Unit Schedule: Modules

Module	Week	Content	Ross
1.	1	introduction to modelling	1,2
2.	2	probability refresher	3
	3	random vars & expected values	4
	4	special distributions	5
3.	5	statistical inference	6&7
	6	confidence intervals	7
	7	hypothesis testing	8
4.	8	dependence & linear regression	9
	9	classification, clustering & mixtures	
5.	10	**random numbers & simulation**	15(bits)
	11	**basic machine learning**	
6.	12	modelling, validation & review	

Revision at *https://flux.qa/43FMK4*

FIT5197 Statistical Data Modelling

Module 5

Further Modelling

2020 Lecture 11

Monash University

Other Modelling Methods

Outline

Supervised Learning – Recap

- Imagine we have measured $p + 1$ variables on n individuals (people, objects, things)
- One variable is our target
- We would like to predict our target using the remaining p variables (predictors) - can be done by minimising errror or maximising likehood
- If the variable we are predicting is categorical, we are performing classification
- If the variable we are predicting is numerical, we are performing regression

Example problem - object recognition: *Imagenet*
and *Imagenet performance over time*

Machine Learning

- Machine learning
  - Also data mining, artificial intelligece
- Intersection of computer science and statistics
- Machine learning methods often algorithmic
- Usually non-linear and flexible
  - Make few assumptions about the data
- Sometimes hard to interpret the resulting “model”
  - Usually focussed on prediction

Machine Learning Methods

- Some well known machine learning methods:
  - Decision trees
  - Random forests
  - *k*-nearest neighbours (kNN)
  - Support vector machines (SVMs)
  - Neural networks
  - Deep neural networks (deep learning)
  - Clustering
  - Mixture modelling

Machine Learning in R

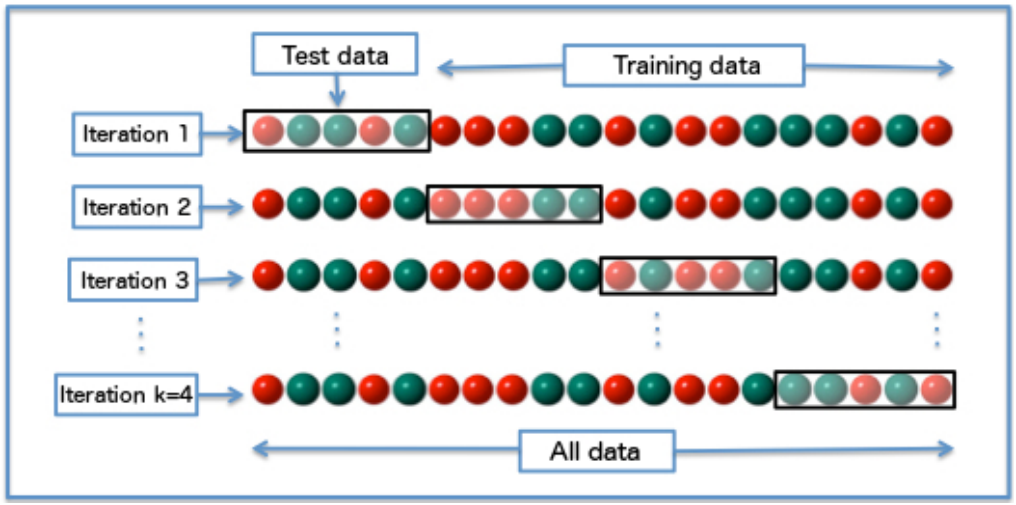
- Decision trees
  - R packages: `tree`, `rpart`
- Artificial neural networks
  - R packages: `nnet`, `neuralnet`, `brnn`
- Support vector machines
  - R packages: `e1071`, `pathClass`, `penalizedSVM`
- Genetic algorithms
  - R packages: `GA`, `genalg`, `mcga`
- Clustering
  - R packages: `cluster`, `mixtools`, `rebmix`

Outline

Cross Validation Method

- Wish to estimate mean error (ME) of some form for a model $\mathcal{M}$
  - e.g., ME is squared error or 0-1 loss
  - e.g., $\mathcal{M}$ is a decision tree with 10 leaves
- To estimate ME for a given $\mathcal{M}$ using cross validation
  1. We randomly partition our data into two sets:
     - A training set $\mathbf{y}_{\text{train}}$,
     - and a testing set $\mathbf{y}_{\text{test}}$
  2. Fit a model $\mathcal{M}$ to the training data $\mathbf{y}_{\text{train}}$
  3. Compute the ME of this model on the testing data $\mathbf{y}_{\text{test}}$
  4. Repeat this procedure *m* times and average the ME
- This gives us an estimate of how well our model would predict future data from the same population
  $\Longrightarrow$ the larger the *m*, the more stable the estimate is
- How could you use this for linear/logistic regression?

K-fold Cross Validation



by

User:Joan.domenech91

CC BY-SA 3.0, Wikimedia

K-fold CV error estimate for fixed model $\mathcal{M}$:

1. Partition data into K equal sized, disjoint subsets $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \mathbf{y}^{(3)}, \ldots, \mathbf{y}^{(K)}$
2. For $k = 1$ to K
   - 2.1 Fit model $\mathcal{M}$ to all $\mathbf{y}^{(i)}$ except for $i = k$
   - 2.2 Use fitted model to predict onto $\mathbf{y}^{(k)}$, yielding error $\widehat{ME}_k$.
3. Average prediction errors, $\widehat{ME} = \frac{1}{K} \sum_{i=1}^{K} \widehat{ME}_k$

K-fold Cross Validation, cont.

- can repeat m times and average yet again
  - the larger the m, the more stable the estimates (but slower)
- how to choose K?
  - lower K is cheaper
  - higher K less biased but more variance
- when $K = |\mathbf{y}|$, so each test set is of size 1, it is called leave one put cross validation (LOOCV)
- use K-fold CV to choose between different algorithms or methods, or different parameter settings of the one method
- there is an approximate theoretical justification for LOOCV
  - it is a commonly used and useful statistical hack

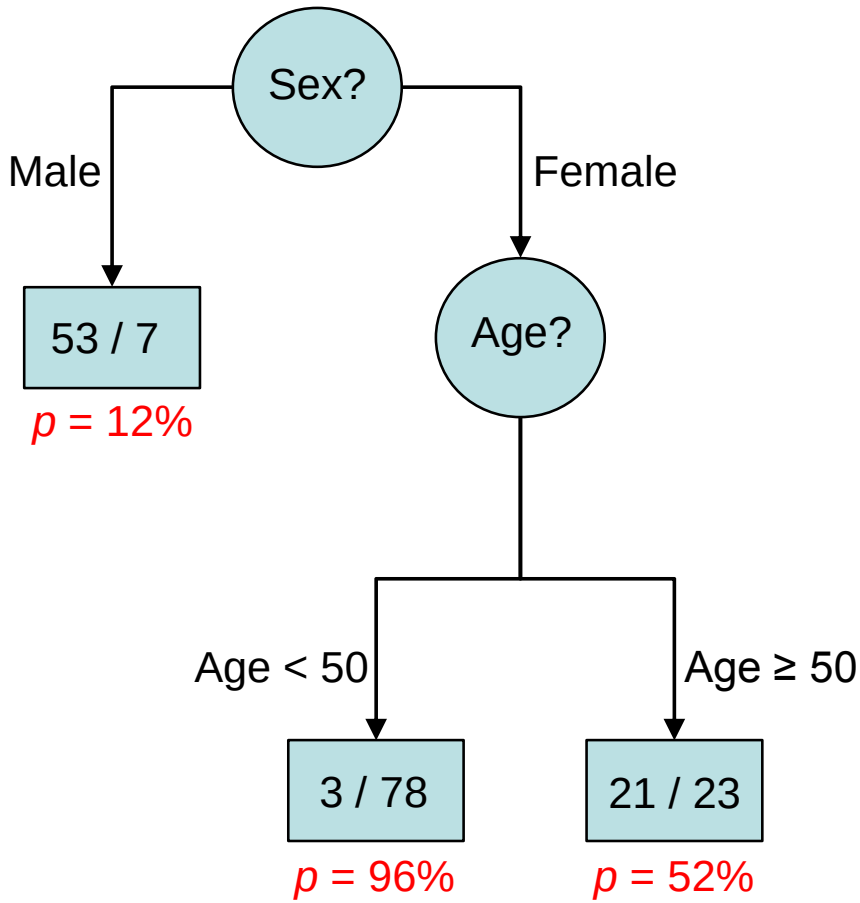
Cross Validation for Choosing Complexity

