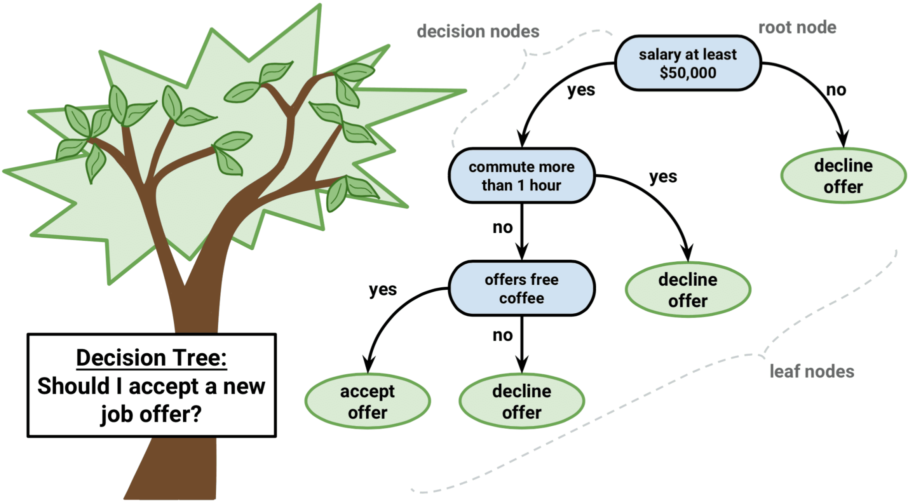
Statistical Learning and Visualization

Tree-based Methods

Maarten Cruyff

Classification trees



Content

1. Classification trees
2. Pruning
3. Bagging, random forests
4. Boosting
5. Support Vector Machines

Classification trees

1. Recursive binary splitting algorithm
2. Splits features on basis of node *purity*

- Gini index
- deviance

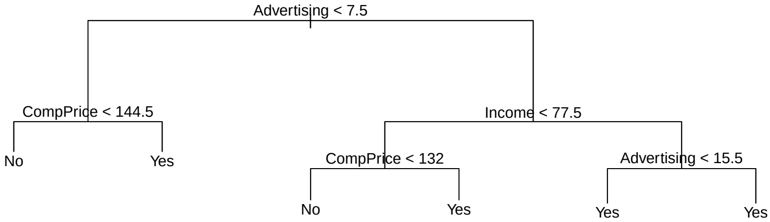


Figure 1: Sale of car seats (yes or No)

Recusrsive binary splitting

Algorithm

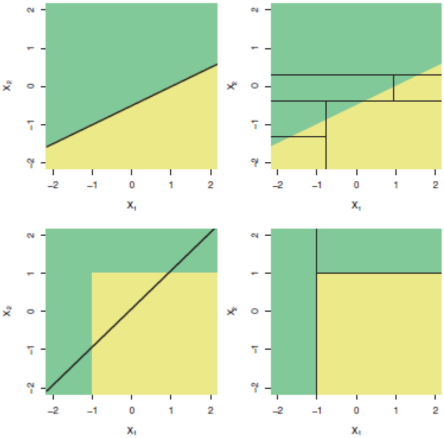
1. Divide feature space in non-overlapping, rectangular regions
2. Choose splits that minimize *node impurity* (homogeneity of nodes)
3. Assign region to class with highest mode
4. Stop when node purity no longer increases

Algorithm is top-down and greedy, so

- high variance

Classification with trees or regression?

- regression works best



- trees work best for B

Package tree

Growing and plotting trees with function tree()

```
train_tree <- tree(formula, data, split = c("deviance", "gini"))

plot(train_tree)
text(train_tree)
```

- minimization of deviance or gini impurity
- text() for adding labels to nodes

Methods to reduce variance:

1. Pruning
   - cut branches with cross-validation and regularization
2. Bagging
   - average predictions of bootstrapped trees
3. Random forests
   - average predictions of decorrelated bootstrapped trees
4. Boosting
   - weighted combination of weak classifiers (small trees)

Section 1

Pruning

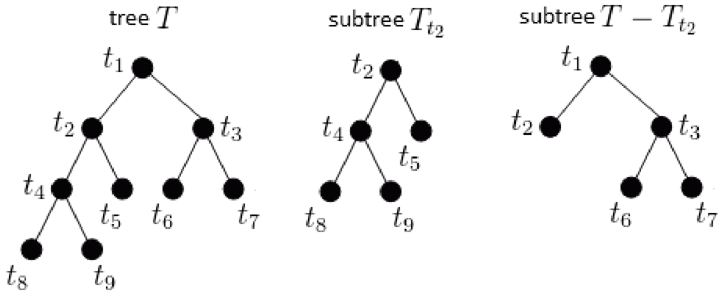
Cost-complexity pruning (package ‘tree)

