Supervised classification with machine learning: A walk through

Annika Tillander

Linköping University - Department of Computer and Information Science:
Statistics and Machine Learning

18 February 2019

Outline

1. Introduction
2. Load data
3. Split data
4. Supervised classification
   - Linear discrimant analysis
   - Neural network
   - k-Nearest Neighbor
   - Random forest
5. Evaluation metrics
6. Cross-validation
7. Final test

Introduction

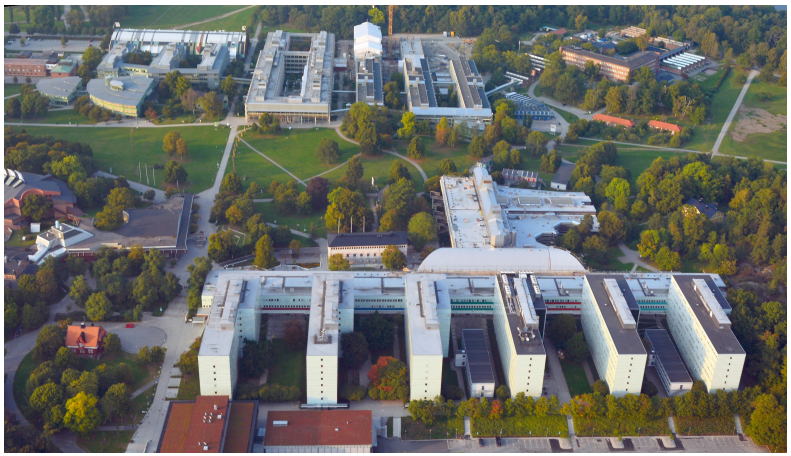


Bild tagen av Johan Fredriksson

Load data

```r
# Microarray data
load(file = "C:/Users/annti27/Dropbox/R_ladies/Data/StockMAS5.10v.Rdata

# Microarray matrix
X <- StockMAS5.10v
p <- dim(X)[1] # Number of features
n <- dim(X)[2] # Number of observations

# Survival data
load(file = "C:/Users/annti27/Dropbox/R_ladies/Data/survival.Rdata")

# Outcome 5-year breast-cancer death
# (true = brca death within 5 years)
y <- as.numeric(survdat$brca5)
```

Select features

```r
# Select the features with the most variation

# 1 for calculating over rows in the matrix
medel <- apply(X, 1, mean)
standardavvikelse <- apply(X, 1, sd)

# Coefficient of variation
cv <- (standardavvikelse / abs(medel)) * 100
s_cv <- sort(cv, decreasing = TRUE, index.return = TRUE)
top_X <- X[s_cv$ix[1:135], ]
x <- t(top_X)
```

Split the data into training and test data

Test data

```
set.seed(20192019)
# Random sample
test_urval <- sample(1:n, 29)

#Test data
test_y <- y[test_urval]
test_x <- x[test_urval, ]
# Check the size
length(test_y)
```

```
## [1] 29
```

```
dim(test_x)
```

```
## [1]  29 135
```

Split the data into training and test data

Training data

```
# Train data
train_y <- y[-test_urval]
train_x <- x[-test_urval,]
# Check the size
length(train_y)
```

```
## [1] 130
```

```
dim(train_x)
```

```
## [1] 130 135
```

Supervised classification

We have n observations where each observation $\mathbf{x} = (x_1, \ldots, x_p)$ corresponds to p number of features.
Supervised classification problem with $\mathcal{C}$ classes, class label $y_j = c$ where $j = 1, \ldots, n$, $c \in \{1, \ldots, \mathcal{C}\}$

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}.$$

Using the training data, a prediction model is built which enables prediction of new observations where the outcome is unknown.

