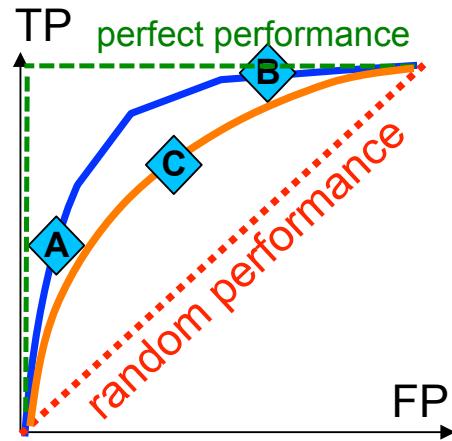
- Sometimes need a single-number evaluation measure
 - optimizing the learner (automatically), competitive evaluation
 - may know costs of different errors, e.g. earthquakes:
 - false positive: cost of preventive measures (evacuation, lost profit)
 - false negative: cost of recovery (reconstruction, liability)
- Detection cost: weighted average of FP, FN rates
 - $\text{Cost} = C_{FP} * FP + C_{FN} * FN$ [event detection]
- F-measure: harmonic mean of recall, precision
 - $F1 = 2 / (1 / \text{Recall} + 1 / \text{Precision})$ [Information Retrieval]
- Domain-specific measures:
 - e.g. observed profit/loss from +/- market prediction

Thresholds in Classification

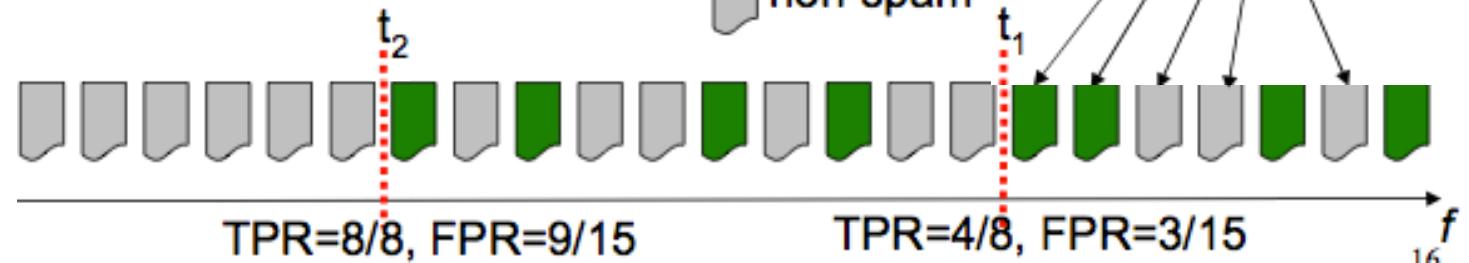
- Two systems have the following performance:
 - A: True Positive = 50%, False Positive = 20%
 - B: True Positive = 100%, False Positive = 60%
- Which is better? (assume no-apriori utility)
 - very misleading question
 - A and B could be the same exact system
 - operating at different thresholds

ROC curves

- Many algorithms compute “confidence” $f(x)$
 - threshold to get decision: spam if $f(x) > t$, non-spam if $f(x) \leq t$
 - Naïve Bayes: $P(\text{spam}|x) > 0.5$, Linear/Logistic/SVM: $w^T x > 0$, Decision Tree: $p_+/p_- > 1$
 - threshold t determines error rates
 - False Positive rate = $P(f(x) > t | \text{ham})$, True Positive rate = $P(f(x) > t | \text{spam})$
- Receiver Operating Characteristic (ROC):
 - plot TPR vs. FPR as t varies from ∞ to $-\infty$



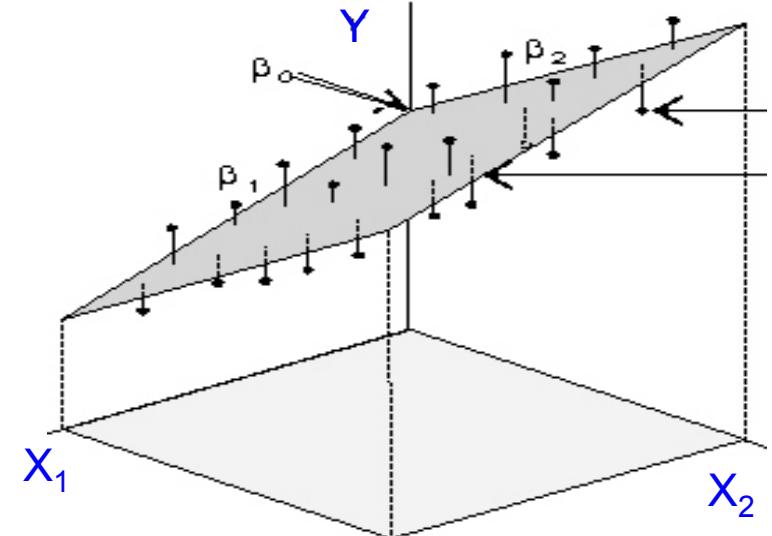
shows performance of system across all possible thresholds
AUC = area under ROC curve popular alternative to Accuracy



		Predicted spam?	
		Yes	No
Really spam?	Yes	TP	FN
No	FP	TN	

Evaluating regression

- Classification:
 - count how often we are wrong
- Regression:
 - predict numbers y_i from inputs x_i
 - always wrong, but by how much?
 - distance between predicted & true values
 - (root) mean squared error:
 - popular, well-understood, nicely differentiable
 - sensitive to single large errors (outliers)
 - mean absolute error:
 - less sensitive to outliers
 - correlation coefficient
 - insensitive to mean & scale



$$\sqrt{\frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2}$$

testing set

predicted true

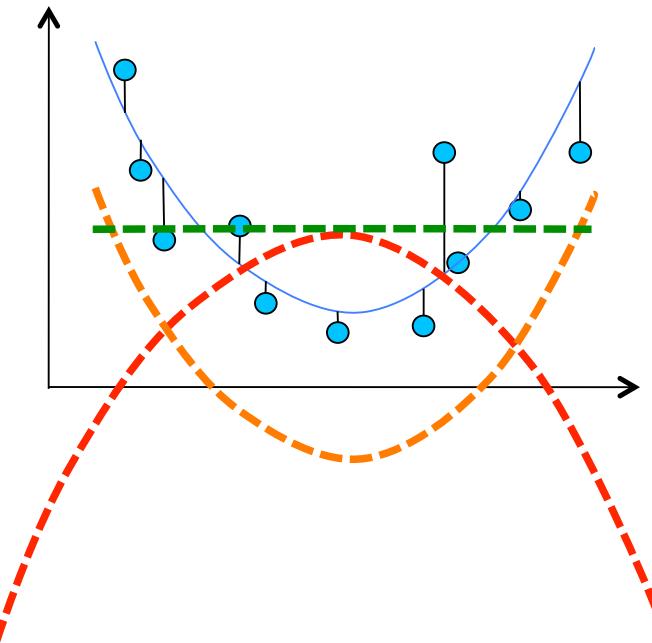
$$\frac{1}{n} \sum_{i=1}^n |f(x_i) - y_i|$$

$$\frac{n \sum_{i=1}^n (f(x_i) - \mu_f)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^n (f(x_i) - \mu_f)^2 \cdot \sum_{i=1}^n (y_i - \mu_y)^2}}$$

Mean Squared Error

- Average (squared) deviation from truth
- Very sensitive to outliers
 - 99 exact, 1 off by \$10
 - all 100 wrong by \$1
- Sensitive to mean / scale
 - $\mu_y = \frac{1}{n} \sum_i y_i$... good baseline
- Relative squared error (Weka)

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2}$$



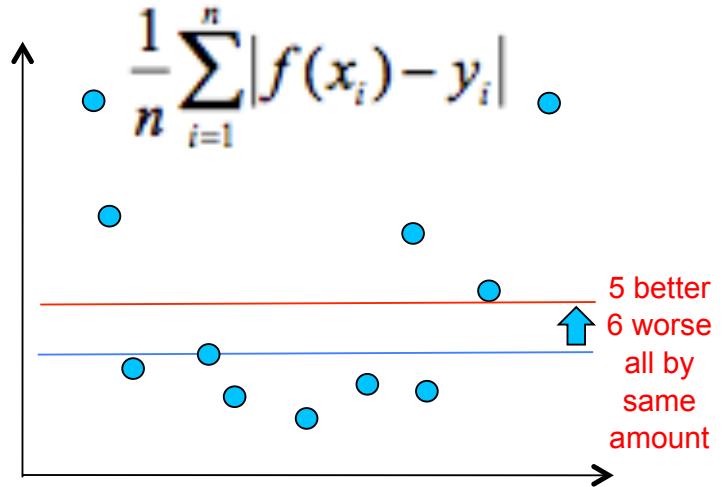
$$\sqrt{\frac{\sum_{i=1}^n (f(x_i) - y_i)^2}{\sum_{i=1}^n (\mu_y - y_i)^2}}$$

MSE of predictor

MSE when using the mean as a predictor

Mean Absolute Error

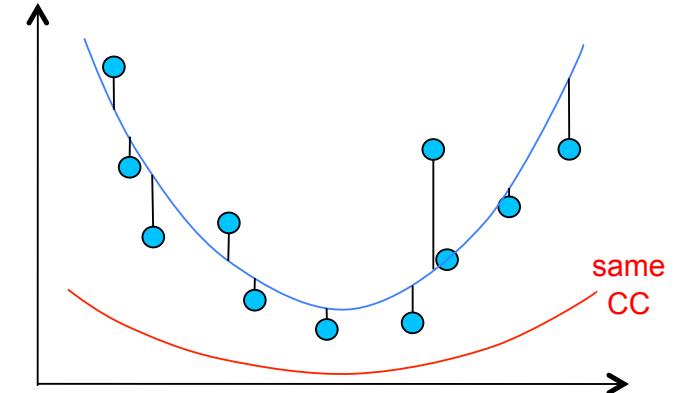
- Mean Absolute Error (MAE):
 - less sensitive to outliers
 - many small errors = one large error
 - best 0th order baseline: $\text{median}\{y_i\}$
 - not the mean as for MSE
- Median Absolute Deviation (MAD): $\text{med}\{|f(x_i)-y_i|\}$
 - robust, completely ignores outliers
 - can define similar squared error: $\text{median}\{(f(x_i)-y_i)^2\}$
 - difficult to work with (can't take derivatives)
- Sensitive to mean, scale



Correlation Coefficient

- Completely insensitive to mean / scale:

$$\frac{n \sum_{i=1}^n (f(x_i) - \mu_f)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^n (f(x_i) - \mu_f)^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \mu_y)^2}} = n \sum_{i=1}^n \underbrace{\frac{f(x_i) - \mu_f}{\sigma_f}}_{\text{prediction relative to mean}} \cdot \underbrace{\frac{y_i - \mu_y}{\sigma_y}}_{\text{truth relative to mean}}$$

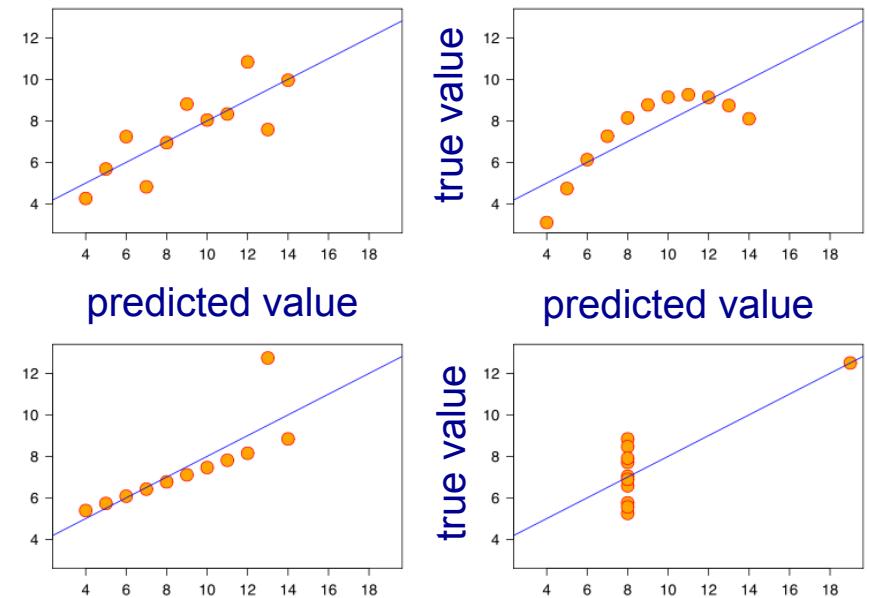


- Intuition: did you capture the relative ordering?

- output larger $f(x_i)$ for larger y_i
- output smaller $f(x_i)$ for smaller y_i
- useful for ranking tasks:

- e.g. recommend a movie to a user

- Important to visualize data
 - same CC for 4 predictors →



Summary

- Training vs. generalization error
 - under-fitting and over-fitting
- Estimate how well your system is doing its job
 - how does it compare to other approaches?
 - what will be the error rate on future data?
- Training and testing
 - cross-validation, leave-one-out, stratification, significance
- Evaluation measures
 - accuracy, miss / false alarm rates, detection cost
 - ROC curves
 - regression: (root) mean squared/absolute error, correlation

IAML: Linear Regression

Nigel Goddard
School of Informatics

Semester 1

Overview

- ▶ The linear model
- ▶ Fitting the linear model to data
- ▶ Probabilistic interpretation of the error function
- ▶ Examples of regression problems
- ▶ Dealing with multiple outputs
- ▶ Generalized linear regression
- ▶ Radial basis function (RBF) models

The Regression Problem

- ▶ Classification and regression problems:
 - ▶ Classification: target of prediction is discrete
 - ▶ Regression: target of prediction is continuous
- ▶ Training data: Set \mathcal{D} of pairs (\mathbf{x}_i, y_i) for $i = 1, \dots, n$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$
- ▶ Today: Linear regression, i.e., relationship between \mathbf{x} and y is linear.
- ▶ Although this is simple (and limited) it is:
 - ▶ More powerful than you would expect
 - ▶ The basis for more complex nonlinear methods
 - ▶ Teaches a lot about regression and classification

Examples of regression problems

- ▶ Robot inverse dynamics: predicting what torques are needed to drive a robot arm along a given trajectory
- ▶ Electricity load forecasting, generate hourly forecasts two days in advance.
- ▶ Predicting staffing requirements at help desks based on historical data and product and sales information,
- ▶ Predicting the time to failure of equipment based on utilization and environmental conditions

The Linear Model

- ▶ Linear model

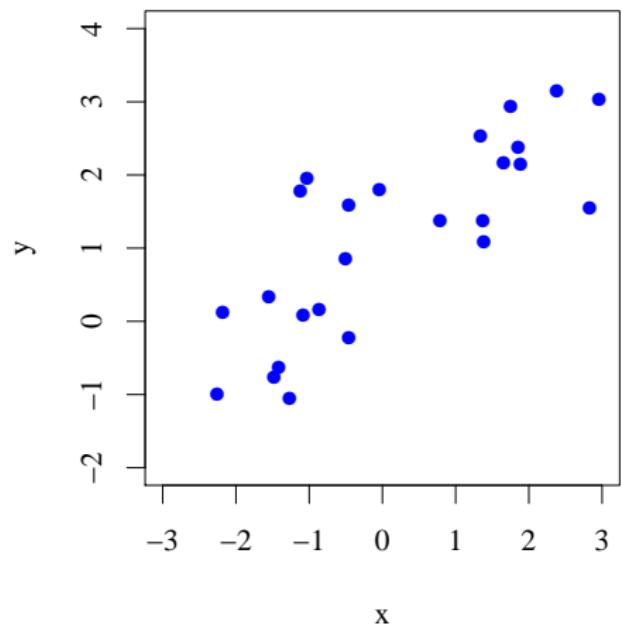
$$\begin{aligned}f(\mathbf{x}; \mathbf{w}) &= w_0 + w_1 x_1 + \dots + w_D x_D \\&= \phi(\mathbf{x})\mathbf{w}\end{aligned}$$

where $\phi(\mathbf{x}) = (1, x_1, \dots, x_D) = (1, \mathbf{x}^T)$
and

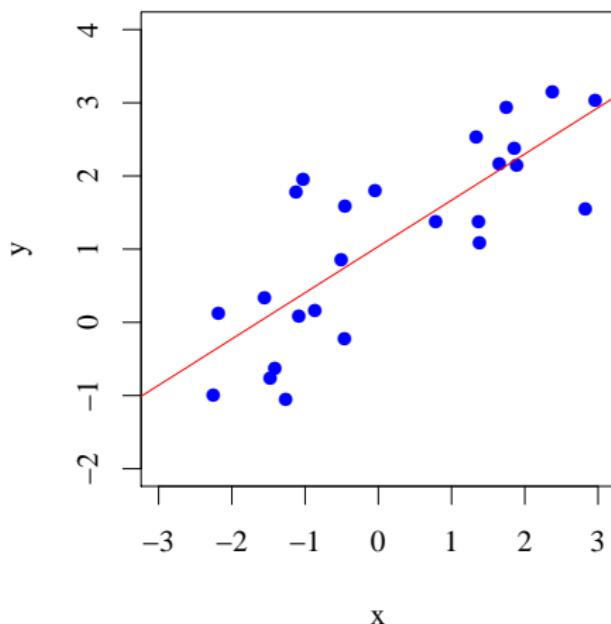
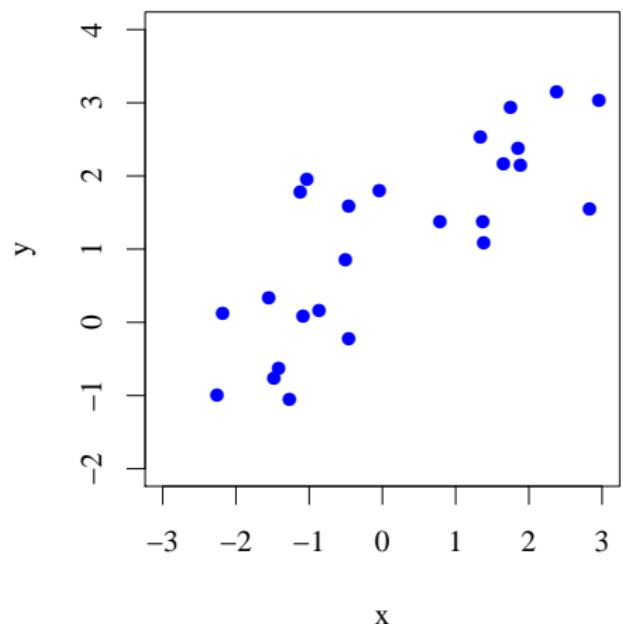
$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_D \end{pmatrix} \quad (1)$$

- ▶ The maths of fitting linear models to data is easy. We use the notation $\phi(\mathbf{x})$ to make generalisation easy later.

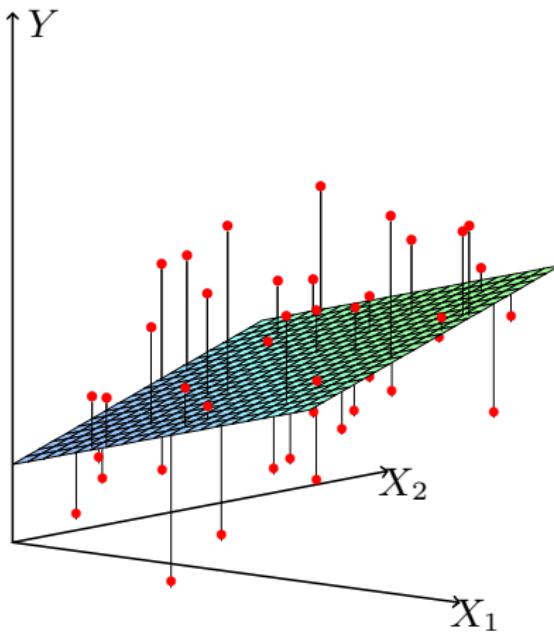
Toy example: Data



Toy example: Data



With two features



Instead of a line, a *plane*. With more features, a *hyperplane*.

Figure: Hastie, Tibshirani, and Friedman

CPU Performance Data Set

- ▶ Predict: PRP: published relative performance
- ▶ MYCT: machine cycle time in nanoseconds (integer)
- ▶ MMIN: minimum main memory in kilobytes (integer)
- ▶ MMAX: maximum main memory in kilobytes (integer)
- ▶ CACH: cache memory in kilobytes (integer)
- ▶ CHMIN: minimum channels in units (integer)
- ▶ CHMAX: maximum channels in units (integer)

With more features

```
PRP = - 56.1  
      + 0.049 MYCT  
      + 0.015 MMIN  
      + 0.006 MMAX  
      + 0.630 CACH  
      - 0.270 CHMIN  
      + 1.46 CHMAX
```

In matrix notation

- ▶ Design matrix is $n \times (D + 1)$

$$\Phi = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nD} \end{pmatrix}$$

- ▶ x_{ij} is the j th component of the training input \mathbf{x}_i
- ▶ Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- ▶ Then $\hat{\mathbf{y}} = \Phi \mathbf{w}$ is ...?

Linear Algebra: The 1-Slide Version

What is matrix multiplication?

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

First consider matrix times vector, i.e., $A\mathbf{b}$. Two answers:

1. $A\mathbf{b}$ is a linear combination of the columns of A

$$A\mathbf{b} = b_1 \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} + b_2 \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix} + b_3 \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}$$

2. $A\mathbf{b}$ is a vector. Each element of the vector is the dot products between b and *one row* of A .

$$A\mathbf{b} = \begin{pmatrix} (a_{11}, a_{12}, a_{13})\mathbf{b} \\ (a_{21}, a_{22}, a_{23})\mathbf{b} \\ (a_{31}, a_{32}, a_{33})\mathbf{b} \end{pmatrix}$$

Linear model (part 2)

In matrix notation:

- ▶ Design matrix is $n \times (D + 1)$

$$\Phi = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nD} \end{pmatrix}$$

- ▶ x_{ij} is the j th component of the training input \mathbf{x}_i
- ▶ Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- ▶ Then $\hat{\mathbf{y}} = \Phi\mathbf{w}$ is the model's predicted values on training inputs.

Solving for Model Parameters

This looks like what we've seen in linear algebra

$$\mathbf{y} = \Phi \mathbf{w}$$

We know \mathbf{y} and Φ but not \mathbf{w} .

So why not take $\mathbf{w} = \Phi^{-1} \mathbf{y}$? (You can't, but why?)

Solving for Model Parameters

This looks like what we've seen in linear algebra

$$\mathbf{y} = \Phi \mathbf{w}$$

We know \mathbf{y} and Φ but not \mathbf{w} .

So why not take $\mathbf{w} = \Phi^{-1}\mathbf{y}$? (You can't, but why?)

Three reasons:

- ▶ Φ is not square. It is $n \times (D + 1)$.
- ▶ The system is overconstrained (n equations for $D + 1$ parameters), in other words
- ▶ The data has noise

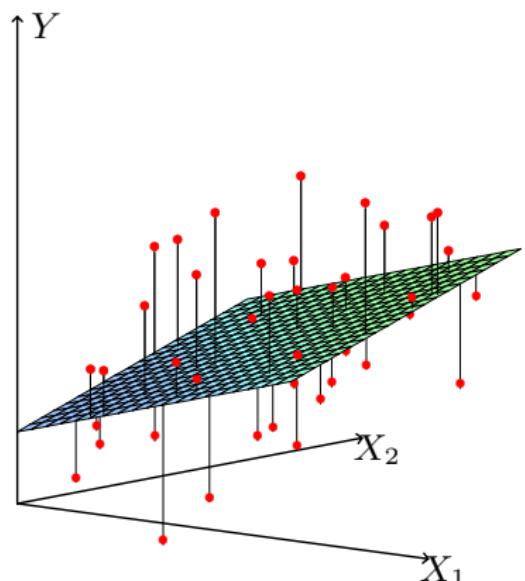
Loss function

Want a *loss function* $O(\mathbf{w})$ that

- ▶ We minimize wrt \mathbf{w} .
- ▶ At minimum, $\hat{\mathbf{y}}$ looks like \mathbf{y} .
- ▶ (Recall: $\hat{\mathbf{y}}$ depends on \mathbf{w})

$$\hat{\mathbf{y}} = \Phi\mathbf{w}$$

Fitting a linear model to data



- ▶ A common choice: *squared error* (makes the maths easy)

$$O(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

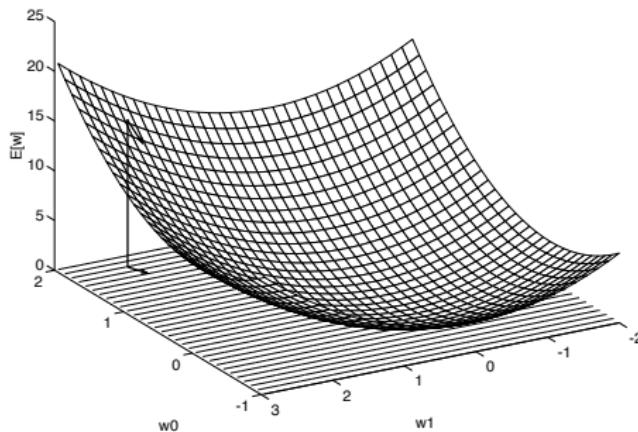
- ▶ In the picture: this is sum of squared length of black sticks.
- ▶ (Each one is called a *residual*, i.e., each $y_i - \mathbf{w}^T \mathbf{x}_i$)

Fitting a linear model to data



$$\begin{aligned}O(\mathbf{w}) &= \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \\&= (\mathbf{y} - \Phi \mathbf{w})^T (\mathbf{y} - \Phi \mathbf{w})\end{aligned}$$

- ▶ We want to minimize this with respect to \mathbf{w} .
- ▶ The error surface is a parabolic bowl



- ▶ How do we do this?

The Solution

- ▶ Answer: to minimize $O(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$, set partial derivatives to 0.
- ▶ This has an analytical solution

$$\hat{\mathbf{w}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

- ▶ $(\Phi^T \Phi)^{-1} \Phi^T$ is the pseudo-inverse of Φ
- ▶ First check: Does this make sense? Do the matrix dimensions line up?
- ▶ Then: Why is this called a pseudo-inverse? ()
- ▶ Finally: What happens if there are no features?

Probabilistic interpretation of $O(\mathbf{w})$

- ▶ Assume that $y = \mathbf{w}^T \mathbf{x} + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$
- ▶ (This is an exact linear relationship plus Gaussian noise.)
- ▶ This implies that $y|\mathbf{x}_i \sim N(\mathbf{w}^T \mathbf{x}_i, \sigma^2)$, i.e.

$$-\log p(y_i|\mathbf{x}_i) = \log \sqrt{2\pi} + \log \sigma + \frac{(y_i - \mathbf{w}^T \mathbf{x}_i)^2}{2\sigma^2}$$

- ▶ So minimising $O(\mathbf{w})$ equivalent to maximising likelihood!
- ▶ Can view $\mathbf{w}^T \mathbf{x}$ as $E[y|\mathbf{x}]$.
- ▶ Squared residuals allow estimation of σ^2

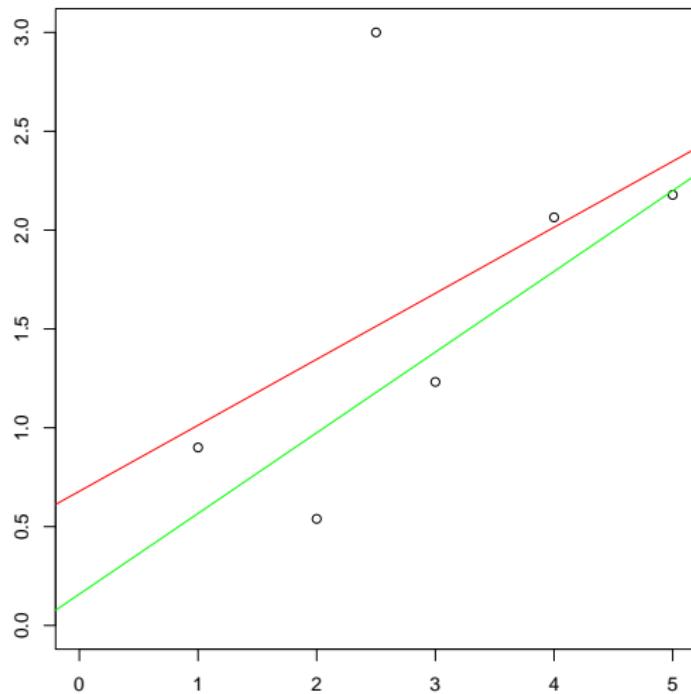
$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Fitting this into the general structure for learning algorithms:

- ▶ Define the **task**: regression
- ▶ Decide on the **model structure**: linear regression model
- ▶ Decide on the **score function**: squared error (likelihood)
- ▶ Decide on **optimization/search method** to optimize the score function: calculus (analytic solution)

Sensitivity to Outliers

- ▶ Linear regression is sensitive to *outliers*
- ▶ Example: Suppose $y = 0.5x + \epsilon$, where $\epsilon \sim N(0, \sqrt{0.25})$, and then add a point (2.5, 3):



Diagnostics

Graphical diagnostics can be useful for checking:

- ▶ Is the relationship obviously nonlinear? Look for structure in residuals?
- ▶ Are there obvious outliers?

The goal isn't to find all problems. You can't. The goal is to find obvious, embarrassing problems.

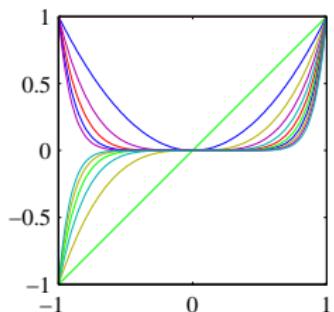
Examples: Plot residuals by fitted values. Stats packages will do this for you.

Dealing with multiple outputs

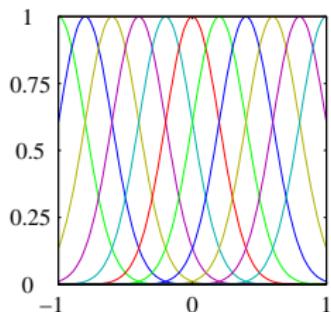
- ▶ Suppose there are q different targets for each input \mathbf{x}
- ▶ We introduce a different \mathbf{w}_i for each target dimension, and do regression separately for each one
- ▶ This is called *multiple regression*

Basis expansion

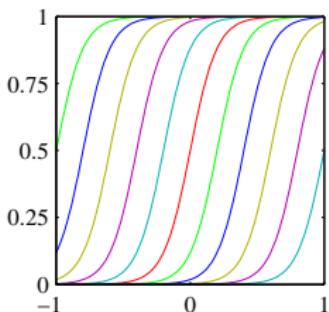
- ▶ We can easily transform the original attributes \mathbf{x} non-linearly into $\phi(\mathbf{x})$ and do linear regression on them



polynomial



Gaussians



sigmoids

Figure credit: Chris Bishop, PRML

- ▶ Design matrix is $n \times m$

$$\Phi = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \dots & \phi_m(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \dots & \phi_m(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\mathbf{x}_n) & \phi_2(\mathbf{x}_n) & \dots & \phi_m(\mathbf{x}_n) \end{pmatrix}$$

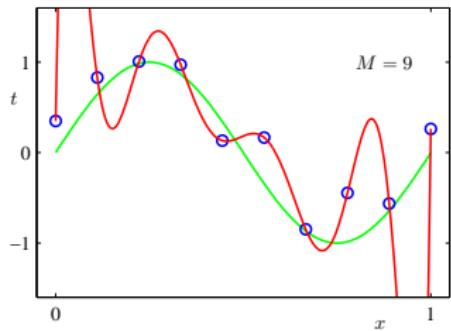
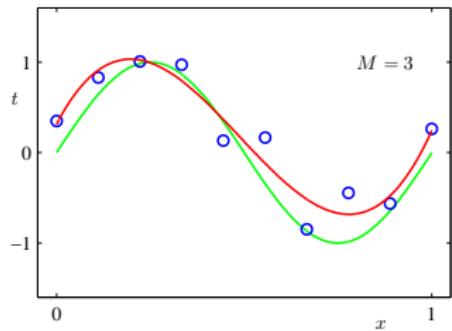
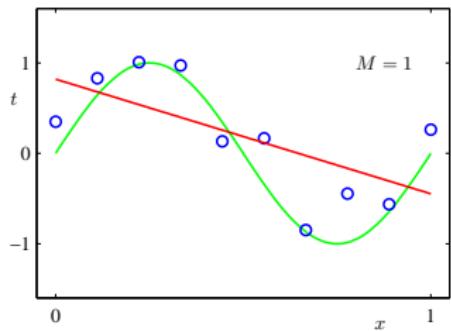
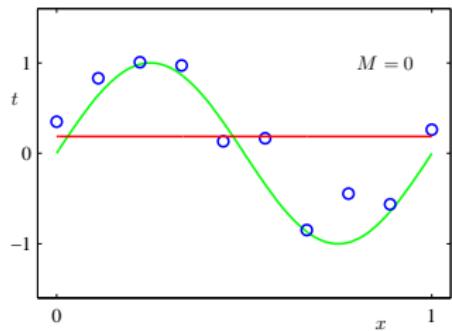
- ▶ Let $\mathbf{y} = (y_1, \dots, y_n)^T$
- ▶ Minimize $E(\mathbf{w}) = |\mathbf{y} - \Phi\mathbf{w}|^2$. As before we have an analytical solution

$$\hat{\mathbf{w}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

- ▶ $(\Phi^T \Phi)^{-1} \Phi^T$ is the pseudo-inverse of Φ

Example: polynomial regression

$$\phi(x) = (1, x, x^2, \dots, x^M)^T$$



More about the features

- ▶ Transforming the features can be important.
- ▶ Example: Suppose I want to predict the CPU performance.
- ▶ Maybe one of the features is *manufacturer*.

$$x_1 = \begin{cases} 1 & \text{if Intel} \\ 2 & \text{if AMD} \\ 3 & \text{if Apple} \\ 4 & \text{if Motorola} \end{cases}$$

- ▶ Let's use this as a feature. Will this work?

More about the features

- ▶ Transforming the features can be important.
- ▶ Example: Suppose I want to predict the CPU performance.
- ▶ Maybe one of the features is *manufacturer*.

$$x_1 = \begin{cases} 1 & \text{if Intel} \\ 2 & \text{if AMD} \\ 3 & \text{if Apple} \\ 4 & \text{if Motorola} \end{cases}$$

- ▶ Let's use this as a feature. Will this work?
- ▶ Not the way you want. Do you really believe AMD is double Intel?

More about the features

- ▶ Instead convert this into 0/1

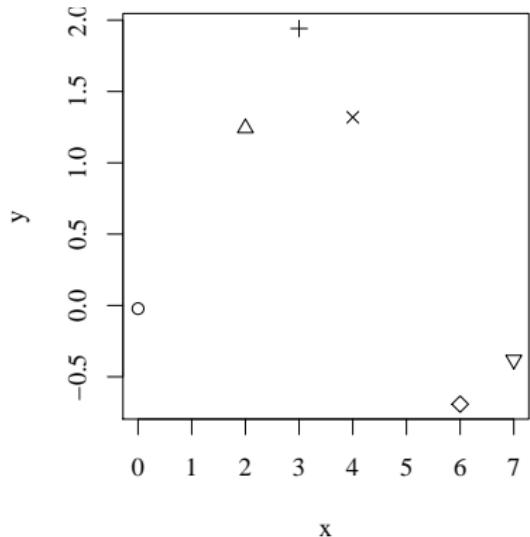
$$x_1 = 1 \text{ if Intel, 0 otherwise}$$
$$x_2 = 1 \text{ if AMD, 0 otherwise}$$
$$\vdots$$

- ▶ Note this is a consequence of linearity. We saw something similar with text in the first week.
- ▶ Other good transformations: log, square, square root

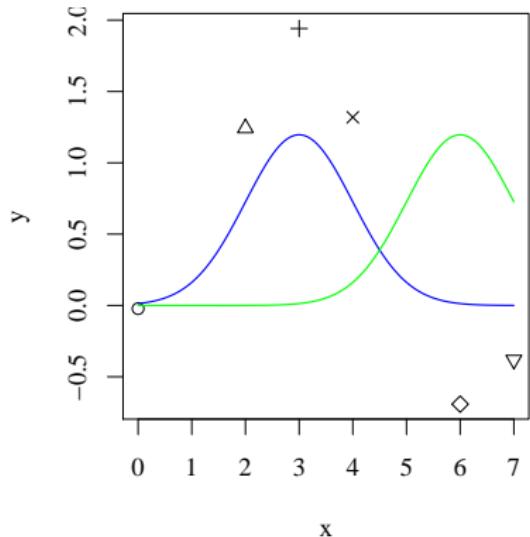
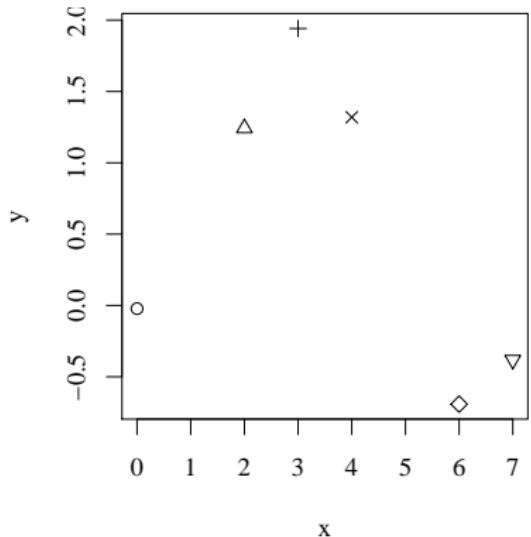
Radial basis function (RBF) models

- ▶ Set $\phi_i(\mathbf{x}) = \exp(-\frac{1}{2}|\mathbf{x} - \mathbf{c}_i|^2/\alpha^2)$.
- ▶ Need to position these “basis functions” at some prior chosen centres \mathbf{c}_i and with a given width α . There are many ways to do this but choosing a subset of the datapoints as centres is one method that is quite effective
- ▶ Finding the weights is the same as ever: the pseudo-inverse solution.

