Introduction to machine learning and pattern recognition

Coryn Bailer-Jones
What is machine learning?

- **Data description and interpretation**
  - finding simpler relationship between variables (predictors and responses)
  - discovering “natural” groups or latent parameters in data
  - relating observables to physical quantities

- **Prediction**
  - capturing relationship between “inputs” and “outputs” for a set of labelled data with the goal of predicting outputs for unlabelled data (“pattern recognition”)

- **Learning from data**
  - dealing with noise
  - coping with high dimensions (many potentially relevant variables)
  - fitting models to data
  - generalizing solutions
Parameter estimation from stellar spectra

- learn mapping from spectra to data using labelled examples
- multidimensional (few hundred)
- nonlinear
- inverse
Source classification

- distinguish between stars, galaxies, quasars, based on colours or low-resolution spectroscopy
- can simulate spectra of known classes
- much variance within these basic classes
Course objectives

- learn the basic concepts of machine learning
- learn about some machine learning algorithms
  - classification
  - regression
  - clustering
- appreciate why and when these methods are required
- provide some understanding of techniques used in the literature
- promote use of useful techniques in your research
- introduction to R
Course methodology

- emphasis is on
  - principles
  - specific techniques
- some examples
- some maths is essential, but few derivations
- slides both contain more than is covered and are incomplete
- R scripts on web page
Course content (nominal plan)

1) supervised vs. unsupervised learning; linear methods; nearest neighbours; curse of dimensionality; regularization & generalization; k-means clustering
2) density estimation; linear discriminant analysis; nonlinear regression; kernels and basis functions
3) neural networks; separating hyperplanes; support vector machines
4) principal components analysis; mixture models; model selection; self-organizing maps
R (S, S-PLUS)

- http://www.r-project.org
- “a language and environment for statistical computing and graphics”
- open source
- runs on Linux, Windows, MacOS
- top-level operations for vectors and matrices
- OO-based
- large number of statistical and machine learning packages
- can link to external code (e.g. C, C++, Fortran)
- Good book on using R for statistics
  - “MASS4”
Two types of learning problem

- **supervised learning**
  - predictors ($x$) and responses ($y$)
  - infer $P(y \mid x)$, perhaps modelled as $f(x ; w)$
  - learn model parameters, $w$, by *training* on labelled data
  - discrete $y$ is a classification problem; real-valued is regression
  - *examples: trees, nearest neighbours, neural networks, SVMs*

- **unsupervised learning**
  - no distinction between predictors and responses
  - infer $P(x)$, or things about this
    - e.g. no. of modes/classes (mixture modelling, peak finding)
    - low dimensional projections (descriptions)
    - outlier detection (discovery)
  - *examples: PCA, ICA, k-mean clustering, MDS, SOM*
Linear regression and least squares

N data vectors, each of p dimensions. Regress against response variable, y
Data: \( x_i, y_i \quad x = \{x_1, x_2, \ldots, x_j, \ldots, x_p\} \)
Model: \( y = x \beta \)

Least squares solution:
\[
\hat{\beta} = \min_{\beta} \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{p} x_{i,j} \beta_j \right)^2
\]
In matrix form this is
\[
RSS = (y - X \beta)^T (y - X \beta)
\]
minimize w.r.t \( \beta \) and the solution is
\[
\hat{\beta} = (X^T X)^{-1} X^T y
\]
Linear regression and linear least squares: more details

Fit a linear model to a set of data \( \{y, x\} \)

\[
\hat{y} = \beta_0 + \sum_{j=1}^{p} x_j \beta_j = x^T \beta \quad x, \beta \text{ are } p \times 1 \text{ column vectors}
\]

Determine parameters by minimizing sum-of-squares error on all \( N \) training data

\[
RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2
\]

\[
\min_{\beta} \|y - X^T \beta\|_2 = \min_{\beta} (y - X \beta)^T (y - X \beta) \quad X \text{ is a } N \times p \text{ matrix}
\]