Linear Discriminant Analysis (LDA)

Assume that the outcome in each class is modeled by the Gaussian distribution, i.e. $\mathbf{x}_c \sim N(\boldsymbol{\mu}_c, \Sigma_c)$, where $\boldsymbol{\mu}_c$ is the class mean and Σ_c is the class-wise covariance matrix.

$$D_c(\mathbf{x}) = \mathbf{x}'\Sigma_c^{-1}\boldsymbol{\mu}_c - \frac{1}{2}\boldsymbol{\mu}_c'\Sigma_c^{-1}\boldsymbol{\mu}_c + \log \pi_c$$

π_c is the prior probability of class c and $\sum_{c=1}^{C} \pi_c = 1$.

$$c^* = \text{argmax}_{c=1,\ldots,C} D_c(\mathbf{x})$$

Two classes

Assign $\mathbf{x}$ to class 1 if

$$\log \frac{\pi_1}{\pi_2} + \left(\mathbf{x} - \tfrac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\right)' \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \geq 0$$

Sparse inverse covariance estimation with the graphical lasso (J. Friedman, Hastie, and Tibshirani 2009)

$$\hat{\Sigma}_{\lambda}^{-1} = \arg\min_{\Sigma^{-1} \succ 0} \left\{ \operatorname{trace}\left(\Sigma^{-1}\hat{\Sigma}\right) - \log\det\Sigma^{-1} + \lambda \|\Sigma\|_1 \right\}$$

$\|\cdot\|_1$ denotes the ℓ^1 norm

gLasso

Partition of $\mathbf{x}$ into $\mathbf{z}_1 = (x_1, \ldots, x_{p-1})$ and $z_2 = x_p$

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{pmatrix}$$

$\mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})'$, $\mathbf{W} = \mathbf{S} + \lambda \mathbf{I}$ and $\beta = \mathbf{W}_{11}^{-1} w_{12}$

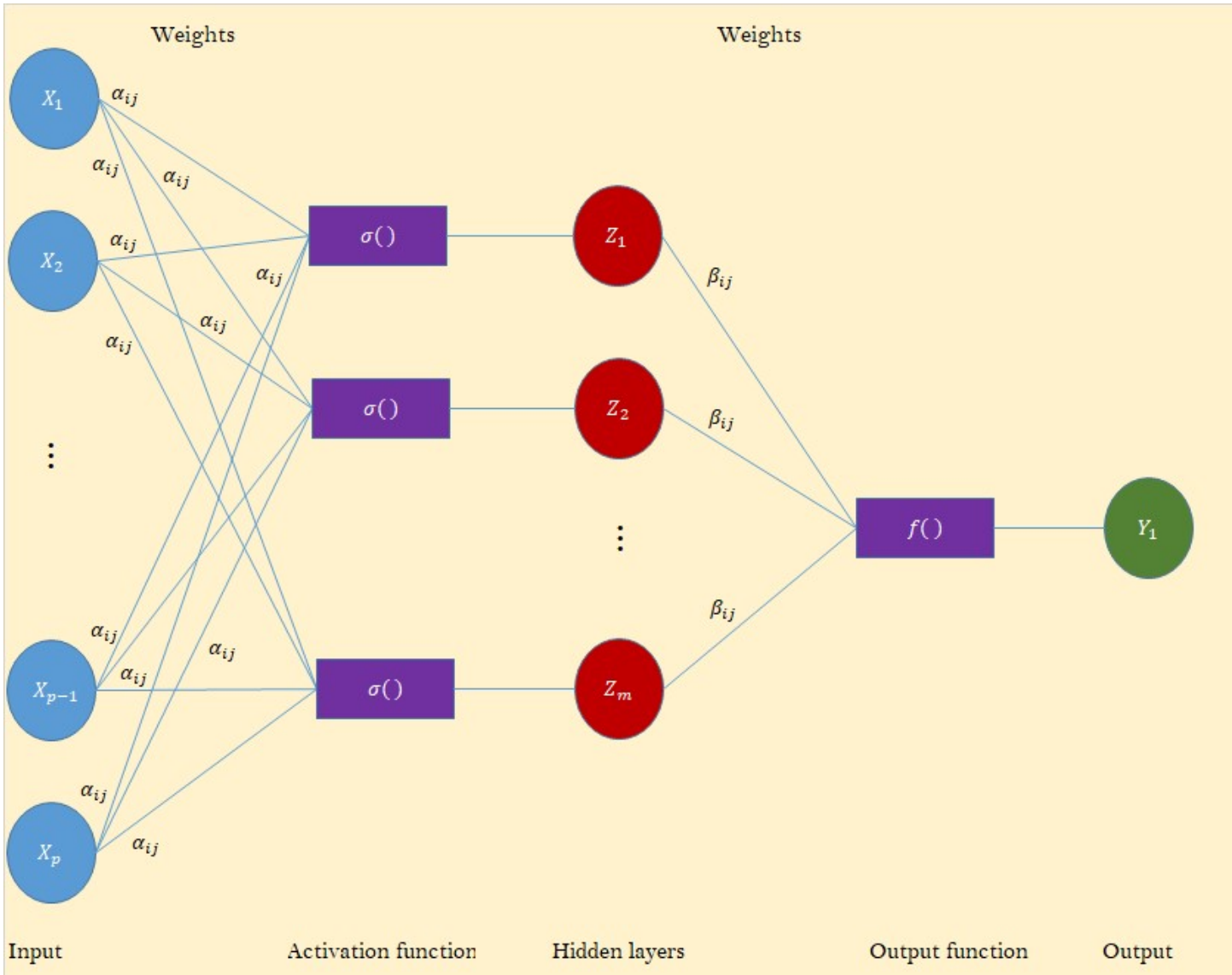
$$\mathbf{W}_{11}\beta - s_{12} + \lambda \cdot \text{sign}(\beta) = 0$$

