FIT2086 Lecture 9
Trees and Nearest Neighbour Methods

Daniel F. Schmidt

Faculty of Information Technology, Monash University

September 22, 2019

Outline

Revision from last week (1)

- How many predictors should we include in our linear model?
- Underfitting
  - Omitting important predictors
  - Leads to systematic error ("bias") in predicting the target
- Overfitting
  - Including spurious predictors
  - Leads our model to "learn" noise and random variation
- Methods to trade off bias and variance
  - Hypothesis testing $\beta_j = 0$ vs $\beta_j \neq 0$
    - Multiple hypothesis testing problem, Bonferroni
  - Penalized likelihood – likelihood plus complexity penalty
    - AIC, KIC, BIC, RIC
  - Cross-validation

Revision from last week (2)

- Statistical instability;
  - Small changes in data $\Rightarrow$ big changes in model
- Ridge regression, squared penalty on coefficients

$$\left(\hat{\beta}_0, \hat{\boldsymbol{\beta}}_\lambda\right) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \left\{ \mathrm{RSS}(\beta_0, \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

- Lasso regression, absolute penalty on coefficients

$$\left(\hat{\beta}_0, \hat{\boldsymbol{\beta}}_\lambda\right) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \left\{ \mathrm{RSS}(\beta_0, \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

- Lasso can estimate coefficients to be zero, ridge cannot
- Vary λ to get a “path” of different complexity models
  - $\lambda = 0$ most complex, $\lambda = \infty$ least complex
- Use cross validation to choose a good λ

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)
- If the variable we are predicting is categorical, we are performing classification
- If the variable we are predicting is numerical, we are performing regression

Cross validation – Revisited (1)

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

Cross validation – Revisited (2)

- To estimate MSPE for a given γ 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 $\mathcal{M}(\gamma)$ to the training data $\mathbf{y}_{\text{train}}$
  3. Compute the MSPE of this model on the testing data $\mathbf{y}_{\text{test}}$
  4. Repeat this procedure m and average the MSPE
- This gives us an estimate of how well our model would predict future data from the same population
  $\Rightarrow$ the larger the m, the more stable the estimate is
- Do this for all our different models, and choose the one that has the smallest CV error

K-fold Cross Validation

- Outer loop: try different model complexities γ
  1. For $i = 1$ to 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
        1. Fit model $\mathcal{M}(\gamma)$ to all $\mathbf{y}^{(j)}$ except for $j = k$
        2. Use fitted model to predict onto $\mathbf{y}^{(k)}$
        3. Calculate and accumulate prediction errors
  2. Average all m accumulated prediction errors
- Once this is done, we have CV errors for each complexity γ
  $\Rightarrow$ choose the γ with the smallest estimated error
- The larger the m, the more stable the estimates (but slower)

Machine Learning (1)

- 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
- Often impossible to interpret the resulting “model”
    - Usually focussed on prediction

Machine Learning (2)

- 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 (3)

- Decision trees
  - R packages: `tree`, `rpart`
- Artificial neural networks
  - R packages: `nnet`, `neuralnet`, `brnn`
- Support vector machines
  - R packages: `e1071`, `pathClass`, `penalizedSVM`
- Clustering/mixture modelling
  - R packages: `cluster`, `mixtools`, `rebmix`

Decision Trees (1)

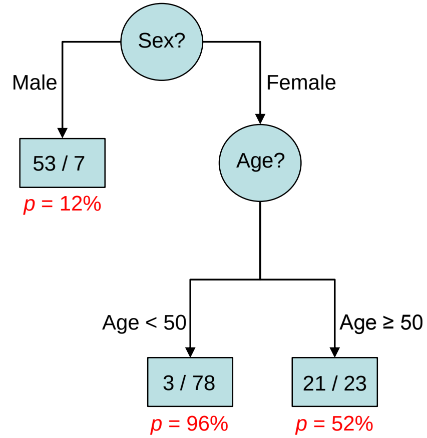
- A decision tree is a type of supervised learning model
- It takes inputs $x_1, \ldots, x_p$ and produces a prediction