RBF example

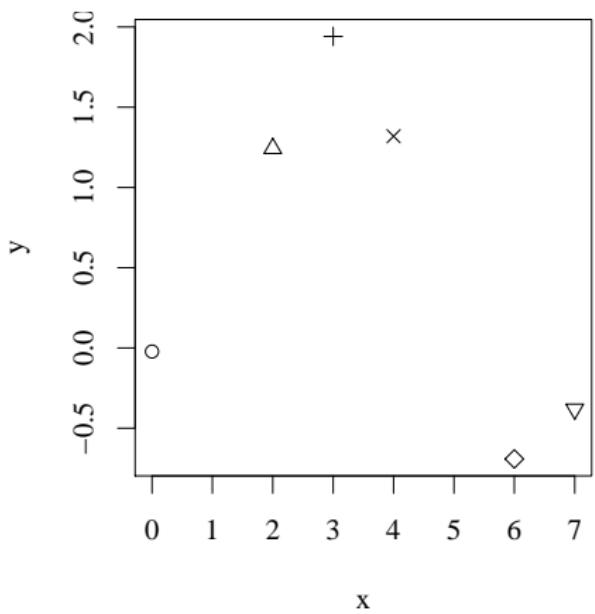


RBF example

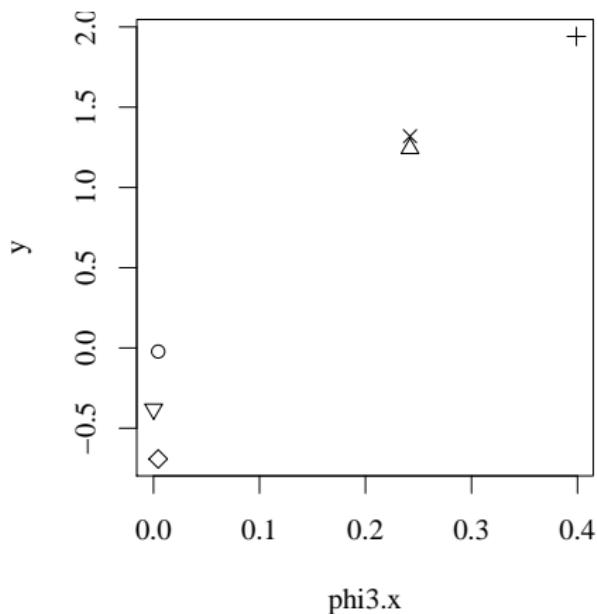


An RBF feature

Original data



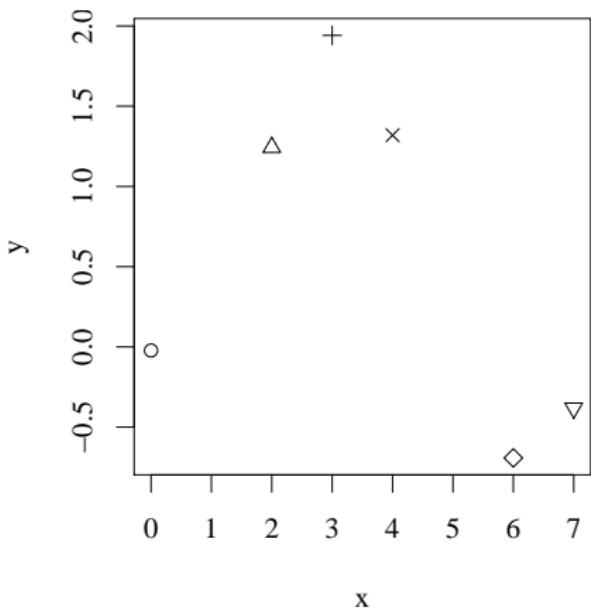
RBF feature, $c_1 = 3$, $\alpha_1 = 1$



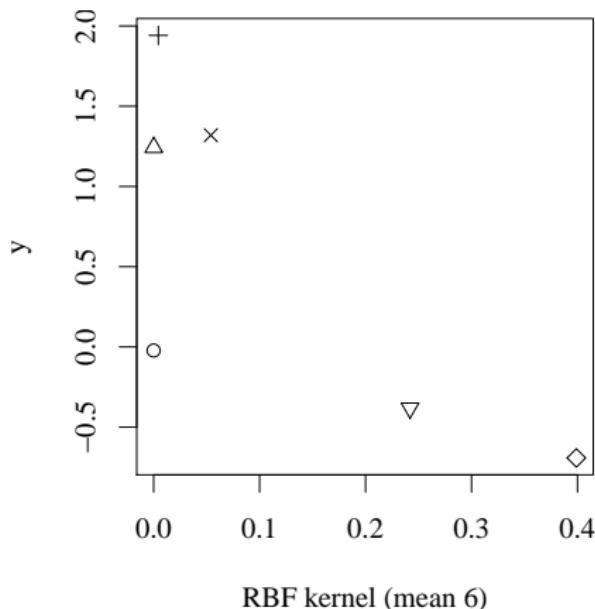
Another RBF feature

Notice how the feature functions “specialize” in input space.

Original data



RBF feature, $c_2 = 6$, $\alpha_2 = 1$



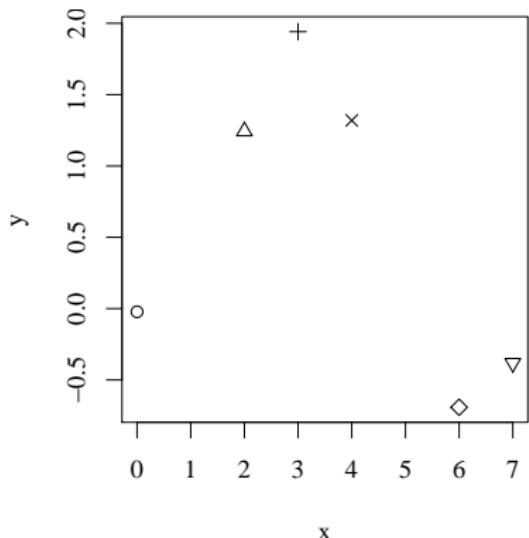
RBF kernel (mean 6)

RBF example

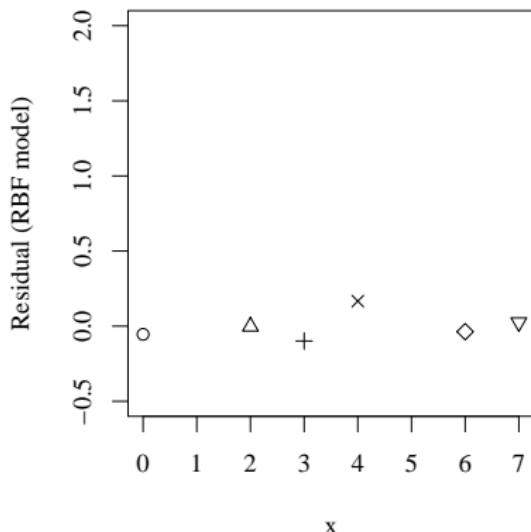
Run the RBF with both basis functions above, plot the residuals

$$y_i - \phi(\mathbf{x}_i)^T \mathbf{w}$$

Original data



Residuals



RBF: Ay, there's the rub

- ▶ So why not use RBFs for everything?
- ▶ Short answer: You might need too many basis functions.
- ▶ This is especially true in high dimensions (we'll say more later)
- ▶ Too many means you probably overfit.
- ▶ Extreme example: Centre one on each training point.
- ▶ Also: notice that we haven't seen yet in the course how to learn the RBF *parameters*, i.e., the mean and standard deviation of each kernel
- ▶ Main point of presenting RBFs now: Set up later methods like support vector machines

Summary

- ▶ Linear regression often useful out of the box.
- ▶ More useful than it would be seem because linear means linear *in the parameters*. You can do a nonlinear transform of the data first, e.g., polynomial, RBF. This point will come up again.
- ▶ Maximum likelihood solution is computationally efficient (pseudo-inverse)
- ▶ Danger of overfitting, especially with many features or basis functions

IAML: Logistic Regression

Nigel Goddard
School of Informatics

Semester 1

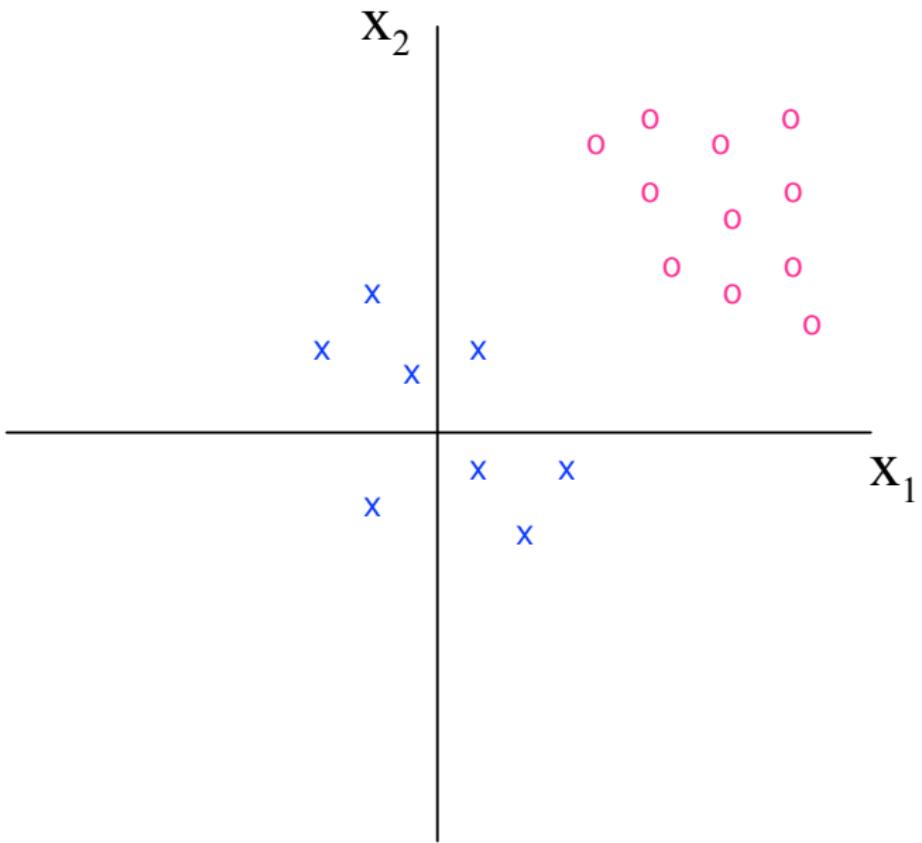
Outline

- ▶ Logistic function
- ▶ Logistic regression
- ▶ Learning logistic regression
- ▶ Optimization
- ▶ The power of non-linear basis functions
- ▶ Least-squares classification
- ▶ Generative and discriminative models
- ▶ Relationships to Generative Models
- ▶ Multiclass classification

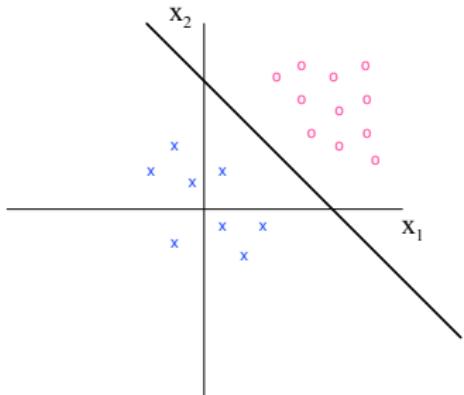
Decision Boundaries

- ▶ In this class we will discuss *linear classifiers*.
- ▶ For each class, there is a *region* of feature space in which the classifier selects one class over the other.
- ▶ The decision boundary is the boundary of this region. (i.e., where the two classes are “tied”)
- ▶ In linear classifiers the decision boundary is a line.

Example Data



Linear Classifiers



- ▶ In a two-class linear classifier, we learn a function

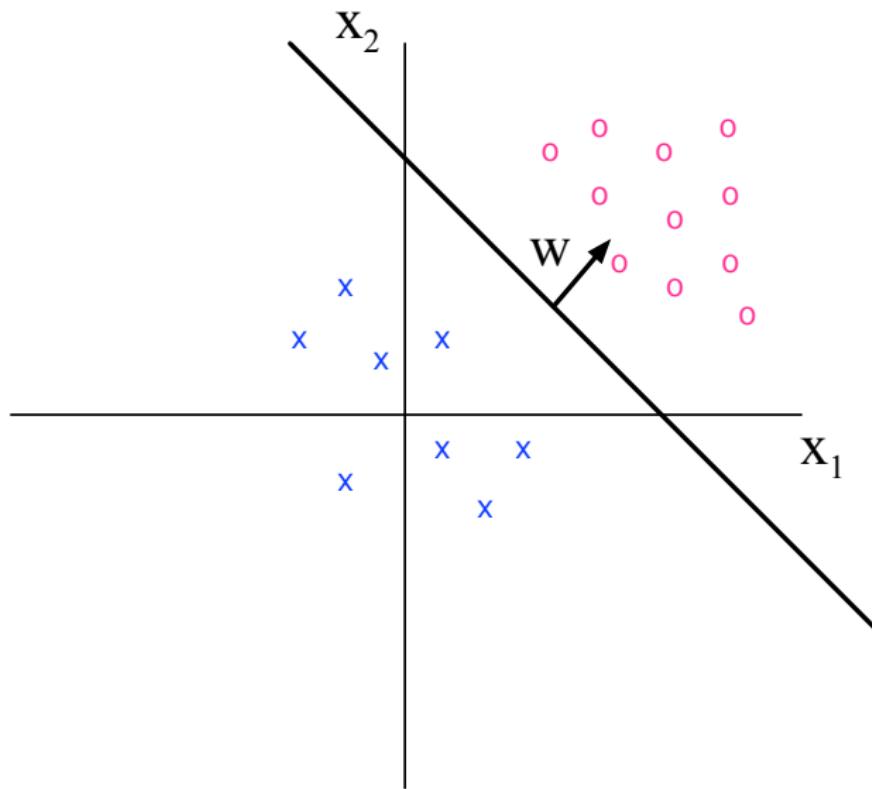
$$F(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \mathbf{x} + w_0$$

that represents how aligned the instance is with $y = 1$.

- ▶ \mathbf{w} are parameters of the classifier that we learn from data.
- ▶ To do classification of an input \mathbf{x} :

$$\mathbf{x} \mapsto (y = 1) \quad \text{if } F(\mathbf{x}, \mathbf{w}) > 0$$

A Geometric View



Explanation of Geometric View

- ▶ The decision boundary in this case is

$$\{\mathbf{x} | \mathbf{w}^\top \mathbf{x} + w_0 = 0\}$$

- ▶ \mathbf{w} is a normal vector to this surface
- ▶ (Remember how lines can be written in terms of their normal vector.)
- ▶ Notice that in more than 2 dimensions, this boundary will be a hyperplane.

Two Class Discrimination

- ▶ For now consider a two class case: $y \in \{0, 1\}$.
- ▶ From now on we'll write $\mathbf{x} = (1, x_1, x_2, \dots, x_d)$ and $\mathbf{w} = (w_0, w_1, \dots, w_d)$.
- ▶ We will want a linear, probabilistic model. We could try $P(y = 1|\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$. But this is stupid.
- ▶ Instead what we will do is

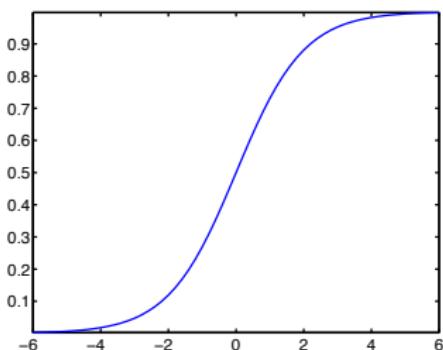
$$P(y = 1|\mathbf{x}) = f(\mathbf{w}^\top \mathbf{x})$$

- ▶ f must be between 0 and 1. It will squash the real line into $[0, 1]$
- ▶ Furthermore the fact that probabilities sum to one means

$$P(y = 0|\mathbf{x}) = 1 - f(\mathbf{w}^\top \mathbf{x})$$

The logistic function

- ▶ We need a function that returns probabilities (i.e. stays between 0 and 1).
- ▶ The logistic function provides this
- ▶ $f(z) = \sigma(z) \equiv 1/(1 + \exp(-z))$.
- ▶ As z goes from $-\infty$ to ∞ , so f goes from 0 to 1, a “squashing function”
- ▶ It has a “sigmoid” shape (i.e. S-like shape)



Linear weights

- ▶ Linear weights + logistic squashing function == logistic regression.
- ▶ We model the class probabilities as

$$p(y = 1|\mathbf{x}) = \sigma\left(\sum_{j=0}^D w_j x_j\right) = \sigma(\mathbf{w}^T \mathbf{x})$$

- ▶ $\sigma(z) = 0.5$ when $z = 0$. Hence the decision boundary is given by $\mathbf{w}^T \mathbf{x} = 0$.
- ▶ Decision boundary is a $M - 1$ hyperplane for a M dimensional problem.

Logistic regression

- ▶ For this slide write $\tilde{\mathbf{w}} = (w_1, w_2, \dots, w_d)$ (i.e., exclude the bias w_0)
- ▶ The bias parameter w_0 shifts the position of the hyperplane, but does not alter the angle
- ▶ The direction of the vector $\tilde{\mathbf{w}}$ affects the angle of the hyperplane. The hyperplane is perpendicular to $\tilde{\mathbf{w}}$
- ▶ The magnitude of the vector $\tilde{\mathbf{w}}$ effects how certain the classifications are
- ▶ For small $\tilde{\mathbf{w}}$ most of the probabilities within the region of the decision boundary will be near to 0.5.
- ▶ For large $\tilde{\mathbf{w}}$ probabilities in the same region will be close to 1 or 0.

Learning Logistic Regression

- ▶ Want to set the parameters \mathbf{w} using training data.
- ▶ As before:
 - ▶ Write out the model and hence the likelihood
 - ▶ Find the derivatives of the log likelihood w.r.t the parameters.
 - ▶ Adjust the parameters to maximize the log likelihood.

- ▶ Assume data is independent and identically distributed.
- ▶ Call the data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- ▶ The likelihood is

$$\begin{aligned} p(D|\mathbf{w}) &= \prod_{i=1}^n p(y = y_i | \mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i=1}^n p(y = 1 | \mathbf{x}_i, \mathbf{w})^{y_i} (1 - p(y = 1 | \mathbf{x}_i, \mathbf{w}))^{1-y_i} \end{aligned}$$

- ▶ Hence the log likelihood $L(\mathbf{w}) = \log p(D|\mathbf{w})$ is given by

$$L(\mathbf{w}) = \sum_{i=1}^n y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))$$

- ▶ It turns out that the likelihood has a unique optimum (given sufficient training examples). It is *convex*.
- ▶ How to maximize? Take gradient

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^n (y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)) x_{ij}$$

- ▶ (Aside: something similar holds for linear regression

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^n (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i) x_{ij}$$

where E is squared error.)

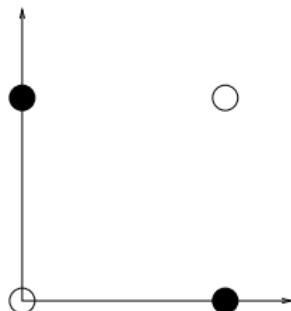
- ▶ Unfortunately, you cannot maximize $L(\mathbf{w})$ explicitly as for linear regression. You need to use a numerical optimisation method, see later.

Fitting this into the general structure for learning algorithms:

- ▶ Define the **task**: classification, discriminative
- ▶ Decide on the **model structure**: logistic regression model
- ▶ Decide on the **score function**: log likelihood
- ▶ Decide on **optimization/search method** to optimize the score function: numerical optimization routine. Note we have several choices here (stochastic gradient descent, conjugate gradient, BFGS).

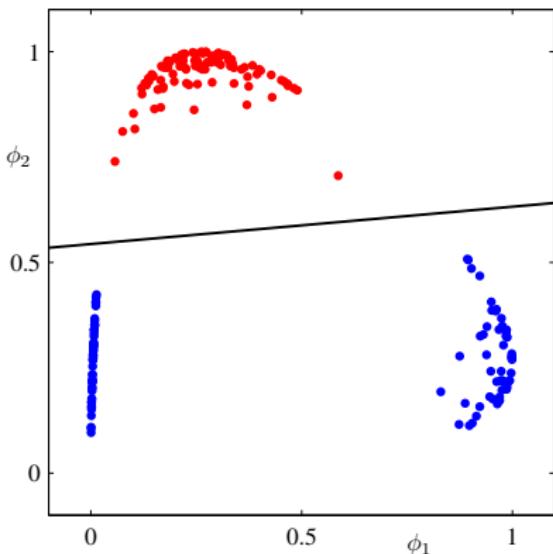
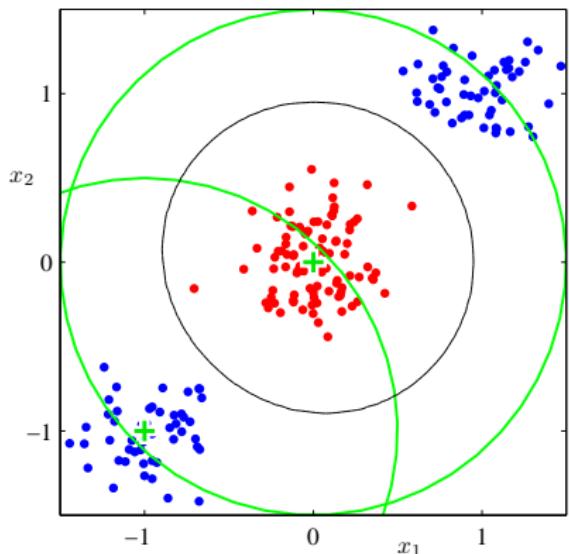
XOR and Linear Separability

- ▶ A problem is linearly separable if we can find weights so that
 - ▶ $\tilde{\mathbf{w}}^T \mathbf{x} + w_0 > 0$ for all positive cases (where $y = 1$), and
 - ▶ $\tilde{\mathbf{w}}^T \mathbf{x} + w_0 \leq 0$ for all negative cases (where $y = 0$)
- ▶ XOR



- ▶ XOR becomes linearly separable if we apply a non-linear transformation $\phi(\mathbf{x})$ of the input — what is one?

The power of non-linear basis functions



Using two Gaussian basis functions $\phi_1(\mathbf{x})$ and $\phi_2(\mathbf{x})$

Figure credit: Chris Bishop, PRML

As for linear regression, we can transform the input space if we want $\mathbf{x} \rightarrow \phi(\mathbf{x})$

Generative and Discriminative Models

- ▶ Notice that we have done something very different here than with naive Bayes.
- ▶ Naive Bayes: Modelled how a class “generated” the feature vector $p(\mathbf{x}|y)$. Then could classify using

$$p(y|\mathbf{x}) \propto p(\mathbf{x}|y)p(y)$$

- . This called is a *generative* approach.
- ▶ Logistic regression: Model $p(y|\mathbf{x})$ directly. This is a *discriminative* approach.
- ▶ Discriminative advantage: Why spend effort modelling $p(\mathbf{x})$? Seems a waste, we’re always given it as input.
- ▶ Generative advantage: Can be good with missing data (remember how naive Bayes handles missing data). Also good for detecting outliers. Or, sometimes you really do want to generate the input.

Generative Classifiers can be Linear Too

Two scenarios where naive Bayes gives you a linear classifier.

1. *Gaussian data with equal covariance.* If

$p(\mathbf{x}|y=1) \sim N(\mu_1, \Sigma)$ and $p(\mathbf{x}|y=0) \sim N(\mu_2, \Sigma)$ then

$$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T \mathbf{x} + w_0)$$

for some $(w_0, \tilde{\mathbf{w}})$ that depends on μ_1 , μ_2 , Σ and the class priors

2. *Binary data.* Let each component x_j be a Bernoulli variable i.e. $x_j \in \{0, 1\}$. Then a Naïve Bayes classifier has the form

$$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T \mathbf{x} + w_0)$$

3. Exercise for keeners: prove these two results

Multiclass classification

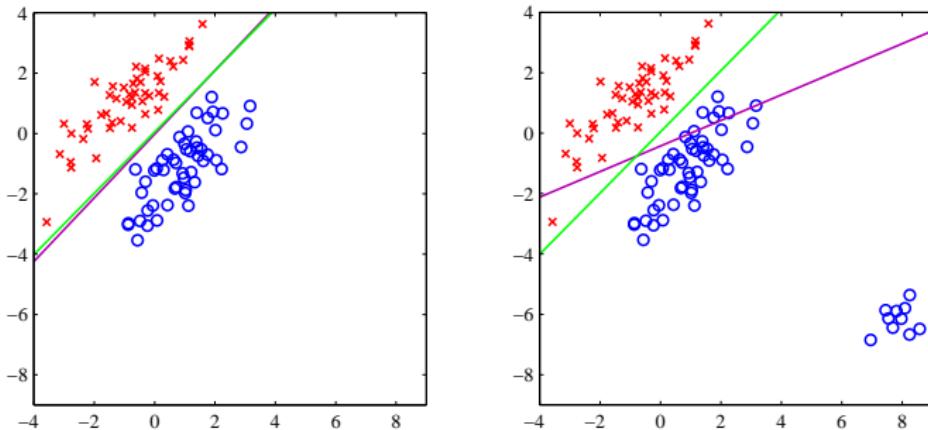
- ▶ Create a different weight vector \mathbf{w}_k for each class, to classify into k and not- k .
- ▶ Then use the “softmax” function

$$p(y = k | \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_{j=1}^C \exp(\mathbf{w}_j^T \mathbf{x})}$$

- ▶ Note that $0 \leq p(y = k | \mathbf{x}) \leq 1$ and $\sum_{j=1}^C p(y = j | \mathbf{x}) = 1$
- ▶ This is the natural generalization of logistic regression to more than 2 classes.

Least-squares classification

- ▶ Logistic regression is more complicated algorithmically than linear regression
- ▶ Why not just use linear regression with 0/1 targets?



Green: logistic regression; magenta, least-squares regression

Figure credit: Chris Bishop, PRML

IAML: Optimization

Nigel Goddard
School of Informatics

Semester 1

Outline

- ▶ Why we use optimization in machine learning
- ▶ The general optimization problem
- ▶ Gradient descent
- ▶ Problems with gradient descent
- ▶ Batch versus online
- ▶ Second-order methods
- ▶ Constrained optimization

Many illustrations, text, and general ideas from these slides are taken from Sam Roweis (1972-2010).

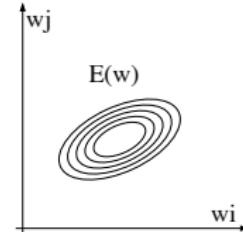
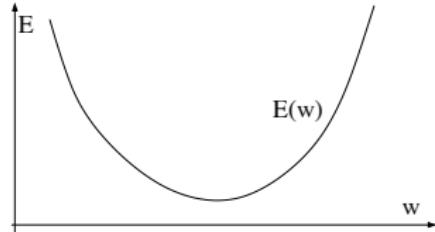
Why Optimization

- ▶ A main idea in machine learning is to convert the learning problem into a continuous optimization problem.
- ▶ Examples: Linear regression, logistic regression (we have seen), neural networks, SVMs (we will see these later)
- ▶ One way to do this is *maximum likelihood*

$$\begin{aligned}\ell(\mathbf{w}) &= \log p(y_1, \mathbf{x}_1, y_2, \mathbf{x}_2, \dots, y_n, \mathbf{x}_n | \mathbf{w}) \\ &= \log \prod_{i=1}^n p(y_i, \mathbf{x}_i | \mathbf{w}) \\ &= \sum_{i=1}^n \log p(y_i, \mathbf{x}_i | \mathbf{w})\end{aligned}$$

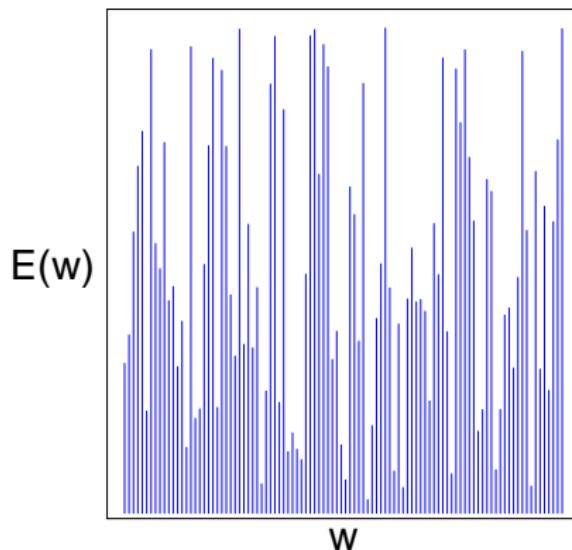
- ▶ Example: Linear regression

- ▶ End result: an “error function” $E(\mathbf{w})$ which we want to minimize.
- ▶ e.g., $E(\mathbf{w})$ can be the negative of the log likelihood.
- ▶ Consider a fixed training set; think in weight (not input) space. At each setting of the weights there is some error (given the fixed training set): this defines an error surface in weight space.
- ▶ Learning == descending the error surface.
- ▶ If the data are IID, the error function E is a sum of error function E_i for each data point



Role of Smoothness

If E completely unconstrained, minimization is impossible.



All we could do is search through all possible values \mathbf{w} .

Key idea: If E is continuous, then measuring $E(\mathbf{w})$ gives information about E at many nearby values.

Role of Derivatives

- ▶ If we wiggle w_k and keep everything else the same, does the error get better or worse?
- ▶ Calculus has an answer to exactly this question: $\frac{\partial E}{\partial w_k}$
- ▶ So: use a differentiable cost function E and compute partial derivatives of each parameter
- ▶ The vector of partial derivatives is called the gradient of the error. It is written $\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_n} \right)$. Alternate notation $\frac{\partial E}{\partial \mathbf{w}}$.
- ▶ It points in the direction of steepest error descent in weight space.
- ▶ Three crucial questions:
 - ▶ How do we compute the gradient ∇E efficiently?
 - ▶ Once we have the gradient, how do we minimize the error?
 - ▶ Where will we end up in weight space?

Numerical Optimization Algorithms

- ▶ **Numerical optimization** algorithms try to solve the general problem

$$\min_{\mathbf{w}} E(\mathbf{w})$$

- ▶ Most commonly, a numerical optimization procedure takes two inputs:
 - ▶ A procedure that computes $E(\mathbf{w})$
 - ▶ A procedure that computes the partial derivative $\frac{\partial E}{\partial w_j}$
- ▶ (Aside: Some use less information, i.e., they don't use gradients. Some use more information, i.e., higher order derivative. We won't go into these algorithms in the course.)

Optimization Algorithm Cartoon

- ▶ Basically, numerical optimization algorithms are iterative.
They generate a sequence of points

$$\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \dots$$

$$E(\mathbf{w}_0), E(\mathbf{w}_1), E(\mathbf{w}_2), \dots$$

$$\nabla E(\mathbf{w}_0), \nabla E(\mathbf{w}_1), \nabla E(\mathbf{w}_2), \dots$$

- ▶ Basic optimization algorithm is

initialize \mathbf{w}

while $E(\mathbf{w})$ is unacceptably high

 calculate $\mathbf{g} = \nabla E$

 Compute direction \mathbf{d} from $\mathbf{w}, E(\mathbf{w}), \mathbf{g}$

 (can use previous gradients as well...)

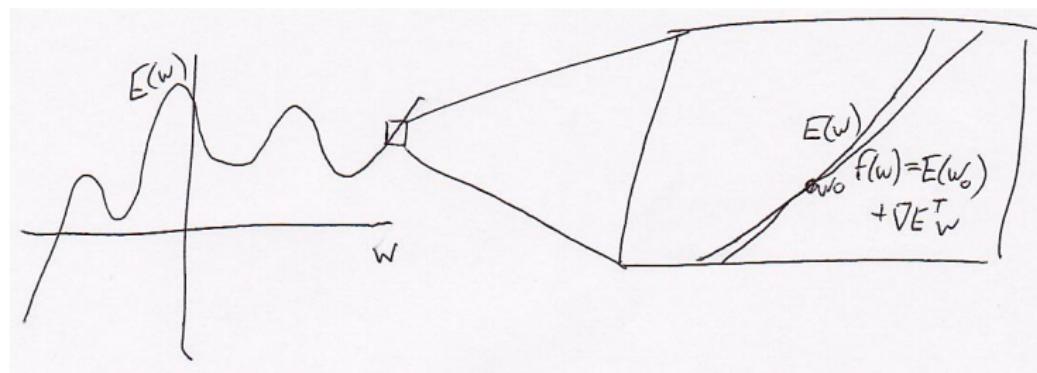
$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{d}$

end while

return \mathbf{w}

A Choice of Direction

- ▶ The simplest choice \mathbf{d} is the current gradient ∇E .
- ▶ It is locally the steepest descent direction.
- ▶ (Technically, the reason for this choice is Taylor's theorem from calculus.)



Gradient Descent

- ▶ Simple gradient descent algorithm:

initialize \mathbf{w}

while $E(\mathbf{w})$ is unacceptably high

 calculate $\mathbf{g} \leftarrow \frac{\partial E}{\partial \mathbf{w}}$

$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$

end while

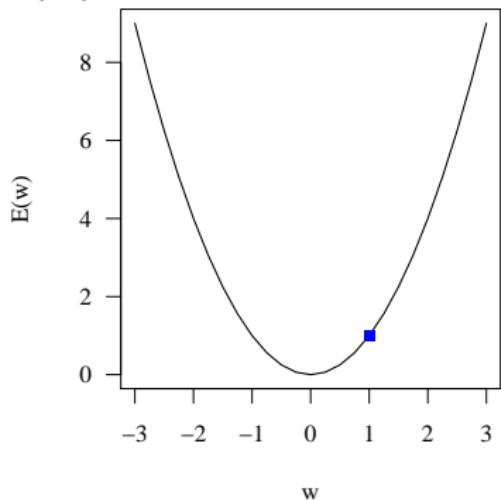
return \mathbf{w}

- ▶ η is known as the *step size* (sometimes called *learning rate*)
 - ▶ We must choose $\eta > 0$.
 - ▶ η too small \rightarrow too slow
 - ▶ η too large \rightarrow instability

Effect of Step Size

Goal: Minimize

$$E(w) = w^2$$



- ▶ Take $\eta = 0.1$. Works well.

$$w_0 = 1.0$$

$$w_1 = w_0 - 0.1 \cdot 2w_0 = 0.8$$

$$w_2 = w_1 - 0.1 \cdot 2w_1 = 0.64$$

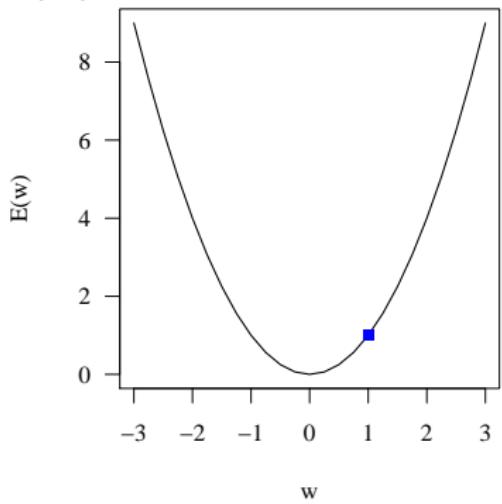
$$w_3 = w_2 - 0.1 \cdot 2w_2 = 0.512$$

...

$$w_{25} = 0.0047$$

Effect of Step Size

Goal: Minimize
 $E(w) = w^2$



- Take $\eta = 1.1$. Not so good. If you step too far, you can leap over the region that contains the minimum

$$w_0 = 1.0$$

$$w_1 = w_0 - 1.1 \cdot 2w_0 = -1.2$$

$$w_2 = w_1 - 1.1 \cdot 2w_1 = 1.44$$

$$w_3 = w_2 - 1.1 \cdot 2w_2 = -1.72$$

...

$$w_{25} = 79.50$$

- Finally, take $\eta = 0.000001$. What happens here?

“Bold Driver” Gradient Descent

- ▶ Simple heuristic for choosing η which you can use if you’re desperate.

initialize \mathbf{w} , η

initialize $e \leftarrow E(\mathbf{w})$; $\mathbf{g} \leftarrow \nabla E(\mathbf{w})$ while $\eta > 0$

$\mathbf{w}_1 \leftarrow \mathbf{w} - \eta \mathbf{g}$

$e_1 = E(\mathbf{w}_1)$; $\mathbf{g}_1 = \nabla E$

if $e_1 \geq e$

$\eta = \eta/2$

else

$\eta = 1.01\eta$; $\mathbf{w} \leftarrow \mathbf{w}_1$; $\mathbf{g} \leftarrow \mathbf{g}_1$; $e = e_1$

end while

return \mathbf{w}

- ▶ Finds a *local* minimum of E .

Batch vs online

- ▶ So far all the objective function we have seen look like:

$$E(\mathbf{w}; D) = \sum_{i=1}^n E_i(\mathbf{w}; y_i, \mathbf{x}_i).$$

$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ is the training set.

- ▶ Each term sum depends on only one training instance
- ▶ Example: Logistic regression: $E_i(\mathbf{w}; y_i, \mathbf{x}_i) = \log p(y_i | \mathbf{x}_i, \mathbf{w})$.
- ▶ The gradient in this case is always

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial E_i}{\partial \mathbf{w}}$$

- ▶ The algorithm on slide 10 scans *all* the training instances before changing the parameters.
- ▶ Seems dumb if we have millions of training instances. Surely we can get a gradient that is “good enough” from fewer instances, e.g., a couple of thousand? Or maybe even from just one?

Batch vs online

- ▶ **Batch** learning: use all patterns in training set, and update weights after calculating

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_i \frac{\partial E_i}{\partial \mathbf{w}}$$

- ▶ **On-line** learning: adapt weights after each pattern presentation, using $\frac{\partial E_i}{\partial \mathbf{w}}$
- ▶ **Batch** more powerful optimization methods
- ▶ **Batch** easier to analyze
- ▶ **On-line** more feasible for huge or continually growing datasets
- ▶ **On-line** may have ability to jump over local optima

Algorithms for Batch Gradient Descent

- ▶ Here is batch gradient descent.

initialize \mathbf{w}

while $E(\mathbf{w})$ is unacceptably high

 calculate $\mathbf{g} \leftarrow \sum_{i=1}^N \frac{\partial E_i}{\partial \mathbf{w}}$

$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$

end while

return \mathbf{w}

- ▶ This is just the algorithm we have seen before. We have just “substituted in” the fact that $E = \sum_{i=1}^N E_i$.