- Define γ to be some "complexity" parameter for the model
  - In linear/logistic regression, could be the number of predictors
  - We will see other "complexity" parameters today
- Let $\mathcal{M}(\gamma)$ be a model with complexity γ fitted to some data
- Which choice of model complexity γ is appropriate?
- General approach is to use cross-valididation to choose γ
  $\Longrightarrow$ find complexity that leads to good prediction error

Outline

Decision Tree Example

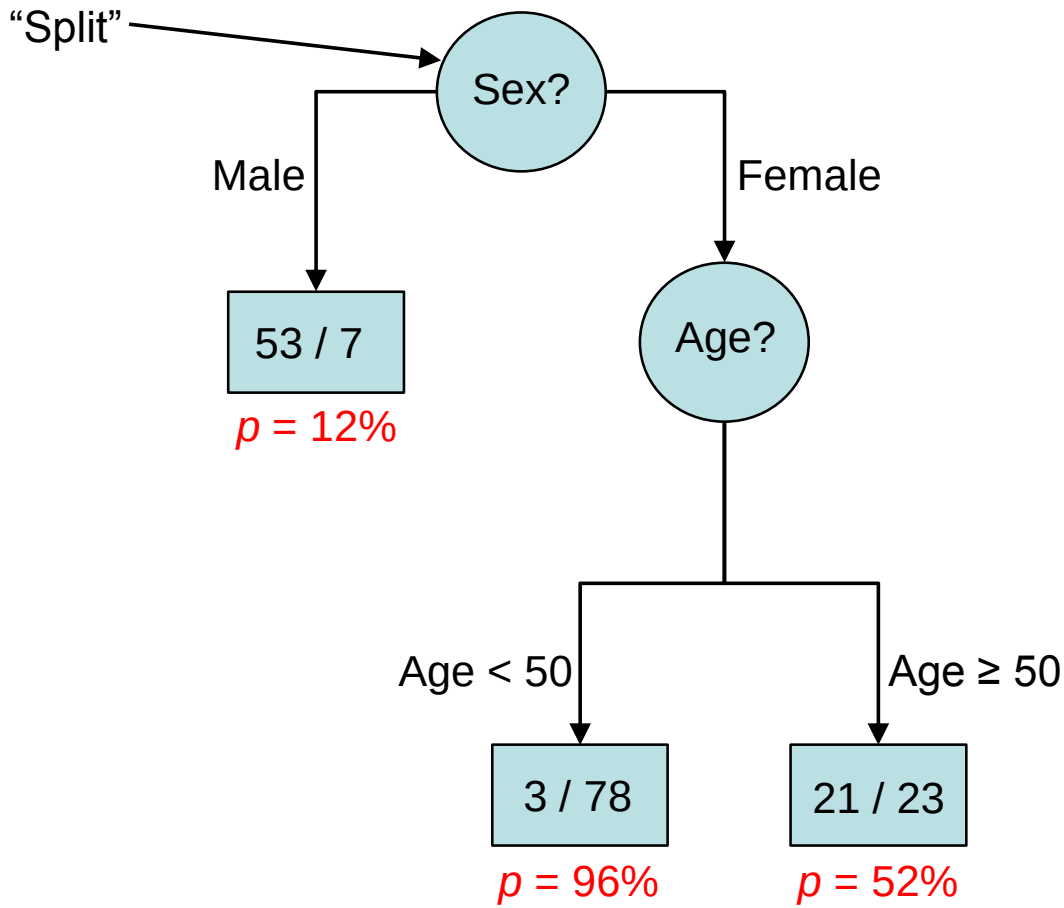
Example tree: predicting high blood pressure



Decision tree definitions

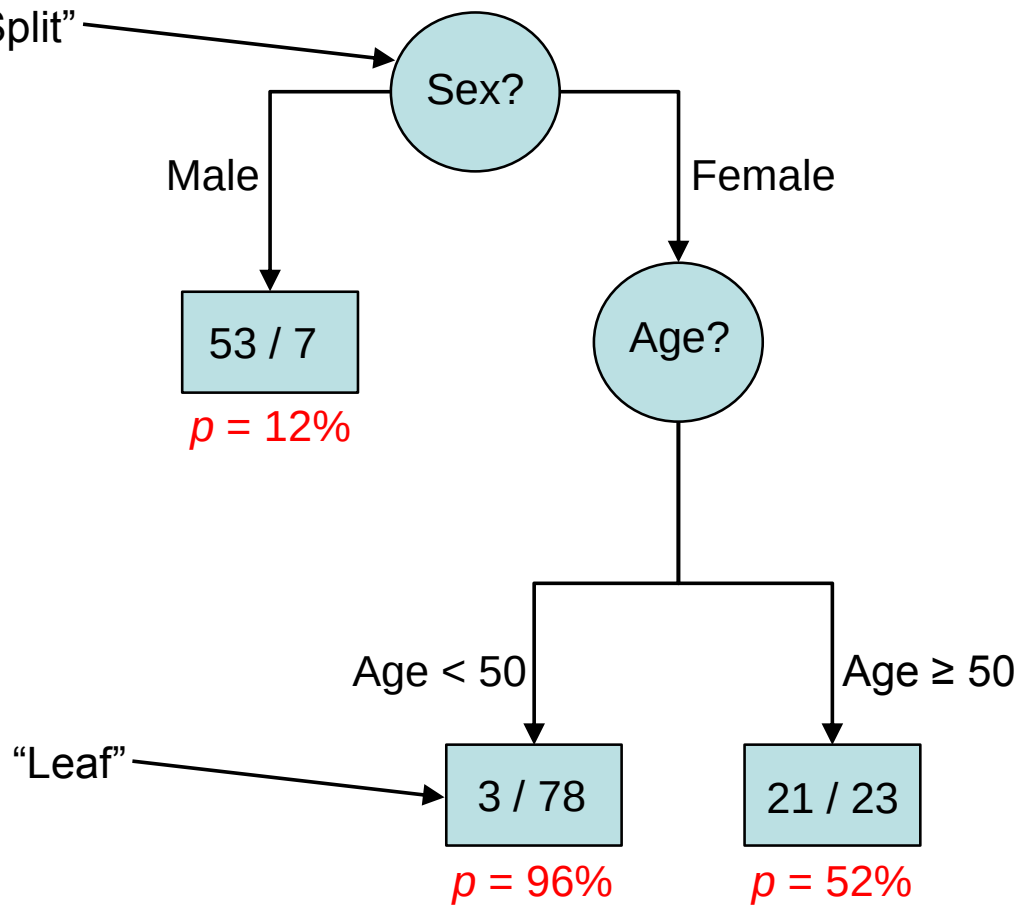
Decision Tree Example (2)

Example tree: predicting high blood pressure



Decision Tree Example (3)

Example tree: predicting high blood pressure



Decision Trees Overview

- A decision tree is a type of supervised learning model
- It takes inputs $x_1, \ldots, x_p$ and produces a prediction

$$\hat{y}(x_1, \ldots, x_p) = f(x_1, \ldots, x_p)$$

- So in this sense, similar to linear regression
- The difference is in the form of $f(\cdot)$
- For decision tree, $f(\cdot)$ is highly non-linear
  $\Longrightarrow$ but still retain a high degree of interpretability
- Can be applied to regression and (multi-class) classification

Decision Tree Explanation

- So a decision tree works by splitting the predictor space up into L disjoint regions $R_1, \ldots, R_L$
- Each region is a "leaf" of the tree and contains a model fitted to the data in the region
  - For regression, the model is usually a normal distribution
  - For classification, it is a Bernoulli distribution
- To predict with a decision tree given $x_1, \ldots, x_p$
  - Traverse the tree, taking the appropriate path for our predictors
  - When we arrive at a leaf, use that model
- The complexity of the model is L (number of leaves)
  $\Longrightarrow$ the more leaves, the better the tree can fit the data

Learning Decision Trees

- How to learn a decision tree from data?
- Given some data, by taking L big enough we can always fit the data *perfectly* using a decision tree
  - But of course, we will overfit
- Instead, use model selection: assign a score to each tree
  - Cross-validation scores
  - other information criteria: AIC, BIC, etc.
- Find the tree with the smallest score
  - Trade-off goodness-of-fit against tree complexity

Learning Decision Trees, cont.

- The space of possible trees is enormous if p is even moderate
- Instead, we use approximate search algorithms
- Most basic: forward search with info. criterion
  1. Start with just one leaf node
  2. Try splitting on every predictor and compute criterion scores
  3. If no split improves tree, stop
  4. Choose the split that results in tree with smallest score
  5. Go back to Step 2

What Makes a Good Split?

