CQS Summer Institute: Machine Learning and Statistics in R

Matthew S. Shotwell, Ph.D.

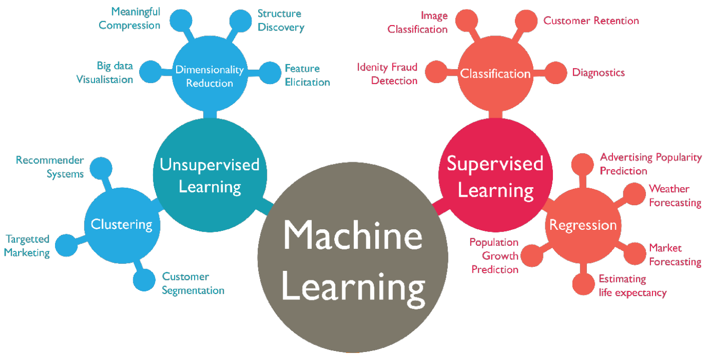
Department of Biostatistics
Vanderbilt University Medical Center
Nashville, TN, USA

August 14, 2018

Course Overview

- Syllabus and R code:
- `https://github.com/biostatmatt/cqs-ml-stat-r`
- Monday: Intro and Data Management
- Tuesday: Supervised Learning Part 1
- Wednesday: Supervised Learning Part 2
- Thursday: Unsupervised Learning
- Friday: Statistical Inference

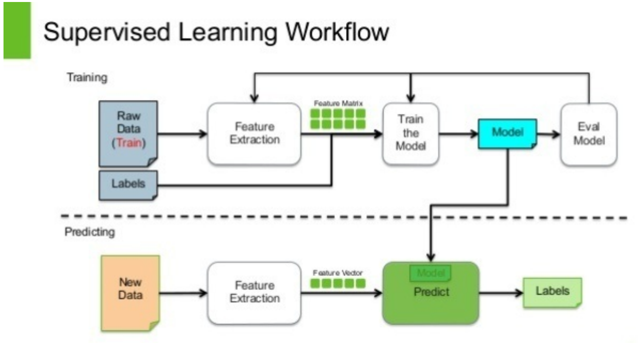
Machine learning



source: https://www.wordstream.com/blog/ws/2017/07/28/machine-learning-applications

Supervised learning

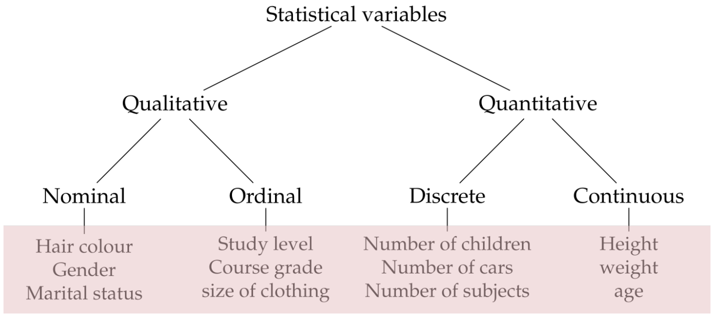
- Have input ('features') AND output ('target')
- Create a model ('learner') using observed inputs and outputs
- Goal is to predict outputs from new inputs
- "Supervised" because both inputs *and outputs* to guide model



source: https://www.quora.com/What-is-pattern-recognition

Definitions: variable types

- quantitative - e.g. blood pressure
- qualitative - e.g. gender, a.k.a. categorical, discrete, factor, numeric codes for qualitative variables called ‘targets’
- ordered - e.g. numerical pain scale (0-10)



source: http://aprendeconalf.es/statistics/manual/introduction.html

Definitions: supervised learning tasks

- regression - model to predict quantitative output
- classification - model to predict qualitative output
- regression or classification - ordered output

Definitions: notation

- inputs - X

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}$$

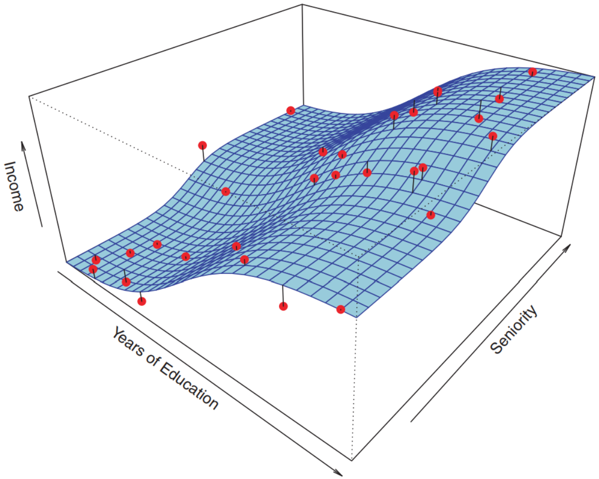
- quantitative outputs - Y

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- qualitative outputs - G
- transpose - X^T
- prediction - $\hat{Y}$