Algorithms for Online Gradient Descent

- ▶ Here is (a particular type of) online gradient descent algorithm

initialize \mathbf{w}

while $E(\mathbf{w})$ is unacceptably high

 Pick j as uniform random integer in $1 \dots N$

 calculate $\mathbf{g} \leftarrow \frac{\partial E_j}{\partial \mathbf{w}}$

$\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$

end while

return \mathbf{w}

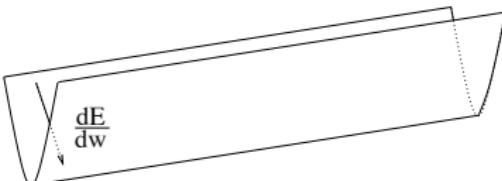
- ▶ This version is also called “stochastic gradient ascent” because we have picked the training instance randomly.
- ▶ There are other variants of online gradient descent.

Problems With Gradient Descent

- ▶ Setting the step size η
- ▶ Shallow valleys
- ▶ Highly curved error surfaces
- ▶ Local minima

Shallow Valleys

- ▶ Typical gradient descent can be fooled in several ways, which is why more sophisticated methods are used when possible. One problem:



- ▶ Gradient descent goes very slowly once it hits the shallow valley.
- ▶ One hack to deal with this is *momentum*

$$\mathbf{d}_t = \beta \mathbf{d}_{t-1} + (1 - \beta) \eta \nabla E(\mathbf{w}_t)$$

- ▶ Now you have to set both η and β . Can be difficult and irritating.

Curved Error Surfaces

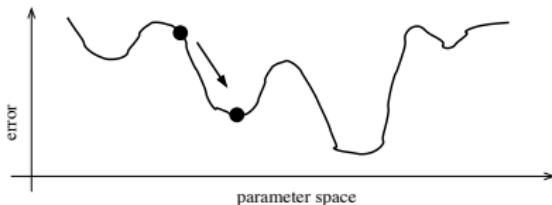
- ▶ A second problem with gradient descent is that the gradient might not point towards the optimum. This is because of curvature



- ▶ Note: gradient is the *locally* steepest direction. Need not directly point toward local optimum.
- ▶ Local curvature is measured by the Hessian matrix:
$$H_{ij} = \partial^2 E / \partial w_i \partial w_j.$$

Local Minima

- If you follow the gradient, where will you end up? Once you hit a local minimum, gradient is 0, so you stop.



- Certain nice functions, such as squared error, logistic regression likelihood are *convex*, meaning that the second derivative is always positive. This implies that any local minimum is global.
- There is no great solution to this problem. It is a fundamental one. Usually, the best you can do is rerun the optimizer multiple times from different random starting points.

Advanced Topics That We Will Not Cover (Part I)

- ▶ Some of these issues (shallow valley, curved error surfaces) can be fixed
 - ▶ Some of these are *second-order* methods like Newton's method that use the second derivatives
 - ▶ Also there are fancy first-order methods like quasi-Newton methods (e.g., *limited memory BFGS*) and conjugate gradient
 - ▶ They are the state of the art methods for logistic regression (as long as there are not too many data points)
 - ▶ We will not discuss these methods in the course.
- ▶ Other issues (like local minima) cannot be easily fixed

Advanced Topics That We Will Not Cover (Part II)

- ▶ Sometimes the optimization problem has *constraints*
 - ▶ Example: Observe the points $\{0.5, 1.0\}$ from a Gaussian with known mean $\mu = 0.8$ and unknown standard deviation σ . Want to estimate σ by maximum likelihood.
 - ▶ Constraint: σ must be positive.
 - ▶ In this case to find the maximum likelihood solution, the optimization problem is

$$\max_{\mu, \sigma} \sum_{i=1}^2 \frac{1}{2\sigma^2} (x_i - \mu)^2$$

subject to $\sigma > 0$

- ▶ There are ways to solve this (in this case: can be done analytically). We will not discuss them in this course.

Summary

- ▶ Complex mathematical area. Do not implement your own optimization algorithms if you can help it!
- ▶ Stuff you should understand:
 - ▶ How and why we convert learning problems into optimization problems
 - ▶ *Modularity* between modelling and optimization
 - ▶ Gradient descent
 - ▶ Why gradient descent can run into problems
 - ▶ Especially local minima
- ▶ Methods of choice: Fancy first-order methods (e.g., quasi-Newton, CG) for moderate amounts of data.
Stochastic gradient for large amounts of data.

IAML: Regularization and Ridge Regression

Nigel Goddard
School of Informatics

Semester 1

Regularization

- ▶ Regularization is a general approach to add a “complexity parameter” to a learning algorithm. Requires that the **model** parameters be continuous. (i.e., Regression OK, Decision trees not.)
- ▶ If we penalize polynomials that have large values for their coefficients we will get less wiggly solutions

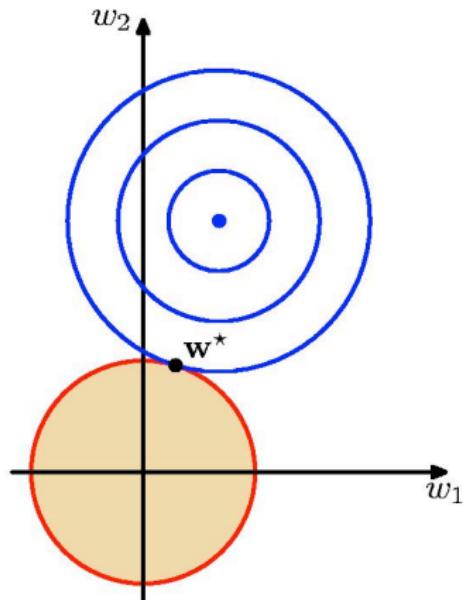
$$\tilde{E}(\mathbf{w}) = |\mathbf{y} - \Phi\mathbf{w}|^2 + \lambda|\mathbf{w}|^2$$

- ▶ Solution is

$$\hat{\mathbf{w}} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{y}$$

- ▶ This is known as *ridge regression*
- ▶ Rather than using a discrete control parameter like M (model order) we can use a continuous parameter λ
- ▶ Caution: Don't shrink the bias term! (The one that corresponds to the all 1 feature.)

Regularized Loss Function



- ▶ The overall cost function is the sum of two parabolic bowls. The sum is also a parabolic bowl.
- ▶ The combined minimum lies on the line between the minimum of the squared error and the origin.
- ▶ The regularizer just shrinks the weights.

Credit: Geoff Hinton

The effect of regularization for $M = 9$

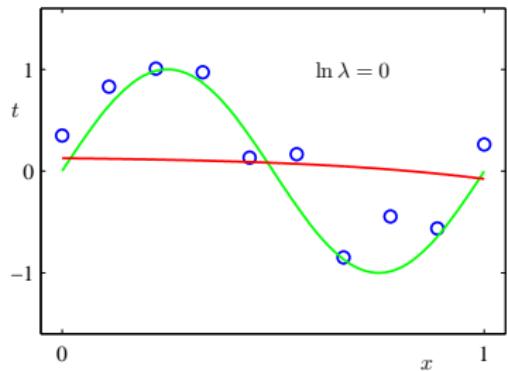
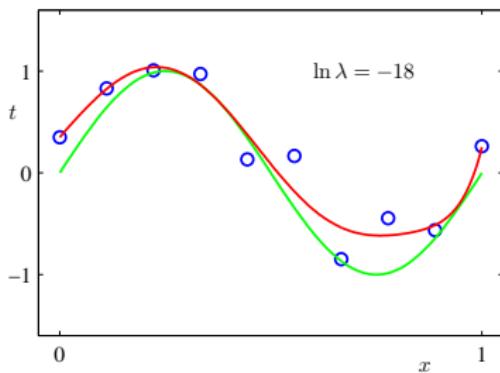
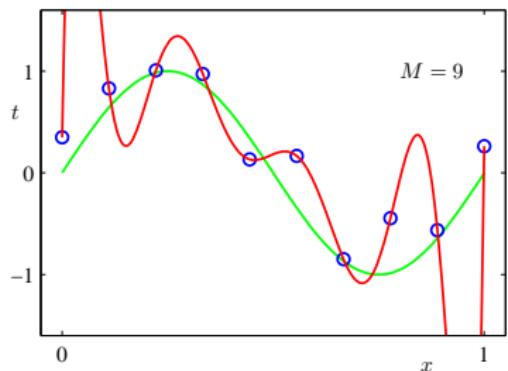
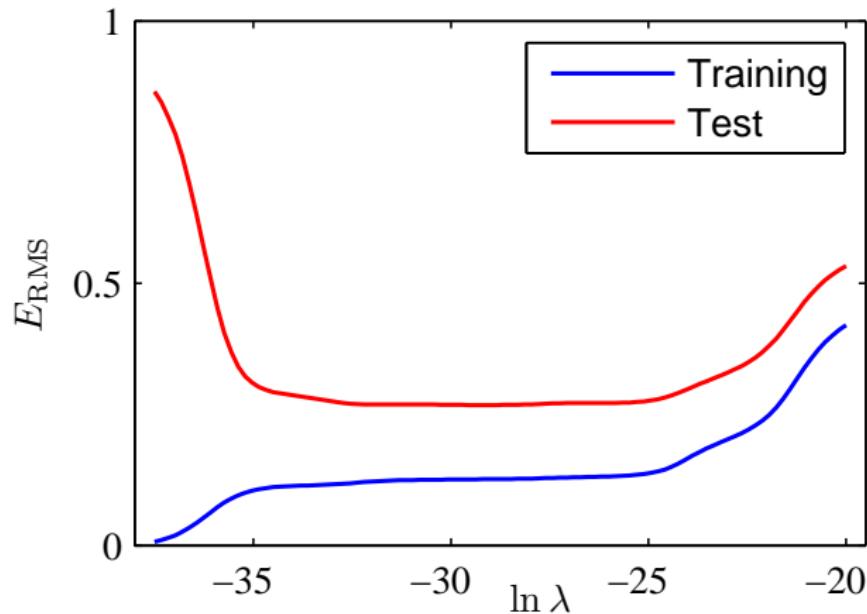


Figure credit: Chris Bishop, PRML

$$M = 9$$



For standard linear regression, we had

- ▶ Define the **task**: regression
- ▶ Decide on the **model structure**: linear regression model
- ▶ Decide on the **score function**: squared error (likelihood)
- ▶ Decide on **optimization/search method** to optimize the score function: calculus (analytic solution)

But with ridge regression we have

- ▶ Define the **task**: regression
- ▶ Decide on the **model structure**: linear regression model
- ▶ Decide on the **score function**: squared error **with quadratic regularization**
- ▶ Decide on **optimization/search method** to optimize the score function: calculus (analytic solution)

Notice how you can train the same model structure with different score functions. This is the first time we have seen this. This is important.

A Control-Parameter-Setting Procedure

- ▶ Regularization was a way of adding a “capacity control” parameter.
- ▶ But how do we set the value? e.g., the regularization parameter λ
- ▶ Won’t work to do it on the training set (why not?)
- ▶ Two choices to consider:
 - ▶ Validation set
 - ▶ Cross-validation

Using a validation set

- ▶ Split the labelled data into a training set, validation set, and a test set.
- ▶ Training set: Use for training
- ▶ Validation set: Tune the “control parameters” according to performance on the validation set
- ▶ Test set: to check how the final model performs
- ▶ No right answers, but for example, could choose 60% training, 20% validation, 20% test

Example of using a validation set

Consider polynomial regression:

1. For each $m = 1, 2, \dots, M$ (you choose M in advance)
2. Train the polynomial regression using
 $\phi(x) = (1, x, x^2, \dots, x^m)^T$ on training set (e.g., by minimizing squared error). This produces a predictor $f_m(\mathbf{x})$.
3. Measure the error of f_m on the validation set
4. End for
5. Choose the f_m with the best validation error.
6. Measure the error of f_m on the test set to see how well you should expect it to perform

Continuous Control Parameters

- ▶ For a discrete control parameter like polynomial order m we could simply search all values.
- ▶ What about a quadratic regularization parameter λ . What do we do then?

Continuous Control Parameters

- ▶ For a discrete control parameter like polynomial order m we could simply search all values.
- ▶ What about a quadratic regularization parameter λ . What do we do then?
- ▶ Pick a grid of values to search. In practice you want the grid to vary geometrically for this sort of parameter. e.g., Try $\lambda \in \{0.01, 0.1, 0.5, 1.0, 5.0, 10.0\}$. Don't bother trying 2.0, 3.0, 7.0.

IAML: Support Vector Machines I

Nigel Goddard
School of Informatics

Semester 1

Outline

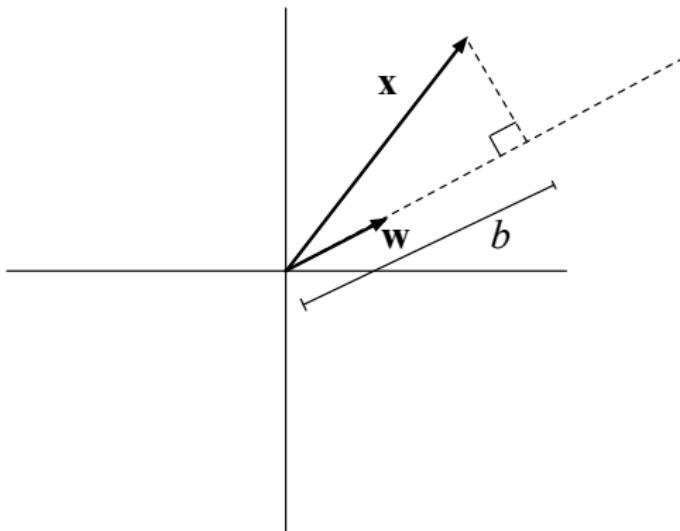
- ▶ Separating hyperplane with maximum margin
- ▶ Non-separable training data
- ▶ Expanding the input into a high-dimensional space
- ▶ Support vector regression
- ▶ Reading: W & F sec 6.3 (maximum margin hyperplane, nonlinear class boundaries), SVM handout. SV regression not examinable.

Overview

- ▶ Support vector machines are one of the most effective and widely used classification algorithms.
- ▶ SVMs are the combination of two ideas
 - ▶ Maximum margin classification
 - ▶ The “kernel trick”
- ▶ SVMs are a linear classifier, like logistic regression and perceptron

Stuff You Need to Remember

$\mathbf{w}^\top \mathbf{x}$ is length of the projection of \mathbf{x} onto \mathbf{w} (if \mathbf{w} is a unit vector)



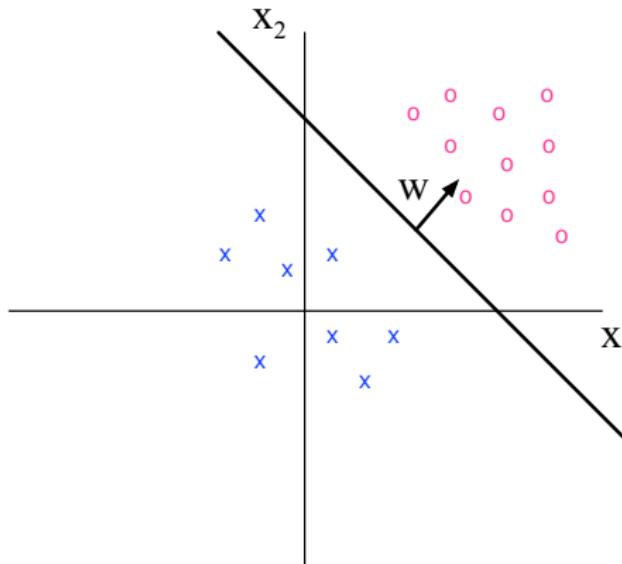
i.e., $b = \mathbf{w}^\top \mathbf{x}$.

(If you do not remember this, see supplementary maths notes on course Web site.)

Separating Hyperplane

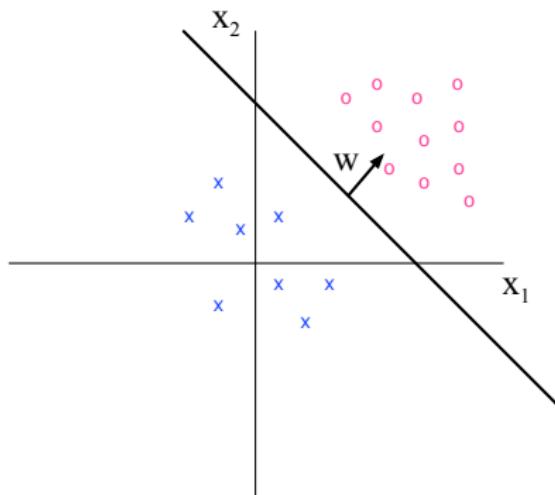
For any linear classifier

- ▶ Training instances (\mathbf{x}_i, y_i) , $i = 1, \dots, n$. $y_i \in \{-1, +1\}$
- ▶ Hyperplane $\mathbf{w}^\top \mathbf{x} + w_0 = 0$
- ▶ Notice for this lecture we use -1 rather than 0 for negative class. This will be convenient for the maths.

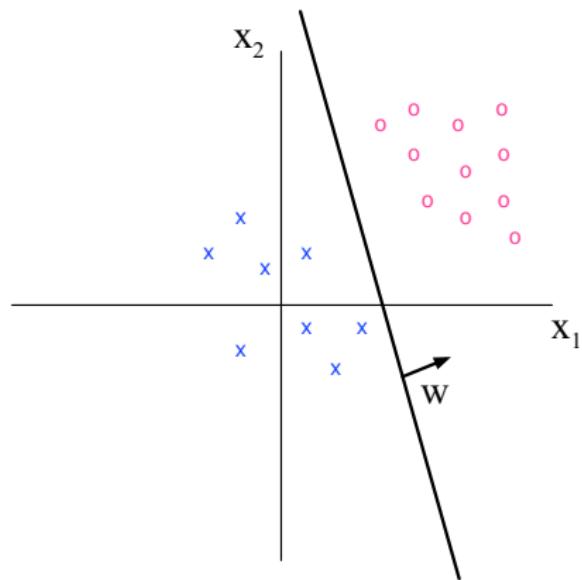


A Crap Decision Boundary

Seems okay

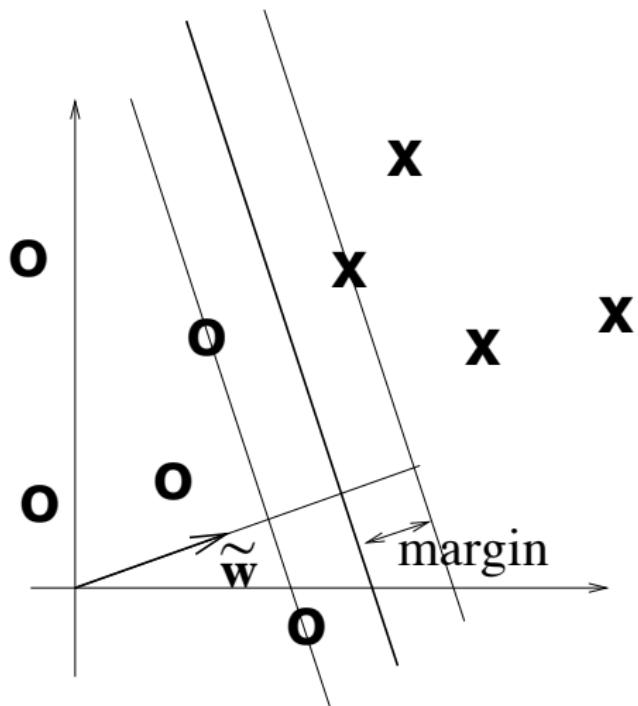


This is crap



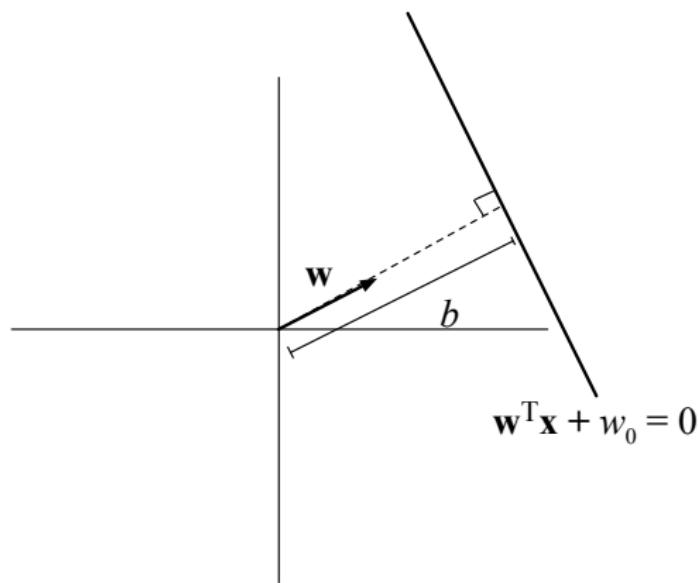
Idea: Maximize the Margin

The **margin** is the distance between the decision boundary (the hyperplane) and the closest training point.

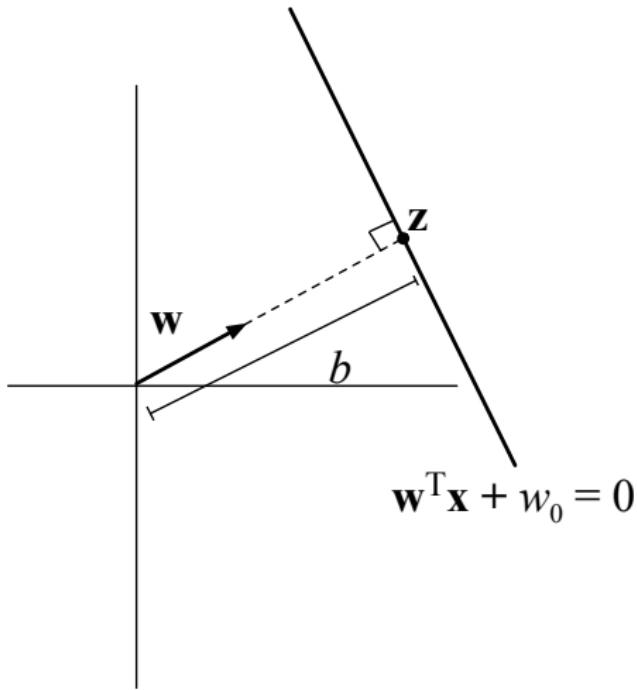


Computing the Margin

- ▶ The tricky part will be to get an equation for the margin
- ▶ We'll start by getting the distance from the origin to the hyperplane
- ▶ i.e., We want to compute the scalar b below



Computing the Distance to Origin



- ▶ Define \mathbf{z} as the point on the hyperplane closest to the origin.
- ▶ \mathbf{z} must be proportional to \mathbf{w} , because \mathbf{w} normal to hyperplane
- ▶ By definition of b , we have the norm of \mathbf{z} given by:

$$\|\mathbf{z}\| = b$$

So

$$b \frac{\mathbf{w}}{\|\mathbf{w}\|} = \mathbf{z}$$

Computing the Distance to Origin

- ▶ We know that (a) \mathbf{z} on the hyperplane and (b) $b \frac{\mathbf{w}}{||\mathbf{w}||} = \mathbf{z}$.
- ▶ First (a) means $\mathbf{w}^T \mathbf{z} + w_0 = 0$
- ▶ Substituting we get

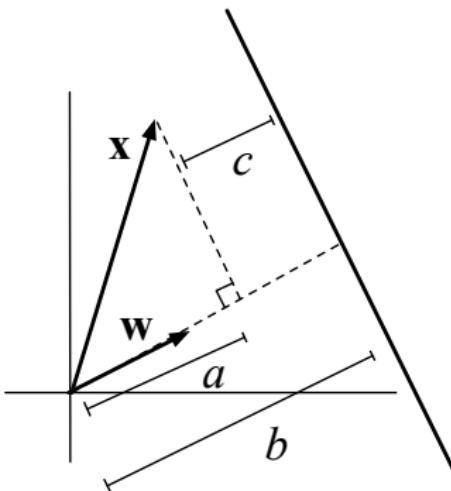
$$\mathbf{w}^T \frac{b\mathbf{w}}{||\mathbf{w}||} + w_0 = 0$$

$$\frac{b\mathbf{w}^T \mathbf{w}}{||\mathbf{w}||} + w_0 = 0$$

$$b = -\frac{w_0}{||\mathbf{w}||}$$

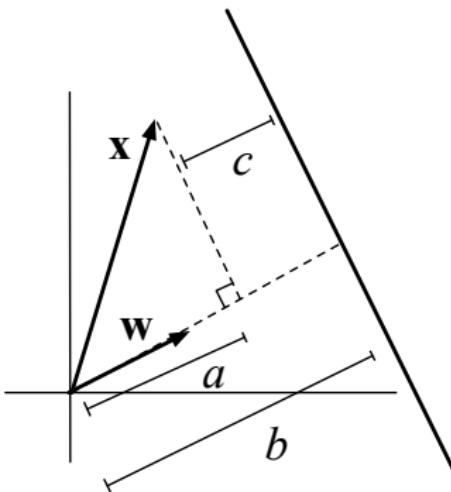
- ▶ Remember $||\mathbf{w}|| = \sqrt{\mathbf{w}^T \mathbf{w}}$.
- ▶ Now we have the distance from the origin to the hyperplane!

Computing the Distance to Hyperplane



- ▶ Now we want c , the distance from \mathbf{x} to the hyperplane.
- ▶ It's clear that $c = |b - a|$, where a the length of the projection of \mathbf{x} onto \mathbf{w} . Quiz: What is a ?

Computing the Distance to Hyperplane



- ▶ Now we want c , the distance from \mathbf{x} to the hyperplane.
- ▶ It's clear that $c = |b - a|$, where a the length of the projection of \mathbf{x} onto \mathbf{w} . Quiz: What is a ?

$$a = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$$

Equation for the Margin

- ▶ The perpendicular distance from a point \mathbf{x} to the hyperplane $\mathbf{w}^T \mathbf{x} + w_0 = 0$ is

$$\frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x} + w_0|$$

- ▶ The **margin** is the distance from the closest training point to the hyperplane

$$\min_i \frac{1}{\|\mathbf{w}\|} |\mathbf{w}^T \mathbf{x}_i + w_0|$$

The Scaling

- ▶ Note that (\mathbf{w}, w_0) and $(c\mathbf{w}, cw_0)$ defines the same hyperplane. The scale is arbitrary.
- ▶ This is because we predict class $y = 1$ if $\mathbf{w}^T \mathbf{x} + w_0 \geq 0$. That's the same thing as saying $c\mathbf{w}^T \mathbf{x} + cw_0 \geq 0$
- ▶ To remove this freedom, we will put a constraint on (\mathbf{w}, w_0)

$$\min_i |\mathbf{w}^T \mathbf{x}_i + w_0| = 1$$

- ▶ With this constraint, the margin is always $1/\|\mathbf{w}\|$.

First version of Max Margin Optimization Problem

- Here is a first version of an optimization problem to maximize the margin (we will simplify)

$$\max_{\mathbf{w}} 1/\|\mathbf{w}\|$$

$$\text{subject to } \mathbf{w}^\top \mathbf{x}_i + w_0 \geq 0 \quad \text{for all } i \text{ with } y_i = 1$$

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \leq 0 \quad \text{for all } i \text{ with } y_i = -1$$

$$\min_i |\mathbf{w}^\top \mathbf{x}_i + w_0| = 1$$

- The first two constraints are too lose. It's the same thing to say

$$\max_{\mathbf{w}} 1/\|\mathbf{w}\|$$

$$\text{subject to } \mathbf{w}^\top \mathbf{x}_i + w_0 \geq 1 \quad \text{for all } i \text{ with } y_i = 1$$

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \leq -1 \quad \text{for all } i \text{ with } y_i = -1$$

$$\min_i |\mathbf{w}^\top \mathbf{x}_i + w_0| = 1$$

- Now the third constraint is redundant

First version of Max Margin Optimization Problem

- ▶ That means we can simplify to

$$\max_{\mathbf{w}} 1/\|\mathbf{w}\|$$

$$\text{subject to } \mathbf{w}^\top \mathbf{x}_i + w_0 \geq 1 \quad \text{for all } i \text{ with } y_i = 1$$

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \leq -1 \quad \text{for all } i \text{ with } y_i = -1$$

- ▶ Here's a compact way to write those two constraints

$$\max_{\mathbf{w}} 1/\|\mathbf{w}\|$$

$$\text{subject to } y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq 1 \quad \text{for all } i$$

- ▶ Finally, note that maximizing $1/\|\mathbf{w}\|$ is the same thing as *minimizing* $\|\mathbf{w}\|^2$

The SVM optimization problem

- ▶ So the SVM weights are determined by solving the optimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & ||\mathbf{w}||^2 \\ \text{s.t. } & y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq +1 \quad \text{for all } i \end{aligned}$$

- ▶ Solving this will require maths that we don't have in this course. But I'll show the form of the solution next time.

Fin (Part I)

Optimising the Margin

Nigel Goddard
School of Informatics

The SVM optimization problem

- ▶ Last time: the max margin weights can be computed by solving a constrained optimization problem

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2$$

$$\text{s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq +1 \quad \text{for all } i$$

- ▶ Many algorithms have been proposed to solve this. One of the earliest efficient algorithms is called SMO [Platt, 1998]. This is outside the scope of the course, but it does explain the name of the SVM method in Weka.

Finding the optimum

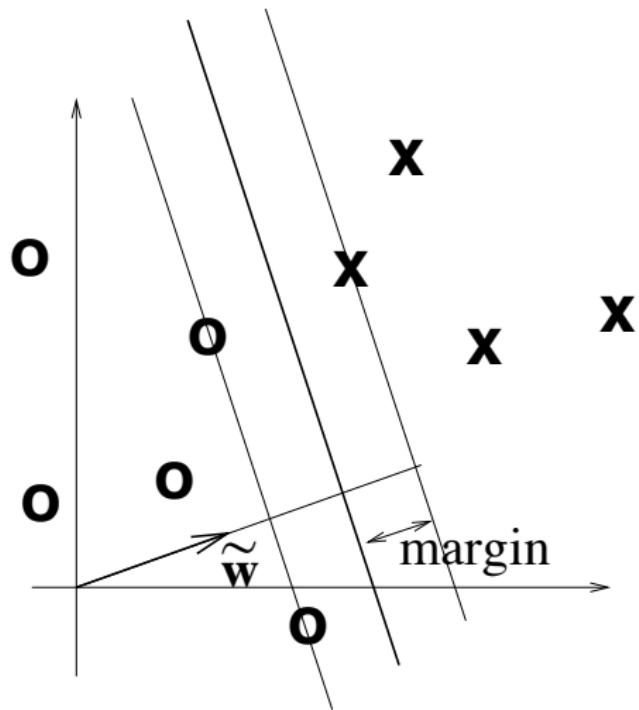
- ▶ If you go through some advanced maths (Lagrange multipliers, etc.), it turns out that you can show something remarkable. Optimal parameters look like

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- ▶ Furthermore, solution is sparse. Optimal hyperplane is determined by just a few examples: call these *support vectors*

Why a solution of this form?

If you move the points not on the marginal hyperplanes, solution doesn't change - therefore those points don't matter.



Finding the optimum

- ▶ If you go through some advanced maths (Lagrange multipliers, etc.), it turns out that you can show something remarkable. Optimal parameters look like

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- ▶ Furthermore, solution is sparse. Optimal hyperplane is determined by just a few examples: call these *support vectors*
- ▶ $\alpha_i = 0$ for non-support patterns
- ▶ Optimization problem to find α_i has no local minima (like logistic regression)
- ▶ Prediction on new data point \mathbf{x}

$$\begin{aligned} f(\mathbf{x}) &= \text{sign}((\mathbf{w}^\top \mathbf{x}) + w_0) \\ &= \text{sign}\left(\sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + w_0\right) \end{aligned}$$

Non-Separable Data

Nigel Goddard
School of Informatics

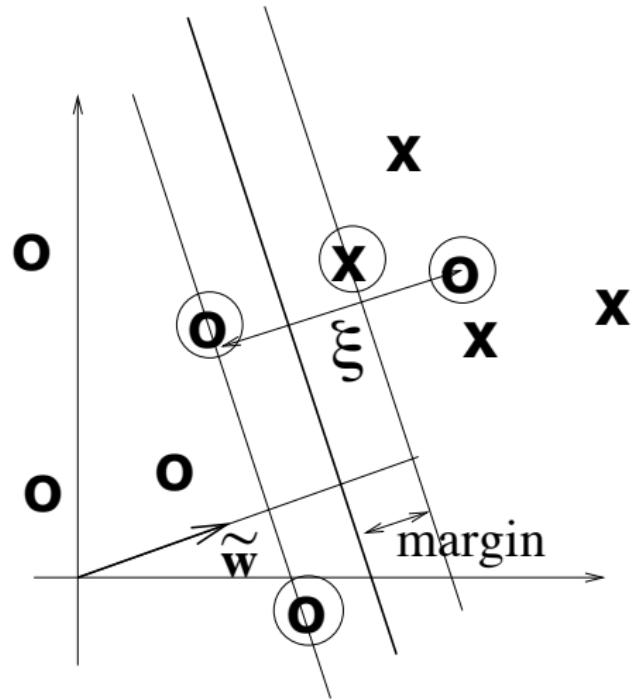
Non-separable training sets

- If data set is not linearly separable, the optimization problem that we have given has *no solution*.

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2$$

$$\text{s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i + w_0) \geq +1 \quad \text{for all } i$$

- Why?
- Solution: Don't require that we classify all points correctly.
Allow the algorithm to choose to ignore some of the points.
- This is obviously dangerous (why not ignore all of them?)
so we need to give it a penalty for doing so.



- ▶ Solution: Add a “slack” variable $\xi_i \geq 0$ for each training example.
- ▶ If the slack variable is high, we get to relax the constraint, but we pay a price
- ▶ New optimization problem is to minimize

$$||\mathbf{w}||^2 + C(\sum_{i=1}^n \xi_i^k)$$

subject to the constraints

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \geq 1 - \xi_i \quad \text{for } y_i = +1$$

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \leq -1 + \xi_i \quad \text{for } y_i = -1$$

- ▶ Usually set $k = 1$. C is a trade-off parameter. Large C gives a large penalty to errors.
- ▶ Solution has same form, but support vectors also include all where $\xi_i \neq 0$. Why?

Finding the optimum

- ▶ If you go through some advanced maths (Lagrange multipliers, etc.), it turns out that you can show something remarkable. Optimal parameters look like

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- ▶ Furthermore, solution is sparse. Optimal hyperplane is determined by just a few examples: call these *support vectors*
- ▶ $\alpha_i = 0$ for non-support patterns
- ▶ Optimization problem to find α_i has no local minima (like logistic regression)
- ▶ Prediction on new data point \mathbf{x}

$$\begin{aligned} f(\mathbf{x}) &= \text{sign}((\mathbf{w}^\top \mathbf{x}) + w_0) \\ &= \text{sign}\left(\sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + w_0\right) \end{aligned}$$

Regularisation

Nigel Goddard
School of Informatics

Think about ridge regression again

- ▶ Our max margin + slack optimization problem is to minimize:

$$\|\mathbf{w}\|^2 + C \left(\sum_{i=1}^n \xi_i^k \right)$$

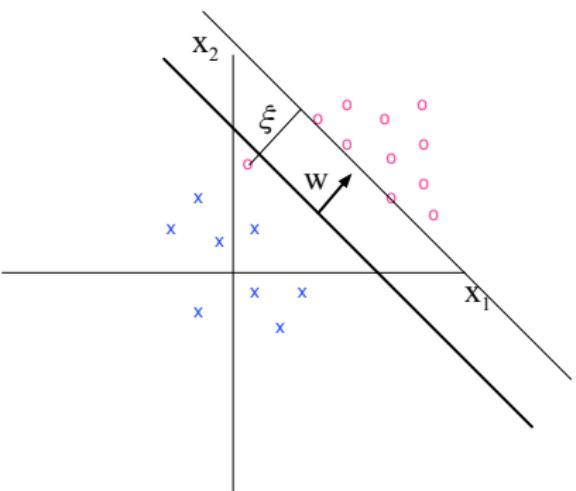
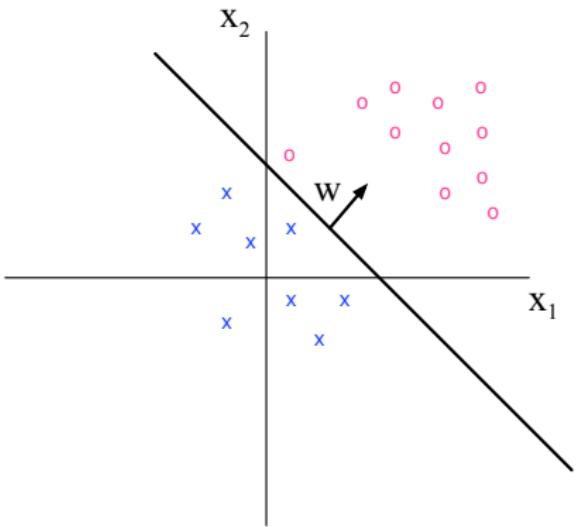
subject to the constraints

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \geq 1 - \xi_i \quad \text{for } y_i = +1$$

$$\mathbf{w}^\top \mathbf{x}_i + w_0 \leq -1 + \xi_i \quad \text{for } y_i = -1$$

- ▶ This looks even more like ridge regression than the non-slack problem:
 - ▶ $C(\sum_{i=1}^n \xi_i)^k$ measures how well we fit the data
 - ▶ $\|\mathbf{w}\|^2$ penalizes weight vectors with a large norm
- ▶ So C can be viewed as a regularization parameter, like λ in ridge regression or regularized logistic regression
- ▶ You're allowed to make this tradeoff even when the data set is separable!

Why you might want slack in a separable data set



Comparing the SVM and Logistic Regression Optimization Problems

Chris Williams

School of Informatics, University of Edinburgh

SVM Optimization Problem

- ▶ Minimize (over \mathbf{w} and $\xi_i \in \mathbb{R}^+ \forall i$)

$$||\mathbf{w}||^2 + C \sum_i \xi_i$$

subject to

$$y_i f(\mathbf{x}_i) \geq 1 - \xi_i, \quad i = 1, \dots, n,$$

where $\xi_i \geq 0$ for all i , and $f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + w_0$

- ▶ This optimization problem can be re-written as

$$\min_{\mathbf{w} \in \mathbb{R}^d} ||\mathbf{w}||^2 + C \sum_i \max(0, 1 - y_i f(\mathbf{x}_i))$$

(To see this check the three conditions where $y_i f(\mathbf{x}_i) > 1$,
 $y_i f(\mathbf{x}_i) = 1$ and $y_i f(\mathbf{x}_i) < 1$.)