1. Cross-validate the tree
2. Shows deviance/misclassification as function of nodes
3. Obtain predictions test set

```
cv.tree(fit_tree, method = c("deviance", "misclass"))

pruned_tree <- prune.tree(fit_tree, best = <number>)

predict(pruned_tree, newdata, type = "class")
```

- `cv.tree()` deviance/misclassification as function number of nodes
- `best` in `prune.tree()` is optimal number of nodes in `cv.tree()`
- default `type` yields predicted probabilities

Pruning example

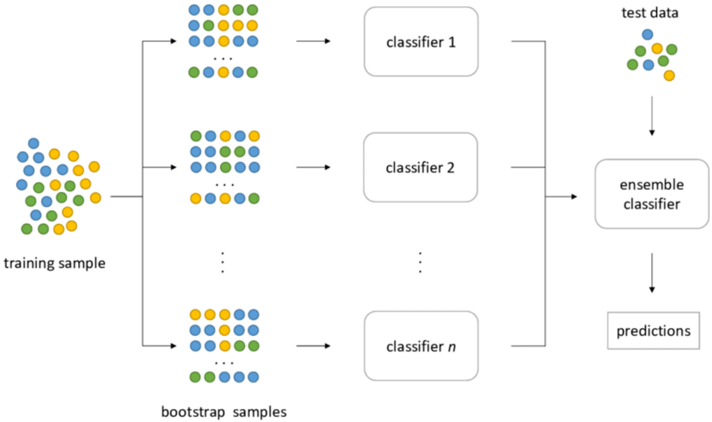


Section 2

Bagging, random forests

Algorithm

1. Fit classification trees to B bootstrap samples
2. Average the predictions
3. Out-Of-Bag (OOB) as estimate validation error



Bagging vs random forests

Bagging

- considers all predictors at each split
- best predictors turn up in each tree
- highly correlated trees
- high variance

Random forests

- considers random sample of predictors at each step
- all predictors get a fair chance
- decorrelated trees
- lower variance

Out-of-bag error rate

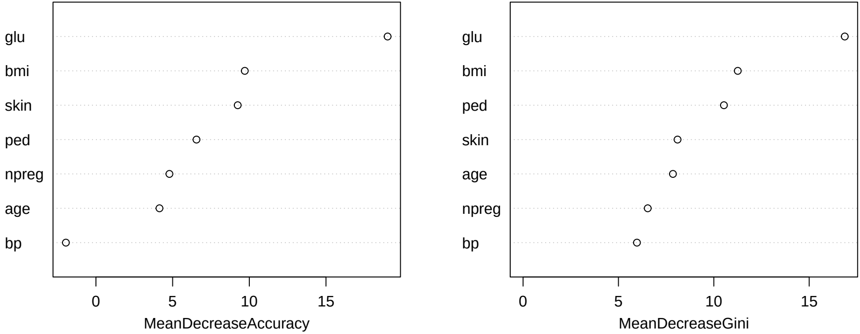
On average 1/3 of observations not in bootstrap (Out-Of-Bag)

- OOB cases used to compute validation error
- no need for cross validation
- computationally very efficient

Variable importance

When averaging trees the tree structure is lost

- how to interpret solution then?
- effect predictors averaged over trees
- visualize with *variable importance* plots



Package randomForest

Training and prediction with bagging/random forest

```
fit <- randomForest(formula, data,
                    ntree    = 500,
                    mtry     = <number predictors at each split>,
                    importance = TRUE)

varImpPlot(fit)

predict(fit, newdata, type = "prob")
```

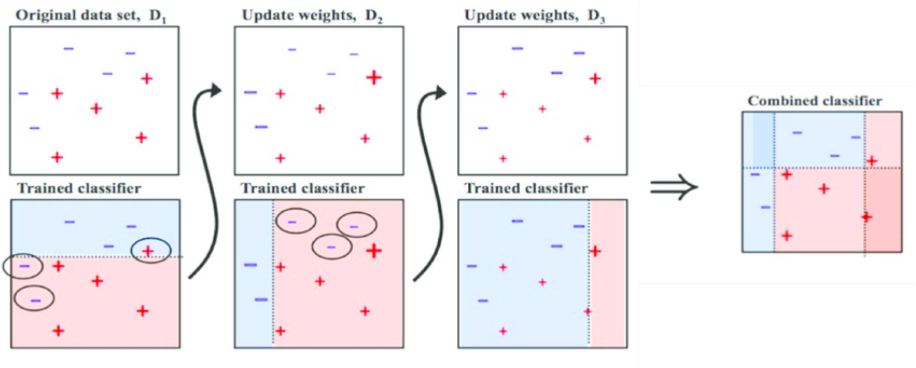
- `mtry`: default random forest (bagging total number predictors)
- `ntree` is tuning parameter (overfitting when too large)
- `importance = TRUE` necessary for `varImpPlot()`
- default type yields predicted class

