Basic Classification Algorithms (2)

Rules,

Linear Regression,

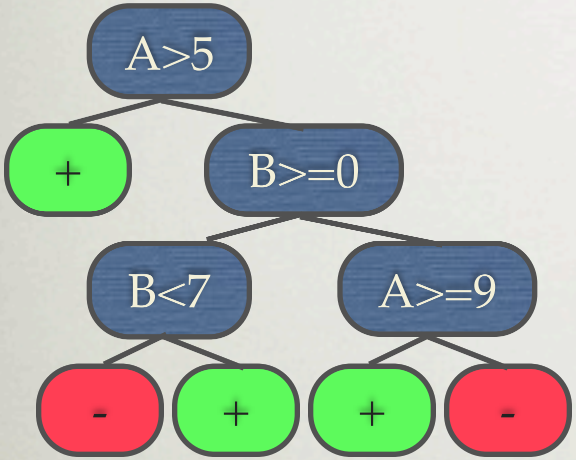
Nearest Neighbour

Outline

- **Rules**
- Linear Regression
- Nearest Neighbour

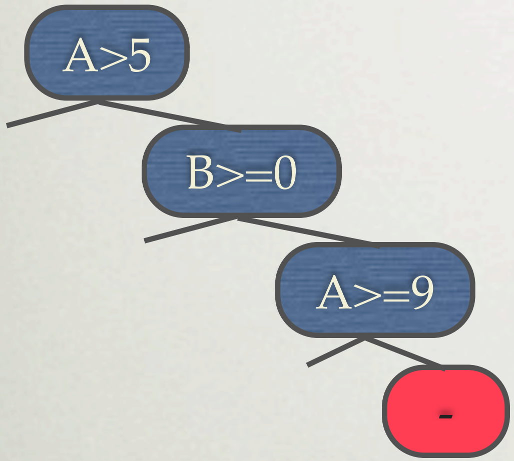
Generating Rules

- A decision tree can be converted into a rule set



Generating Rules

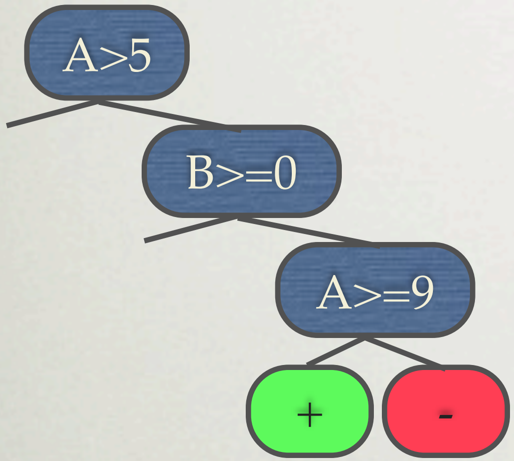
- A decision tree can be converted into a rule set




```
A>5 && B>=0 && A>=9  ->  -
```

Generating Rules

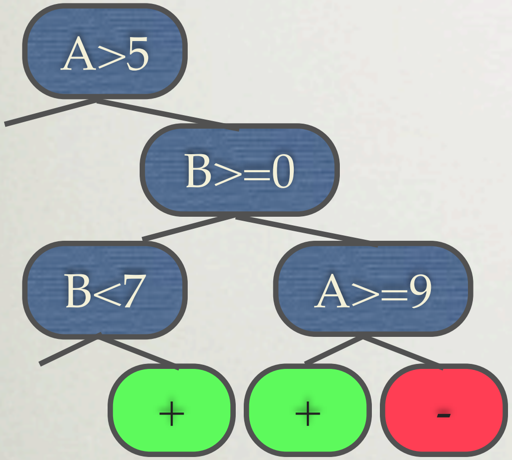
- A decision tree can be converted into a rule set




```
A>5 && B>=0 && A>=9   ->   -
A>5 && B>=0 && A<9    ->   +
```

Generating Rules

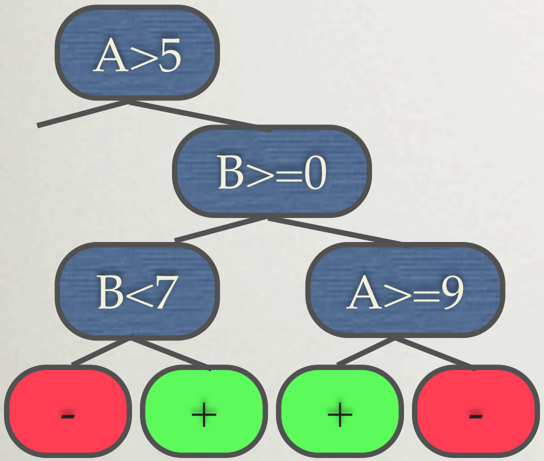
- A decision tree can be converted into a rule set




```
A>5  &&  B>=0  &&  A>=9   ->   -
A>5  &&  B>=0  &&  A<9    ->   +
A>5  &&  B<0   &&  B<7    ->   +
```

Generating Rules

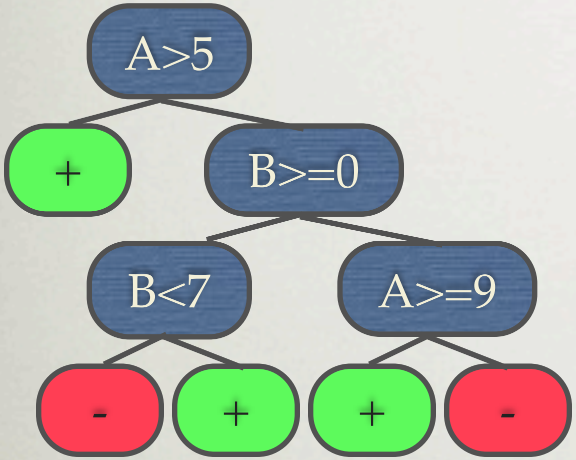
- A decision tree can be converted into a rule set




```
A>5  &&  B>=0  &&  A>=9    ->   -
A>5  &&  B>=0  &&  A<9     ->   +
A>5  &&  B<0   &&  B<7     ->   +
A>5  &&  B<0   &&  B>=7    ->   -
```

Generating Rules

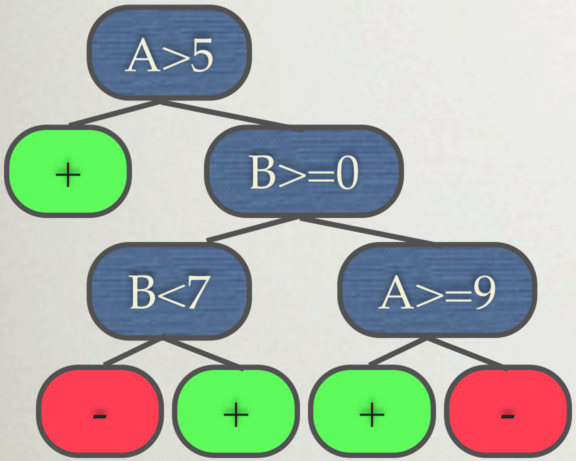
- A decision tree can be converted into a rule set




```
A>5  &&  B>=0  &&  A>=9   ->   -
A>5  &&  B>=0  &&  A<9    ->   +
A>5  &&  B<0   &&  B<7    ->   +
A>5  &&  B<0   &&  B>=7   ->   -
A<=5  ->  +
```

Generating Rules

- A decision tree can be converted into a rule set



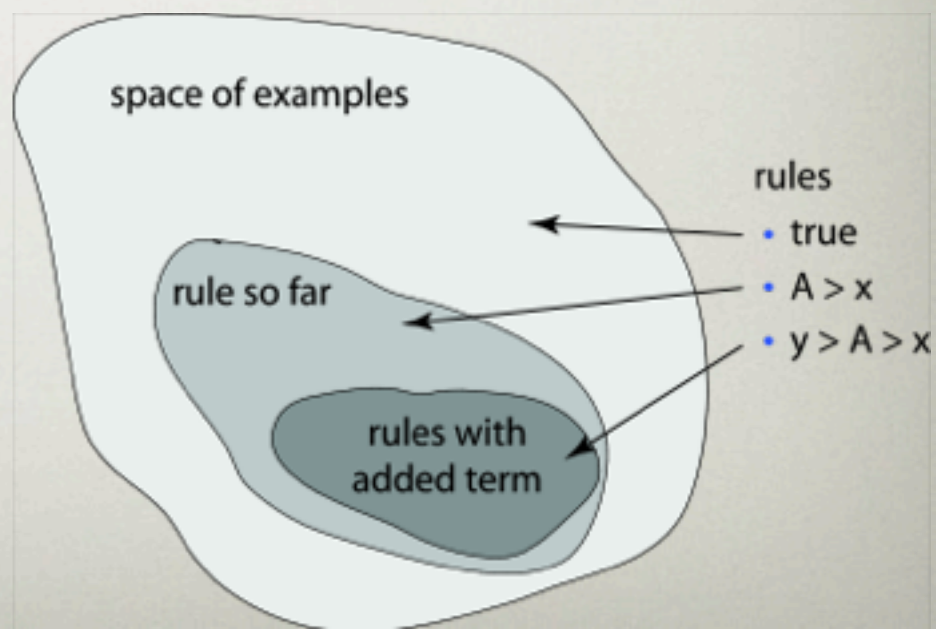

```
A>5  &&  B>=0  &&  A>=9   ->   -
A>5  &&  B>=0  &&  A<9    ->   +
A>5  &&  B<0   &&  B<7    ->   +
A>5  &&  B<0   &&  B>=7   ->   -
A<=5  ->  +
```

