**Feature Selection**

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and is computationally intractable for all but the smallest of feature sets. The choice of evaluation metric heavily influences the algorithm, and it is these evaluation metrics which distinguish between the three main categories of feature selection algorithms: wrappers, filters and embedded methods.

**Wrapper methods**

Wrapper methods use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset. As wrapper methods train a new model for each subset, they are very computationally intensive, but usually provide the best performing feature set for that particular type of model.

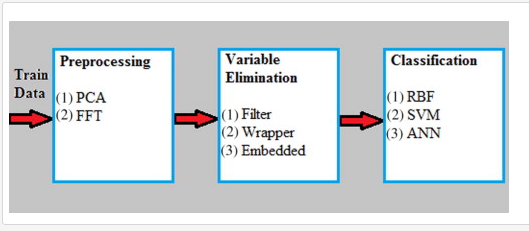
**Filter methods**

Filter methods use a proxy measure instead of the error rate to score a feature subset. This measure is chosen to be fast to compute, while still capturing the usefulness of the feature set. Common measures include the [mutual information](https://en.wikipedia.org/wiki/Mutual_information), the [pointwise mutual information](https://en.wikipedia.org/wiki/Pointwise_mutual_information), [Pearson product-moment correlation coefficient](https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient), inter/intra class distance or the scores of [significance tests](https://en.wikipedia.org/wiki/Statistical_hypothesis_testing) for each class/feature combinations. Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. This lack of tuning means a feature set from a filter is more general than the set from a wrapper, usually giving lower prediction performance than a wrapper. However the feature set doesn't contain the assumptions of a prediction model, and so is more useful for exposing the relationships between the features. Many filters provide a feature ranking rather than an explicit best feature subset, and the cutoff point in the ranking is chosen via [cross-validation](https://en.wikipedia.org/wiki/Cross-validation_(statistics)). Filter methods have also been used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems.

**Embedded methods**

Embedded methods are a catch-all group of techniques which perform feature selection as part of the model construction process. The exemplar of this approach is the [LASSO](https://en.wikipedia.org/wiki/Lasso_(statistics)) method for constructing a linear model, which penalizes the regression coefficients with an L1 penalty, shrinking many of them to zero. Any features which have non-zero regression coefficients are 'selected' by the LASSO algorithm. Improvements to the **LASSO** include Bolasso which bootstraps samples,[[6]](https://en.wikipedia.org/wiki/Feature_selection" \l "cite_note-Bolasso-6) and FeaLect which scores all the features based on combinatorial analysis of regression coefficients.[[7]](https://en.wikipedia.org/wiki/Feature_selection#cite_note-FeaLect-7) One other popular approach is the Recursive Feature Elimination algorithm, commonly used with [Support Vector Machines](https://en.wikipedia.org/wiki/Support_Vector_Machines) to repeatedly construct a model and remove features with low weights. These approaches tend to be between filters and wrappers in terms of computational complexity.

In traditional statistics, the most popular form of feature selection is [stepwise regression](https://en.wikipedia.org/wiki/Stepwise_regression), which is a wrapper technique. It is a [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) that adds the best feature (or deletes the worst feature) at each round. The main control issue is deciding when to stop the algorithm. In machine learning, this is typically done by [cross-validation](https://en.wikipedia.org/wiki/Cross-validation_(statistics)). In statistics, some criteria are optimized. This leads to the inherent problem of nesting. More robust methods have been explored, such as [branch and bound](https://en.wikipedia.org/wiki/Branch_and_bound) and piecewise linear network.



**################################ Feature Selection ################################**

# Feature selection algorithms: wrappers, filters and embedded methods

# Select Features

#

# Feature selection is a process where you automatically select those features in your data that contribute

# most to the prediction variable or output in which you are interested.

#

# Having too many irrelevant features in your data can decrease the accuracy of the models.

# Three benefits of performing feature selection before modeling your data are:

# Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.