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

- 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
  $\Rightarrow$ but still retain a high degree of interpretability
- Can be applied to regression and (multi-class) classification

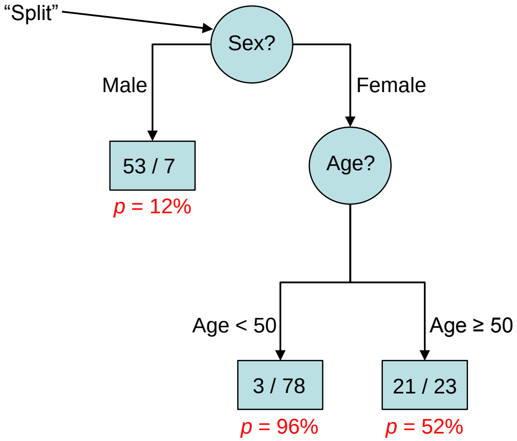
Decision Trees (1)

- Example tree: predicting high blood pressure



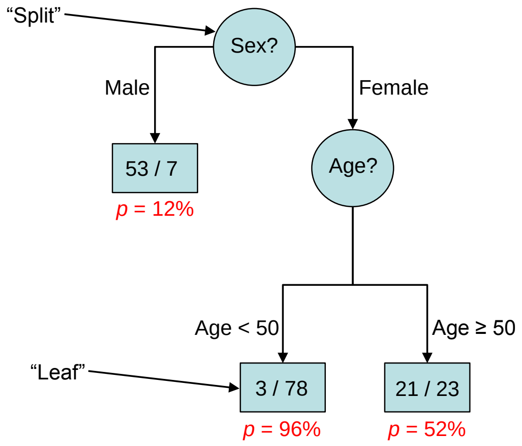
Decision Trees (2)

- Example tree: predicting high blood pressure



Decision Trees (3)

- Example tree: predicting high blood pressure



Decision Trees (4)

- 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)
  $\Rightarrow$ the more leaves, the better the tree can fit the data

Learning Decision Trees (1)

- 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
  - Information criteria: AIC, BIC, etc.
  - Cross-validation scores
- Find the tree with the smallest score
  - Trade-off goodness-of-fit against tree complexity

Learning Decision Trees (2)

- The space of possible trees is enormous if p is even moderate
- Instead, we use approximate search algorithms
- Most basic: forward search with information 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? (1)

- 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?
  $\Rightarrow$ 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 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

What Makes a Good Split? (2)

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

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

- Recall that the maximum likelihood estimator $\hat{\theta} = n_1/n$
- The maximised likelihood is

$$p(\mathbf{y} \,|\, \hat{\theta}) = \left(\frac{n_1}{n}\right)^{n_1} \left(\frac{n_0}{n}\right)^{n_0}$$

- The minimised negative log-likelihood of the data is then

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

- This is:
  - largest when $n_1/n = 1/2$ (least "pure");
  - smallest when $n_1 = 0$ or $n_1 = n$ (most "pure")
- The "purer" the data, the smaller the negative-log likelihood

What Makes a Good Split? (2)

- 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 maximised likelihood is

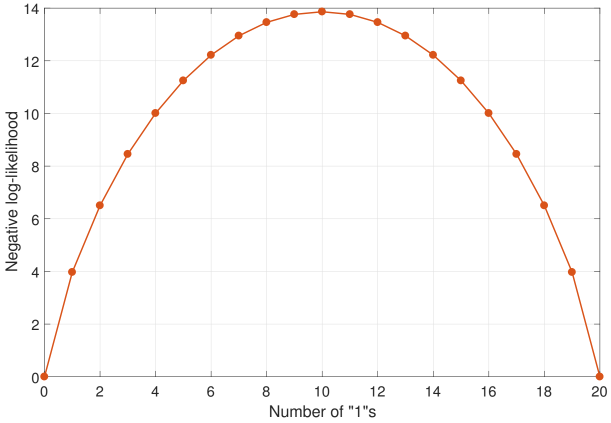
$$p(\mathbf{y} \mid \hat{\theta}) = \left(\frac{n_1}{n}\right)^{n_1} \left(\frac{n_0}{n}\right)^{n_0}$$

