Machine learning

Machine learning

Arthur Lee Samuel

- Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed (Samuel 1959).

- The name machine learning was coined in 1959 by Arthur Samuel, evolved from the study of pattern recognition and computational learning theory in artificial intelligence (Samuel 1988).

- Task: Supervised learning (semi-supervised learning, active learning, reinforcement learning) and unsupervised learning.

Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. 3 (3). doi:10.1147/rd.33.0210.
Samuel, Arthur L. (1988). Computer Games I. Springer, New York, NY. pp. 335–365. doi:10.1007/978-1-4613-8716-9_14. ISBN 9781461387183.

Publications

- Danilo Bzdok, Naomi S. Altman, Martin K. 2018. Statistics versus machine learning | **Nature Methods** 15:233–234 .
- Keith T. Butler, Daniel W. Davies, Hugh Cartwright, O. Isayev, A. Walsh, 2018. Machine learning for molecular and materials science, **Nature** 559:547–555 .
- Yann LeCun, Yoshua Bengio & Geoffrey Hinton. 2015. Deep learning. **Nature** 521:436–444
- Xing Lin*, Yair Rivenson*, Nezih T. Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi. 2018. All-optical machine learning using diffractive deep neural networks. **Science** 361:1004-1008.
- Erik Brynjolfsson,Tom Mitchell. 2017. What can machine learning do? Workforce implications. **Science** 358:1530-1534
- David Silver, et al. 2017. Mastering the game of Go without human knowledge. **Nature** 550:354–359

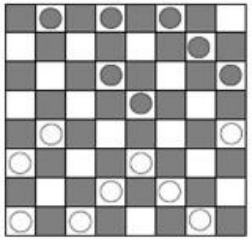
Machine learning in the context of statistics and computer sciences

- **Statistical Learning**

  Statistics has applied in computer vision, robotics, speech recognition, and many more. When statistics and computer science are combined together that machine learning emerges as a new discipline.

- **Machine Learning (ML)**

  The Samuel Checkers-Playing Program, which is known to be the first computer program that could learn, was developed in 1959 by Arthur Lee Samuel.

  Machine learning: automatically improve the model with new data

  Statistics: use the model to interpret data with a certain reliability

- **Artificial Intelligence (AI)**

  The science and engineering of making intelligent machines (John McCarthy).

  AI goes further than ML, including linguistics, philosophy, psychology, neuroscience, mathematics. Machine learning is often considered to be a subset of AI.

- **Data Mining**

  A parallel concept as ML, e.g. in Knowledge Discovery and Data Mining (KDD), a premier forum for data mining.
  The book **"Data Mining: Practical machine learning tools and techniques with Java"** being originally named just **"Practical Machine Learning"**, and the term data mining was only added for marketing reasons.
  ML is the core of data mining.

- **Data Science**

  Emphasize the importance of data more than the algorithms of learning.

Machine learning algorithms

- Artificial neural networks (ANN)
- Classification and regression tree (CART)
- Random forest
- Generalized boosting models (GBM)
- Genetic algorithm for rule set production (GARP)
- Maximum entropy method (Maxent)
- Support vector machine (SVM)
- …

Artificial Neural Networks (ANN)

- Single Layer Perceptrons (SLP)
- Multilayer perceptrons (MLP)
- Recurrent neural network (RNN)
- Convolutional neural network (CNN)
- Backpropagation (BP)
- Boltzmann machines
- …

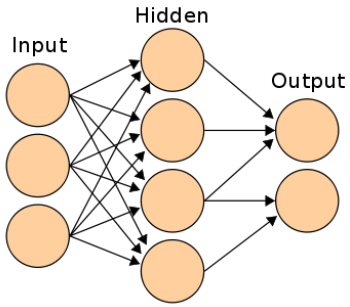
R packages

Package	**Available architectures of neural networks**
MXNet	Feed-forward neural network, convolutional neural network (CNN)
darch	Restricted Boltzmann machine, deep belief network
deepnet	Feed-forward neural network, restricted Boltzmann machine, deep belief network, stacked autoencoders
H2O	Feed-forward neural network, deep autoencoders
deepr	Simplify some functions from H2O and deepnet packages
neuralnet	multi-layer perceptrons, backpropagation
…	

Artificial Neural Networks (ANN)

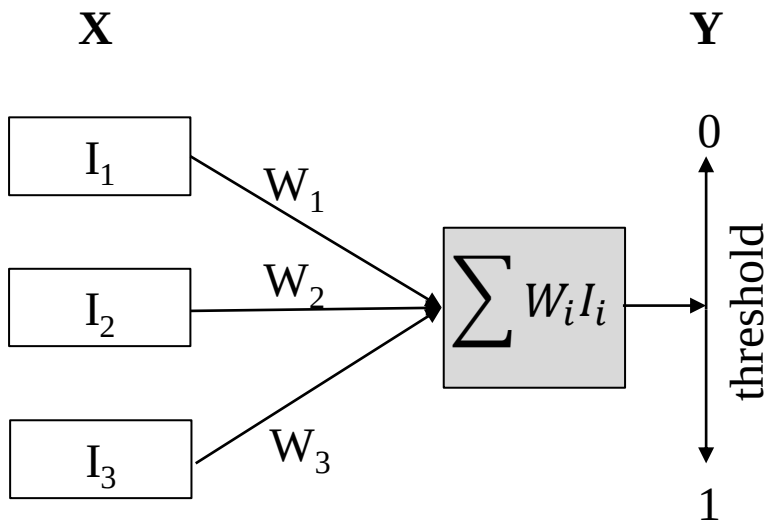
- A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation.
- In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.
- Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data (Ripley 1996) .

Ripley, B. D. 1996. Pattern Recognition and Neural Networks. Cambridge.



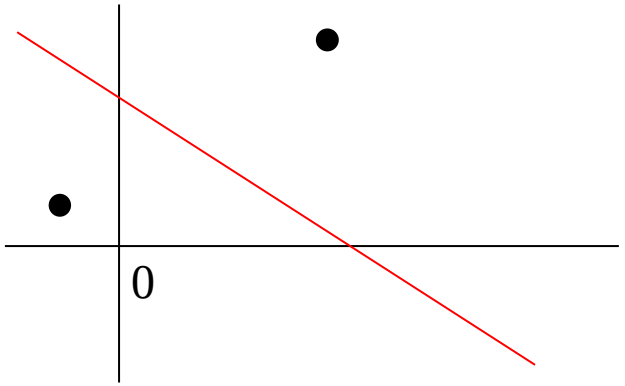
Single Layer Perceptrons (SLP)

The simplest neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights.



The perceptron is simply separating the input into 2 categories. In the 2-dimensional case, when the two inputs and the threshold are given, the following line separates the space of 0 and 1 for Y.

$w_1 I_1 + w_2 I_2 = \text{threshold}$



Perceptron Learning Rule

No need to design these networks. SLP could have *learnt* those weights and thresholds, by showing it the correct answers as it wants to generate.

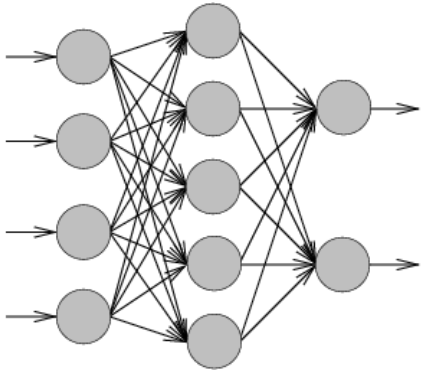
Let input x = (I_1, I_2, .., I_n).
And let output y = 0 or 1.

Algorithm repeats:

1. Given input x = (I_1, I_2, .., I_n). Perceptron produces output y. We are told correct output **O**.
2. If we had wrong answer, change w_i's and t, otherwise do nothing.

Input x	Predicted y	Observed y (**O**)	What to do with w's	What to do with t
(0,1,0,0)	0	1	increase	reduce
(1,0,0,0)	0	0	no change	no change
(0,1,1,1)	1	0	reduce	increase
(1,0,1,0)	1	0	reduce	increase
(1,1,1,1)	0	1	increase	reduce
(0,1,0,0)	1	1	no change	no change
....	....	....	....	....

Multilayer perceptron (MLP)



A **multilayer perceptron** (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.

Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training.

Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Activation functions and loss function

Activation functions

Sigmoid Function: $\phi(x) = \frac{1}{1+e^{-x}}$

Hyperbolic Tangent Function (tanh): $\phi(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$

Rectifier Function: $\phi(x) = \max(x, 0)$

Loss functions

Mean absolute error

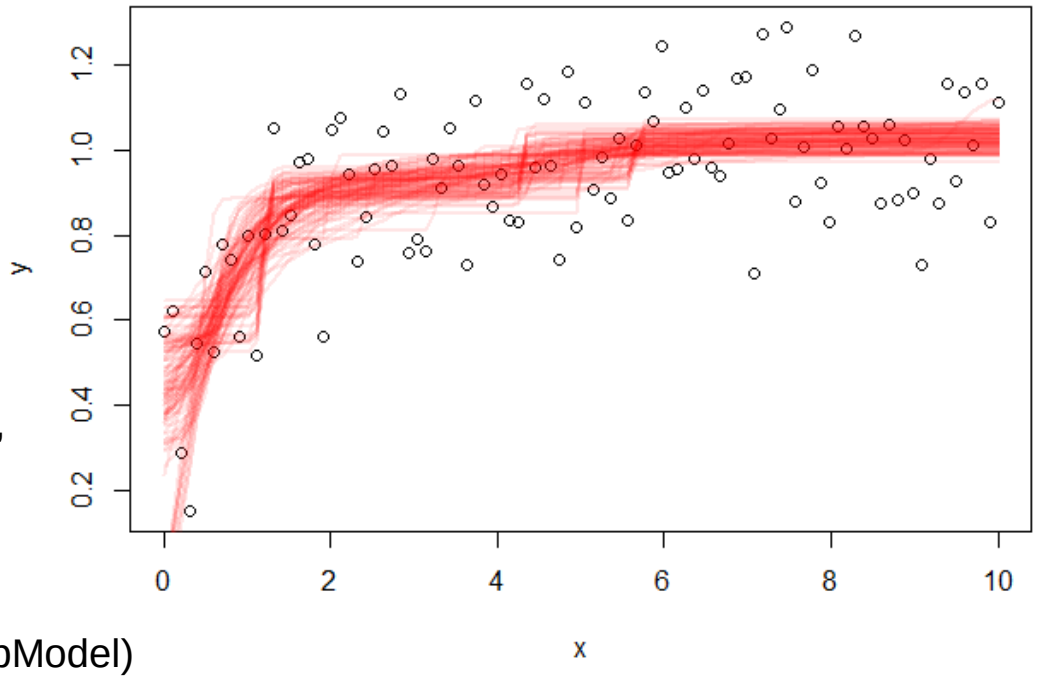
Mean square error

Cross-entropy loss $I(p,q) = -\sum p(x) log\ q(x)$

Where **x** represents the predicted results by ML algorithm, **p(x)** is the probability distribution of "true" label from training samples and **q(x)** depicts the estimation of the ML algorithm.

Multilayer perceptron (MLP)

```
require(monmlp)
# MLP Models
# Generating New Data
x <- as.matrix(seq(0, 10, length = 100))
y <- logistic(x) + rnorm(100, sd = 0.15)

# Fitting Model, require(monmlp)
mlpModel <- monmlp.fit(x = x, y = y, hidden1 = 3,
                       monotone = 1,
                       n.ensemble = 100,
                       bag = TRUE)
mlpModel <- monmlp.predict(x = x, weights = mlpModel)

# Plotting predicted value over actual values
plot(x, y)
for(i in 1:100){ lines(x, attr(mlpModel, "ensemble")[[i]], col = adjustcolor( "red", alpha.f = 0.1), lwd = 2) }

# one prediction
mlpModel <- monmlp.fit(x = x, y = y, hidden1 = 3, monotone = 1,  n.ensemble = 1, bag = TRUE)
mlpModel <- monmlp.predict(x = x, weights = mlpModel)
plot(x, mlpModel)
```



How many hidden layers to use and how many neurons are in it?

We typically choose to use hidden layers only in the event that data is not linearly separable.

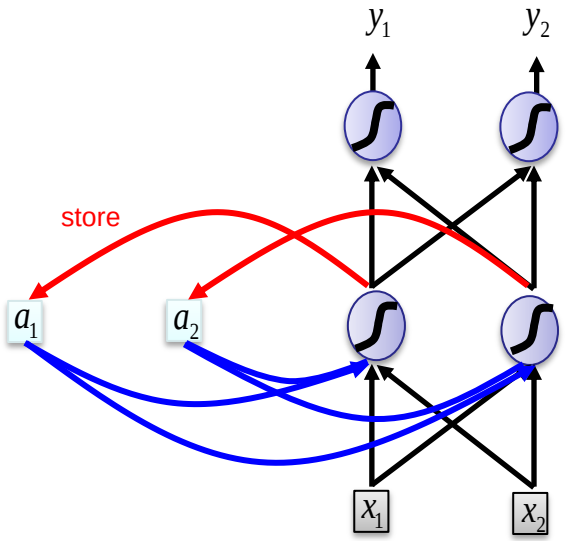
Whenever step, heaviside, or threshold activation functions are utilized, it is generally advisable to use two hidden layers. With respect to using more than one hidden layer, it's largely unnecessary because the increase in performance from using two or more layers is negligible in most situations.

Often, when adding a layer to a neural network model, this will be simple as editing an argument in a function or, in the case of some deep learning frameworks such as mxnet, passing values from a previous layer through an entirely new function.

With respect to how many neurons should be within a given hidden layer, this must be tested for with the objective of minimizing the training error. Some suggest that it has to be between the input and output layer size, never more than twice the number of inputs, capturing .70-.90 variance of the initial data set—or to use the following formula: # *HiddenUnits* = (# *inputs* +# *outputs*)* 2/3 (Beysolow 2017).

Beysolow, Taweh II.2017. Introduction to Deep Learning Using R. Apress Media. San Francisco, California, USA

Recurrent Neural Network (RNN)



Connections between nodes form a directed graph along a temporal sequence -> exhibit temporal dynamic behavior.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

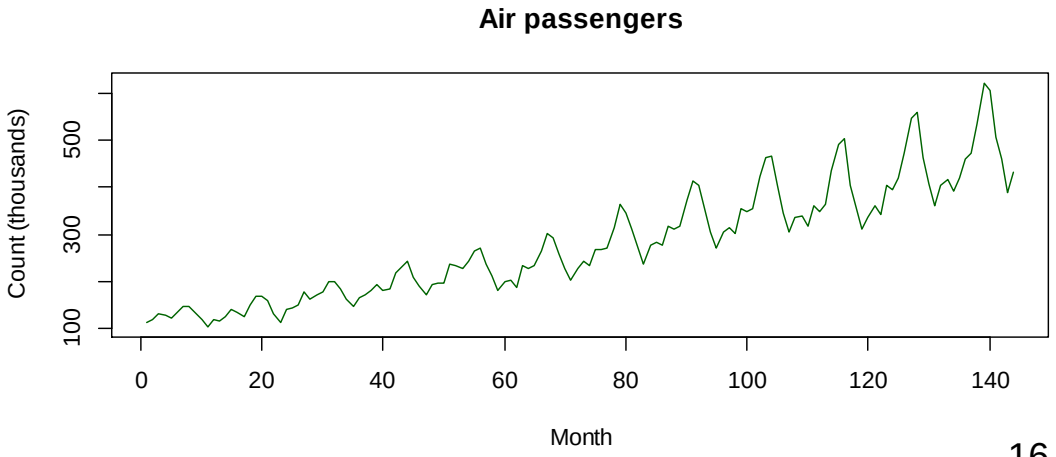
Applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

Recurrent Neural Networks (RNNs) for time series patterns

```
require(rnn)
# Function for creating T and T+1 dataset
dataset <- function(data){
  x <- y <- c()
  for (i in 1:(nrow(data)-2)){
    x <- append(x, data[i, 2])
    y <- append(y, data[i+1, 2])
  }
  output <- cbind(x,y)
  return(output[1:nrow(output)-1,])
}
```

```
# The classic Box & Jenkins airline data. Monthly totals of
# international airline passengers, 1949 to 1960
tt = AirPassengers # time series data
dmn <- dimnames(.preformat.ts(tt))
data = as.data.frame(t(matrix(tt, 12, dimnames = dmn)))
data = t(data); data = as.vector(data)
data = cbind(Month=1:length(data), data)

# Plotting the Sequence
plot(data[,2], main = "Air passengers", xlab = "Month",
     ylab = "Count (thousands)", lwd = 1.5,
     col = "darkgreen", type = "l")
```

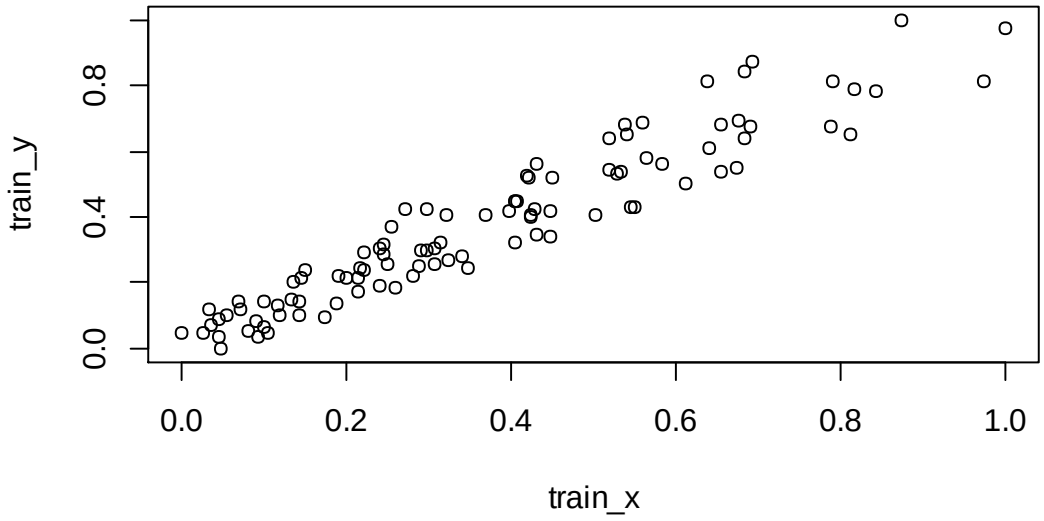
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	196	236	235	229	243	264	272	237	211	180	201
1954	204	188	235	227	234	264	302	293	259	229	203	229
1955	242	233	267	269	270	315	364	347	312	274	237	278
1956	284	277	317	313	318	374	413	405	355	306	271	306
1957	315	301	356	348	355	422	465	467	404	347	305	336
1958	340	318	362	348	363	435	491	505	404	359	310	337
1959	360	342	406	396	420	472	548	559	463	407	362	405
1960	417	391	419	461	472	535	622	606	508	461	390	432



Recurrent Neural Networks (RNNs) for time series patterns

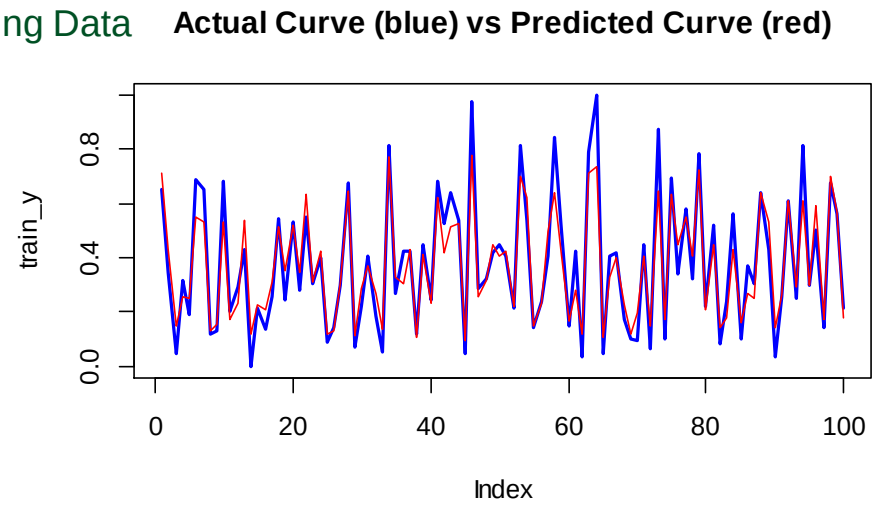
```
#Creating Test and Training Sets
newData <- dataset(data = data)
head(newData)

#Creating Test and Train Data
rows <- sample(1:100, 100)
trainingData <- newData[rows, ] # 1:100 in 144
testData <- newData[-rows, ] #101:144 in 144

#Max-Min Scaling
x <- trainingData[,1]
y <- trainingData[,2]
train_x <- (x - min(x))/(max(x)-min(x))
train_y <- (y - min(y))/(max(y)-min(y))

plot(train_x, train_y) #autocorrelation at lag 1
```

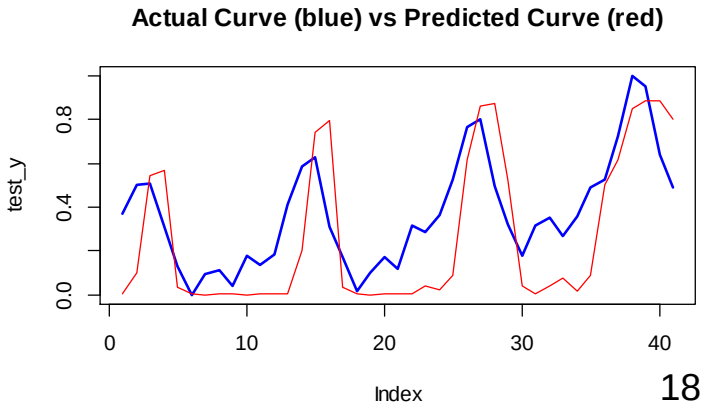
x	y
112	118
118	132
132	129
129	121
121	135
135	148



Recurrent Neural Networks (RNNs) for time series patterns

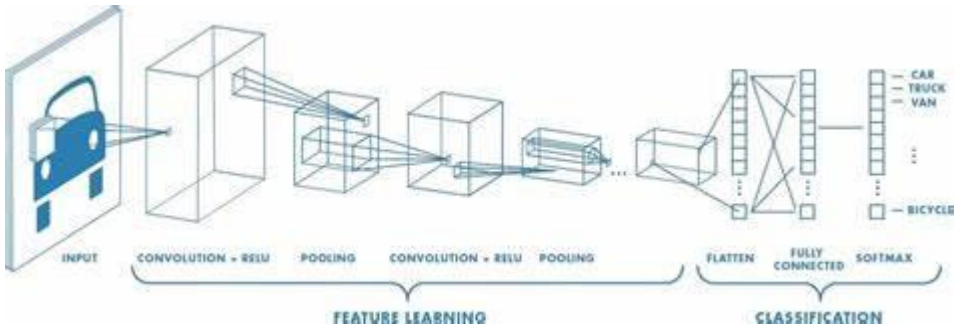
```
#RNN Model
RNN <- trainr(Y = as.matrix(train_y), X = as.matrix(train_x),
              learningrate = 0.05, momentum = 0.1,
              network_type = "rnn", numepochs = 300, hidden_dim = 3)
y_h <- predictr(RNN, as.matrix(train_x))

#Comparing Plots of Predicted Curve vs Actual Curve: Training Data
plot(train_y, col = “blue”, type = “l”, lwd = 2,
      main = "Actual Curve (blue) vs Predicted Curve (red)")
lines(y_h, type = "l", col = "red", lwd = 1)
cat("Train MSE: ", mse(y_h, train_y)) # 0.00677879

#Test Data
testData <- newData[-rows, ]
x <- testData[,1]; y <- testData[,2]
test_x <- (x - min(x))/(max(x)-min(x))
test_y <- (y - min(y))/(max(y)-min(y))
y_h2 <- predictr(RNN, as.matrix(test_x))
#Comparing Plots of Predicted Curve vs Actual Curve: Test Data
plot(test_y, col = "blue", type = "l", lwd = 2,
      main = "Actual Curve (blue) vs Predicted Curve (red)")
lines(y_h2, type = "l", col = "red", lwd = 1)
cat("Test MSE: ", mse(y_h2, test_y)) #0.060894
```





Convolutional Neural Networks (CNN)

- Unlike a fully connected neural network, in a Convolutional Neural Network (CNN) the neurons in one layer don't connect to all the neurons in the next layer.
- CNN uses a three-dimensional structure, where each set of neurons analyzes a specific region or "feature" of the image. CNNs filters connections by proximity (pixels are only analyzed in relation to pixels nearby), making the training process computationally achievable.
- The final output is a vector of probabilities, which predicts, for each feature in the image, how likely it is to belong to a class or category.