$\mathbf{V} = \mathbf{W}_{11}$

$$\hat{\beta}_j \leftarrow \frac{T\left(s_{12_j} - \sum_{k \neq j} V_{kj}\hat{\beta}_k, \lambda\right)}{V_{jj}}, \text{ Update } w_{12} = \mathbf{W}_{11}\hat{\beta}$$

Then $\hat{\theta}_{12} = -\hat{\beta} \cdot \hat{\theta}_{11}$ where $\hat{\theta}_{11} = \frac{1}{w_{22} - w_{12}'\hat{\beta}}$

Neural Networks



Activation/output functions

Identity: $f(x) = x$

Step:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

ReLU:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

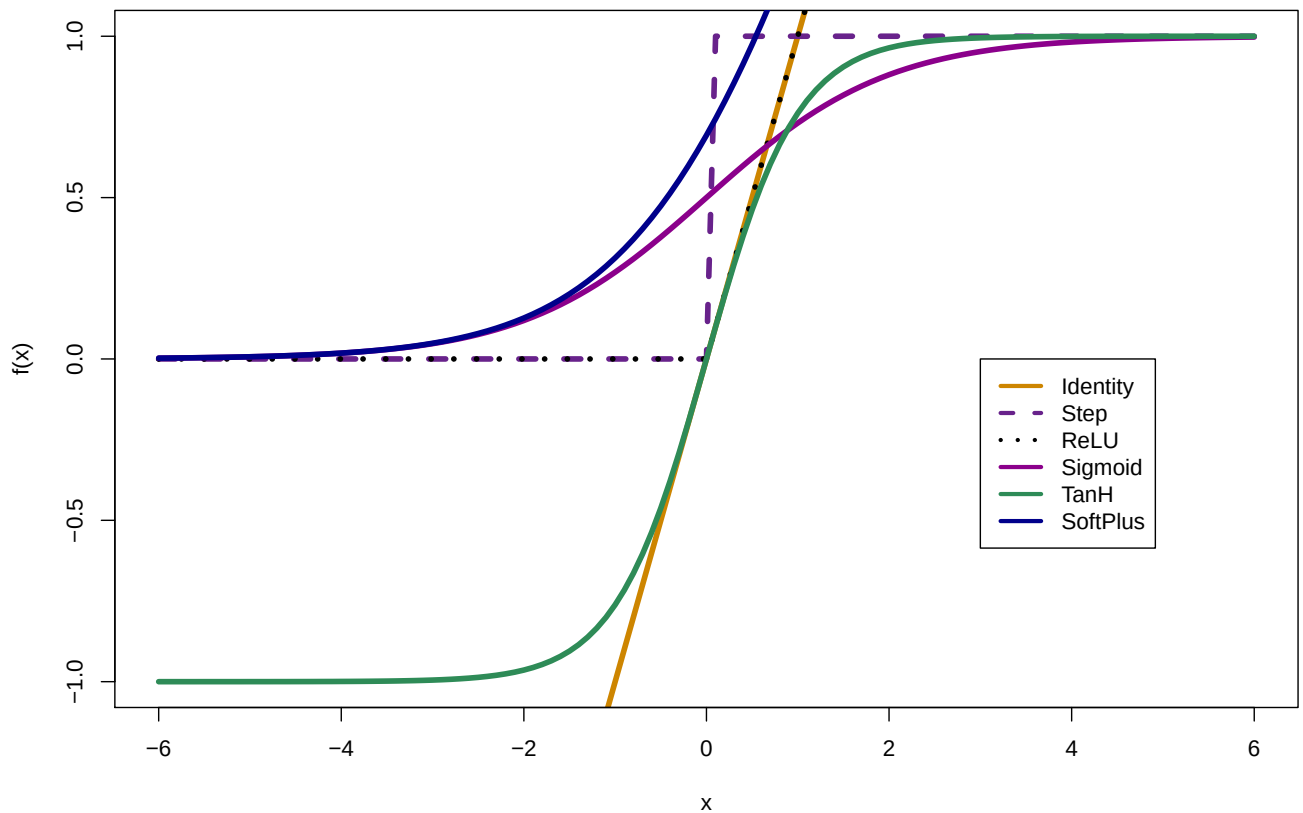
Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