Section 3

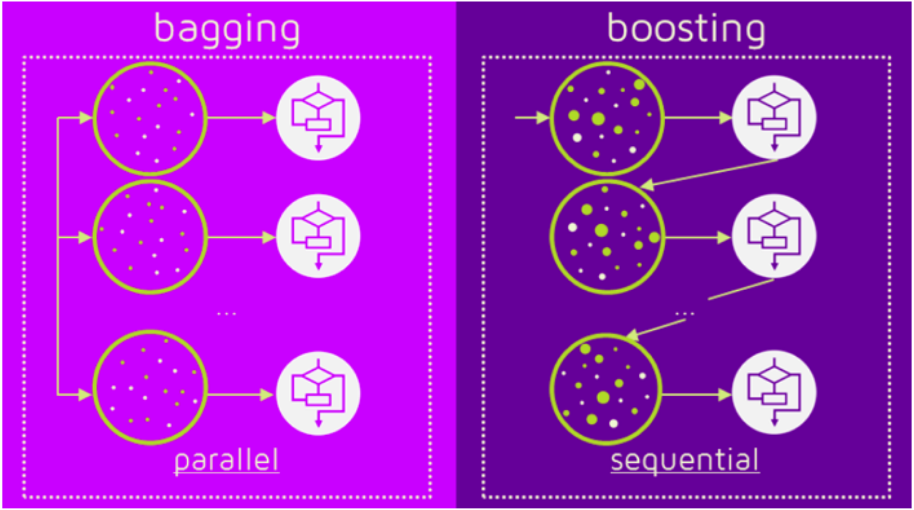
Boosting

Algorithm

1. Apply a weak classifier (e.g. stump) to training data
2. Increase weights for incorrect classifications, and repeat
3. Classifier is linear combination of weak classifiers



Boosting vs bagging/random forest



Boosting with package fastAdaboost

Boosting a single model

- `nIter` is number of weak classifiers

```
ada <- adaboost(formula, data, nIter)

predict(ada, newdata)
```

- `nIter` is tuning parameter (overfitting when too large)
- predictions include classes, probabilities and misclassification error

Boosting with package caret

Determine nIter with cross validation (caret)

```
ada <- train(formula, data,
             method    = "adaboost",
             trControl = trainControl(method = "cv", number = 5),
             tuneGrid  = expand.grid(method = "Adaboost.M1",
                                     nIter  = <test sequence>))

predict(ada, newdata, type = "prob")
```

- "Adaboost.M1" restricts search to one of two methods
- default type yields predicted class

Section 4

Support Vector Machines (SVM)

SVM for binary classification

Classifiers using support vectors

1. *maximal margin classifier*
   - classes perfectly separable by hyperplane

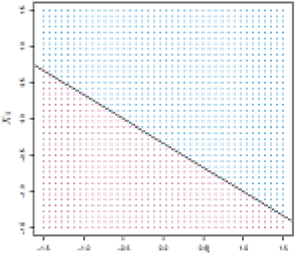
2. *support vector classifier*
   - allows for non-separable cases

3. *support vector machine*
   - allows for non-linear boundaries

Hyperplane

Divides the feature space in two

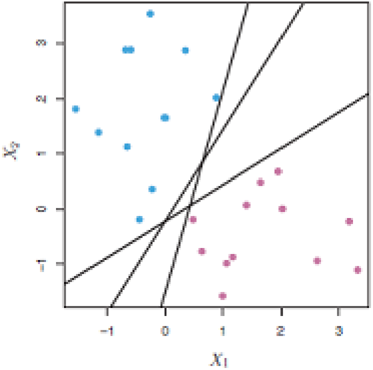
- in two dimensions hyperplane is simply a line



Separating hyperplane

Perfectly separates the two classes of the outcome variable

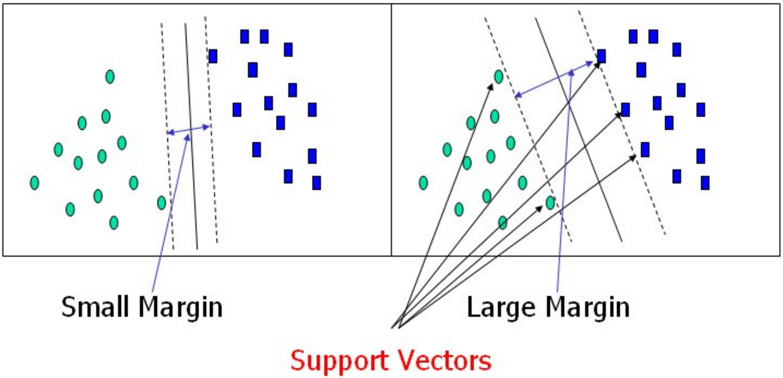
- hyperplane not uniquely identified
- high variance