- Often overly complex, simplifying is not trivial
  - tests each node in root-leaf path to see if it can be eliminated without loss in accuracy (C4.5rule)

Covering algorithms

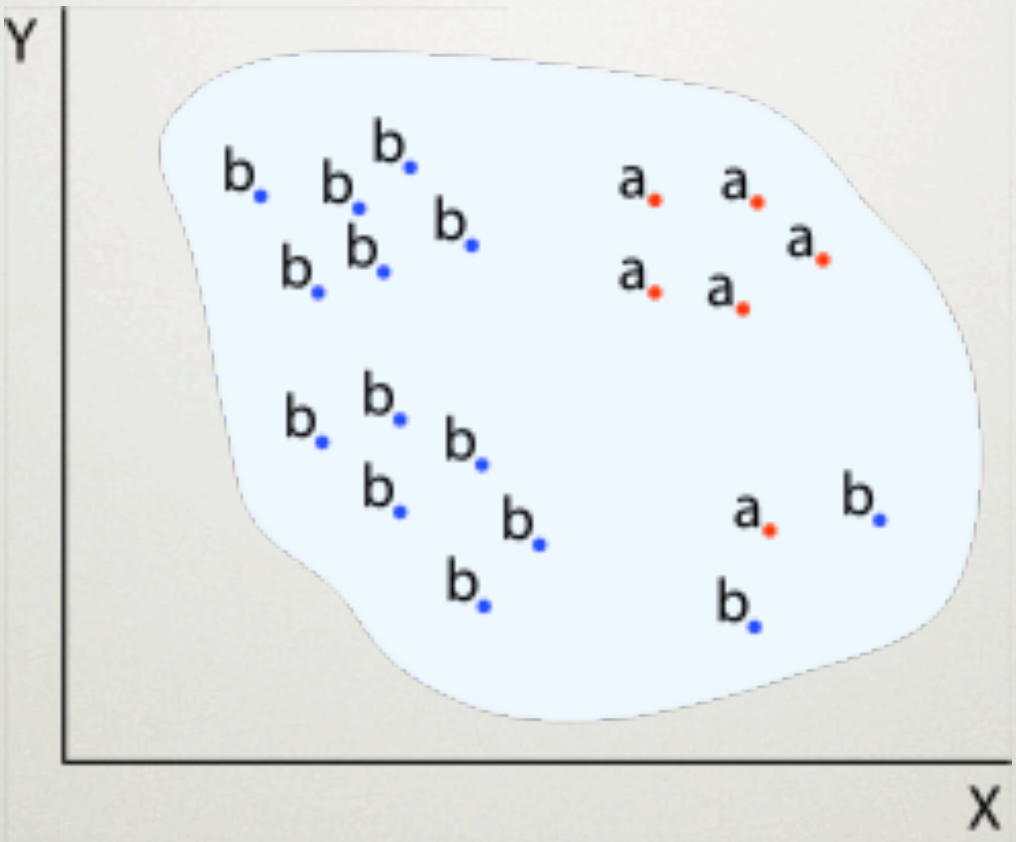
- Generate rule sets directly
  - for each class:
    - find rule set that covers all instances in it (excluding instances of other classes)

- *Covering* approach
  - at each stage a rule is identified that covers some of the instances