https://miro.medium.com/max/1200/1*XbuW8WuRrAY5pC4t-9DZAQ.jpeg

Character-level language modeling using CNN (MXNet tutorial)

```
cran <- getOption("repos")
cran["dmlc"] <- "https://s3-us-west-2.amazonaws.com/apache-mxnet/R/CRAN/"
options(repos = cran)
install.packages("mxnet",dependencies = T)
library(mxnet)
library("readr")
library("stringr")
library("stringi")

download.data <- function(data_dir) {
  dir.create(data_dir, showWarnings = FALSE)
  if (!file.exists(paste0(data_dir,'input.txt'))) {
    download.file(url='https://raw.githubusercontent.com/dmlc/web-data/master/mxnet/tinyshakespeare/input.txt',
                  destfile=paste0(data_dir,'input.txt'), method='wget')
  }
}

make_data <- function(path, seq.len = 32, dic=NULL) {
  text_vec <- read_file(file = path)
  text_vec <- stri_enc_toascii(str = text_vec)
  text_vec <- str_replace_all(string = text_vec, pattern = "[^[:print:]]", replacement = "")
  text_vec <- strsplit(text_vec, '') %>% unlist

  if (is.null(dic)) {
    char_keep <- sort(unique(text_vec))
  } else char_keep <- names(dic)[!dic == 0]

  # Remove terms not part of dictionary
  text_vec <- text_vec[text_vec %in% char_keep]

  # Build dictionary
  dic <- 1:length(char_keep)
  names(dic) <- char_keep

  # reverse dictionary
  rev_dic <- names(dic)
  names(rev_dic) <- dic

  # Adjust by -1 to have a 1-lag for labels
  num.seq <- (length(text_vec) - 1) %/% seq.len

  features <- dic[text_vec[1:(seq.len * num.seq)]]
  labels <- dic[text_vec[1:(seq.len*num.seq) + 1]]

  features_array <- array(features, dim = c(seq.len, num.seq))
  labels_array <- array(labels, dim = c(seq.len, num.seq))

  return (list(features_array = features_array, labels_array = labels_array, dic = dic, rev_dic = rev_dic))
}


seq.len <- 100
data_prep <- make_data(path = "input.txt", seq.len = seq.len, dic=NULL)

str(data_prep)


X <- data_prep$features_array; head(X[1:3,1:3])
Y <- data_prep$labels_array; head(Y[1:3,1:3])
dic <- data_prep$dic
rev_dic <- data_prep$rev_dic
vocab <- length(dic)

samples <- tail(dim(X), 1)
train.val.fraction <- 0.9

X.train.data <- X[, 1:as.integer(samples * train.val.fraction)]
X.val.data <- X[, -(1:as.integer(samples * train.val.fraction))]
X.train.label <- Y[, 1:as.integer(samples * train.val.fraction)]
X.val.label <- Y[, -(1:as.integer(samples * train.val.fraction))]
train_buckets <- list("100" = list(data = X.train.data, label = X.train.label))
eval_buckets <- list("100" = list(data = X.val.data, label = X.val.label))
train_buckets <- list(buckets = train_buckets, dic = dic, rev_dic = rev_dic)
eval_buckets <- list(buckets = eval_buckets, dic = dic, rev_dic = rev_dic)

vocab <- length(eval_buckets$dic)
batch.size <- 32
train.data <- mx.io.bucket.iter(buckets = train_buckets$buckets, batch.size = batch.size,
                                data.mask.element = 0, shuffle = TRUE)
eval.data <- mx.io.bucket.iter(buckets = eval_buckets$buckets, batch.size = batch.size,
                               data.mask.element = 0, shuffle = FALSE)
# the model
rnn_graph_one_one <- rnn.graph(num_rnn_layer = 3,
                               num_hidden = 96,
                               input_size = vocab,
                               num_embed = 64,
                               num_decode = vocab,
                               dropout = 0.2,
                               ignore_label = 0,
                               cell_type = "lstm",
                               masking = F,
                               output_last_state = T,
                               loss_output = "softmax",
                               config = "one-to-one")


graph.viz(rnn_graph_one_one, type = "graph", direction = "LR",
          graph.height.px = 180, shape=c(100, 64))
devices <- mx.cpu()
initializer <- mx.init.Xavier(rnd_type = "gaussian", factor_type = "avg", magnitude = 3)
optimizer <- mx.opt.create("adadelta", rho = 0.9, eps = 1e-5, wd = 1e-8,
                           clip_gradient = 5, rescale.grad = 1/batch.size)
logger <- mx.metric.logger()
epoch.end.callback <- mx.callback.log.train.metric(period = 1, logger = logger)
batch.end.callback <- mx.callback.log.train.metric(period = 50)

mx.metric.custom_nd <- function(name, feval) {
  init <- function() {
    c(0, 0)
  }
  update <- function(label, pred, state) {
    m <- feval(label, pred)
    state <- c(state[[1]] + 1, state[[2]] + m)
    return(state)
  }
  get <- function(state) {
    list(name=name, value = (state[[2]] / state[[1]]))
  }
  ret <- (list(init = init, update = update, get = get))
  class(ret) <- "mx.metric"
  return(ret)
}

mx.metric.Perplexity <- mx.metric.custom_nd("Perplexity", function(label, pred) {
  label <- mx.nd.reshape(label, shape = -1)
  label_probs <- as.array(mx.nd.choose.element.0index(pred, label))
  batch <- length(label_probs)
  NLL <- -sum(log(pmax(1e-15, as.array(label_probs)))) / batch
  Perplexity <- exp(NLL)
  return(Perplexity)
})

model <- mx.model.buckets(symbol = rnn_graph_one_one,
                          train.data = train.data, eval.data = eval.data,
                          num.round = 2, ctx = devices, verbose = TRUE,  #was 20
                          metric = mx.metric.Perplexity,
                          initializer = initializer, optimizer = optimizer,
                          batch.end.callback = NULL,
                          epoch.end.callback = epoch.end.callback)

mx.model.save(model, prefix = "one_to_one_seq_model", iteration = 2) #was 20

# Inference on the Model
# use the saved model to do inference and sample text character by character
# that will look like the original training data
set.seed(0)
model <- mx.model.load(prefix = "one_to_one_seq_model", iteration = 2) #was 20

internals <- model$symbol$get.internals()
sym_state <- internals$get.output(which(internals$outputs %in% "RNN_state"))
sym_state_cell <- internals$get.output(which(internals$outputs %in% "RNN_state_cell"))
sym_output <- internals$get.output(which(internals$outputs %in% "loss_output"))
symbol <- mx.symbol.Group(sym_output, sym_state, sym_state_cell)

infer_raw <- c("Thou ")
infer_split <- dic[strsplit(infer_raw, '') %>% unlist]
infer_length <- length(infer_split)

infer.data <- mx.io.arrayiter(data = matrix(infer_split), label = matrix(infer_split),
                              batch.size = 1, shuffle = FALSE)

infer <- mx.infer.rnn.one(infer.data = infer.data,
                          symbol = symbol,
                          arg.params = model$arg.params,
                          aux.params = model$aux.params,
                          input.params = NULL,
                          ctx = devices)

pred_prob <- as.numeric(as.array(mx.nd.slice.axis(
  infer$loss_output, axis = 0, begin = infer_length-1, end = infer_length)))
pred <- sample(length(pred_prob), prob = pred_prob, size = 1) - 1
predict <- c(predict, pred)

for (i in 1:200) {

  infer.data <- mx.io.arrayiter(data = as.matrix(pred), label = as.matrix(pred),
                                batch.size = 1, shuffle = FALSE)

  infer <- mx.infer.rnn.one(infer.data = infer.data,
                            symbol = symbol,
                            arg.params = model$arg.params,
                            aux.params = model$aux.params,
                            input.params = list(rnn.state = infer[[2]],
                                                rnn.state.cell = infer[[3]]),
                            ctx = devices)

  pred_prob <- as.numeric(as.array(infer$loss_output))
  pred <- sample(length(pred_prob), prob = pred_prob, size = 1, replace = T) - 1
  predict <- c(predict, pred)
}

predict_txt <- paste0(rev_dic[as.character(predict)], collapse = "")
predict_txt_tot <- paste0(infer_raw, predict_txt, collapse = "")
print(predict_txt_tot)
```

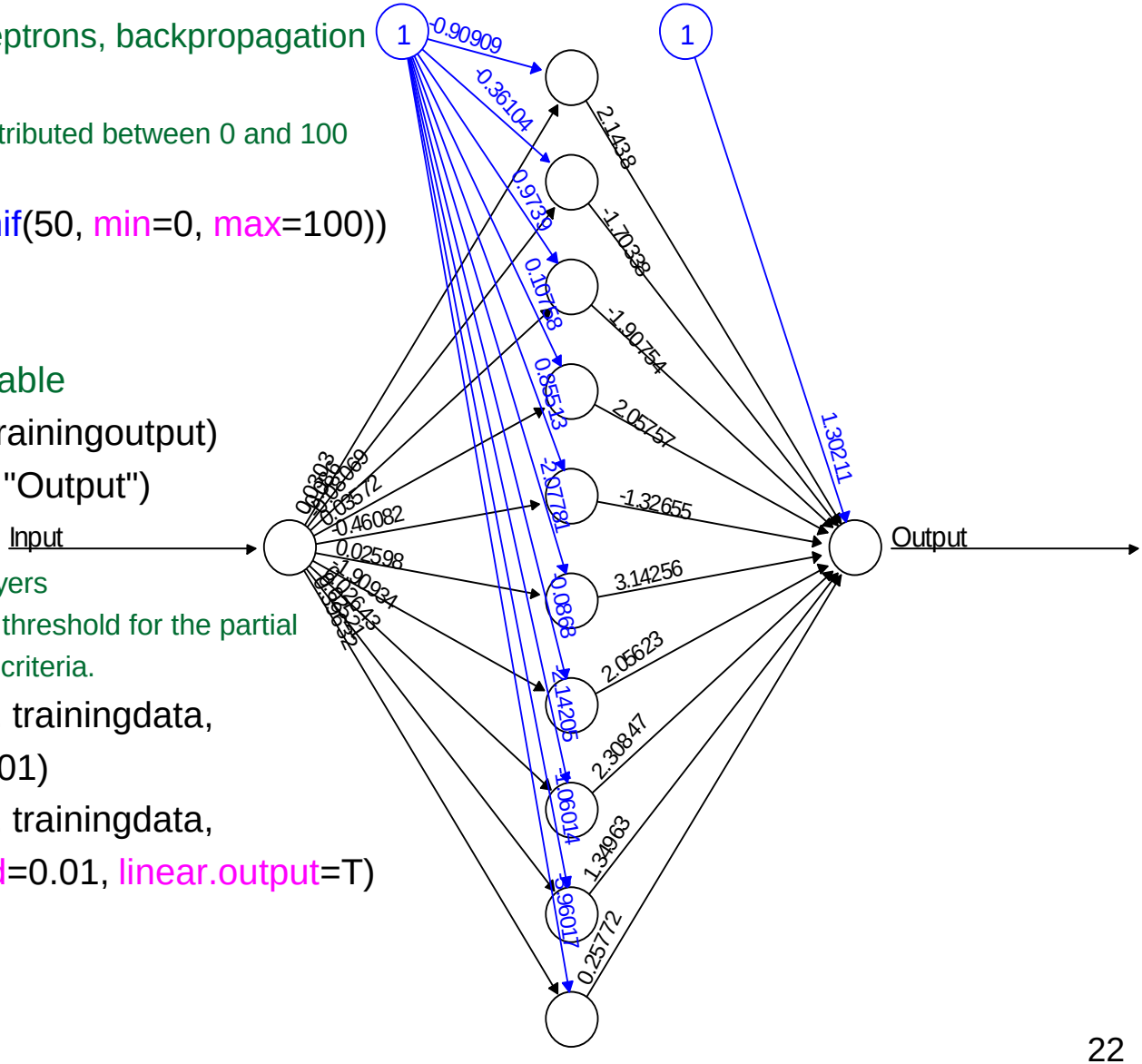
Character-level language modeling using CNN (MXNet tutorial)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
'	-		!	$	&	,	.	:	;	?	3	a	A	b	B	c	C	d	D	e	E	f	F	g	G	h	H
29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56
i	l	j	J	k	K	l	L	m	M	n	N	o	O	p	P	q	Q	r	R	s	S	t	T	u	U	v	V
57	58	59	60	61	62	63	64																				
w	W	x	X	y	Y	z	Z																				

```
"Thou NAnot gosiness stillBeny me forgeallomas
wealishou did.GLOUCESTER:Are offectiol; night
will me nexs it comture,Yow he see id shall
most o'er Cormine on love.DUKE VINCENTIO:If do
chilge ustere.KING RICHAR"
```

Artificial Neural Networks (ANN)

```
library("neuralnet") #multi-layer perceptrons, backpropagation

#Generate 50 random numbers uniformly distributed between 0 and 100
# and store them as a dataframe
traininginput   <-  as.data.frame(runif(50, min=0, max=100))
trainingoutput  <-  sqrt(traininginput)

#Column bind the data into one variable
trainingdata <- cbind(traininginput, trainingoutput)
colnames(trainingdata) <- c("Input", "Output")

#Train the neural network using 10 hidden layers
#Threshold is a numeric value specifying the threshold for the partial
#derivatives of the error function as stopping criteria.
net.sqrt <- neuralnet(Output ~ Input, trainingdata,
                      hidden=10, threshold=0.01)
net.sqrt <- neuralnet(Output ~ Input, trainingdata,
                      hidden=c(5, 5), threshold=0.01, linear.output=T)
print(net.sqrt)
#Plot the neural network
plot(net.sqrt)
```



Artificial Neural Networks (ANN)

```
#Test the neural network on some training data
testdata     <- as.data.frame((1:10)^2) #Generate some squared numbers
net.results <- compute(net.sqrt, testdata) #Run them through the neural network

# Display results
cleanoutput <- cbind(testdata, sqrt(testdata),  as.data.frame(net.results$net.result))
colnames(cleanoutput) <- c("Input", "Expected_Output", "Neural_Net_Output")
print(cleanoutput)
```

	Input	Expected_Output	Neural_Net_Output
1	1	1	1.121176
2	4	2	1.996214
3	9	3	2.999129
4	16	4	4.001833
5	25	5	4.997902
6	36	6	6.001399
7	49	7	7.005971
8	64	8	7.995779
9	81	9	9.001921
10	100	10	9.977014

Machine Learning Frameworks

Machine learning framework has been defined as a tool, library, or interface that gives developers the ease of creating machine learning models.

Torch (2002)

It is a scientific machine learning framework that supports various machine learning utilities and algorithms. The salient feature of this framework is that it puts GPU first. It has community-driven packages in machine learning, computer vision, image processing, deep learning and many more. Its main is to provide high scalability, flexibility, and speed while creating machine learning models. It is definitely a framework to look for while building machine learning models.

Theano (2007)

It is built using python. It allows us to define, create and optimize mathematical calculations. Like Torch, It can also use GPU, which helps in optimization and scalability.

Scikit-Learn (2007)

It is a free machine learning library that is built on SciPy (scientific python). It is used very extensively by Python Programmers. David Cournapeau developed it. You can do feature engineering with your data (increasing the number of features), scaling, pre-processing, splitting your data into training and test subsets. It also includes many machine learning algorithms like Linear Regression, Logistic regression, K–mean algorithm, support vector machines. It is very popular because it can easily work with NumPy and SciPy.

Caffe (2013)

It is a deep learning framework that was created using speed, modularity in mind. It is mainly used with neural network problems and was founded by Berkeley Vision and Learning Center. It was created by Yangqing Jia.

Amazon Machine Learning (2015)

It is provided by Amazon. It is a service that developers can use to create models. It can be used as a visualization tool and can be used by machine learning engineers to create models without knowing every model's very detail. It can run or create all kinds of models like Binary classification, multi-class classification ensemble algorithms, regression models.

Machine Learning Frameworks

Tensor Flow (2015)

It is also an open-source library that is generally used for deep learning or machine learning algorithms using neural networks. **Google** creates it. Tensor Flow is a library for data flow programming; It uses various optimization techniques for the calculation of the mathematical expression, which is used to get the desired results. The salient feature of sci-kit learn are:

1. It works great with a mathematical expression that involves multidimensional arrays.

2. It is highly scalable across machines.

3. It works with a wide variety of data sets.

These features make it a very useful framework for deploying production models.

Keras acts as an interface for the TensorFlow library. Keras is an open-source software library that provides a Python interface for artificial neural networks.

PyTorch (2017)

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing,[6] primarily developed by **Facebook**'s AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, and Catalyst.

PyTorch provides two high-level features:

Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).

Deep neural networks built on a type-based automatic differentiation system.

Tree

classification trees or regression trees (Breiman et al. 1984)

- A tree can be built by splitting the source set into subsets based on an attribute value test.
- This process is repeated on each derived subset in a recursive manner called recursive partitioning.
- The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions.

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 1984. Classification and Regression Trees. Chapman and Hall, New York.

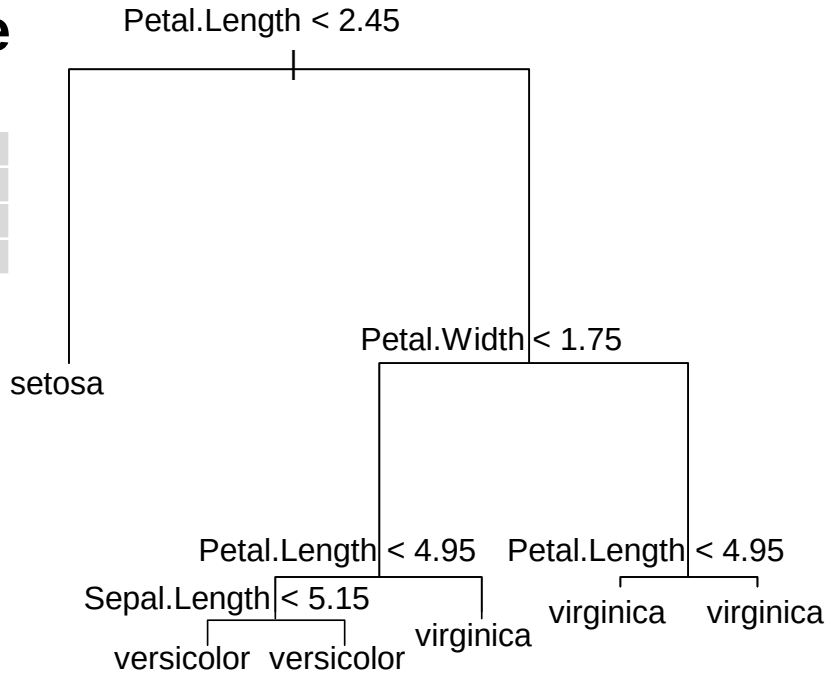
Classification tree and regression tree

```
library(tree)
# classification tree
data(iris); iris[1:3,]
```

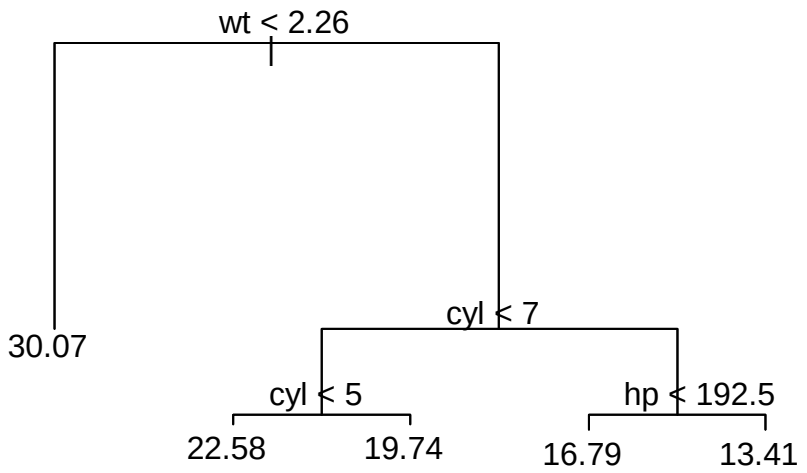
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa

```
TR1 = tree(Species ~ Sepal.Length +
           Sepal.Width + Petal.Length +
           Petal.Width, iris)
summary(TR1)
plot(TR1);  text(TR1)
predict(TR1, iris[61:66,])
```

	setosa	versicolor	virginica
61	0.00	0.80	0.20
62	0.00	1.00	0.00
63	0.00	1.00	0.00
64	0.00	1.00	0.00
65	0.00	1.00	0.00
66	0.00	1.00	0.00

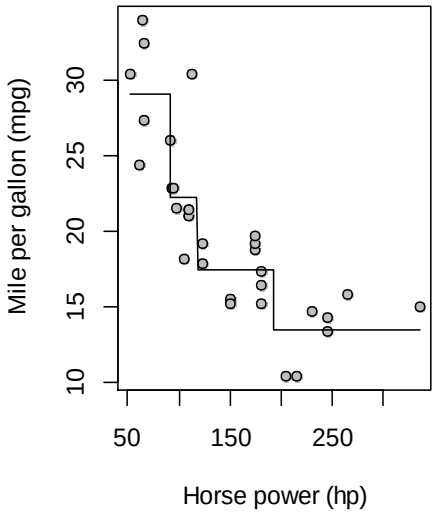



```
# regression tree
data(mtcars)
TR2 = tree(mpg ~ cyl + disp + hp +
           drat + wt + qsec, mtcars)
summary(TR2)
plot(TR2);  text(TR2)
as.data.frame(predict(TR2, mtcars[1:10,]))
```

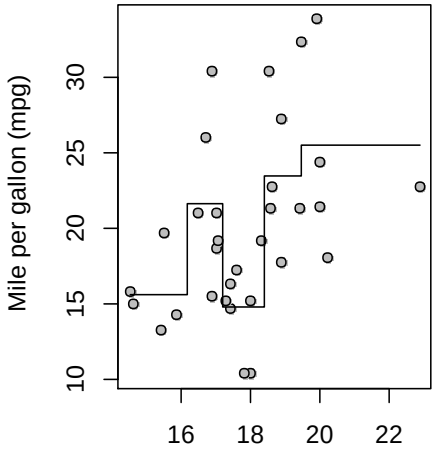


Regression tree $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_n x_{ni} + \varepsilon_i$

```
library(tree)
par(mfrow=c(2,1))
TREE <- tree(mpg ~ hp, data = mtcars)
X <- seq(min(mtcars$hp),max(mtcars$hp), length.out = 1000)
Y <- predict(TREE, list(hp=X))
plot(mtcars$hp, mtcars$mpg, pch=21, col="black",bg="gray",
     xlab = "Horse power (hp)", ylab = 'Mile per gallon (mpg)')
lines(X, Y)
```




```
TREE <- tree(mpg ~ qsec, data = mtcars)
X <- seq(min(mtcars$qsec),max(mtcars$qsec), length.out = 1000)
Y <- predict(TREE, list(qsec =X))
plot(mtcars$qsec, mtcars$mpg, pch=21, col="black",bg="gray",
     xlab = " 1/4 mile time (qsec)", ylab = 'Mile per gallon (mpg)')
lines(X, Y)
```

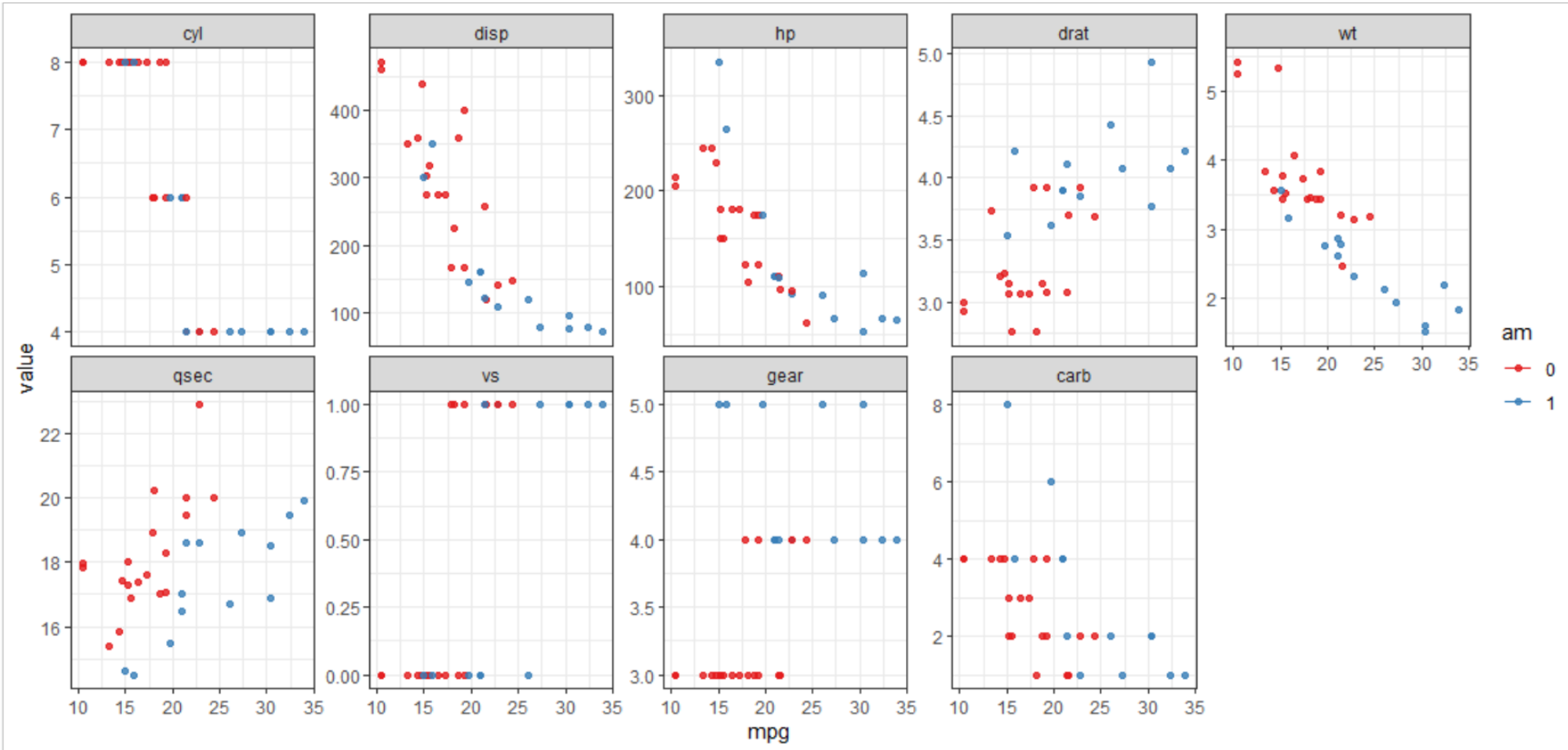


李欣海，2019。随机森林是特点鲜明的模型，不是万能的模型。应用昆虫学报 56(1): 170–179

reshape2

mpg	am	variable	value
21	1	cyl	6
21	1	cyl	6
22.8	1	cyl	4
21.4	0	cyl	6
18.7	0	cyl	8
18.1	0	cyl	6

```
library(reshape2)
mtcars$am = as.factor(mtcars$am)
dta <- melt(mtcars, id.vars=c("mpg","am"))
ggplot(dta, aes(x=mpg, y=value, color=am)) + geom_point(alpha=.8) + geom_rug(data=dta %>%
    filter(is.na(value))) + scale_color_brewer(palette="Set2") + facet_wrap(~variable, scales="free_y", ncol=5)
```



Machine Learning, 45, 5–32, 2001

Random Forests

LEO BREIMAN
Statistics Department, University of California, Berkeley, CA 94720

Editor: Robert E. Schapire

Abstract. Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost (Y. Freund & R. Schapire, *Machine Learning: Proceedings of the Thirteenth International Conference*, ***, 148–156), but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance. These ideas are also applicable to regression.