- At every step of the search, there are usually many predictors we could use to split our data
- How do we say that one split is better than another?
  $\Longrightarrow$ one measure is through the negative log-likelihood
- For simplicity, let us consider a tree with binary targets
  - Let $\mathbf{y} = (y_1, \ldots, y_n)$ be the targets of data in some leaf
  - Let

$$n_1 = \sum_{i=1}^{n} y_i$$

be the total number of "1"s in our data, and $n_0 = n - n_1$ be the total number of "0"s

A Good Split

- For binary data, we can use the Bernoulli distribution; likelihood is

$$p(\mathbf{y} \mid \theta) = \theta^{n_1}(1-\theta)^{n_0}$$

- Recall that the maximum likelihood estimator $\hat{\theta} = n_1/n$
- The minimised negative log-likelihood of the data is then

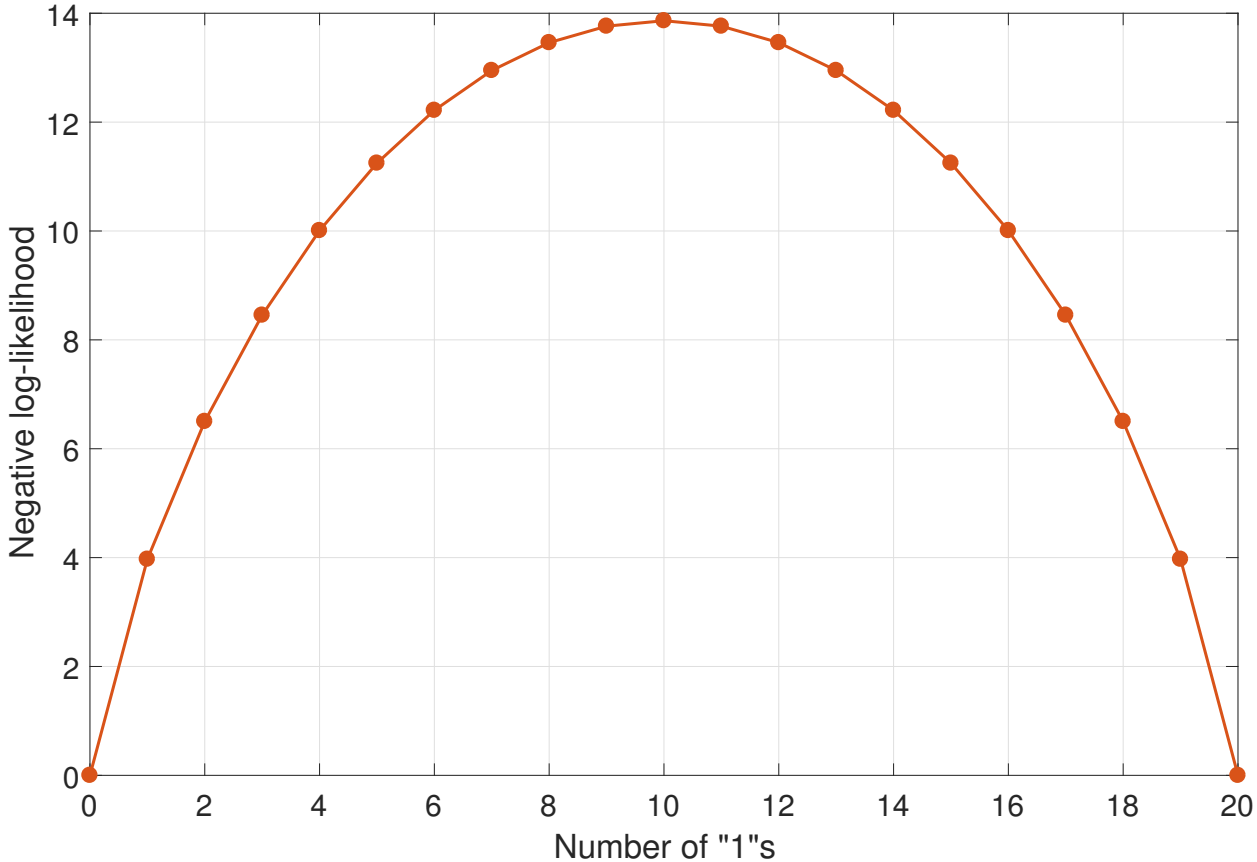
$$-\log p(\mathbf{y} \mid \hat{\theta}) = -n_1 \log\left(\frac{n_1}{n}\right) - n_0 \log\left(\frac{n_0}{n}\right)$$

- But traditionally, we instead use (empirical) entropy

$$H(y \mid \hat{\theta}) = -\frac{1}{n}\log_2 p(\mathbf{y} \mid \hat{\theta}) = -\frac{n_1}{n}\log_2\left(\frac{n_1}{n}\right) - \frac{n_0}{n}\log_2\left(\frac{n_0}{n}\right)$$

- This is:
  - largest when $n_1/n = 1/2$;
  - smallest when $n_1 = 0$ or $n_1 = n$
- The "purer" the data, the smaller the entropy

Plotting Entropy



Minimised entropy for $n = 20$ samples as the number of “1”s in the sample varies. Note the curve is symmetric, and is minimised when the data comprises either all “1”s or no “1”s (is most “pure”).

Toy Example

- Toy example
  - $n = 10$ data points, binary targets
  - Two binary predictors, x_1 and x_2

y	1	1	0	1	0	0	0	1	1	0
x_1	0	0	0	0	0	1	1	1	1	1
x_2	1	1	0	1	1	0	0	0	1	0

- Without splitting, we have a single leaf (a "root")

$$H(y) = -5/10 \log_2(5/10) - 5/10 \log_2(5/10) = 1.0$$

- When building a tree, we could split our data into two using either predictor x_1 or predictor x_2
- We would prefer the split that leads to the smallest average entropy

Toy Example, cont.

y	1	1	0	1	0	0	0	1	1	0
x_1	0	0	0	0	0	1	1	1	1	1
x_2	1	1	0	1	1	0	0	0	1	0

- Let us trying splitting on predictor x_1
- Entropy of the two leaves is then:

$$H(y \mid x_1 = 0) = -3/5 \log_2(3/5) - 2/5 \log_2(2/5) \approx 0.9710$$
$$H(y \mid x_1 = 1) = -2/5 \log_2(2/5) - 3/5 \log_2(3/5) \approx 0.9710$$

- So the average conditional entropy is approx.

$$\begin{aligned} H(y|x_1) &= p(x_1{=}0)H(y|x_1{=}0) + p(x_1{=}1)H(y|x_1{=}1) \\ &= \frac{5}{10}0.9710 + \frac{5}{10}0.9710 = 0.9710 \end{aligned}$$

$\Longrightarrow$ a bit better than “no split”

Toy Example, cont.

y	1	1	0	1	0	0	0	1	1	0
x_1	0	0	0	0	0	1	1	1	1	1
x_2	1	1	0	1	1	0	0	0	1	0

- Now try splitting on predictor x_2
- Entropy of the two leaves is then:

$$\begin{aligned} H(y \mid x_2 = 0) &= -1/5 \log_2(1/5) - 4/5 \log_2(4/5) \approx 0.7219 \\ H(y \mid x_2 = 1) &= -4/5 \log_2(4/5) - 1/5 \log_2(1/5) \approx 0.7219 \end{aligned}$$

- So the conditional entropy $H(y|x_2)$ is approx. 0.7219
  $\Longrightarrow$ a lot better than "no split" of 1.0
- So splitting on x_2 leads to purer leaf nodes

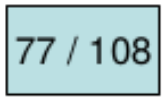
What Makes a Good Split?

- To split on numeric attribute x_j
  - Find the best value c such that the leaves formed by
    1. taking y when $x_j < c$, and
    2. taking y when $x_j \geq c$

    have the smallest entropy
- Remember, splitting data will always decrease entropy
- Hence, entropy can be used to guide splits in a greedy manner
- But to select a tree, we need to penalize the tree by complexity
- Either cross-validation approach or other model selection method

Example Decision Tree

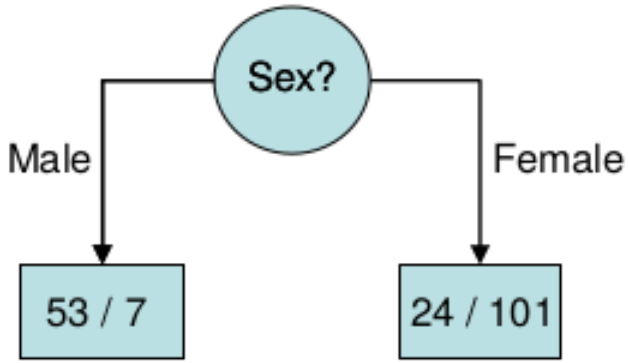
Example: predicting HBP = “high blood pressure”, start with a single leaf node (a “root”)



$$H(HBP) = \frac{77}{185}\log_2\frac{185}{77} + \frac{108}{185}\log_2\frac{185}{108} = 0.9797$$

Example Decision Tree, cont.