Regression

- Find function f to predict Y: $\hat{Y} = \hat{f}(X)$



source: ISLR Fig. 2.3

Regression

- To find f in $\hat{Y} = \hat{f}(X)$, need an **objective function**:
- Least squared error: minimize $1/n \sum_{i=1}^{n} (y_i - f(x_i))^2$
- Least absolute error: minimize $1/n \sum_{i=1}^{n} |y_i - f(x_i)|$
- and need to select a class of models $f(X)$:
- linear models:
$$f(X) = X\beta$$
- k-nearest neighbor:
$$f(X) = \frac{1}{k} \sum_{x_i \in N_k(X)} y_i$$

Linear regression (LS)

- Given $n \times p$ matrix X, predict Y ($n \times 1$ matrix) as follows:

  $$\hat{Y} = \hat{f}(X) = X\hat{\beta}$$

  where $\hat{\beta}$ is the value that minimizes the sum of squared error in the training data residual mean of squared errors:

  $$1/n \sum_{i=1}^{n} (y_i - x_i\beta)^2$$

- The R function 'lm' will estimate β in this way

k-nearest neighbor regression (kNN)

- Predict $\hat{Y}$ corresponding to X by averaging the Y values of the k nearest neighbors to X:

$$\hat{Y}(X) = \frac{1}{k} \sum_{x_i \in N_k(X)} y_i$$

- $N_k(X)$ is set of k points nearest X, as determined by a distance metric, e.g., the Euclidean distance:

$$d(X, X') = \sqrt{(X - X')^T (X - X')}$$

- k parameters is a 'smoothing parameter' or 'tuning parameter'
- 'caret::knnreg' implements kNN regression with Euclidean dist.

LS vs. NN method: flexibility, bias-variance

- LS is a parametric method; limited flexibility
- NN is a semiparametric method; very flexible, tunable using k
- LS cannot automatically discover complex associations
- NN can automatically discover complex associations (given enough data)
- However, LS can be made more flexible by making the linear predictor more flexible, using 1) more predictors, 2) predictor interactions, and 3) nonlinear transformations of predictors (e.g., splines)
- more flexibility results in less bias, more variance in estimator

R Code for LS and kNN

Classification (binary case)

- Think of $\hat{Y}$ as the probability of a '1' outcome
- The predicted class $\hat{G}$ is '1' if $\hat{Y} > 0.5$ and '0' otherwise
- Can often estimate $\hat{Y}$ in a manner similar to regression (e.g., LS and NN).

Classification example

- Y- qualitative (binary) outcome (orange - 1, blue - 0)
- X_1 - quantitative predictor
- X_2 - quantitative predictor
- classification rule: $\hat{G} =$ orange if $\hat{Y} > 0.5$ else blue

Linear classification

Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 2

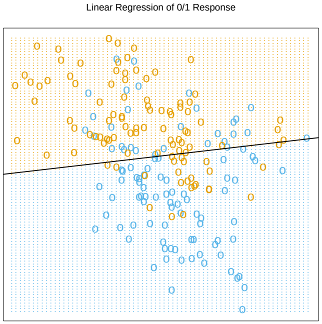


FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable (*BLUE $= 0$, ORANGE $= 1$*), and then fit by linear regression. The line is the decision boundary defined by* $x^T\hat{\beta} = 0.5$*. The orange shaded region denotes that part of input space classified as* ORANGE*, while the blue region is classified as* BLUE*.*

k-nearest neighbor classification

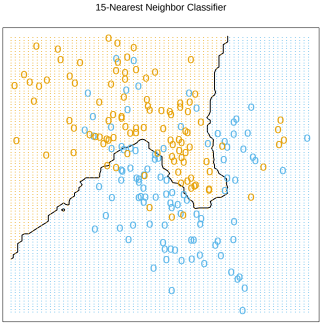


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

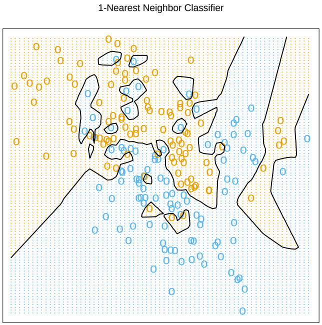


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$), *and then predicted by* 1*-nearest-neighbor classification.*

Evaluating the model and 'optimism'