Keywords: classification, regression, ensemble

1. Random forests

1.1. Introduction

Significant improvements in classification accuracy have resulted from growing an ensemble of trees and letting them vote for the most popular class. In order to grow these ensembles, often random vectors are generated that govern the growth of each tree in the ensemble. An early example is bagging (Breiman, 1996), where to grow each tree a random selection (without replacement) is made from the examples in the training set.

Another example is random split selection (Dietterich, 1998) where at each node the split is selected at random from among the K best splits. Breiman (1999) generates new training sets by randomizing the outputs in the original training set. Another approach is to select the training set from a random set of weights on the examples in the training set. Ho (1998) has written a number of papers on "the random subspace" method which does a random selection of a subset of features to use to grow each tree.

In an important paper on written character recognition, Amit and Geman (1997) define a large number of geometric features and search over a random selection of these for the best split at each node. This latter paper has been influential in my thinking.

The common element in all of these procedures is that for the kth tree, a random vector Θ_k is generated, independent of the past random vectors $\Theta_1, \ldots, \Theta_{k-1}$ but with the same distribution; and a tree is grown using the training set and Θ_k, resulting in a classifier $h(\mathbf{x}, \Theta_k)$ where $\mathbf{x}$ is an input vector. For instance, in bagging the random vector Θ is

Random Forest

http://jfbelisle.com/2011/03/an-introduction-to-data-mining-for-marketing-and-business-intelligence/

- Random Forest is an ensemble classifier that consists of many decision trees.
- It outputs the class that is the mode of the class's output by individual trees (Breiman 2001).
- It deals with “small n large p” problems, high-order interactions, correlated predictor variables.

Breiman, L. 2001. Random forests. Machine Learning 45:5-32. (Being cited 71857 times until April 5, 2021 (Google scholar))

http://jfbelisle.com/2011/03/an-introduction-to-data-mining-for-marketing-and-business-intelligence/

History

The algorithm for inducing a random forest was developed by Leo Breiman (2001) and Adele Cutler, and "Random Forests" is their trademark.

The term came from **random decision forests** that was first proposed by Tin Kam Ho of Bell Labs in 1995.

The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by Ho (1995) and Amit and Geman (1997) in order to construct a collection of decision trees with controlled variation.

The statistical community has led to irrelevant theory, questionable conclusions…

David R. Cox

Brad Efron

NO

Emanuel Parzen

Bruce Hoadley

YES

Statistical Science
2001, Vol. 16, No. 3, 199–231

Statistical Modeling: The Two Cultures

Leo Breiman

Abstract. There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown. The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools.

http://med.stanford.edu/profiles/Trevor_Hastie/

Ensemble classifiers

Tree models are simple, often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forest (Breiman 1999): Fancier version of bagging.

http://med.stanford.edu/profiles/Trevor_Hastie/

How Random Forest Works

- At each tree split, a random sample of m features (input variables) is drawn, and only those m features are considered for splitting, and the best split is calculated only within this subset. Typically m = sqrt(p) or log(p), where p is the number of features.

- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the out-of-bag (OOB) error rate.

- Random forests tries to improve on bagging by “de-correlating” the trees. Each tree has the same expectation.

  (Trevor Hastie, p21 in *Trees, Bagging, Random Forests and Boosting*)

- No pruning step is performed so all the trees of the forest are maximal trees.

OOB

A great advantage of random forests is that the bootstrapping or subsampling for each tree yields subsets of observations, termed out-of-bag (OOB) observations, which are not included in the tree growing process.

These are used to estimate the prediction performance or variable importance.

R Packages

R packages for random forest

package	RStudio downloads in the a month in 2016
randomForest	28353
xgboost	4537
randomForestSRC	2291
ranger	1347
Rborist	284

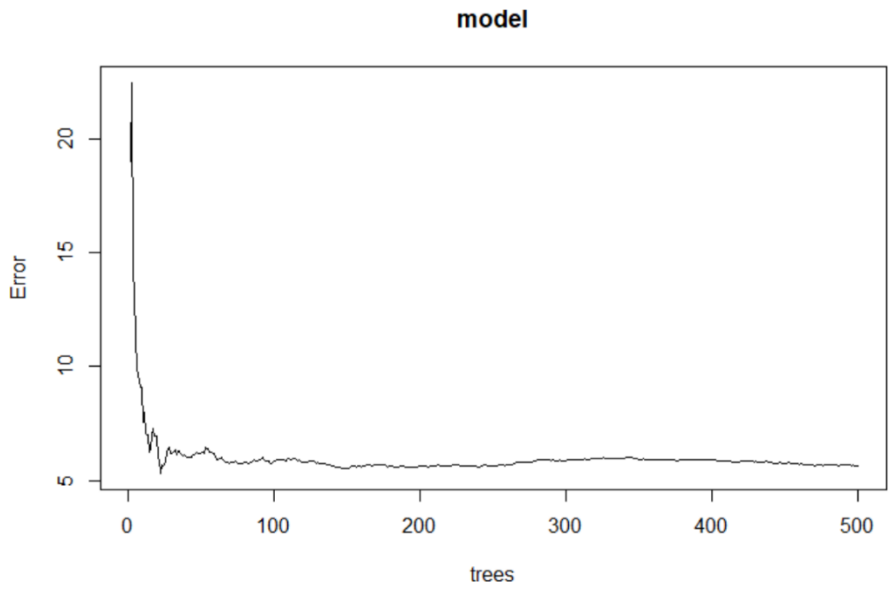
Associated R packages for random forest

package	RStudio downloads in a last month in 2016
rpart	21585
party	14338
extraTrees	1408
RRF	516
rotationForest	437
rFerns	431
obliqueRF	252
wsrf	232
randomUniformForest	189
trimTrees	134
roughrf	126
randomForestSRC(Syn)	2291

For regression

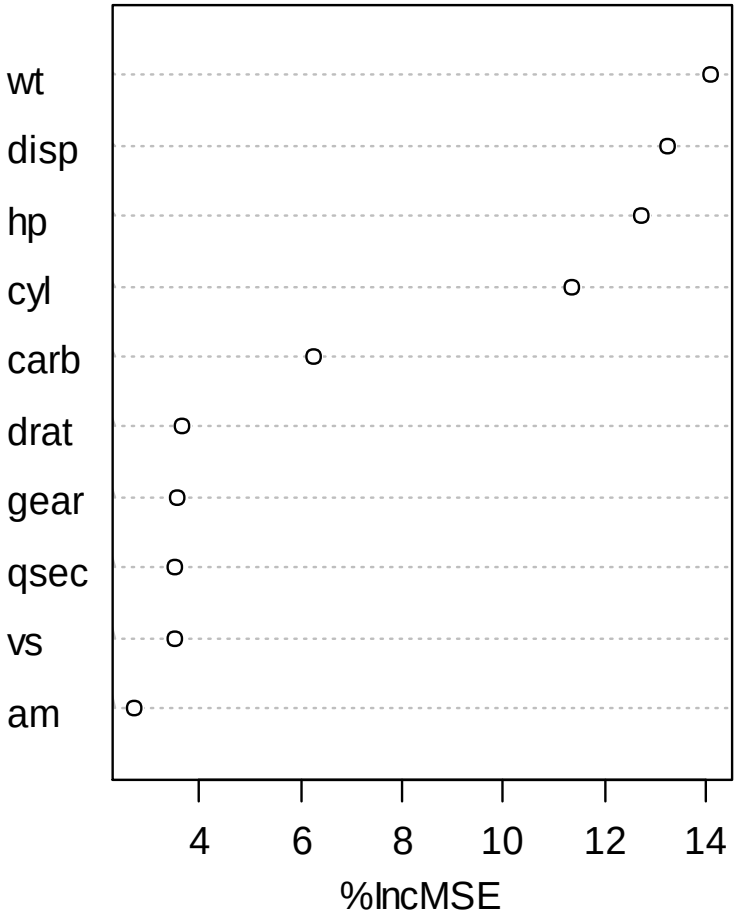
```
library(randomForest)
data(mtcars)
#replace NAs with column medians
for(i in 1:ncol(mtcars))
    mtcars[, i][is.na(mtcars[, i])] <- median(mtcars[, i],
    na.rm=TRUE)

mtcars$cyl = as.factor(mtcars$cyl)
mtcars$vs  = as.factor(mtcars$vs)
mtcars$am = as.factor(mtcars$am)
str(mtcars)

#make this example reproducible
set.seed(1)

#fit the random forest model
model <- randomForest(formula = mpg ~ .,  data =
mtcars, importance=TRUE)

#display fitted model
model
```

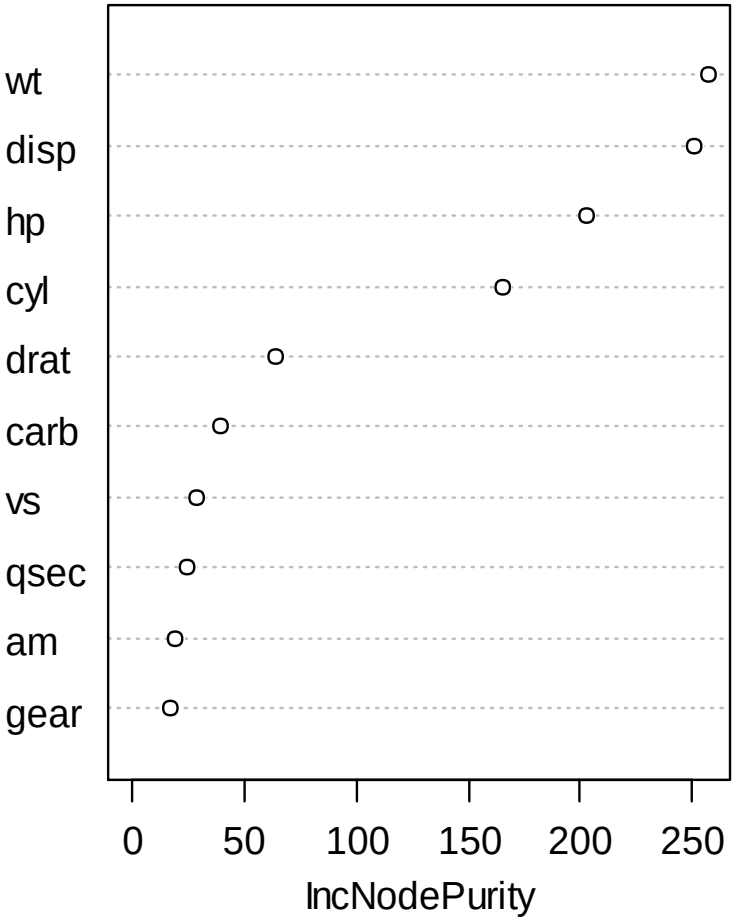
```
#find number of trees that produce lowest test MSE
which.min(model$mse)

#find RMSE of best model
sqrt(model$mse[which.min(model$mse)])

#plot the test MSE by number of trees
plot(model)
```




```
#produce variable importance plot
varImpPlot(model, main='') # importance=TRUE
```

varImpPlot(model, main='')





Negative R square

As indicated by Breiman (2001) and the R randomForest documentation the (regression only) “pseudo R-squared” is derived as:

$R^2 = 1 – (\text{mean squared error}) / \text{var}(y)$

$= 1 - SSmodel / SStotal = \text{sum}((\hat{y}-y)^2) / \text{sum}((\text{mean}(y)-y)^2) = 1 - mse / \text{var}(y)$

Which, mathematically can produce negative values. A simple interpretation of a negative R^2 (rsq), is that you are better off predicting any given sample as equal to overall estimated mean, indicating very poor model performance.

The Gini Impurity

Number of classes: C

Number of data points: N

Number of data points of class i: N_i

$$I_G = \sum_{i=1}^{C} \frac{N_i}{N}\left(1 - \frac{N_i}{N}\right)$$

true class

wrong prediction

4 **red**, 3 **green**, 3 **blue** data points

- Class probabilities:
  - red: 4/10 green: 3/10 blue: 3/10
- misclassification:
  - red: 4/10 * (3/10 + 3/10)

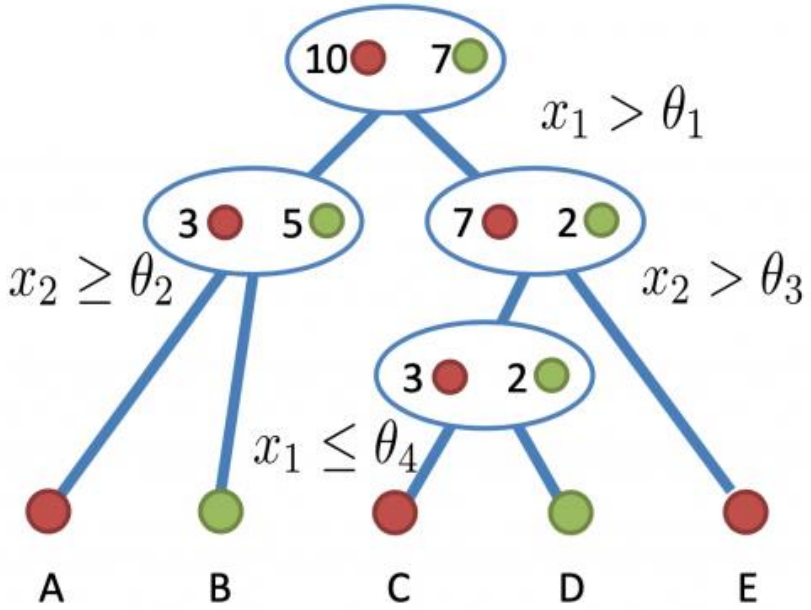
Picking red

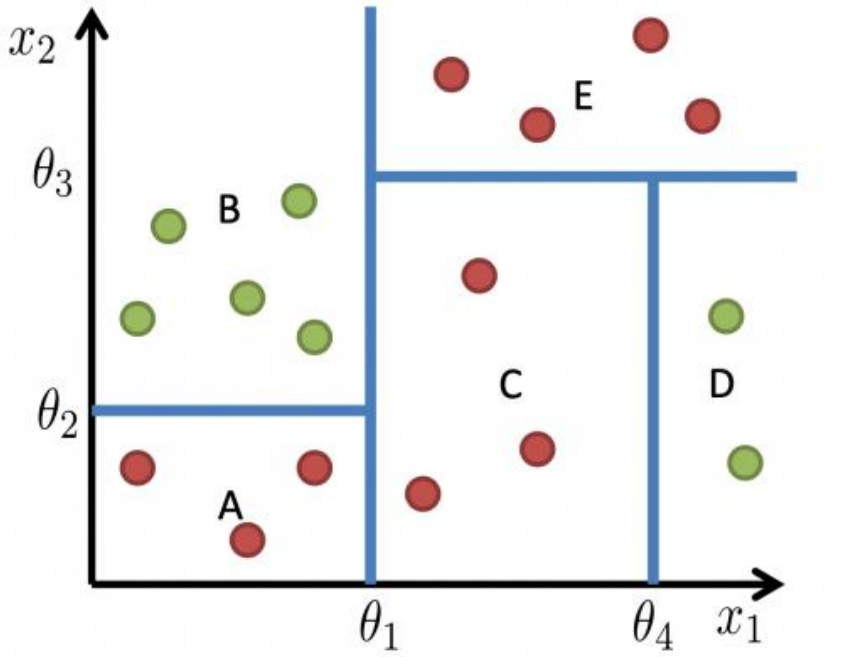
Making an error

The misclassification result (the **Gini Impurity**) for red, green, and blue is:
0.24 + 0.21 + 0.21 = 0.66

https://diegolosey.com/business-intelligence/decision-trees/

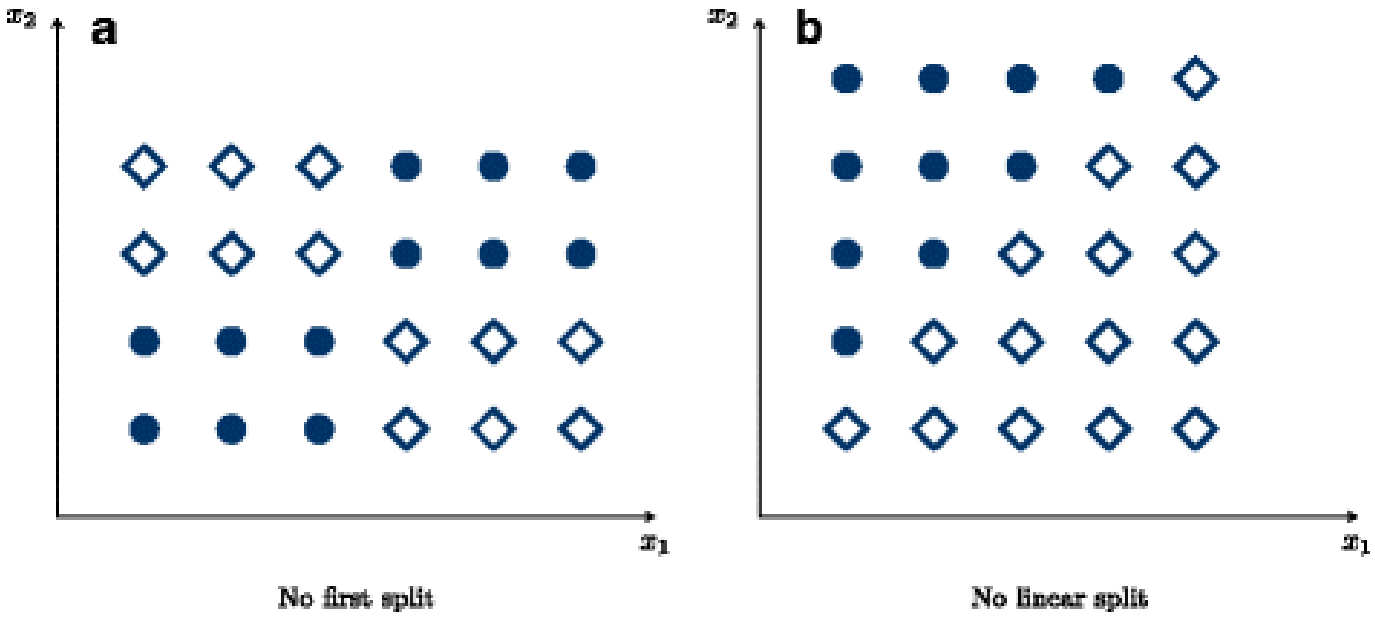
How tree algorithm works





https://diegolosey.com/business-intelligence/decision-trees/

Weakness of random forest



Problematic splits for classification trees and random forests. In (**a**) no reasonable first split on the variables x_1 or x_2 can be made. However, two subsequent splits on x_1 and x_2 split the data perfectly. In (**b**), again no reasonable first split is possible, even though the classes are linear separable. Both the variables x_1 or x_2 have to be considered simultaneously and even with several subsequent splits on x_1 and x_2, no accurate classification is possible

Wright, M.N., Ziegler, A. & König, I.R. Do little interactions get lost in dark random forests?. *BMC Bioinformatics* **17,** 145 (2016). https://doi.org/10.1186/s12859-016-0995-8

Gini importance

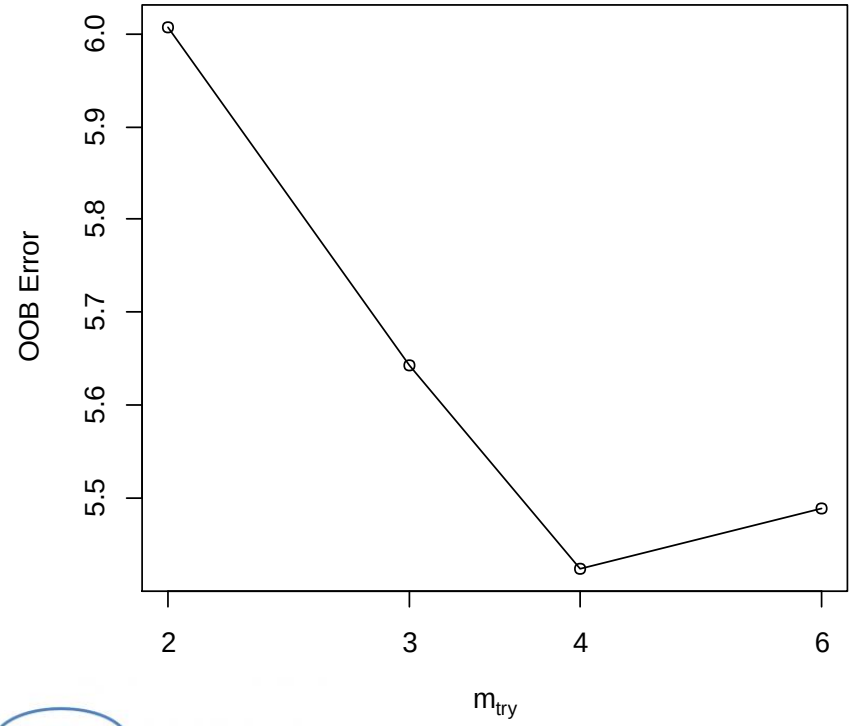
- The standard splitting rule in random forests for classification outcomes is to maximize the decrease of impurity that is introduced by a split.

- For this, the impurity is typically measured by the Gini index. Since a large Gini index suggests a large decrease of impurity, a split with large Gini index can be considered to be important for classification.

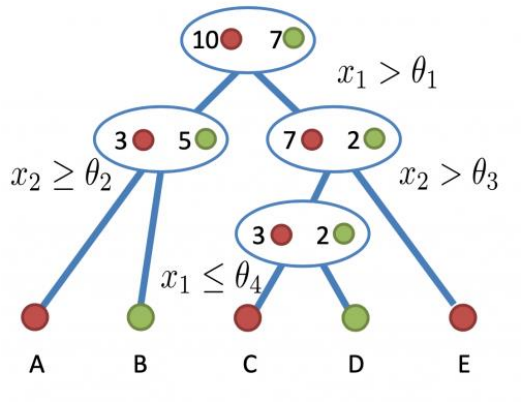
- Thus, the Gini importance for a variable x_i in a tree can be computed by summing the Gini index values of all nodes in which a split on x_i has been conducted. The average of all tree importance values for x_i is then termed Gini importance of the random forest for x_i.