- The minimised negative log-likelihood of the data is then

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

- This is:
    - largest when $n_1/n = 1/2$ (least "pure");
    - smallest when $n_1 = 0$ or $n_1 = n$ (most "pure")
- The "purer" the data, the smaller the negative-log likelihood

What Makes a Good Split? (3)



Minimised negative log-likelihood of the Bernoulli distribution 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").

What Makes a Good Split? (4)

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

- Toy example
  - $n = 10$ data points, binary targets
  - Two binary predictors, x_1 and x_2
- Without splitting, we have a single leaf (a "root")
- We have $\hat{\theta} = 5/10$, so

$$-\log p(\mathbf{y} \,|\, \hat{\theta}) = -5 \log(5/10) - 5 \log(5/10) \approx 6.9315$$

- 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 negative log-likelihood

What Makes a Good Split? (5)

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
  - Take y when $x_1 = 0 \Rightarrow (1, 1, 0, 1, 0)$, $\hat{\theta}_{x_1=0} = 3/5$
  - Take y when $x_1 = 1 \Rightarrow (0, 0, 1, 1, 0)$, $\hat{\theta}_{x_1=1} = 2/5$
- Negative-log-likelihood of the two leaves is then:

$$-\log p(\mathbf{y}\,|\,\hat{\theta}, x_1 = 0) = -3\log(3/5) - 2\log(2/5) \approx 3.3651$$
$$-\log p(\mathbf{y}\,|\,\hat{\theta}, x_1 = 1) = -2\log(2/5) - 3\log(3/5) \approx 3.3651$$

- So the total negative-log likelihood is approx. 6.7302
  $\Rightarrow$ a bit better than "no split" neg-log-like of 6.9315

What Makes a Good Split? (6)

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
  - Take y when $x_2 = 0 \Rightarrow (0, 0, 0, 1, 0)$, $\hat{\theta}_{x_2=0} = 1/5$
  - Take y when $x_2 = 1 \Rightarrow (1, 1, 1, 0, 1)$, $\hat{\theta}_{x_2=1} = 4/5$
- Negative-log-likelihood of the two leaves is then:

$$\begin{aligned} -\log p(\mathbf{y} \,|\, \hat{\theta}, x_2 = 0) &= -1\log(1/5) - 4\log(4/5) \approx 2.5020 \\ -\log p(\mathbf{y} \,|\, \hat{\theta}, x_2 = 1) &= -4\log(4/5) - 1\log(1/5) \approx 2.5020 \end{aligned}$$

- So the total negative-log likelihood is approx. 5.04
  $\Rightarrow$ a lot better than "no split" neg-log-like of 6.9315
- So splitting on x_2 leads to purer leaf nodes

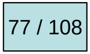
What Makes a Good Split? (7)

- 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 negative log-likelihood
- Remember, splitting data will always decrease negative-log-likelihood
- Hence, likelihood can be used to guide splits
- But to select a tree, we need to penalize the tree by complexity
- Either use information criteria, or cross-validation approaches

Learning Decision Trees - Example (1)

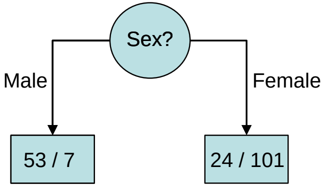
- Example: start with a single leaf node (a “root”)



Score: 175.13

Learning Decision Trees - Example (2)

