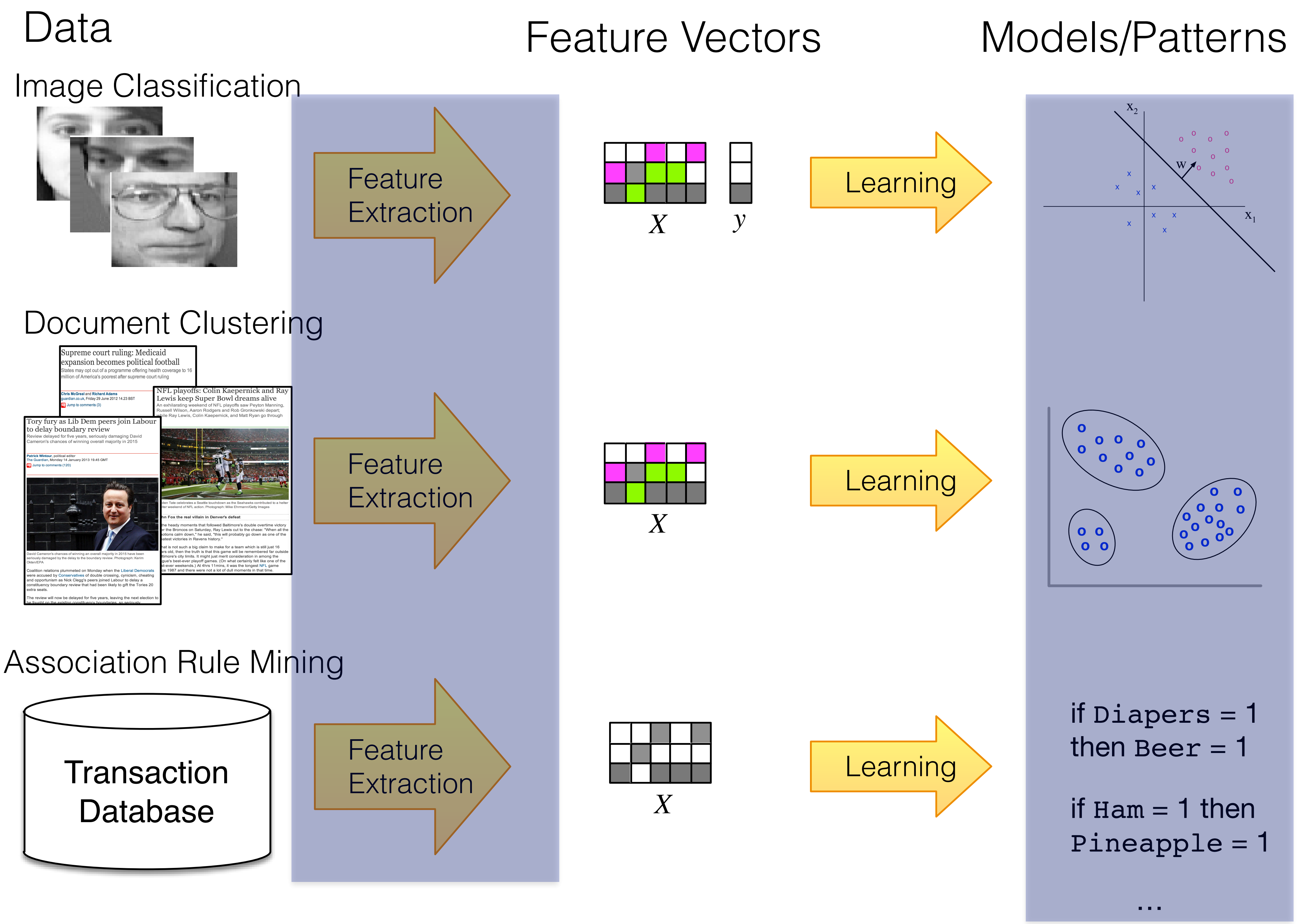
IRDS: Choosing Features

Charles Sutton
University of Edinburgh

Why features?

- Every learning algorithm somehow assumes:
  - “similar input vectors have similar labels”
- **Features** determine what is similar
- For practical ML, two best ways to improve performance
  - Get more training data
  - Come up with better features
  - (For ML research, advice would be different!)
- Feature engineering is a way to introduce prior knowledge about the problem

Two Representation Problems



1. Given input, what is the feature vector?

2. What is the set of possible models?

Two Representation Problems

1. What features to use
2. What is the space of possible models

- In these lectures, we discuss features.
    - For model, see —> IAML, PMR, MLPR
- But: To pick features, must understand model.
- See bonus slides

General Principles

- Feature engineering is **iterative** (and messy)
  - Come up with a new feature
  - Try it on a validation set, measure error
  - Repeat
- Use an **ablative** design (NB gains don't always accumulate nicely)