Tune mtry

```
model_tuned <- tuneRF(
  x=mtcars[,-1], #define predictor variables
  y=mtcars$mpg, #define response variable
  ntreeTry=5000,
  mtryStart=3,
  stepFactor=1.5,
  improve=0.01,
  trace=T # show real-time progress
)
model_tuned
```




```
 mtry OOBError
 2      6.007131
 3      5.643909
 4      5.424304
 6      5.489048
```



MlBayesOpt::rf_opt

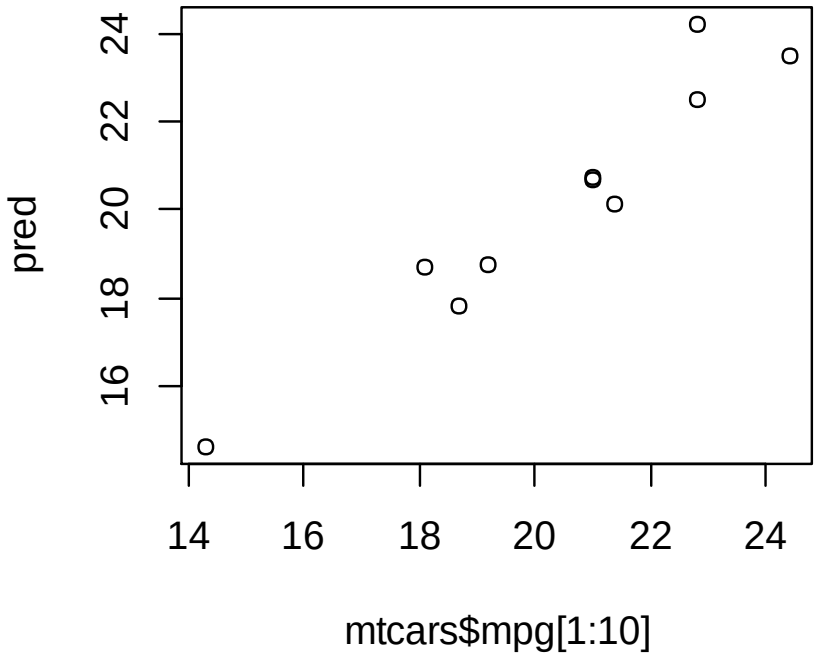
```
library(devtools)
install_github("ymattu/MlBayesOpt")
library(MlBayesOpt)

set.seed(71)
res0 <- rf_opt(train_data = iris_train,
               train_label = Species,
               test_data = iris_test,
               test_label = Species,
               mtry_range = c(1L, ncol(iris_train) - 1),
               num_tree = 10L,
               init_points = 10,
               n_iter = 1)
res0
```

```
$Best_Par
    mtry_opt    min_node_size
    1.21758     9.00000
```

```
res1 <- rf_opt(train_data = mtcars,
               train_label = mpg,
               test_data = mtcars,
               test_label = mpg,
               mtry_range = c(1L, ncol(mtcars) - 1),
               num_tree = 10L,
               init_points = 10,
               n_iter = 1)
res1
```

Prediction

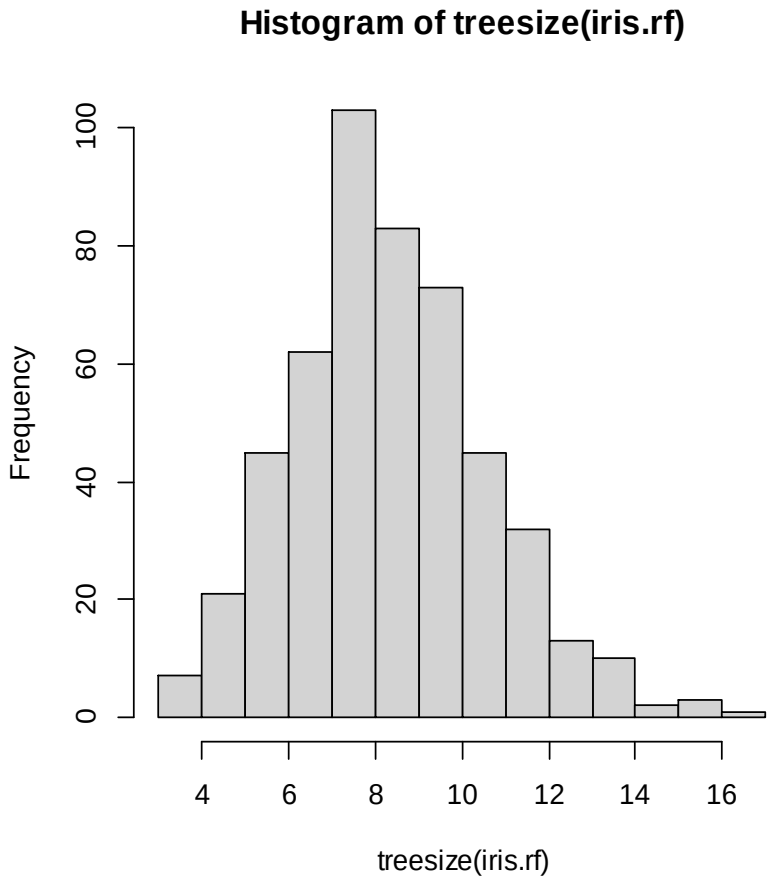
```
# use tuned parameter
model <- randomForest(formula = mpg ~ .,  data = mtcars,
             mtry = model_tuned[order(model_tuned[,2]), 1][1])

# define new observation
new <- mtcars[1:10,]

# prediction
(pred = predict(model, newdata=new))
plot(mtcars$mpg[1:10], pred)
```




```
Mazda RX4     Mazda RX4 Wag   Datsun 710    Hornet 4 Drive  Hornet Sportabout  Valiant     Duster 360    Merc 240D
20.73041      20.70481        24.23004      20.15940        17.78367           18.70764    14.61613      23.50386
Merc 230      Merc 280
22.52068      18.72840
```

Tree size

```
iris.rf <- randomForest(Species ~ ., iris)
hist(treesize(iris.rf))
```



Binary classification problem (logistic regression)

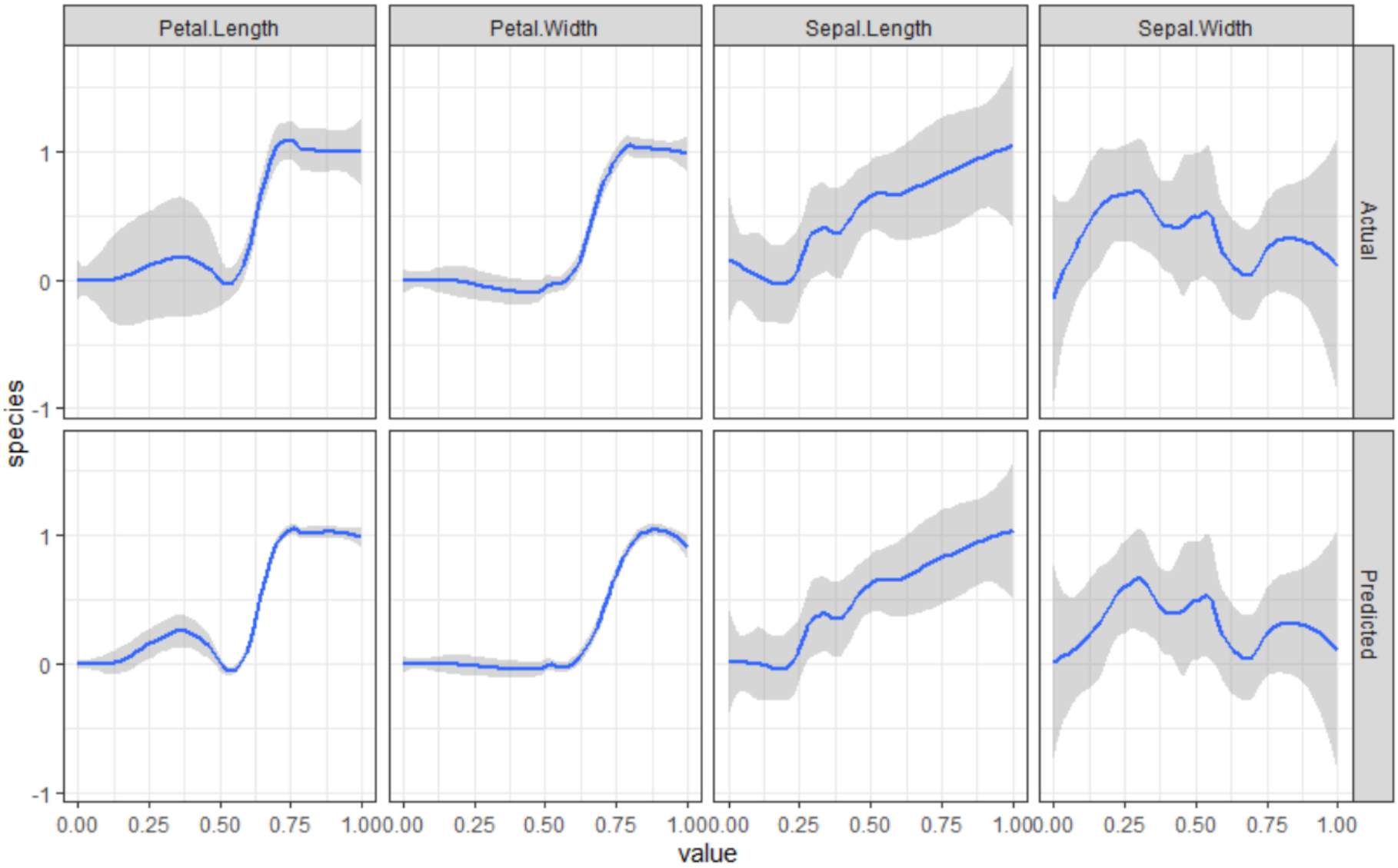
```
# Setup a binary classification problem (logistic regression)
require(randomForest)
data(iris)
set.seed(1)
dat <- iris
dat$Species <- factor(ifelse(dat$Species == 'virginica', 'virginica', 'other'))
trainrows <- runif(nrow(dat)) > 0.3
train <- dat[ trainrows,]
test  <- dat[!trainrows,]

model.rf <- randomForest(Species~., train, ntree=500, importance=TRUE, nodesize=5)

pred <- predict(model.rf, test)
table(pred)
```

```
other virginica
   25        14
```

Marginal effect

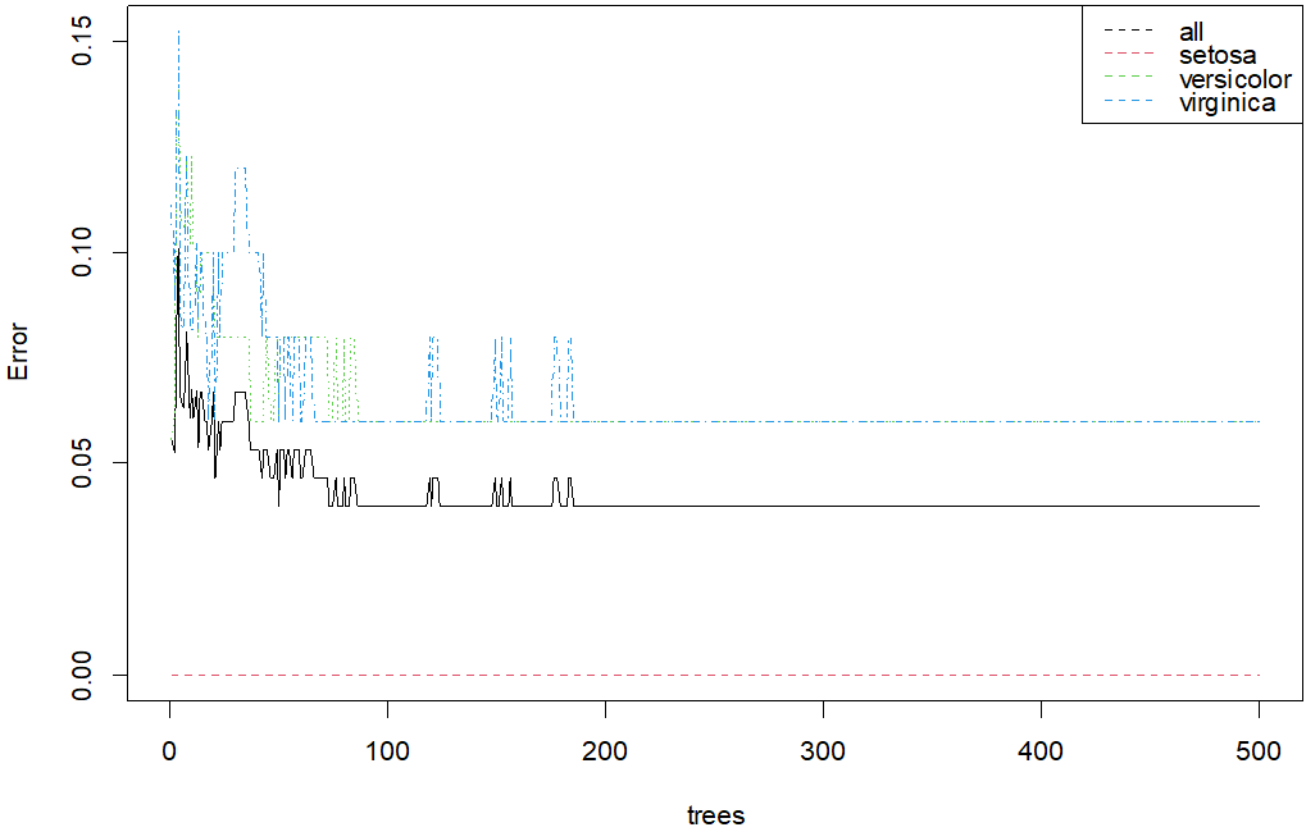
```
library(ggplot2)
pSpecies <- predict(model.rf, test,'vote')[,2]
plotData <- lapply(names(test[,1:4]), function(x){ # "Sepal.Length" "Sepal.Width"
                                                   # "Petal.Length" "Petal.Width"
  out <- data.frame(
    var = x, # "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
    type = c(rep('Actual', nrow(test)), rep('Predicted', nrow(test))),
    value = c(test[,x], test[,x]),
    species = c(as.numeric(test$Species)-1, pSpecies)
  )
  out$value <- out$value-min(out$value) #Normalize to [0,1]
  out$value <- out$value/max(out$value)
  out
})
plotData <- do.call(rbind, plotData)
qplot(value, species, data=plotData, facets = type ~ var, geom='smooth', span = 0.5)
```

Marginal effect



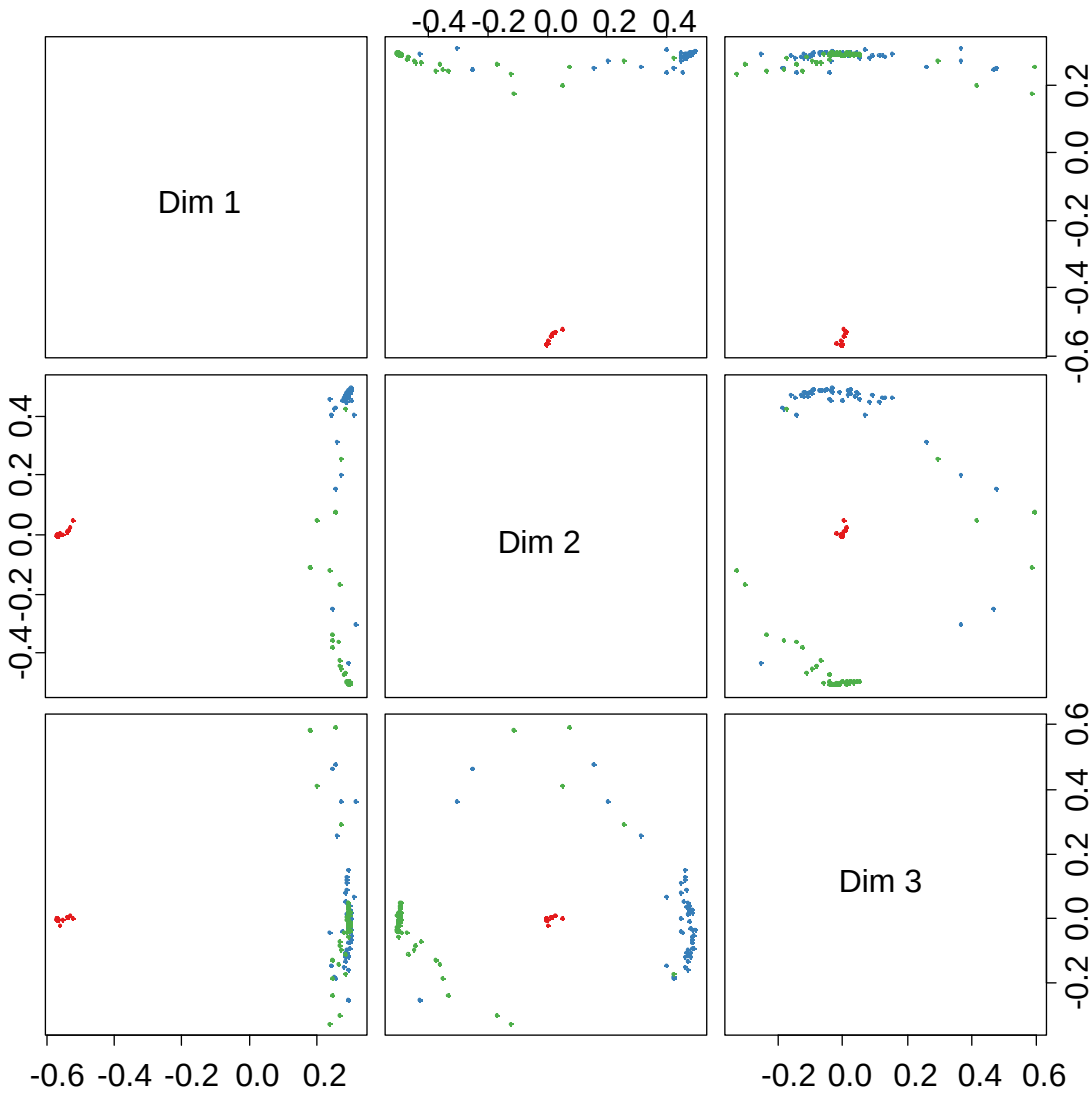
Number of trees

```
model.rf <- randomForest(Species~., iris, ntree=500, proximity=TRUE,
                         importance=TRUE, nodesize=5)
plot(model.rf)
legend("topright", c("OOB", levels(iris$Species)), lty=2, col=1:4)
```



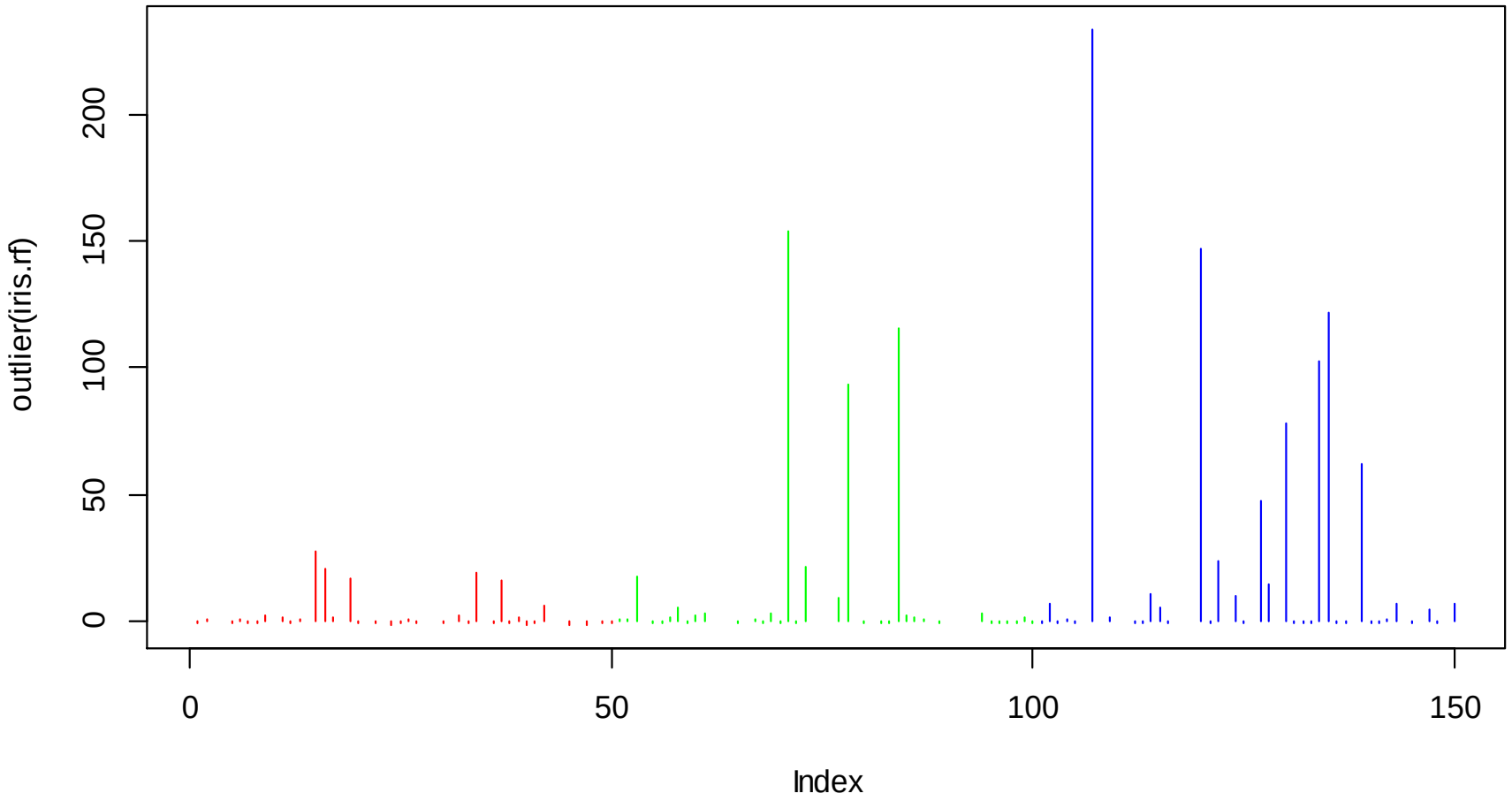
Multi-dimensional Scaling Plot of Proximity matrix from randomForest

MDSplot(model.rf, iris$Species, k=3)



Outliers

```
iris.rf <- randomForest(iris[,-5], iris[,5], proximity=TRUE)
plot(outlier(iris.rf), type="h", col=c("red", "green", "blue")[as.numeric(iris$Species)])
```



library(forestFloor)

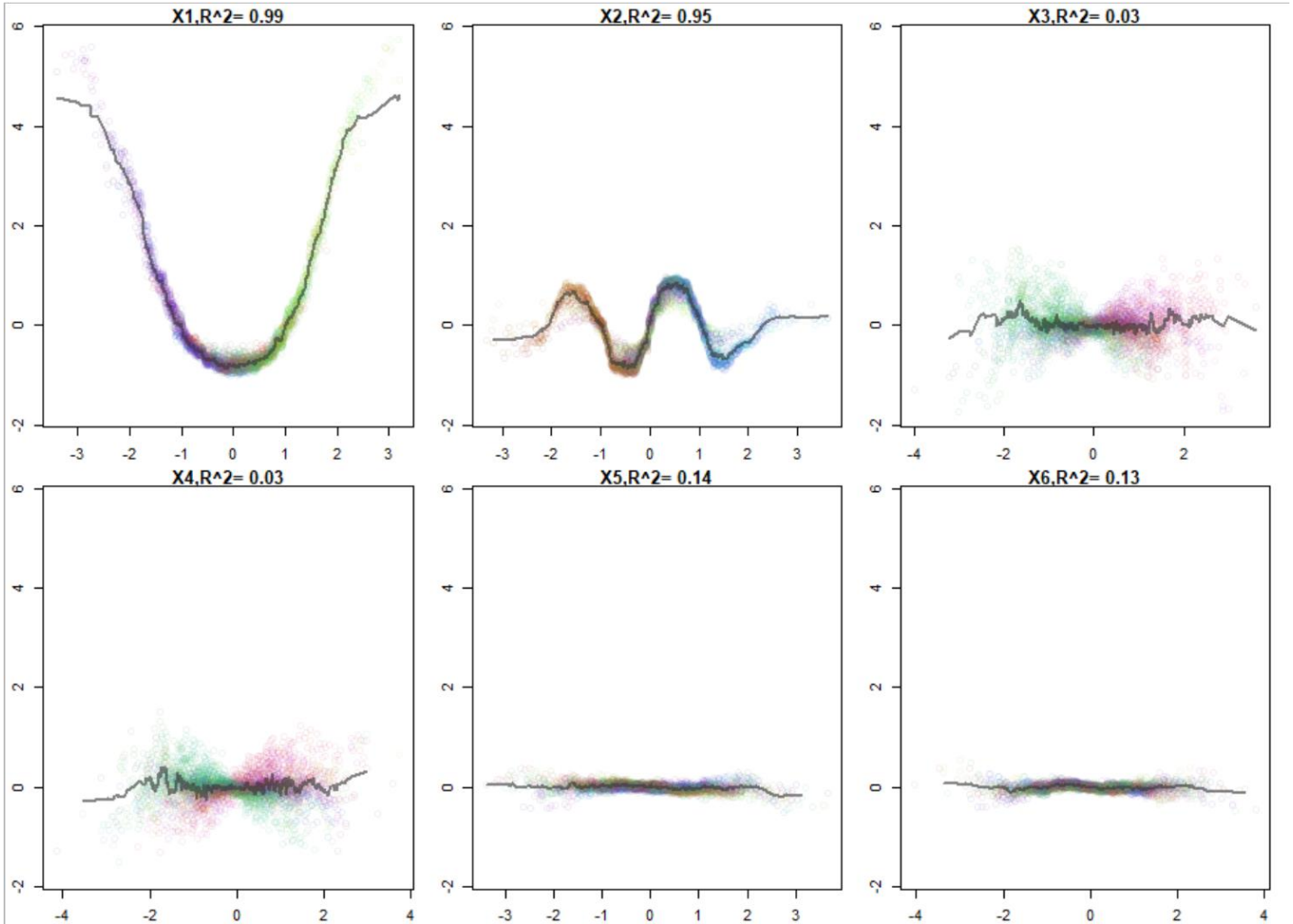
```
#1 - Regression example:
set.seed(1234)
library(forestFloor)
library(randomForest)

#simulate data y = x1^2+sin(x2*pi)+x3*x4 + noise
obs = 5000 #how many observations/samples
vars = 6   #how many variables/features
#create 6 normal distr. uncorr. variables
X = data.frame(replicate(vars, rnorm(obs)))
#create target by hidden function
Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

#grow a forest
rfo = randomForest(
  X, #features, data.frame or matrix. Recommended to name columns.
  Y, #targets, vector of integers or floats
  keep.inbag = TRUE,  # mandatory,
  importance = TRUE,  # recommended, else ordering by giniImpurity (unstable)
  sampsize = 1500 ,   # optional, reduce tree sizes to compute faster
  ntree = if(interactive()) 500 else 50 #speedup CRAN testing
)
```