- ▶ The function

$$g_{\text{hinge}}(z) = \max(0, 1 - z)$$

is known as the *hinge loss*

Logistic Regression with a ridge penalty

Let y labels take on values 1 or -1.

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-f(\mathbf{x})}}$$

$$p(y = -1|\mathbf{x}) = 1 - \frac{1}{1 + e^{-f(\mathbf{x})}} = \frac{1}{1 + e^{f(\mathbf{x})}}$$

Combining these, we have

$$p(y_i|\mathbf{x}_i) = \frac{1}{1 + e^{-y_i f(\mathbf{x}_i)}}$$

Hence the log loss is given by

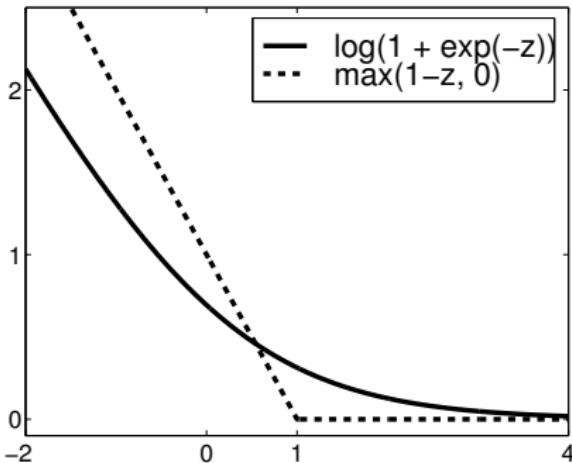
$$-\log p(y_i|\mathbf{x}_i) = \log(1 + e^{-y_i f(\mathbf{x}_i)})$$

- ▶ Incorporating a ridge regression penalty, optimization problem is

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i \log(1 + e^{-y_i f(\mathbf{x}_i)})$$

where $C = 1/\lambda$

- ▶ Define $g_\sigma(z) = \log(1 + e^{-z})$
- ▶ SVM classifier has a similar optimization problem, but with $g_{hinge}(z)$



- ▶ Both optimization problems are convex
- ▶ Note that the MAP solution using g_σ solution is not sparse, but that logistic regression gives a probability output

Non-Linear SVMs

Nigel Goddard
School of Informatics

Non-linear SVMs

- ▶ SVMs can be made nonlinear just like any other linear algorithm we've seen (i.e., using a basis expansion)
- ▶ But in an SVM, the basis expansion is implemented in a very special way, using something called a *kernel*
- ▶ The reason for this is that kernels can be faster to compute with if the expanded feature space is very high dimensional (even infinite)!
- ▶ This is a fairly advanced topic mathematically, so we will just go through a high-level version

- ▶ A kernel is in some sense an alternate “API” for specifying to the classifier what your expanded feature space is.
- ▶ Up to now, we have always given the classifier a new set of training vectors $\phi(\mathbf{x}_i)$ for all i , e.g., just as a list of numbers.
$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$$
- ▶ If D is large, this will be expensive; if D is infinite, this will be impossible

Non-linear SVMs

- ▶ Transform \mathbf{x} to $\phi(\mathbf{x})$
- ▶ Linear algorithm depends only on $\mathbf{x}^\top \mathbf{x}_i$. Hence transformed algorithm depends only on $\phi(\mathbf{x})^\top \phi(\mathbf{x}_i)$
- ▶ Use a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ such that

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- ▶ (This is called the “kernel trick”, and can be used with a wide variety of learning algorithms, not just max margin.)

Finding the optimum

- ▶ If you go through some advanced maths (Lagrange multipliers, etc.), it turns out that you can show something remarkable. Optimal parameters look like

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- ▶ Furthermore, solution is sparse. Optimal hyperplane is determined by just a few examples: call these *support vectors*
- ▶ $\alpha_i = 0$ for non-support patterns
- ▶ Optimization problem to find α_i has no local minima (like logistic regression)
- ▶ Prediction on new data point \mathbf{x}

$$\begin{aligned} f(\mathbf{x}) &= \text{sign}((\mathbf{w}^\top \mathbf{x}) + w_0) \\ &= \text{sign}\left(\sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + w_0\right) \end{aligned}$$

Kernals for SVMs

Nigel Goddard
School of Informatics

Example of kernel

- ▶ Example 1: for 2-d input space

$$\phi(\mathbf{x}_i) = \begin{pmatrix} x_{i,1}^2 \\ \sqrt{2}x_{i,1}x_{i,2} \\ x_{i,2}^2 \end{pmatrix}$$

then

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j)^2$$

Non-linear SVMs

- ▶ Transform \mathbf{x} to $\phi(\mathbf{x})$
- ▶ Linear algorithm depends only on $\mathbf{x}^\top \mathbf{x}_i$. Hence transformed algorithm depends only on $\phi(\mathbf{x})^\top \phi(\mathbf{x}_i)$
- ▶ Use a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ such that

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- ▶ (This is called the “kernel trick”, and can be used with a wide variety of learning algorithms, not just max margin.)

Kernels, dot products, and distance

- ▶ The Euclidean distance squared between two vectors can be computed using dot products

$$\begin{aligned}d(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x}_1 - \mathbf{x}_2)^T (\mathbf{x}_1 - \mathbf{x}_2) \\&= \mathbf{x}_1^T \mathbf{x}_1 - 2\mathbf{x}_1^T \mathbf{x}_2 + \mathbf{x}_2^T \mathbf{x}_2\end{aligned}$$

- ▶ Using a linear kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_2$ we can rewrite this as

$$d(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_1) - 2k(\mathbf{x}_1, \mathbf{x}_2) + k(\mathbf{x}_2, \mathbf{x}_2)$$

- ▶ Any kernel gives you an associated distance measure this way. Think of a kernel as an indirect way of specifying distances.

Kernelised Max Margin

Nigel Goddard
School of Informatics

Support Vector Machine

- ▶ A **support vector machine** is a kernelized maximum margin classifier.
- ▶ For max margin remember that we had the magic property

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

- ▶ This means we would predict the label of a test example \mathbf{x} as

$$\hat{y} = \text{sign}[\mathbf{w}^T \mathbf{x} + w_0] = \text{sign}\left[\sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + w_0\right]$$

- ▶ Kernelizing this we get

$$\hat{y} = \text{sign}\left[\sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b\right]$$

Prediction on new example

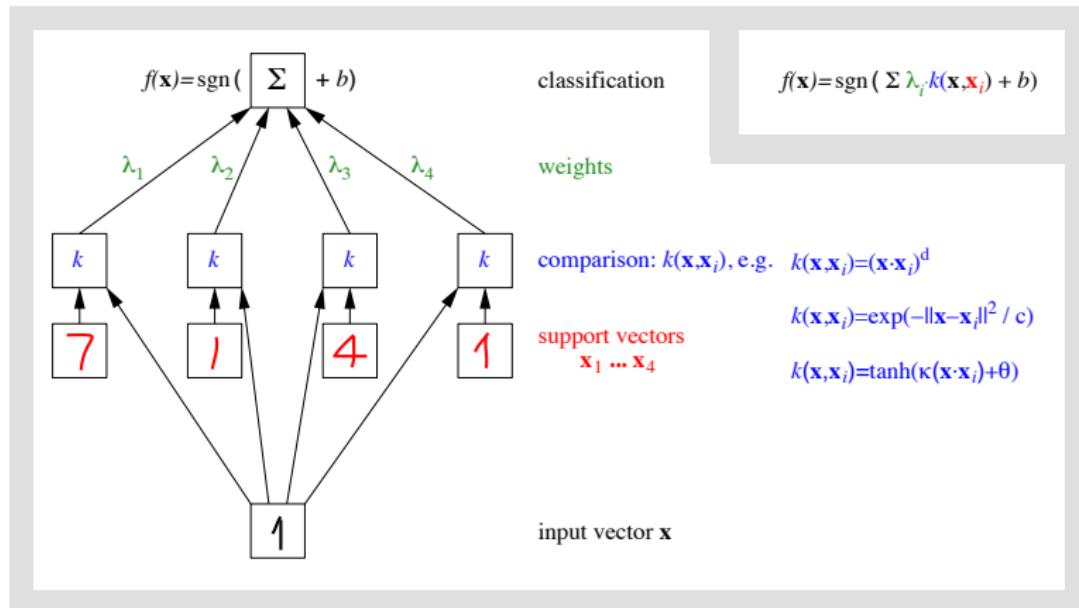


Figure Credit: Bernhard Schoelkopf

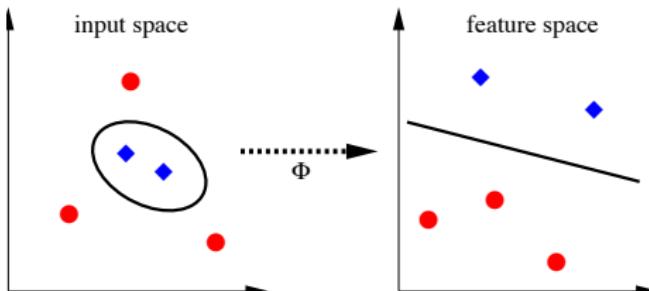


Figure Credit: Bernhard Schoelkopf

► Example 2

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp -||\mathbf{x}_i - \mathbf{x}_j||^2/\alpha^2$$

In this case the dimension of ϕ is infinite. i.e., It can be shown that no ϕ that maps into a finite-dimensional space will give you this kernel.

- We can *never* calculate $\phi(\mathbf{x})$, but the algorithm only needs us to calculate k for different pairs of points.

SVMs in Practice

Nigel Goddard
School of Informatics

Choosing ϕ , C

- ▶ There are theoretical results, but we will not cover them. (If you want to look them up, there are actually upper bounds on the generalization error: look for VC-dimension and structural risk minimization.)
- ▶ However, in practice cross-validation methods are commonly used

Example application

- ▶ US Postal Service digit data (7291 examples, 16×16 images). Three SVMs using polynomial, RBF and MLP-type kernels were used (see Schölkopf and Smola, *Learning with Kernels*, 2002 for details)
- ▶ Use almost the same ($\simeq 90\%$) small sets (4% of data base) of SVs
- ▶ All systems perform well ($\simeq 4\%$ error)
- ▶ Many other applications, e.g.
 - ▶ Text categorization
 - ▶ Face detection
 - ▶ DNA analysis

SVM Summary and Comparison

Nigel Goddard
School of Informatics

Comparison with linear and logistic regression

- ▶ Underlying basic idea of linear prediction is the same, but error functions differ
- ▶ Logistic regression (non-sparse) vs SVM (“hinge loss”, sparse solution)
- ▶ Linear regression (squared error) vs ϵ -insensitive error
- ▶ Linear regression and logistic regression can be “kernelized” too

SVM summary

- ▶ SVMs are the combination of max-margin and the kernel trick
- ▶ Learn linear decision boundaries (like logistic regression, perceptrons)
 - ▶ Pick hyperplane that maximizes margin
 - ▶ Use slack variables to deal with non-separable data
 - ▶ Optimal hyperplane can be written in terms of support patterns
- ▶ Transform to higher-dimensional space using kernel functions
- ▶ Good empirical results on many problems
- ▶ Appears to avoid overfitting in high dimensional spaces (cf regularization)



Unless explicitly stated otherwise, all material is
copyright ©The University of Edinburgh 2016.

Ethics and Machine Learning

Chris Williams

School of Informatics, University of Edinburgh, UK

- ▶ Machine learning and AI technologies are increasingly in use for problems that affect people
- ▶ When I started ML research, handwritten digit recognition was a big topic
- ▶ Now we see applications in healthcare, law enforcement, smart cities, retail, social media etc which may have direct effects on people's lives

Some examples

- ▶ Advances in precision medicine allowing e.g. cancer treatments tailored to individuals
- ▶ The use of predictive models to assist judges with sentencing decisions (e.g. Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) system)
- ▶ Advance warning of serious flooding
- ▶ Live facial recognition (LFR) in public spaces



Figure credits www.cancer.gov, www.gov.uk

Outline

1. Ethical benefits and harms linked to data practices
2. Ethical Issues in Machine Learning
 - ▶ Fairness
 - ▶ Accountability
 - ▶ Transparency

Ethical benefits linked to data practices

- ▶ Increasing human understanding of nature, society and our personal lives
 - ▶ The more we understand about the world and how it works, the more intelligently we can act in it
- ▶ Social, institutional and economic efficiency
 - ▶ Reduce wasted effort and resources, and improve alignment between a social system or institution's policies/processes and our goals
- ▶ Predictive accuracy and personalization
 - ▶ More precisely tailor actions to be effective in achieving good outcomes for specific individuals and groups
 - ▶ Potential to remove human biases from decision making

See *An Introduction to Data Ethics* by Shannon Vallor pp 9-10

Ethical harms linked to data practices

- ▶ Harms to fairness and justice
 - ▶ Arbitrariness; avoidable errors and inaccuracies; unjust and often hidden biases
- ▶ Harms to transparency and accountability
 - ▶ Can you explain how the system works, and who is responsible for it?
- ▶ Harms to privacy and security
 - ▶ E.g. your personal data may be part of data profiles constructed and stored somewhere unknown to you, often without your knowledge or informed consent

See *An Introduction to Data Ethics* by Shannon Vallor pp 10-13

Common ethical challenges for data practitioners and users

- ▶ Appropriate data collection and use
- ▶ Data storage, security and responsible data stewardship
- ▶ Data cleanliness and data relevance
- ▶ Identifying and addressing ethically harmful data bias
- ▶ Validation and testing of data models and analytics
- ▶ Human accountability in data practices and systems
- ▶ Understanding personal, social, and business impacts of data practice

See *An Introduction to Data Ethics* by Shannon Vallor p 17-21

Is technology neutral?

- ▶ Please PAUSE the video here and think about this question before moving on to the next slide

Is technology neutral?

- ▶ Technologies are not ethically neutral, they reflect the values that we “bake in” to them with our design choices, as well as the values which guide our use of them

See *An Introduction to Data Ethics* by Shannon Vallor p 3

Questions to ask about your technology

Ethics is not a checklist. It is an ongoing conversation, and requires you to question possible outcomes

- ▶ Who are the stakeholders? This includes anyone who funds, develops, or uses your technology, and anyone it is used upon.
- ▶ Who benefits from the technology? How?
- ▶ Who could be harmed by the technology? How?

These are questions you must ask *yourself and all of the stakeholders*

2. Ethical Issues in Machine Learning: FAT

- ▶ Fairness
 - ▶ Data fairness, outcome fairness
- ▶ Accountability
- ▶ Transparency
 - ▶ Process transparency, outcome transparency

Defining Fairness

- ▶ In a legal context fairness means that people are not discriminated against based on their membership of a *protected* group or class, such as race, gender, sexual orientation etc
- ▶ Avoiding *disparate impact*: where a decision disproportionately harms people with certain protected attributes

Data Fairness: Sampling bias in data collection

- ▶ Buolamwini and Gebru (2018) evaluated bias in automated facial analysis algorithms and datasets with respect to phenotypic subgroups
- ▶ Standard benchmark datasets were overwhelmingly composed of lighter-skinned subjects
- ▶ They constructed a new facial analysis dataset (Pilot Parliaments Benchmark) balanced by gender and skin type

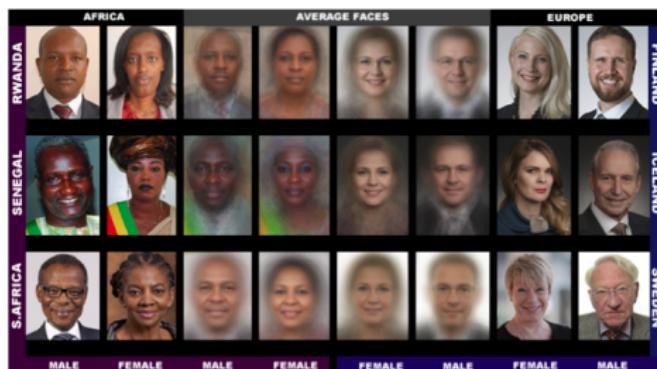


Figure Credit: Buolamwini and Gebru (2018) Fig 1

- ▶ Three commercial gender classification systems evaluated on the new dataset
- ▶ Darker-skinned females are the most misclassified group (error rates up to 34.7%)
- ▶ The maximum error rate for lighter-skinned males was 0.8%
- ▶ They conclude that “the substantial disparities in the accuracy of classifying darker females, lighter females, darker males, and lighter males in gender classification systems require urgent attention if commercial companies are to build genuinely fair, transparent and accountable facial analysis algorithms”
- ▶ Who is harmed by poor facial recognition? Note that facial recognition technology is marketed to police departments as a tool to aid law enforcement in identifying criminal suspects

Data fairness: Bias in data collection

- ▶ If a group of factory workers is more stringently or frequently monitored, more errors will be observed in that group
- ▶ This can lead to a feedback loop whereby the group is subject to further monitoring because of the apparent higher rate of mistakes
- ▶ Danger of this bias in predictive policing, where minority communities are often more highly policed

Data Fairness: Bias in Labelling

- ▶ Supervised learning requires labels
- ▶ Example: predict who is a “good worker” based on CVs
- ▶ If we train such a system on the existing workforce, and there were biases in their hiring, then these will be perpetuated by the trained system
- ▶ E.g. Amazon developed such an experimental system, but identified the problems and abandoned its use

Reference: <https://slate.com/business/2018/10/amazon-artificial-intelligence-hiring-discrimination-women.html>

Outcome fairness

Example: predicting loan repayments

Customer	Income	Credit Score	Gender	Repayment
A	10,000	1	Female	0
B	20,000	2	Female	1
C	30,000	3	Male	1
D	30,000	4	Female	0
E	40,000	4	Male	1
:	:	:	:	:

- ▶ This data is used to train a model to predict loan repayment based on the three attributes
- ▶ Here gender is a *protected attribute*

Example based on Data Ethics, AI and Responsible Innovation course, week 2

Fairness through Unawareness?

- ▶ Can we simply omit the protected attribute from our predictor to ensure fairness?
- ▶ Please PAUSE the video here and think about this question before moving on to the next slide

- ▶ NO, this is not sufficient; there may be other attributes (known as *proxies*) correlated with a protected attribute
- ▶ E.g. the credit score may be correlated with gender, so we can predict gender from other (unprotected) features

Assessing fairness

- ▶ Compare ROC curves for each protected group – do they differ?
- ▶ There are several definitions of statistical (or group) fairness, with names like demographic parity, equal opportunity etc
- ▶ These consider different aspects of the TP/FP/TN/FN distribution, conditioned on the protected attribute(s)
- ▶ Different definitions of fairness can be mutually inconsistent, so need to assess which of them is most suitable for a given use case
- ▶ Imposing fairness constraints can lead to a drop in predictive accuracy

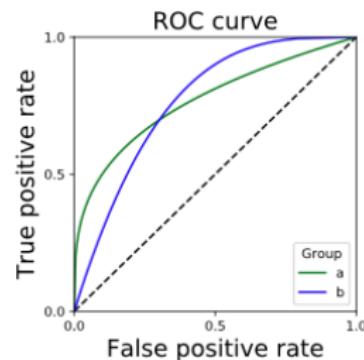


Figure credit: <https://fairmlbook.org, ch2>

Accountability

- ▶ Human agents can be called upon to account for their judgements and decisions
- ▶ *Accountability gap*: Statistical models are not responsible in the same morally relevant sense
- ▶ Requires clear human **answerability** can be attached to decisions assisted or produced by an AI system (*who* is accountable?)
- ▶ **Auditability**: *how* are the designers and implementers of AI systems are to be held accountable?
- ▶ *Document* every step of the process of designing and implementing the project
- ▶ *Document* the monitoring of data provenance and analysis from collection, pre-processing, and modelling to training, testing, and deployment
- ▶ Examples: self-driving cars, medical decision support

See: D Leslie *Understanding artificial intelligence ethics and safety*, pp 23-24

Transparency

- ▶ **Outcome transparency:** The ability to know how and why a model performed the way it did in a specific context and therefore to understand the rationale behind its decision or behaviour (opening the “black box”)
- ▶ Note: we may not always know why humans made a decision (and sometimes they make *post hoc* rationalizations)
- ▶ **Process transparency:** The justifiability both of the processes that go into its design and implementation and of a system's outcomes

See: D Leslie *Understanding artificial intelligence ethics and safety*, pp 35-36

A question for you to think about

- ▶ You have a medical problem, and there are two decision support tools available to your doctor
- ▶ One is a simple and interpretable method, but its overall AUROC is 0.85
- ▶ The other is a deep neural network which no-one understands fully how it works, but its overall AUROC is 0.95
- ▶ Which would you like your doctor to use, and why?

Summary: Ethics in ML

- ▶ General ethical concerns in data practice
- ▶ Fairness
 - ▶ Data fairness, outcome fairness,
- ▶ Accountability
- ▶ Transparency
 - ▶ Process transparency, outcome transparency

References

- ▶ Shannon Vallor, *An Introduction to Data Ethics*
[https://www.scu.edu/media/ethics-center/
technology-ethics/IntroToDataEthics.pdf](https://www.scu.edu/media/ethics-center/technology-ethics/IntroToDataEthics.pdf)
- ▶ David Leslie, *Understanding artificial intelligence ethics and safety*, the Alan Turing Institute (2019)
<https://doi.org/10.5281/zenodo.3240529>
- ▶ Joy Buolamwini and Timnit Gebru, *Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification*, Proc. Machine Learning Research 81 1-15, (2018)

Introductory Applied Machine Learning

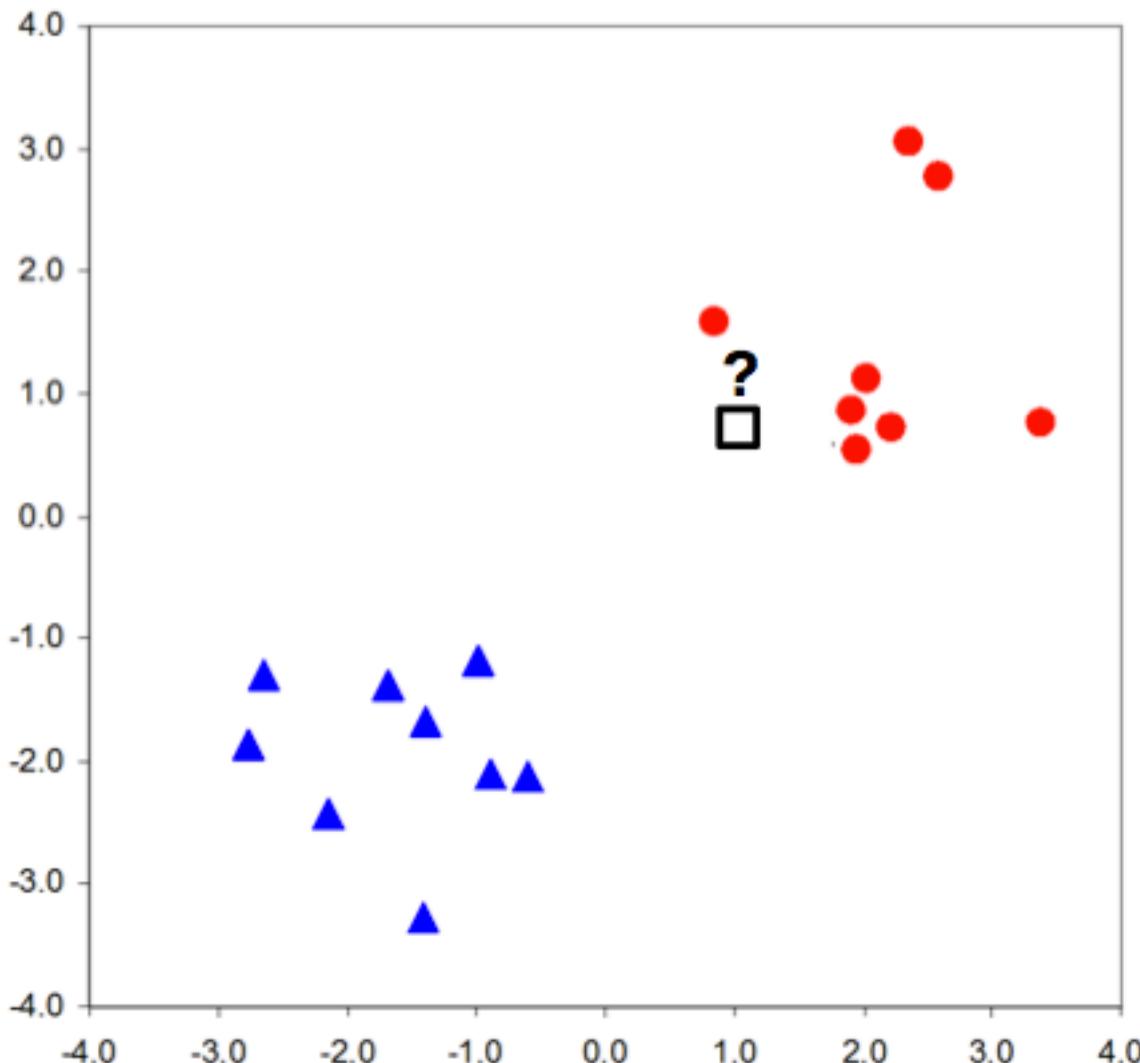
Nearest Neighbour Methods

Victor Lavrenko and Nigel Goddard
School of Informatics

Overview

- Nearest neighbour method
 - classification and regression
 - practical issues: k, distance, ties, missing values
 - optimality and assumptions
- Making kNN fast:
 - K-D trees
 - inverted indices
 - fingerprinting
- References: W&F sections 4.7 and 6.4

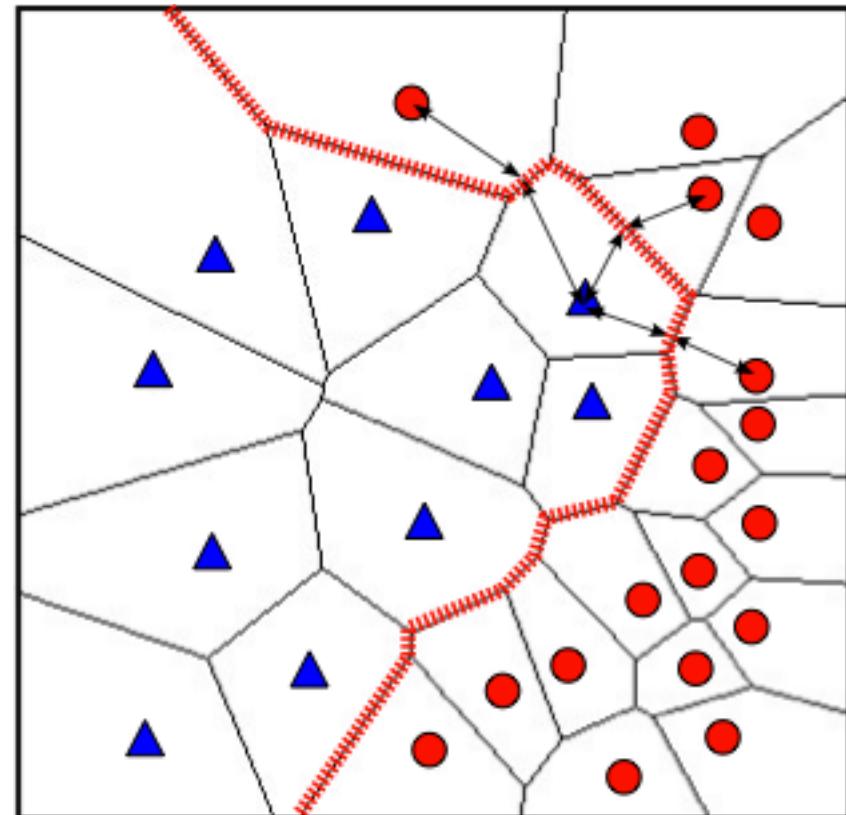
Intuition for kNN



- set of points (x,y)
 - two classes
- is the box red or blue
- how did you do it
 - use Bayes rule?
 - a decision tree?
 - fit a hyperplane?
- nearby points are red
 - use this as a basis for a learning algorithm

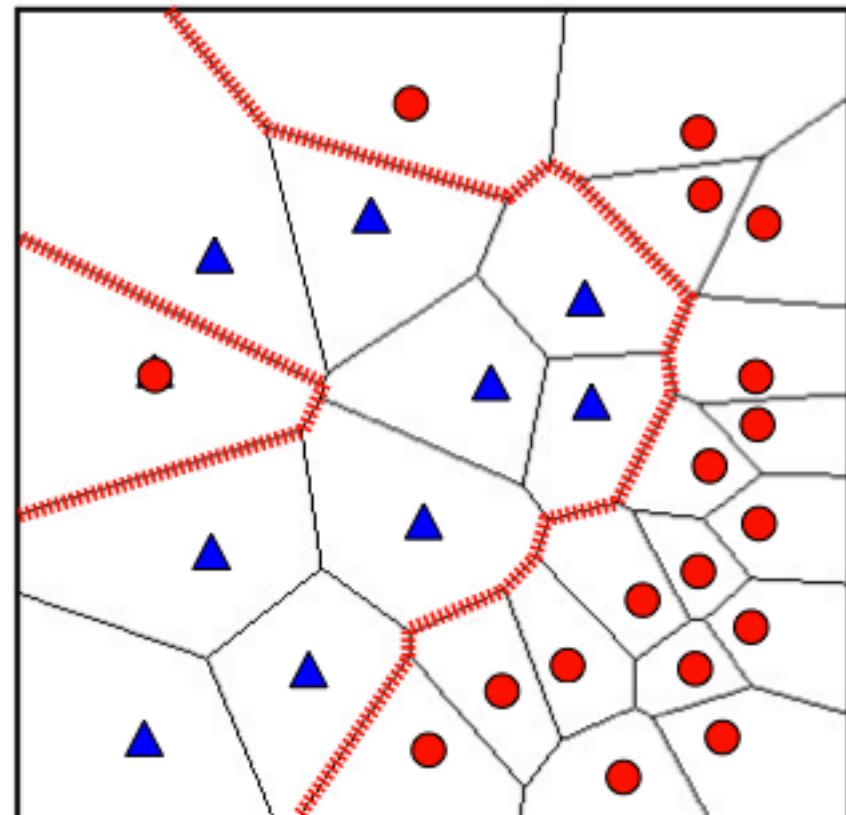
Nearest-neighbor classification

- Use the intuition to classify a new point x :
 - find the most similar training example x'
 - predict its class y'
- Voronoi tessellation
 - partitions space into regions
 - boundary: points at same distance from two different training examples
- decision boundary
 - non-linear, reflects classes well
 - compare to NB, DT, logistic
 - impressive for simple method



Nearest neighbour: outliers

- Algorithm is sensitive to outliers
 - single mislabeled example dramatically changes boundary
- No confidence $P(y|x)$
- Insensitive to class prior
- Idea:
 - use more than one nearest neighbor to make decision
 - count class labels in k most similar training examples
 - many “triangles” will outweigh single “circle” outlier

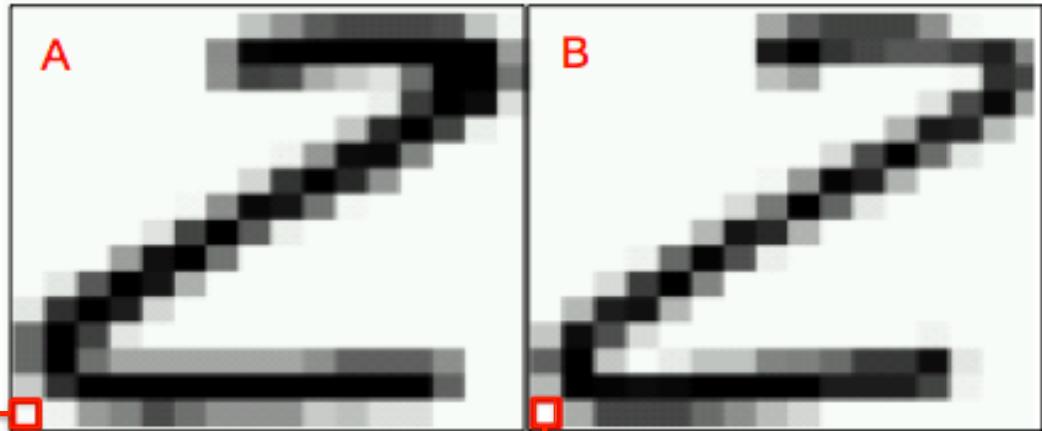


kNN classification algorithm

- Given:
 - training examples $\{x_i, y_i\}$
 - x_i ... attribute-value representation of examples
 - y_i ... class label: {ham,spam}, digit {0,1,...9} etc.
 - testing point x that we want to classify
- Algorithm:
 - compute distance $D(x, x_i)$ to every training example x_i
 - select k closest instances $x_{i1} \dots x_{ik}$ and their labels $y_{i1} \dots y_{ik}$
 - output the class y^* which is most frequent in $y_{i1} \dots y_{ik}$

Example: handwritten digits

- 16x16 bitmaps
- 8-bit grayscale
- Euclidian distance
 - over raw pixels
- Accuracy:
 - 7-NN ~ 95.2%
 - SVM ~ 95.8%
 - humans ~ 97.5%



$$D(A, B) = \sqrt{\sum_r \sum_c (A_{r,c} - B_{r,c})^2}$$

example

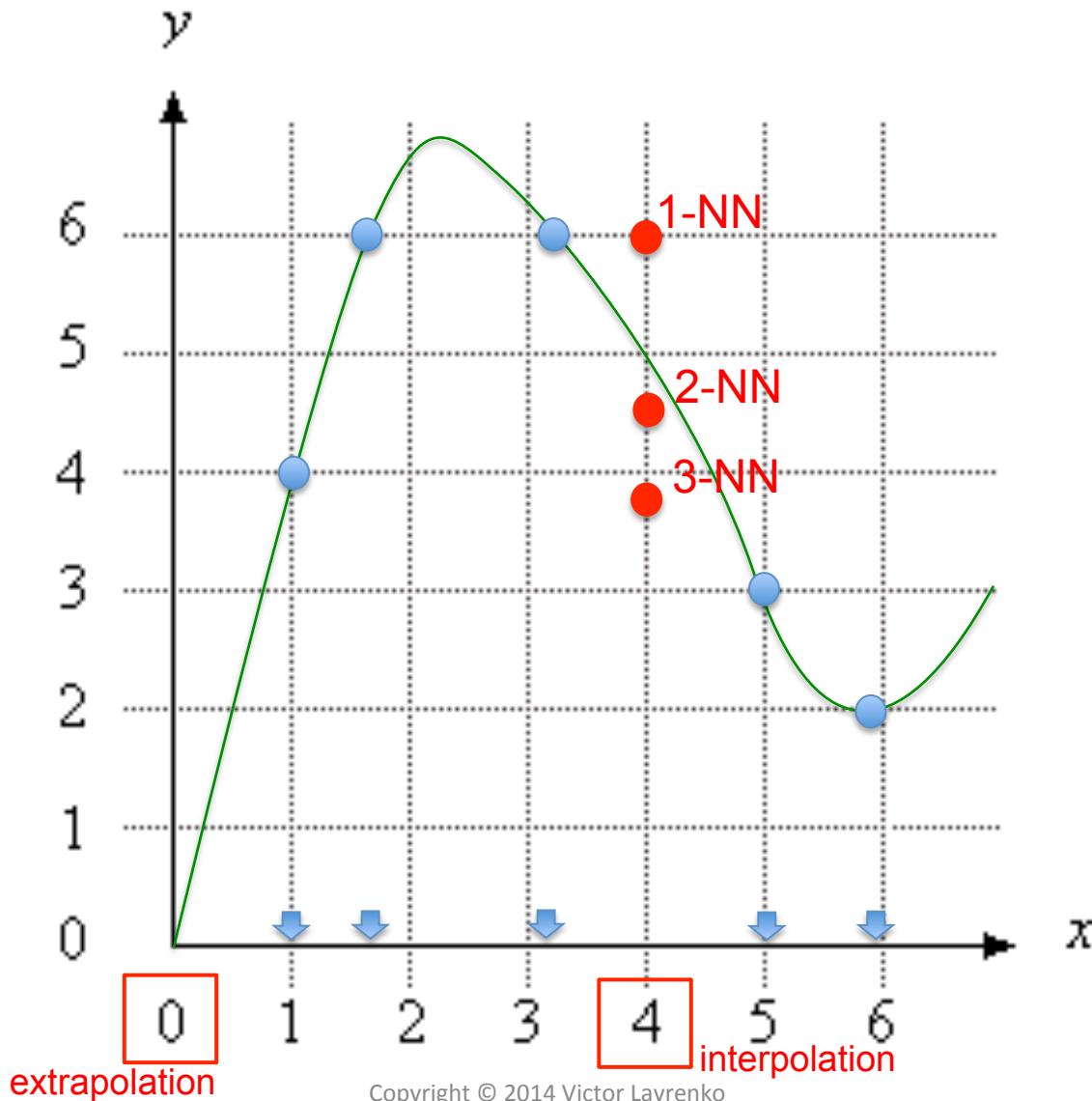


kNN regression algorithm

- Given:
 - training examples $\{x_i, y_i\}$
 - x_i ... attribute-value representation of examples
 - y_i ... real-valued target (profit, rating on YouTube, etc)
 - testing point x that we want to predict the target
- Algorithm:
 - compute distance $D(x, x_i)$ to every training example x_i
 - select k closest instances $x_{i1} \dots x_{ik}$ and their labels $y_{i1} \dots y_{ik}$
 - output the mean of $y_{i1} \dots y_{ik}$:

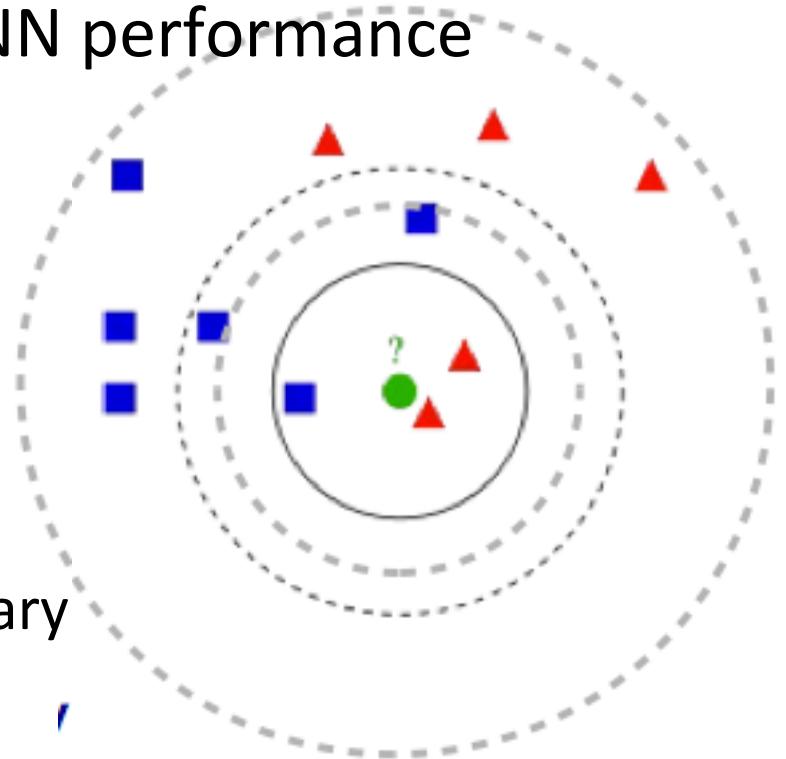
$$\hat{y} = f(x) = \frac{1}{k} \sum_{j=1}^k y_{ij}$$

Example: kNN regression in 1-d



Choosing the value of k

- Value of k has strong effect on kNN performance
 - large value → everything classified as the most probable class: $P(y)$
 - small value → highly variable, unstable decision boundaries
 - small changes to training set → large changes in classification
 - affects “smoothness” of the boundary
- Selecting the value of k
 - set aside a portion of the training data (validation set)
 - vary k, observe training → validation error
 - pick k that gives best generalization performance

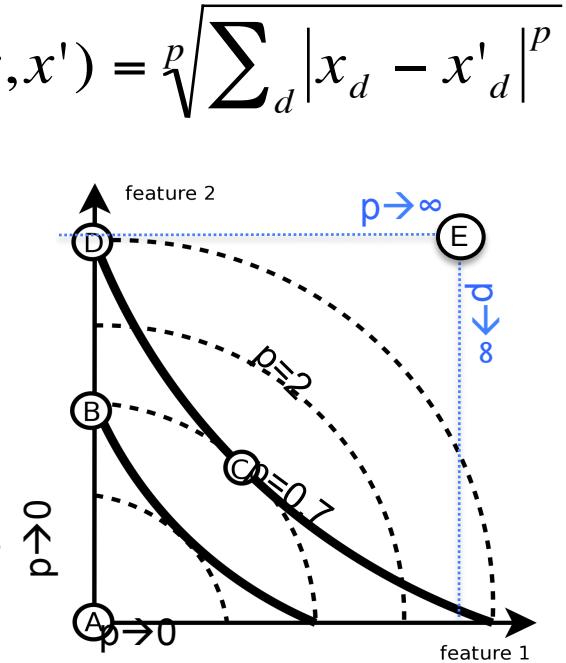
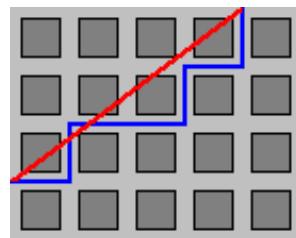


Distance measures

- Key component of the kNN algorithm
 - defines which examples are similar & which aren't
 - can have strong effect on performance
- Euclidean (numeric attributes): $D(x, x') = \sqrt{\sum_d |x_d - x'_d|^2}$
 - symmetric, spherical, treats all dimensions equally
 - sensitive to extreme differences in single attribute
- Hamming (categorical attributes): $D(x, x') = \sum_d 1_{x_d \neq x'_d}$
 - number of attributes where x, x' differ

Distance measures (2)

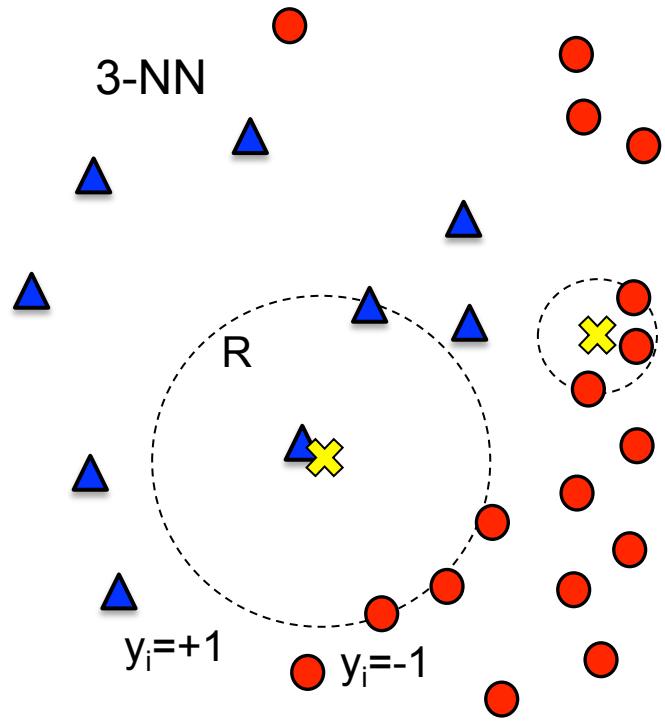
- Minkowski distance (p -norm): $D(x, x') = \sqrt[p]{\sum_d |x_d - x'_d|^p}$
 - $p = 2$: Euclidean
 - $p = 1$: Manhattan
 - $p \rightarrow \infty$: $\max_d |x_d - x'_d|$
 - $p \rightarrow 0$: number of non-zero differences
- Custom distance measures (e.g. for text, or distributions)



kNN: practical issues

- Resolving ties:
 - equal number of positive/negative neighbours
 - use odd k (doesn't solve multi-class)
 - breaking ties:
 - random: flip a coin to decide positive / negative
 - prior: pick class with greater prior
 - nearest: use 1-nn classifier to decide
- Missing values
 - have to “fill in”, otherwise can't compute distance
 - key concern: should affect distance as little as possible
 - reasonable choice: average value across entire dataset

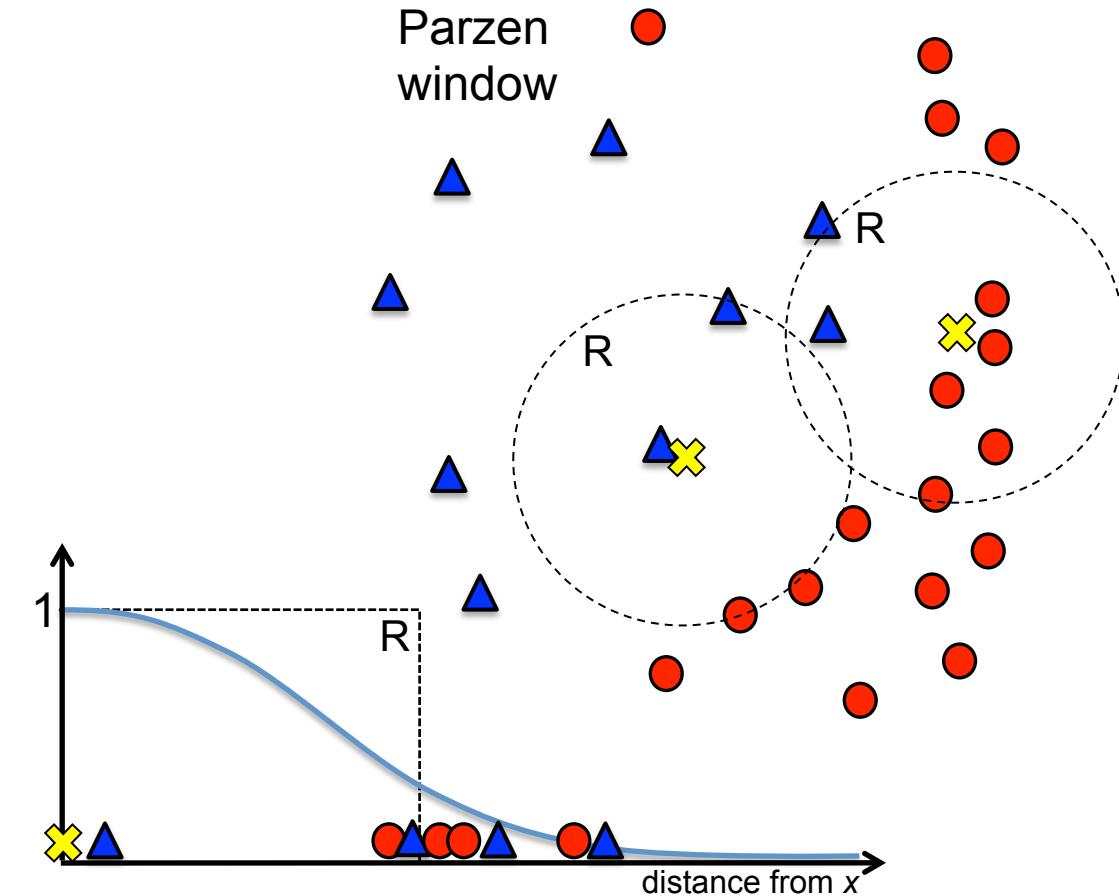
kNN, Parzen Windows and Kernels



$$f(x) = \text{sgn} \left[\sum_{i: x_i \in R(x)} y_i \right]$$

$$= \text{sgn} \left[\sum_i y_i \cdot 1_{\|x_i - x\| \leq R} \right] = \text{sgn} \left[\sum_i y_i K(x_i, x) \right] \text{ vs. } \text{sgn} \left[\sum_i \alpha_i y_i K(x_i, x) \right]$$

kernelized SVM



kNN pros and cons

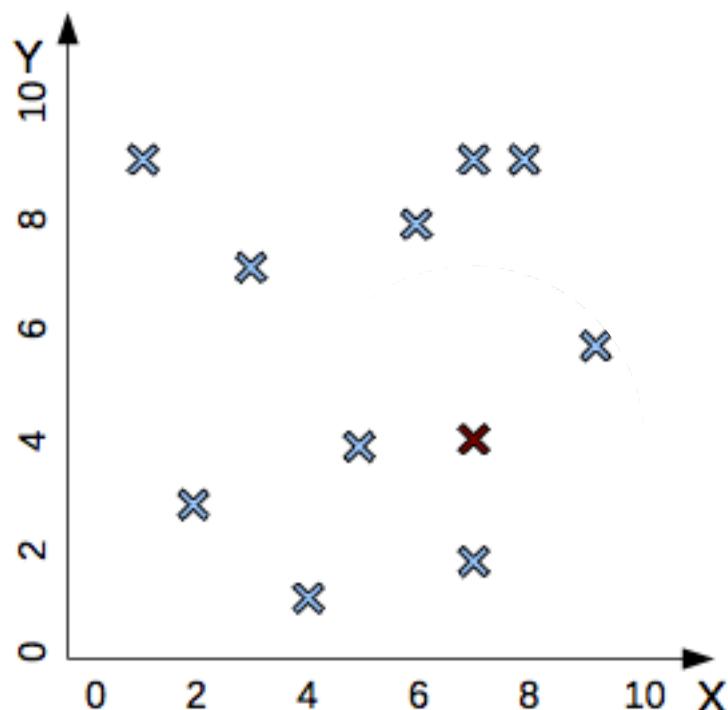
- Almost no assumptions about the data
 - smoothness: nearby regions of space → same class
 - assumptions implied by distance function (only locally!)
 - non-parametric approach: “let the data speak for itself”
 - nothing to infer from the data, except k and possibly $D()$
 - easy to update in online setting: just add new item to training set
- Need to handle missing data: fill-in or create a special distance
- Sensitive to class-outliers (mislabeled training instances)
- Sensitive to lots of irrelevant attributes (affect distance)
- Computationally expensive:
 - space: need to store all training examples
 - time: need to compute distance to all examples: $O(nd)$
 - n ... number of training examples, d ... cost of computing distance
 - n grows → system will become slower and slower
 - expense is at *testing*, not *training* time (bad)

Summary: kNN

- Key idea: nearby points → same class
 - important to select good distance function
- Can be used for classification and regression
- Simple, non-linear, asymptotically optimal
 - does not make assumptions about the data
 - “let the data speak for itself”
- Select k by optimizing error on held-out set
- Naïve implementations slow for big datasets
 - use K-D trees (low-d) or inverted lists (high-d)

Why is kNN slow?

What you see



Find nearest neighbors
of the testing point (red)

What algorithm sees

- Training set:
 $\{(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)\}$
- Testing instance:
 $(7,4)$
- Nearest neighbors?
compare one-by-one
to each training instance
- n comparisons
- each takes d operations

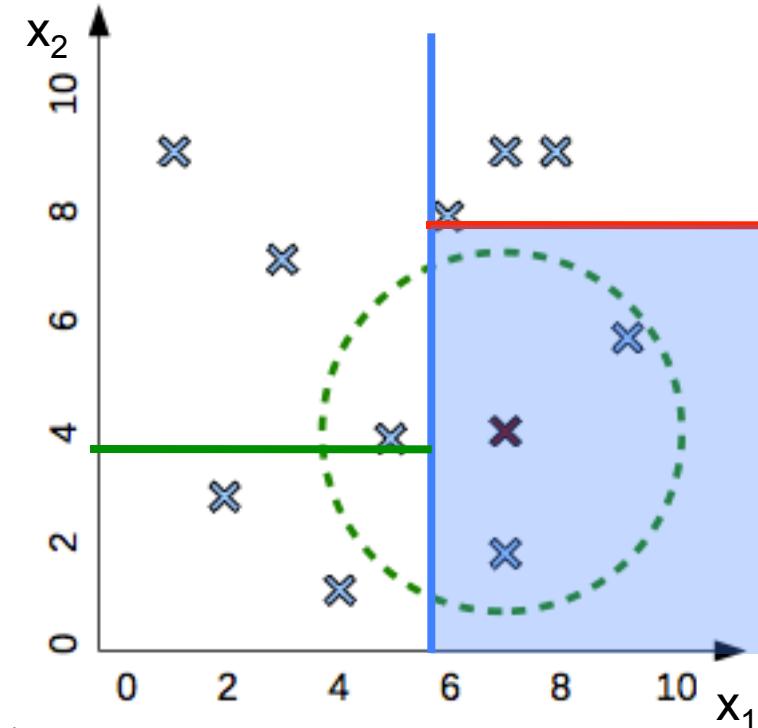
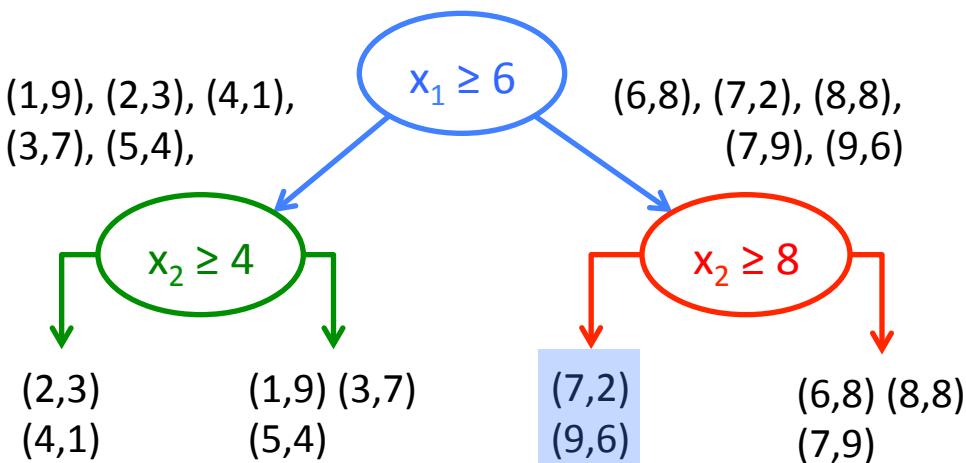
Making kNN fast

- Training: $O(d)$, but testing: $O(nd)$
- **Reduce d :** dimensionality reduction
 - simple feature selection, other methods $O(d^3)$
- **Reduce n :** don't compare to **all** training examples
 - idea: quickly identify $m \ll n$ potential near neighbors
 - compare only to those, pick k nearest neighbors $\rightarrow O(md)$ time
 - **K-D trees:** low-dimensional, real-valued data
 - $O(d \log_2 n)$, only works when $d \ll n$, inexact: **can miss neighbors**
 - **inverted lists:** high-dimensional, discrete (sparse) data
 - $O(n'd')$ where $d' \ll d$, $n' \ll n$, only for sparse data (e.g. text), exact
 - **locality-sensitive hashing:** high-d, real-valued or discrete
 - $O(n'd)$, $n' \ll n$... bits in fingerprint, inexact: **can miss near neighbors**

K-D tree example

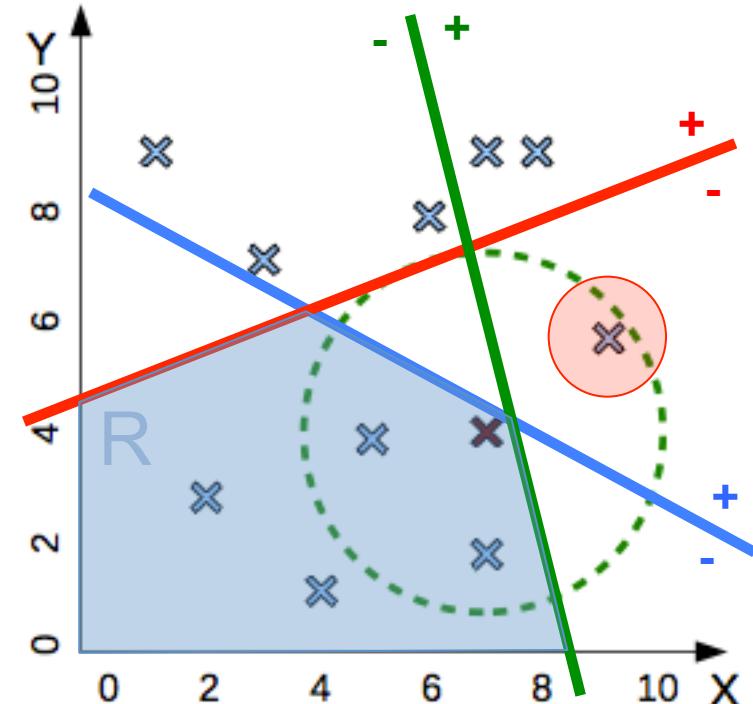
- Building a K-D tree from training data:
 - pick random dimension, find median, split data, repeat
- Find NNs for new point (7,4)
 - find region containing (7,4)
 - compare to all points in region

(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)



Locality-Sensitive Hashing (LSH)

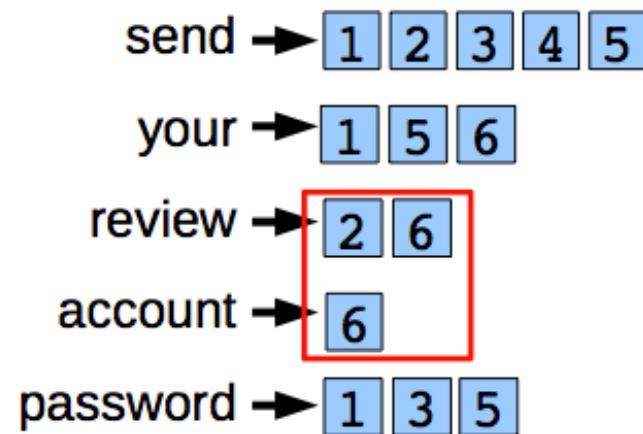
- Random hyper-planes $\mathbf{h}_1 \dots \mathbf{h}_k$
 - space sliced into 2^k regions (polytopes)
 - compare \mathbf{x} only to training points in the same region \mathbf{R}
- Complexity: $O(kd + dn/2^k) \approx O(d \log n)$ vs. $O(dn)$
 - $O(kd)$ to find region \mathbf{R} , $k \ll n$
 - dot-product \mathbf{x} with $\mathbf{h}_1 \dots \mathbf{h}_k$
 - compare to $n/2^k$ points in \mathbf{R}
 - dot-product \mathbf{x} with each point
- Inexact: missed neighbors
 - repeat with different $\mathbf{h}_1 \dots \mathbf{h}_k$
- Why not K-D tree?



Inverted list example

- Data structure used by search engines (Google, etc)
 - list all training examples that contain particular attribute
 - assumption: most attribute values are zero (sparseness)
- Given a new testing example:
 - merge inverted lists for attributes present in new example
 - $O(d\sqrt{n})$: d ... nonzero attributes, \sqrt{n} ... avg. length of list

D1: "send your password"	spam
D2: "send us review"	ham
D3: "send us password"	spam
D4: "send us details"	ham
D5: "send your password"	spam
D6: "review your account"	ham
new email: "account review"	



IAML: Dimensionality Reduction

Victor Lavrenko and Nigel Goddard
School of Informatics

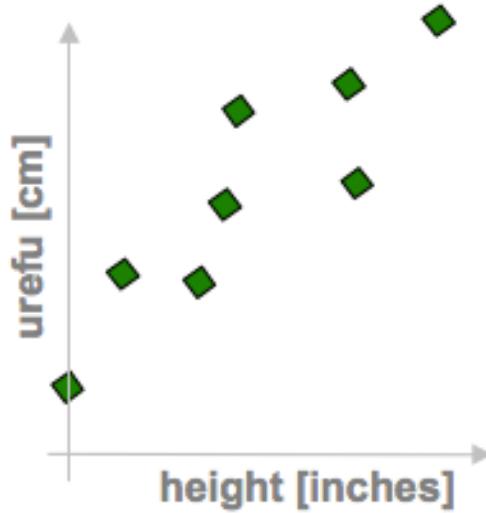
Semester 1

Overview

- Curse of dimensionality
- Different ways to reduce dimensionality
- Principal Components Analysis (PCA)
- Example: Eigen Faces
- PCA for classification
- Witten & Frank section 7.3
 - only the PCA section required

True vs. observed dimensionality

- Get a population, predict some property
 - instances represented as {urefu, height} pairs
 - what is the dimensionality of this data?

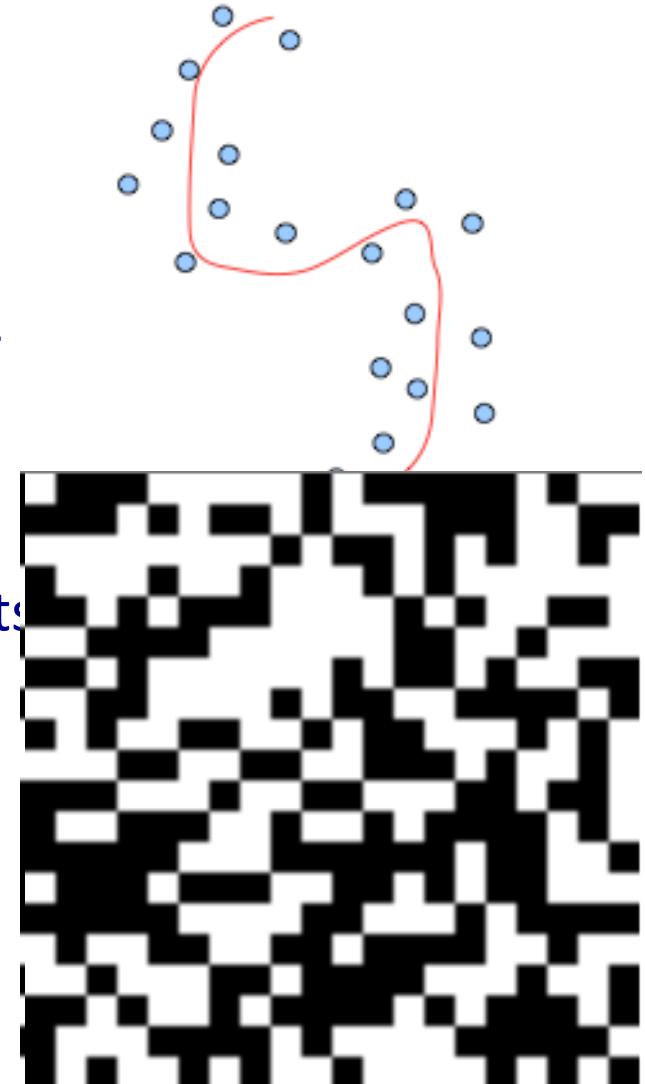


- Data points over time from different geographic areas over time:
 - X_1 : # of skidding accidents
 - X_2 : # of burst water pipes
 - X_3 : snow-plow expenditures
 - X_4 : # of school closures
 - X_5 : # patients with heat stroke

Temperature?

Curse of dimensionality

- Datasets typically high dimensional
 - vision: 10^4 pixels, text: 10^6 words
 - the way we observe / record them
 - true dimensionality often much lower
 - a manifold (sheet) in a high-d space
- Example: handwritten digits
 - 20×20 bitmap: $\{0,1\}^{400}$ possible events
 - will never see most of these events
 - actual digits: tiny fraction of events
 - true dimensionality:
 - possible variations of the pen-stroke



Curse of dimensionality (2)

- Machine learning methods are statistical by nature
 - count observations in various regions of some space
 - use counts to construct the predictor $f(x)$
 - e.g. decision trees: p_+/p_- in $\{o=\text{rain}, w=\text{strong}, T>28^\circ\}$
 - text: #documents in {"hp" and "3d" and not "\$" and ...})
- As dimensionality grows: fewer observations per region
 - 1d: 3 regions, 2d: 3^2 regions, 1000d – hopeless
 - statistics need repetition
 - flip a coin once → head
 - $P(\text{head}) = 100\%?$

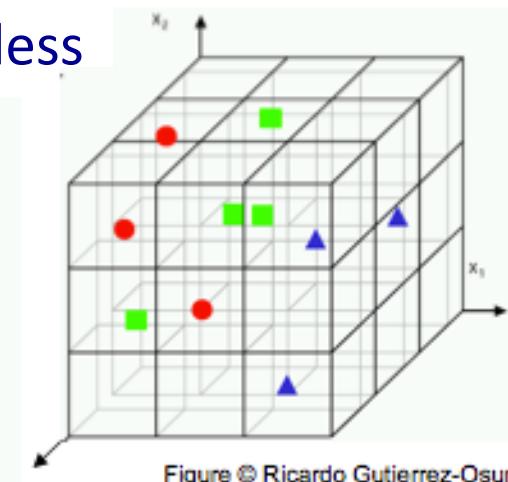
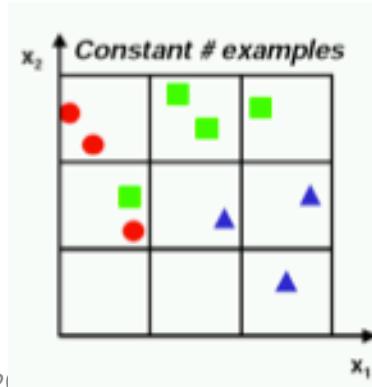
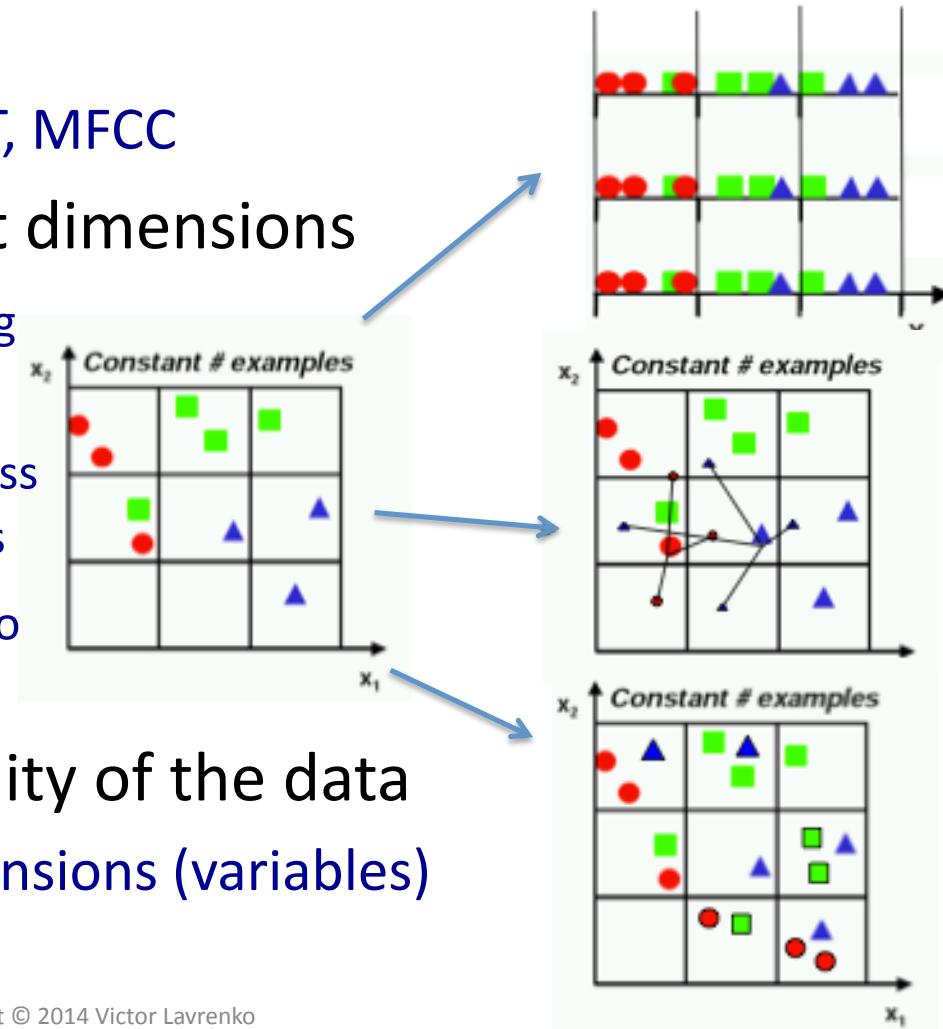


Figure © Ricardo Gutierrez-Osuna

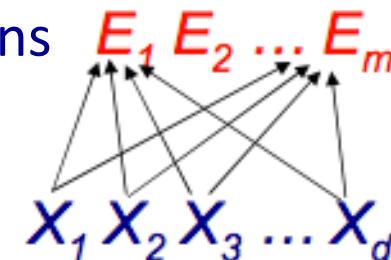
Dealing with high dimensionality

- Use domain knowledge
 - feature engineering: SIFT, MFCC
- Make assumption about dimensions
 - independence: count along each dimension separately
 - smoothness: propagate class counts to neighboring regions
 - symmetry: e.g. invariance to order of dimensions: $x_1 \Leftrightarrow x_2$
- Reduce the dimensionality of the data
 - create a new set of dimensions (variables)



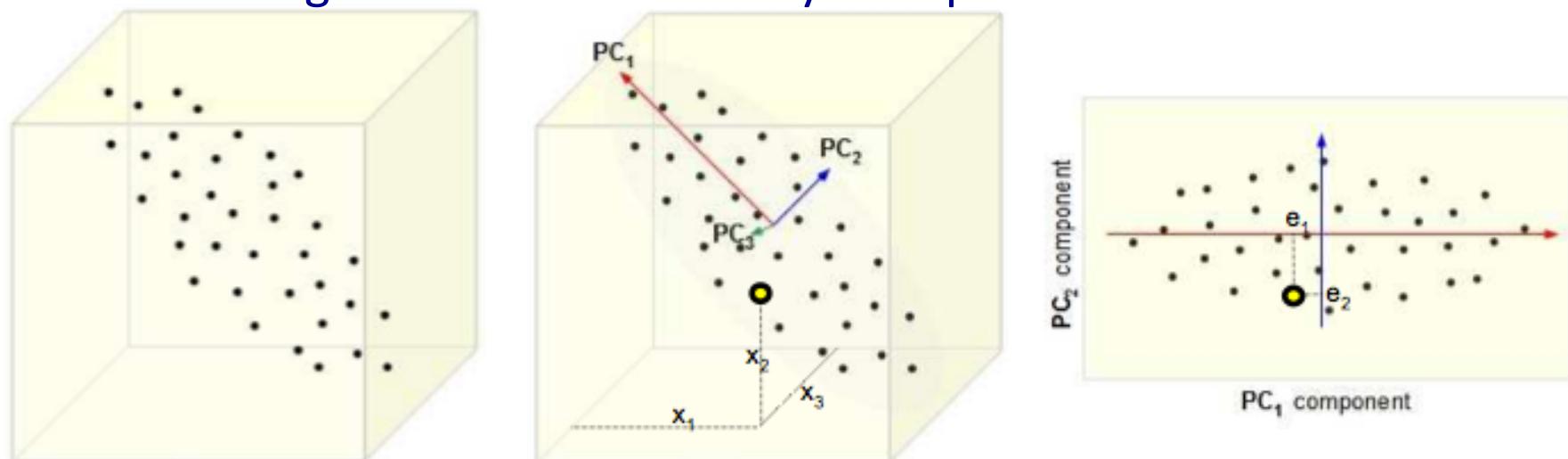
Dimensionality reduction

- Goal: represent instances with fewer variables
 - try to preserve as much structure in the data as possible
 - discriminative: only structure that affects class separability
- Feature selection
 - pick a subset of the original dimensions $X_1 X_2 X_3 \dots X_{d-1} X_d$
 - discriminative: pick good class “predictors” (e.g. gain)
- Feature extraction
 - construct a new set of dimensions $E_1 E_2 \dots E_m$
 $E_i = f(X_1 \dots X_d)$
 - (linear) combinations of original



Principal Components Analysis

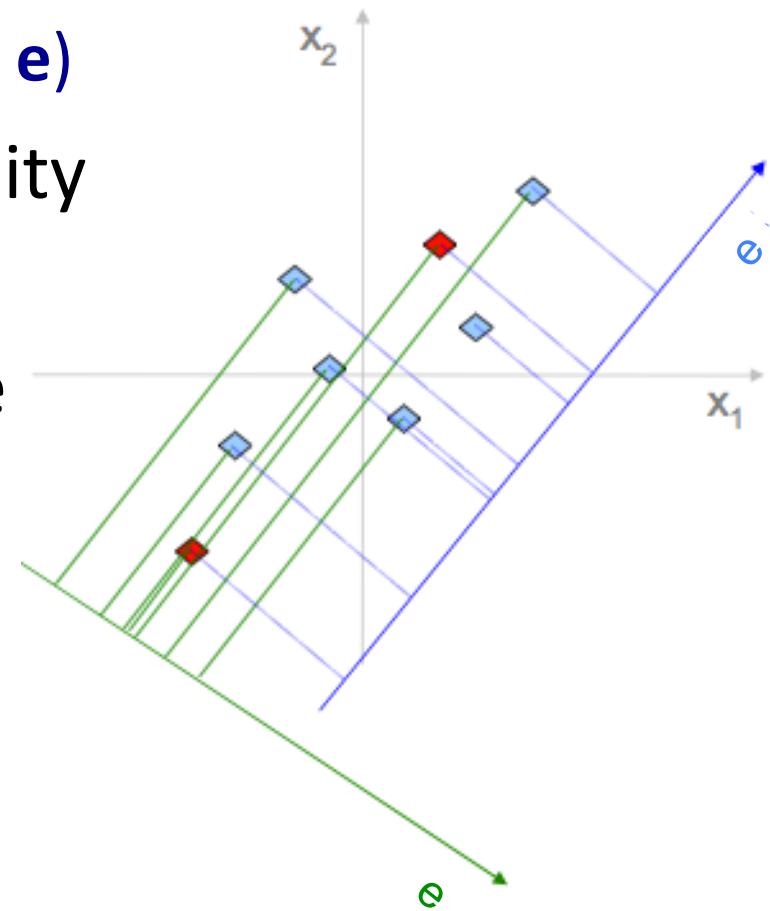
- Defines a set of principal components
 - 1st: direction of the greatest variability in the data
 - 2nd: perpendicular to 1st, greatest variability of what's left
 - ... and so on until d (original dimensionality)
- First $m < d$ components become m new dimensions
 - change coordinates of every data point to these dimensions



Copyright © 2014 Victor Lavrenko

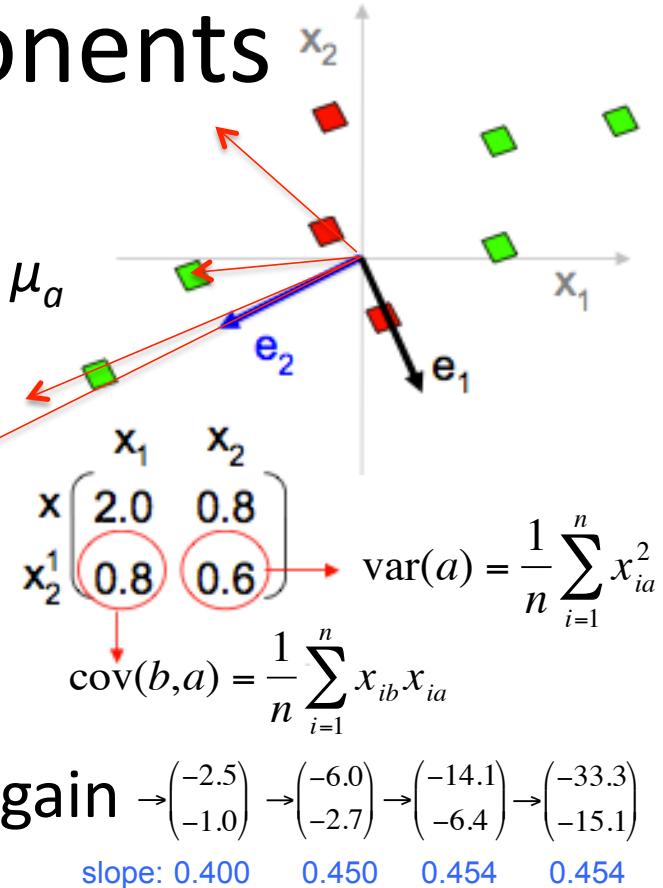
Why greatest variability?