Example: generating a rule

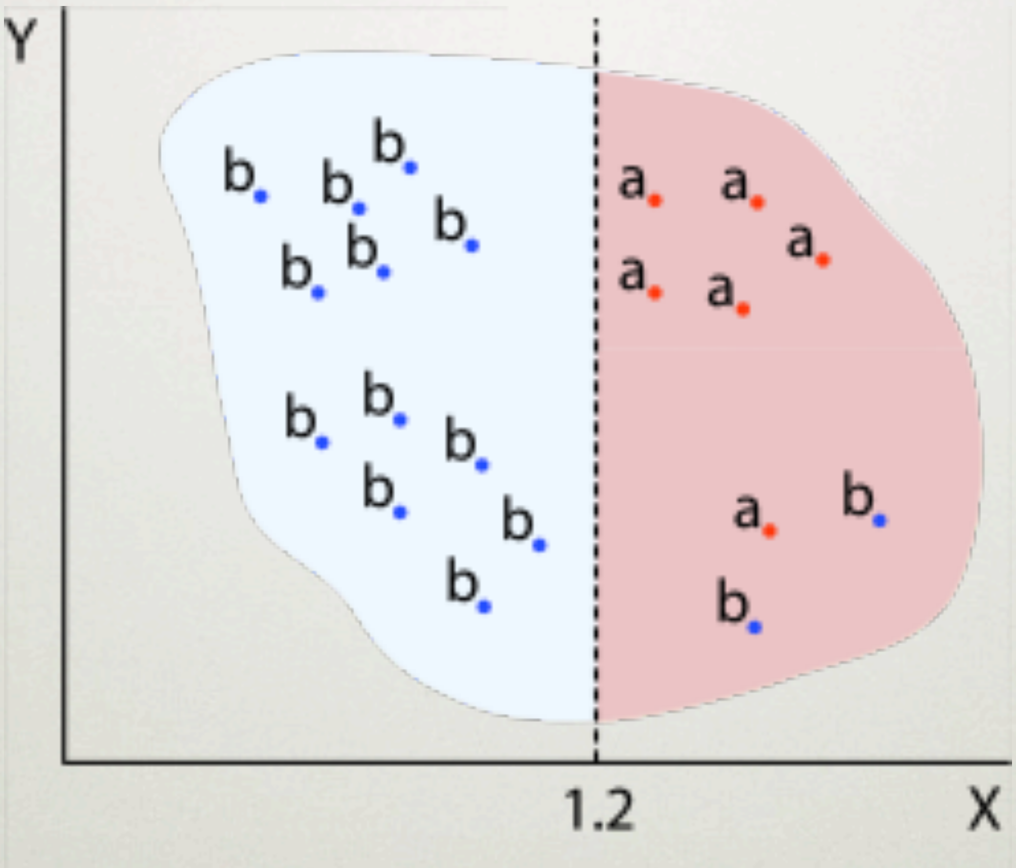
Class a



IF TRUE then class = a

Example: generating a rule

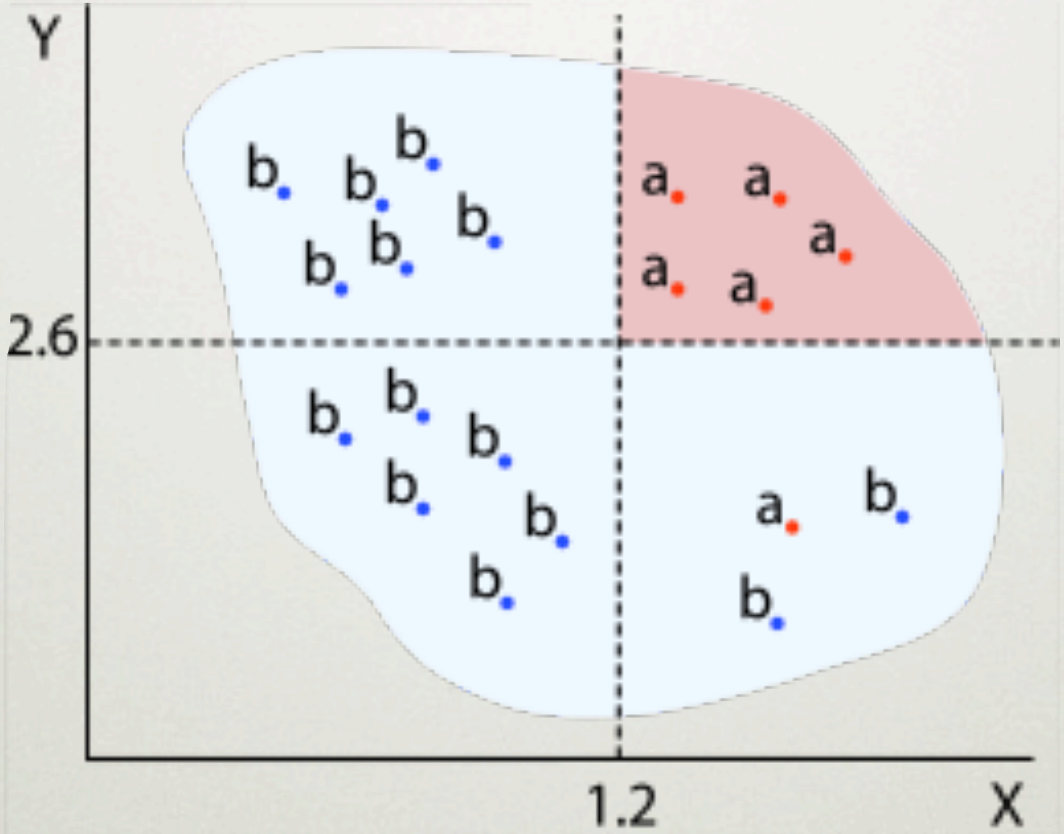
Class a



IF X>1.2 then class = a

Example: generating a rule

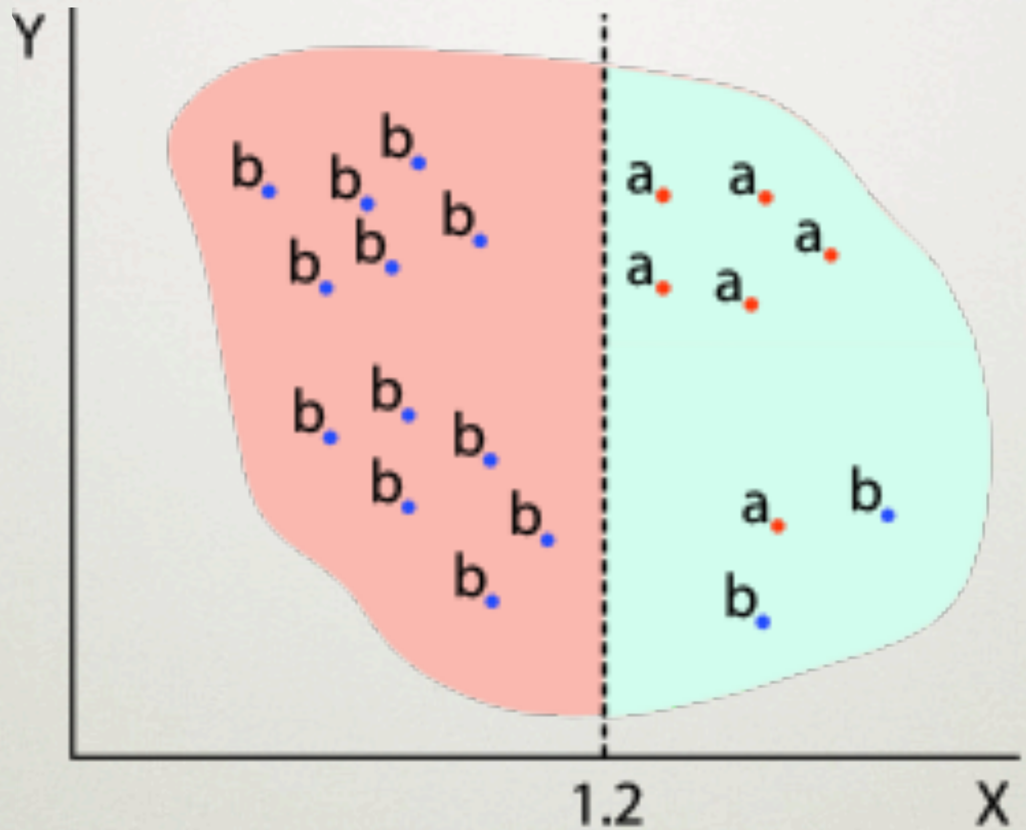
Class a



IF X>1.2 and Y > 2.6 then class = a

Example: generating a rule

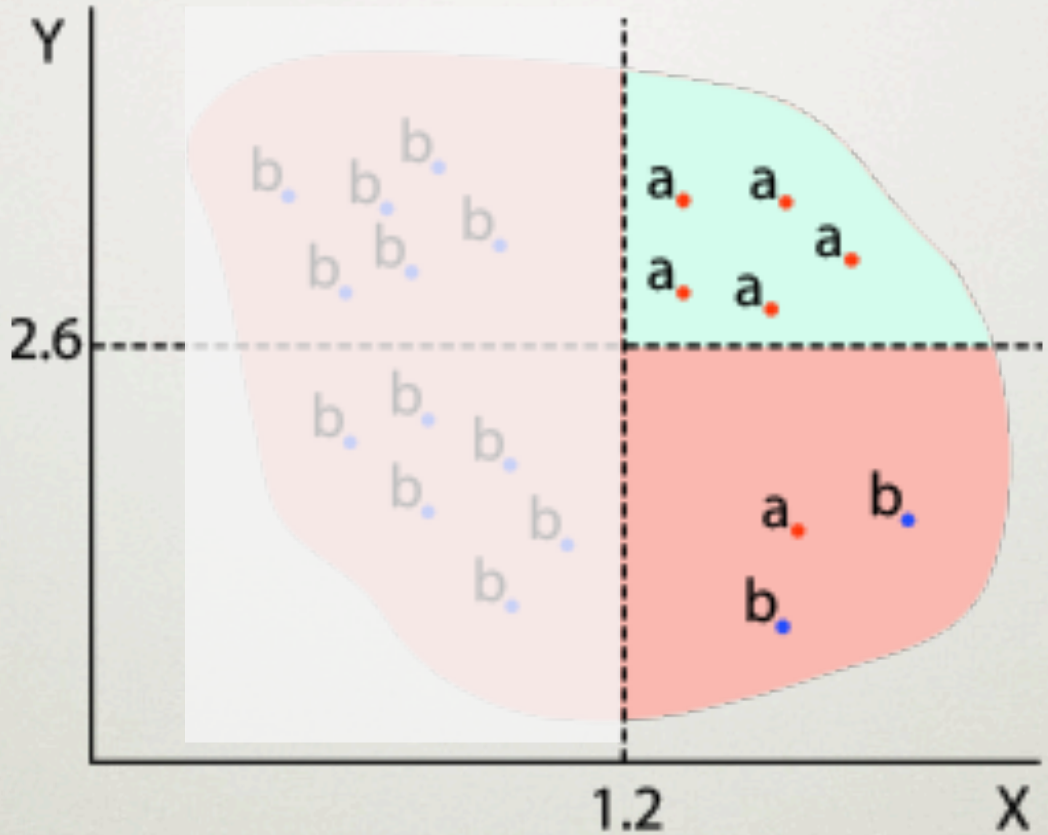
Class b, rule 1



IF X≤1.2 then class = b

Example: generating a rule

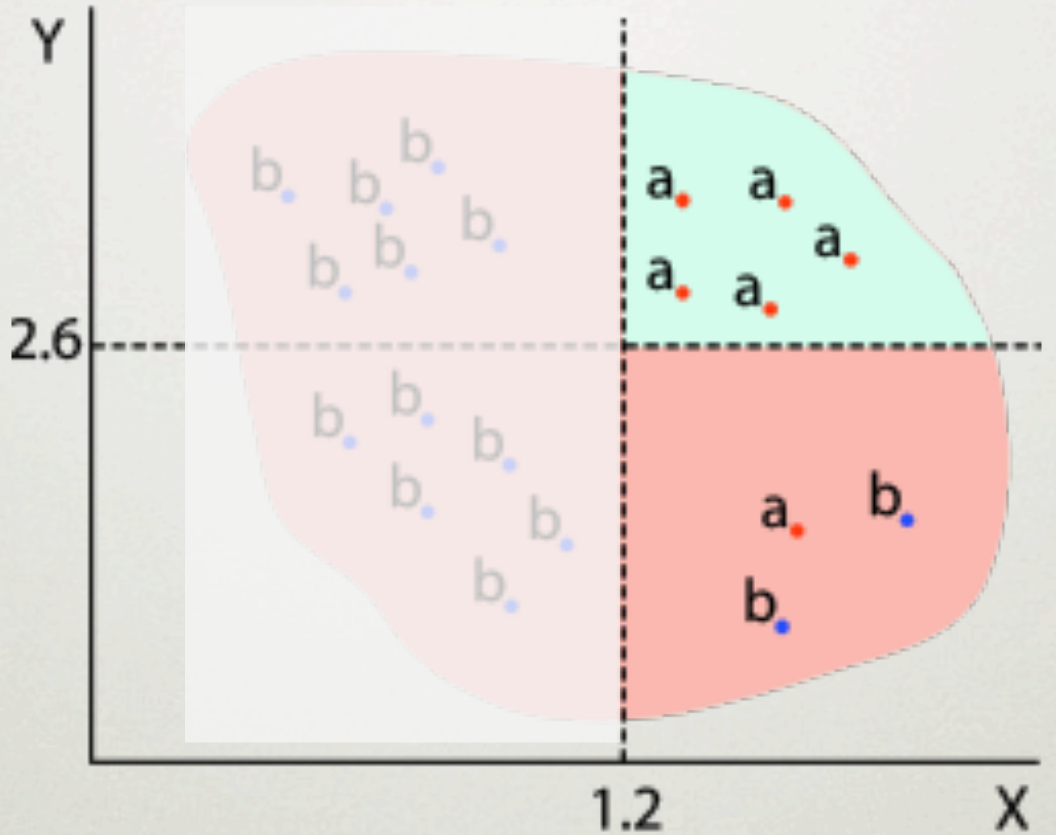
Class b, rule 2



IF X>1.2 and Y ≤ 2.6 then class = b

Example: generating a rule

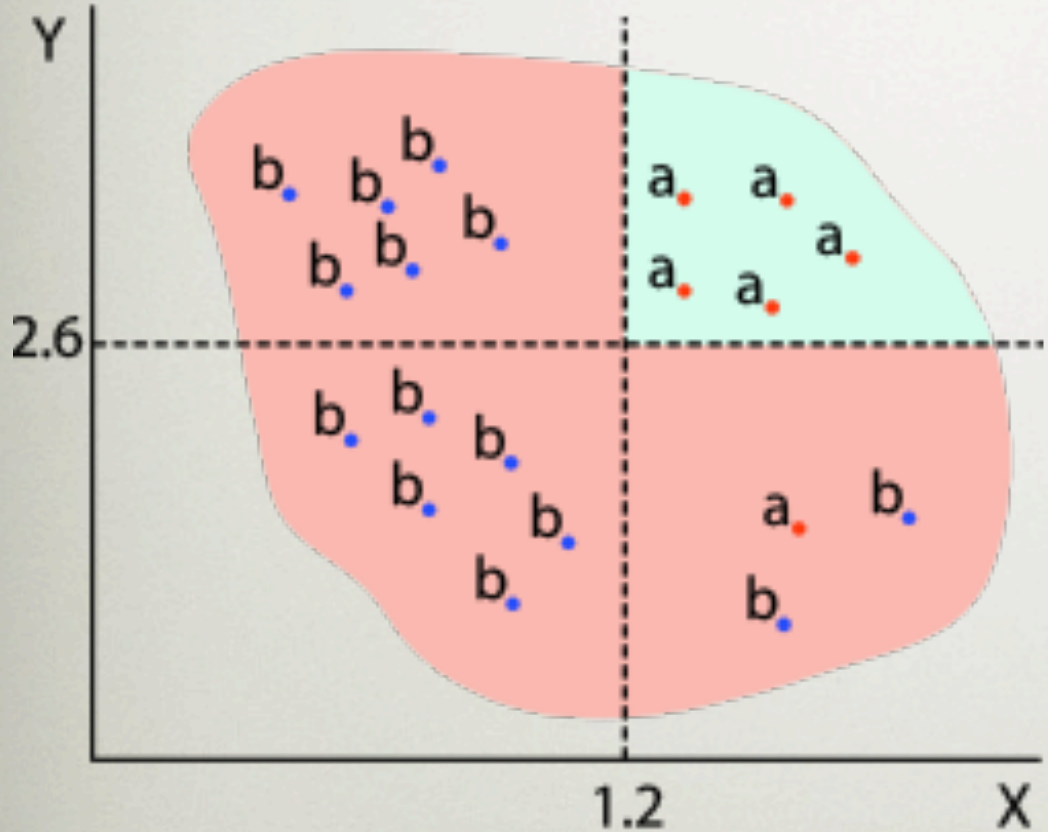
Class b, rule 2



IF X>1.2 and Y ≤ 2.6 then class = b

- More rules could be added for a "perfect" rule set

Example: generating a rule

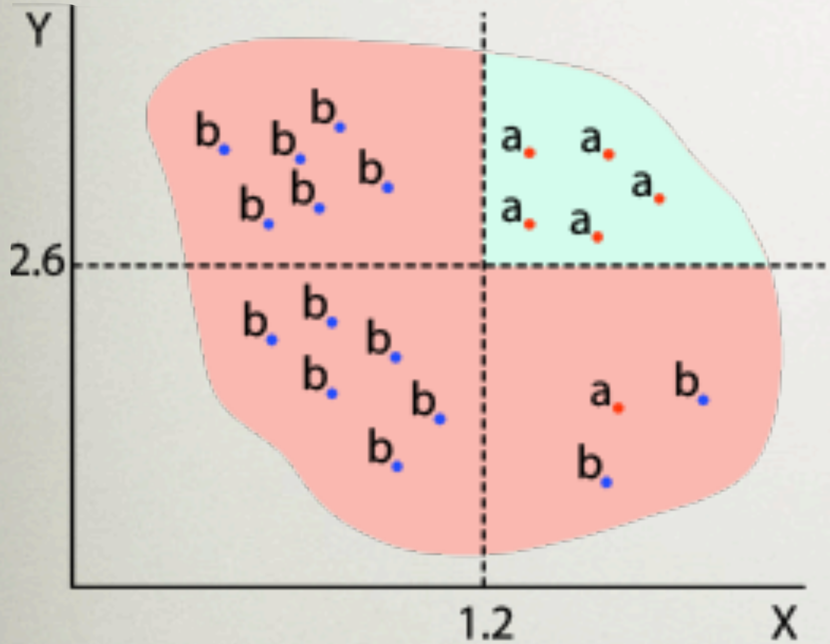


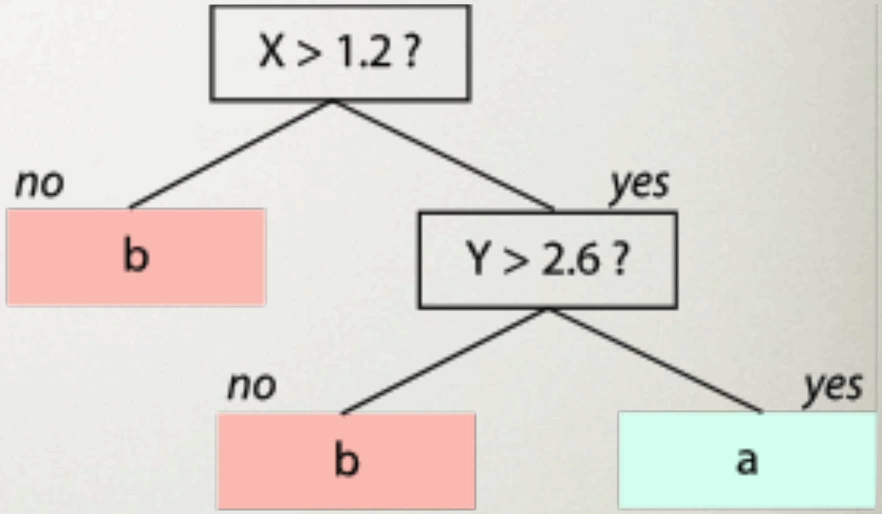
IF X≤1.2 then class = b

ELSE IF X>1.2 and Y ≤ 2.6 then class = b

ELSE class = a

Rules => Trees





IF X≤1.2 then class = b

ELSE IF X>1.2 and Y ≤ 2.6 then class = b

ELSE class = a

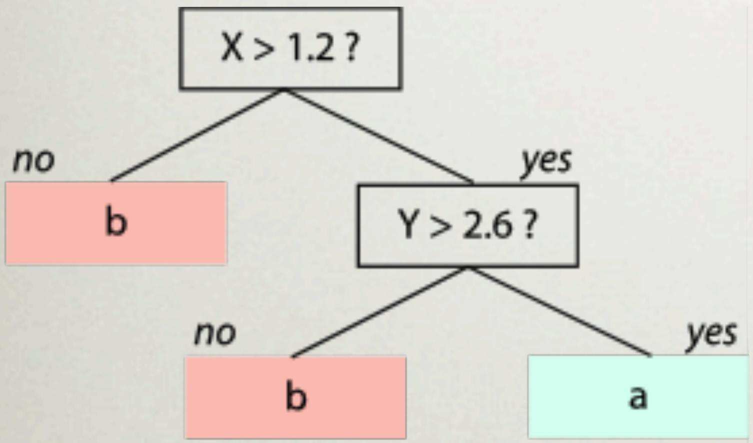
Rules vs. Trees

Rules (PRISM)

IF X≤1.2 then class = b

ELSE IF X>1.2 and Y ≤ 2.6 then class = b

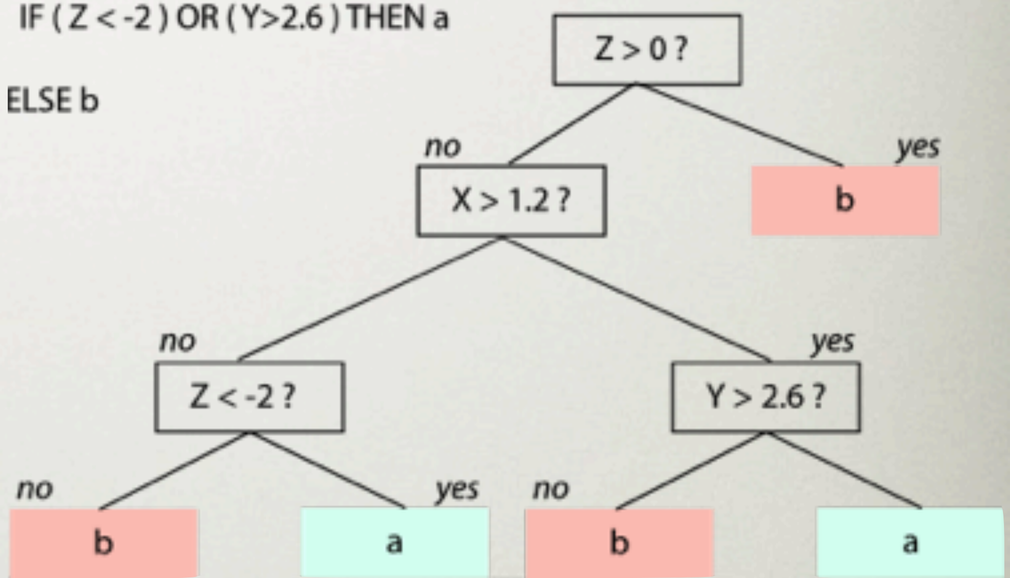
ELSE class = a



Trees (C4.5)

IF (Z > 0) AND (X <= 1.2)
IF (Z < -2) OR (Y>2.6) THEN a

ELSE b



Overall, rules generate clearer subsets, especially when decision trees suffer from replicated subtrees

A simple covering algorithm (PRISM)

- Generate a rule by adding tests that maximize rule's accuracy
- Goal: maximize accuracy ***p/t***
  - *t:* total number of instances covered by rule
  - *p:* `positive' examples of the class covered by rule
  - *t – p:* number of errors made by rule
- Stop when $p/t = 1$ or the set of instances can't be split any further (can't test twice on same attribute)

PRISM Pseudo-code

```
For each class C
    Initialize D to the instance set
    While D contains instances in class C
        Create a rule R with an empty left-hand side that predicts class C
        Until R is perfect (or there are no more attributes to use) do
            For each attribute A not mentioned in R, and each value v,
                Consider adding the condition A = v to the left-hand side of R
                Select A and v to maximize the accuracy p/t
                  (break ties by choosing the condition with the largest p)
            Add A = v to R
        Remove the instances covered by R from D
```

CONTACT LENS DATA

age	spectacle-prescrip	astigmatism	tear-prod-rate	contact-lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Rule: IF *true*, Then *hard*

Next step?

age	**spectacle-prescrip**	**astigmatism**	**tear-prod-rate**	**contact-lenses**
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Example: contact lens data

- Rule we seek to refine:
- Possible tests:

If ?
then recommendation = hard

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

Example: contact lens data

- Rule we seek to refine:
- Possible tests:

If ?
then recommendation = hard

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

(tied, same coverage)

Rule: IF *astigmatism=yes*, Then *hard*

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

Next step?