This is quadratic in \( \beta \) so always has a minimum. Differentiate w.r.t \( \beta \)

\[
X^T (y - X \beta) = 0
\]

\[
X^T X \beta = X^T y
\]

If \( X^T X \) (the "information matrix") is non-singular then the unique solution is

\[
\hat{\beta} = (X^T X)^{-1} X^T y
\]

The predicted values are

\[
\hat{y} = X \hat{\beta} = X (X^T X)^{-1} X^T y
\]

\[
H = X (X^T X)^{-1} X^T \quad \text{is sometimes called the 'hat' matrix}
\]
Linear regression

- See R scripts on web page
  - taken from section 1.3 of Venables & Ripley
Model testing: cross validation

- Solve for model parameters using a training data set, but must evaluate performance on an independent set to avoid “overtraining”
- To determine generalization performance and (in particular) to optimize free parameters we can split/sample available data
  - train/test sets
  - N-fold cross-validation (N models)
  - train, test and evaluation sets
  - bootstrapping
Nearest neighbours ($\chi^2$ minimization)

New object: $s = (s_1, s_2, \ldots, s_i, \ldots, s_I)$

Assign class (or parameters) of nearest template in a labelled grid of $K$ object templates $\{x^{(k)}\} \quad k = 1..K$

$$\min_k \sum_i (s_i - x_i^{(k)})^2$$

In general can apply weights to each dimension:

$$\min_k \sum_i \left( \frac{s_i - x_i^{(k)}}{w_i} \right)^2$$

$\chi^2$ minimization is case where the weights estimate the s.d. of the noise.

In k-nn estimate parameters from average of k nearest neighbours.
2-class classification: K-nearest neighbours

**FIGURE 2.2.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (GREEN = 0, RED = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.

**FIGURE 2.3.** The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (GREEN = 0, RED = 1), and then predicted by 1-nearest-neighbor classification.
Classification via linear regression

\[ f_G(x) = \hat{\beta}_k^T x \]

parameters estimated by least squares
\[ \min_\beta \| f - X^T \beta \|_2 \]

\[ f_{G_1}(x) = 0 \quad \text{for green class} \]
\[ f_{G_2}(x) = 1 \quad \text{for red class} \]

Boundary is
\[ f_{G_1}(x) = f_{G_2}(x) = 0.5 \]

**FIGURE 2.1.** A classification example in two dimensions. The classes are coded as a binary variable—GREEN = 0, RED = 1—and then fit by linear regression. The line is the decision boundary defined by \( x^T \hat{\beta} = 0.5 \). The red shaded region denotes that part of input space classified as RED, while the green region is classified as GREEN.
Classification via linear regression: more details

\( K \) classes, represented by \( K \) indicators, \( Y_k, k=1, \ldots, K \) with
\[ Y_k = 1 \text{ if } G = k \text{ else } Y_k = 0 \]
\( Y \) is the \( N \times K \) indicator response matrix.
Fitting linear regression model to columns of \( Y \) simultaneously gives
\[ \hat{Y} = X B = X (X^T X)^{-1} X^T Y \]
where \( B \) is a \((p+1) \times K\) coefficient matrix.
To classify a new observations with input \( x \) do
1. \( y = [(1, x) B]^T \) \( 1 \times K \) vector
2. \( \hat{G}(x) = \arg\max_k y_k(x) \)

Can interpret/motivate probabilistically:
\[ E(Y_k | X = x) = \Pr(G = k | X = x) \]
Comparison

- **Linear model**
  - makes a very strong assumption about the data viz. well-approximated by a globally linear function
    - stable but biased
  - learn relationship between \((X, y)\) and encapsulate into parameters, \(\beta\)

- **K-nearest neighbours**
  - no assumption about functional form of relationship \((X, y)\), i.e. it is nonparametric
  - but does assume that function is well-approximated by a locally constant function
    - less stable but less biased
  - no free parameters to learn, so application to new data relatively slow: brute force search for neighbours takes \(O(N)\)
Which solution is optimal?

