NUI Galway
OÉ Gaillimh

Machine Learning

Week 9: Linear Regression in One and Multiple Variables (Part 2)

Prof. Michael Madden
Chair of Computer Science
Head of Machine Learning & Data Mining Group
National University of Ireland Galway

Review - Multiple Linear Regression [1]

Given multiple input variables (x_i) that relate to a quantity of interest (y) …

Size (m^2)	# Beds	# Floors	Age (yrs)	Price (k€)
195	5	1	40	450
130	3	2	35	220
140	3	2	26	310
80	2	1	30	170
180	5	2	38	400

$\boldsymbol{x}^{(3)}$

$y^{(3)}$

$x_2^{(3)}$

N
no. of cases

x_1 x_2 x_3 x_4 y

Goal is to find parameters $\boldsymbol{\theta}$ for hypothesis $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ of the form

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m$$

such that for a given input vector $\boldsymbol{x}^{(i)}$, $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$ best approximates $y^{(i)}$

Review- Solving Multiple LR with Gradient Descent

The algorithm for Gradient Descent for Multiple LR is:

```
initialise $\boldsymbol{\theta}$ to any set of valid initial values
repeat until convergence or until limit is reached:
    simultaneously foreach $\theta_j$ in $\boldsymbol{\theta}$ do:
        $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$
```

The partial derivatives are given by:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

Feature Scaling

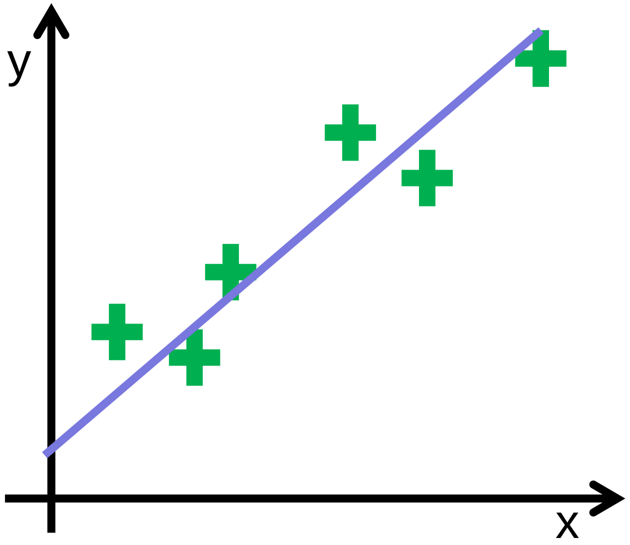
- Feature scaling can help improve G.D. convergence
  - G.D. uses a single tolerance and learning rate:
    helps if all features cover approximately similar ranges
  - Mean-centring the data can help if initial guesses are 0
- Techniques we discussed for kNN work well to rescale all dimensions independently
  - Mean=0, Std deviation=1 [Z-Normalisation]

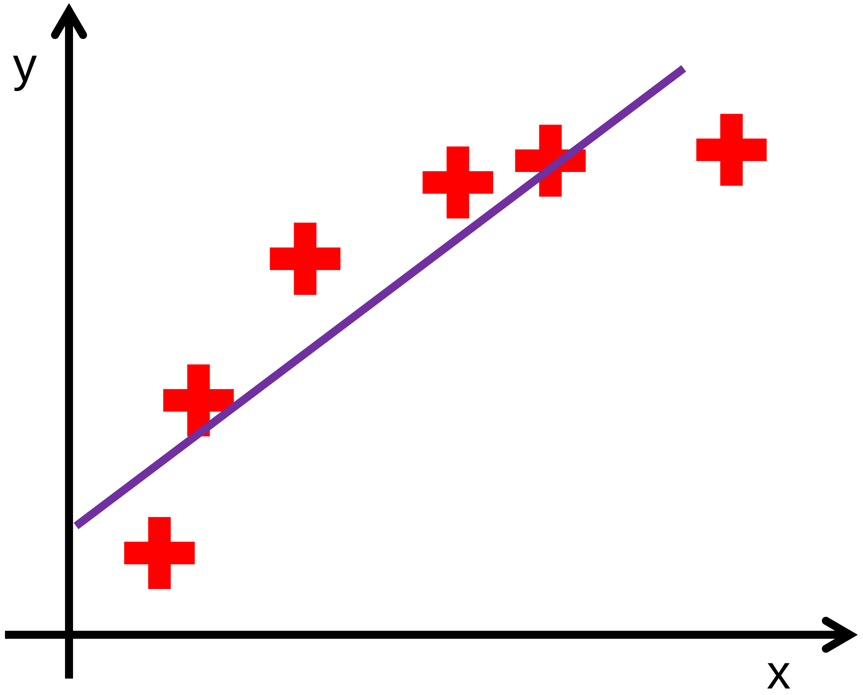
    D ← (D - Mean) / StDev
  - Min=-1, Max=1 [-1/1 Normalisation]

    D ← 2(D – Min) / (Max – Min) – 1

Linearity Assumption

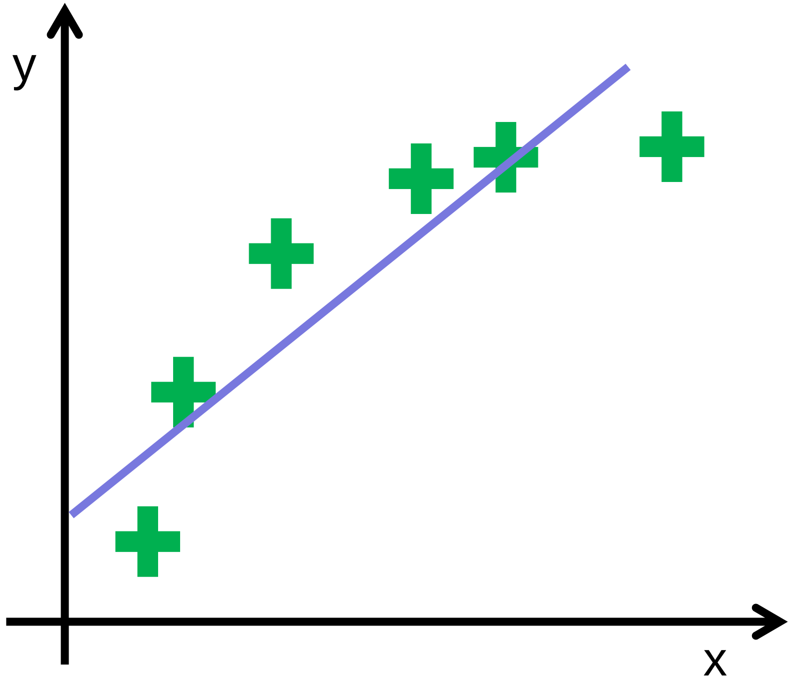
- Linear Regression has a very strong **bias**:
  - Assumes that there is a linear relationship between inputs and outputs
  - Useful if the relationship really is approximately linear, apart from some noise or small variations we don't need to capture
  - Less useful if not …