TanH:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

SoftPlus: $\log(1 + e^{x})$

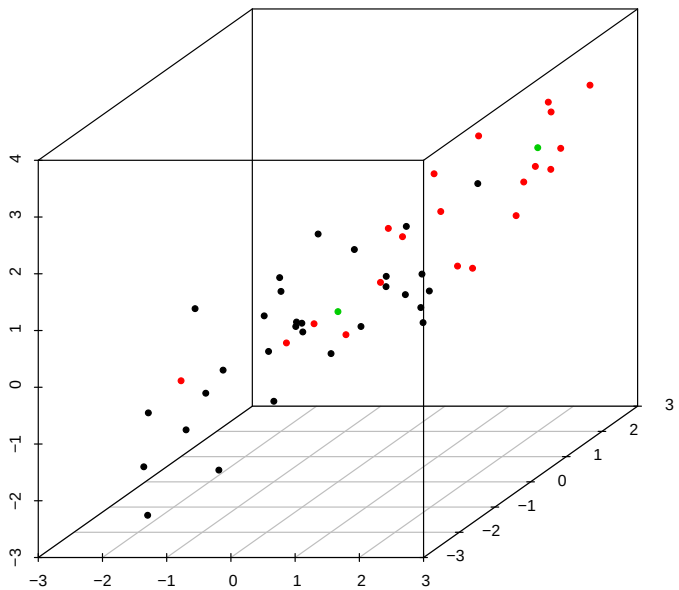


k-Nearest Neighbor

For a given a point x_0 identify the k closest points $x_{(r)}, r \in (1, \ldots, k)$ in the training data and then classify using majority vote among the k-neighbors. To find the k-closest some kind of distance measure d is used e.g. Euclidian distance in feature space (T. Hastie, Tibshirani, and Friedman 2003):

$$d_{(i)} = \|x_i - x_0\|_2$$

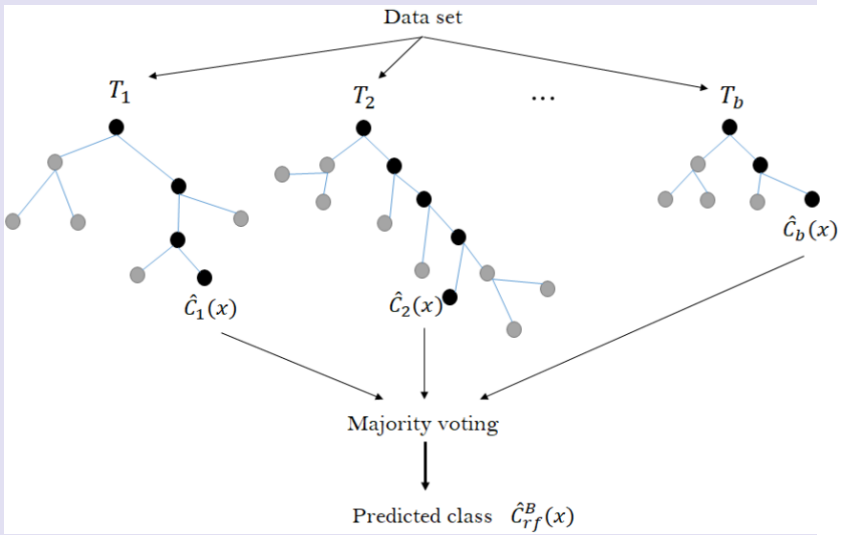
where $\|\cdot\|_2$ denotes the ℓ^2 norm.