Plot forestFloor() results

```
# compute forestFloor object, often only 5-10% time of growing forest
ff = forestFloor(
  rf.fit = rfo,       # mandatory
  X = X,              # mandatory
  calc_np = FALSE,    # TRUE or FALSE both works, makes no difference
  binary_reg = FALSE  # takes no effect here when rfo$type="regression"
)


Col = fcol(ff, cols=1:6, outlier.lim = 2.5)
#plot partial functions of most important variables first
plot(ff,                          # forestFloor object
     plot_seq = 1:6,              # optional sequence of features to plot
     orderByImportance=TRUE,   # if TRUE index sequence by importance, else by X column
     col=Col, plot_GOF = T
)
```

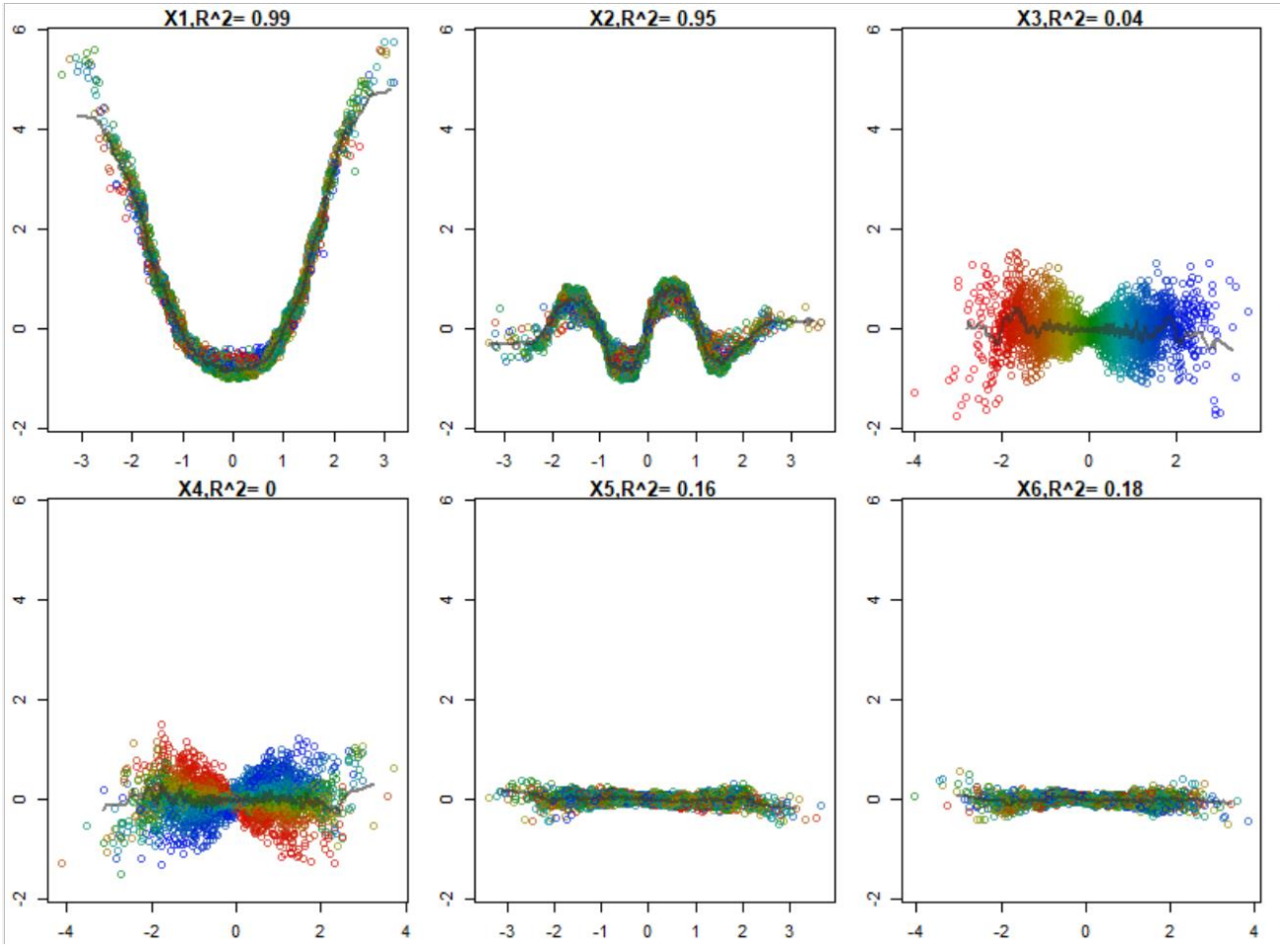
Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

Col = fcol(ff, cols=1:6, outlier.lim = 2.5)



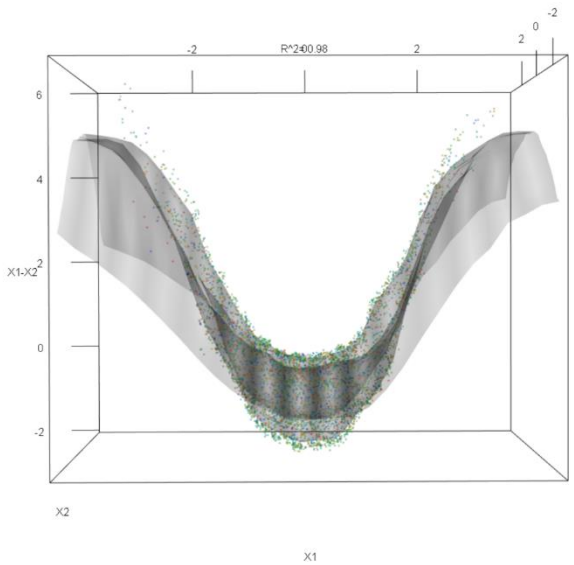
forestFloor: Y = with(X, X1^2 + sin(X2*pi) + 2 * X3 * X4 + 0.5 * rnorm(obs))

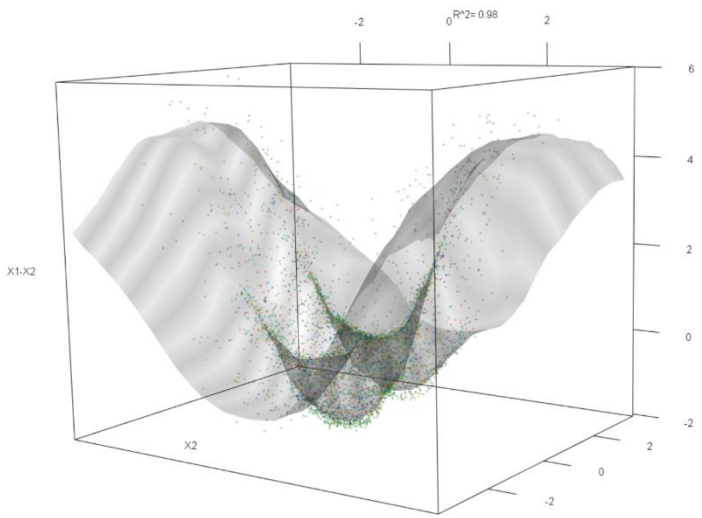
```
# Non interacting features are well displayed, whereas X3 and X4 are not
# by applying color gradient, interactions reveal themself
# also a k-nearest neighbor fit is applied to evaluate goodness-of-fit
Col=fcol(ff, 3, orderByImportance = T) # create color gradient of X3
plot(ff, col = Col, plot_GOF = TRUE)
```

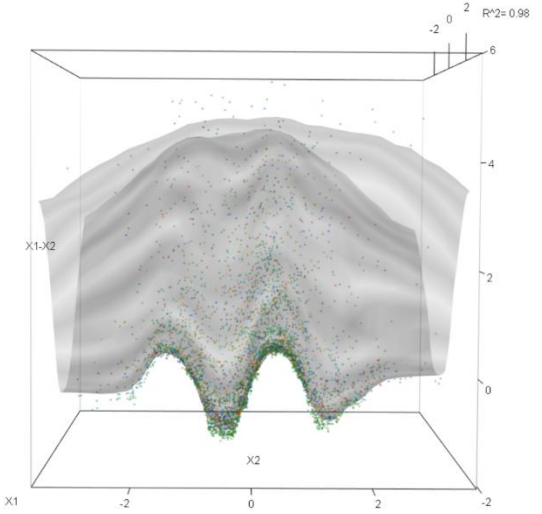


forestFloor::show3d()

```
# 3d plots
show3d(ff, c(1,2), 2, col = Col, plot_GOF = T)
# 2: integer vector of length 1 to p variables indices of feature contributions columns
```

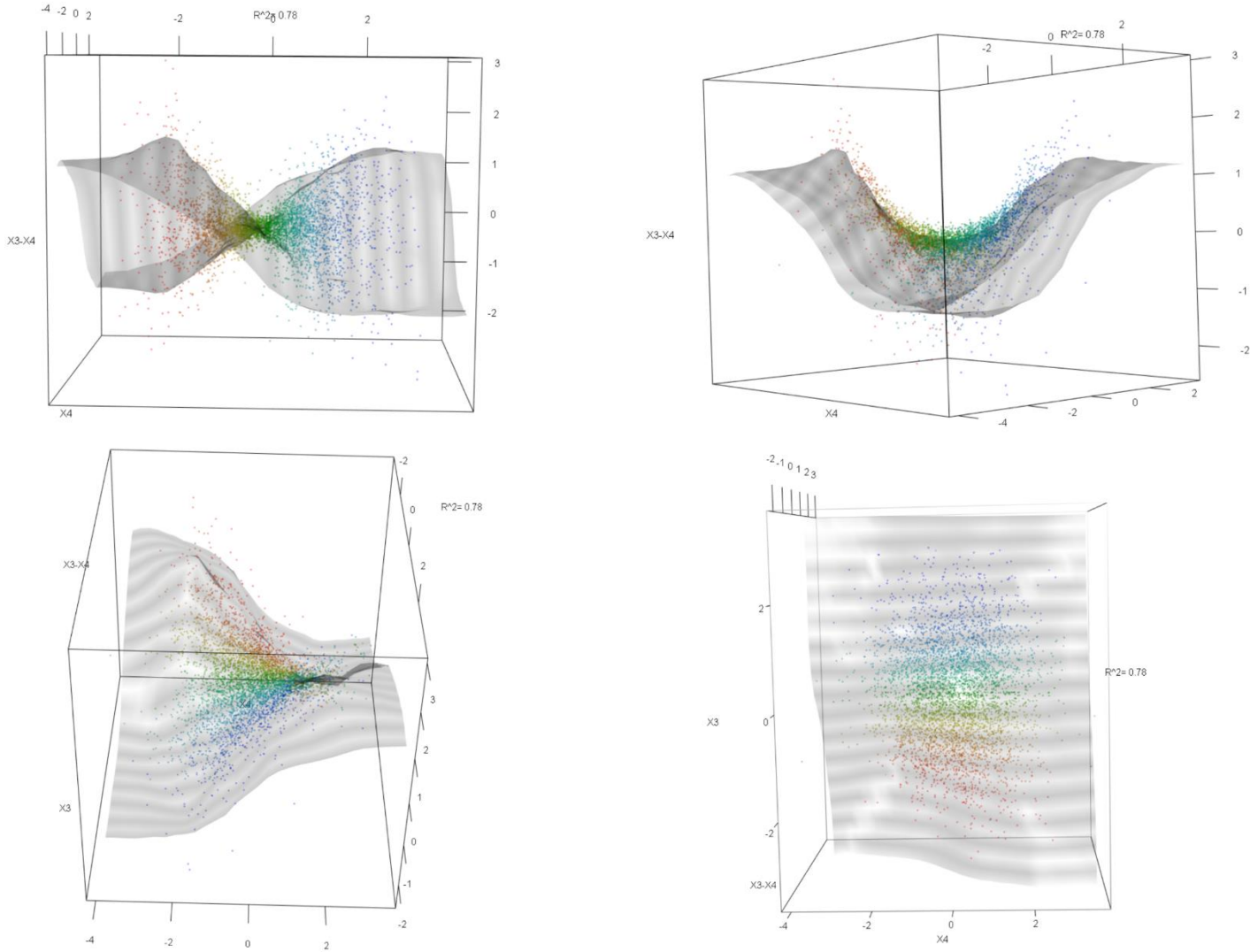




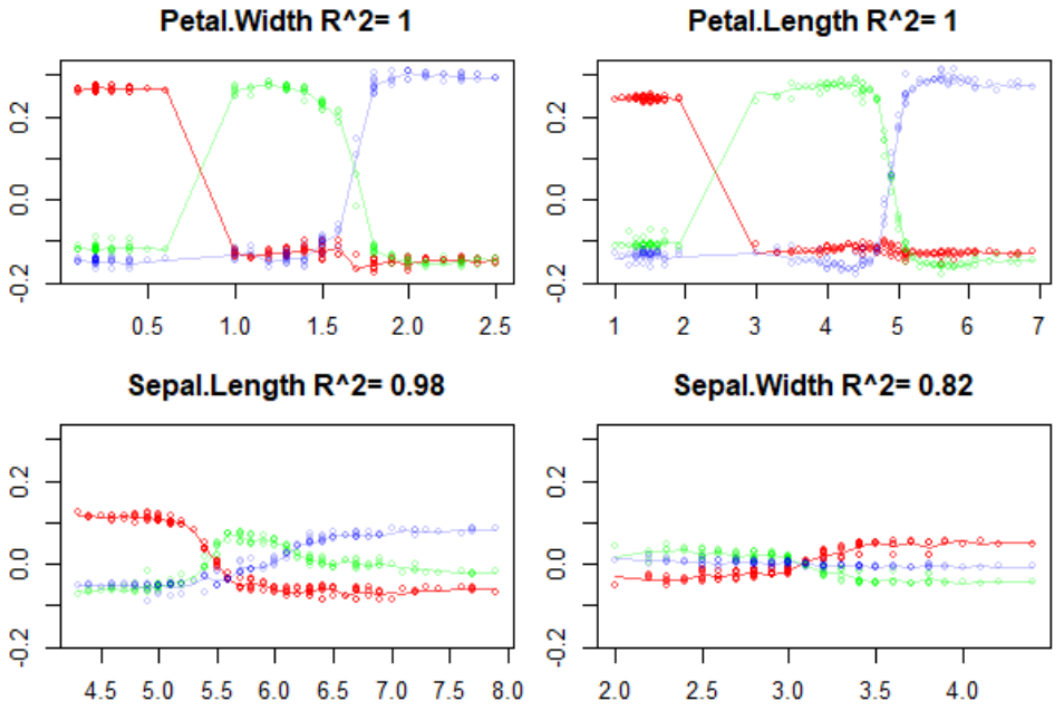


forestFloor::show3d()

show3d(ff, 3:4, col = Col, plot_GOF = TRUE, orderByImportance = FALSE)



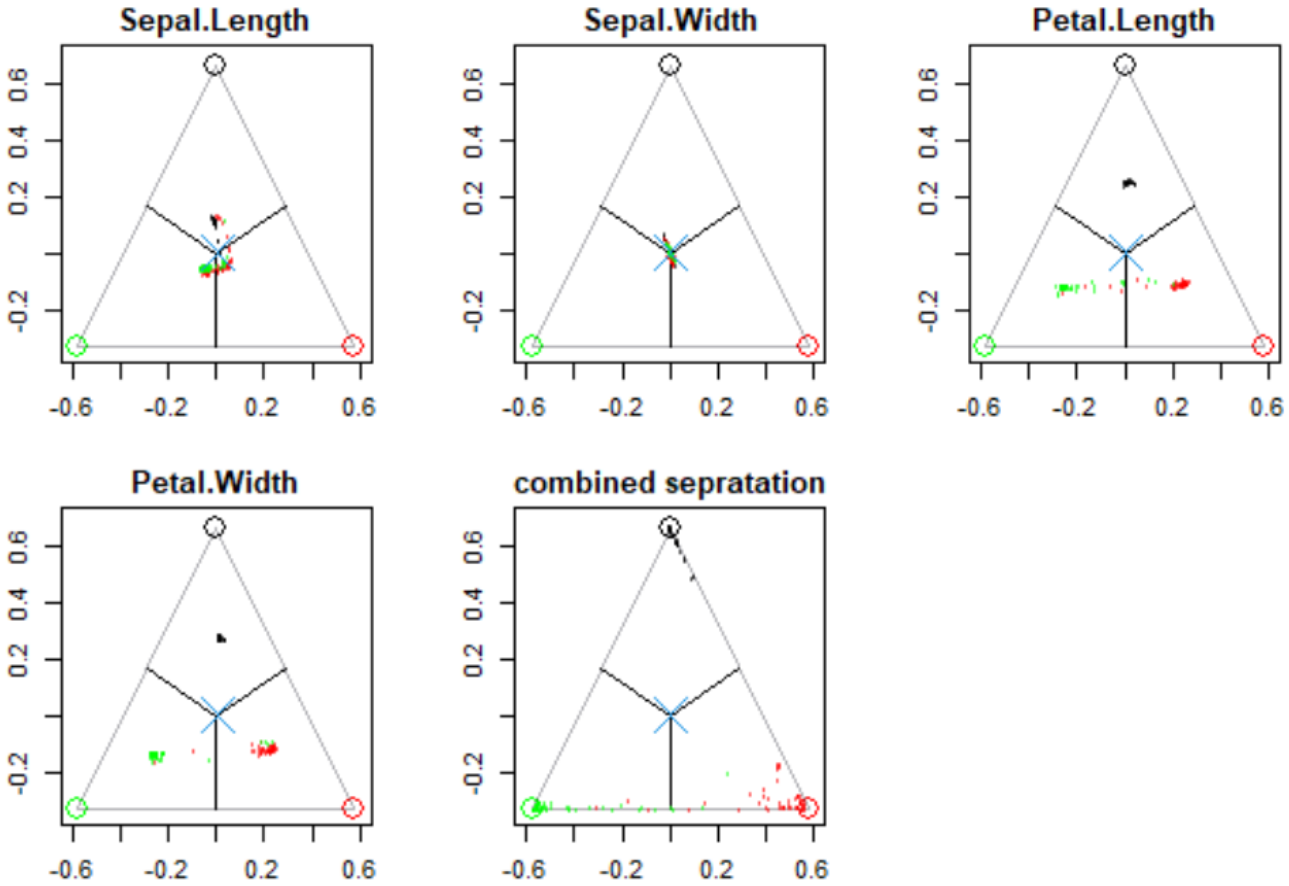
Multi classification example




```
set.seed(1234)
library(forestFloor)
library(randomForest)

data(iris)
X = iris[,!names(iris) %in% "Species"]
Y = iris[,"Species"]

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,           # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE # recommended, else ordering by giniImpurity (unstable)
)

ff = forestFloor(rf,X)

plot(ff,plot_GOF=TRUE,cex=.7,
     colLists=list(c("#FF0000A5"),
                   c("#00FF0050"),
                   c("#0000FF35")))
```

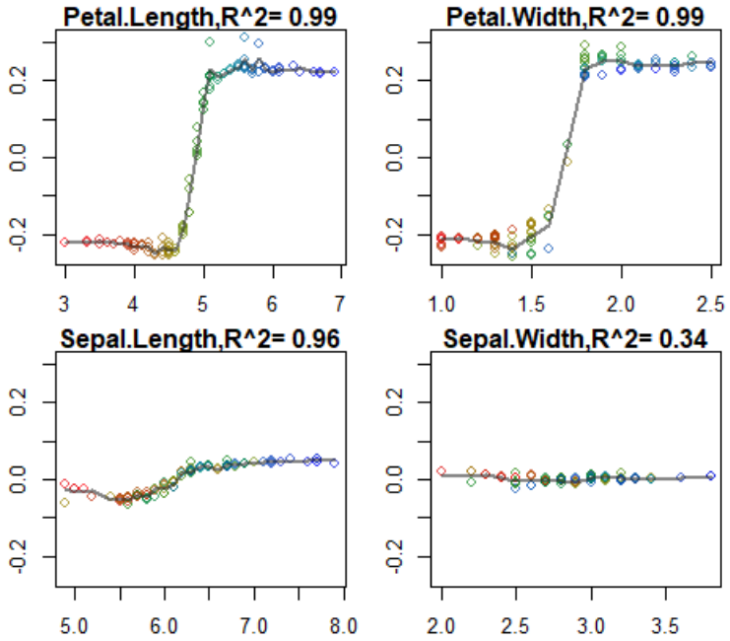
Multi classification example

```
show3d(ff,1:2,1:2, plot_GOF=TRUE) # 3D plot

# simplex plot (only for three class problems)
plot_simplex3(ff)
plot_simplex3(ff, zoom.fit = TRUE)

# 3D simplex plots (rough look, Z-axis is feature)
plot_simplex3(ff, fig3d = TRUE)
```