Polynomial Regression [1]

- Key insight: inputs to Linear Regression do not have to be the original raw data
  - Can generate new input variables based on **transformations** of the original ones
- That is basis for **Polynomial Regression**:
  - For some/all of the input variables x_n, add additional new variables: e.g. x_1^2, x_1^3, x_1x_2 …
  - Then perform linear regression as normal



Polynomial Regression [2]

- Example: suppose initial dataset has attributes x_1 , x_2 , x_3
  - Second-Order Polynomial Regression: create new version of dataset where attributes are: x_1 , x_2 , x_3 , x_1x_2 , x_1x_3 , x_2x_3 , x_1^2 , x_2^2 , x_3^2
  - Third-Order Polynomial Regression: create new version of dataset where attributes are **Second-Order ones and**: $x_1x_2x_3$, x_1^3 , x_2^3 , x_3^3 , $x_1x_2^2$, $x_1^2x_2$, …
- Using this new version of the dataset, run Multiple Linear Regression
  - Using standard Linear Regression algorithm to discover hypotheses that are non-linear relative to original attributes, since the new inputs to LR are non-linear transformations of original attributes
  - Warning: number of features increases dramatically
- Won't go into full algorithm details, but here is a simple illustrative example with just 1 input variable: SimplePolynomialRegression.ipynb

Polynomial Regression – Final Comments

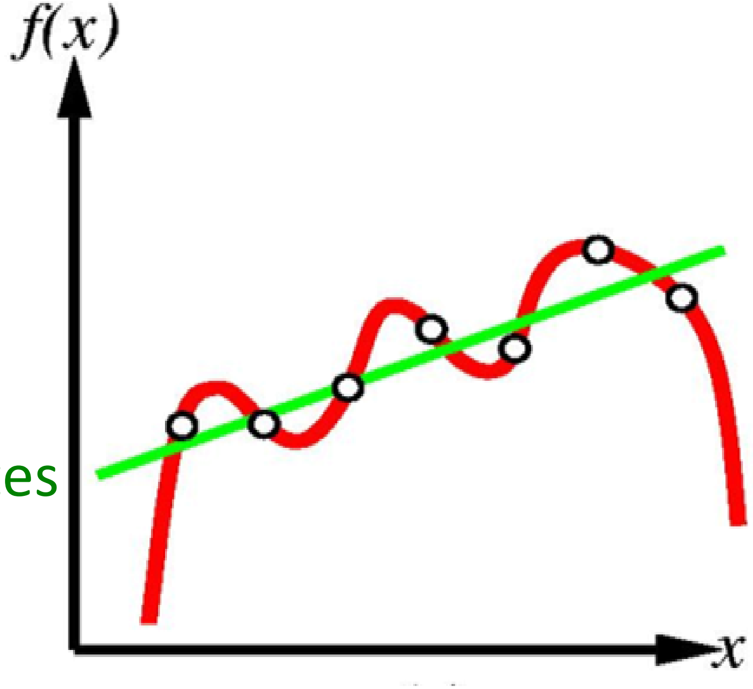
- While polynomial regression is a reasonably common variant of linear regression, it is also possible to use other transformations
- There are other ways of extending linear regression to deal with non-linear data, that are outside of the scope of this module
  - Support Vector Machine with non-linear kernel
  - Multi-layer perceptron: feed-forward neural network with at least one hidden layer

Regression: Bias, Variance, Underfitting and Overfitting [1]

- ML algorithms may be characterised in terms of **bias** and **variance**:
- Variance errors: over-sensitivity to small changes in training data.
  - High variance => algorithm may fit to random noise in training data, rather than real relationships in data => **over-fitting**
- Bias errors: due to bad assumptions in the learning algorithm.
  - High bias => algorithm cannot capture all relevant relationships between attributes and output => **underfitting**
- Important: this is an entirely separate concept from bias in the sense of errors caused by humans who have biased judgements
  - Select ML models that support their biases, or supply biased training data
  - Many data sets may have historical biases; e.g. recruitment; sentencing

Regression : Bias, Variance, Underfitting and Overfitting [2]

- ML Algorithms with **complex hypothesis language** are prone to **overfitting**
- Those with **simpler hypothesis language** are prone to **underfitting**
  - If you increase complexity of hypothesis, you increase ability to fit to the data, but might also increase risk of overfitting
- Linear Regression:
  - Simple hypotheses: straight lines
  - High bias and low variance; risk of underfitting
- Polynomial Regression:
  - More complex hypotheses: many combinations of attributes
  - Higher variance and weaker bias; risk of overfitting
  - Need more data



Gradient Descent with Regularization

- High number of variables (whether from Polynomial Regression or otherwise) increases risk of overfitting
- Potential solutions:
  - Reduce number of variables (see later slides)
  - **Regularize**: encourage low/zero values for weights $\boldsymbol{\theta}$: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots$
- **Regularization**:
  - Add a term to cost function to reflect some measure of **complexity** of the hypothesis:
    $Cost(\boldsymbol{\theta}) = EmpiricalError(\boldsymbol{\theta}) + \lambda\, Complexity(\boldsymbol{\theta})$
  - Then, Linear Regression will find parameters for $\boldsymbol{\theta}$ that make a trade-off of error vs. complexity
  - Can also be seen as trading off ***bias vs. variance***

Gradient Descent with Regularization

- New cost function:

  Cost($\boldsymbol{\theta}$) = *EmpiricalError*($\boldsymbol{\theta}$) + λ *Complexity*($\boldsymbol{\theta}$)

  Empirical Error: old cost function *J*()

  Complexity: $$L_q(\theta) = \sum_{i=1}^{m} |\theta_i|^q$$

- Note:

  λ controls tradeoff between error & complexity.

  We don't regularize θ_0 as we don't want to eliminate it.