# Improves Accuracy: Less misleading data means modeling accuracy improves.

# Reduces Training Time: Less data means that algorithms train faster.

##################################################################################

#Feature Selection with the Caret R Package

# ensure the results are repeatable

set.seed(7)

# load the library

library(mlbench)

library(caret)

library(corrplot)

# load the data

data(PimaIndiansDiabetes)

### Summary of data

glimpse(PimaIndiansDiabetes)

##################################################################################

# Remove variables having high missing percentage (50%)

PimaIndiansDiabetes <- PimaIndiansDiabetes[, colMeans(is.na(PimaIndiansDiabetes)) <= .5]

dim(PimaIndiansDiabetes)

##################################################################################

############################## Filter Method ################################

### #Remove Zero and Near Zero-Variance Predictors

### #Remove highly Correlated Predictors

#################################################

#Remove Zero and Near Zero-Variance Predictors

nzv <- nearZeroVar(PimaIndiansDiabetes)

nzvcol <- colnames(PimaIndiansDiabetes)[nzv]

if (length(nzvcol)) {

# Remove highly correlated variables and create a new dataset

PimaIndiansDiabetes <- PimaIndiansDiabetes[, -which(colnames(PimaIndiansDiabetes) %in% nzvcol)]

}

dim(PimaIndiansDiabetes)

rm(nzv, nzvcol)

#################################################

#Remove highly Correlated Predictors

# Identifying numeric variables

numericData <- PimaIndiansDiabetes[sapply(PimaIndiansDiabetes, is.numeric)]

# Calculate correlation matrix

descrCor <- cor(numericData)

# Print correlation matrix and look at max correlation

print(descrCor)

summary(descrCor[upper.tri(descrCor)])

# Check Correlation Plot

corrplot(descrCor, order = "FPC", method = "color", type = "lower", tl.cex = 0.7, tl.col = rgb(0, 0, 0))

# find attributes that are highly corrected

highlyCorrelated <- findCorrelation(descrCor, cutoff=0.75)

# print indexes of highly correlated attributes

print(highlyCorrelated)

# Indentifying Variable Names of Highly Correlated Variables

highlyCorCol <- colnames(numericData)[highlyCorrelated]

# Print highly correlated attributes

highlyCorCol

# Remove highly correlated variables and create a new dataset

if (length(highlyCorCol)) {

# Remove highly correlated variables and create a new dataset

PimaIndiansDiabetes <- PimaIndiansDiabetes[, -which(colnames(PimaIndiansDiabetes) %in% highlyCorCol)]

}

dim(PimaIndiansDiabetes)

rm(descrCor, numericData, highlyCorCol, highlyCorrelated)

##################################################################################

#### Models with Built-In Feature Selection

#These models are thought to have built-in feature selection:

# ada, bagEarth, bagFDA, bstLs, bstSm, C5.0, C5.0Cost, C5.0Rules, C5.0Tree, cforest,

# ctree, ctree2, cubist, earth, enet, evtree, extraTrees, fda, gamboost, gbm, gcvEarth,

# glmnet, glmStepAIC, J48, JRip, lars, lars2, lasso, LMT, LogitBoost, M5, M5Rules, nodeHarvest,

# oblique.tree, OneR, ORFlog, ORFpls, ORFridge, ORFsvm, pam, parRF, PART, penalized, PenalizedLDA,

# qrf, relaxo, rf, rFerns, rpart, rpart2, rpartCost, RRF, RRFglobal, smda, sparseLDA.

## Rank Features By Importance

# ensure results are repeatable

set.seed(7)

# load the library

# library(mlbench)

# library(caret)

# prepare training scheme

control <- trainControl(method="repeatedcv", number=10, repeats=3)

# train the model

#http://topepo.github.io/caret/modelList.html

model <- train(diabetes~., data=PimaIndiansDiabetes, method="xgbLinear", preProcess="scale", trControl=control)

# estimate variable importance

importance <- varImp(model, scale=FALSE)

# summarize importance

print(importance)

# plot importance

plot(importance)

##################################################################################

############################## Wrapper Methods ################################

# Backwards selection (a.k.a. recursive feature elimination).

# Genetic algorithms, and

# Simulated annealing.

#################################################

## Feature Selection using Recursive Feature Elimination or RFE

# ensure the results are repeatable

set.seed(7)

# load the library

# library(mlbench)

# library(caret)

# load the data

# data(PimaIndiansDiabetes)

# There are a number of pre-defined sets of functions for several models, including:

# linear regression (in the object lmFuncs),

# random forests (rfFuncs),

# naive Bayes (nbFuncs),

# bagged trees (treebagFuncs)

# and functions that can be used with caret’s train function (caretFuncs).

# define the control using a random forest selection function

control <- rfeControl(functions=rfFuncs, method="repeatedcv", number=10)

# run the RFE algorithm

results <- rfe(PimaIndiansDiabetes[,1:8], PimaIndiansDiabetes[,9], sizes=c(1:8), rfeControl=control)

# summarize the results

print(results)

# list the chosen features

predictors(results)

# plot the results

plot(results, type=c("g", "o"))

#################################################

##################################################################################