Further refinement

- Current state:

If astigmatism = yes
  and ?
 then recommendation = hard

- Possible tests:

Age = Young	2/4
Age = Pre-presbyopic	1/4
Age = Presbyopic	1/4
Spectacle prescription = Myope	3/6
Spectacle prescription = Hypermetrope	1/6
Tear production rate = Reduced	0/6
Tear production rate = Normal	4/6

Further refinement

- Current state:

If astigmatism = yes
and ?
then recommendation = hard

- Possible tests:

Age = Young	2/4
Age = Pre-presbyopic	1/4
Age = Presbyopic	1/4
Spectacle prescription = Myope	3/6
Spectacle prescription = Hypermetrope	1/6
Tear production rate = Reduced	0/6
Tear production rate = Normal	4/6

IF *astigmatism=yes & tear_production_rate=normal,* Then *hard*

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Normal	None

Next step?

Further refinement

- Current state:

> If astigmatism = yes
> and tear production rate = normal
> and ?
> then recommendation = hard

- Possible tests:

Age = Young	2/2
Age = Pre-presbyopic	1/2
Age = Presbyopic	1/2
Spectacle prescription = Myope	3/3
Spectacle prescription = Hypermetrope	1/3

- Tie between the first and the fourth test
  - We choose the one with greater coverage

Further refinement

- Current state:

> If astigmatism = yes
> and tear production rate = normal
> and ?
> then recommendation = hard

- Possible tests:

`Age = Young`	2/2
`Age = Pre-presbyopic`	1/2
`Age = Presbyopic`	1/2
`Spectacle prescription = Myope`	3/3
`Spectacle prescription = Hypermetrope`	1/3

- Tie between the first and the fourth test
  - We choose the one with greater coverage

IF *astigmatism=yes & tear_production_rate=normal & spectacle_prescription=myope*, Then *hard*

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Normal	Hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Myope	Yes	Normal	Hard

Next step?

The result

- Final rule:

  If astigmatism = yes
  and tear production rate = normal
  and spectacle prescription = myope
  then recommendation = hard

- Second rule for recommending “hard lenses”:
  (built from instances not covered by first rule)

  If age = young and astigmatism = yes
  and tear production rate = normal
  then recommendation = hard

- These two rules cover all “hard lenses”:
  - Process is repeated with other two classes

Rules vs. Decision Lists

- PRISM with outer loop removed generates a ***decision list*** for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - Order doesn't matter: all rules predict the same class
- Outer loop considers all classes separately: no class order
- Order-independent rules are problematic:
  - Example has multiple classifications (overlapping rules)
    - Choose rule with highest coverage
  - Example has no classification at all (default rule)
    - Default class

Rules vs. Decision Trees

- Methods like PRISM (dealing with one class) are *separate-and-conquer* algorithms:
  - First, a rule is identified
  - Then, all instances covered by the rule are separated out
  - Finally, the remaining instances are "conquered"
- Others, like Decision Trees, are *divide-and-conquer* methods:
  - First, data is split
  - Then, each split modeled/conquered independently

Outline

- Rules
- **Linear Regression**
- Nearest Neighbor

Linear models

- Work most naturally with numeric attributes
- Basic technique for numeric prediction: linear regression
  - Outcome is linear combination of attributes

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

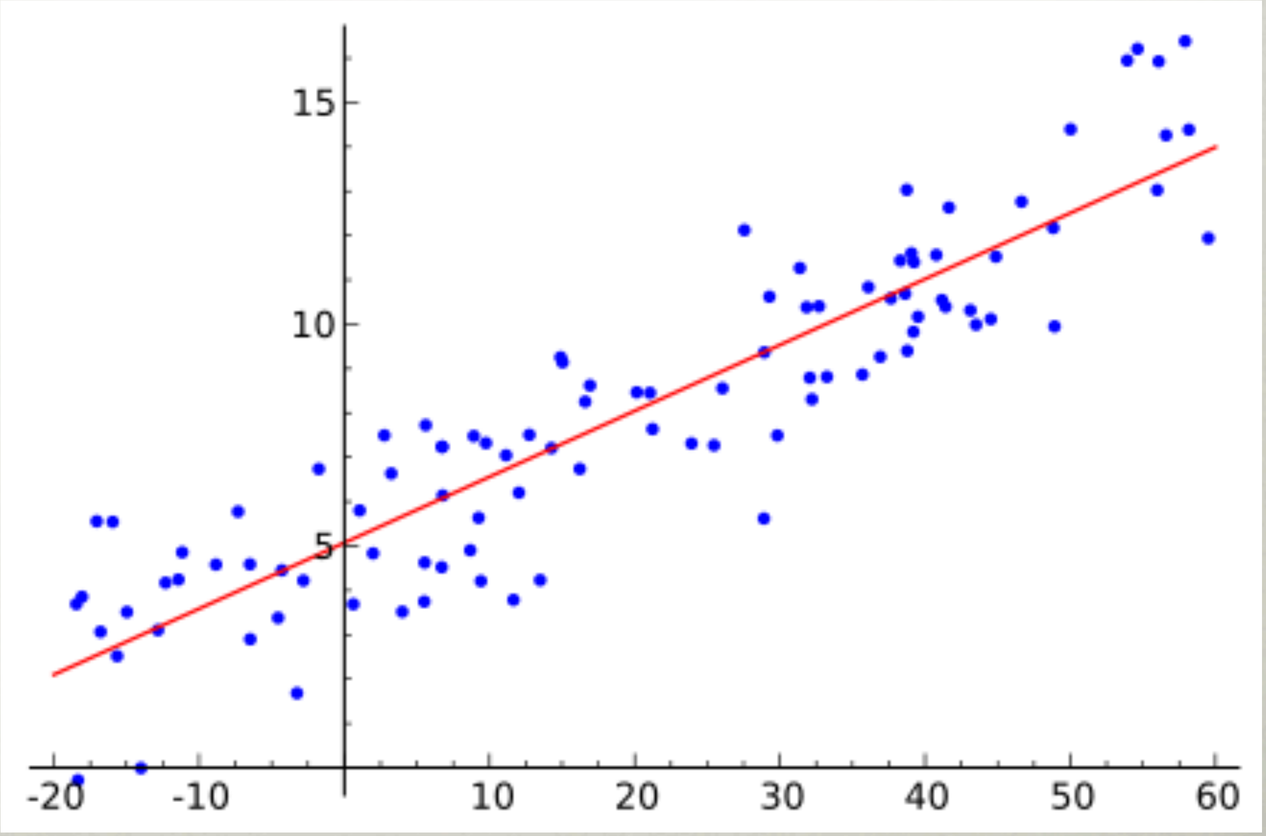
- Weights are calculated from the training data
- Predicted value for first training instance $\mathbf{a}^{(1)}$

$$x^{(1)} = w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)}$$

$a_0 = 1$ (added for convenience)