- Need a mechanism to evaluate predictive quality of model and thus tune the model.
- The **prediction error** of a model is evaluated using the same objective function for estimation (e.g., mean of squared errors: $1/n \sum_{i=1}^{n}(y_i - x_i\beta)^2$). However, this can be an *optimistic* estimate if computed using the training data.
- The **training error** is computed on the training data
- The **test error** is computed on new data
- The **optimism** is the difference in training and test error
- Training error is optimistic, especially when the model is overfitted. It should not be used to select tuning parameters: for k-NN, $k = 1$ results in zero training error; the model is "overfitted"
- Need an estimate of test error to select tuning parameters

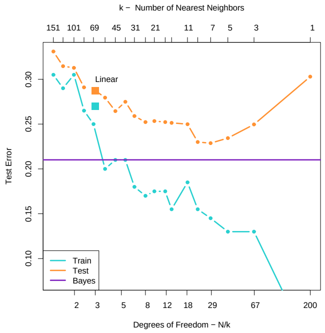


FIGURE 2.4. *Misclassification curves for the simulation example used in Figures 2.1, 2.2 and 2.3. A single training sample of size* 200 *was used, and a test sample of size* 10,000*. The orange curves are test and the blue are training error for k-nearest-neighbor classification. The results for linear regression are the bigger orange and blue squares at three degrees of freedom. The purple line is the optimal Bayes error rate.*

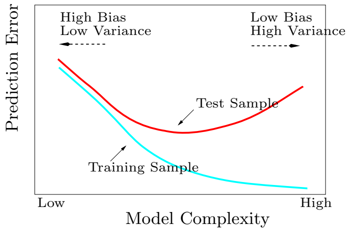


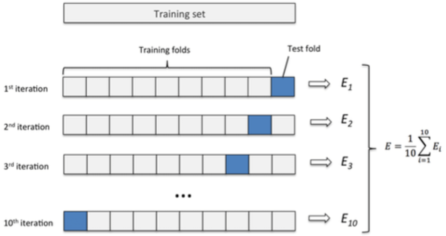
FIGURE 2.11. *Test and training error as a function of model complexity.*

Estimating test error: testing/training split

- Split data into training and testing data
- Use training data to build models
- Use testing data to evaluate models
- Simple, but not most efficient use of data

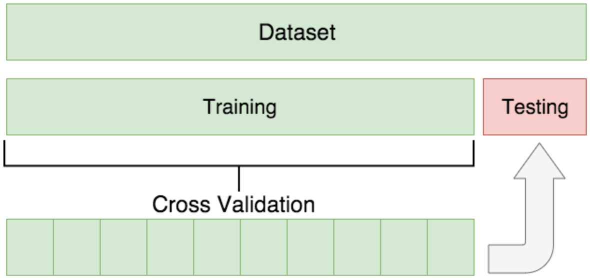
Estimating test error: k-fold cross validation

- Split data into k subsets or ‘folds’
- Use k-1 subsets to build model
- Use holdout subset to evaluate model
- Repeat for all k permutations
- Results in k estimates of test error; use mean and sd
- More complicated, but also more efficient use of data



Estimating test error: combined CV and train/test

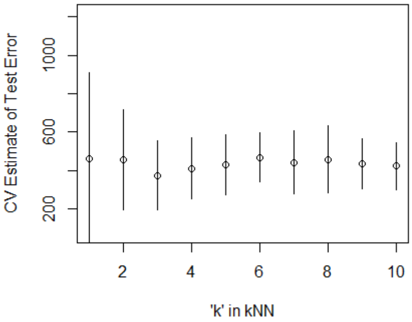
Supervised learning workflow can involves both train/test and CV:



Many model-building functions do cross-validation automatically.

Estimating test error: k-fold CV one-SD rule

- k-fold CV generates k estimates
- Can use their SD to gauge uncertainty
- When tuning a model, use the one-SD rule for parsimony
- Select least complex model whose CV error is within one SD of the best model



R Code for kNN 5-fold CV

Classification and regression trees (CART)

- trees partition feature space into a set of rectangles
- fit simple model in each partition (e.g., constant)
- for regression, mean within each partition
- for binary classification, compute probability within each partition
- trees are flexible like kNN, can automatically discover complexity

- top-left: partitions can't be described by tree
- top-right: partitions are recursive, can be described by tree
- bottom-left: tree describing top-right partitions
- bottom-right: regression surface for top-right partitions
- trees have (root, internal, and terminal) nodes and branches
- terminal nodes also called leaf node