  Depending on *q*, get different regularisation effects:

  q=1 is popular because it encourages sparse models (often sets many weights to 0), but *q*=2 simplifies the maths

Gradient Descent with Regularization

The cost function therefore becomes:

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\left[\sum_{i=1}^{N}\left(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{m}\Theta_j^2\right]$$

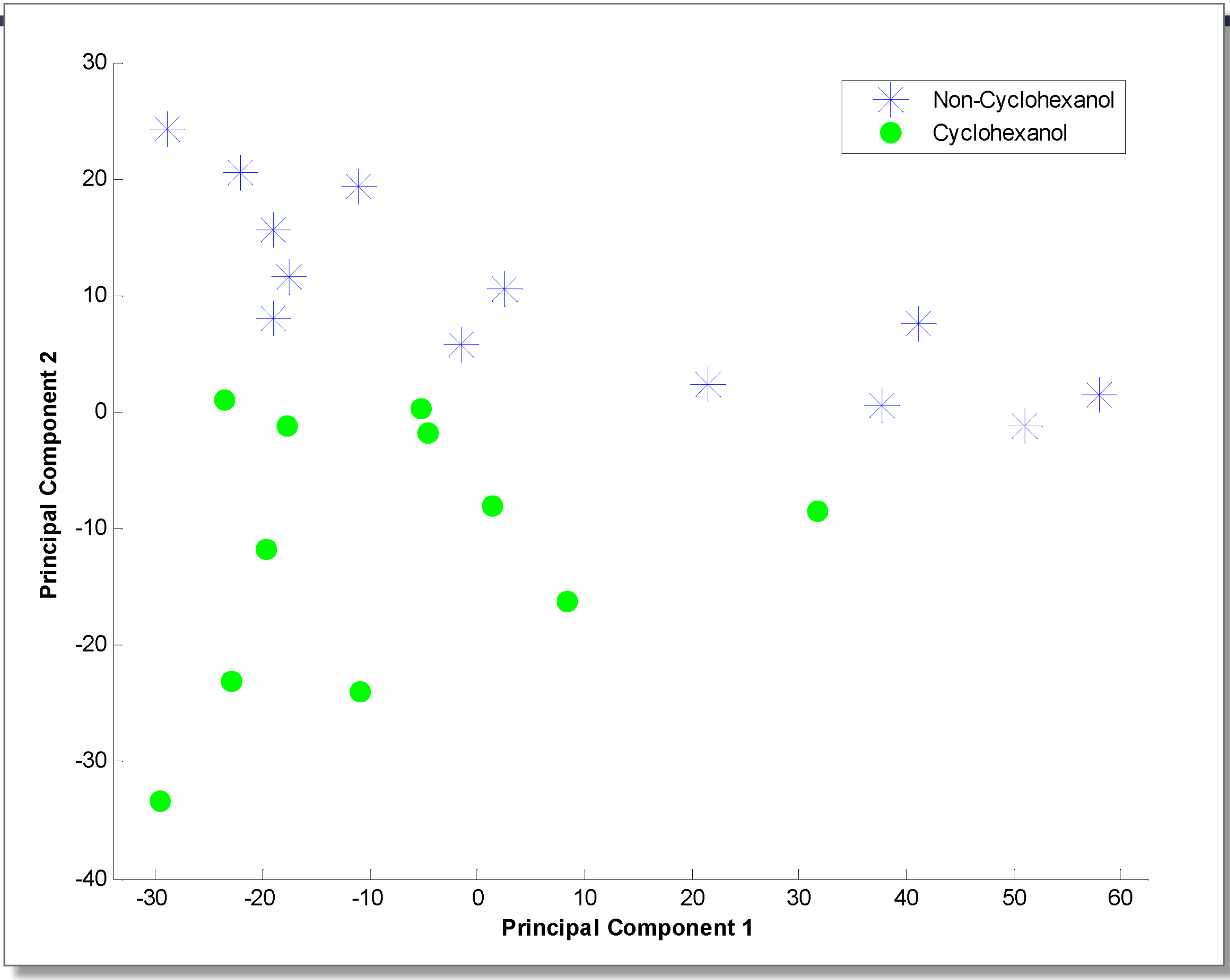
And its partial derivative for Gradient Descent is:

$$\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(h_{\theta}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)\, x_j^{(i)} + \frac{\lambda}{N}\Theta_j \quad (\text{where } j > 1)$$