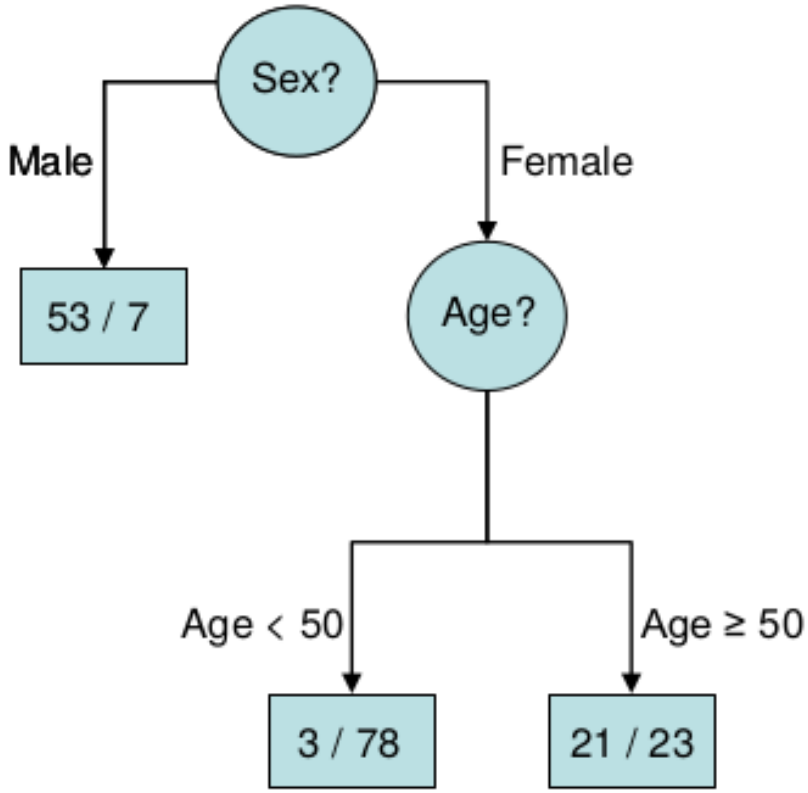
Split on Sex – score improves



$$H(HBP|Sex) = \frac{60}{185} \times 0.5197 + \frac{125}{185} \times 0.7056 = 0.6453$$

Example Decision Tree, cont.

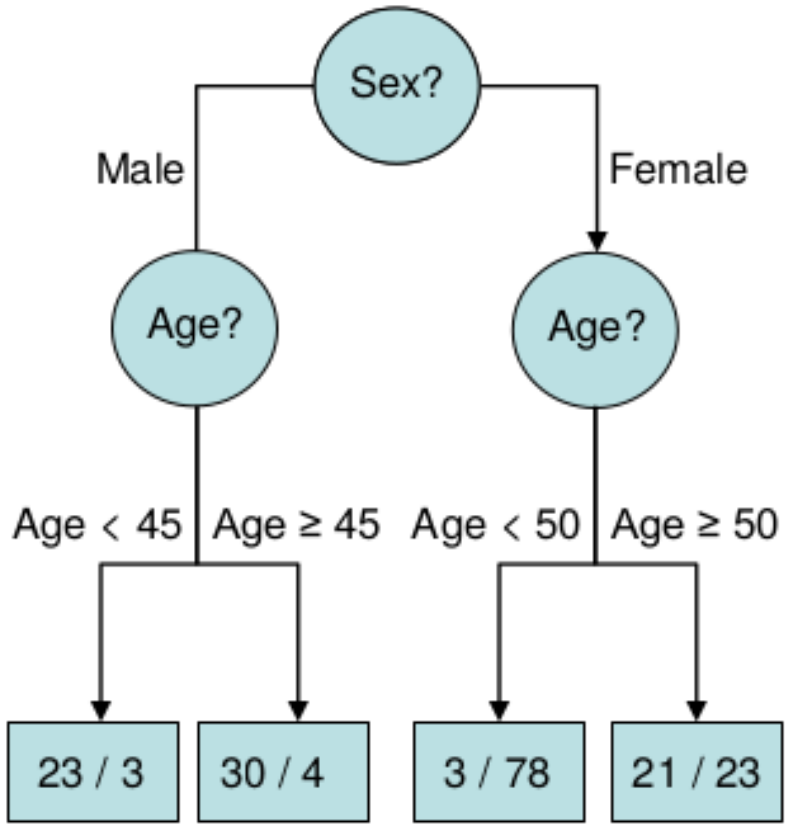
Split on Age when Sex is “Female” – score improves



$$H(HBP|Sex, Age/50 \text{ for } Sex{=}Female) =$$
$$\frac{60}{185} \times 0.5197 + \frac{81}{185} \times 0.2285 + \frac{44}{185} \times 0.9985 = 0.5061$$

Example Decision Tree, cont.

Split on Age when Sex is "Male" – score does not improve



$$H(HBP|Sex, Age/50 \text{ for } Sex{=}Female, Age/45 \text{ for } Sex{=}Male) =$$
$$\frac{26}{185} \times 0.5159 + \frac{34}{185} \times 0.5226 + \frac{81}{185} \times 0.2285 + \ldots = 0.5061$$

m-repeat K-fold CV for Trees

Treat the number of leaves L as complexity parameter.

Use K-fold CV to select L, but repeat m times.

- Outer loop: try $L = 1, \ldots, L_{\max}$
  1. Repeat for $i = 1$ to m
     - 1.1 Partition data into K equal sized, disjoint subsets

$$Y^i = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \mathbf{y}^{(3)}, \ldots, \mathbf{y}^{(K)}\}$$

     - 1.2 $error_i$ = K-fold CV error using partition Y_i to build tree to length L
  2. Average m accumulated prediction errors ($error_1$, ..., $error_m$)
- Choose L that has smallest accumulated error
  - Grow tree with L leaves on all data as final model

Revision at *https://flux.qa/43FMK4*

Learning Decision Trees - Pruning Methods

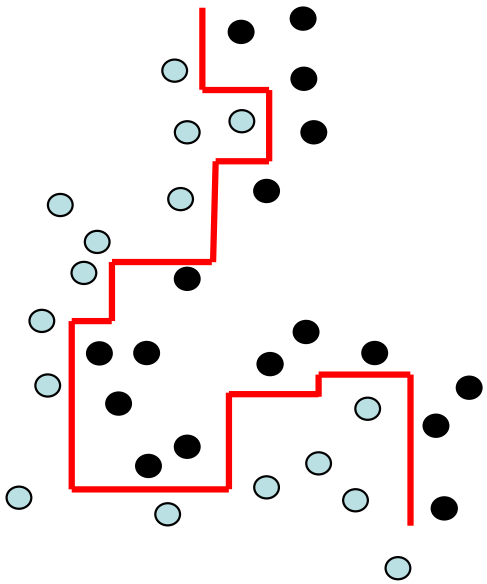
- In this approach, we first grow a large, overfitted tree
  - Use forward search and split on predictors that decreases negative log-likelihood by the most at each step
- Then, given the big tree, "prune" some of it back
- Amount pruned back determined by information criteria
- Alternatively, use cross-validation:
  - Grow a full tree then prune back to L leaves
  - Select L that minimises the cross-validation error
- Only approximate search, but often better than forward search

Decision Trees – Strengths

- The decision tree approach has many strengths:
  - Highly interpretable
  - Handles non-linear relationships
  - Handles variable interactions
    - If we split on one variable, then another, they are interacting
  - Seamlessly handles continuous, categorical and count exposure variables
  - Easily generalises to multiclass (categorical) outcome variables
  - Variable selection naturally incorporated

Classification Example – Nonlinearities

Decision tree separating line



Logistic regression separating line

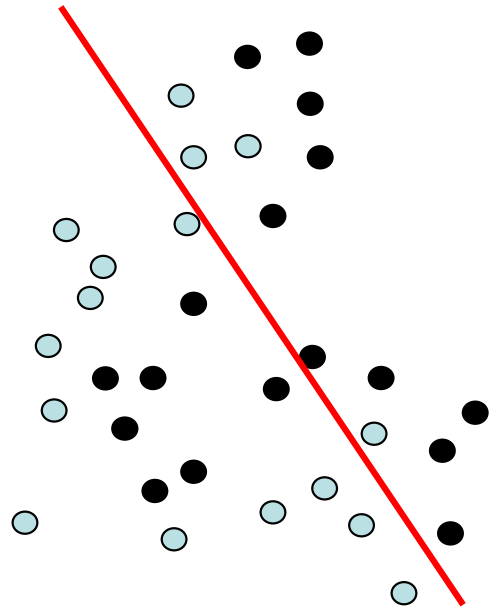


Figure:

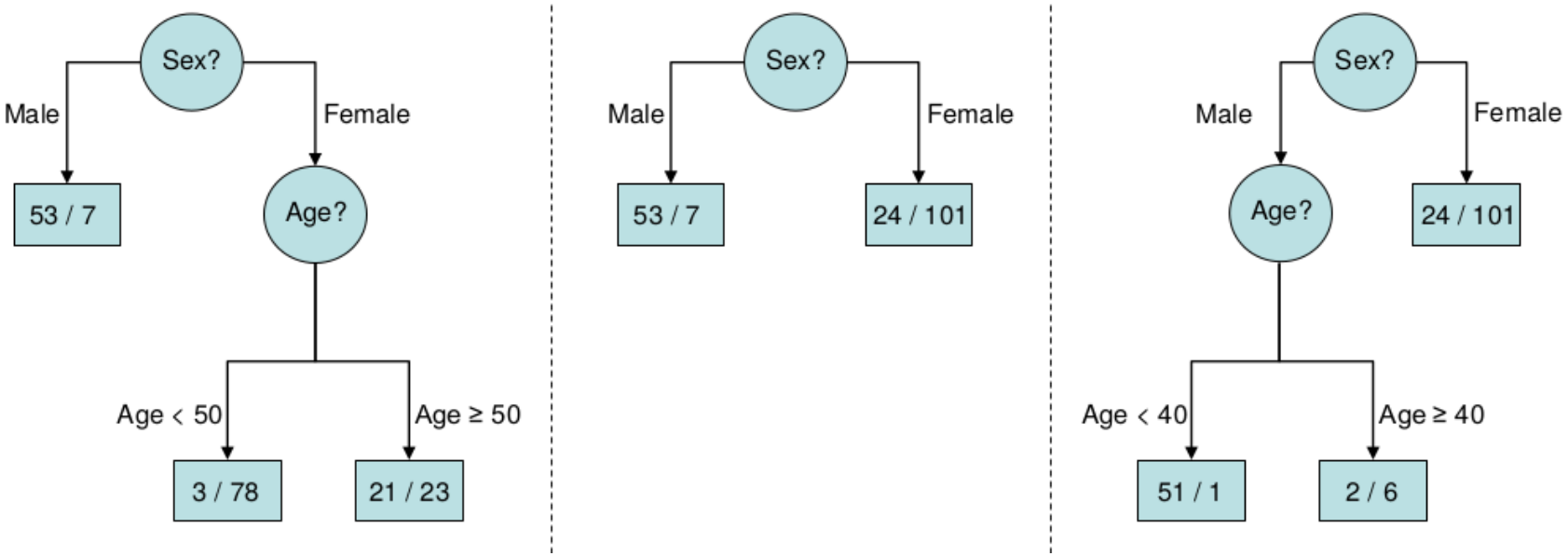
Decision Trees – Weaknesses

- The decision tree approach also has weaknesses:
  - Sometimes difficult to find a good tree
  - Sensitive to small perturbations in the data
    - A slight change can result in a drastically different tree
  - Potentially inefficient
    - If the true model is well explained by a linear model, then tree needs many leaves, and many parameters to fit data well
- To overcome the first two we will now look at random forests

Outline

Instability of Decision Trees

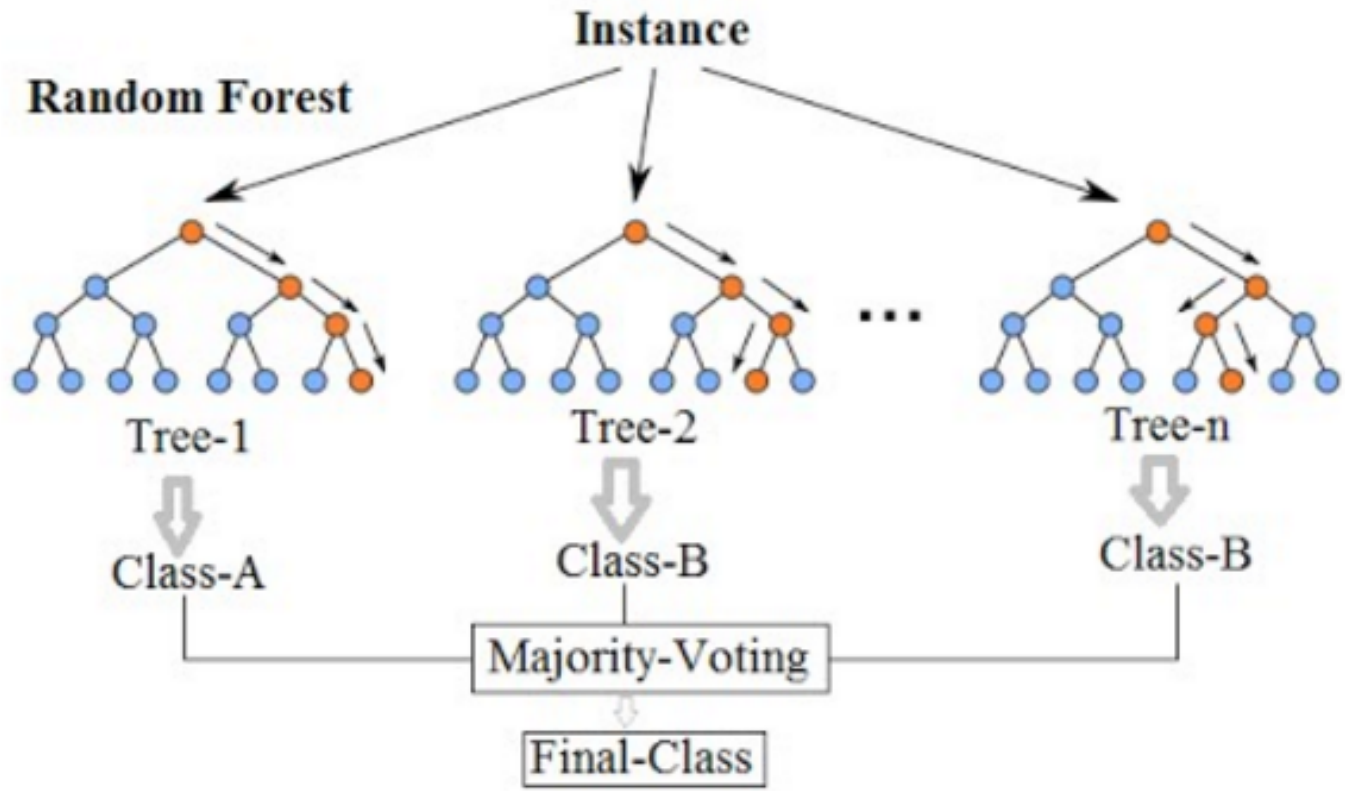
- Decision trees are flexible but unstable



Three different trees fitted to the same data. They all have similar goodness-of-fit scores – difficult to decide which is best

Using Multiple Trees

Why not do prediction using multiple trees?



(by Venkata Jagannath [CC BY-SA 4.0], via Wikimedia)

Random Forests

- A random forest is a collection of q trees
  - This approach is called *ensemble learning*
- How to create the forest?
- For each of the q trees in the forest, build by a random forward search:
  - Grown by controlled, but random, splitting
  - Candidate variables to split on at each step of search is random subset
  - Splits are selected based on cost function (such as entropy)
- Helps overcome "greedy" nature of forward search

Using Random Forests

- To make a prediction given a forest:
  - Compute predictions for each of the q trees
  - Combine predictions into a single prediction
    - Average predicted means (for regression) or predicted probabilities (for classification)
- How does this help?
  - Individual trees have low bias, but high variance (unstable)
  - By combining many trees we retain low bias but reduce variance
- The importance of a variable can be measured by how many times in how many trees it was selected
- Endless variations of rules for splitting, searching, etc.