Linear regression

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

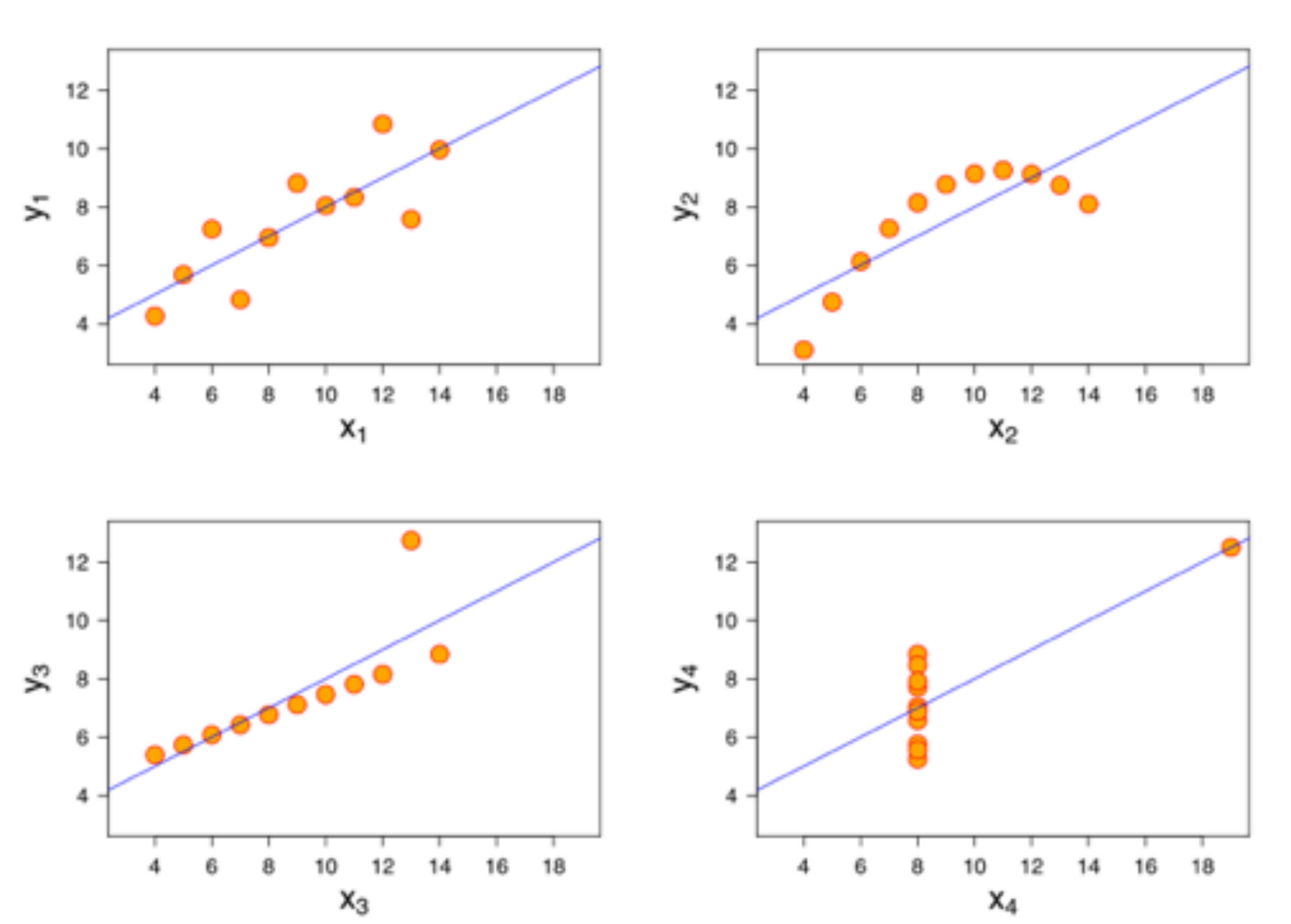


Linear regression

It doesn't always fit

Linear regression

It doesn't always fit



Minimizing the squared error

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

- Choose k +1 coefficients (weights) to minimize the squared error on the training data:

$$squared \quad error = \sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right)^2$$

- Derive coefficients using standard matrix operations
- Accurate method if enough data available
- Minimizing the *absolute error* is more difficult

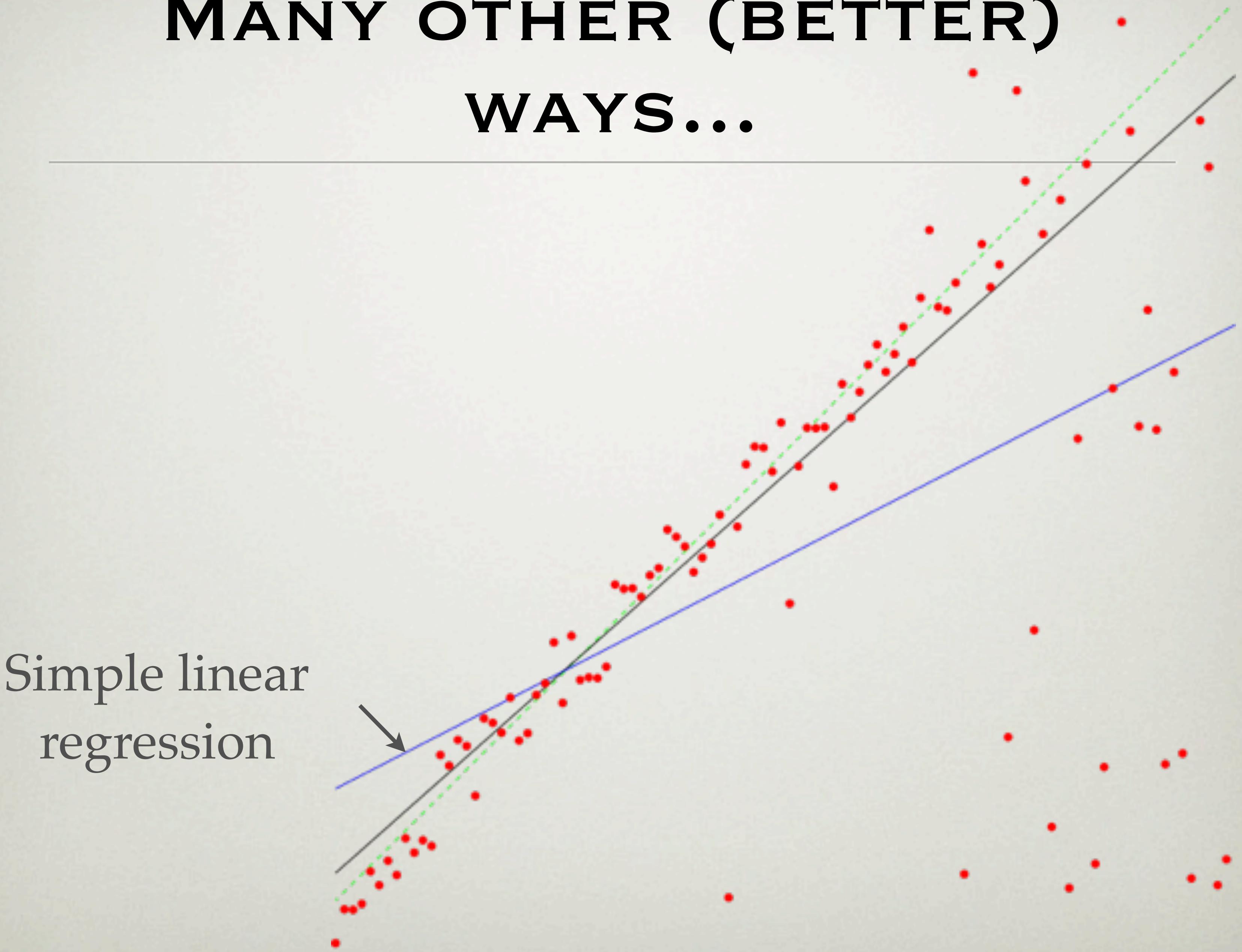
Standard Matrix Operations? (extra)

- Residuals: $\epsilon = \mathbf{X} - \mathbf{Aw}$

$$\sum \epsilon_i^2 = [\epsilon_1\, \epsilon_2 \cdots \epsilon_n] \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} = \epsilon' \epsilon$$

- Minimize $\epsilon'\epsilon = (\mathbf{X} - \mathbf{Aw})' (\mathbf{X} - \mathbf{Aw})$
- Derivative: $d/dw((\mathbf{X} - \mathbf{Aw})' (\mathbf{X} - \mathbf{Aw})) = -2\mathbf{A}'(\mathbf{X} - \mathbf{Aw})'$
- Minimal for: $-2\mathbf{A}'(\mathbf{X} - \mathbf{Aw})' = \mathbf{0}$
- Thus: $\mathbf{A}'\mathbf{X} = \mathbf{A}'\mathbf{Aw}$
- Solve: $w = (\mathbf{A}'\mathbf{A})^{-1} \mathbf{A}'\mathbf{X}$

Many other (better) ways...



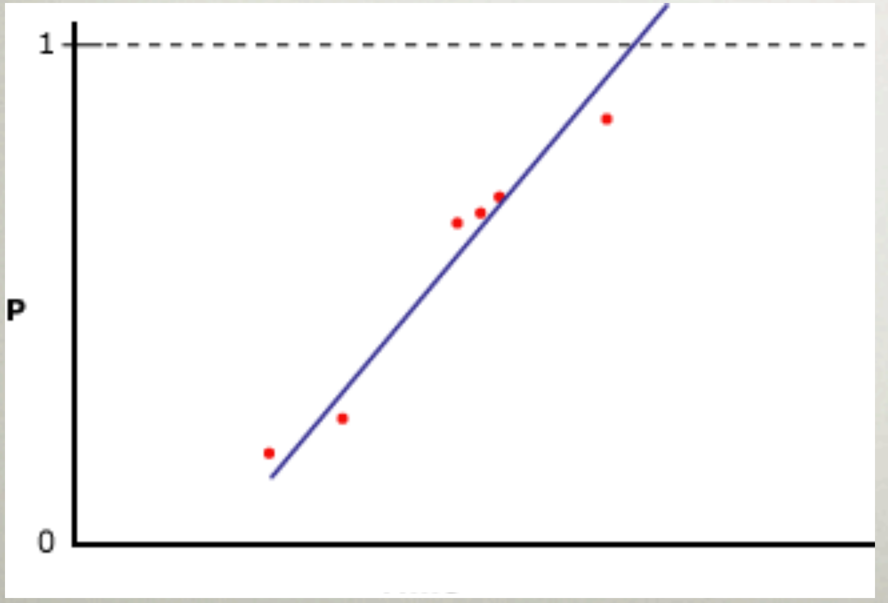
Regression for Classification

- *Any* regression technique can be used for classification
  - Similar to a membership function
  - Training:
    - Perform a **regression for each class**, setting the output to 1 for training instances that belong to class, and 0 for others
  - Prediction:
    - Predict class corresponding to model with largest output value
- For linear regression this is known as *multi-response linear regression*