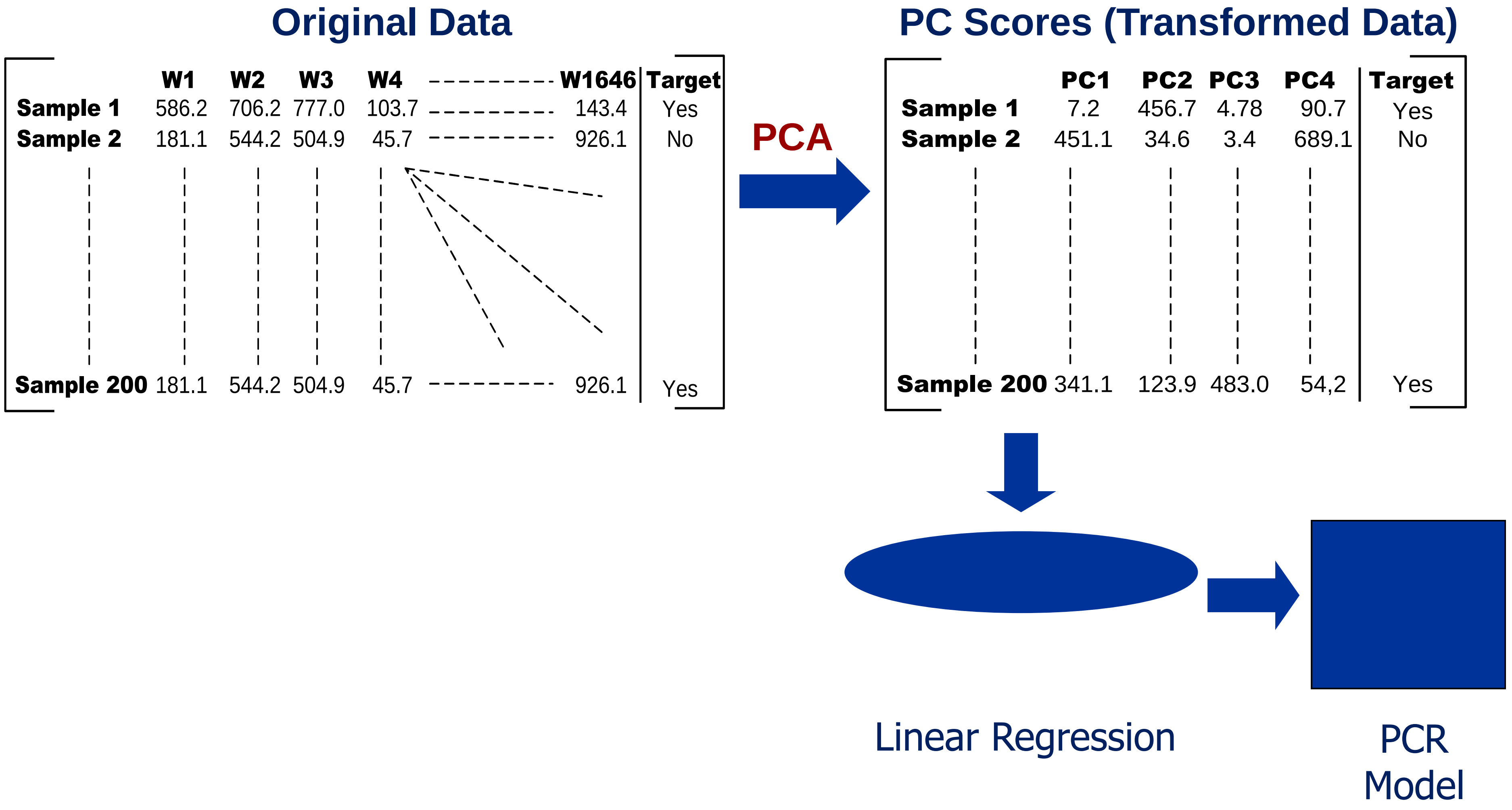
Linear Regression & Dimension Reduction

- Linear Regression can fail (and give very bad predictions) if the number of samples is **smaller than** the number of dimensions per sample, *unless* you use regularisation
  - Closed form is ill-conditioned; lose convergence guarantees of GD
  - In some application domains, this happens routinely
  - Chemometrics, DNA analysis, medical image analysis, …
- Solution: Feature Selection or Transformation+Selection
  - Reduce dimensionality
  - Manual or algorithmic
  - Also improves speed of computations
- Principal Component Analysis is popular:
  - Transforms data to new axes where dimensions are ordered in terms of how much of the data variance they explain
  - Discard all but top *N* dimensions (*N* is application-dependent)
  - Good fit with Linear Regression assumptions: linear & non-correlated

PCA Example


Non-Cyclohexanol
Cyclohexanol
Principal Component 2
Principal Component 1
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40
50
60

Principal Component Regression



Original Data

	W1	W2	W3	W4		W1646	Target
Sample 1	586.2	706.2	777.0	103.7		143.4	Yes
Sample 2	181.1	544.2	504.9	45.7		926.1	No
Sample 200	181.1	544.2	504.9	45.7		926.1	Yes

PC Scores (Transformed Data)

	PC1	PC2	PC3	PC4	Target
Sample 1	7.2	456.7	4.78	90.7	Yes
Sample 2	451.1	34.6	3.4	689.1	No
Sample 200	341.1	123.9	483.0	54,2	Yes

PCA Features

- When to consider using PCA:
  - Useful pre-treatment for many ML methods
  - Can capture most of variance in data, while greatly reducing dimensionality
- Particularly useful with methods that do not work well with high-dimensional data, e.g.
  - Linear Regression
  - *k* Nearest Neighbours
- Limitations:
  - ‘Blind’ to the data labels
  - In some situations major variance is not associated with your question of interest
  - E.g. want to measure small variations in an active ingredient; anomaly detection

When to Use Linear Regression

- Consider using when:
  - More training cases than attributes per case
- Benefits:
  - Reasonably fast
  - Simple hypothesis representation
    => Good comprehensibility
    => low risk of overfitting, at least in low dimensions
- Drawbacks:
  - Simple hypothesis representation
    => not good for complex decision planes

Feature Engineering

- Feature Engineering - broad term encompassing:
  - **Feature extraction:**
    e.g. extracting some high-level features from an image
  - **Feature transformation:**
    e.g. principal component analysis, FFTs, etc
  - **Feature subset selection:**
    keeping only a subset of the original features, discarding the rest

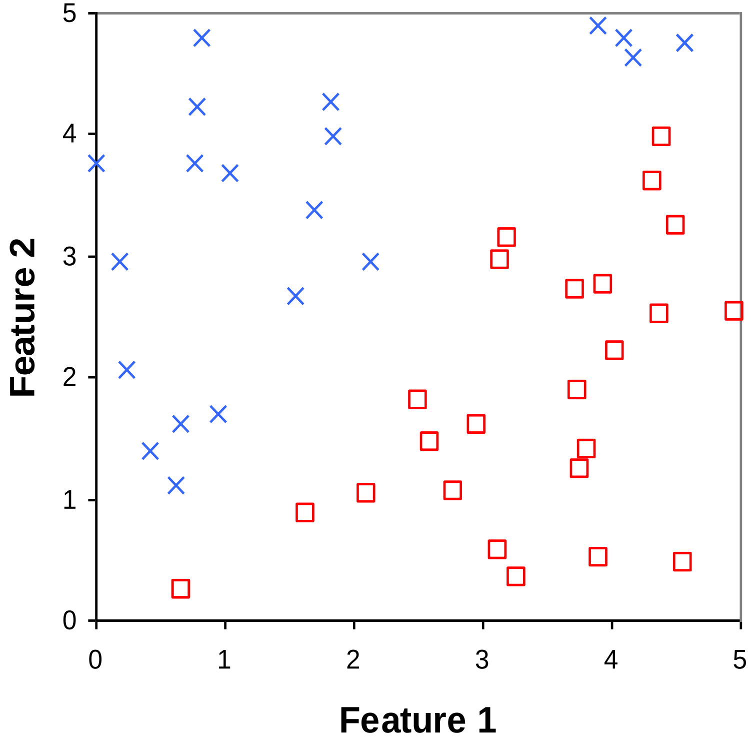
Feature Subset Selection: Overview (1)

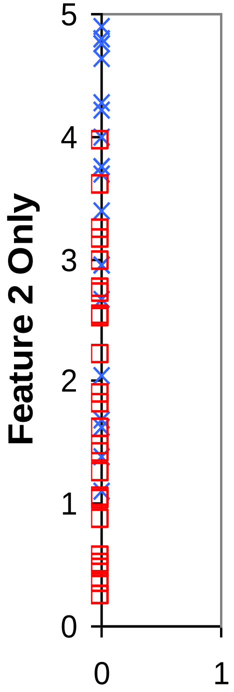
- Feature selection implicit in all data pre-processing
  - Normally use domain knowledge to decide what features are relevant
  - PlayTennis example: do we need to consider whether the stock market rose or fell today?
- Aside:
  - Sometimes we are mistaken in thinking a feature is irrelvant!
  - Sometimes we’d like to include a feature but we can’t measure it
- Automatic Feature Subset Selection:
  - Starting from list of features that **might be** relevant, identify those that are **actually** relevant
  - Hence, Feature **Subset** Selection

Feature Subset Selection: Overview (2)

- A general technique …
  - Particularly useful for kNN, Naïve Bayes, …
  - Any method that assumes all attributes significant and/or independent
- Can also improve performance of algs that don't!
  - E.g. Decision trees, neural nets
- Why?
  - The less junk they have to deal with, the less likely they are to be confused
- Can also improve economy of representation

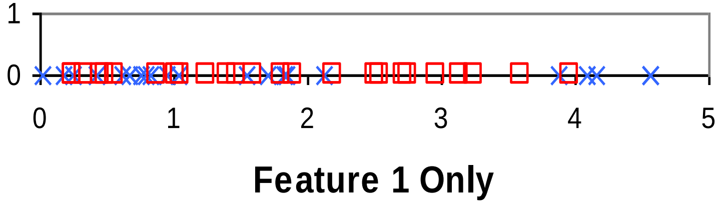
Features Can’t be Considered Independently





Here, we must look at both Feature 1 and Feature 2

Either one alone is insufficient to distinguish between the two different classes