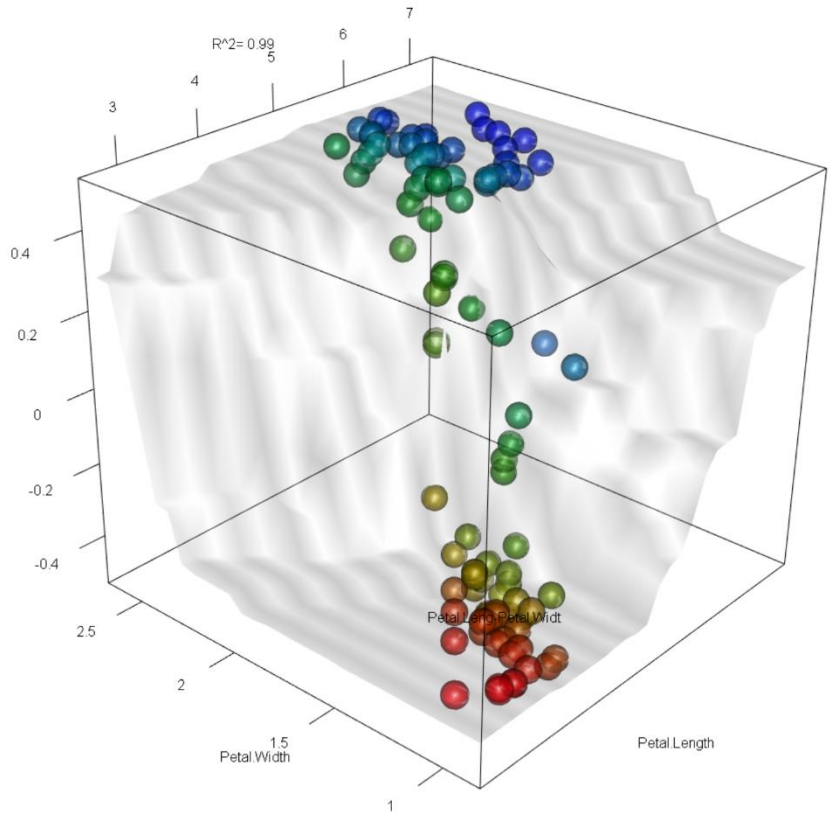
Binary regression example

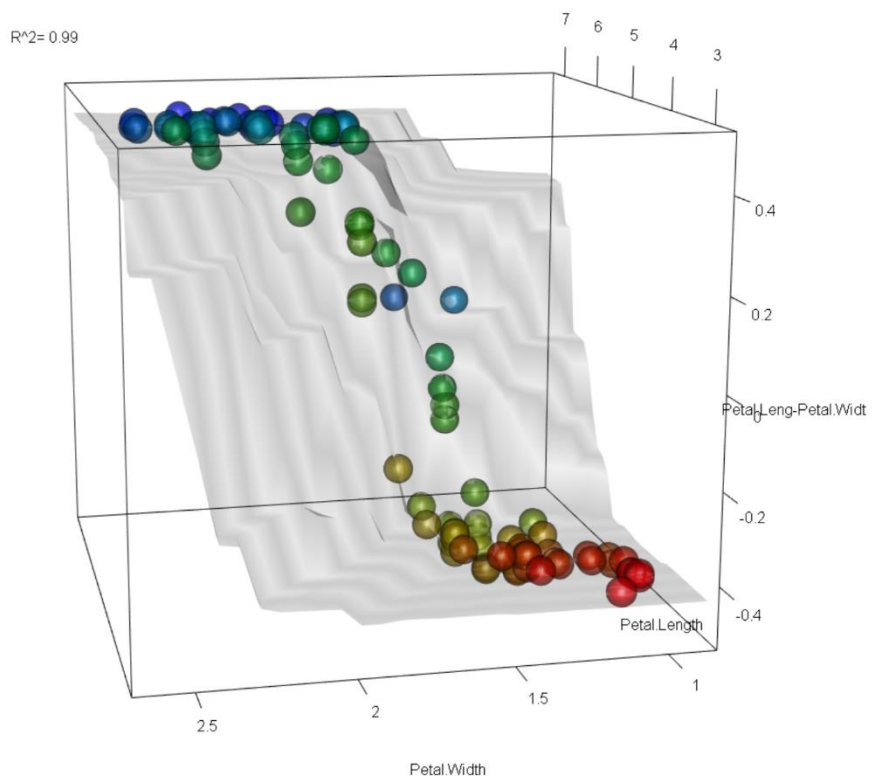
```
#3 - binary regression example
#classification of two classes can be seen as regression in 0 to 1 scale
set.seed(1234)
library(forestFloor)
library(randomForest)
data(iris)
#drop third class virginica
X = iris[-1:-50,!names(iris) %in% "Species"]
Y = iris[-1:-50,"Species"]
Y = droplevels((Y)) #drop unused level virginica

rf = randomForest(
  X,Y,
  keep.forest=TRUE,  # mandatory
  keep.inbag=TRUE,   # mandatory
  samp=20,           # reduce complexity of mapping structure, with same OOB%-explained
  importance  = TRUE # recommended, else giniImpurity
)

ff = forestFloor(rf,X,
                 calc_np=TRUE,    #mandatory to recalculate
                 binary_reg=TRUE) #binary regression, scale direction is printed
Col = fcol(ff,1) #color by most important feature
plot(ff,col=Col)   #plot features
```



3D plot

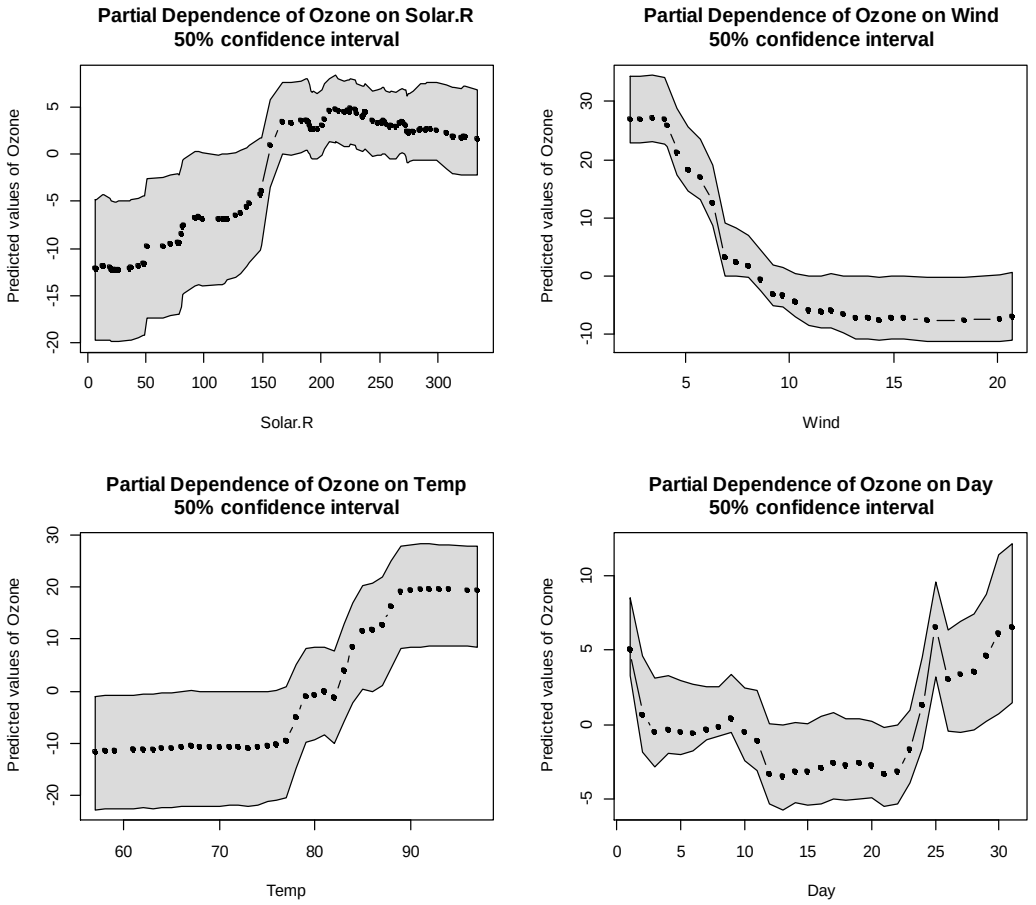
```
#interfacing with rgl::plot3d
show3d(ff, 1:2, col = Col, plot.rgl.args = list(size=2, type= "s", alpha=.5))
```





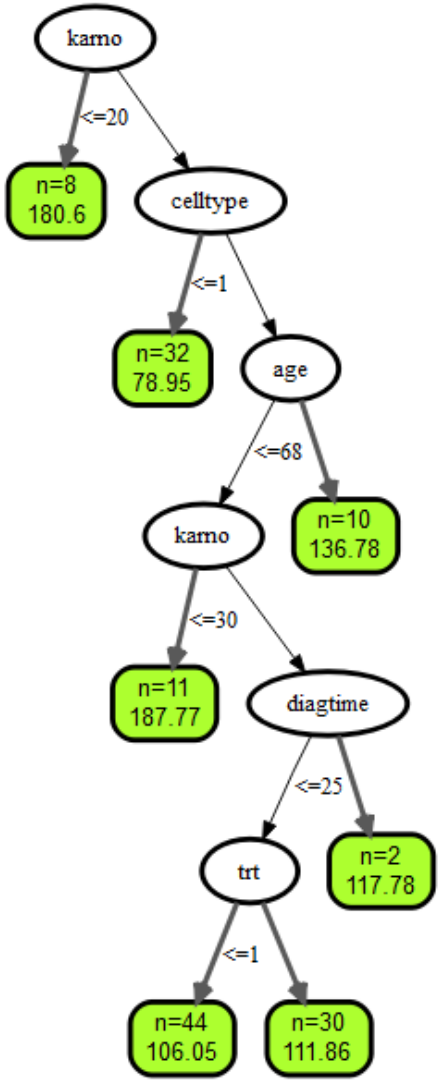
rfUtilities::rf.partial.ci

```
library(rfUtilities)
library(randomForest)
data(airquality)
airquality <- na.omit(airquality)
rf.ozone <- randomForest(y=airquality[,"Ozone"], airquality[,2:ncol(airquality)])

par(mfrow=c(2,2))
for(i in c("Solar.R", "Wind", "Temp", "Day")){
  rf.partial.ci(m=rf.ozone, x=airquality,
     yname="Ozone", xname=i, delta=TRUE)
}
```



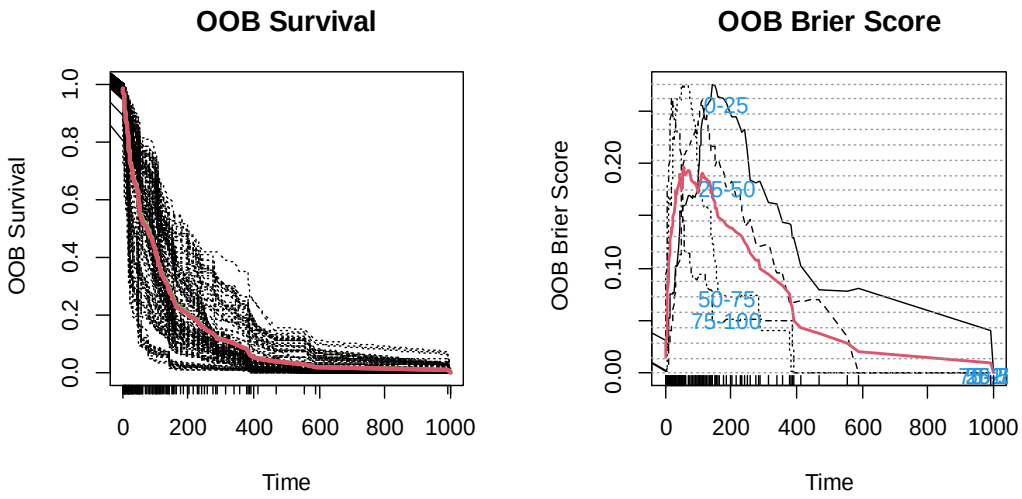
randomForestSRC::rfsrc(Surv(time, status) ~ .

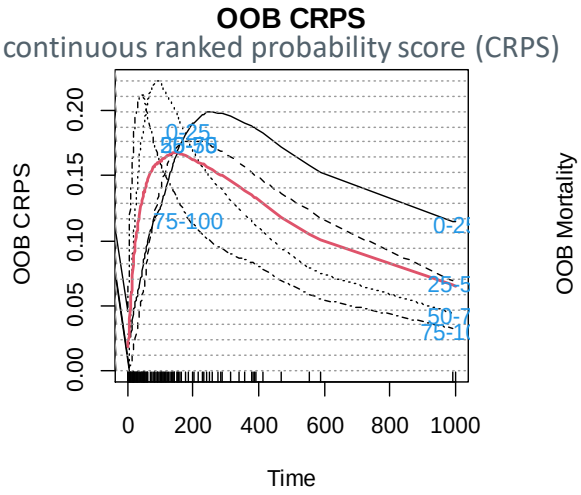
```
library(randomForestSRC)

## survival analysis

## veteran data
## randomized trial of two treatment regimens for lung cancer
data(veteran, package = "randomForestSRC")
table(veteran$status)
v.obj <- rfsrc(Surv(time, status) ~ ., data = veteran,
                        ntree = 1000, block.size = 1)

## print tree number 1
plot(get.tree(v.obj, 1))
plot(get.tree(v.obj, 1, surv.type = c("mort")))
plot(get.tree(v.obj, 1, surv.type = c("rel.freq")))
plot(get.tree(v.obj, 1, surv.type = c("surv")))

D = D[D$karno<=20, ]; nrow(D) # 8
```

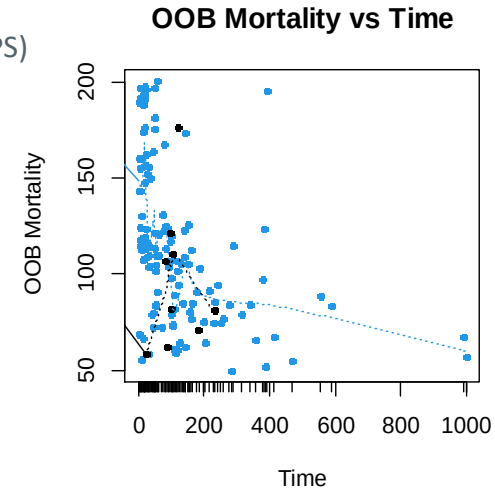


plot.survival()

```
data(veteran, package = "randomForestSRC")
plot.survival(rfsrc(Surv(time, status)~ ., veteran), cens.model = "rfsrc")
```

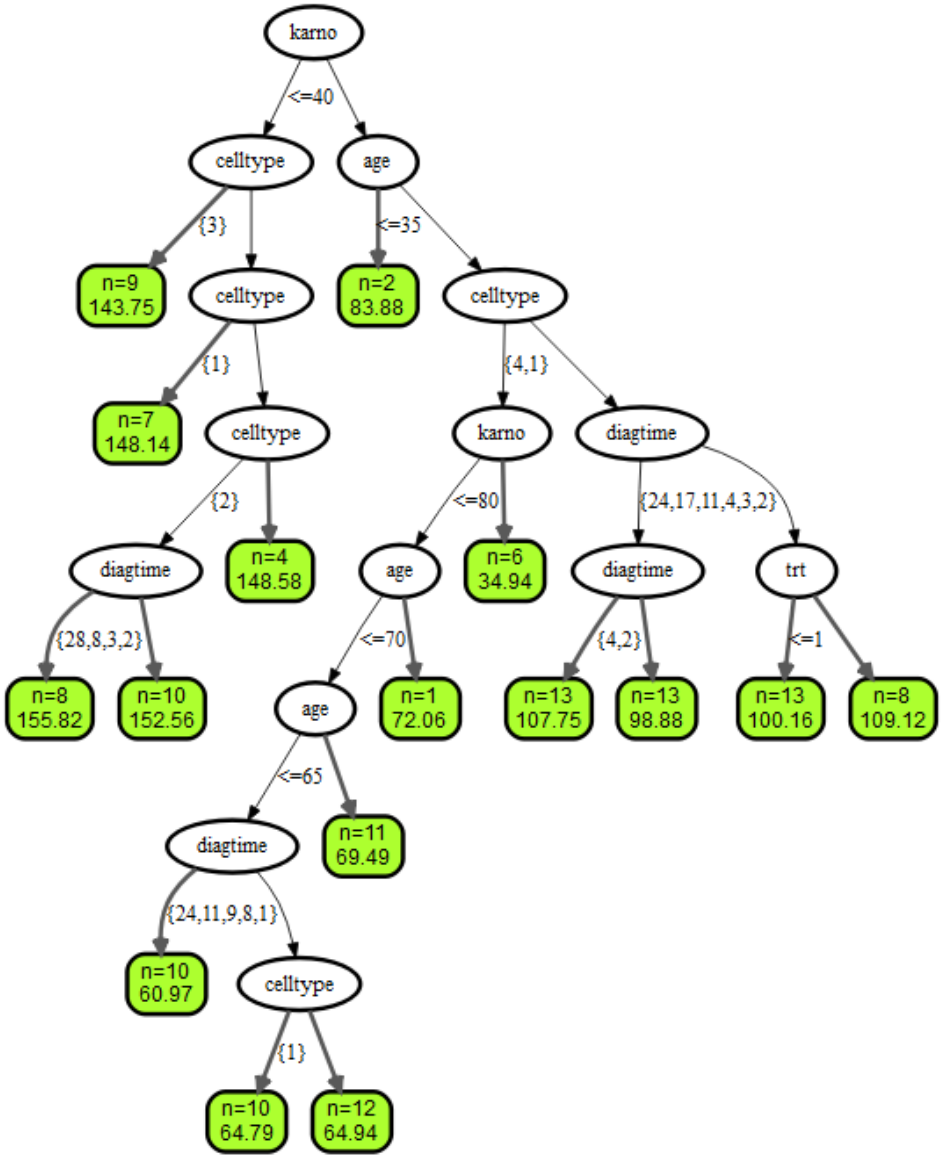




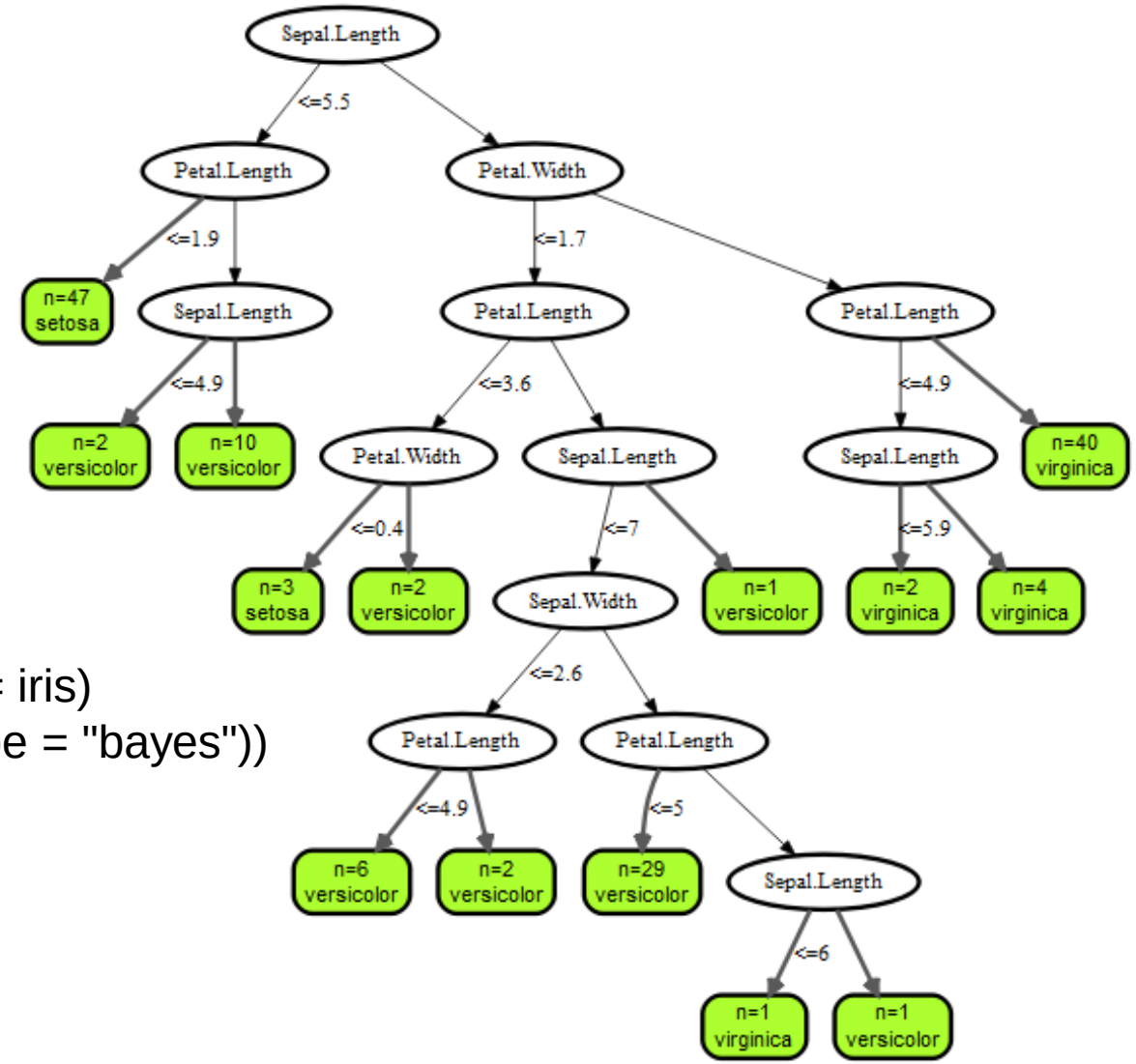


Survival analysis where factors have many levels

```
## survival where factors have many levels
data(veteran, package = "randomForestSRC")
vd <- veteran
vd$celltype=factor(vd$celltype)
vd$diagtime=factor(vd$diagtime)
vd.obj <- rfsrc(Surv(time,status)~., vd,
            ntree = 100, nodesize = 5)
plot(get.tree(vd.obj, 3))
```



plot(get.tree(iris.obj, 25, class.type = "bayes"))




```
## classification
iris.obj <- rfsrc(Species ~., data = iris)
plot(get.tree(iris.obj, 25, class.type = "bayes"))
```

Random forest example: classification for an insect species

```
insect = read.csv('d:/insect.csv', header = T)
library(randomForest)
RF = randomForest(insect[,c('L1','L2','L3', 'L4')],
        insect[,'Species'], importance=TRUE, ntree=10000)
RF
```

```
Call:
randomForest(x = insect[, c("L1", "L2", "L3", "L4")], y = insect[,      "Species"],
ntree = 10000, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 10000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 10%
Confusion matrix:
  A B C class.error
A 5 2 0   0.2857143
B 0 6 0   0.0000000
C 0 0 7   0.0000000
```

```
new.data = data.frame(L1=20, L2=50, L3=30, L4=20)
predict(RF, new.data, type="prob")
```

```
A             B             C
0.8233        0.0913        0.0854
```

```
predict(RF, new.data, type="response")
```

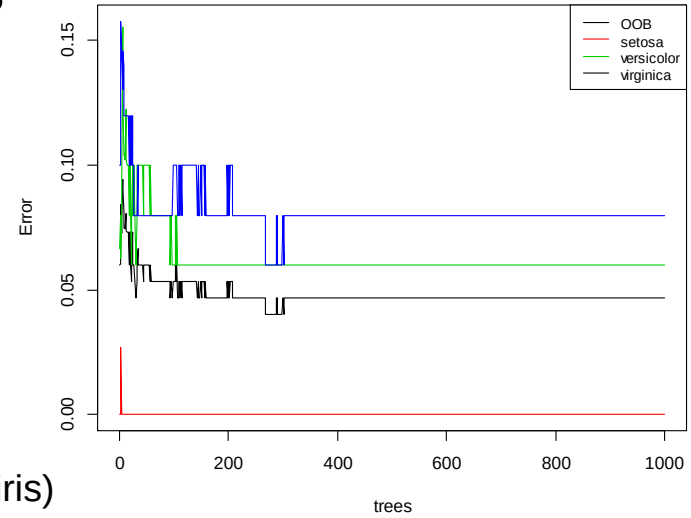
```
A
```

Species	L1	L2	L3	L4
A	16	27	31	33
A	15	23	30	30
A	16	27	27	26
A	18	20	25	23
A	15	15	31	32
A	15	32	32	15
A	12	15	16	31
B	8	23	23	11
B	7	24	25	12
B	6	25	23	10
B	8	45	24	15
B	9	28	15	12
B	5	32	31	11
C	22	23	12	42
C	25	25	14	60
C	34	25	16	52
C	30	23	21	54
C	25	20	11	55
C	30	23	21	54
C	25	20	11	55

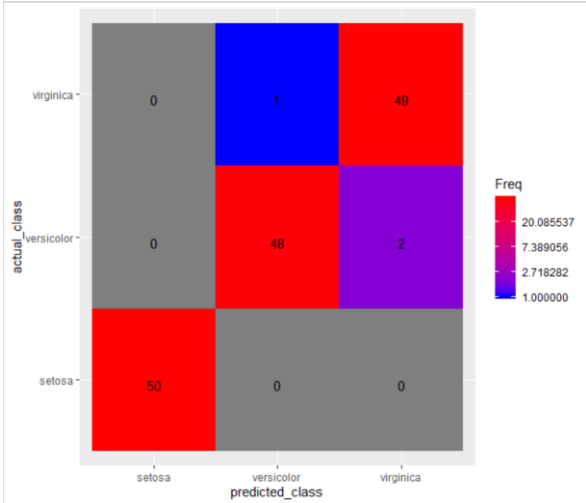
李欣海 (2013) 随机森林模型在分类与回归分析中的应用. 应用昆虫学报, **50,** 1190-1197.