Random forest

Algorithm

1. for (b in 1:B)
   a. Draw a random bootstrap sample $\mathbf{Z}^*$ of size N from the traning data.
   b. Grow a random forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
   i. Select m variables at random from the p variables.
   ii. Pick the best variable/split-point aming the m.
   iii. Split the node into to daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$
   To make a prediction at a new point x:
   Let $\hat{C}_b(x)$ be the class prediction of the bth random forest tree. Then $\hat{C}_{rf}^B(x) = \text{majority vote}\left\{\hat{C}_b(x)\right\}_1^B$
   (T. Hastie, Tibshirani, and Friedman 2003)



Evaluation metrics

1. True Negatives (TN): True negatives are the cases when the actual class of the data point was Negative (0) and the predicted is also Negative (0).
2. False Positives (FP): False positives are the cases when the actual class of the data point was Negative (0) and the predicted is Positive (1).
3. False Negatives (FN): False negatives are the cases when the actual class of the data point was Positive (1) and the predicted is Negative (0).
4. True Positives (TP): True positives are the cases when the actual class of the data point was Positive (1) and the predicted is also Positive (1).

Confusion Matrix

Predicted	Actual/True	
	Negative (0)	Positive (1)
Negative (0)	TN	FN
Positive (1)	FP	TP

Evaluation metrics

Accuracy:

$$\frac{TN+TP}{TN+FN+FP+TP}$$

Require balanced classes.

Precision (Positive Predictive Value):

$$\frac{TP}{TP+FP}$$

Sensitivity (True Positive Rate TPR/Recall):

$$\frac{TP}{TP+FN}$$

Specificity (True Negative Rate):

$$\frac{TN}{TN+FP}$$

False Positive Rate FPR:

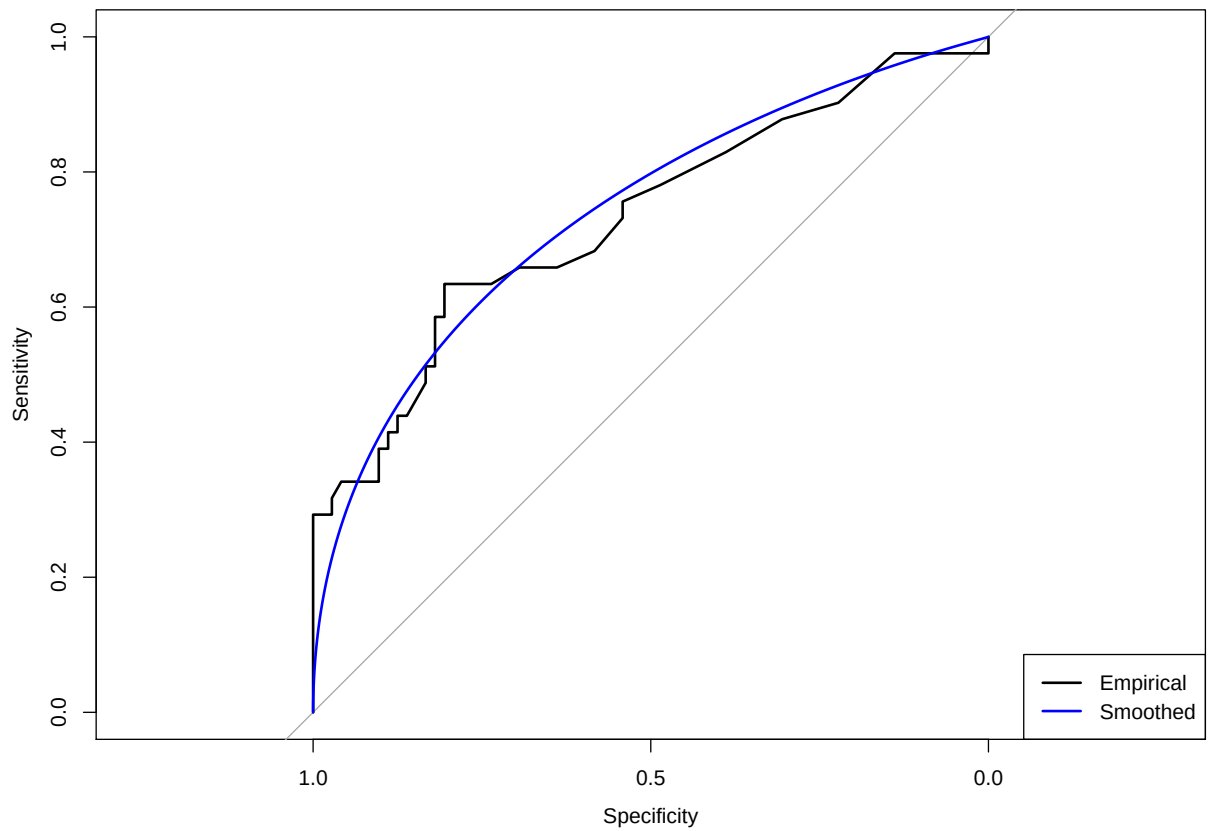
$$\frac{FP}{TN+FP} = 1 - \frac{TN}{TN+FP}$$