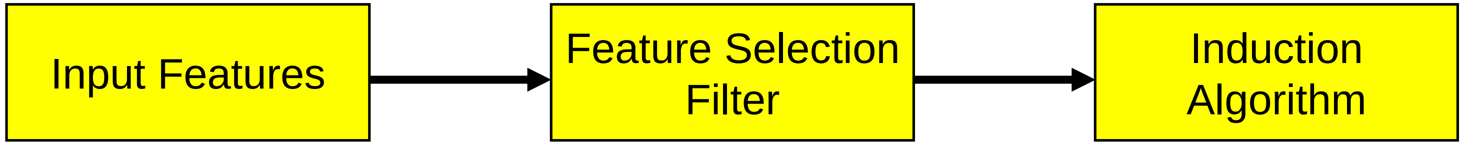


Feature Subset Selection is Hard …

- Suppose we have 100 features, but 98 are irrelevant
- If features could be considered independently:
  - Just test each in turn to find the important ones: 100 tests
  - But they can't!
- Need to consider feature combinations
  - 2^{100} tests!
  - **So when I say hard, I mean NP**
- Two approaches: Filter and Wrapper
  - Both require search and evaluation: differ in evaluation
  - 2^N combinations of N features => heuristic search

Feature Selection: Filter

- Evaluates features independent of learning alg.
  - Requires measurement of relevance of features to class:
    - Statistical tests
    - Features selected by decision tree
    - kNN: same/different values for nearby instances with different/same class
    - other
  - Filter bias different to induction alg bias



Feature Selection: Filter

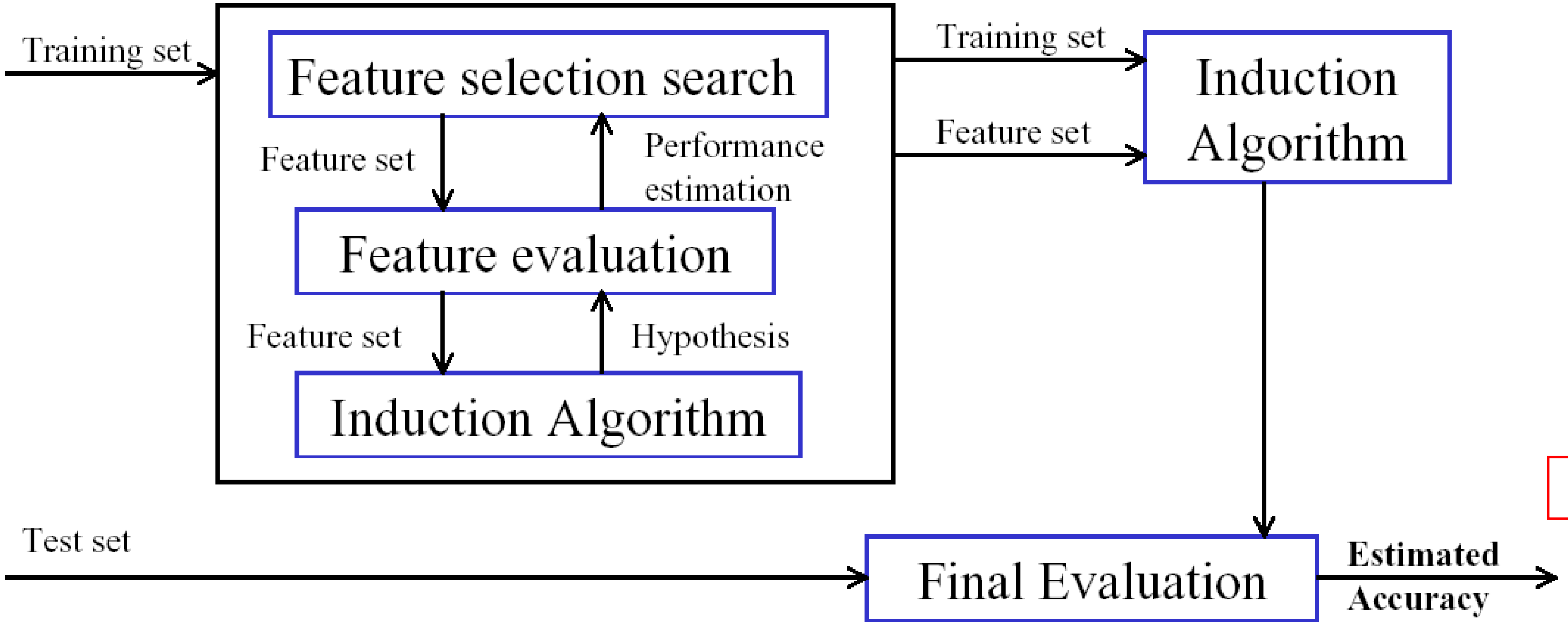
- Filters are usually less computationally intensive than wrappers. Often used in the pre-process phase.

- But they produce a feature set which is not tuned to the specific algorithm more general than a wrapper.

- Can be useful for exposing the relationships between features.

Feature Selection: Wrapper

- Selection process wrapped around learning alg
  - Features selected are suited to learning alg
  - More computationally intensive



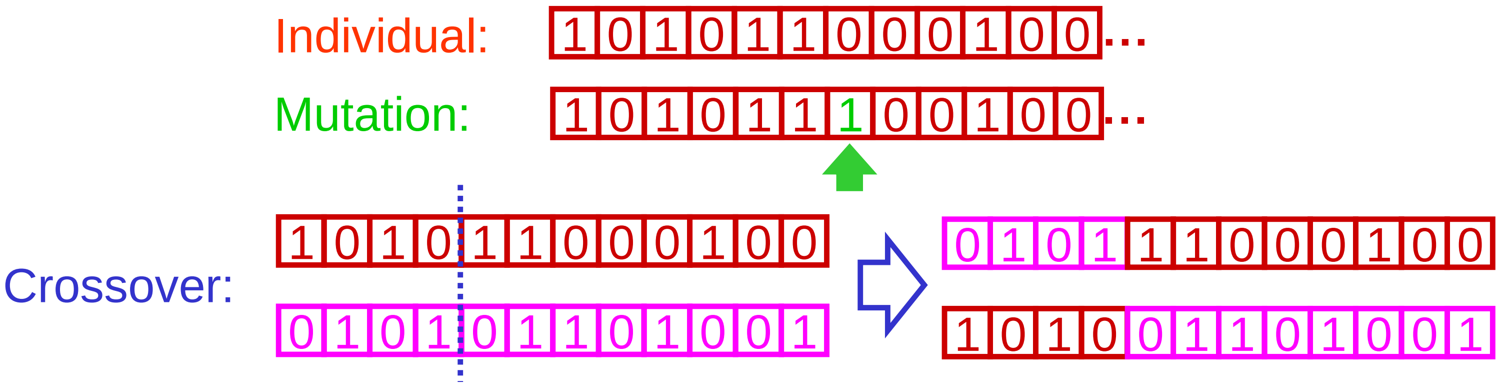
[P Cunningham, UCD]