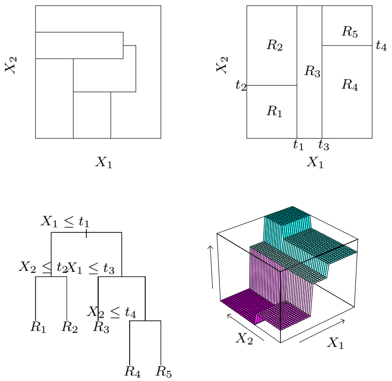


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

Tree-growing and tuning parameter

- step 1. find optimal split (on all possible variables)
- step 2. within each resulting partition find next optimal split
- repeat steps 1 and 2 until stopping criterion met
- tree complexity is a tuning parameter
- one approach: grow a large tree, stopping only when minimum node size (i.e., minimum number of $x_i \in R_m$) is reached, then "prune" tree back using a "cost-complexity" criterion

Pros and cons of (single) trees

Pros:

- interpretable (nice tree structure)
- flexible
- easily handle quant. and qual. data
- simple to implement

Cons:

- instability; sample variability in tree structure
- lack of smoothness of prediction surface in feature space
- categorical features difficult to split

R Code for CART: ‘rpart’

Random Forests

- Random forests fits many trees to the same dataset and aggregates predictions across trees. Each tree is fit to a slightly modified version of the dataset, which is generated by resampling (bootstrap). This is generically called ‘bagging’ or ‘bootstrap aggregation’.
- regression: average predictions
- classification: avarage probs. or “committee vote”
- reduces variance of estimated predictor

Random Forests

- resamples are not independent samples
- bagged trees are correlated
- weakens the “wisdom of crowds”
- random forests generates *de-correlated* trees
- performance similar to boosting for many problems

Random Forests

- in bagging, trees are identically distributed (i.d.)
- $\rightarrow$ bias of bagged trees is same as individual tree
- $\rightarrow$ can only hope to improve variance
- in boosting, trees are adaptive, not i.d.

Random Forests

- variance of the average of B i.d. random variables with variance σ^2 and pairwise correlation ρ:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

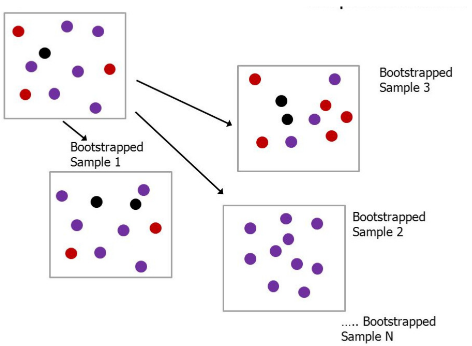
- as B grows, second term vanishes
- can only reduce first term by reducing ρ
- random forests: reduce ρ without increasing σ^2 too much

Random Forests

- random forests modified the tree-growing procedure
- before each split, select $m \leq p$ input variables at random as candidates for splitting
- typically $m = \lfloor \sqrt{p} \rfloor$ for regression
- typically $m = \lfloor p/3 \rfloor$ for classification
- random forest predictor after B trees grown is

$$hat f^{B}_{\mathrm{rf}}(x) = \frac{1}{B} \sum_{b=1}^{B} T(x; \Theta_b)$$

Resampling (bootstrap)

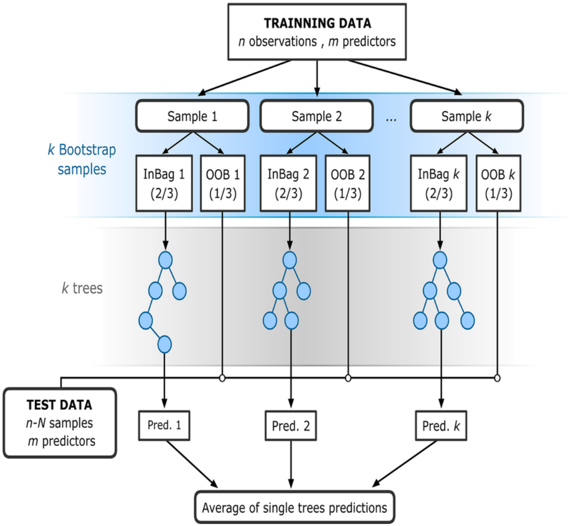


source: https://slideplayer.com/slide/8340508/

Random forests tree building

- before each tree grown, resample is split into training (in bag) and testing (out of bag; OOB); built-in CV
- random forests modifies the tree-growing procedure
- before each split, select $m \leq p$ input variables at random as candidates for splitting

Random forest



source: Rodriguez-Galiano et al. (2016). Biogeosciences. 13. 3305-3317

Pros and cons of random forest

Pros:

- more stable (less variance) than single trees
- better predictive performance
- flexible
- easily handle quant. and qual. data

Cons:

- complex to implement
- less interpretable (cannot represent as tree)
- lack of smoothness of prediction surface in feature space
- categorical features difficult to split

R Code for random forest: ‘randomForest’