ROC and AUC

Receiver Operating Characteristics (ROC) and Area Under The Curve (AUC) is a performance measurement for a binary classification problem at various thresholds settings.
ROC is a probability curve plotted for FPR (1-Specificity) (x) vs TPR (y).
AUC represents degree or measure of separability.
It tells how much model is capable of distinguishing between classes, the higher AUC the better the model is at predicting.

ROC and AUC

```
data(aSAH) # Using package pROC
rocobj <- roc(aSAH$outcome, aSAH$s100b)
plot.roc(rocobj)
plot.roc(smooth(rocobj), add=TRUE, col="blue")
legend("bottomright", legend=c("Empirical", "Smoothed"),
       col=c(par("fg"), "blue"), lwd=2)
```



k-fold Cross validation

A technique for assessing the performance of machine learning models, estimate how accurate the predictions will be. Use for tuning the parameters for the different classifications methods to get optimal prediction.
The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
   - Take the group as test data
   - Take the remaining groups as training data
   - Fit a model on the training set and evaluate it on the test set
   - Retain the evaluation score
4. Summarize the skill of the model using the sample of model evaluation scores

Different packages in *R* for efficient cross validation e.g. `caret`, `cvTools` and `DAAG`

The steps

1. Choose relevant parameter space to tune over for each classifier e.g.
   - LDA with gLASSO λ
   - NN number of hidden layers
   - knn number of neighbors
   - Random forest number of decision trees in the forest and the number of features considered by each tree when splitting a node
2. Choose k and split the data into k-sets
3. Run the cross-validation for each parameter space
4. Select evalutation metric

Some set up

```
# Cross validation
# Numer of folds
my_k <- 8
# Subset data into folds
set.seed(20190909)
# In caret
flds <- createFolds(train_y, k = my_k, list = TRUE, returnTrain = FALSE

# In cvTools
folds <- cvFolds(NROW(train_x), K=my_k)

# Some functions for cross-validation
# CVknn in bcROCsurface knn.cv in class
# function train and trainControl in caret
# rf.crossValidation in rfUtilities
# cv.lm in DAAG
```

Some set up

```r
# Paramter space use only 6 possible
ps <- 6
lambda <-  (3:8)/10 # Lambda for gLASSO
layers <- 2:7 # hidden layers for nn
nb <- 2:7 # neighbors for knn
nf <- 5*(2:7) # Number of features randomly sampled as candidates at ea

# Scaling and Centering advised for neural networks
s_train_x <- scale(train_x, center = TRUE, scale = TRUE)
xy <- cbind(train_y, s_train_x)

# Turning outcome into factor for random forest
rf_y <- as.factor(train_y)
rf_xy <- data.frame(rf_y, train_x)

# Collect outcomes
temp_auc <- matrix(NaN, 4, my_k)

# For the result
result_auc <- matrix(NaN, 4, ps)
rownames(result_auc) <- c("LDA", "NN", "kNN", "RF")
colnames(result_auc) <- 1:ps
```