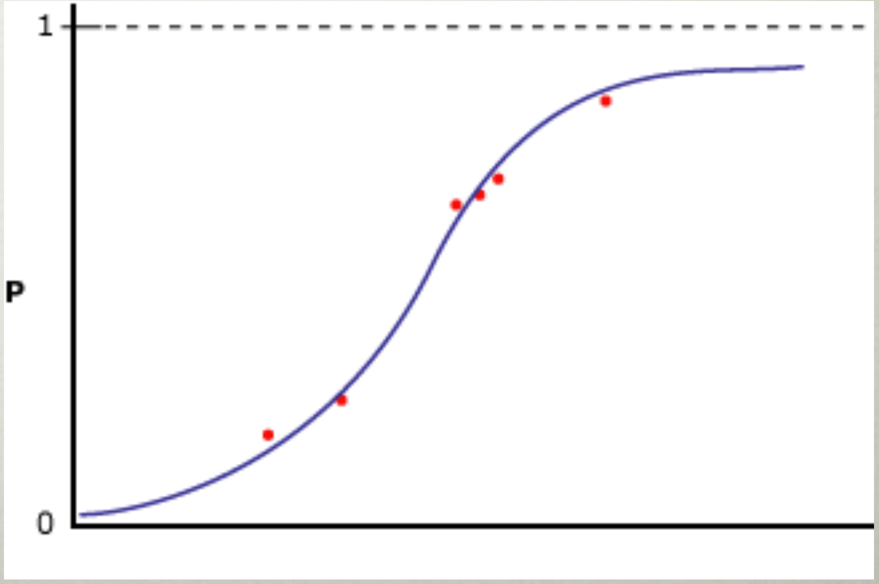
Logistic regression

- Problem:
  - model output is not a proper probability (can be >1)
  - least squares assumes that errors are statistical independent and normally distributed (wrong: only 0′s and 1′s)

Linear regression



Logistic regression



Logistic regression

- *Logistic* regression: alternative to linear regression
  - Designed for classification problems
  - Transform {0,1} values to [-inf, +inf], build model, transform to [0,1]
  - Similar to `odds'
    - P(y=1)=0.75 -> P/(1-P) = 3 -> 1 is 3x more likely than 0
  - Replace target variable $P[1 | w_0, w_1, \ldots w_k]$ by *logit transform*

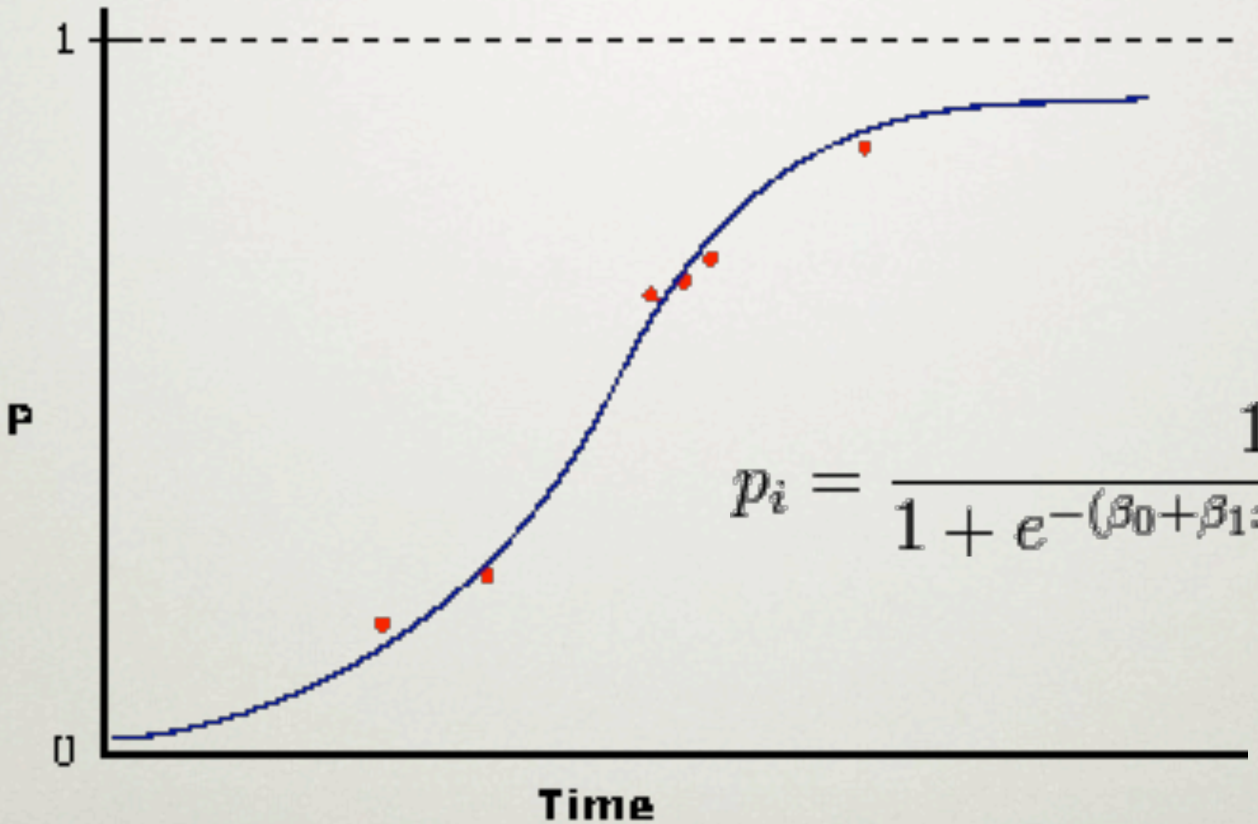
$$\log\left(\frac{P}{1-P}\right) = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

P= Class probability = $P[1 | w_0, w_1, \ldots w_k]$

- Choose **w** to maximize log-likelihood (not so simple)
  - *maximum likelihood* method

Logistic regression

- Resulting model:



$$p_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_k x_{k,i})}}.$$

- Classification: class with highest probability

LINEAR MODELS
FINAL THOUGHTS

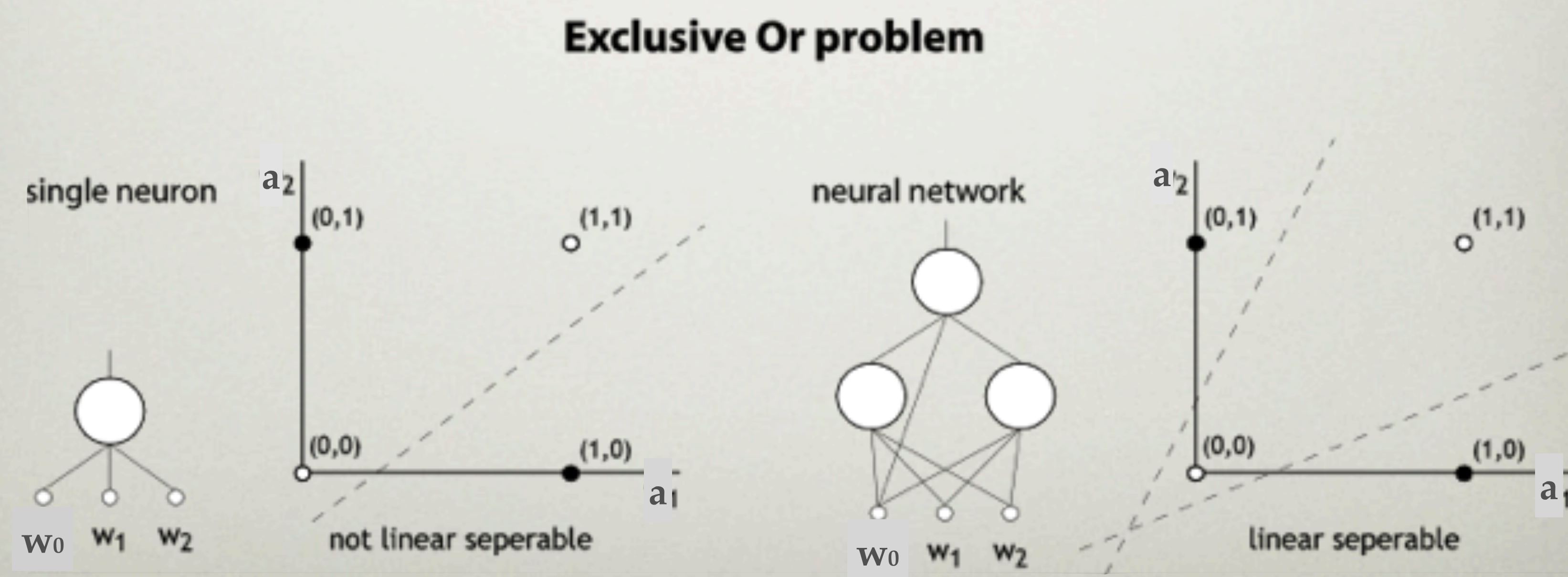
- Not appropriate if data exhibits non-linear dependencies
- But: can serve as building blocks for more complex schemes (i.e. model trees: trees with models in the leaves)
- Example: multi-response linear regression defines a *hyperplane* for any two given classes
  - Given two weight vectors for two classes, predict class 1 when:

$$w_0^{(1)}a_0 + w_1^{(1)}a_1 + w_2^{(1)}a_2 + \ldots + w_k^{(1)}a_k > w_0^{(2)}a_0 + w_1^{(2)}a_1 + w_2^{(2)}a_2 + \ldots + w_k^{(2)}a_k$$

$$(w_0^{(1)} - w_0^{(2)})a_0 + (w_1^{(1)} - w_1^{(2)})a_1 + (w_2^{(1)} - w_2^{(2)})a_2 + \ldots + (w_k^{(1)} - w_k^{(2)})a_k > 0$$