Classification errors

```
RF  <- randomForest(Species ~ ., data=iris, importance=T, ntree=1000)
RF.predicted  <- predict(RF, newdata=iris)

plot(RF, lty=1)
legend('topright', colnames(RF$err.rate),
          col=1:3, cex=0.8, lty=1, box.lwd = 0)
```




```
library(MASS); library(ggplot2)
lda.result <- lda(Species ~ ., data=iris); lda.predict <- predict(lda.result, iris)
compare  <- data.frame(predicted_class = lda.predict$class, actual_class = iris$Species)
compare  <- data.frame(predicted_class = RF.predicted, actual_class = iris$Species)
confusion_matrix <- as.data.frame(table(compare))
ggplot(data = confusion_matrix, mapping = aes(x = predicted_class, y = actual_class)) +
  geom_tile(aes(fill = Freq)) + geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "blue", high = "red", trans = "log")
```

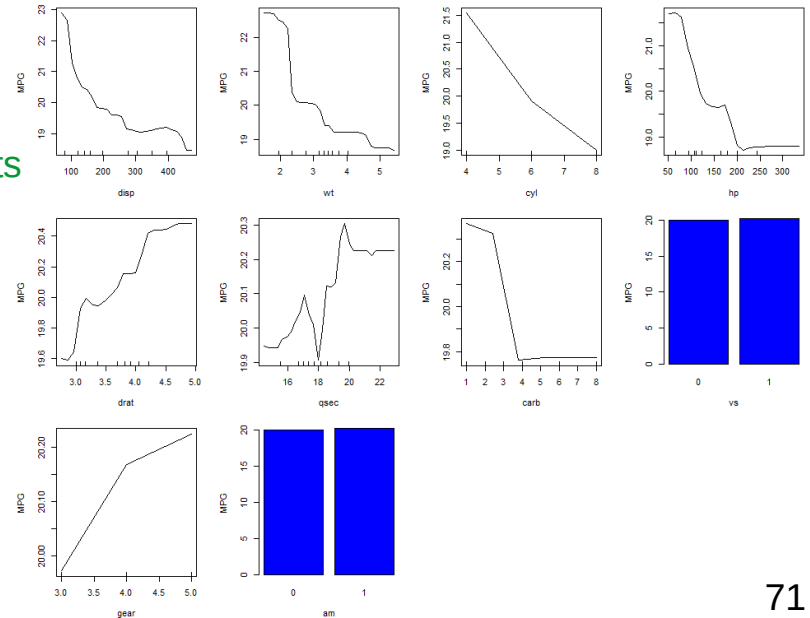


Cases for using random forest

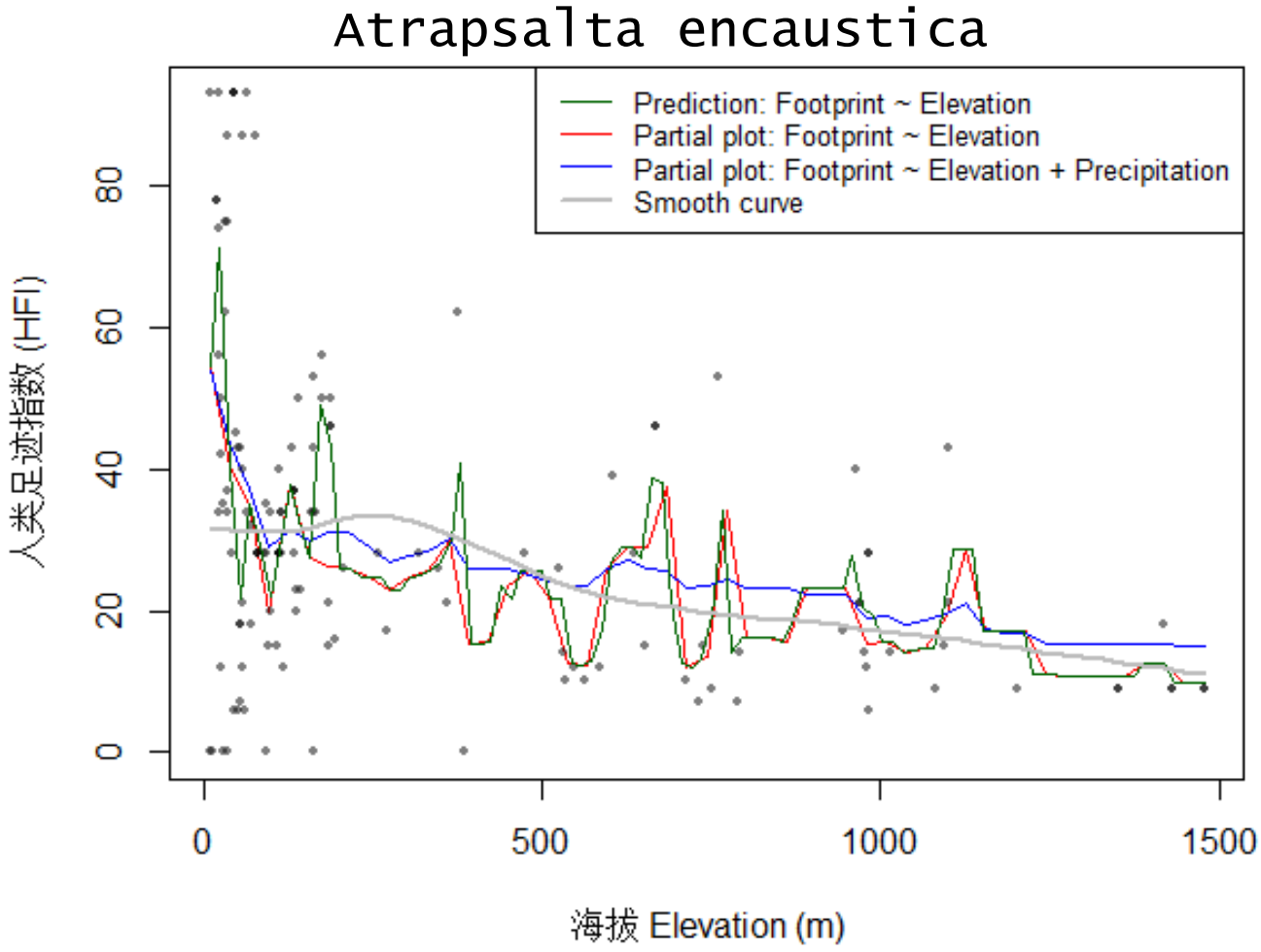
```
library(randomForest)
head(mtcars)
mtcars$vs = as.factor(mtcars$vs)
mtcars$am = as.factor(mtcars$am)
RF <- randomForest(mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb, data = mtcars,
                   prox = TRUE, importance = TRUE)
RF
```

```
% var explained: 83.46
```

```
imp <- importance(RF)
impvar <- rownames(imp)[order(imp[, 2], decreasing=TRUE)] #sort importance

# Plot partial effects
op <- par(mfrow=c(3, 3),mar=c(4,4,2,2))
for (i in seq_along(impvar)) {
      partialPlot(RF, mtcars, impvar[i], xlab=impvar[i], #Partial effects
      ylab='Longitude', ylim=c(26,30), main='')
}
```



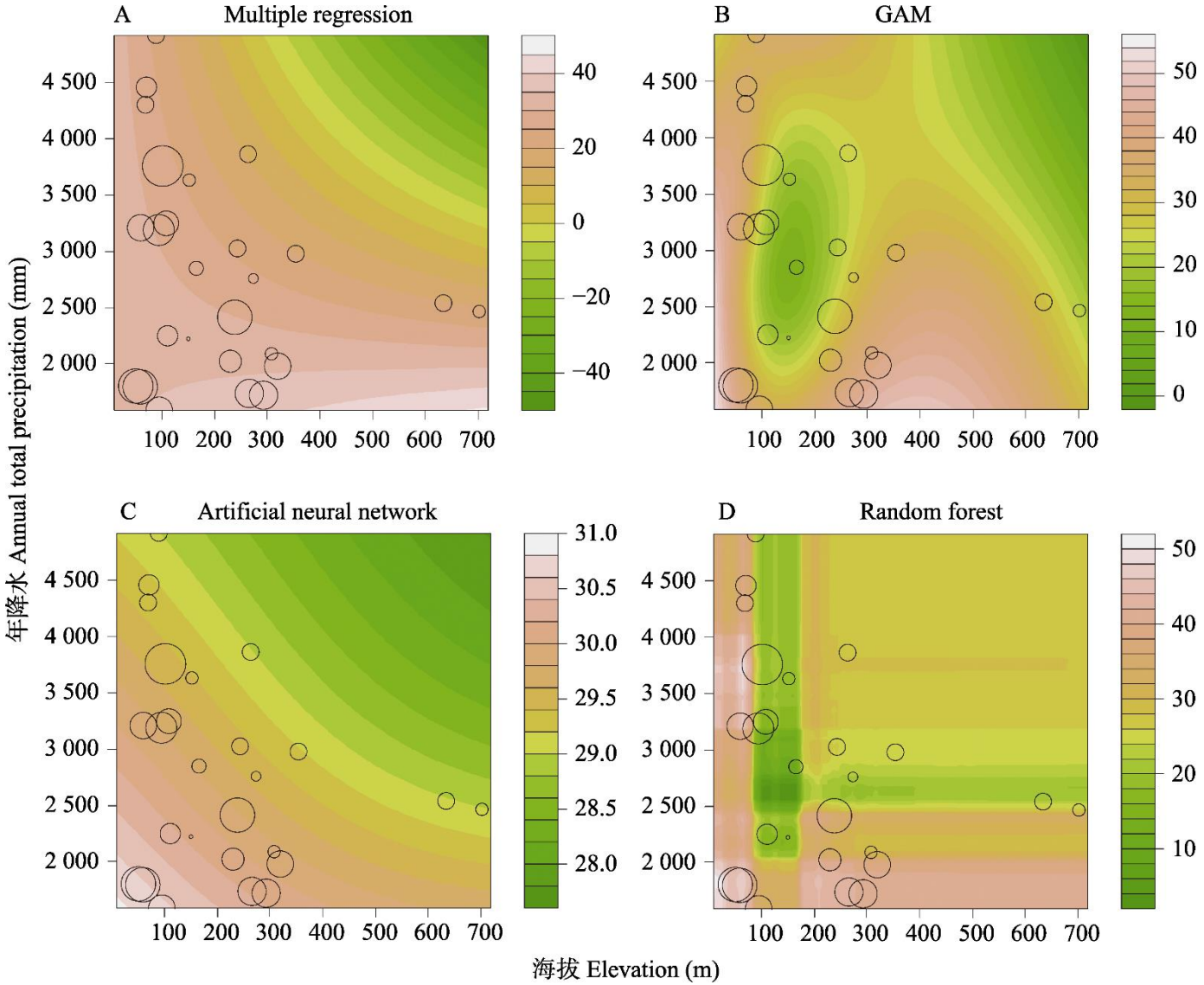
Partial plot of random forest



The human footprint index (HFI) at the occurrences of species *Atrapsalta encaustica* was explained by elevation only (random forest model 1 (RF1)), and by both elevation and annual total precipitation (random forest model 2 (RF2)). The green line is the predicted HFI using RF1 based on 100 evenly distributed elevations. The red line is the partial plot (elevation vs. HFI) of RF1. The blue line is the partial plot (elevation vs. HFI) of RF2. The gray line is the smooth line for elevation-HFI relationship.

李欣海，2019。随机森林是特点鲜明的模型，不是万能的模型。应用昆虫学报 56(1): 170–179

HFI = elevation*precipitation



Using multiple regression (A), generalized additive model (B), artificial neural network (C), and random forest (D) to predict the human footprint index (the gradient color) at the occurrences of the species *Zammara smaragdula*

李欣海，2019。随机森林是特点鲜明的模型，不是万能的模型。应用昆虫学报 56(1): 170–179

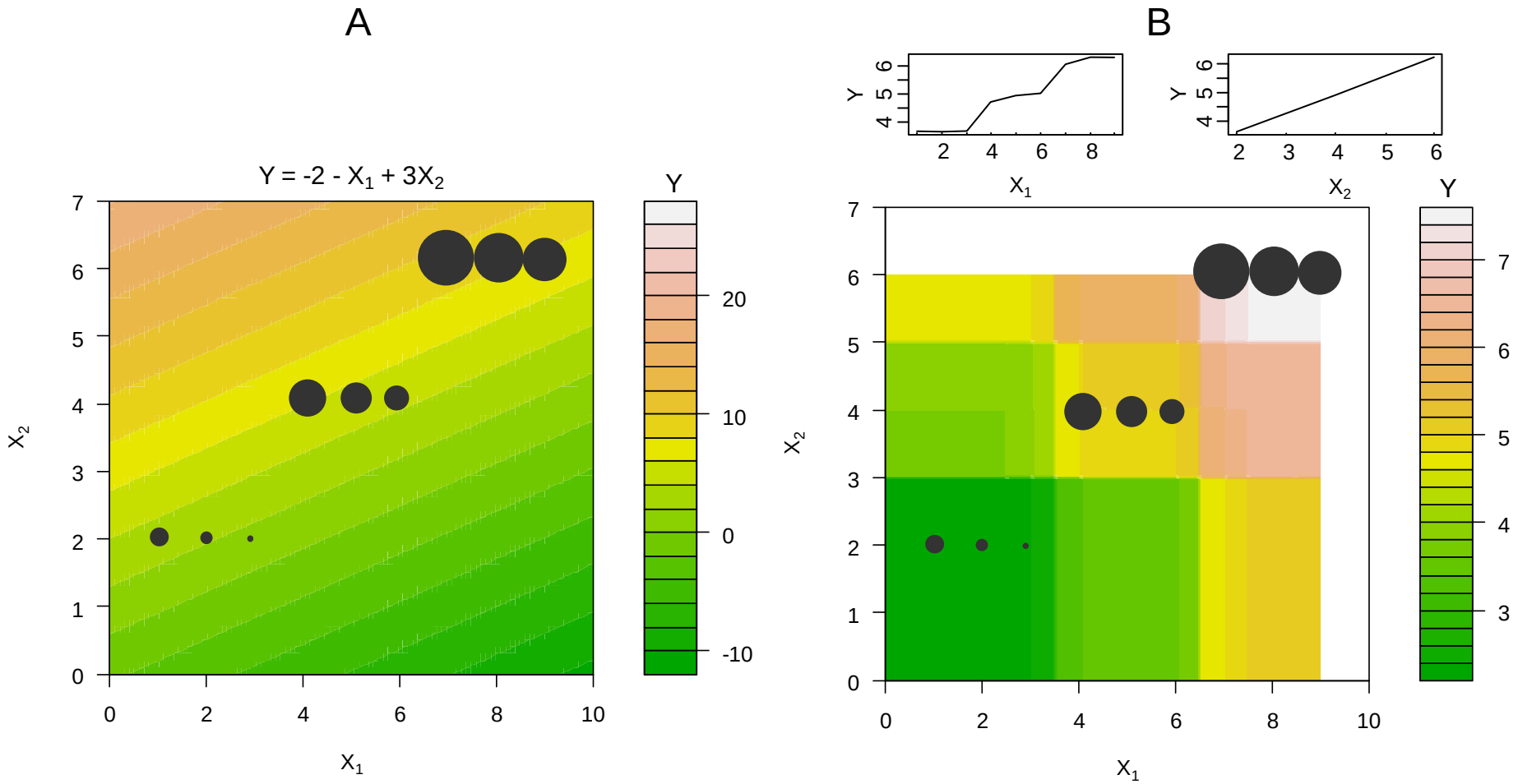
The advantages of Random Forest

- For many data sets, it produces a highly accurate classifier
- It handles a very large number of input variables
- It estimates the importance of variables in determining classification
- It generates an internal unbiased estimate of the generalization error as the forest building progresses
- It includes a good method for estimating missing data and maintains accuracy when a large proportion of the data are missing
- It provides an experimental way to detect variable interactions
- It can balance error in class population unbalanced data sets
- It computes proximities between cases, useful for clustering, detecting outliers, and (by scaling) visualizing the data
- Using the above, it can be extended to unlabeled data, leading to unsupervised clustering, outlier detection and data views
- Learning is fast

The disadvantages of Random Forest

- Random forests are prone to overfitting for some datasets. This is even more pronounced in noisy classification/regression tasks.
- Random forests do not handle large numbers of irrelevant features as well as ensembles of entropy-reducing decision trees.
- It is more efficient to select a random decision boundary than an entropy-reducing decision boundary, thus making larger ensembles more feasible. Although this may seem to be an advantage at first, it has the effect of shifting the computation from training time to evaluation time, which is actually a disadvantage for most applications.

Random forest failed to predict partial effects



Predicting the values of *Y* using multiple regression (A) and random forest (B) on the basis of two variables X_1 and X_2, which have strong partial effect on *Y*

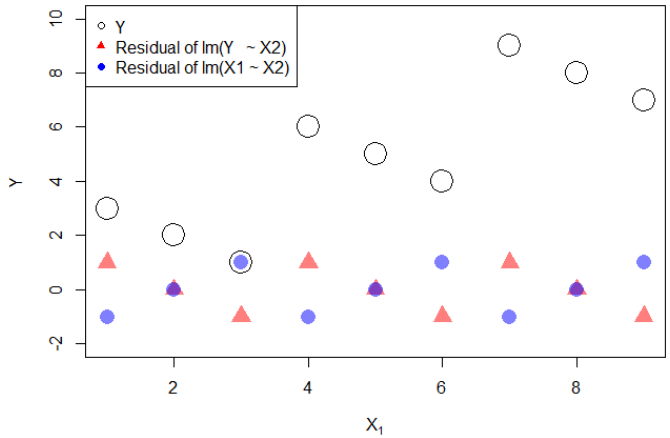
```
y  <- c(3, 2, 1, 6, 5, 4, 9, 8, 7)
x1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
x2 <- c(2, 2, 2, 4, 4, 4, 6, 6, 6)
```

李欣海，2019。随机森林是特点鲜明的模型，不是万能的模型。
应用昆虫学报 56(1): 170–179

Random forest does not give true partial effect

```
y   <- c(3, 2, 1, 6, 5, 4, 9, 8, 7)
x1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
x2 <- c(2, 2, 2, 4, 4, 4, 6, 6, 6)

# multiple regression
fit = summary(lm(y~x1+x2))
interception  = fit[[4]][1,1]
coef_x1       = fit[[4]][2,1]
coef_x2       = fit[[4]][3,1]
x.1 <- seq(min(x1)-1, max(x1)+1, length= 100)
x.2 <- seq(min(x2)-2, max(x2)+1, length= 100)
f <- function(x.1, x.2) { r <- interception + coef_x1*x.1 + coef_x2*x.2 };
y.pred <- outer(x.1, x.2, f)
filled.contour(x.1, x.2, y.pred, main='', color = terrain.colors,
            xlab=expression(paste(X[1])), ylab=expression(paste(X[2])),
            ylim=c(0,7), xlim=c(0,10))
points(x1/1.3, x2, pch=16, cex=y)

library(ggm) # partial correlation
D = cbind(y, x1, x2)
D = jitter(D, factor = .01)
pcor(c("y", "x1"), var(D)) # 0.8
pcor(c("y", "x1","x2"), var(D)) # -0.9999

resid.y   = residuals(lm(y  ~ x2))
resid.x1 = residuals(lm(x1 ~ x2))
cor(resid.y, resid.x1) # -1

# Remove the effect of X2
plot(x1, y, ylim=c(-2,10), pch=1, cex=3, xlab=expression(paste(X[1])), ylab="Y")
points(x1, resid.y, col=adjustcolor( "red", alpha.f = 0.5), pch=17, cex=2)
points(x1, resid.x1, col=adjustcolor( "blue", alpha.f = 0.5), pch=16, cex=2)
legend("topleft", c("Y", "Residual of lm(Y   ~ X2)", "Residual of lm(X1 ~ X2)"), pch=c(1, 17, 16), col=c(1,2,4))
```

```
# random forest
RF = randomForest(y ~ x1 + x2, importance = TRUE,
                  ntree = 1000)
x.1 <- seq(min(x1), max(x1), length = 100)
x.2 <- seq(min(x2), max(x2), length = 100)
data = expand.grid(x.1, x.2)
names(data) = c('x1','x2')
Y = predict(RF, newdata = data)
Y = matrix(Y, nrow=100, ncol=100)
filled.contour(x.1, x.2, Y, main=paste(''), color = terrain.colors,
            xlab=expression(paste(X[1])),
            ylab=expression(paste(X[2])))
# Plot partial effects
op <- par(mfrow = c(1, 2),mar = c(4,4,2,2))
D <- cbind(x1, x2, y)
partialPlot(RF, D, x1, xlab = expression(paste(X[1])), ylab='Y',
            main='')
partialPlot(RF, D, x2, xlab = expression(paste(X[2])), ylab='Y',
            main='')
```



Generalized Boosting Models (BRT)

- Boosting is the process of iteratively adding basis functions in a greedy fashion so that each additional basis function further reduces the selected loss function (Friedman et al. 2000; Friedman 2001).

- For example, Gradient Boosting is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Friedman, J., T. Hastie, and R. Tibshirani. 2000. Additive logistic regression: A statistical view of boosting. Annals of Statistics 28:337-374.

Friedman, J. H. 2001. Greedy function approximation: A gradient boosting machine. Annals of Statistics 29:1189-1232.

Generalized Boosting Models (BRT)

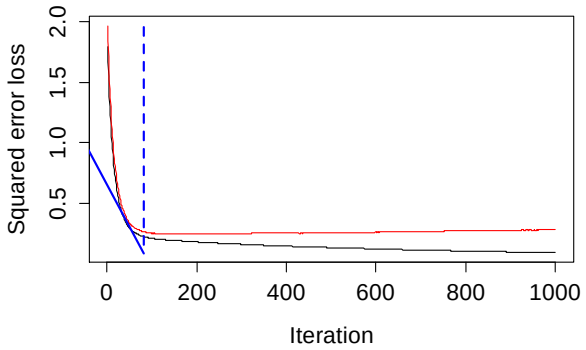
```
# create some data
library(gbm)
N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4], N, replace=TRUE),
      levels=letters[4:1])
X4 <- factor(sample(letters[1:6], N, replace=TRUE))
X5 <- factor(sample(letters[1:3], N, replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1, 0, 1, 2)[as.numeric(X3)]

SNR <- 10 # signal-to-noise ratio
Y <- X1**1.5 + 2 * (X2**.5) + mu
sigma <- sqrt(var(Y)/SNR)
Y <- Y + rnorm(N, 0, sigma)

# introduce some missing values
X1[sample(1:N, size=500)] <- NA
X4[sample(1:N, size=300)] <- NA

data <- data.frame(Y=Y, X1=X1, X2=X2, X3=X3,
                   X4=X4, X5=X5, X6=X6)
```

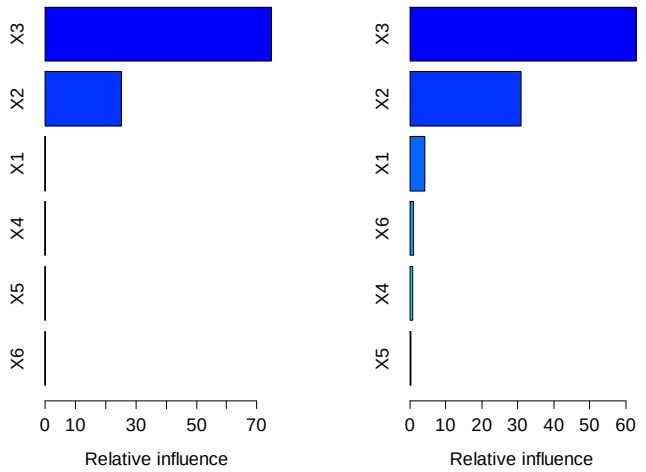
```
# fit initial model
gbm1 <-
  gbm(Y~X1+X2+X3+X4+X5+X6,    # formula
      data=data,                   # dataset
      var.monotone=c(0,0,0,0,0,0), # -1: monotone decrease,
                                   # +1: monotone increase,
                                   #  0: no monotone restrictions
      distribution="gaussian",     # see the help for other choices
      n.trees=1000,                # number of trees
      shrinkage=0.05,              # shrinkage or learning rate,
                                   # 0.001 to 0.1 usually work
      interaction.depth=3,         # 1: additive model, 2: two-way interactions, etc.
      bag.fraction = 0.5,          # subsampling fraction, 0.5 is probably best
      train.fraction = 0.5,        # fraction of data for training,
                                   # first train.fraction*N used for training
      n.minobsinnode = 10,         # minimum number of obs in the trees terminal nodes
      cv.folds = 3,                # do 3-fold cross-validation
      keep.data=TRUE,              # keep a copy of the dataset with the object
      verbose=FALSE,               # don't print out progress
      n.cores=1)                   # use only a single core
```

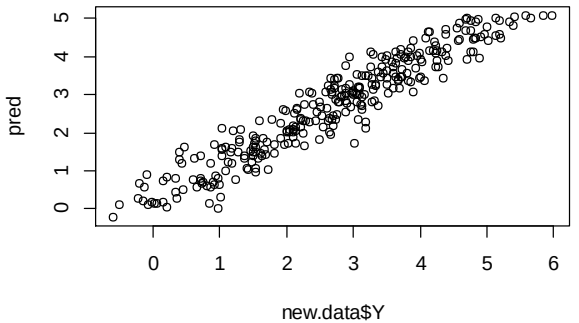
	Y	X1	X2	X3	X4	X5	X6
1	2.948341	NA	1.068696	c	e	a	2.779125
2	2.71057	0.227893	0.881948	c	d	a	2.360077
3	3.593163	0.911871	0.705704	a	a	a	1.695979
4	3.922507	NA	0.454434	a	c	c	1.487928
5	1.536815	0.780956	1.511377	d	c	c	0.469543
6	0.881173	NA	0.494163	d	d	b	1.210679

Generalized Boosting Models (BRT)

```
# check performance using an out-of-bag estimator
# object$train.error (in black) and object$valid.error (in red)
best.iter <- gbm.perf(gbm1, method="OOB")
print(best.iter) # 94

# check performance using a 50% heldout test set
best.iter <- gbm.perf(gbm1, method="test")
print(best.iter) # 128

# check performance using 5-fold cross-validation
best.iter <- gbm.perf(gbm1, method="cv")
print(best.iter) # 101

# plot variable influence
summary(gbm1, n.trees=1)        # based on the first tree
summary(gbm1, n.trees=best.iter) # based on the estimated best number of trees

# prediction
new.data = data[201:500,]
pred <- predict(gbm1, new.data, best.iter)
plot(new.data$Y, pred)
```







GARP (Genetic Algorithm for Rule-Set Prediction)

http://www.nhm.ku.edu/desktopgarp/

A computer program based on genetic algorithm that creates ecological niche models for species. It describes environmental conditions (precipitation, temperatures, elevation, etc.) under which the species should be able to maintain populations (Stockwell and Noble 1992).