Feature Selection: Wrapper Example (1)

- Prediction of concentration of chemicals from spectra
  - 510 points/spectrum: most correlated or irrelevant
  - Search for a subset of attributes that minimises prediction error on known cases
- Search using Genetic Algorithm:
  - Optimisation technique inspired by evolution
- Procedure:
  - Create random population of potential solutions
  - Assess fitness of each (1/error)
  - Delete unfit individuals; create new population by crossover and mutation
  - Repeat for a large number of generations until a stable solution is found

Feature Selection: Wrapper Example (2)

- In this application:
  - Each individual is string of 510 binary digits: 1=select this point in all spectra, 0=ignore it
  - Fitness assessed by applying ML algorithm (kNN or NNet) and calculating Root Mean Square Error
  - Mutation: randomly toggle a bit in an individual
  - Crossover: select a random point in two individuals, cut and swap 'heads' and 'tails'



Individual: 1 0 1 0 1 1 0 0 0 1 0 0…

Mutation: 1 0 1 0 1 1 1 0 0 1 0 0…

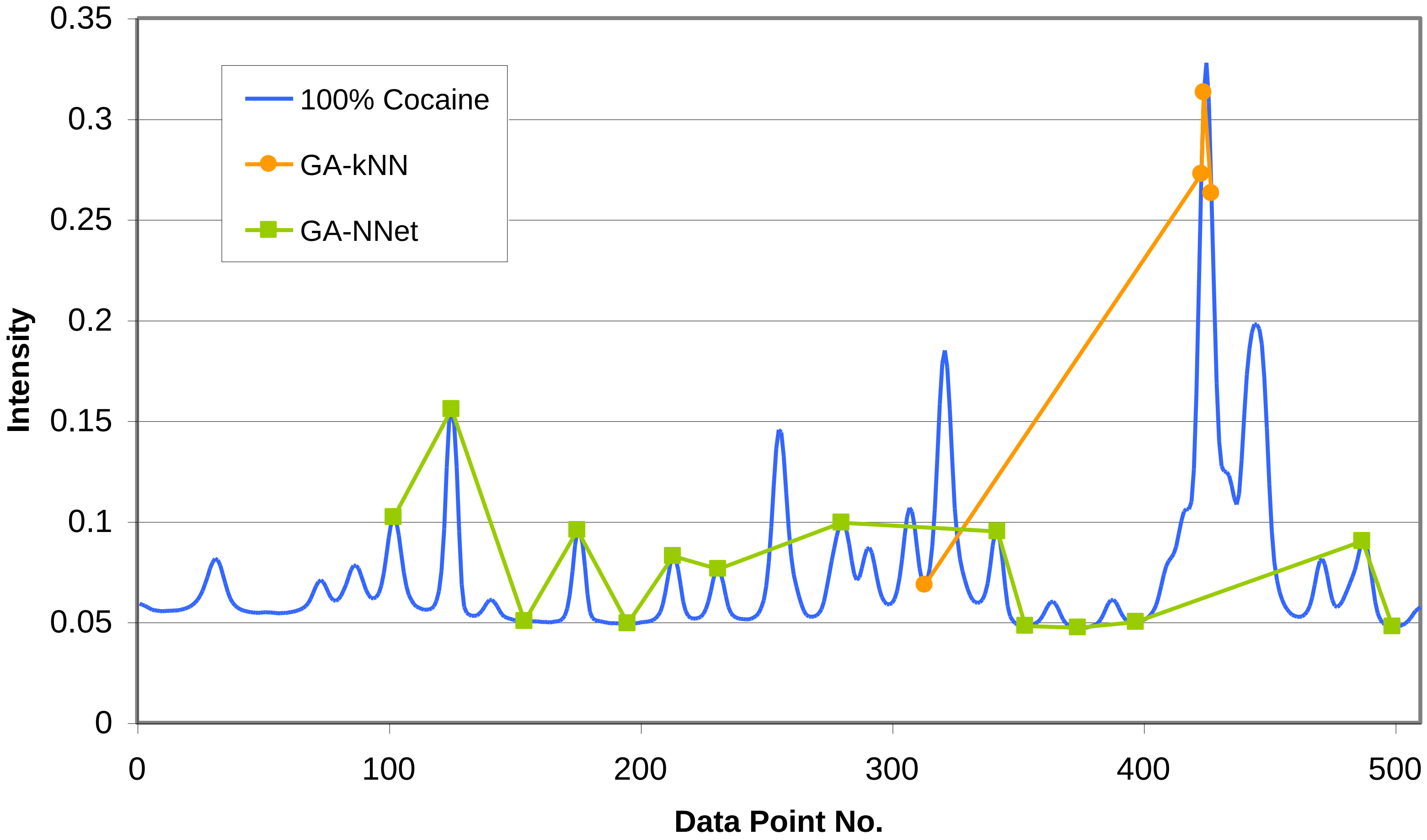
Crossover:

1 0 1 0 | 1 1 0 0 0 1 0 0 ⇨ 0 1 0 1 1 1 0 0 0 1 0 0

0 1 0 1 | 0 1 1 0 1 0 0 1 ⇨ 1 0 1 0 0 1 1 0 1 0 0 1

Feature Selection: Wrapper Example (3)

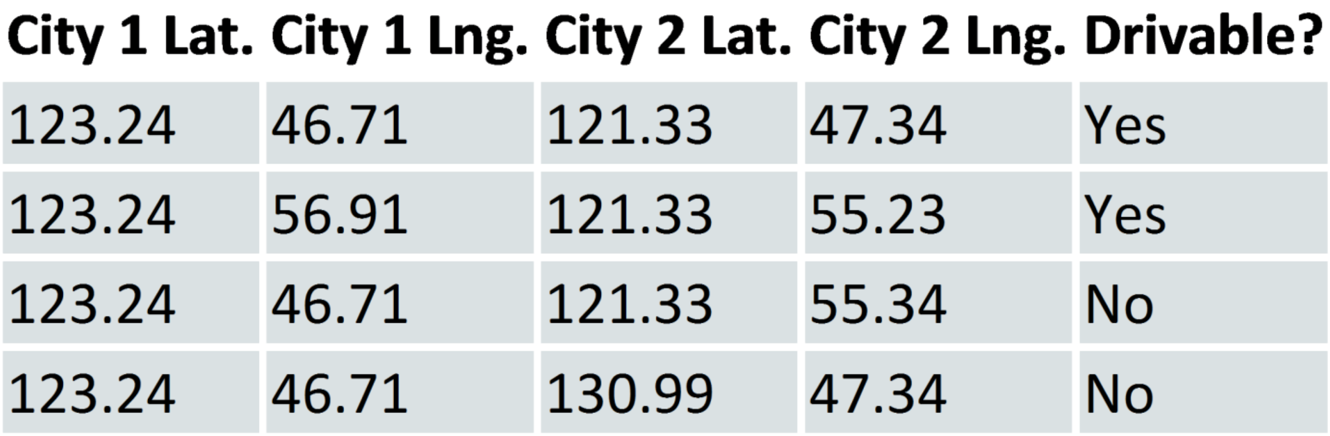
- Hennessy, Madden & Ryder, 2004
- NNet selected 14 points; kNN selected just 4