- Example: reduce 2-dimensional data to 1-d
 - $\{x_1, x_2\} \rightarrow e'$ (along new axis e)
- Pick e to maximize variability
- Reduces cases when two points are close in e -space but very far in (x,y) -space
- Minimizes distances between original points and their projections



Principal components

- “Center” the data at zero: $x_{i,a} = x_{i,a} - \mu_a$
 - subtract mean from each attribute
- Compute covariance matrix Σ
 - covariance of dimensions x_1 and x_2 :
 - do x_1 and x_2 tend to increase together?
 - or does x_2 decrease as x_1 increases?
- Multiply a vector by Σ : $\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} -1 \\ +1 \end{pmatrix} \rightarrow \begin{pmatrix} -1.2 \\ -0.2 \end{pmatrix}$ again
 - turns towards direction of variance
- Want vectors \mathbf{e} which aren’t turned: $\Sigma \mathbf{e} = \lambda \mathbf{e}$
 - \mathbf{e} ... eigenvectors of Σ , λ ... corresponding eigenvalues
 - principal components = eigenvectors w. largest eigenvalues



Finding Principal Components

1. find eigenvalues by solving: $\det(\Sigma - \lambda I) = 0$

$$\det \begin{pmatrix} 2.0 - \lambda & 0.8 \\ 0.8 & 0.6 - \lambda \end{pmatrix} = (2 - \lambda)(0.6 - \lambda) - (0.8)(0.8) = \lambda^2 - 2.6\lambda + 0.56 = 0$$
$$\{\lambda_1, \lambda_2\} = \frac{1}{2} \left(2.6 \pm \sqrt{2.6^2 - 4 * 0.56} \right) = \{2.36, 0.23\}$$

2. find i^{th} eigenvector by solving: $\Sigma e_i = \lambda_i e_i$

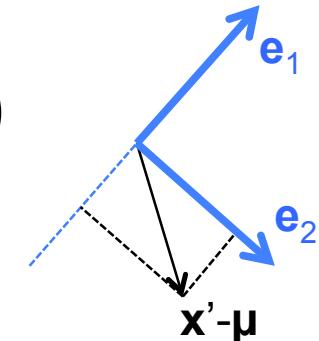
$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix} = 2.36 \begin{pmatrix} e_{1,1} \\ e_{1,2} \end{pmatrix} \quad \begin{array}{l} 2.0e_{1,1} + 0.8e_{1,2} = 2.36e_{1,1} \\ 0.8e_{1,1} + 0.6e_{1,2} = 2.36e_{1,2} \end{array} \quad \begin{array}{l} e_{1,1} = 2.2e_{1,2} \\ \downarrow \\ e_1 \sim \begin{bmatrix} 2.2 \\ 1 \end{bmatrix} \end{array}$$

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix} = 0.23 \begin{pmatrix} e_{2,1} \\ e_{2,2} \end{pmatrix} \quad \begin{array}{l} e_2 = \begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix} \\ \downarrow \\ \text{want: } \|e_1\| = 1 \end{array}$$

$$3. 1^{st} \text{ PC: } \begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix}, 2^{nd} \text{ PC: } \begin{bmatrix} -0.41 \\ 0.91 \end{bmatrix} \quad e_1 = \begin{bmatrix} 0.91 \\ 0.41 \end{bmatrix} \quad \text{slope: 0.454}$$

Projecting to new dimensions

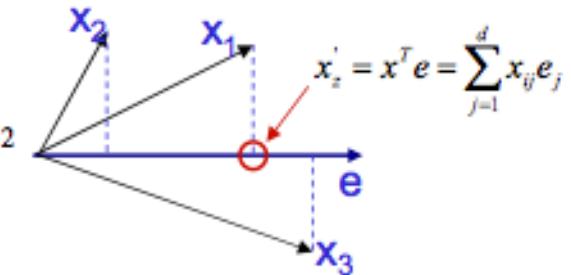
- $e_1 \dots e_m$ are new dimension vectors
- Have instance $x = \{x_1 \dots x_d\}$ (original coordinates)
- Want new coordinates $x' = \{x'_1 \dots x'_m\}$:
 1. “center” the instance (subtract the mean): $x' - \mu$
 2. “project” to each dimension: $(x' - \mu)^T e_j$ for $j=1\dots m$



$$\begin{aligned}
 (\vec{x} - \vec{\mu}) &= \begin{bmatrix} (x_1 - \mu_1) & (x_2 - \mu_2) & \cdots & (x_d - \mu_d) \end{bmatrix} \\
 \begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_m' \end{bmatrix} &= \begin{bmatrix} (\vec{x} - \vec{\mu})^T \vec{e}_1 \\ (\vec{x} - \vec{\mu})^T \vec{e}_2 \\ \vdots \\ (\vec{x} - \vec{\mu})^T \vec{e}_m \end{bmatrix} = \begin{bmatrix} (x_1 - \mu_1)e_{1,1} + (x_2 - \mu_2)e_{1,2} + \cdots + (x_d - \mu_d)e_{1,d} \\ (x_1 - \mu_1)e_{2,1} + (x_2 - \mu_2)e_{2,2} + \cdots + (x_d - \mu_d)e_{2,d} \\ \vdots \\ (x_1 - \mu_1)e_{m,1} + (x_2 - \mu_2)e_{m,2} + \cdots + (x_d - \mu_d)e_{m,d} \end{bmatrix}
 \end{aligned}$$

Direction of greatest variability

- Select dimension \mathbf{e} which maximizes the variance
- Points \mathbf{x}_i “projected” onto vector \mathbf{e} :
- Variance of projections: $\frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j - \mu \right)^2 = \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right)^2$
- Maximize variance
 - want unit length: $\|\mathbf{e}\|=1$
 - add Lagrange multiplier



$$V = \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right)^2 - \lambda \left(\left(\sum_{k=1}^d e_k^2 \right) - 1 \right)$$

$$\frac{\partial V}{\partial e_a} = \frac{2}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right) x_{ia} - 2\lambda e_a = 0$$

$\Sigma e = \lambda e$ ←
 \mathbf{e} must be an eigenvector

$$\left\{ \begin{array}{l} \sum_{j=1}^d \text{cov}(1,j) e_j = \lambda e_1 \\ \vdots \\ \sum_{j=1}^d \text{cov}(d,j) e_j = \lambda e_d \end{array} \right.$$

hold for $a=1..d$

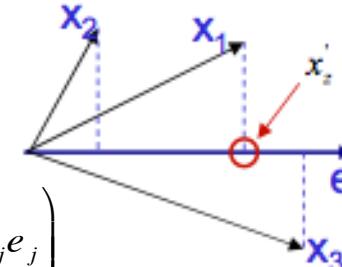
$$2 \sum_{j=1}^d e_j \left(\underbrace{\frac{1}{n} \sum_{i=1}^n x_{ia} x_{ij}}_{\text{covariance of } a,j} \right) = 2\lambda e_a$$

Variance along eigenvector

Variance of projected points ($\mathbf{x}^T \mathbf{e}$):

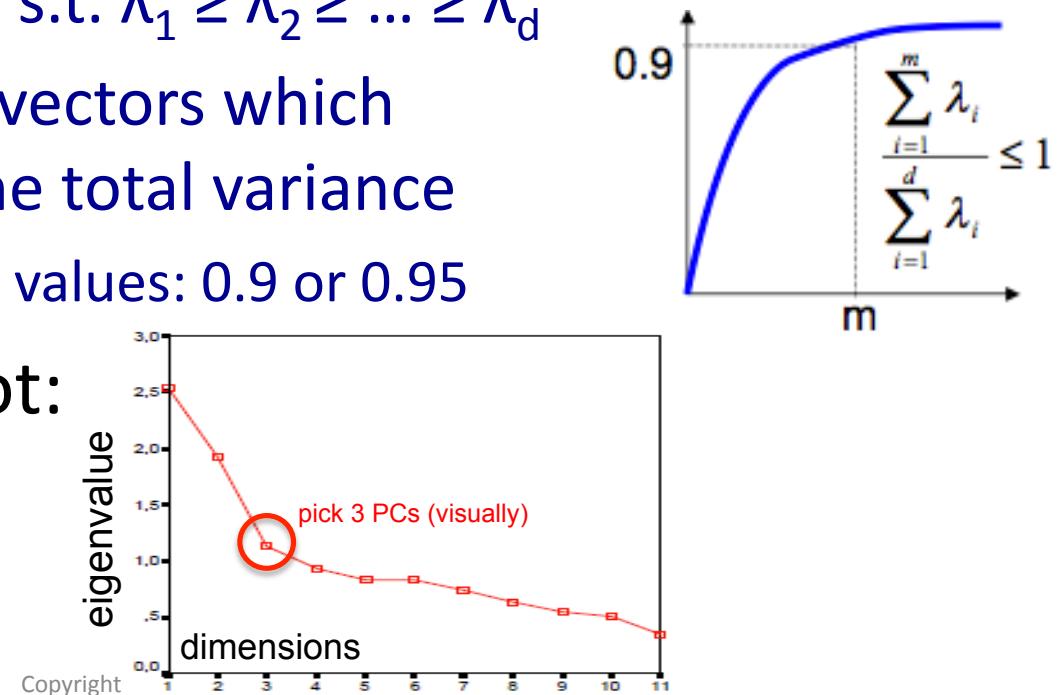
$$\begin{aligned}
 \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j - \mu \right)^2 &= \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right)^2 \\
 &= \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right) \left(\sum_{a=1}^d x_{ia} e_a \right) \\
 &= \sum_{a=1}^d \sum_{j=1}^d \left(\frac{1}{n} \sum_{i=1}^n x_{ia} x_{ij} \right) e_j e_a \\
 &= \sum_{a=1}^d \left(\sum_{j=1}^d \text{cov}(a, j) e_j \right) e_a \\
 &= \sum_{a=1}^d (\lambda e_a) e_a \\
 &= \lambda \|e\|^2 = \lambda
 \end{aligned}$$

$$\begin{aligned}
 \mu &= \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^d x_{ij} e_j \right) \\
 &= \sum_{j=1}^d \left(\frac{1}{n} \sum_{i=1}^n x_{ij} \right) e_j \\
 \text{cov}(a, j) &= \frac{1}{n} \sum_{i=1}^n x_{ia} x_{ij} \\
 \sum_{j=1}^d \text{cov}(a, j) e_j &= \lambda e_a \quad \mathbf{e} \text{ is an eigenvector of the covariance matrix}
 \end{aligned}$$



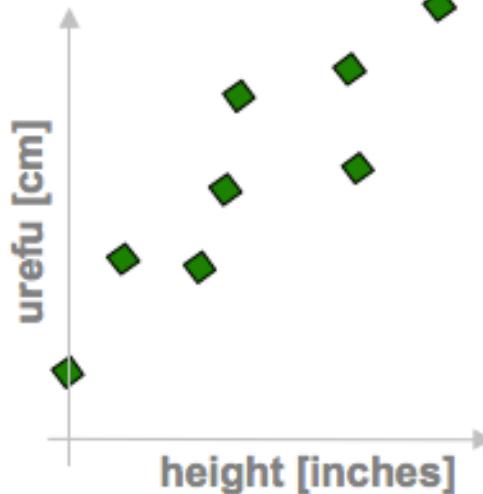
How many dimensions?

- Have: eigenvectors $\mathbf{e}_1 \dots \mathbf{e}_d$ want: $m \ll d$
- Proved: eigenvalue λ_i = variance along \mathbf{e}_i
- Pick \mathbf{e}_i that “explain” the most variance
 - sort eigenvectors s.t. $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$
 - pick first m eigenvectors which explain 90% or the total variance
 - typical threshold values: 0.9 or 0.95
- Or use a scree plot:
 - like K-means

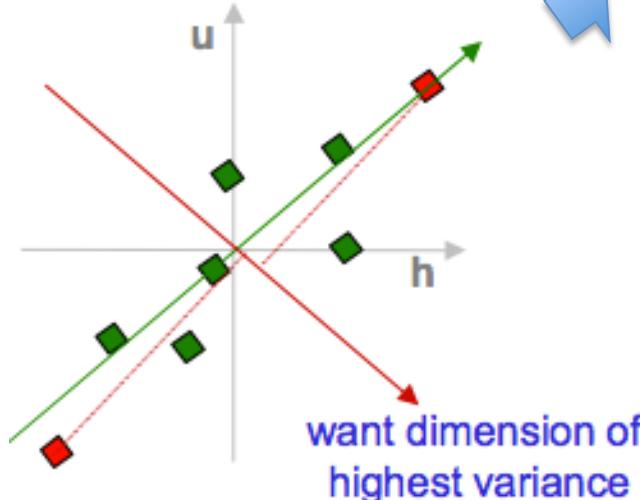


PCA in a nutshell

1. correlated hi-d data
("urefu" means "height" in Swahili)



2. center the points



3. compute covariance matrix

$$\begin{matrix} h & u \\ h & 2.0 & 0.8 \\ u & 0.8 & 0.6 \end{matrix} \rightarrow \text{cov}(h, u) = \frac{1}{n} \sum_{i=1}^n h_i u_i$$

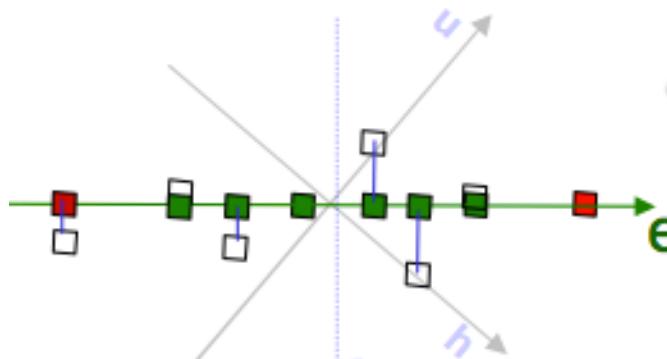
4. eigenvectors + eigenvalues

$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} e_h \\ e_u \end{pmatrix} = \Lambda_e \begin{pmatrix} e_h \\ e_u \end{pmatrix}$$

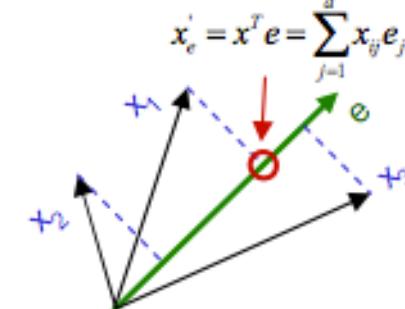
$$\begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} f_h \\ f_u \end{pmatrix} = \Lambda_f \begin{pmatrix} f_h \\ f_u \end{pmatrix}$$

`eig(cov(data))`

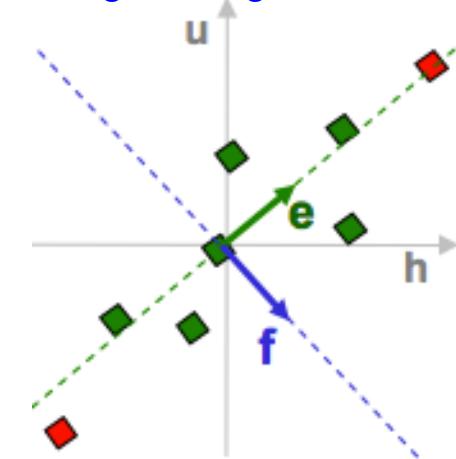
7. uncorrelated low-d data



6. project data points to those eigenvectors



5. pick m<d eigenvectors w. highest eigenvalues



PCA example: Eigen Faces

input: dataset of N face images



face: $K \times K$ bitmap of pixels



“unfold” each bitmap to
 K^2 -dimensional vector

arrange in a matrix
each face = column



Matlab demo on course webpage



can visualize
eigenvectors:
 m “aspects”
of prototypical
facial features



“fold” into a $K \times K$ bitmap



PCA

$K^2 \times m$

set of m eigenvectors
each is K^2 -dimensional

Eigen Faces: Projection



= mean + 0.9 *



- 0.2 *



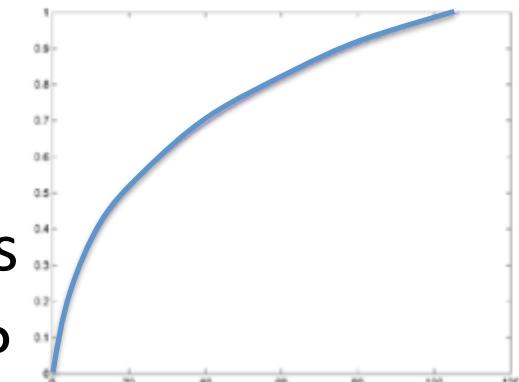
+ 0.4 *



+ ...



- Project new face to space of eigen-faces
- Represent vector as a linear combination of principal components
- How many do we need?



(Eigen) Face Recognition

- Face similarity
 - in the reduced space
 - insensitive to lighting expression, orientation
- Projecting new “faces”
 - everything is a face

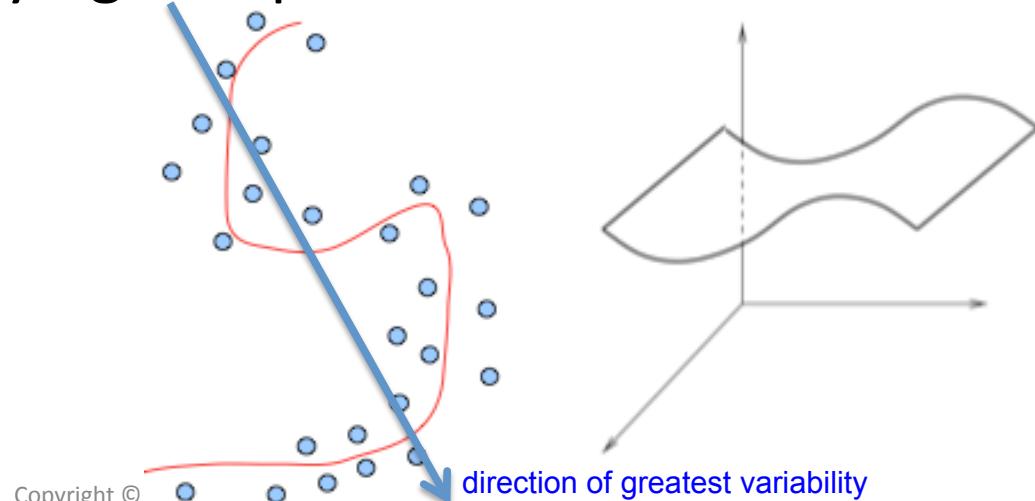


new face

projected to eigenfaces

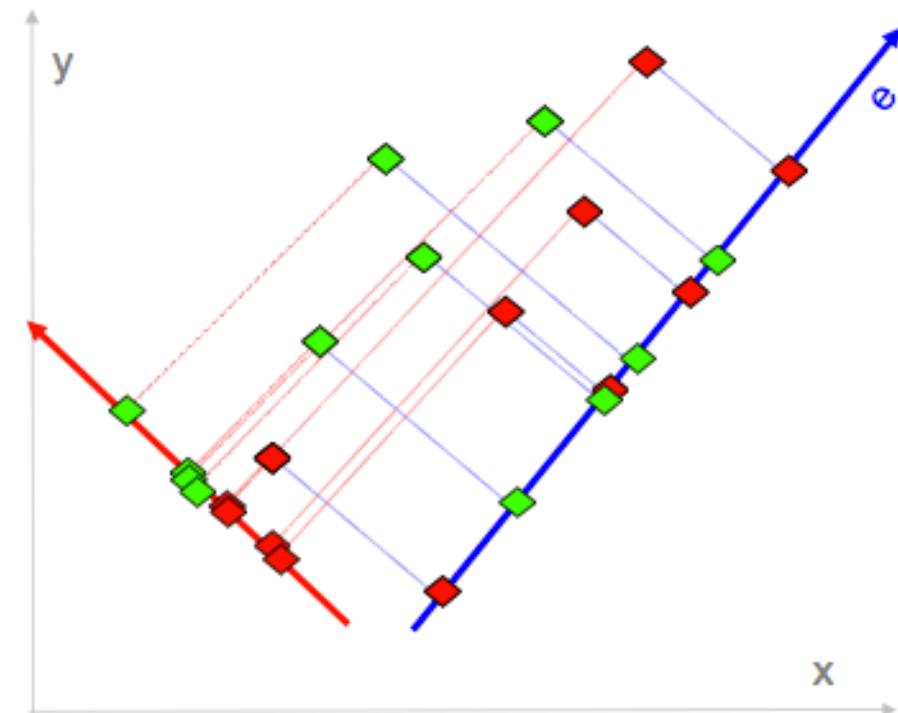
PCA: practical issues

- Covariance extremely sensitive to large values
 - multiply some dimension by 1000
 - dominates covariance
 - becomes a principal component
 - normalize each dimension to zero mean and unit variance:
 $x' = (x - \text{mean}) / \text{st.dev}$
- PCA assumes underlying subspace is linear
 - 1d: straight line
 - 2d: flat sheet
 - transform to handle non-linear spaces (manifolds)



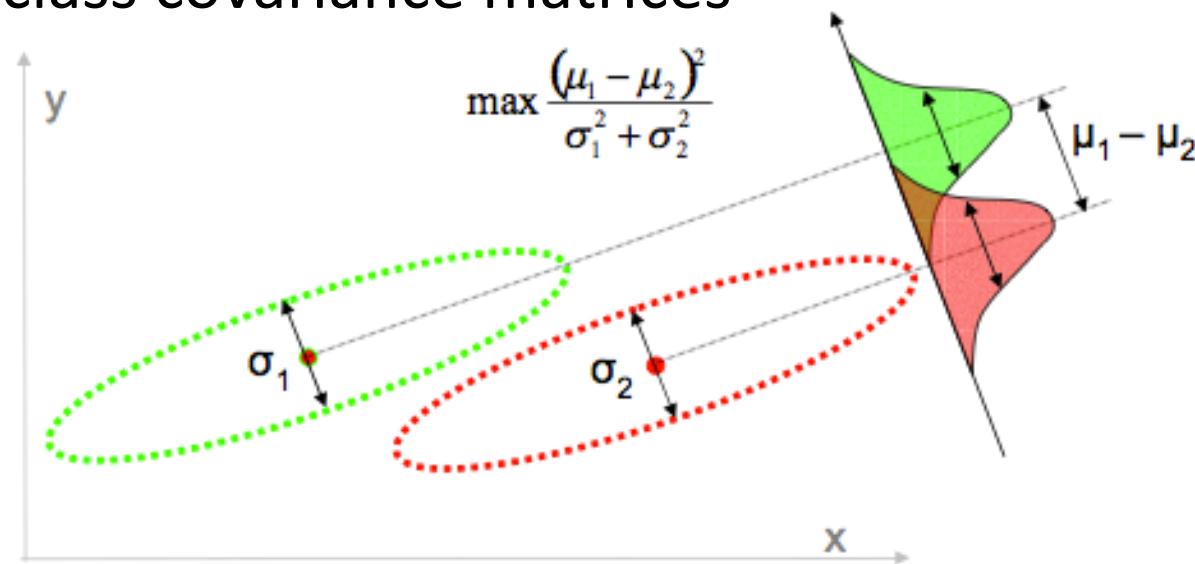
PCA and classification

- PCA is unsupervised
 - maximizes overall variance of the data along a small set of directions
 - does not know anything about class labels
 - can pick direction that makes it hard to separate classes
- Discriminative approach
 - look for a dimension that makes it easy to separate classes



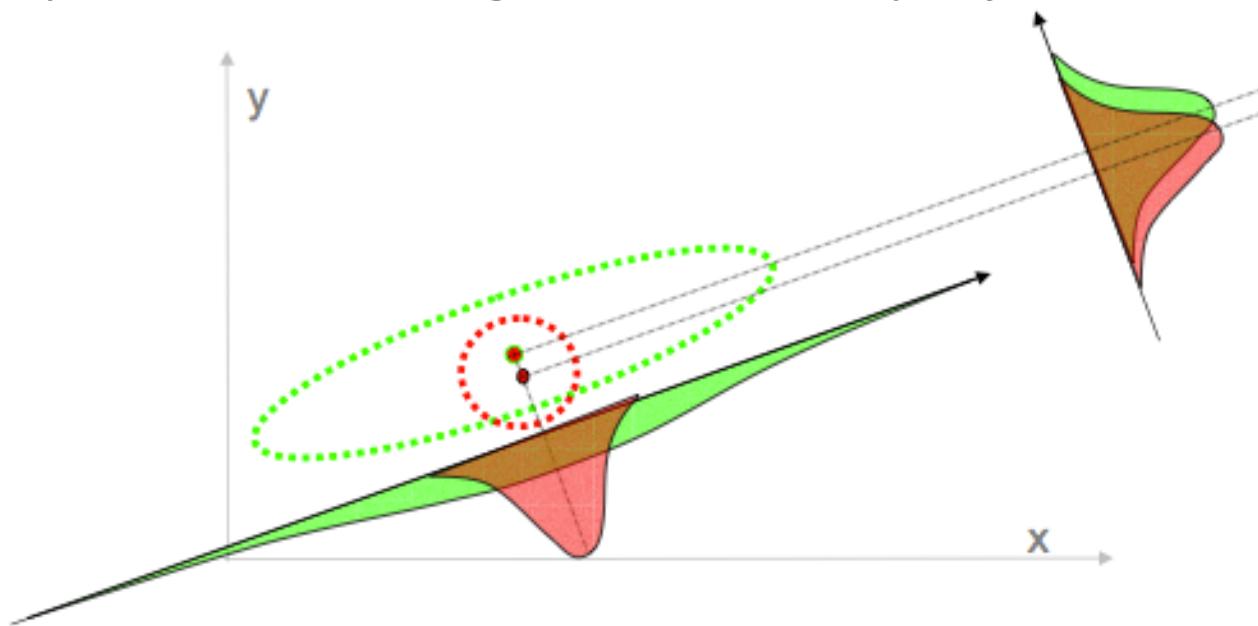
Linear Discriminant Analysis

- LDA: pick a new dimension that gives:
 - maximum separation between means of projected classes
 - minimum variance within each projected class
- Solution: eigenvectors based on between-class and within-class covariance matrices



PCA vs. LDA

- LDA not guaranteed to be better for classification
 - assumes classes are unimodal Gaussians
 - fails when discriminatory information is not in the mean, but in the variance of the data
- Example where PCA gives a better projection:



Dimensionality reduction

- Pros
 - reflects our intuitions about the data
 - allows estimating probabilities in high-dimensional data
 - no need to assume independence etc.
 - dramatic reduction in size of data
 - faster processing (as long as reduction is fast), smaller storage
- Cons
 - too expensive for many applications (Twitter, web)
 - disastrous for tasks with fine-grained classes
 - understand assumptions behind the methods (linearity etc.)
 - there may be better ways to deal with sparseness

Summary

- True dimensionality << observed dimensionality
- High dimensionality → sparse, unstable estimates
- Dealing with high dimensionality:
 - use domain knowledge
 - make an assumption: independence / smoothness / symmetry
 - dimensionality reduction: feature selection / feature extraction
- Principal Components Analysis (PCA)
 - picks dimensions that maximize variability
 - eigenvectors of the covariance matrix
 - examples: Eigen Faces
 - variant for classification: Linear Discriminant Analysis

IAML: K-means Clustering

Victor Lavrenko and Nigel Goddard

School of Informatics

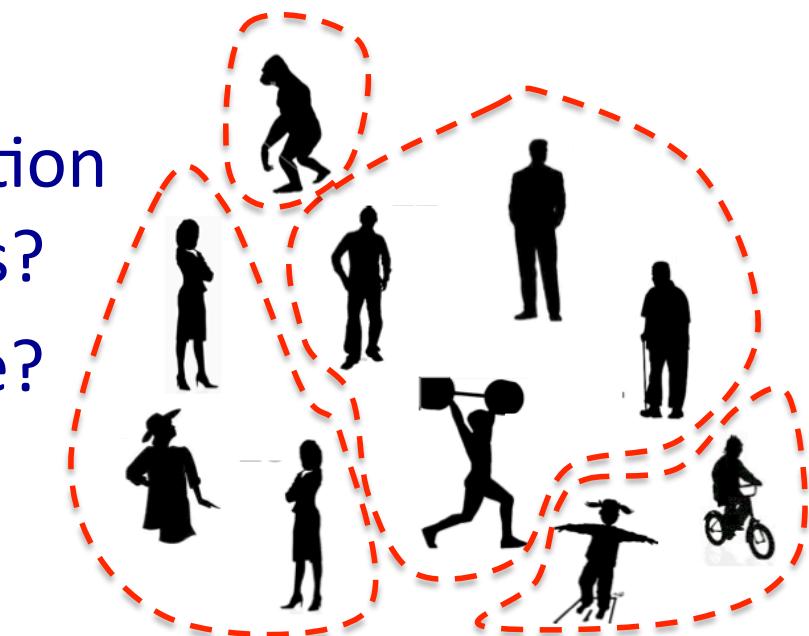
Semester 1

Overview

- Clustering
- K-means algorithm
- Practical issues: local optimum, selecting K
- Evaluating clustering algorithms
- Application: image representation
- Reading:
 - Witten & Frank sections 4.8 and 6.6

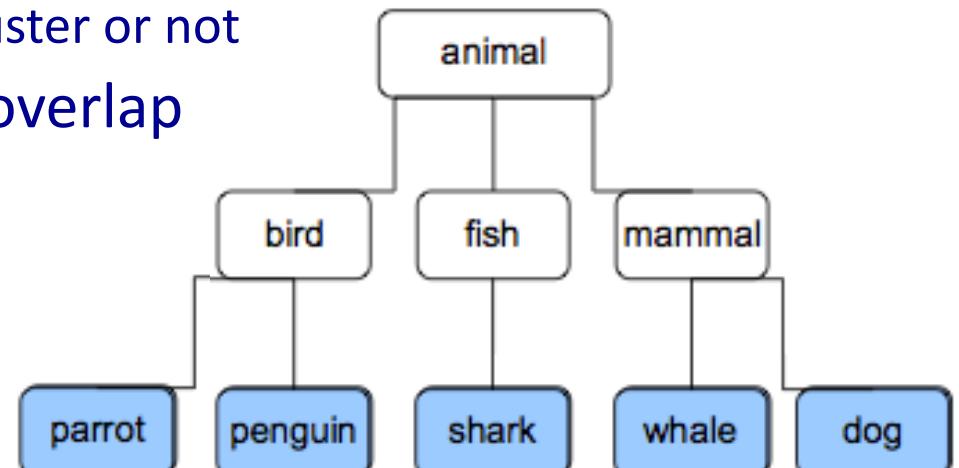
Clustering

- Discover the underlying structure of the data
 - unsupervised task, not predicting anything specific
- What sub-populations exist in the data?
 - how many are there?
 - what are their sizes?
 - do elements in a sub-population have any common properties?
 - are sub-populations cohesive?
can they be further split up?
 - are there outliers?



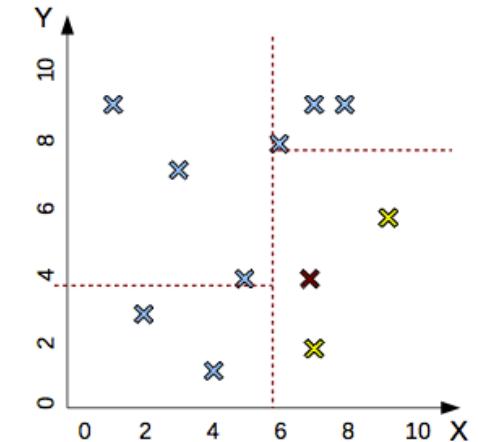
Types of clustering methods

- Goal:
 - monothetic: cluster members have some common property
 - e.g. all are males aged 20-35, or all have X% response to test B
 - polythetic: cluster members are similar to each other
 - distance between elements defines membership
- Overlap:
 - hard clustering: clusters do not overlap
 - element either belongs to a cluster or not
 - soft clustering: clusters may overlap
 - “strength of association” between element and cluster
- Flat or hierarchical
 - set of groups vs. taxonomy



Methods we will cover

- K-D Trees (see k-NN lecture)
 - monothetic, hard boundaries, hierarchical
- K-means clustering
 - splits data into a specified number of populations
 - polythetic, hard boundaries, flat
- Gaussian mixtures (EM algorithm)
 - fits a mixture of K Gaussians to the data
 - polythetic, soft boundaries, flat
- Agglomerative clustering
 - creates an “ontology” of nested sub-populations
 - polythetic, hard boundaries, hierarchical



K-means clustering

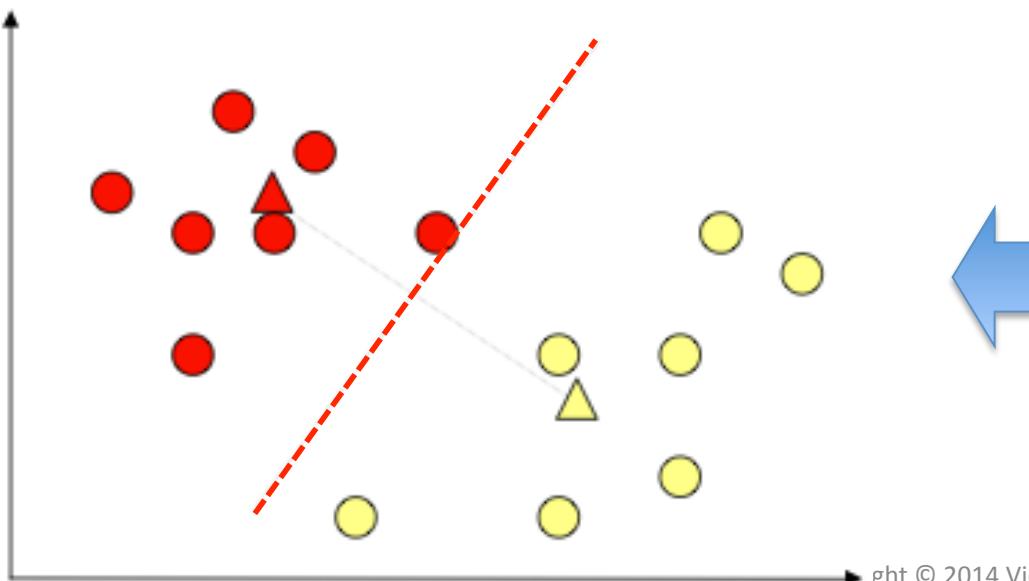
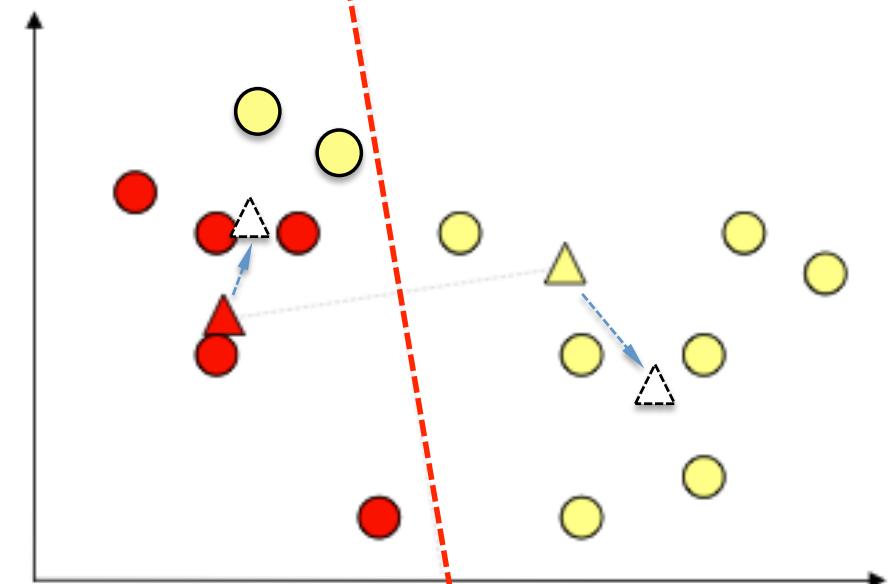
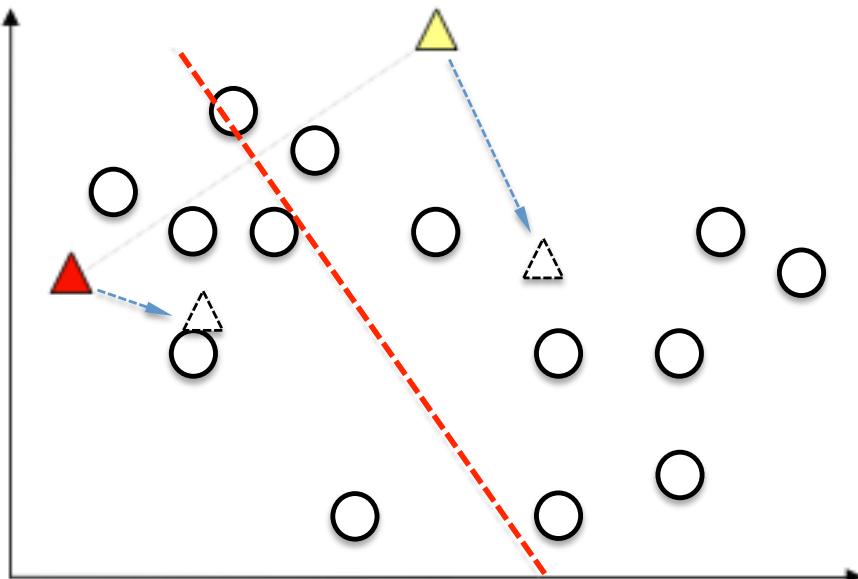


- Produces hard, flat, polythetic clusters
 - data partitioned into K sub-populations (need to know K)
 - points in each sub-population similar to a “centroid”
 - centroid = attribute-value “representation” of a cluster
 - “prototypical” individual in a sub-population
- Uses:
 - discover classes in an unsupervised manner
 - e.g. cluster images of handwritten digits (with $K = 10$)
 - smoothness over space
 - in the same cluster → similar representations / class labels / ...
 - dimensionality reduction: clusters = “latent factors”
 - replace representation of each data point with its cluster number
 - assumes all pertinent qualities reflected in cluster membership
 - related to basis / kernels in linear classifiers

K-means clustering algorithm

- Input: K, set of points $x_1 \dots x_n$
 - Place centroids $c_1 \dots c_K$ at random locations
 - Repeat until convergence:
 - for each point x_i :
 - find nearest centroid c_j $\underbrace{\arg \min_j D(x_i, c_j)}$
 - assign the point x_i to cluster j
 - for each cluster $j = 1 \dots K$: $c_j(a) = \frac{1}{n_{j|x_i \rightarrow c_j}} \sum x_i(a)$ for $a = 1 \dots d$
 - new centroid c_j = mean of all points x_i assigned to cluster j in previous step
 - Stop when none of the cluster assignments change
- $O(\#iterations * \#clusters * \#instances * \#dimensions)$

K-means clustering example



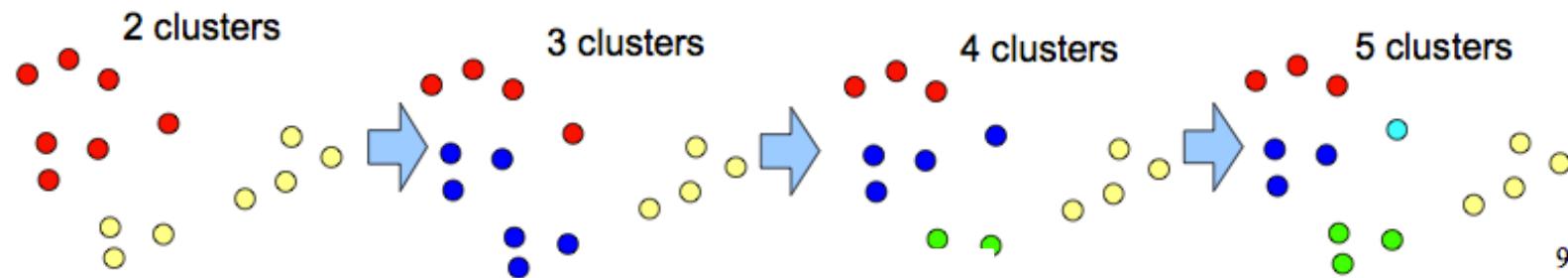
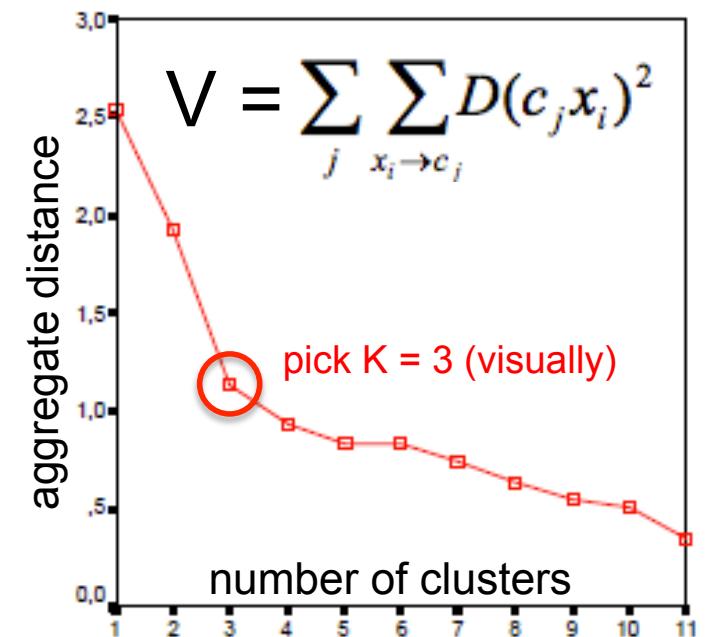
K-means properties

- Minimizes aggregate intra-cluster distance $\sum_j \sum_{x_i \rightarrow c_j} D(c_j, x_i)^2$
 - total squared distance from point to centre of its cluster
 - same as variance if Euclidian distance is used
- Converges to a local minimum
 - different starting points \rightarrow very different results
 - run several times with random starting points
 - pick clustering that yields smallest aggregate distance
- Nearby points may not end up in the same cluster
 - the following clustering is a stable local minimum:



Optimal number of clusters

- How many clusters are there in your data?
 - class labels may suggest the value of K (e.g. digits 0..9)
 - optimize distance V: for K = 2,3,...
 - run K-means, record distance
 - problem: V minimized when K = n
 - what if we use a validation set?
 - W&F: Minimum Description Length
 - total bits to encode K centroids + V
 - visually from scree plot:
 - point where “mountain” ends, “rubble” begins
 - elbow method: maximize 2nd derivative of V:
point where rate of decline changes the most

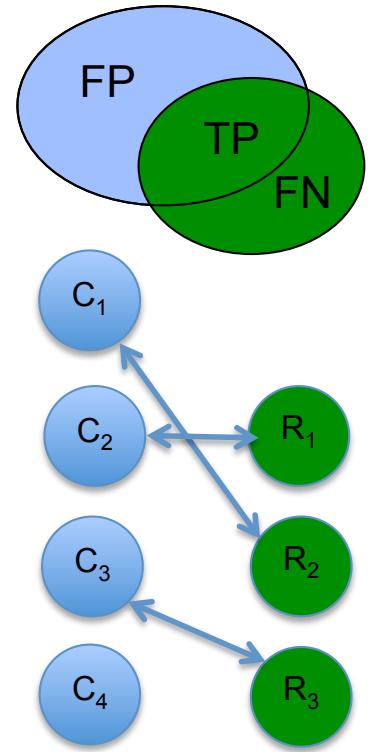


Evaluating Clustering Algorithms

- **Extrinsic** (helps us solve another problem)
 - represent images with cluster features
 - train different classifier for each sub-population
 - identify and eliminate outliers / corrupted points
 - **Intrinsic** (useful in and of itself)
 - helps understand the makeup of our data (qualitative)
 - clusters correspond to classes (digits → 10 clusters)
 - align, evaluate as you would a normal classifier
 - compare to human judgments
 - can't ask humans to “cluster” a dataset manually
 - sample pairs x_i, x_j , ask humans if they “match”
- 
- did your classifier improve?