Maximal Margin Classifier

Identifies hyperplane by specification of a maximal marging

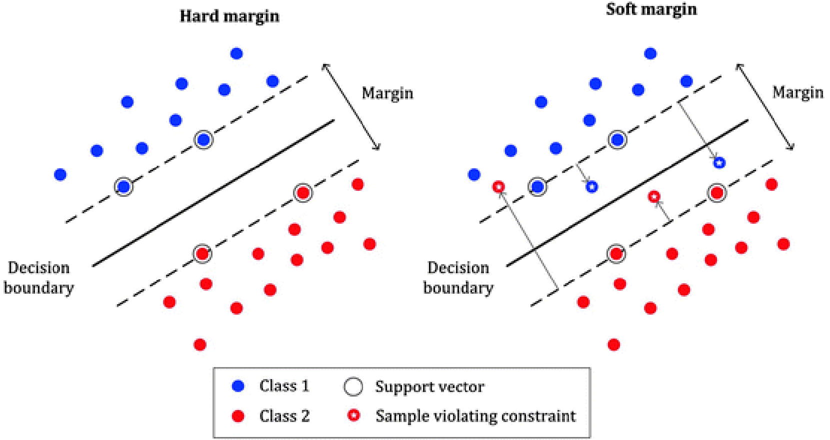
- points on margin are support vectors
- only works if cases are *separable*



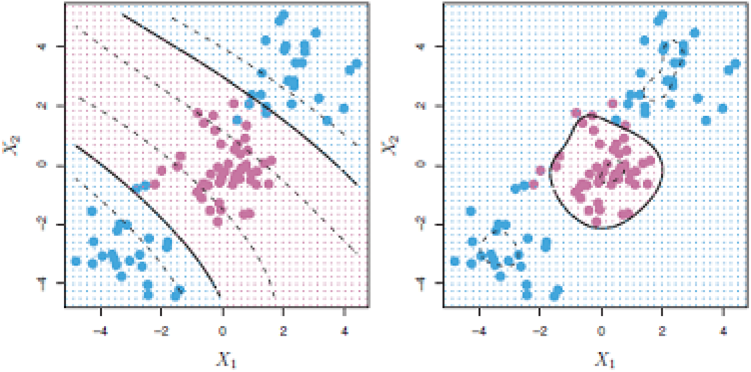
Support Vector Classifier (SVC)

Allows for violations of the margin (soft margin)

- budget for violations is called *cost* (C)
- cases the wrong side of hyperplane contribute to the cost



Allows for nonlinear hyperplanes, e.g. polynomial and radial



SVM with pacakge e1071

Training and prediction

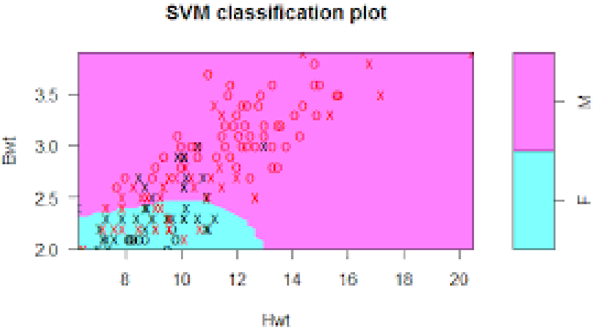
```
svm_train <- tune(svm, formula, data,
                  degree = 3, #default
                  coef0  = 0, #default
                  cost   = 1, #default
                  kernel = c("linear", "polynomial", "radial"),
                  ranges = list(cost = <sequence>), etc.)

svm_train$best.model # performance summary

svm_class <- predict(svm_train, newdata, probability = TRUE)
svm_prob  <- attr(svm_class, "probabilities")
```

- `cost`, `degree` and `coef0` are tuning parameters
- `ranges` works similar as `tuneGrid()`

SVM classification plot

Compression hyperplane two dimensions

```
plot(svm_train$best.model, data, x1 ~ x2)
```



Pro's and con's classifiers

BLR

- robust against outliers but potentially unstable

LDA

- better stability but sensitive to normality violations

Tree-based methods

- top-down and greedy, so bias-variance control needed
- boosting considered as one of the best methods

SVM

- similar to BLR (same loss function), but allows for non-linearity