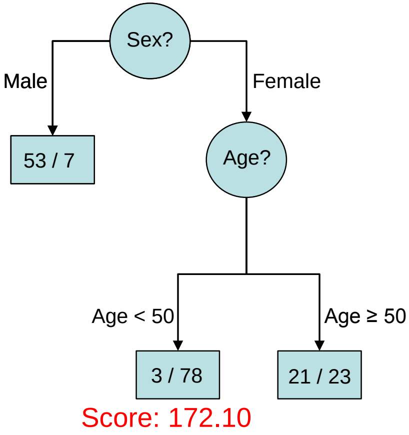
- Split on Sex – score improves



Score: 173.70

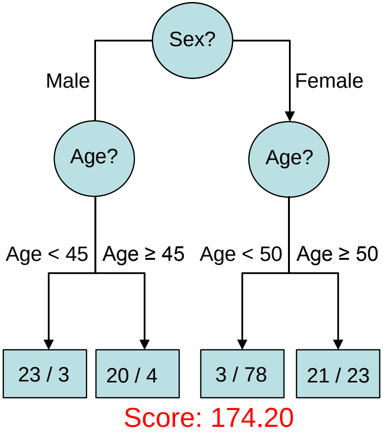
Learning Decision Trees - Example (3)

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



Learning Decision Trees - Example (4)

- Split on Age when Sex is “Male” – score does not improve



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

K-fold Cross Validation for Trees

- Set the number of leaves L as complexity parameter
- Input: $L_{\max}$, maximum number of leaves to consider
  1. For $i = 1$ to 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
        1. Grow a tree with $\gg L_{\max}$ leaves using all $\mathbf{y}^{(j)}$ except for $j = k$
        2. Prune this tree back to $L = 1, \ldots, L_{\max}$ leaves
        3. Use the $L_{\max}$ fitted models to predict onto $\mathbf{y}^{(k)}$
        4. Calculate and accumulate prediction errors
  2. Average all m accumulated prediction errors
- Choose L that has smallest accumulated error
  - Grow tree with L leaves on all data as final model

Decision Trees – Strengths

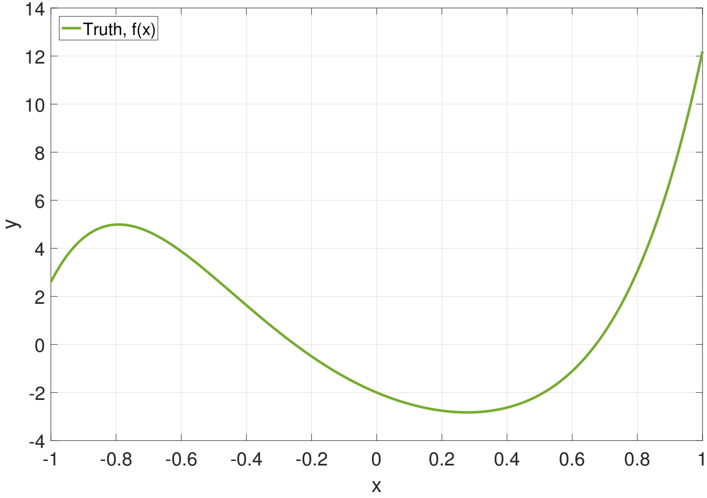
- The decision tree approach has many strengths:
  - Highly interpretable
  - Handles non-linear relationships
  - Handles variable interactions
    - If we split on one variable, then another, they are interacting
  - Seamlessly handles continuous, categorical and count predictor variables
  - Easily generalises to multiclass (categorical) outcome variables
  - Variable selection naturally incorporated
- To see how decision trees handle non-linearities, easiest to look at regression
  - Each leaf is a normal model with its own mean and variance

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 predictor variables
  - Easily generalises to multiclass (categorical) outcome variables
  - Variable selection naturally incorporated
- To see how decision trees handle non-linearities, easiest to look at regression
  - Each leaf is a normal model with its own mean and variance