Intrinsic Evaluation 1

- System produces clusters $C_1 C_2 \dots C_K$
- Reference clusters (classes) $R_1 R_2 \dots R_N$
- Align up $R_i \leftrightarrow C_j$, measure accuracy, F1, ...
 - many different ways to align:
 - Weka: $C_j \rightarrow R_i$ with max overlap
 - if many $C_j \rightarrow$ same R_i :
 - re-assign in a greedy manner
 - non-greedy: $K!/(N-K)!$ ways (very slow)
 - can we have multiple $C_j \rightarrow$ same R_i ?
 - can we have multiple $R_i \rightarrow$ same C_j ?
 - can we have overlapping clusters?



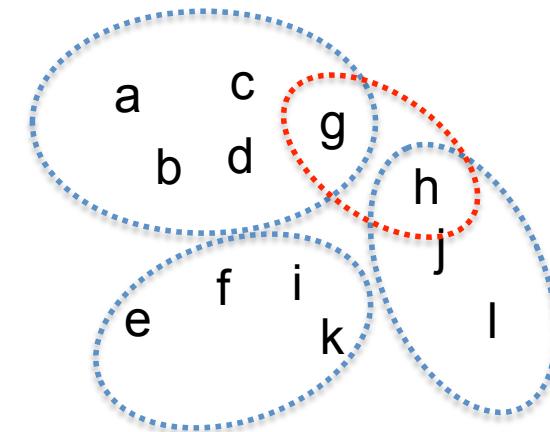
true class	$\rightarrow R_2$	$\rightarrow R_1$	$\rightarrow R_3$	
cluster	$\rightarrow c_1$	$\rightarrow c_2$	$\rightarrow c_3$	$\rightarrow c_4$
R_2	3	1	2	6
R_1	0	0	1	1
R_3	7	1	8	16
c_1	3	1	2	6
c_2	0	0	1	1
c_3	7	1	8	16
c_4	2	0	1	3
	12	2	12	

$$\text{Accuracy} = (3+0+8)/26$$

Intrinsic Evaluation 2

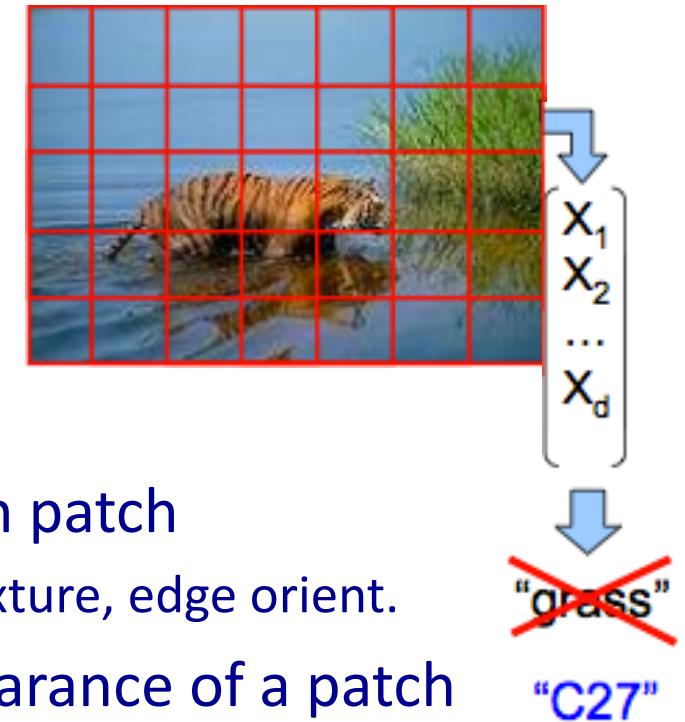
- Sample pairs x_i, x_j
 - ask human if x_i, x_j should be in the same group
 - easy task (cognitively)
 - can't ask them to "cluster" dataset manually
- System produces clusters
- Count errors, compute accuracy, F1, etc
 - FN: matching pairs x_i, x_j that are in different clusters (**e,h**)
 - FP: non-matching pairs x_i, x_j that are in same cluster (**c,d**)
- Doesn't require a pairing strategy
- Can handle overlapping clusters (a bit tricky)
 - same pair can count as both TN and FP (**g,h = No**)
- Can generate pairs from classes

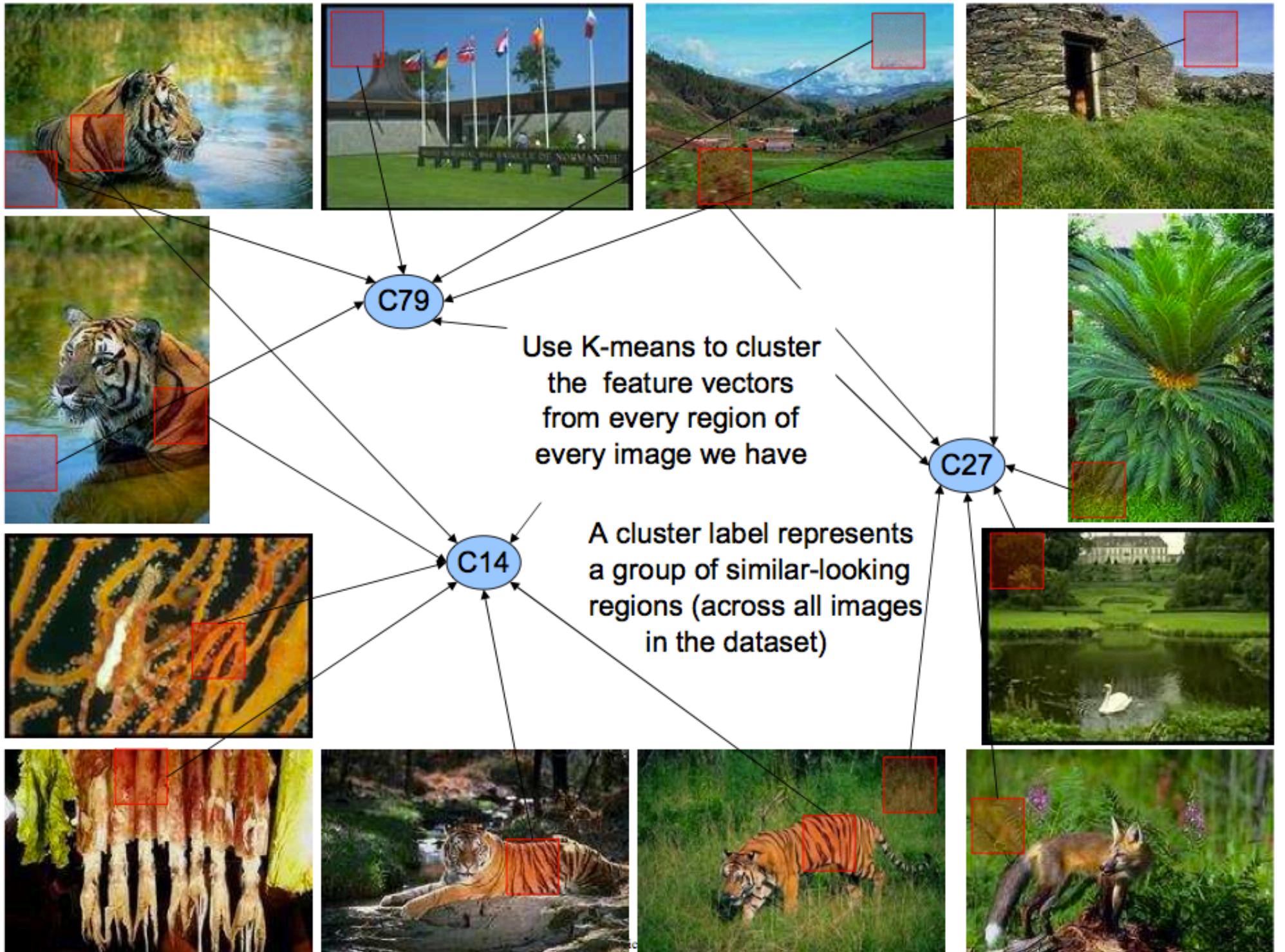
Copyright © 2014 Victor Lavrenko



Application: image representation

- Goal: detect presence / absence of objects in image
- First step: represent images as attribute-value pairs
 - pixels as attributes: $10^3 \times 10^3 \times 10^3$ (conservative)
 - large and not very meaningful for learning
 - bag-of-words would be nice
 - {"water", "grass", "tiger", "cat", "ripples"}
 - requires human annotation
 - break image into a set of patches
 - patch = part of some object
 - compute appearance features for each patch
 - relative position, distribution of colors, texture, edge orient.
 - convert to a “word” that reflects appearance of a patch
 - similar-looking feature vectors → same word to represent them



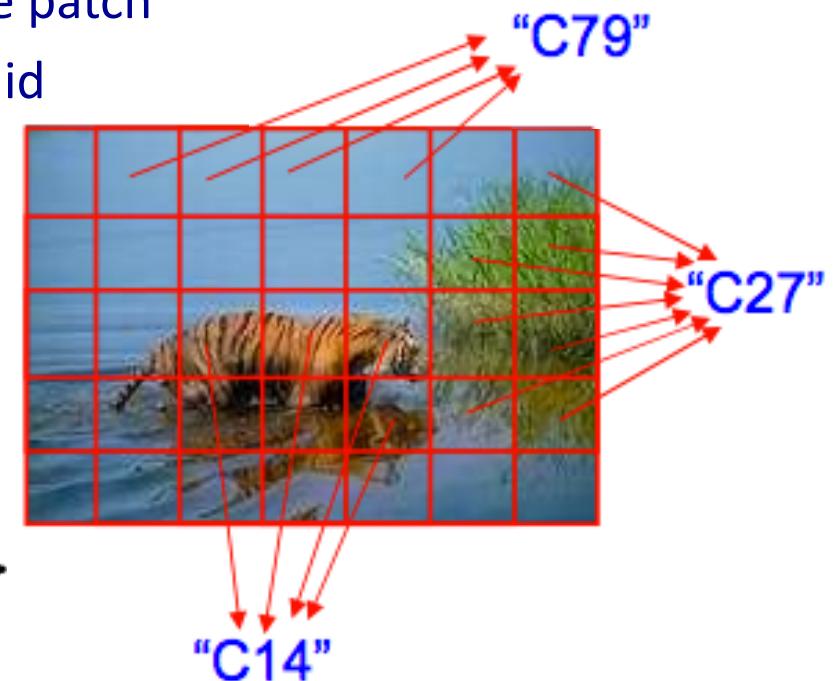


K-means for image representation

- What K-means does:
 - groups all feature vectors from all images into K clusters
 - provides a cluster id for every patch in every image
 - represents the salient properties of the patch
 - similar-looking patches have the same id
- Represent patch with cluster id
 - image = bag of cluster ids
 - one for each patch in the image
 - K-dimensional representation:

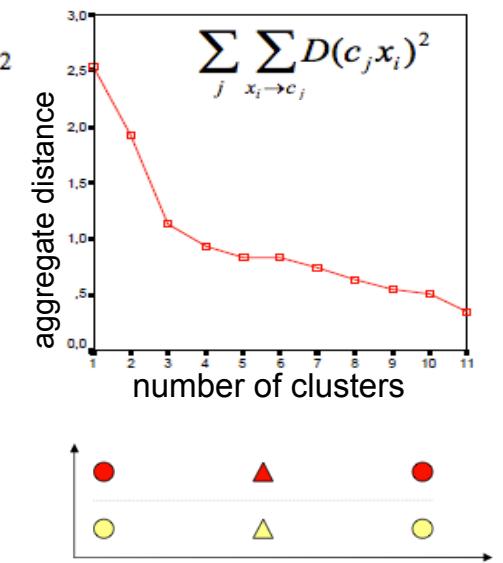
{ 4 x "C14", 7 x "C27", 24 x "C79", 0 x everything else }

- similar to bag-of-words
- cluster ids sometimes called vis-terms or “visual words”



Summary

- Clustering: discover underlying sub-populations
- K-means
 - fast, iterative method: $O(i*K*n*d)$
 - converges to a local minimum of $\sum_j \sum_{x_i \rightarrow c_j} D(c_j, x_i)^2$
 - run several times with different starting points
 - need to pick K: use scree plot
 - need to pick distance function (Euclidean)
 - nearby points may end up in diff. clusters
- Application: image representation
 - cluster image patches based on visual similarity
 - cluster numbers (vis-terms) becomes attributes
- Evaluation: intrinsic vs. extrinsic



Clustering: general structure

- Task: unsupervised / generative
 - group instances into K clusters
- Model structure
 - K cluster centroids (d -dimensional vectors)
- Score function
 - average distance from instance to cluster centre
- Optimization / search method
 - iteratively re-assign instances to clusters and update cluster centroids

IAML: Mixture models and EM

Victor Lavrenko and Nigel Goddard

School of Informatics

Semester 1

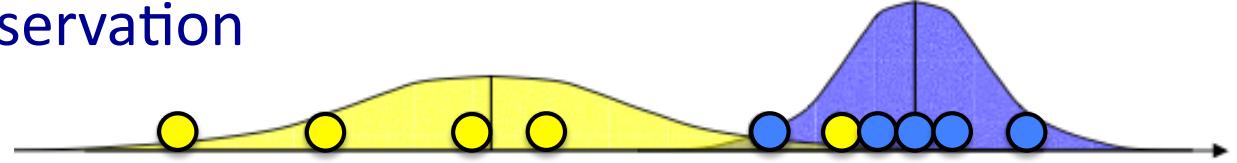
Mixture models

- Recall types of clustering methods
 - hard clustering: clusters do not overlap
 - element either belongs to cluster or it does not
 - soft clustering: clusters may overlap
 - strength of association between clusters and instances
- Mixture models
 - probabilistically-grounded way of doing soft clustering
 - each cluster: a generative model (Gaussian or multinomial)
 - parameters (e.g. mean/covariance are unknown)
- Expectation Maximization (EM) algorithm
 - automatically discover all parameters for the K “sources”

Mixture models in 1-d

- Observations $x_1 \dots x_n$
 - K=2 Gaussians with unknown μ, σ^2
 - estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_b)^2 + \dots + (x_{n_b} - \mu_b)^2}{n_b}$$



- What if we don't know the source?
- If we knew parameters of the Gaussians (μ, σ^2)
 - can guess whether point is more likely to be a or b

$$P(b|x_i) = \frac{P(x_i|b)P(b)}{P(x_i|b)P(b) + P(x_i|a)P(a)}$$

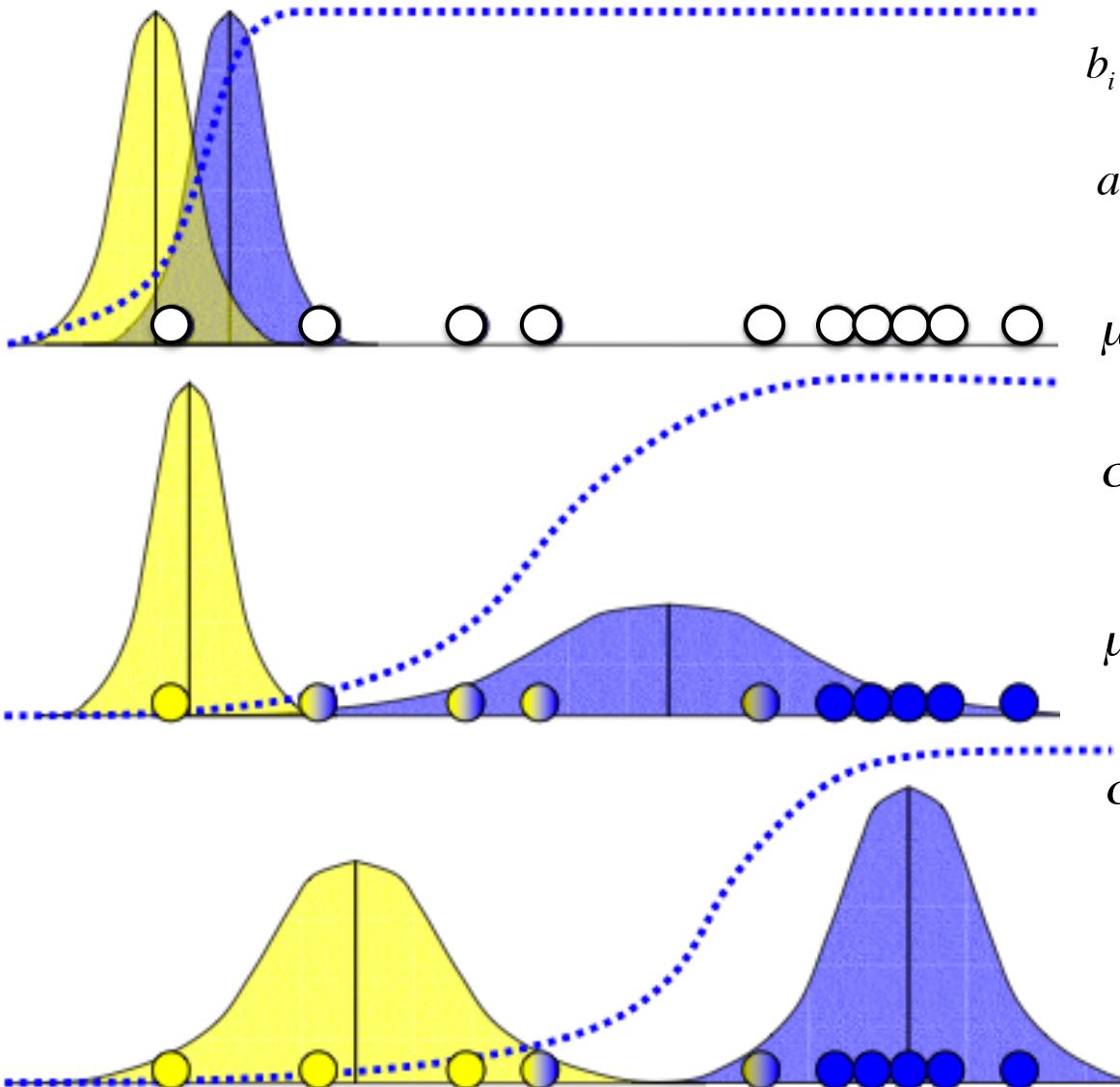
$$P(x_i|b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left\{-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right\}$$



Expectation Maximization (EM)

- Chicken and egg problem
 - need (μ_a, σ_a^2) and (μ_b, σ_b^2) to guess source of points
 - need to know source to estimate (μ_a, σ_a^2) and (μ_b, σ_b^2)
 - EM algorithm
 - start with two randomly placed Gaussians $(\mu_a, \sigma_a^2), (\mu_b, \sigma_b^2)$
- E-step: – for each point: $P(b|x_i) =$ does it look like it came from b?
- M-step: – adjust (μ_a, σ_a^2) and (μ_b, σ_b^2) to fit points assigned to them
 - iterate until convergence

EM: 1-d example



$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left\{-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right\}$$

$$b_i = P(b | x_i) = \frac{P(x_i | b)P(b)}{P(x_i | b)P(b) + P(x_i | a)P(a)}$$

$$a_i = P(a | x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \dots + b_n x_n}{b_1 + b_2 + \dots + b_n}$$

$$\sigma_b^2 = \frac{b_1(x_1 - \mu_b)^2 + \dots + b_n(x_n - \mu_b)^2}{b_1 + b_2 + \dots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \dots + a_n x_n}{a_1 + a_2 + \dots + a_n}$$

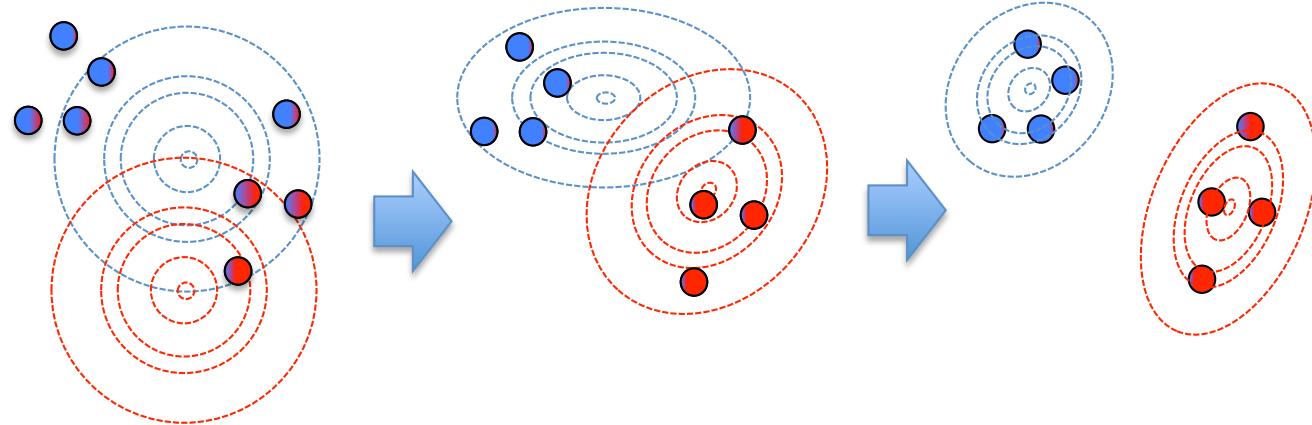
$$\sigma_a^2 = \frac{a_1(x_1 - \mu_a)^2 + \dots + a_n(x_n - \mu_a)^2}{a_1 + a_2 + \dots + a_n}$$

could also estimate priors:

$$P(b) = (b_1 + b_2 + \dots + b_n) / n$$

$$P(a) = 1 - P(b)$$

Gaussian Mixture Model



- Data with D attributes, from Gaussian sources $c_1 \dots c_k$
 - how typical is \mathbf{x}_i under source c
 - how likely that \mathbf{x}_i came from c
 - how important is \mathbf{x}_i for source c : $w_{i,c} = P(c | \vec{x}_i) / (P(c | \vec{x}_1) + \dots + P(c | \vec{x}_n))$
 - mean of attribute a in items assigned to c : $\mu_{ca} = w_{c1}x_{1a} + \dots + w_{cn}x_{na}$
 - covariance of a and b in items from c : $\Sigma_{cab} = \sum_{i=1}^n w_{ci} (x_{ia} - \mu_{ca})(x_{ib} - \mu_{cb})$
 - prior: how many items assigned to c : $P(c) = \frac{1}{n} (P(c | \vec{x}_1) + \dots + P(c | \vec{x}_n))$

How to pick K?

- Probabilistic model $L = \log P(x_1 \dots x_n) = \sum_{i=1}^n \log \sum_{k=1}^K P(x_i | k)P(k)$
 - tries to “fit” the data (maximize likelihood)
- Pick K that makes L as large as possible?
 - $K = n$: each data point has its own “source”
 - may not work well for new data points
- Split points into training set T and validation set V
 - for each K: fit parameters of T, measure likelihood of V
 - sometimes still best when $K = n$
- Occam’s razor: pick “simplest” of all models that fit
 - Bayes Inf. Criterion (BIC): $\max_p \{ L - \frac{1}{2} p \log n \}$
 - Akaike Inf. Criterion (AIC): $\min_p \{ 2 p - L \}$

L ... likelihood, how well our model fits the data
 p ... number of parameters how “simple” is the model

Summary

- Walked through 1-d version
 - works for higher dimensions
 - d-dimensional Gaussians, can be non-spherical
 - works for discrete data (text)
 - d-dimensional multinomial distributions (pLSI)
- Maximizes likelihood of the data: $P(x_1 \dots x_n) = \prod_{i=1}^n \sum_{k=1}^K P(x_i | k)P(k)$
- Similar to K-means
 - sensitive to starting point, converges to a local maximum
 - convergence: when change in $P(x_1 \dots x_n)$ is sufficiently small
 - cannot discover K (likelihood keeps growing with K)
- Different from K-means
 - soft clustering: instance can come from multiple “clusters”
 - co-variance: notion of “distance” changes over time
- How can you make GMM = K-means?

$$L = \log \prod_{i=1}^N P(x_i) = \sum_{i=1}^N \log \sum_{k=1}^K P(k) \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(x_i - \mu_k)^2}{2\sigma^2} \right\}$$

$$\frac{\partial L}{\partial \mu_j} = \sum_{i=1}^N \frac{p_j N(x_i; \mu_j, \sigma_j^2)}{\sum_{k=1}^K p_k N(x_i; \mu_k, \sigma_k^2)} \left\{ + \frac{2(x_i - \mu_k)}{2\sigma_j^2} \right\}$$

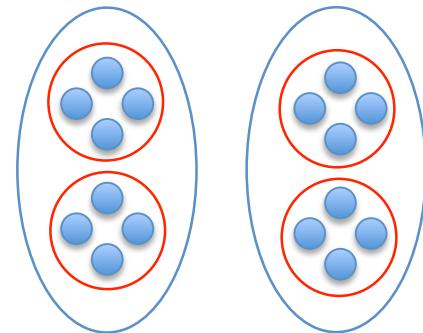
IAML: Hierarchical Clustering

Victor Lavrenko and Nigel Goddard

School of Informatics

Semester 1

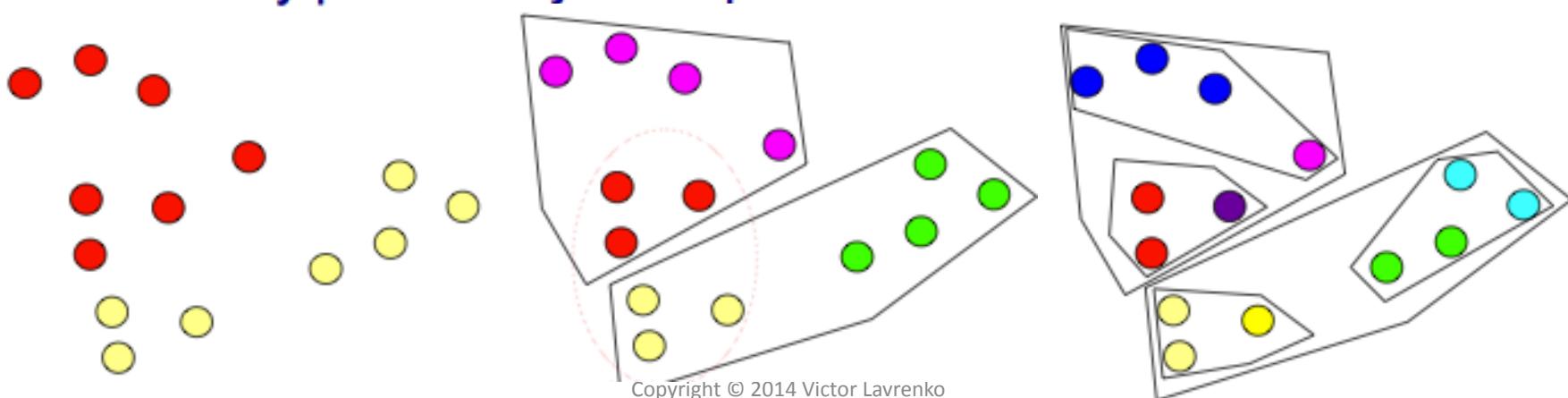
Hierarchical clustering



- Selecting K – question of granularity
 - how coarse or fine-grained is the structure in your data?
 - analogy: tidal waves or ripples on the surface?
 - real data: both, and probably everything in-between
 - no clustering algorithm able to pick K (some claim to)
- Instead of picking K – find a hierarchy of structure
 - top levels – coarse effects, low levels – fine-grained
 - topmost cluster – contains every point in the dataset
 - bottom level – set of n singleton clusters, one per data point
 - strategies:
 - top-down: start with all items in one cluster, split recursively
 - bottom-up: start with singletons, merge by some criterion

Hierarchical K-means

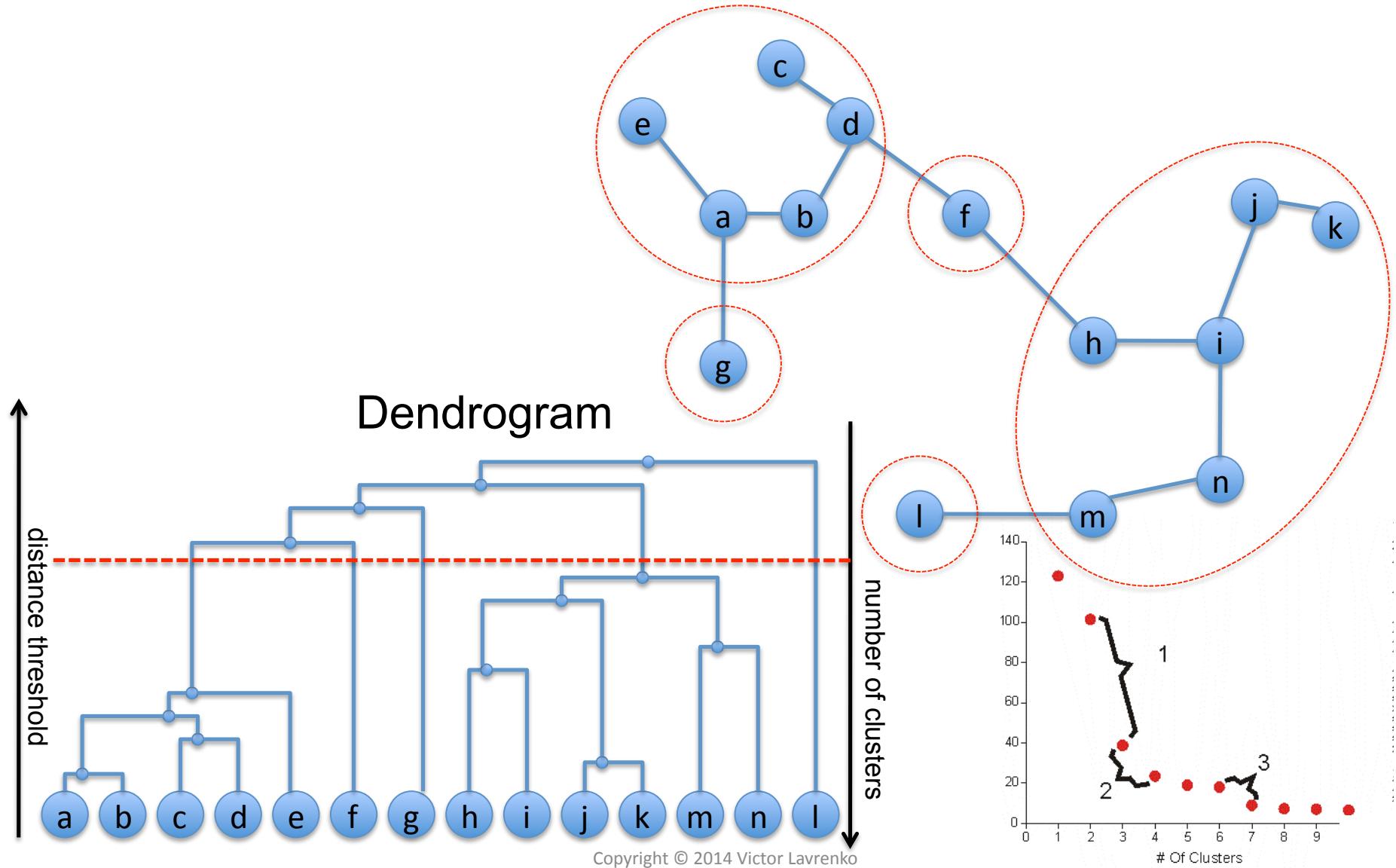
- Top-down approach:
 - run K-means algorithm on the original data $x_1 \dots x_n$
 - for each of the resulting clusters c_i : $i = 1 \dots K$
 - recursively run K-means on points in c_i
- Fast: recursive calls operate on a slice: $O(Knd \log_K n)$
- Greedy: can't cross boundaries imposed by top levels
 - nearby points may end up in different clusters



Agglomerative clustering

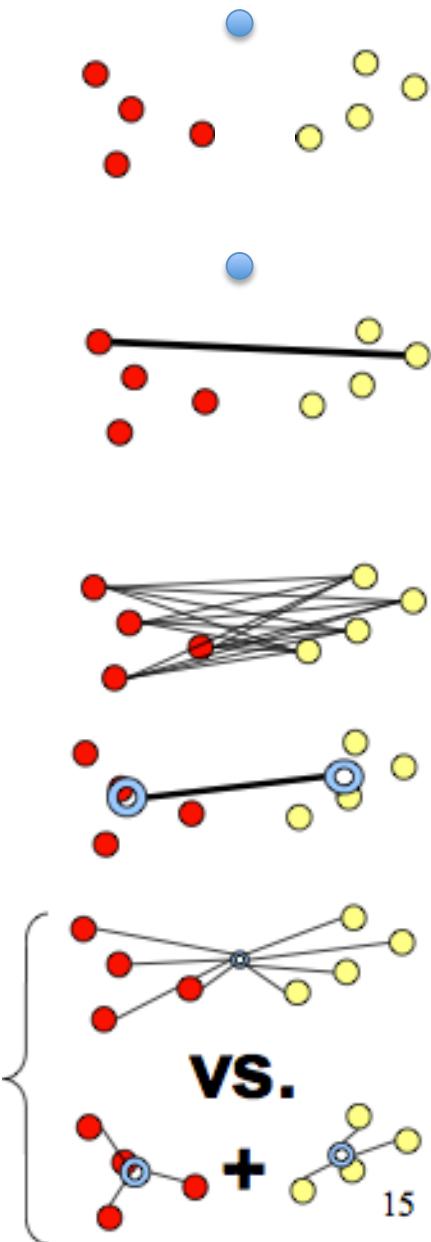
- Idea: ensure nearby points end up in the same cluster
- Start with a collection C of n singleton clusters
 - each cluster contains one data point: $c_i = \{x_i\}$
- Repeat until only one cluster is left:
 - find a pair of clusters that is closest: $\min_{i,j} D(c_i, c_j)$
 - merge the clusters c_i, c_j into a new cluster c_{i+j}
 - remove c_i, c_j from the collection C , add c_{i+j}
- Produces a dendrogram: hierarchical tree of clusters
- Need to define a distance metric over clusters
- Slow: $O(n^2d + n^3)$ – create, traverse distance matrix

Agglomerative clustering: example



Cluster distance measures

- Single link: $D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$
 - distance between closest elements in clusters
 - produces long chains a→b→c→...→z
- Complete link: $D(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$
 - distance between farthest elements in clusters
 - forces “spherical” clusters with consistent “diameter”
- Average link: $D(c_1, c_2) = \frac{1}{|c_1|} \frac{1}{|c_2|} \sum_{x_1 \in c_1} \sum_{x_2 \in c_2} D(x_1, x_2)$
 - average of all pairwise distances
 - less affected by outliers
- Centroids: $D(c_1, c_2) = D\left(\left(\frac{1}{|c_1|} \sum_{x \in c_1} \vec{x}\right), \left(\frac{1}{|c_2|} \sum_{x \in c_2} \vec{x}\right)\right)$
 - distance between centroids (means) of two clusters
- Ward's method: $TD_{c_1 \cup c_2} = \sum_{x \in c_1 \cup c_2} D(x, \mu_{c_1 \cup c_2})^2$
 - consider joining two clusters, how does it change the total distance (TD) from centroids?