On Random Forests

- Strengths of random forests:
  - Very stable
    - Resistant to perturbations in data
    - Resistant to “local minima” in search space
  - Improved predictive accuracy in general
  - Easily handle multiclass or count targets
  - Flexible, handle non-linearities just like regular trees
  - Inherently incorporate variable/predictor selection
- Weaknesses of random forests:
  - Poor interpretability
    - Can get an idea of variable importance, but that is about it
  - Complex to learn, lots of variations and parameters to tweak

Outline

k-Nearest Neighbours

- Sometimes called “nearest neighbours smoothing”, or k-NNs
- Does not produce a model
  - We have a set of n example predictor/target pairs
    - Predictor values $x_{i,1}, \ldots, x_{i,p}$ paired with target y_i
  - We want to predict target value for new individual with predictor values $x'_1, \ldots, x'_p$
- Find k individuals in our data “most similar” to the new individual
  - Use target values of these k individuals to predict target for our new individual
- Very weak assumptions
  - Individuals similar to each in other in terms of predictor values will be similar in terms of targets

k-NN Example $k = 3$

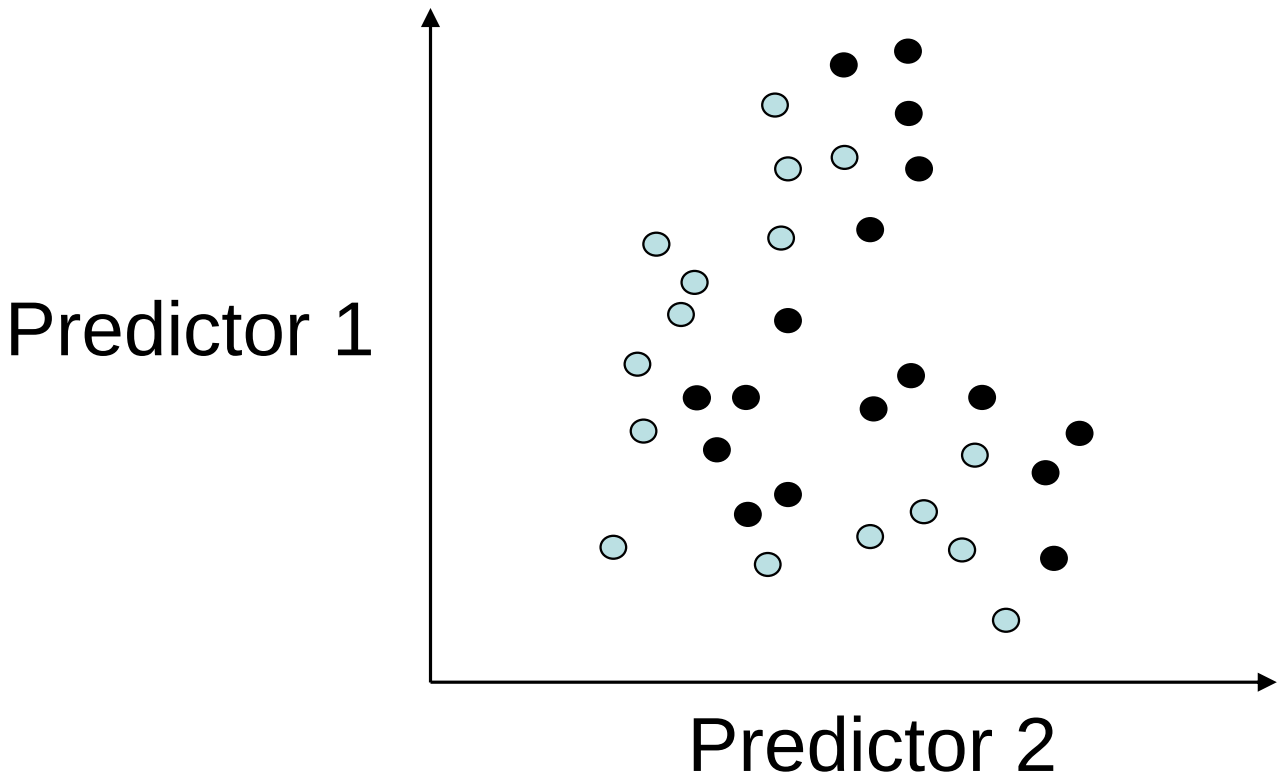


Figure: Example data set: black are individuals with disease, blue are those without

k-NN Example $k = 3$

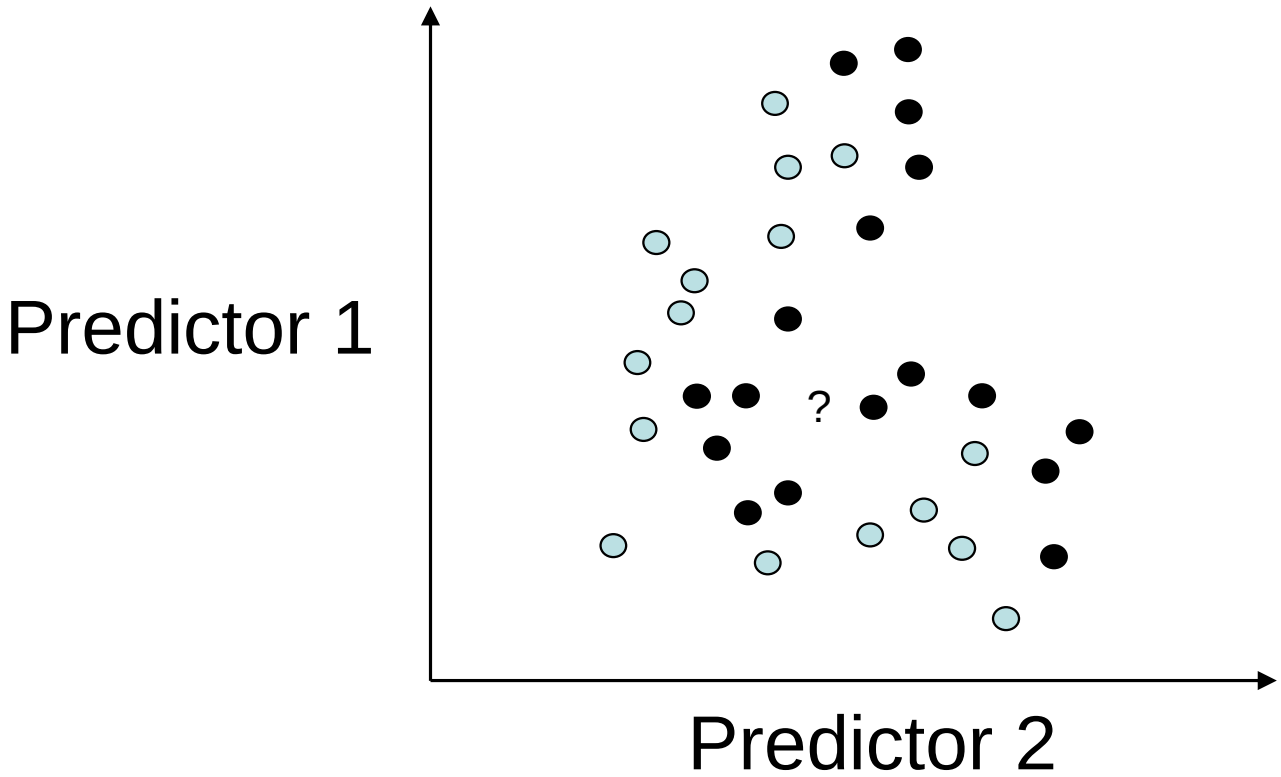
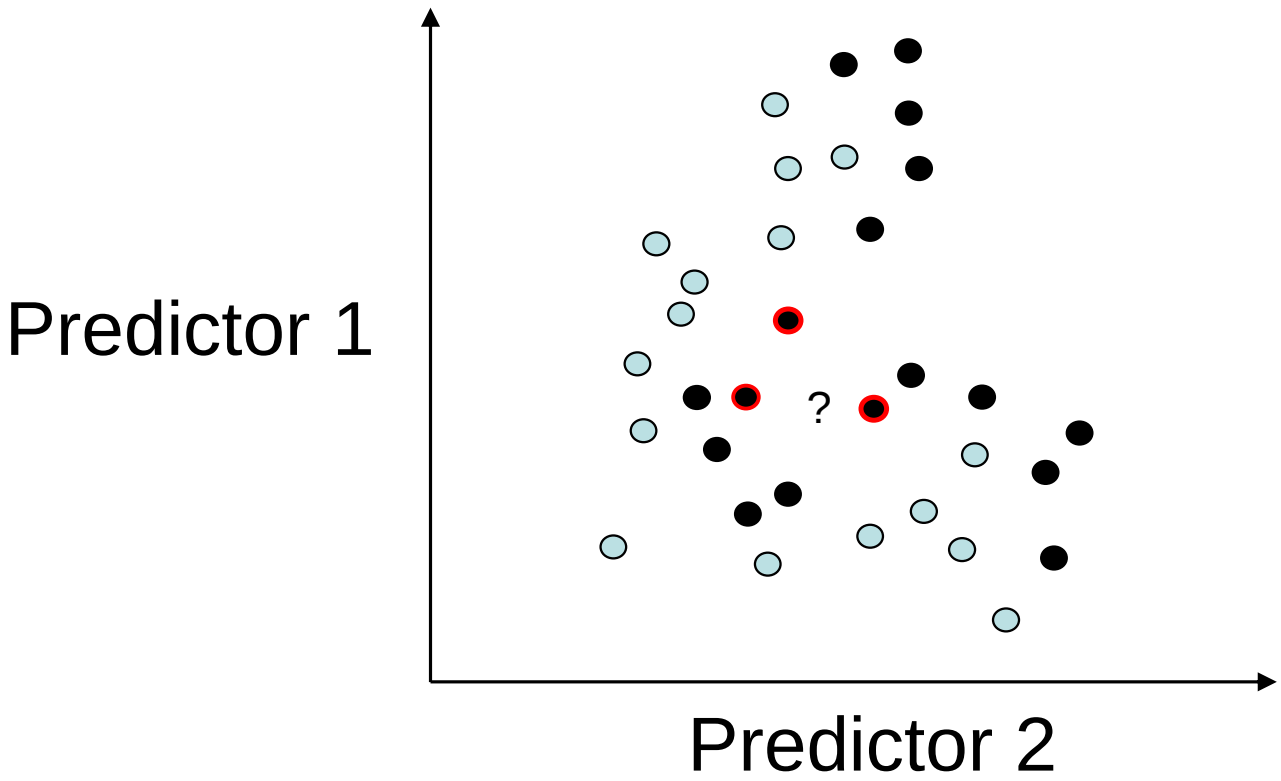