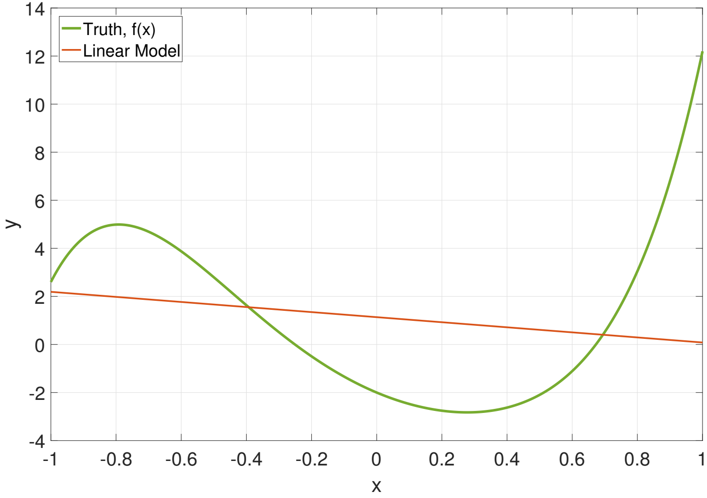
Regression Tree Example (1)

- $y = 9.7x^5 + 0.8x^3 + 9.4x^2 - 5.7x - 2$



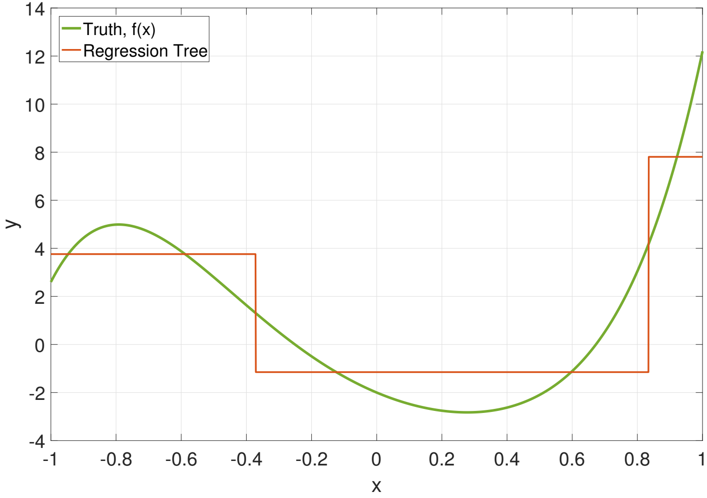
Regression Tree Example (2)

- Linear regression, $y = 1.1365 - 1.0541x$



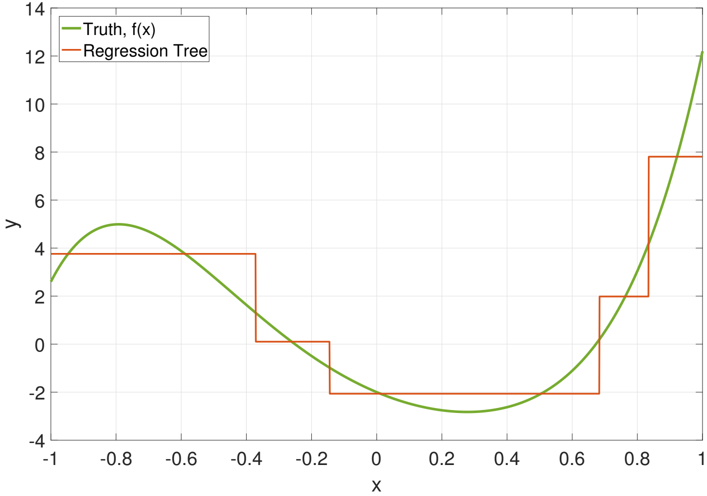
Regression Tree Example (3)