- if we know nothing about how the data were generated (underlying model, noise), we don't know
- if data drawn from two uncorrelated Gaussians: linear decision boundary is “optimal”
- if data drawn from mixture of multiple distributions: linear boundary not optimal (nonlinear, disjoint)
- what is optimal?
  - smallest generalization errors
  - simple solution (interpretability)
- more complex models permit lower errors on training data
  - but we want models to generalize
  - need to control complexity / nonlinearity (regularization)
Learning, generalization and regularization

See R scripts on web page
Bias-Variance decomposition

- Error in a fit or prediction can be divided into three components:
  - Error = \text{bias}^2 + \text{variance} + \text{irreducible error}
- Bias measures the degree to which our estimates typically differ from the truth
- Variance is the extent to which our estimates vary or scatter (e.g. as a result of using slightly different data, small changes in the parameters etc.)
  - expected standard deviation of estimate around mean estimate
- In practice permitting a small amount of bias in a model can lead to large reduction in variance (and thus total error)
Bias-variance trade-off

local polynomial regression
with a variable kernel
using
\texttt{locpoly}\{\texttt{KernSmooth}\}

black line: true function
black points: data set

\textbf{Blue line}: $h = 0.02$
“complex”, “rough”
low bias, high variance

\textbf{Red line}: $h = 0.5$
“simple”, “smooth”
high bias, low variance

C.A.L. Bailer-Jones. \textit{Machine learning and pattern recognition}
Bias and variance

![Graph showing bias and variance](image)

- High Bias
  - Low Variance
- Low Bias
  - High Variance

Prediction Error vs. Model Complexity

- Test Sample
- Training Sample
Limitations of nearest neighbours (and $\chi^2$ minimization)

- how do we determine appropriate weights for each dimension?
  - need a “significance” measure, which noise s.d. is not
- limited to constant number of neighbours (or constant volume)
  - no adaptation depending on variable density of grid
- problematic for multiple parameter estimation
  - “strong” vs. “weak” parameters
- curse of dimensionality

$$\min_k \sum_i \left( \frac{S_i - x_i^{(k)}}{w_i} \right)^2$$
The curse of dimensionality

Data uniformly distributed in unit hypercube
Define neighbour volume with edge length $x$ ($x<1$)
neighbour volume = $x^p$
p = no. of dimensions
f = fraction of unit data volume covered by
    neighbours volume. Thus $x = f^{1/p}$

- for $p=10$, to capture 1% of data must
  cover 63% of range of each input
  variable (95% for $p=100$)
- as $p$ increases
  - distance to neighbours increases
  - most neighbours are near boundary
- to maintain density (i.e. properly sample variance), number of templates must
  increase as $N^p$
Overcoming the curse

- Avoid it by dimensionality reduction
  - throw away less relevant inputs
  - combine inputs
  - use domain knowledge to select/define features
- Make assumptions about the data
  - structured regression
    - this is essential: an infinite number of functions pass through a finite number of data points
  - complexity control
    - e.g. smoothness in a local region
K-means clustering

- group data into a pre-specified number of clusters which minimize within-class RMS about each cluster centre
- algorithm
  1. initialize K cluster centres
  2. assign each point to the nearest cluster
  3. recalculate cluster centres as the mean of the member coordinates
  4. iterate steps 2 and 3 until cluster centres no longer change
- R script: `kmeans{stats}`
- Variations
  - \textit{k-medoids}: only need dissimilarity measures (and not data) if we confine class centers to the set of vectors. R scripts are `pam,clara{cluster}`
K-means clustering on the swiss data
...now with different starting vectors
K-means clustering produces a Voronoi tessellation

...if you use a Euclidean distance metric

K-means is also just a special case of mixture modelling ... how?
Summary

- supervised and unsupervised methods
  - former are fit (trained) on a labelled data set
- need adaptive methods which learn the significance of the input data in predicting the outputs
- need to *regularize* fitting in order to achieve *generalization*
  - trade-off between fit *bias* and *variance*
- curse of dimensionality
  - in practice must make assumptions (e.g. smoothness) and much of machine learning is about how to do this
- unsupervised method
  - find k clusters to reduce within-class variance