The classifiers in R

```
# Graphical lasso in package glasso to
# estimate inverse covariance matrix
i <- 1
j <- 1
s <- cov(train_x[-flds[[j]],])
inv_s <- glasso(s, rho=lambda[i])$wi
# Neural network in package nnet gives
mod_nn <- nnet(train_y ~ ., data=xy[-flds[[j]],],
               size = layers[i], trace = FALSE)
# To make prediction from model use predict in package stats
pred_nn <- as.numeric(predict(mod_nn, xy[flds[[j]],])>0.1)
# k-Nearest Neighbour in package class
pred_knn <- knn(train = train_x[-flds[[j]],],
                test = train_x[flds[[j]],],
                cl = train_y[-flds[[j]]], k = nb[i])
# Random forest in package random Forest
mod_rf <- randomForest(rf_y  ~ ., data = rf_xy[-flds[[j]],],
                       mtry = nf[i])
```

Loop

```
for(i in 1:ps) # Loop over parameter space
{
  for (j in 1:my_k) # Loop over folds
  {
    true_y <- train_y[flds[[j]]]
    s <- cov(train_x[-flds[[j]],])
    is <- glasso(s, rho=lambda[i])$wi
    pred_lda <- my_lda(x = train_x[-flds[[j]],], y = train_y[-flds[[j]]
                       sigma_inv=is, tx=train_x[flds[[j]],])
    mod_nn <- nnet(train_y ~ ., data=xy[-flds[[j]],], size = layers[i],
    pred_nn <- as.numeric(predict(mod_nn, xy[flds[[j]],])>0.1)
    pred_knn <- knn(train = train_x[-flds[[j]],], test = train_x[flds[[
                cl = train_y[-flds[[j]]], k = nb[i])
    mod_rf <- randomForest(rf_y  ~ ., data = rf_xy[-flds[[j]],], mtry =
    pred_rf <- as.numeric(predict(mod_rf, rf_xy[flds[[j]],]))-1
    # Using auc from ModelMetrics
    temp_auc[, j] <- c(auc(true_y, pred_lda), auc(true_y, pred_nn), auc
  }
    result_auc[,i] <- apply(temp_auc, 1, mean)
}
```

Training result

```
result_auc
```

```
##              1         2         3         4         5         6
## LDA 0.5238668 0.5572001 0.5858745 0.6064103 0.6272436 0.6700549
## NN  0.5133929 0.5357143 0.5580357 0.5000000 0.5147092 0.5362752
## kNN 0.5309180 0.5118132 0.5166781 0.5081845 0.5446429 0.4866071
## RF  0.4910714 0.4862637 0.4862637 0.4907280 0.4862637 0.4907280
```

```
# Get parameter values that generate best AUC
lda_max <- as.numeric(colnames(result_auc)[result_auc[1,]==max(result_a
nn_max <- as.numeric(colnames(result_auc)[result_auc[2,]==max(result_au
knn_max <- as.numeric(colnames(result_auc)[result_auc[3,]==max(result_a
rf_max <- as.numeric(colnames(result_auc)[result_auc[4,]==max(result_au
```