- Regression tree with $t = 3$ leaf nodes



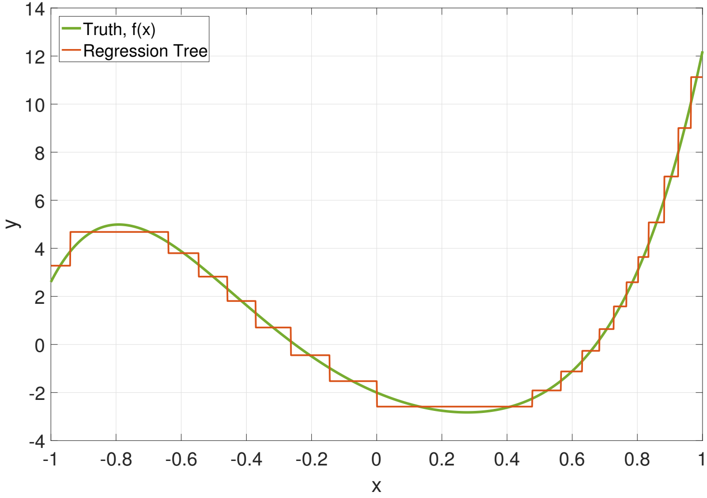
Regression Tree Example (4)

- Regression tree with $t = 5$ leaf nodes



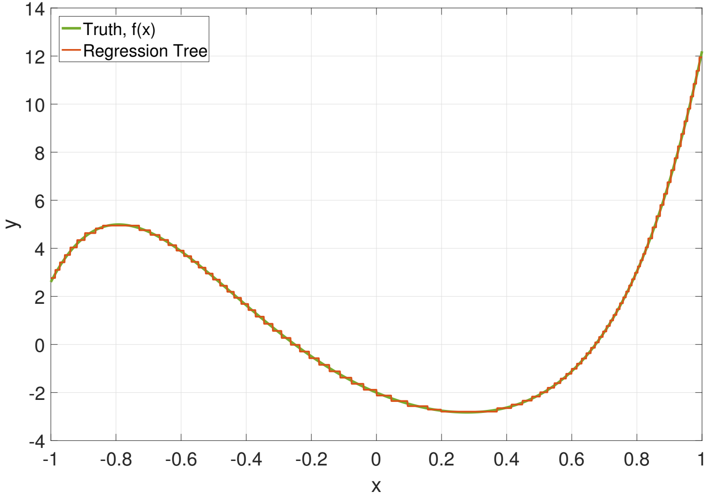
Regression Tree Example (5)

- Regression tree with $t = 20$ leaf nodes



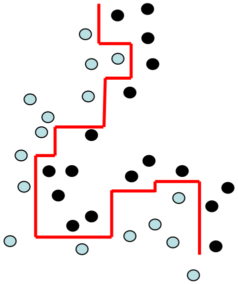
Regression Tree Example (6)

- Regression tree with $t = 88$ leaf nodes

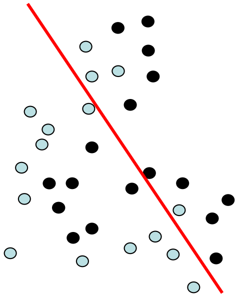


Classification Example – Nonlinearities

Decision tree separating line



Logistic regression separating line

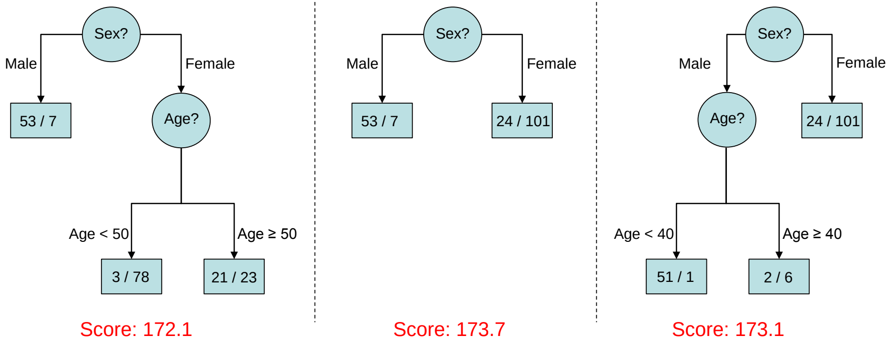


Decision Trees – Weaknesses

- 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

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

Random Forests (1)

- 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 likelihood)
- Helps overcome "greedy" nature of forward search

Random Forests (2)

- 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.