LINEAR MODELS FINAL THOUGHTS

- Linear classifiers have limitations, e.g. can't learn XOR
  - But: combinations of them can (→ Neural Nets)
  - Perceptron (1-layer neural network): adjust weights to move hyperplane towards misclassified examples by adding/subtracting the example



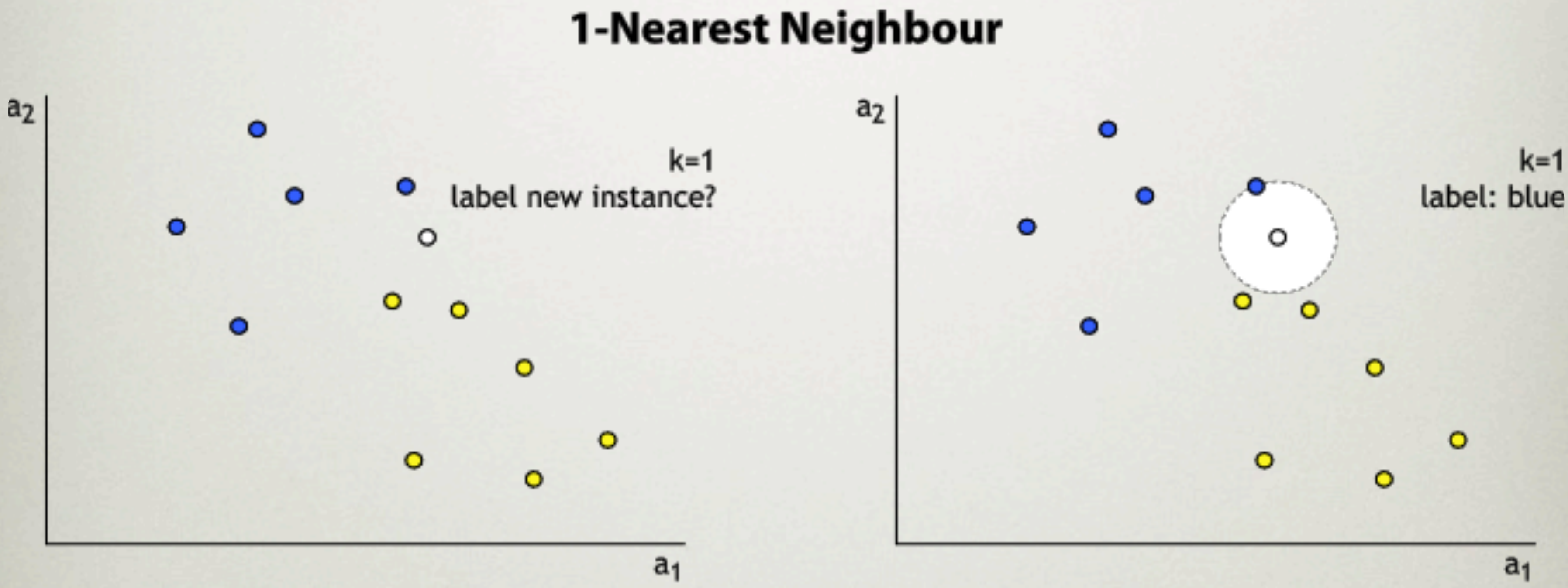
Outline

- Rules
- Linear Regression
- **Nearest Neighbor**

Instance-based representation

- Simplest form of learning: *rote learning*
  - Don't build a model, `remember' the training instances
  - Training instances are searched for instance that most closely resembles new instance
  - The instances themselves represent the knowledge
  - Also called *instance-based learning*, or *lazy learning*
- *Similarity function* defines which instances are `similar'
- Methods:
  - *nearest-neighbor*
  - *k-nearest-neighbor*
  - …

1-NN EXAMPLE


1-Nearest Neighbour
a2
k=1
label new instance?
a1
a2
k=1
label: blue
a1

The distance function

- One numeric attribute
  - Distance = difference between the two attribute values involved (or a function thereof)
- Several numeric attribute
  - e.g. Euclidean distance is used and attributes are normalized
- Nominal attributes:
  - Distance = 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - Usually not, weighting the attributes might be necessary

Euclidean distance

- Most instance-based schemes use *Euclidean distance*:

$$\sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + \ldots + (a_k^{(1)} - a_k^{(2)})^2}$$

  $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$: two instances with k attributes

- Taking the square root is not required when comparing distances
- Other popular metric: *city-block (Manhattan) metric*
  - Adds differences without squaring them

Normalization

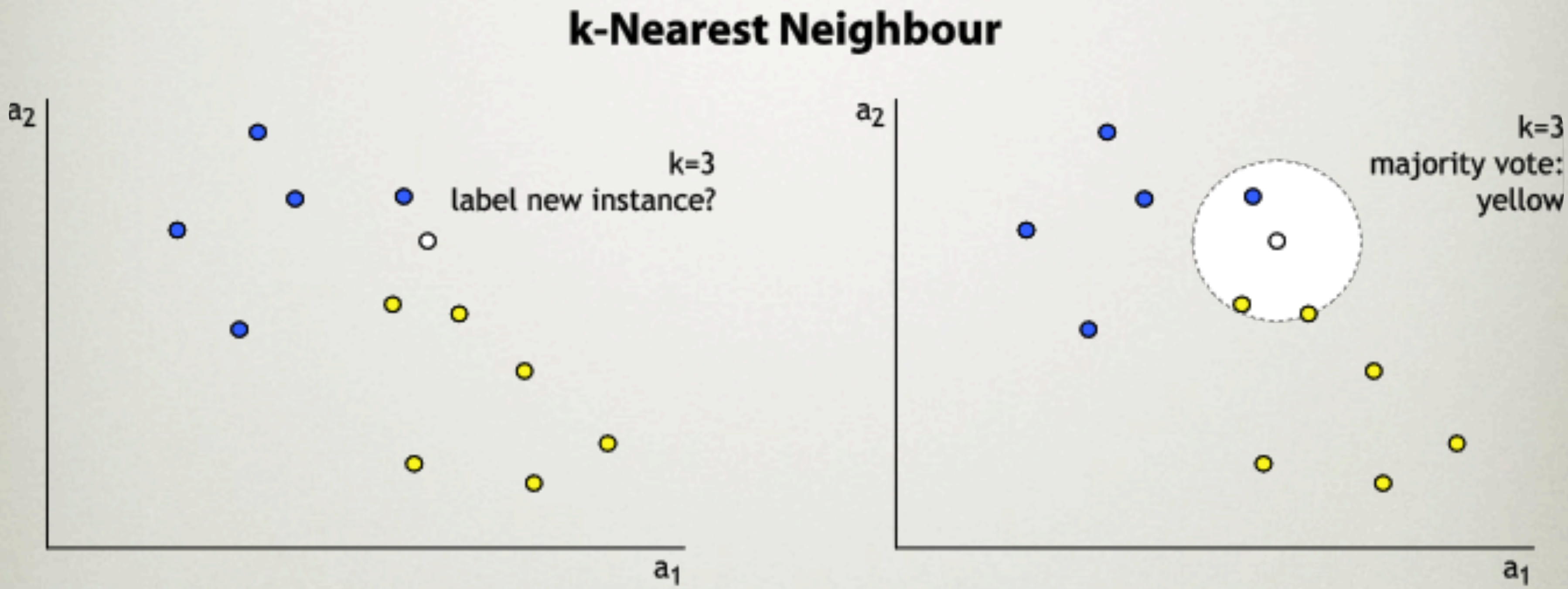
- Different attributes are measured on different scales ⇒ need to be *normalized*:

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} \quad \text{or} \quad a_i = \frac{v_i - Avg(v_i)}{StDev(v_i)}$$

  v_i: the actual value of attribute i

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

k-NN example



- k-NN approach: majority vote (or other function) to derive label
- k = regularization parameter: higher k means smoother decision boundary, less overfitting

Nearest Neighbors

- Very accurate (for few attributes, lots of data)
  - *Curse of dimensionality:* Every added dimension increases distances, exponentially more training data needed
- Typically very slow (at prediction time):
  - simple versions scan all training data to make prediction
  - better training set representations exist: kD-tree, ball tree,...
- Assumes all attributes are equally important
  - Remedy: attribute selection or weighted distance measures
- Noisy data:
  - Take a majority vote over the k nearest neighbors
  - Removing noisy instances from dataset (difficult!)
- Statisticians have used k-NN since early 1950s
  - If $n \rightarrow \infty$ and $k/n \rightarrow 0$, error approaches minimum