The test

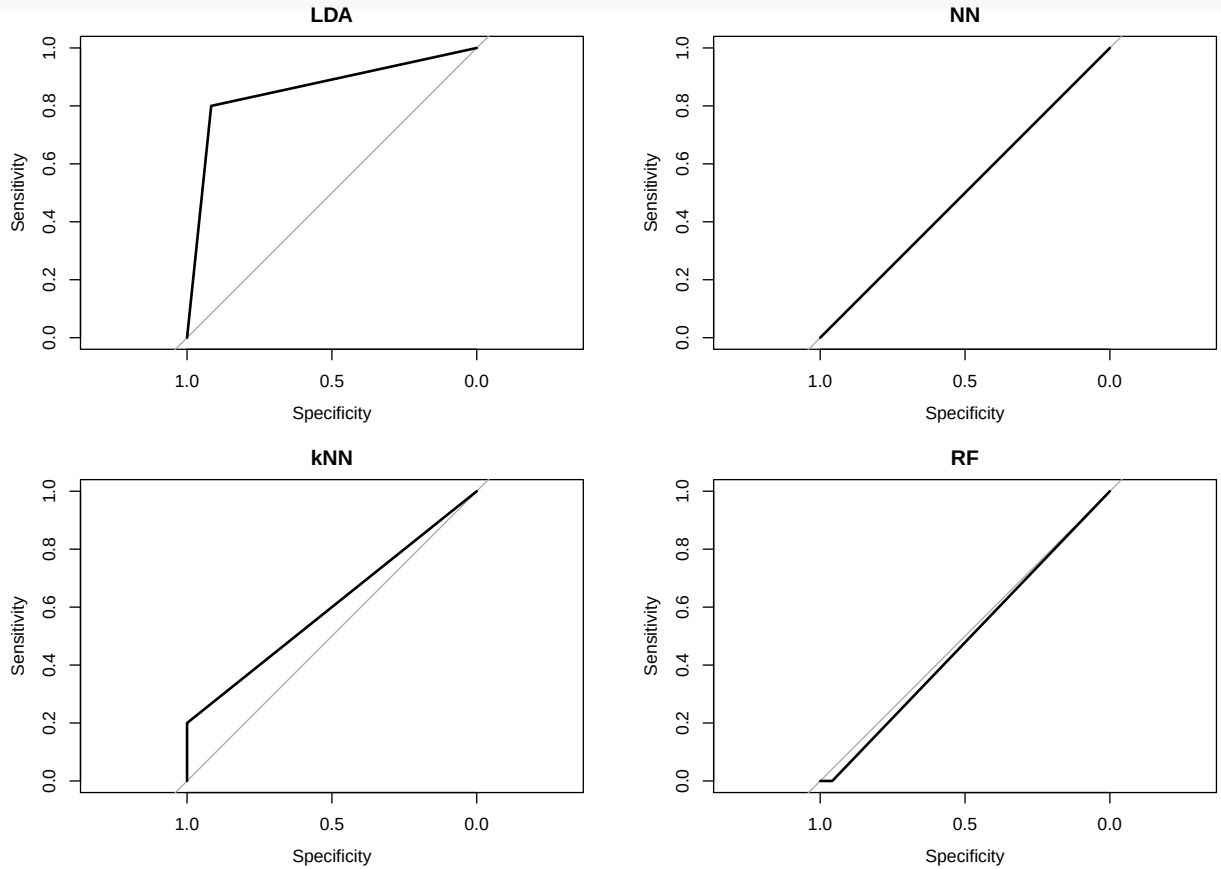
```
s <- cov(train_x)
is <- glasso(s, rho=lambda[lda_max])$wi
pred_lda <- my_lda(x = train_x, y = train_y,
                        sigma_inv=is, tx=test_x)
mod_nn <- nnet(train_y ~ ., data=xy, size = layers[nn_max], trace = FAL
c_test_x <-  scale(test_x, center = TRUE, scale = TRUE)
pred_nn <- as.numeric(predict(mod_nn, c_test_x)>0.1)
pred_knn <- knn(train = train_x, test = test_x, cl = train_y, k = nb[kn
mod_rf <- randomForest(rf_y  ~ ., data = rf_xy, mtry = nf[rf_max])
rf_test_x <- test_x
colnames(rf_test_x) <- colnames(rf_xy)[-1]
pred_rf <- as.numeric(predict(mod_rf, rf_test_x))-1
```

Calculating the ROC-curve

```
# roc in package pROC
lda_roc <- roc(test_y, pred_lda)
nn_roc <- roc(test_y, pred_nn)
pred_knn <- as.numeric(pred_knn)-1
knn_roc <- roc(test_y, pred_knn)
rf_roc <- roc(test_y, pred_rf)
```

The test result

```
par(mfrow=c(2,2))
plot.roc(lda_roc, main="LDA")
plot.roc(nn_roc, main="NN")
plot.roc(knn_roc, main="kNN")
plot.roc(rf_roc, main="RF")
```



References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2009. *Graphical Lasso- Estimation of Gaussian Graphical Models*.
Hastie, T., R. Tibshirani, and J. H. Friedman. 2003. *The Elements of Statistical Learning*. Corrected. Hardcover; Springer.