Random Forests (3)

- 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 (1)

- Sometimes called “nearest neighbours smoothing”, or kNNs
- 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-Nearest Neighbours (2)

- 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-Nearest Neighbours (3)

- 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 $y^{(1)}, \dots, y^{(n)}$
     - $y^{(1)}$ 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'

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

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

k-Nearest Neighbours (4)

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

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

- Variations include using weighted averaging

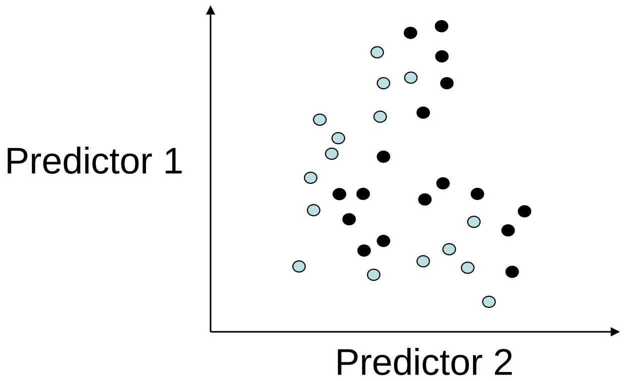
$$\hat{y}' = \frac{1}{k} \sum_{i=1}^{k} g(y^{(i)})$$

  where $g(d)$ is a function that gets smaller as d gets larger
  $\Rightarrow$ further observations contribute less than close ones
- The functions $g(\cdot)$ are often called kernel functions

k-Nearest Neighbours (5)

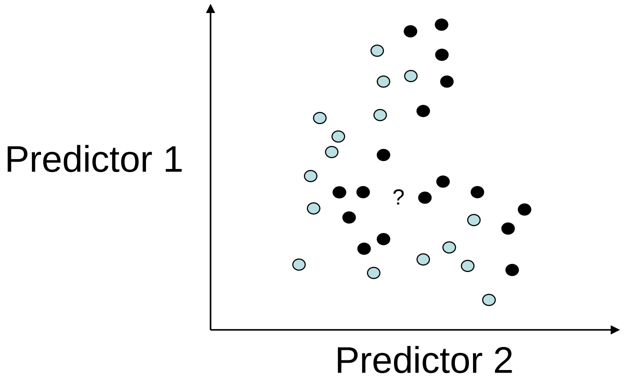
- k-NN methods have many "tunable" parameters/options:
  - Neighbourhood size k
  - Distance functions
  - Kernel weighting functions
- How to choose these?
- Use leave-one-out cross-validation
- For each combination of parameter choices:
  1. Predict each example y_i using k-NN (with y_i removed from the training set)
  2. Accumulate error in predicting y_i
- Choose the parameters that minimise this score

k-Nearest Neighbours Example 1 (1)



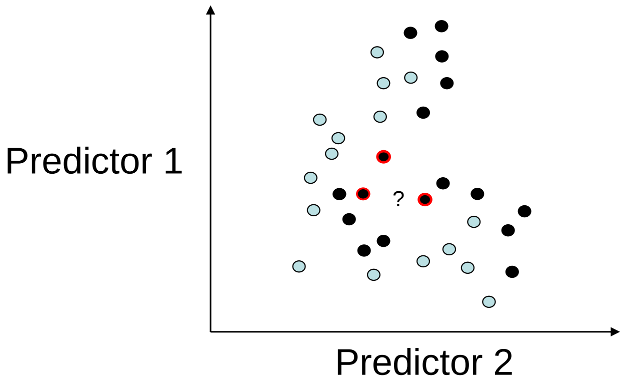
Example data set: black are individuals with disease, blue are those without

k-Nearest Neighbours Example 1 (2)