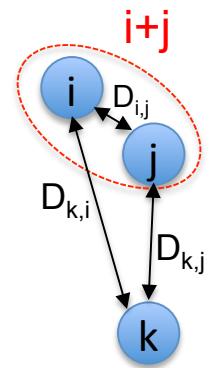


Lance-Williams Algorithm

- $D = \{D_{i,j} : \text{distance between } x_i \text{ and } x_j \text{ for } i,j=1..N\}$
- for N iterations:
 - $i,j = \arg \min D_{i,j}$... pair of closest clusters
 - add cluster: $i+j$, delete clusters i, j
 - for each remaining cluster k :

$$D_{k,i+j} = \alpha_i D_{k,i} + \alpha_j D_{k,j} + \beta D_{i,j} + \gamma |D_{k,i} - D_{k,j}|$$

Method	α_i	α_j	β	γ
Single linkage	0.5	0.5	0	-0.5
Complete linkage	0.5	0.5	0	0.5
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Weighted group average	0.5	0.5	0	0
Centroid	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward	$\frac{n_i}{(n_i+n_j+n_k)}$	$\frac{n_j}{(n_i+n_j+n_k)}$	$\frac{-n_k}{(n_i+n_j+n_k)}$	0



Single link:

$$D_{k,i+j} = \frac{1}{2} (D_{ki} + D_{kj} - |D_{ki} - D_{kj}|) = \min \{D_{ki}, D_{kj}\}$$

$$\min_{a,b} = \max_{a,b} - |a-b|$$

Summary

- Clustering: discover underlying sub-populations
- K-means
 - fast, iterative, leads to a local minimum
 - need to pick k : look for unusual reduction in variance
- Mixture models
 - probabilistic version of K-means
 - Expectation Maximization (EM) algorithm
- Hierarchical clustering
 - top-down (K-means) and bottom-up (agglomerative)
 - single / complete / average link variations

The Perceptron

Nigel Goddard
School of Informatics

A Simple Linear Algorithm

- ▶ Can we do something simpler than logistic regression?
And still be linear?
- ▶ For logistic regression we had this squashing function

$$f(z) = \sigma(z) \equiv 1/(1 + \exp(-z))$$

- ▶ What if we just have a step function?

$$f(z) = \text{sign}[z] = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- ▶ Notice that we call the classes $y \in \{-1, 1\}$. This is just for convenience later on.
- ▶ This architecture is called a *perceptron*, and has a very long history.

Classifying Using a Perceptron

- ▶ Like any other linear classifier. Given $\tilde{\mathbf{w}}$, w_0 and a \mathbf{x} to classify, do

$$\hat{y} = \begin{cases} 1 & \text{if } \tilde{\mathbf{w}}^T \mathbf{x} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- ▶ This is OK, but how are you going to train it?
- ▶ The problem is that you can't use gradient descent anymore.

The Perceptron Learning Rule

- ▶ The following rule was studied by Rosenblatt (1956)

repeat

```
for i in 1, 2, ... n  
     $\hat{y} \leftarrow \text{sign}[\mathbf{w}^T \mathbf{x}_i]$   
    if  $\hat{y} \neq y_i$   
         $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ 
```

until all training examples correctly classified

- ▶ Why does this make sense? Use same reasoning as logistic regression gradient.
- ▶ Say $y_i = 1$ and $\hat{y} = 0$. Then, after the update $\mathbf{w}^T \mathbf{x}_i$ gets bigger.

The Perceptron Learning Rule

- ▶ Amazing fact: If the data is linearly separable, the above algorithm always converges to a weight vector that separates the data.
- ▶ If the data is not separable, algorithm does not converge. Need to somehow pick which weight vector to go with.
- ▶ There are ways to do this (not examinable), such as the *averaged perceptron* and *voted perceptron*.
- ▶ This algorithm is a bit old and frumpy, but can still be very useful. Especially when you add kernels, to get the *kernel perceptron* algorithm.
- ▶ Also can be seen as a very simple neural network. see later.

Artificial Neural Networks

Charles Sutton
School of Informatics

Outline

- ▶ Why multilayer artificial neural networks (ANNs)?
- ▶ Representation Power of ANNs
- ▶ Training ANNs: backpropagation
- ▶ Learning Hidden Layer Representations
- ▶ Examples

What's wrong with the IAML course

When we write programs that “learn”, it turns out that we do and they don’t. —Alan Perlis

- ▶ Many of the methods in this course are linear. All of them depend on *representation*, i.e., having good features.
- ▶ What if we want to learn the features?
- ▶ This lecture: Nonlinear regression and nonlinear classification
- ▶ Can think of this as: A linear method where we learn the features
- ▶ These are motivated by a (weak) analogy to the human brain, hence the name *artificial neural networks*

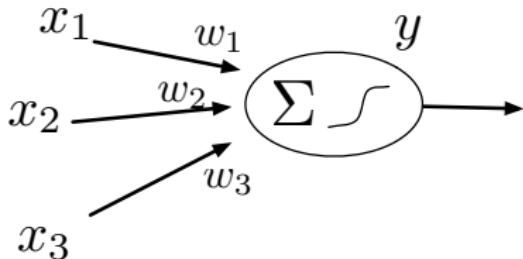
How artificial neural networks fit into the course

	Supervised	Unsupervised		
	Class.	Regr.	Clust.	D. R.
Naive Bayes	✓			
Decision Trees	✓			
k -nearest neighbour	✓			
Linear Regression		✓		
Logistic Regression	✓			
SVMs	✓			
k -means			✓	
Gaussian mixtures			✓	
PCA				✓
Evaluation				
ANNs	✓	✓		

Artificial Neural Networks (ANNs)

- ▶ The field of neural networks grew up out of simple models of neurons
- ▶ Each single neuron looks like a linear unit
- ▶ (In fact, *unit* is name for “simulated neuron”.)
- ▶ A network of them is nonlinear

Classification Using a Single Neuron



Take a single input $\mathbf{x} = (x_1, x_2, \dots, x_d)$. To compute a class label

1. Compute the neuron's activation
$$a = \mathbf{x}^\top \mathbf{w} + w_0 = \sum_{d=1}^D x_d w_d + w_0$$
2. Set the neuron output y as a function of its activation
$$y = g(a)$$
. For now let's say

$$g(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$

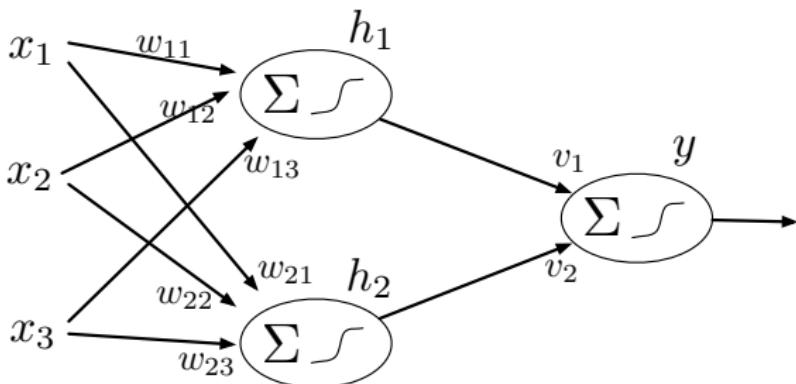
i.e., sigmoid

3. If $y > 0.5$, assign \mathbf{x} to class 1. Otherwise, class 0.

Why we need multilayer networks

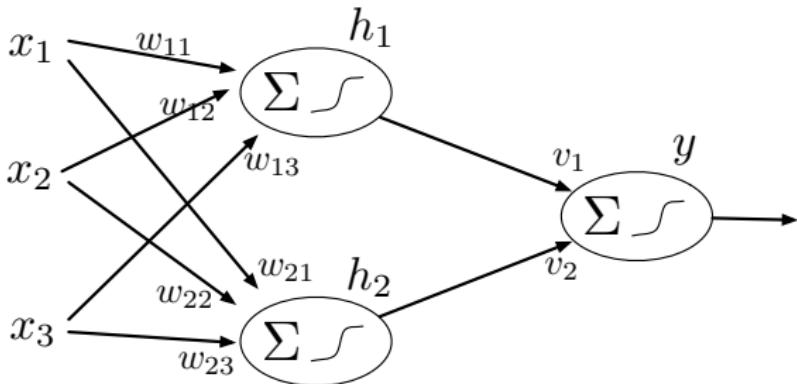
- ▶ We haven't done anything new yet.
- ▶ This is just a very strange way of presenting logistic regression
- ▶ Idea: Use recursion. Use the output of some neurons as input to another neuron that actually predicts the label

A Slightly More Complex ANN: The Units



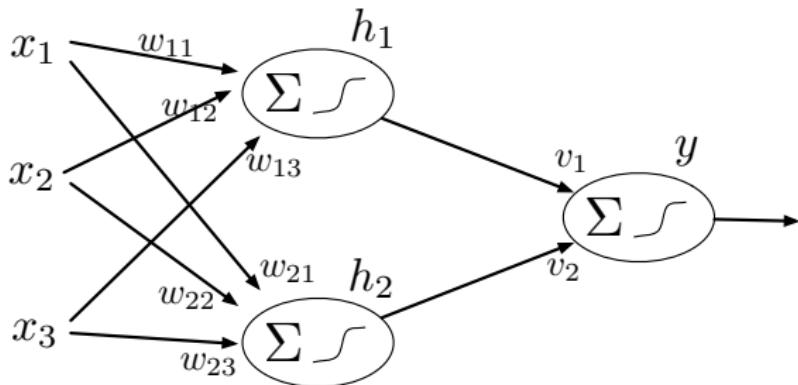
- ▶ x_1 , x_2 , and x_3 are the input features, just like always.
- ▶ y is the output of the classifier. In an ANN this is sometimes called an *output unit*.
- ▶ The units h_1 and h_2 don't directly correspond to anything in the data. They are called *hidden units*

A Slightly More Complex ANN: The Weights



- ▶ Each unit gets its own weight vector.
- ▶ $\mathbf{w}_1 = (w_{11}, w_{12}, w_{13})$ are the weights for h_1 .
- ▶ $\mathbf{w}_2 = (w_{21}, w_{22}, w_{23})$ are the weights for h_2 .
- ▶ $\mathbf{v} = (v_1, v_2)$ are the weights for y .
- ▶ Also each unit gets a “bias weight” w_{10} for unit h_1 , w_{20} for unit h_2 and v_0 for unit y .
- ▶ Use $\mathbf{w} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{v}, w_{10}, w_{20}, v_0)$ to refer to all of the weights stacked into one vector.

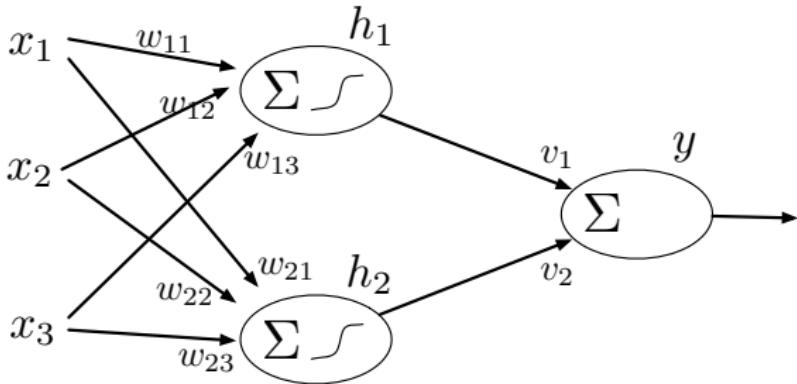
A Slightly More Complex ANN: Predicting



Here is how to compute a class label in this network:

1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^D w_{1d}x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^D w_{2d}x_d + w_{20})$
3. $y \leftarrow g(\mathbf{v}^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_0) = g(v_1h_1 + v_2h_2 + v_0)$
4. If $y > 0.5$, assign to class 1, i.e., $f(\mathbf{x}) = 1$. Otherwise $f(\mathbf{x}) = 0$.

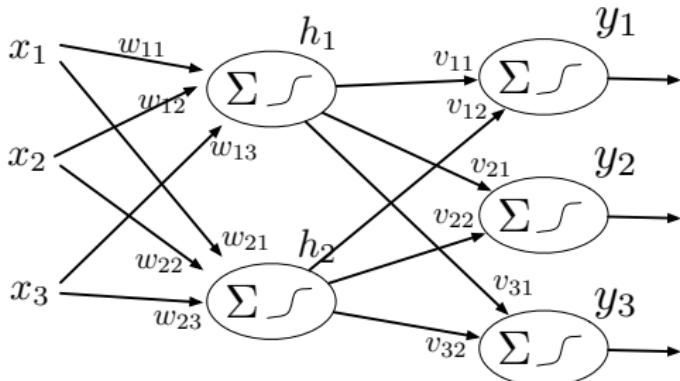
ANN for Regression



If you want to do regression instead of classification, it's simple. Just don't squash the output. Here is how to make a real-valued prediction:

1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^D w_{1d} x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^D w_{2d} x_d + w_{20})$
3. $y \leftarrow g_3(\mathbf{v}^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_0) = g_3(v_1 h_1 + v_2 h_2 + v_0)$
where $g_3(a) = a$ the identity function.
4. Return $f(\mathbf{x}) = y$ as the prediction of the real-valued output

ANN for Multiclass Classification



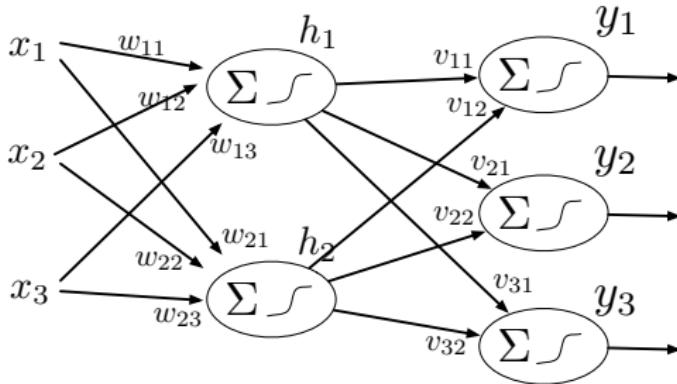
More than two classes? No problem. Only change is to output layer. Define one output unit for each class.

$y_i \leftarrow$ how likely it is that \mathbf{x} in class i , ($i = 1 \dots M$)

Then convert to probabilities using a softmax function.

$$p(y = m|\mathbf{x}) = \frac{e^{y_m}}{\sum_{k=1}^M e^{y_k}}$$

Multiclass ANN: Making a Prediction

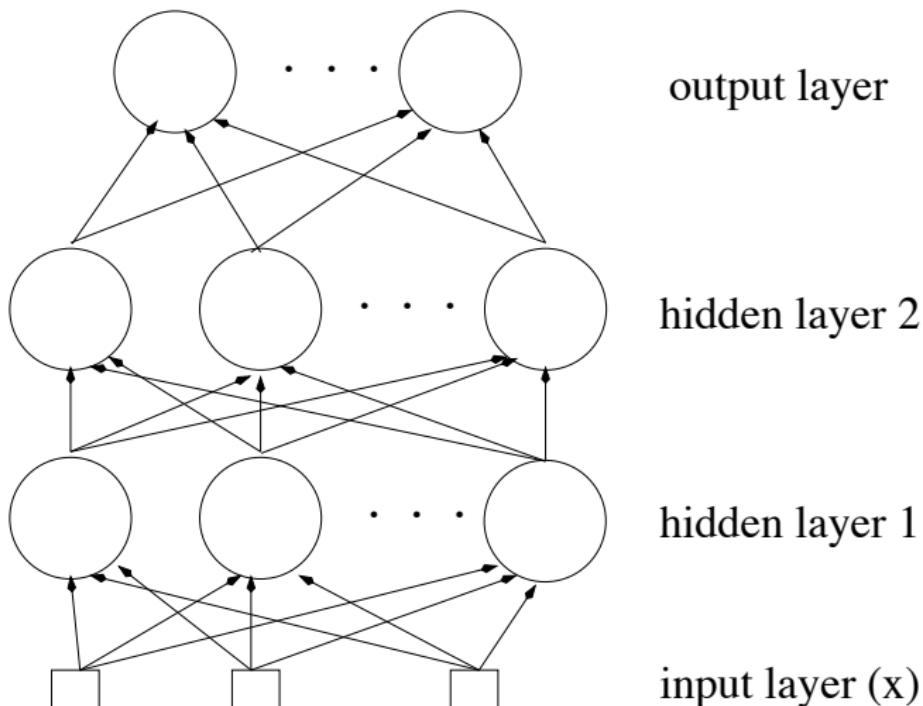


1. $h_1 \leftarrow g(\mathbf{w}_1^T \mathbf{x} + w_{10}) = g(\sum_{d=1}^D w_{1d} x_d + w_{10})$
2. $h_2 \leftarrow g(\mathbf{w}_2^T \mathbf{x} + w_{20}) = g(\sum_{d=1}^D w_{2d} x_d + w_{20})$
3. for all $m \in 1, 2, \dots, M$,
$$y_m \leftarrow \mathbf{v}_m^T \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} + v_{m0} = v_{m1} h_1 + v_{m2} h_2 + v_{m0}$$
4. Prediction $f(\mathbf{x})$ is the class with the highest probability

$$p(y = m | \mathbf{x}) = \frac{e^{y_m}}{\sum_{k=1}^M e^{y_k}}$$

$$f(\mathbf{x}) = \max_{m=1}^M p(y = m | \mathbf{x})$$

You can have more hidden layers and more units. An example network with 2 hidden layers



- ▶ There can be an arbitrary number of hidden layers
- ▶ The networks that we have seen are called *feedforward* because the structure is a directed acyclic graph (DAG).
- ▶ Each unit in the first hidden layer computes a non-linear function of the input \mathbf{x}
- ▶ Each unit in a higher hidden layer computes a non-linear function of the outputs of the layer below

Things that you get to tweak

- ▶ The structure of the network: How many layers? How many hidden units?
- ▶ What activation function g to use for all the units.
- ▶ For the output layer this is easy:
 - ▶ g is the identity function for a regression task
 - ▶ g is the logistic function for a two-class classification task
- ▶ For the hidden layers you have more choice:

$$g(a) = \sigma(a) \quad \text{i.e., sigmoid}$$

$$g(a) = \tanh(a)$$

$$g(a) = a \quad \text{linear unit}$$

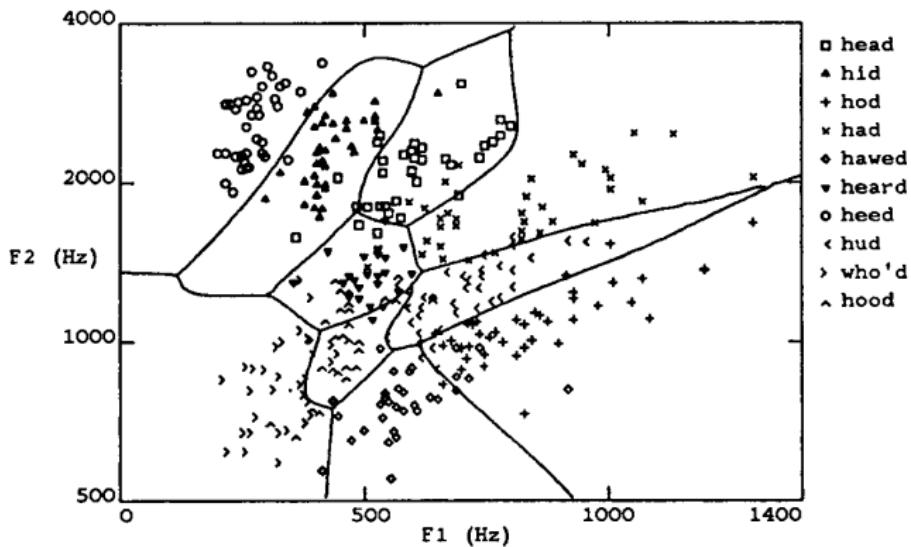
$$g(a) = \text{Gaussian density} \quad \text{radial basis network}$$

$$g(a) = \Theta(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases} \quad \text{threshold unit}$$

- ▶ Tweaking all of these can be a black art

Representation Power of ANNs

- ▶ Boolean functions:
 - ▶ Every boolean function can be represented by network with single hidden layer
 - ▶ but might require exponentially many (in number of inputs) hidden units
- ▶ Continuous functions:
 - ▶ Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
 - ▶ Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]. This follows from a famous result of Kolmogorov.
 - ▶ Neural Networks are *universal approximators*.
 - ▶ But again, if the function is complex, two hidden layers may require an extremely large number of units
- ▶ Advanced (non-examinable): For more on this see,
 - ▶ F. Girosi and T. Poggio. “Kolmogorov’s theorem is irrelevant.” *Neural Computation*, 1(4):465469, 1989.
 - ▶ V. Kurkova, “Kolmogorov’s Theorem Is Relevant”, *Neural Computation*, 1991, Vol. 3, pp. 617622.



ANN predicting 1 of 10 vowel sounds based on formats F1 and F2

Figure from Mitchell (1997)

Training ANNs

- ▶ Training: Finding the best weights for each unit
- ▶ We create an error function that measures the agreement of the target y_i and the prediction $f(\mathbf{x})$
- ▶ Linear regression, squared error: $E = \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2$
- ▶ Logistic regression (0/1 labels):
$$E = \sum_{i=1}^n y_i \log f(\mathbf{x}_i) + (1 - y_i) \log(1 - f(\mathbf{x}_i))$$
- ▶ It can make sense to use a regularization penalty (e.g. $\lambda|\mathbf{w}|^2$) to help control overfitting; in the ANN literature this is called *weight decay*
- ▶ The name of the game will be to find \mathbf{w} so that E is minimized.
- ▶ For linear and logistic regression the optimization problem for \mathbf{w} had a unique optimum; this is no longer the case for ANNs (e.g. hidden layer neurons can be permuted)

Backpropagation

- ▶ As discussed for logistic regression, we need the gradient of E wrt all the parameters \mathbf{w} , i.e. $\mathbf{g}(\mathbf{w}) = \frac{\partial E}{\partial \mathbf{w}}$
- ▶ There is a clever recursive algorithm for computing the derivatives. It uses the chain rule, but stores some intermediate terms. This is called *backpropagation*.
- ▶ We make use of the layered structure of the net to compute the derivatives, heading backwards from the output layer to the inputs
- ▶ Once you have $\mathbf{g}(\mathbf{w})$, you can use your favourite optimization routines to minimize E ; see discussion of gradient descent and other methods in Logistic Regression slides

Convergence of Backpropagation

- ▶ Dealing with local minima. Train multiple nets from different starting places, and then choose best (or combine in some way)
- ▶ Initialize weights near zero; therefore, initial networks are near-linear
- ▶ Increasingly non-linear functions possible as training progresses

Training ANNs: Summary

- ▶ Optimize over vector of all weights/biases in a network
- ▶ All methods considered find *local* optima
- ▶ Gradient descent is simple but slow
- ▶ In practice, second-order methods (*conjugate gradients*) are used for batch learning
- ▶ Overfitting can be a problem

Applications of Neural Networks

- ▶ Recognizing handwritten digits on cheques and post codes (LeCun and Bengio, 1995)
- ▶ Language modelling: Given a partial sentence “Neural networks are”, predict the next word (Y Bengio et al, 2003)
- ▶ Financial forecasting
- ▶ Speech recognition

ANNs: Summary

- ▶ Artificial neural networks are a powerful nonlinear modelling tool for classification and regression
- ▶ These were never very good models of the brain. But as classifiers, they work.
- ▶ The hidden units are new representation of the original input. Think of this as learning the features
- ▶ Trained by optimization methods making use of the backpropagation algorithm to compute derivatives
- ▶ Local optima in optimization are present, cf linear and logistic regression (and kernelized versions thereof, e.g. SVM)
- ▶ Ability to automatically discover useful hidden-layer representations

Neural Networks: Recent Developments

Chris Williams

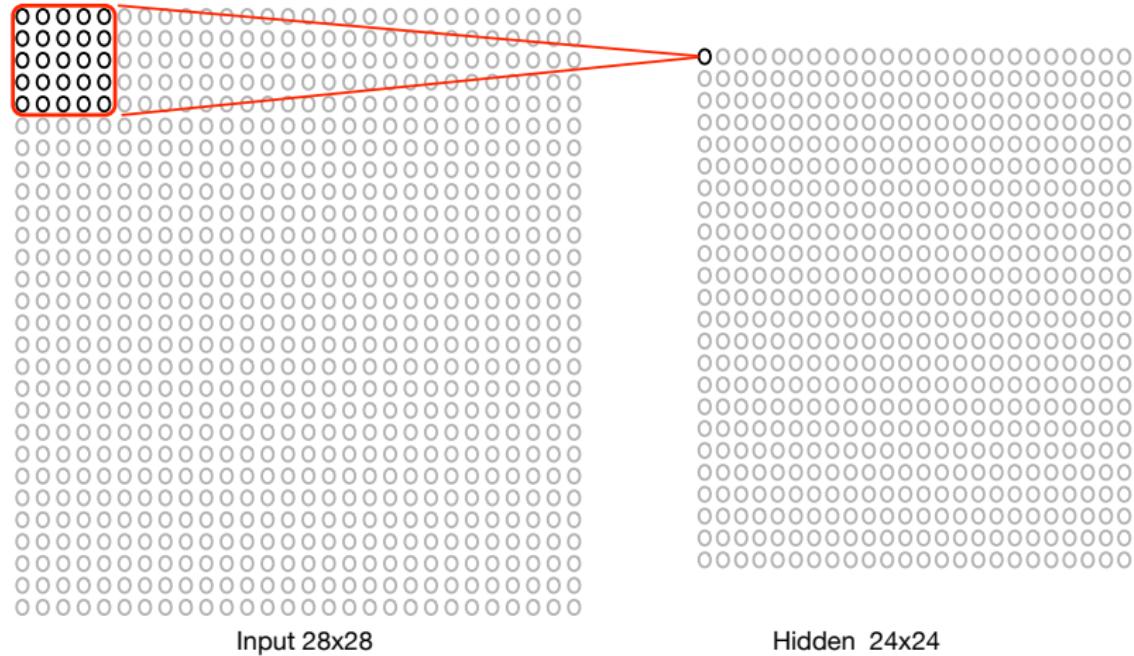
School of Informatics, University of Edinburgh, UK

- ▶ There was a wave of neural networks research from around 1985 to the early 1990s
- ▶ There is a lot of recent interest around “deep learning”, starting from around 2010
- ▶ For images: convolutional neural networks (CNNs)
- ▶ For sequences: recurrent neural networks

Convolutional Neural Networks – Why?

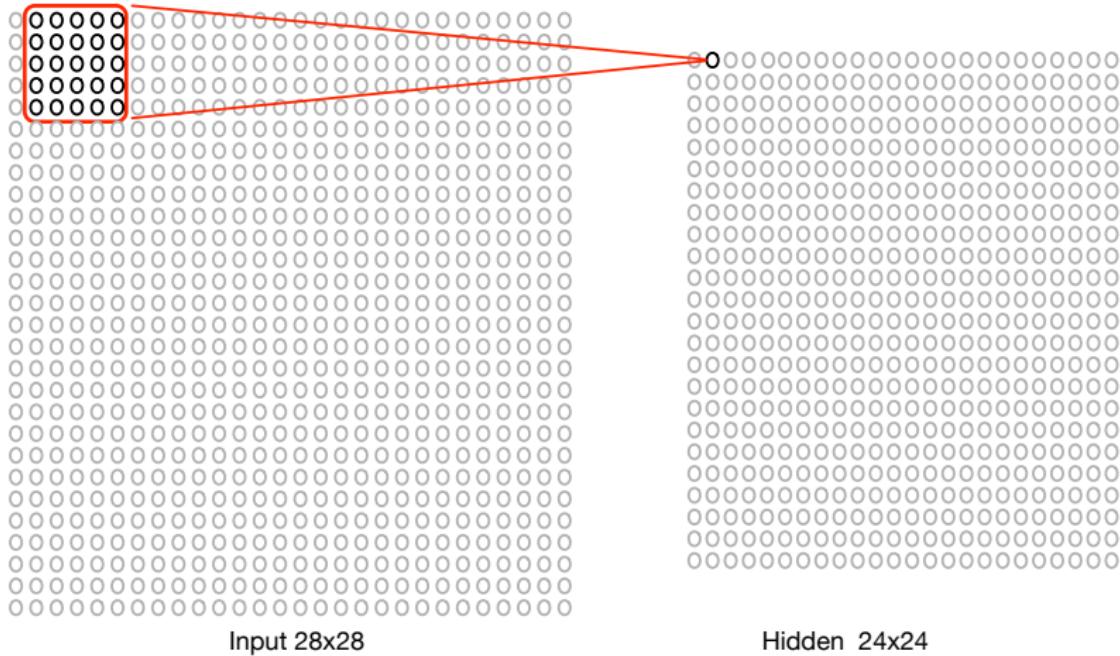
- ▶ Fully connected networks with high-dimensional inputs have a *lot* of parameters
- ▶ E.g. one hidden unit that gets input from a 200×200 image requires 40,000 weights
- ▶ This gives too many parameters to learn
- ▶ The spatial structure of the input image is ignored

Local receptive fields



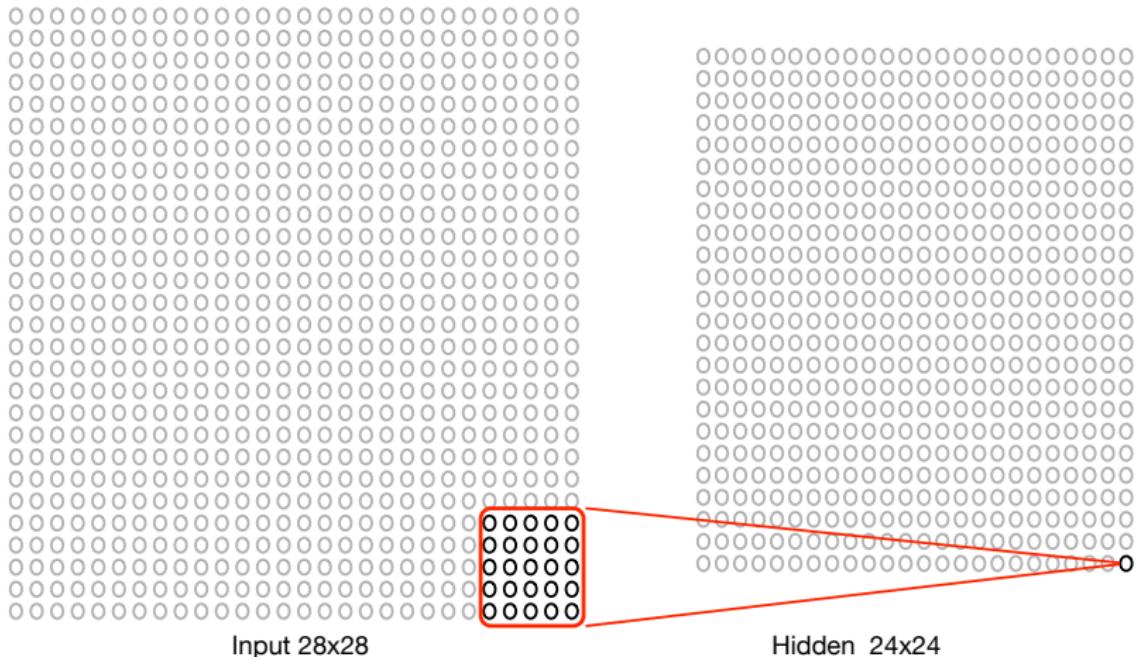
: Figure credit: Prof Steve Renals

Local receptive fields



: Figure credit: Prof Steve Renals

Local receptive fields



: Figure credit: Prof Steve Renals

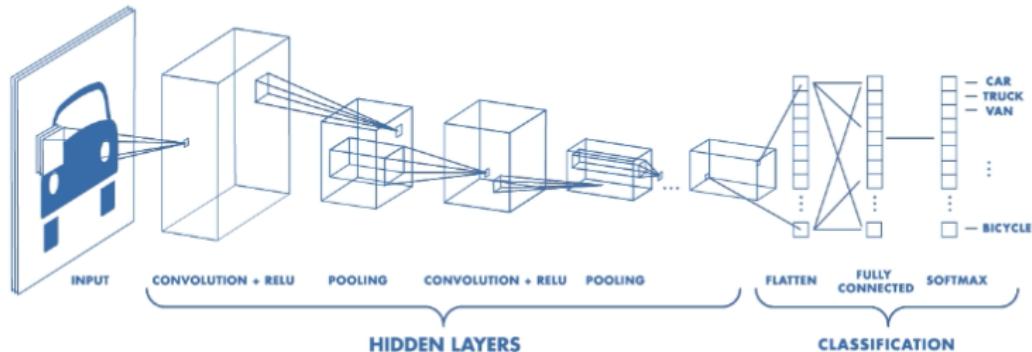
Convolutional Feature Detection

- ▶ Constrain each hidden unit $h_{i,j}$ to extract the feature by sharing weights across the receptive fields
- ▶ For hidden unit $h_{i,j}$ and $m \times m$ weight matrix W

$$h_{i,j} = \text{sigmoid} \left(\sum_{k=0}^{m-1} \sum_{\ell=0}^{m-1} w_{k,\ell} x_{i+k, j+\ell} + b \right)$$

- ▶ The output is a *feature map*
- ▶ Many different weight matrices can be used, to produce multiple feature maps

Convolutional Neural Networks (CNNs)

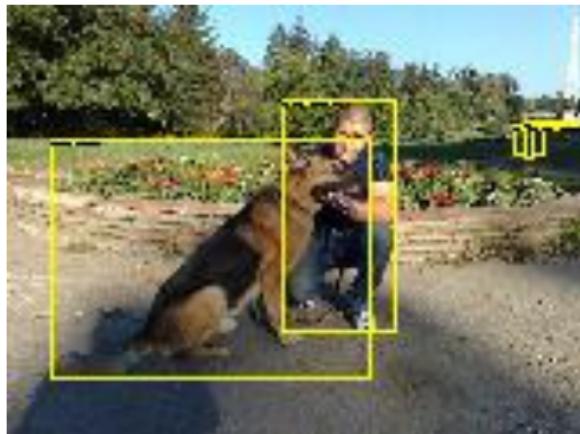


: Figure credit: <https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks-1489512765771.html>

- ▶ Recent wave started with “AlexNet” by Krizhevsky, Sutskever, Hinton (2012), although similar architectures can be traced back decades earlier

Tasks:

- ▶ Object classification: is there an X in this image?
- ▶ Object localization: put a bounding box around each X in the image
- ▶ Instance segmentation: identify the pixels belonging to each X in the image



Recurrent Neural Networks (RNNs)

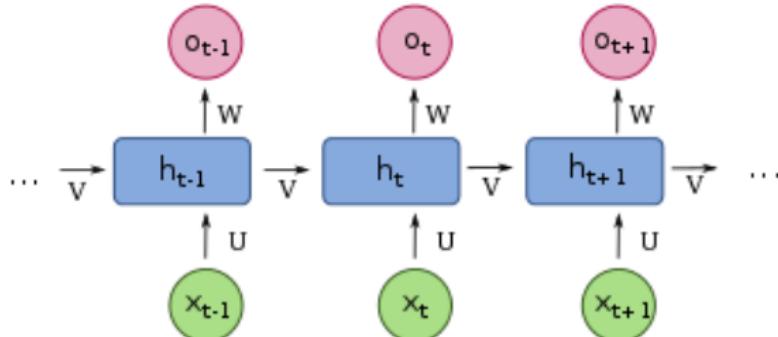


Figure Credit: François Deloche: CC-BY-SA-4.0

- ▶ A model for sequence data (e.g. time series)
- ▶ Example use: annotation of motion-capture data to label action type
- ▶ Example use: part-of-speech tagging for natural language
- ▶ Many complex network architectures, e.g. long short-term-memories (LSTMs, Hochreiter and Schmidhuber, 1997)



Unless explicitly stated otherwise, all material is copyright
©The University of Edinburgh 2016-19.