Feature Engineering: Transforming & Extracting Features

- "Feature Engineering is the Key"
  - Domingos, CASM 2012
  - "Easily the most important factor is the features used" in determining if a ML project will succeed/fail
- Some estimate that they spend 70% of their time on feature engineering activities
- Such work is mainly manual in nature, based on understanding the domain and the algorithms you are working on

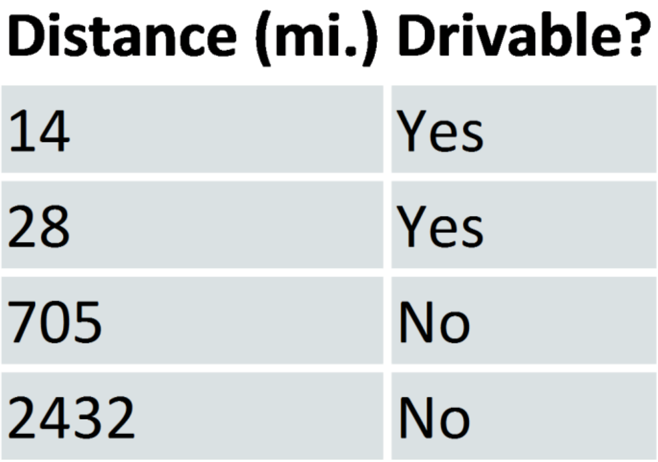
Feature Transformation/Extraction: Motivating Example



City 1 Lat.	City 1 Lng.	City 2 Lat.	City 2 Lng.	Drivable?
123.24	46.71	121.33	47.34	Yes
123.24	56.91	121.33	55.23	Yes
123.24	46.71	121.33	55.34	No
123.24	46.71	130.99	47.34	No

ML algorithms would likely fail at this task

But if we use our knowledge of trigonometry to compute distances …



Distance (mi.)	Drivable?
14	Yes
28	Yes
705	No
2432	No

http://blog.bigml.com/2013/02/21/everything-you-wanted-to-know-about-machine-learning-but-were-too-afraid-to-ask-part-two/

Feature Engineering: Key Steps

- Understand your input data
  - Where does it come from?
  - What assumptions come from how it was collected?
  - Any sources of noise and error?
- Understand your algorithms
  - Can they work with the raw data?
  - Would they be helped if the data was adapted?
- Apply an iterative approach
  - Don’t just do 10 things at once:
    understand the effect of each step you perform
  - Keep testing to see if your engineering steps help

Feature Extraction: Cars Example (1)

- Munroe & Madden, 2005
- Objective: Identify the make and model of a vehicle from images
- Five different vehicle types to recognise
  - Apart from these 5, other types included in test set