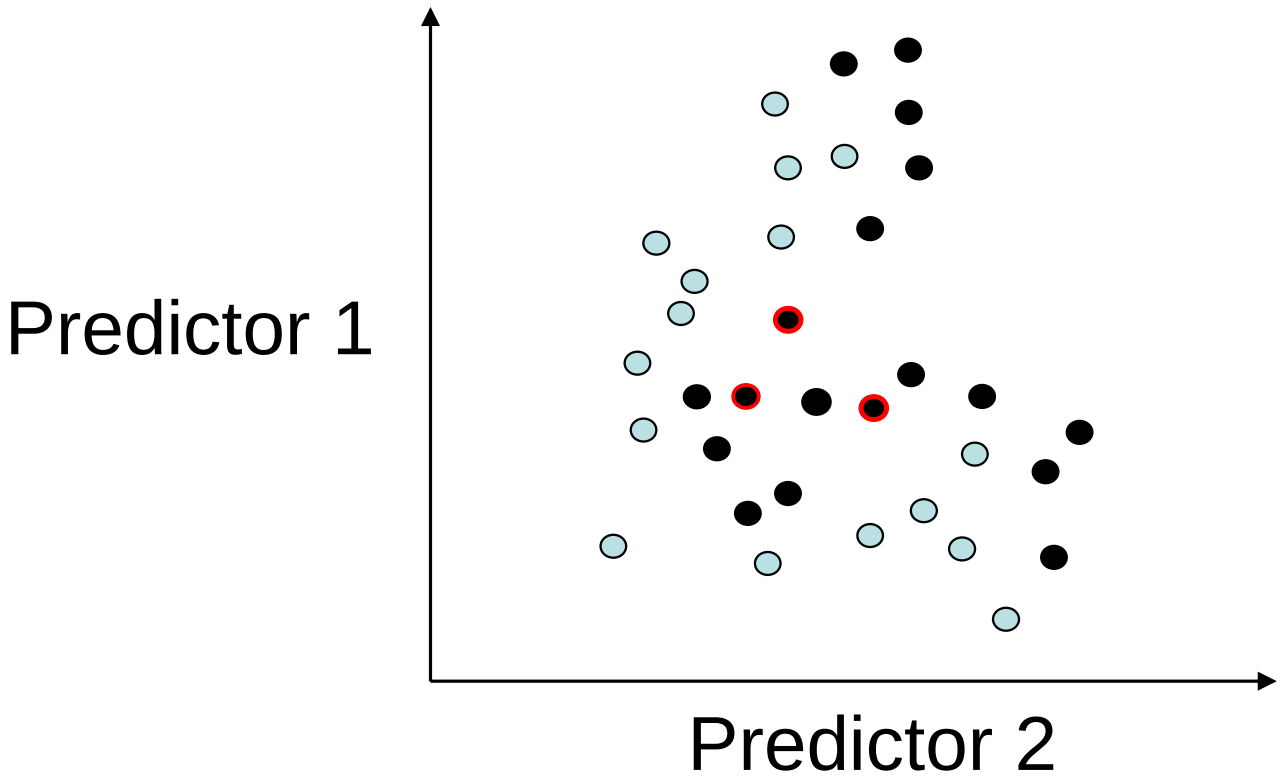
Figure: We want to predict the disease status of the individual marked with a "?" using a k-NN method with $k = 3$ neighbours