Feature Set A	70%
Feature Set A+B	75%
Feature Set A+B+C	75.2%

- Use **error analysis**
  - Look at the most embarrassing mistakes
  - What features might help with those
- Training set versus validation set versus test set
  - Once you have tuned features on a data set, you can't use the error to predict future performance
- Flexibility versus overfitting

In this lecture

- Focus on **general tricks** that help in many domains
  - Normally when the features are already “somewhat meaningful”
- In particular, we’ll talk about tricks for:
  - categorical features
  - continuous features
  - nonlinear features
  - features “computed” from other processes
  - cheap and cheerful transfer learning

1-of-K ("one hot") encoding

Age	Fav. Colour	Label
26		+
57		-
34		+

Age	Fav. Colour	Label
26	0	+
57	1	-
34	2	+

This can cause problems.
(Is yellow really twice as related to label as blue?)

Age	Red?	Yellow?	Blue?	Label
26	1	0	0	+
57	0	0	1	-
34	0	1	0	+

Convert to K binary features
("1-of-K" or "one hot" encoding)

For which algorithms will this matter?

Normalization (Whitening)

For continuous features, can be best to have zero mean and unit variance

Feature index *k*

Sample mean, feature *k*

$$x_k \mapsto \frac{x_k - \hat{\mu}_k}{\hat{\sigma}_k}$$

Sample standard deviation

or in vector notation

Entire training example

$$\mathbf{x} \mapsto \Sigma^{-1/2}(\mathbf{x} - \hat{\mu})$$

Sample covariance matrix (or: just diagonal entries)

For which algorithms will this matter?

Binning (Discretization)

We've mentioned nonlinear feature transforms

$$x_k \;\mapsto\; x_k^2$$

What if you do not expect a simple functional form is appropriate?

One possibility: Convert to M binary variables

$$x_k \;\mapsto\; \begin{pmatrix} \mathbb{I}\{x_k \in (-\infty, \tau_1]\} \\ \mathbb{I}\{x_k \in (-\tau_1, \tau_2]\} \\ \vdots \\ \mathbb{I}\{x_k \in (\tau_{M-1}, \infty)\} \end{pmatrix}$$

Feature Conjunctions

If features binary, natural interpretation:

- each feature is a proposition, e.g.
“does document *i* contain the word ‘geranium’”

Then, why not combine different features?, e.g.,

- “does document *i* contain both the word ‘geranium’ and ‘magnolia’”

This is a product of feature values, i.e.,

$$\begin{pmatrix} x_j \\ x_k \end{pmatrix} \mapsto \begin{pmatrix} x_j \\ x_k \\ x_j x_k \end{pmatrix}$$

In principle we could do this for all pairs (or higher).
Might reduce this using feature selection.

Sequences of Predictions

Examples:

- Predict part of speech for each word in a sentence
- Predict number of web requests for each day
- Predict for each window of an image whether it contains a face

For these, think about features

- At different “lags”
- At different levels of granularity

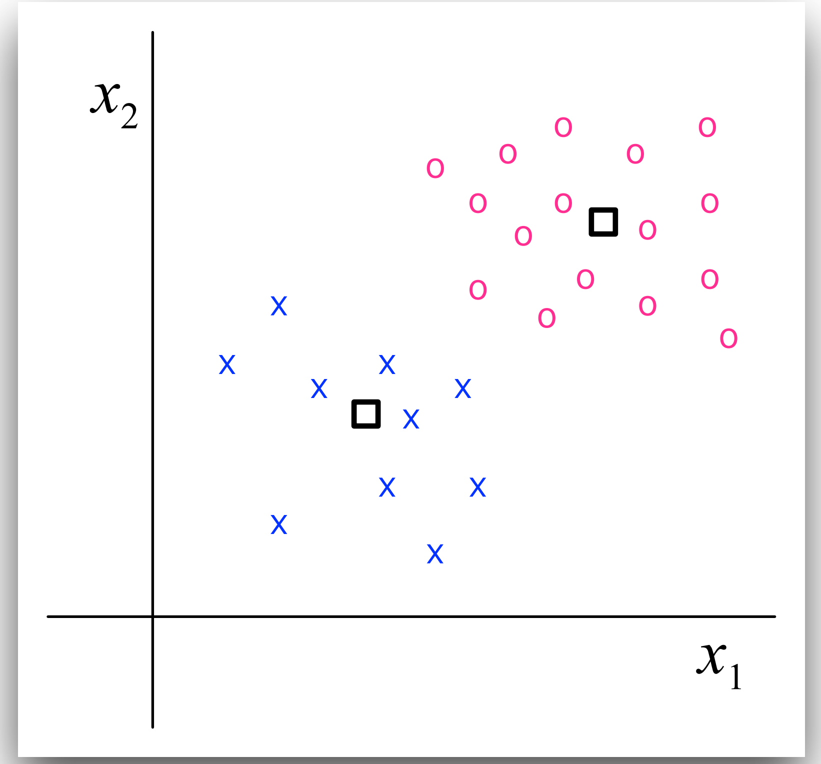
Such as:

- Identity of word at location *t, t-1, t-2 …*
- Average number of searches in past week, month, year
- Feature statistics from surrounding regions
- True (or predicted) value from previous time step

Vector Quantization

Use the output of some other algorithm to get features:

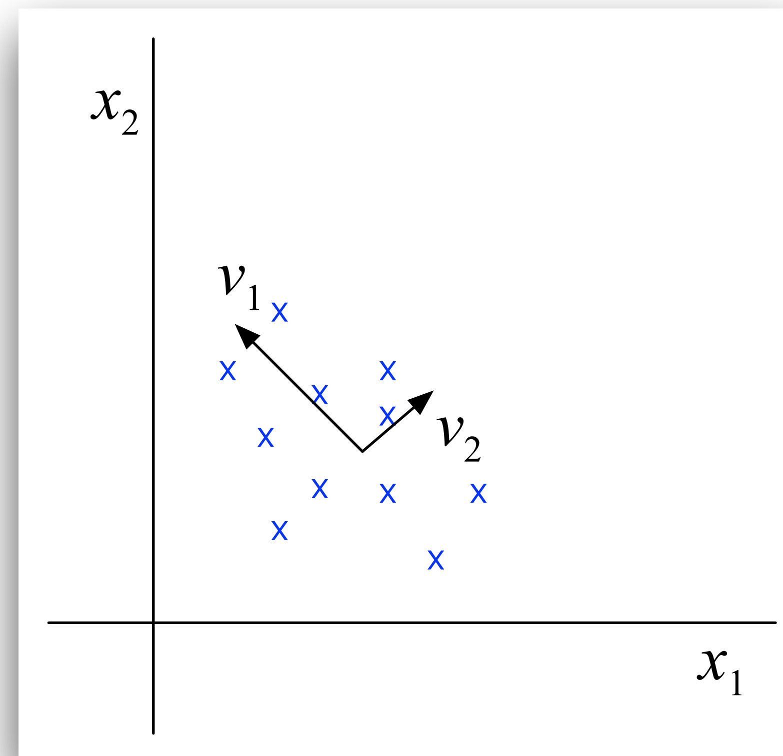
- Run k-means clustering
- For each data point, add a feature that gives the index of the closest cluster centroid.
- (Could use one of k encoding.)
- This is a generalisation of the 1-D binning idea from previous slide



Dimensionality Reduction

Principal Components Analysis returns a linear map

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \mapsto \begin{pmatrix} z_1 \\ \vdots \\ z_P \end{pmatrix} \qquad P << M$$



Use **z** as features instead of (or in addition to?) **x**

Could use fancier techniques, e.g.,

- manifold learning
- topic modelling
- deep neural networks
  (activations of hidden layer)

Model Combination

Suppose you want to improve on existing systems.
Just add their output as a feature to your classifier!

If they provide a confidence, e.g., a probability
could use predicted log probability as feature

Examples:

- Machine translation
- Netflix prize

Simple Transfer Learning

Common: Need to solve "lots of little prediction problems"

- Email spam filter for each person
- Fraud detection of personal credit card accounts

Compare *domain adaptation, transfer learning, multitask learning*

Different prediction tasks not identical

Features can have different meanings across tasks, e.g.,

- "Viagra" commonly included in spam emails
- But a GP might often see it in regular emails

But similar and only a small amount of data for each

Simple Transfer Learning

Common: Need to solve “lots of little prediction problems”

- Email spam filter for each person
- Fraud detection of personal credit card accounts

Compare *domain adaptation, transfer learning, multitask learning*

Trick: Both “general” and “specific” features:

- USER872324601_CONTAINS:Viagra
    - binary feature, 1 only if email contains “Viagra” and inbox from specified user
- CONTAINS:Viagra
    - binary feature, 1 if email contains “Viagra”

Example in research literature:

Daumé, *Frustratingly Easy Domain Adaptation*. ACL 2007

Feature Selection

Sometimes too many features bad.
Start with “full set” of features, prune less useful ones.

- Filters: Rank features by some “relevance” measure, e.g., mutual information, correlation with output. Choose top *K.* (Also called ranking, screening).
- Wrapper methods: Search through space of subsets of full feature set, to maximise performance on validation set. Many different strategies (forward versus backward)
- Wrapper as filter : Use a wrapper method on a linear classifier to find a good set of features, then train a (more computationally expensive) nonlinear one
- Lasso (l1 regularization) : Classification/regression and feature selection simultaneously

But sometimes… many features are just fine!