Opel Corsa

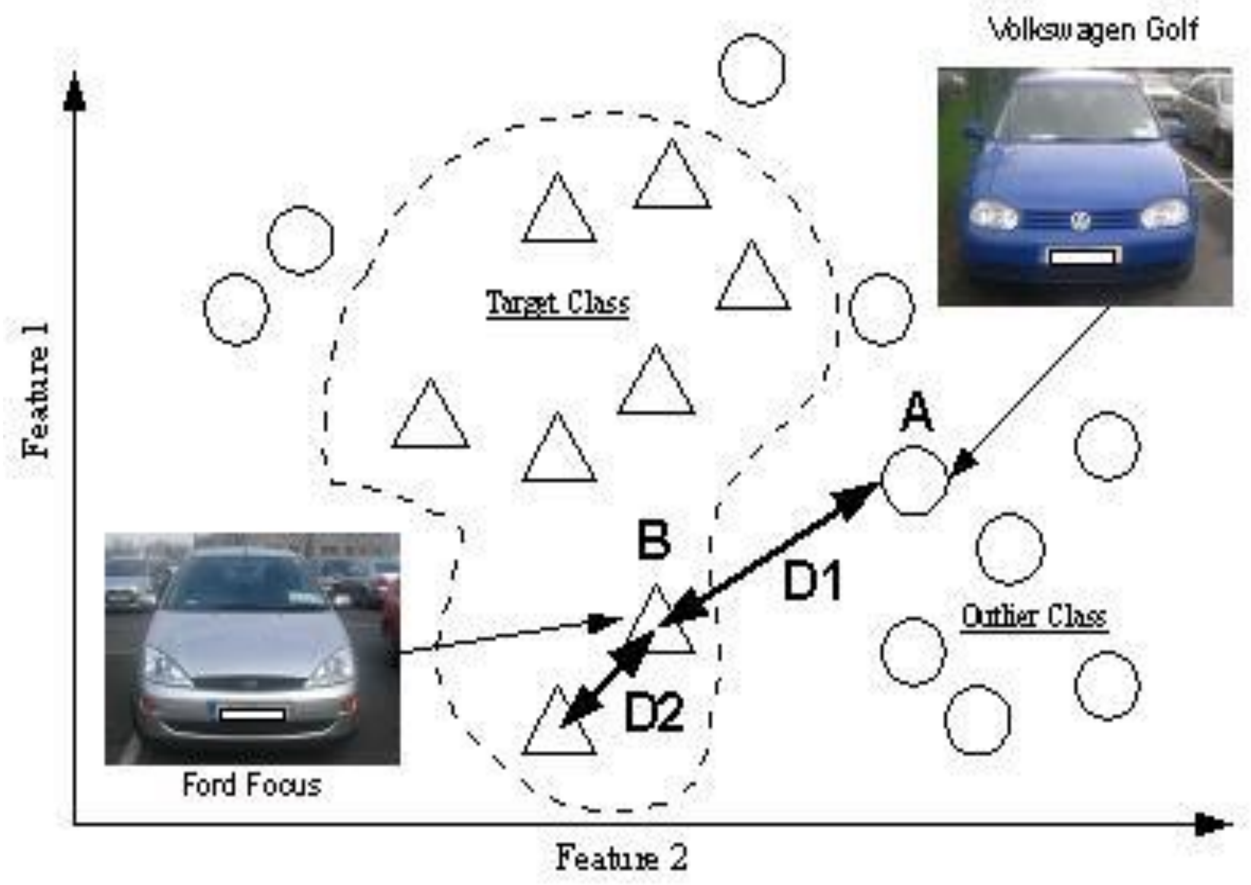
Ford Focus

Ford Fiesta

VW Golf

VW Polo

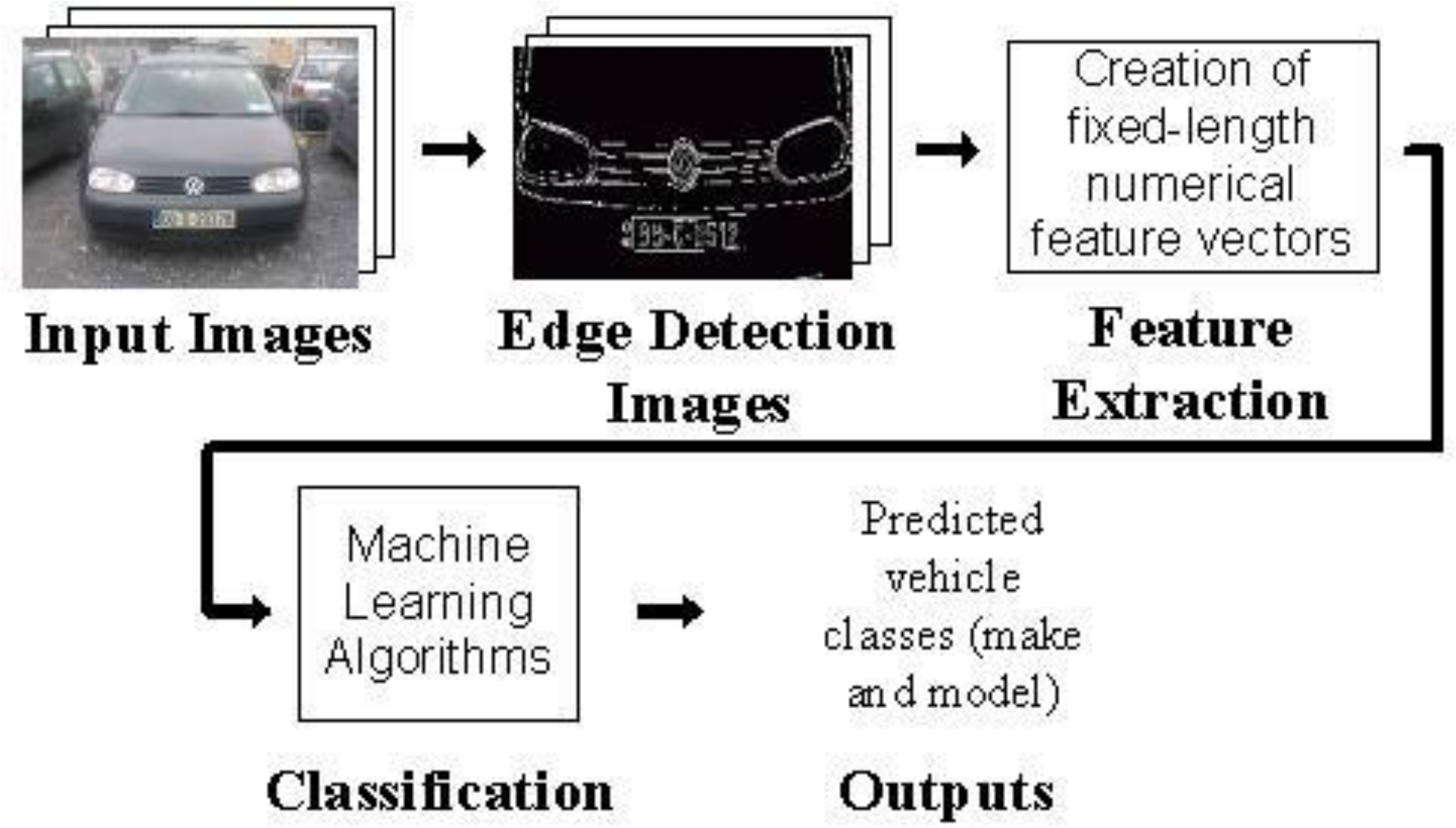
Feature Extraction: Cars Example (2)



Devised the **One-Class k-Nearest Neighbour** algorithm for this task.
Also evaluated off-the-shelf **multi-class classification** algorithms.

Feature Extraction: Cars Example (3)

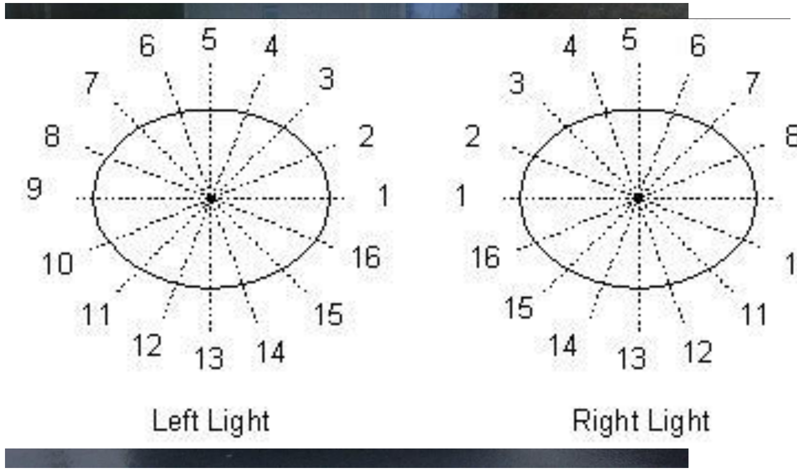
Overall System for Vehicle Make/Model Recognition from Images


Input Images
Edge Detection Images
Creation of fixed-length numerical feature vectors
Feature Extraction
Machine Learning Algorithms
Classification
Predicted vehicle classes (make and model)
Outputs

Feature Extraction: Cars Example (4)

Feature Extraction Processes

1. Crop the image
2. Canny Edge Detection: Detect the edges
3. Dilation: Grows or thickens the object outlines
4. Detection of vehicle lights
5. Shape Signature: 1-D functional representations of the boundaries of objects



Original Image
Cropped Image
Edges Detected
Dilated Image
Lights Detected
Shape Signature

Feature Engineering: Final Comments

- Manual feature engineering techniques often make classification tasks, but are problem-specific
  - Scaling problem: every new task needs new work
  - Would be preferable if we could use ML to automate this

- If you study deep learning, you will learn that one of its goals is to integrate:
  - Feature engineering
  - Classification/regression
  - Model selection

  into a unified analysis framework.

Learning Objectives - Review

Having successfully completed the material from last week and this week, you can now …

- Explain what Linear Regression is and its use in ML
- Describe and implement a Gradient Descent algorithm for learning LR parameters with one or multiple variables
- Describe and implement extensions such as Stochastic GD, regularisation, and polynomial regression
- Considering the characteristics of linear regression, recommend when it would be appropriate to an application
- Discuss and apply feature engineering methods including feature scaling, feature reduction, and transformation methods (e.g. PCA)