k-NN Example $k = 3$



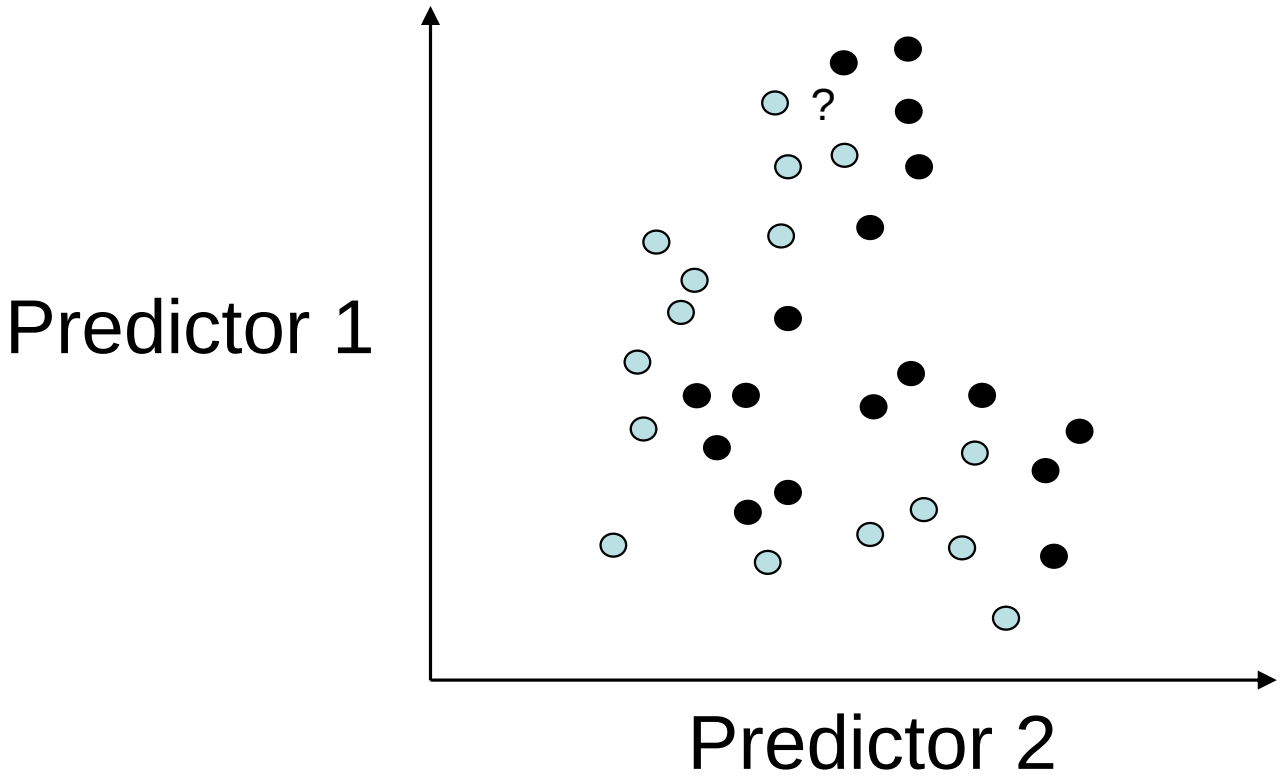
We find the closest $k = 3$ individuals

k-NN Example $k = 3$



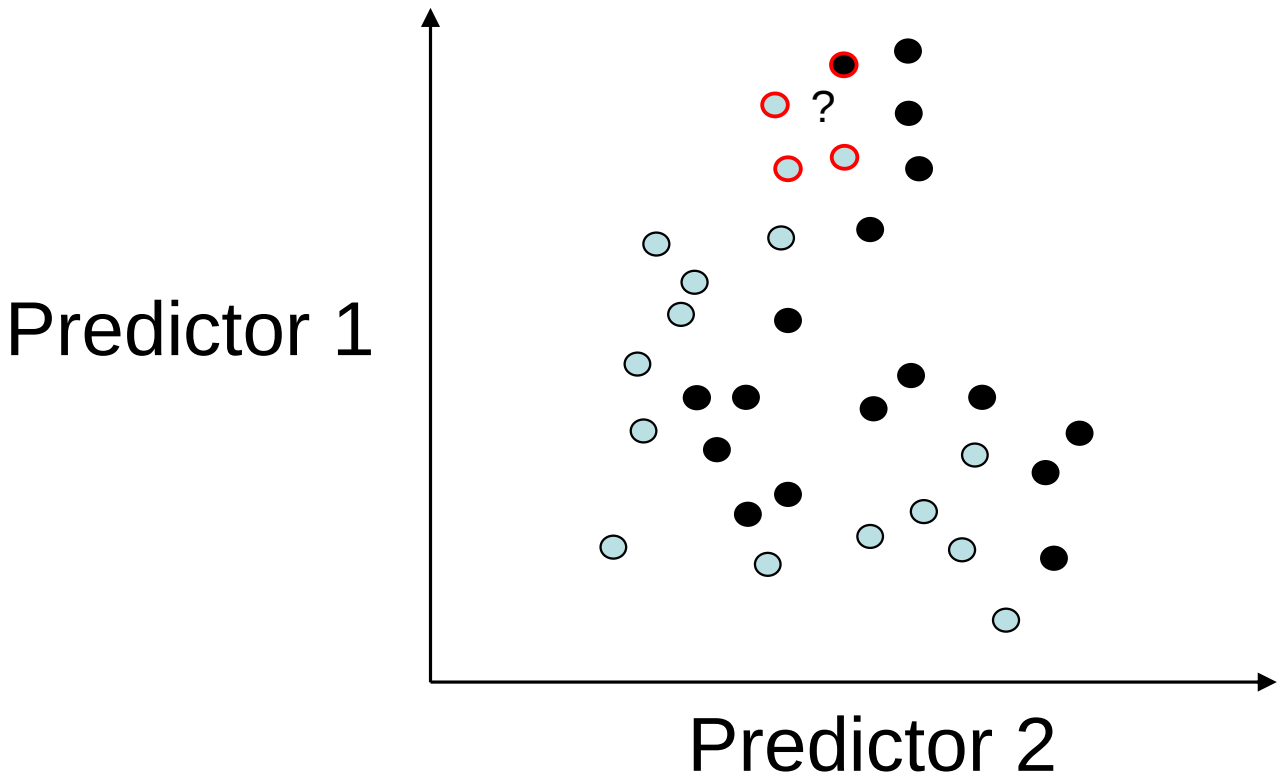
They all have the disease, so we predict that our new individual will also have the disease

k-NN Example $k = 4$



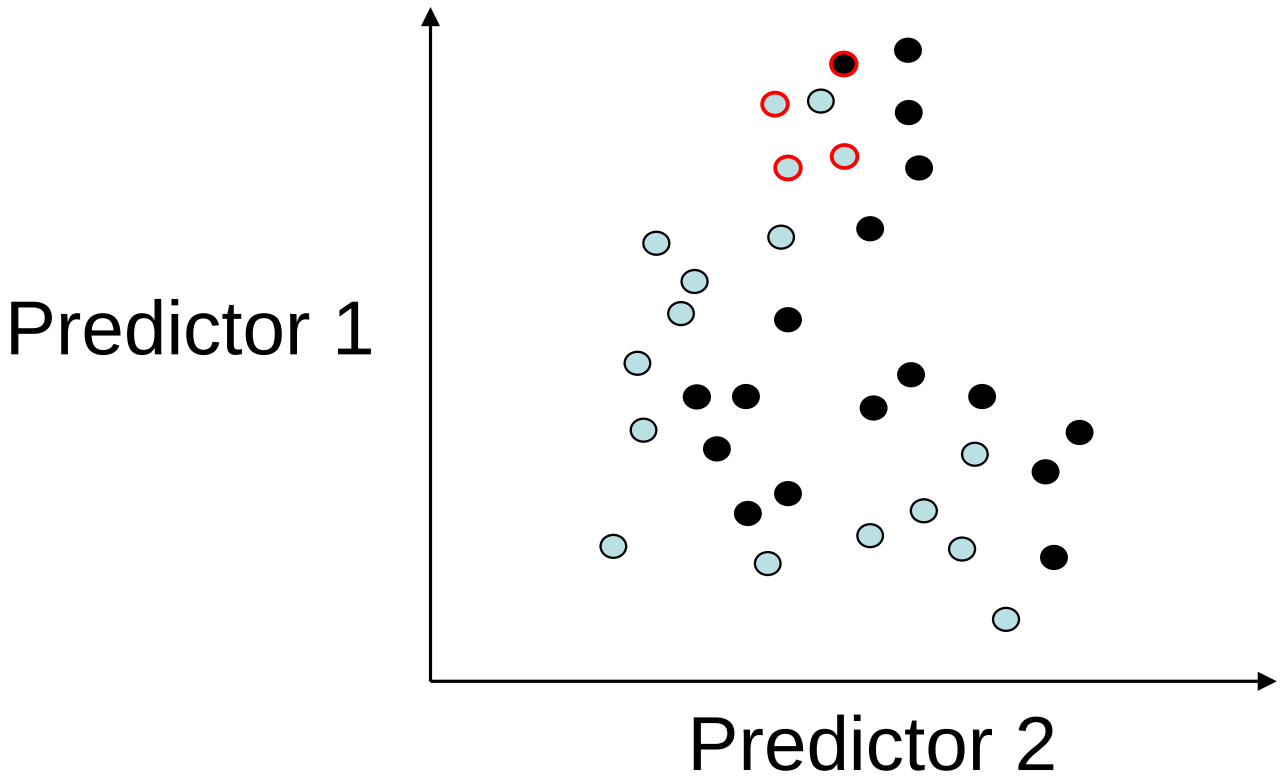
We want to predict the disease status of the individual marked with a "?" using a k-NN method with $k = 4$ neighbours

k-NN $k = 4$



We find the 4 closest individuals

k-NN $k = 4$



This time three of the four do not have the disease while only one has the disease, so we predict that our new individual will not have the disease

Neighbour Distance

- How to measure "similarity" of individuals
- The most common method is the Euclidean distance between the two in predictor space:

$$d(\mathbf{x}, \mathbf{x}') = \left[\sum_{j=1}^{p} (x_j - x'_j)^2 \right]^{\frac{1}{2}}$$

- This measure may not always be appropriate/best
  - Can use different measures for categorical predictors
- Different distance measures produce different k-nearest neighbour predictions

k-NN Algorithm

- Let d_i be the distance of our new individual from individual i in our training data
- k-nearest neighbours algorithm:
  1. Sort individuals from smallest d_i to largest
  2. Denote the targets in the sorted list by $y^{(1)}, \ldots, y^{(n)}$
     - $y^{(1)}$ is observed target for closest individual
     - $y^{(n)}$ is observed target for furthest individual
  3. Use the k individuals with smallest d_i to predict y'

$$\hat{y}' = f(y^{(1)}, \ldots, y^{(k)})$$

- How to select the prediction function $f(\cdot)$?

Using k-NN

- For classification problems, we can using voting
  - Choose class most frequent in the k nearest neighbours
- For regression, we can using averaging

$$\hat{y}' = \frac{1}{k} \sum_{i=1}^{k} y^{(i)}$$

- Variations include using weighted averaging

$$\hat{y}' = \frac{1}{k} \sum_{i=1}^{k} g(y^{(i)})$$

where $g(d)$ is a function that gets smaller as d gets larger
$\Longrightarrow$ further observations contribute less than close ones

- The functions $g(\cdot)$ are often called kernel functions

On k-NNs

- k-NN methods have many "tunable" parameters/options:
  - Neighbourhood size k
  - Distance functions
  - Kernel weighting functions
- How to choose these?

- Use leave-one-out cross-validation
- For each combination of parameter choices:
  1. Predict each example y_i using k-NN
  2. Accumulate error in predicting y_i
- Choose the parameters that minimise this score

Strengths and Weaknesses

- Strengths of k-nearest neighbours methods:
  - Very weak assumptions about data
  - Efficient learning in high dimensions
  - Easily handle continuous and categorical variables
  - Instance-based intepretation
- Weaknesses
  - Lots of parameters to configure
    - How many neighbours (k) to use?
    - How to measure “nearest”?
    - How to average or weight the neighbours
    - Should all predictors be treated the same?
  - Predictor selection is not straightforward
  - No general intepretation

Reading/Terms to Revise

Terms you should know:

- Cross-validation
- Decision tree
- Split and leaf
- Random forest
- k-nearest neighbours method

End of Week 11

Revision at *https://flux.qa/43FMK4*