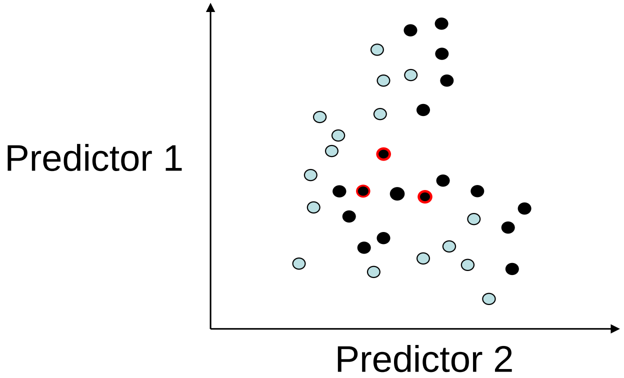
We want to predict the disease status of the individual marked with a "?" using a k nearest neighbour method with $k = 3$ neighbours

k-Nearest Neighbours Example 1 (3)



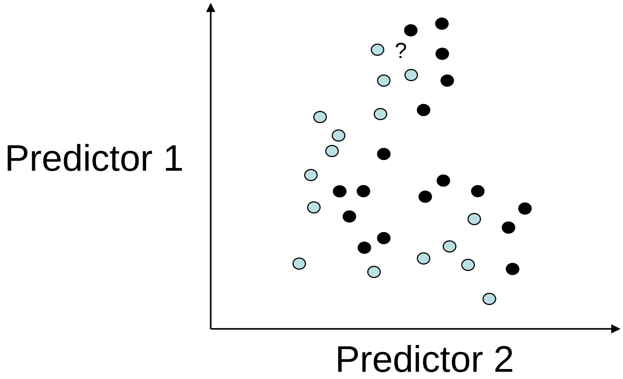
We find the closest $k = 3$ individuals

k-Nearest Neighbours Example 1 (4)



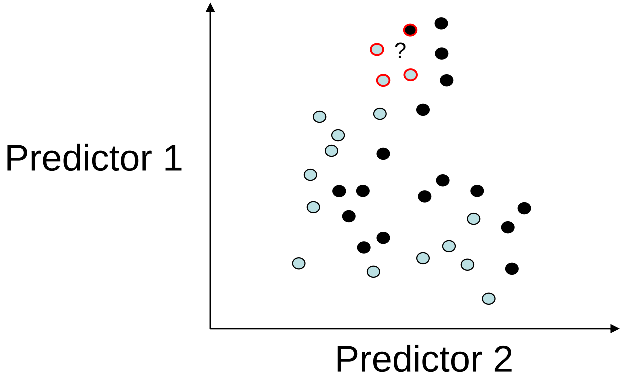
They all have the disease, so we predict that our new individual will also have the disease

k-Nearest Neighbours Example 2 (1)



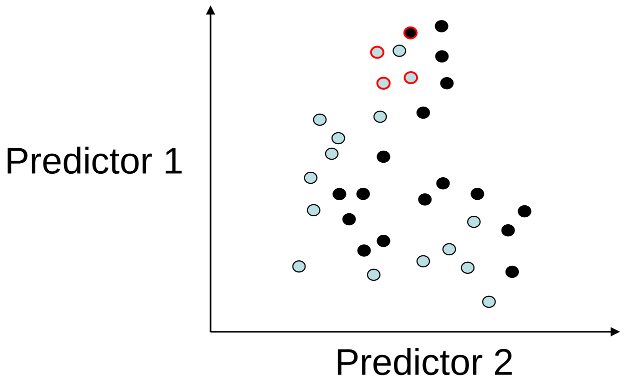
We want to predict the disease status of the individual marked with a "?" using a k-nearest neighbour method with $k = 4$ neighbours

k-Nearest Neighbours Example 2 (2)



We find the 4 closest individuals

k-Nearest Neighbours Example 2 (3)



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

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
- 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 interpretability

Reading/Terms to Revise

- Terms you should know:
  - Cross-validation
  - Decision tree
  - Split and leaf
  - Random forest
  - k-nearest neighbours method
- Next week: machine learning algorithms for unsupervised data discovery (clustering, mixture modelling, matrix completion)