As input, local observations of species and related environmental parameters are used which describe potential limits of the species' capabilities to survive.

A GARP model is a random set of mathematical rules which can be read as limiting environmental conditions. Each rule is considered as a gene; the set of genes is combined in random ways to further generate many possible models describing the potential of the species to occur (Anderson et al. 2003).

Stockwell, D. R. B. and I. R. Noble. 1992. Induction of sets of rules from animal distribution data: a robust and informative method of data analysis. Mathematics and Computers in Simulation **33**: 385-390.

Anderson, R. P., D. Lew, and A. T. Peterson. 2003. Evaluating predictive models of species' distributions: criteria for selecting optimal models. **Ecological Modelling** 162:211-232.

One application of GARP

A. Townsend Peterson, Miguel A. Ortega-Huerta, Jeremy Bartley, Victor Sanchez-Cordero, Jorge Soberonk, Robert H. Buddemeier & David R. B. Stockwell. 2002. Future projections for Mexican faunas under global climate change scenarios. **Nature** 461:626-629

Maxent (Maximum entropy method)

http://www.cs.princeton.edu/~schapire/maxent/

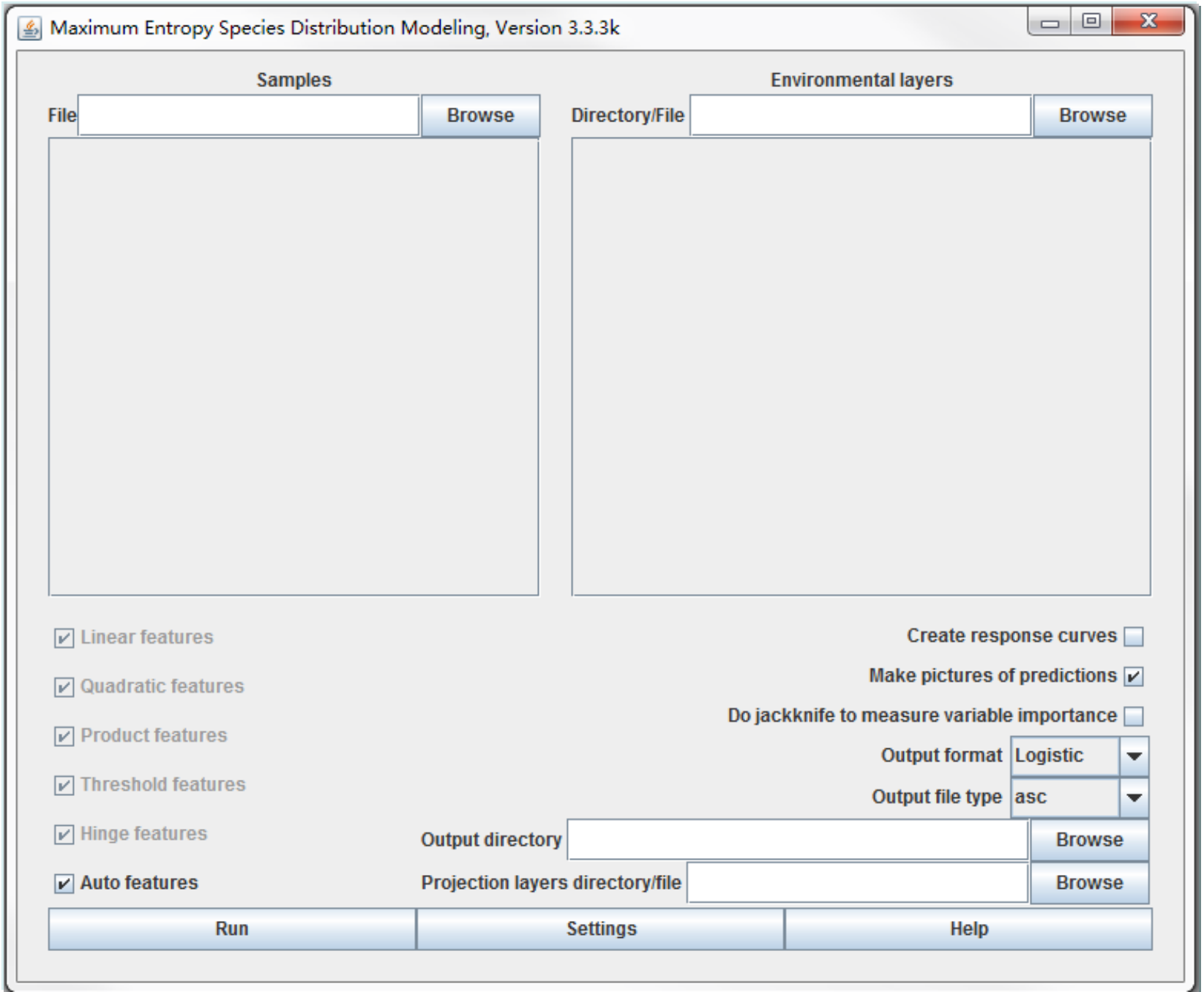
The idea of Maxent is to estimate a target probability distribution by finding the probability distribution of maximum entropy (i.e., that is most spread out, or closest to uniform), subject to a set of constraints that represent our incomplete information about the target distribution (Phillips et al. 2006).

The information available about the target distribution often presents itself as a set of real-valued variables, called “features”, and the constraints are that the expected value of each feature should match its empirical average (average value for a set of sample points taken from the target distribution (Phillips et al. 2006).

Maxent models species geographic distributions with presence-only data.

Phillips, S. J., R. P. Anderson, and R. E. Schapire. 2006. Maximum entropy modeling of species geographic distributions. **Ecological Modelling** 190:231-259.

Maxent (maximum entropy method)



Maxent

Maxent is a general-purpose machine learning method with a simple and precise mathematical formulation, and it has a number of aspects that make it well-suited for species distribution modeling (Phillips et al. 2006; Phillips and Dudik 2008).

Maxent is not as mature a statistical method as GLM, so there are fewer guidelines for its use in general, and fewer methods for estimating the amount of error in a prediction (Phillips et al. 2006).

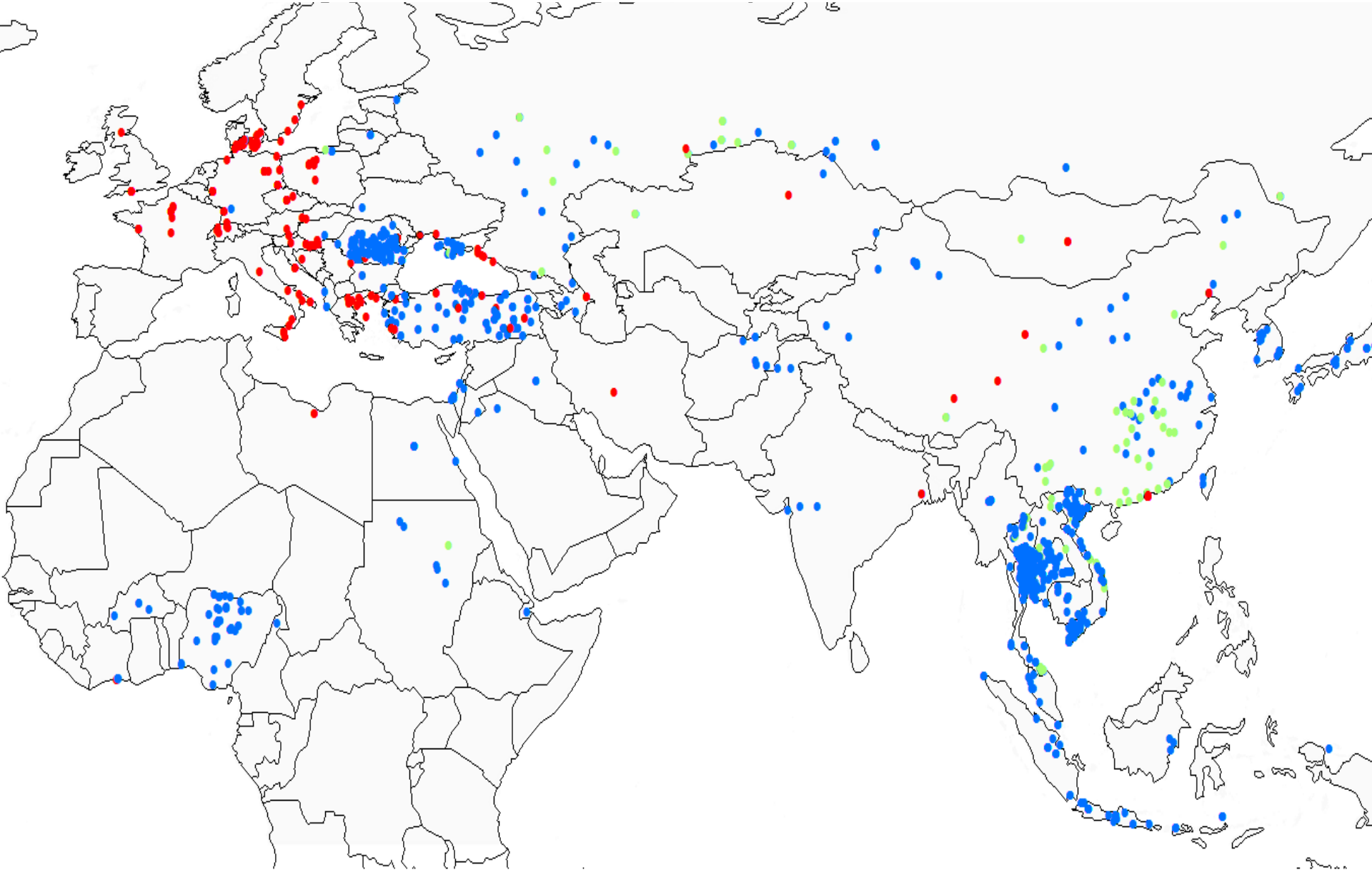
Maximum entropy modeling is an active area of research in statistics and machine learning (Phillips et al. 2006). For recent machine learning review, see Olden et al. (2008)

Phillips, S. J., R. P. Anderson, and R. E. Schapire. 2006. Maximum entropy modeling of species geographic distributions. **Ecological Modelling** 190:231-259.

Phillips, S. J., and M. Dudik. 2008. Modeling of species distributions with Maxent: new extensions and a comprehensive evaluation. **Ecography** 31:161-175.

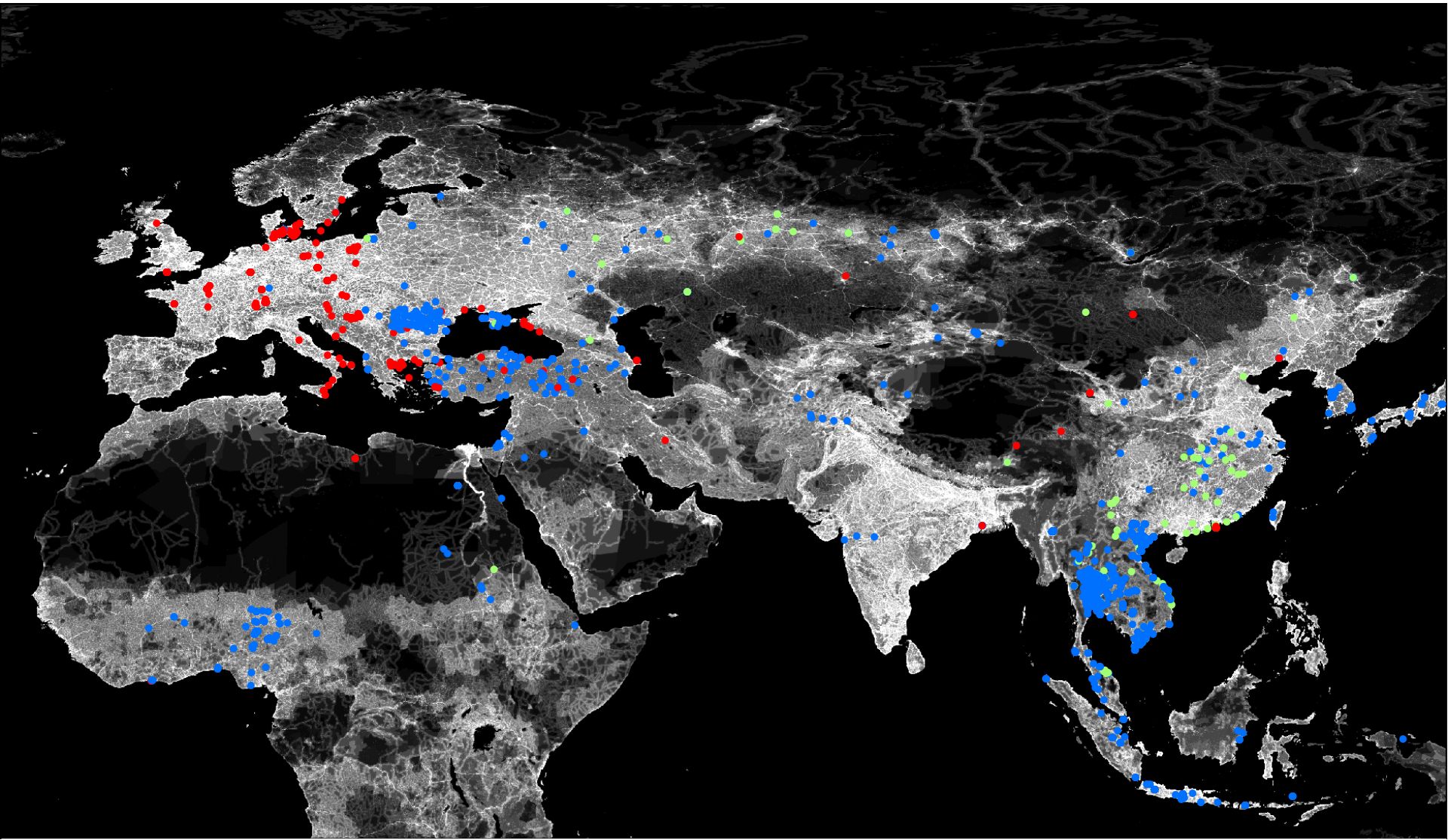
Olden, J. D., J. J. Lawler, and N. L. Poff. 2008. Machine learning methods without tears: A primer for ecologists. **Quarterly Review of Biology** 83:171-193.

Occurrence sites of avian influenza (H5 type) in domestic poultry (blue), wild birds (red), and unknown animals (green) in Asia, Europe and Africa from December 2003 to June 2006.



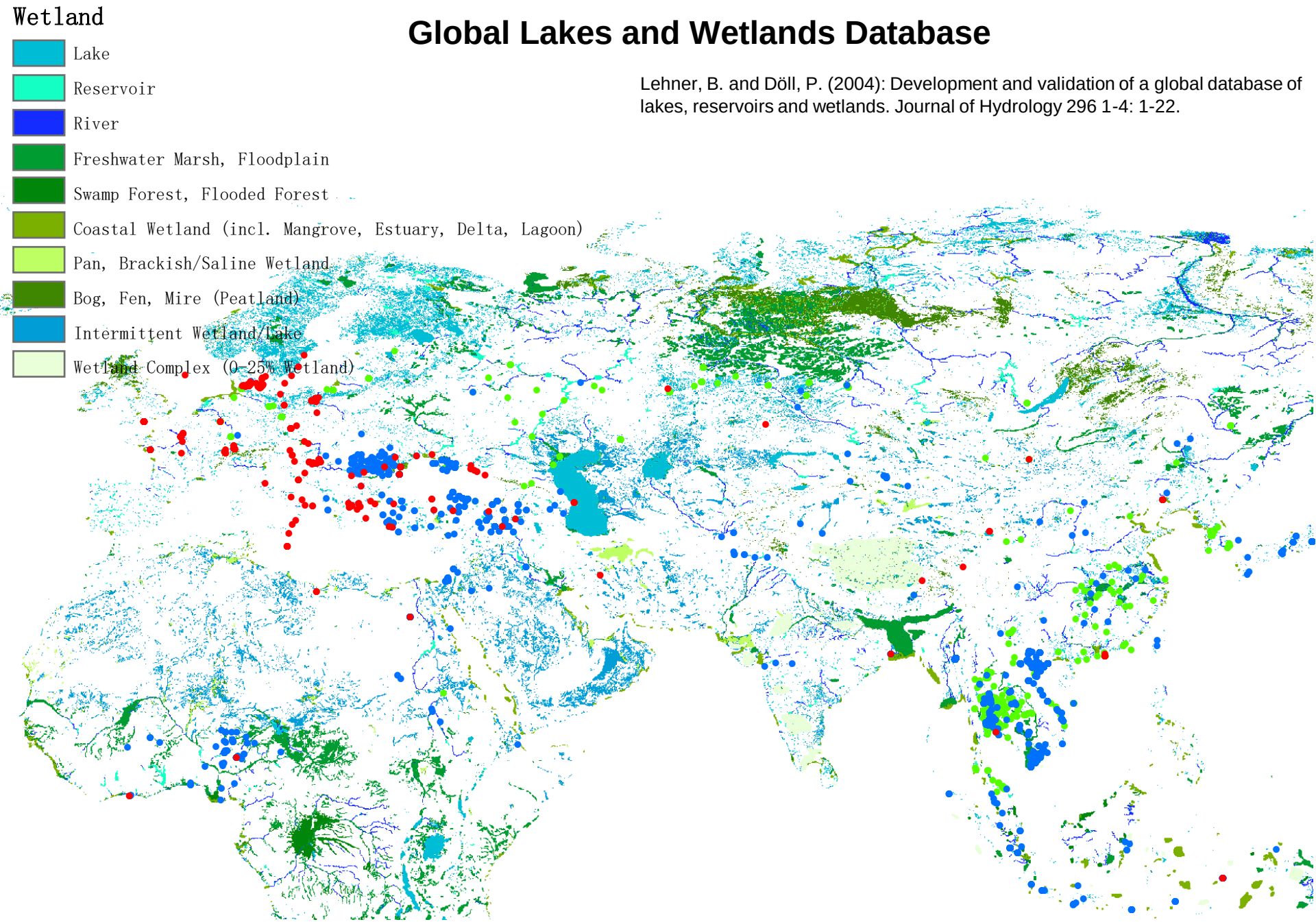
Human impact – Footprint

The index of human activities, the human footprint, was downloaded from the website: http://www.ciesin.columbia.edu/wild_areas/



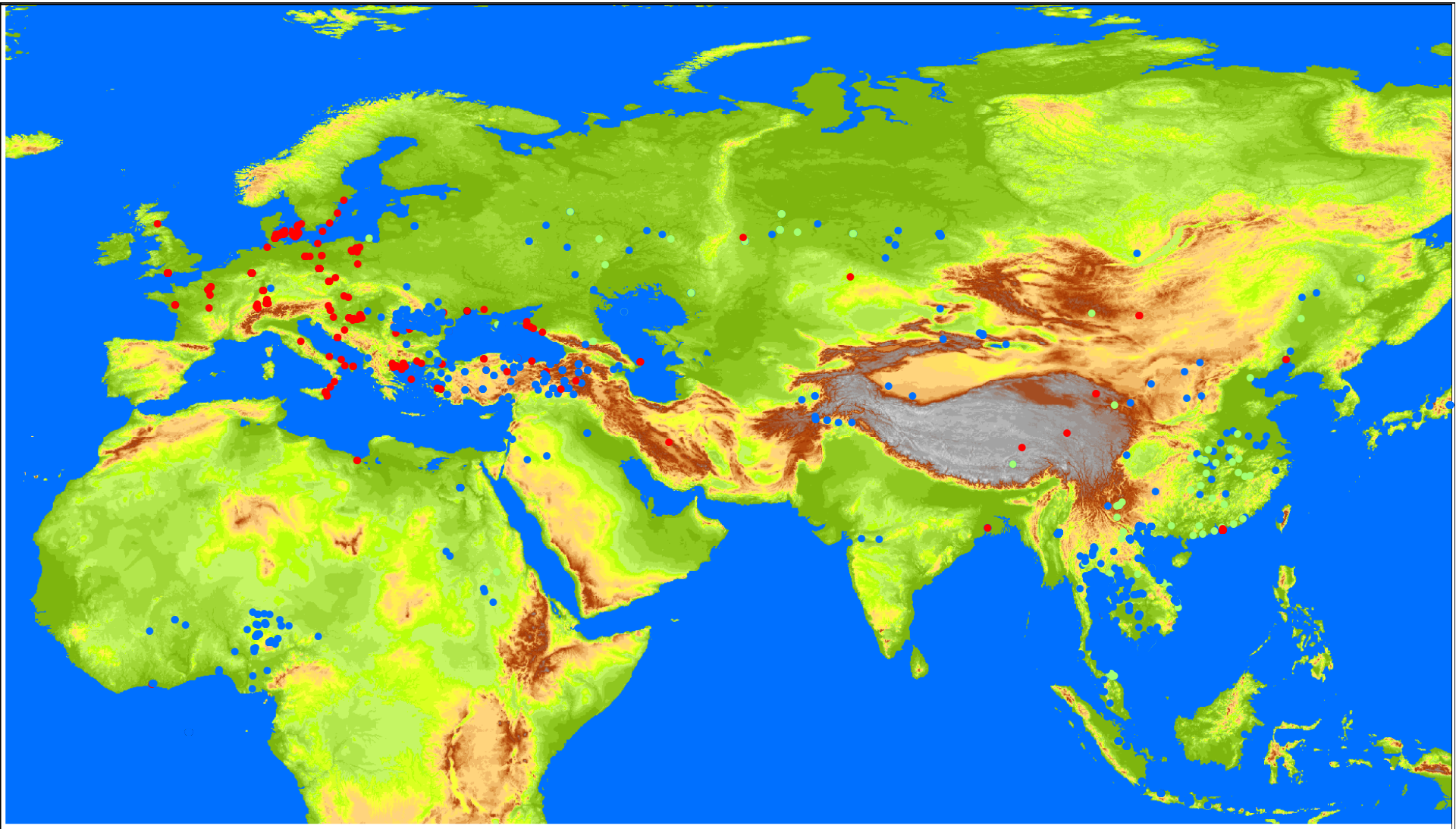
Global Lakes and Wetlands Database

Lehner, B. and Döll, P. (2004): Development and validation of a global database of lakes, reservoirs and wetlands. Journal of Hydrology 296 1-4: 1-22.



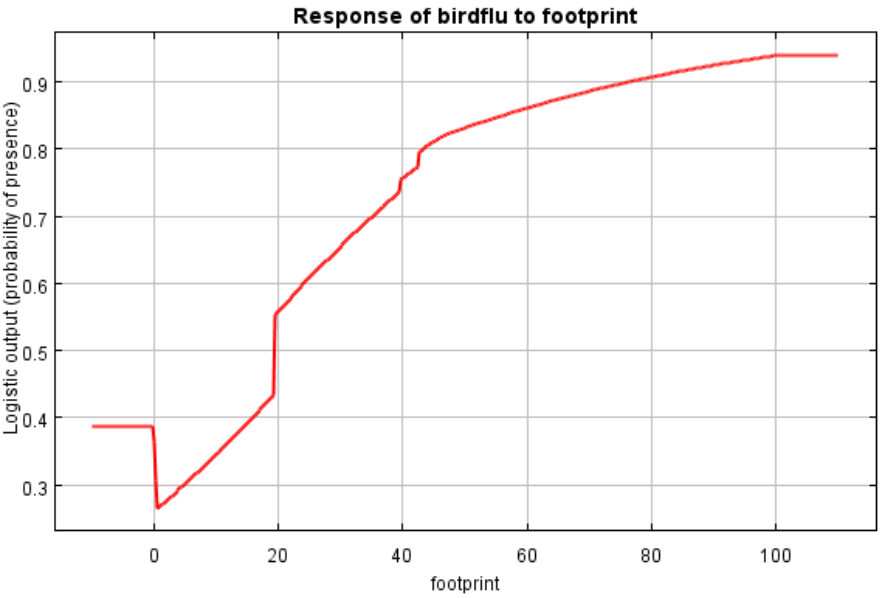
Elevation

The elevation data were downloaded from the US Geological Survey (USGS) website:
http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html



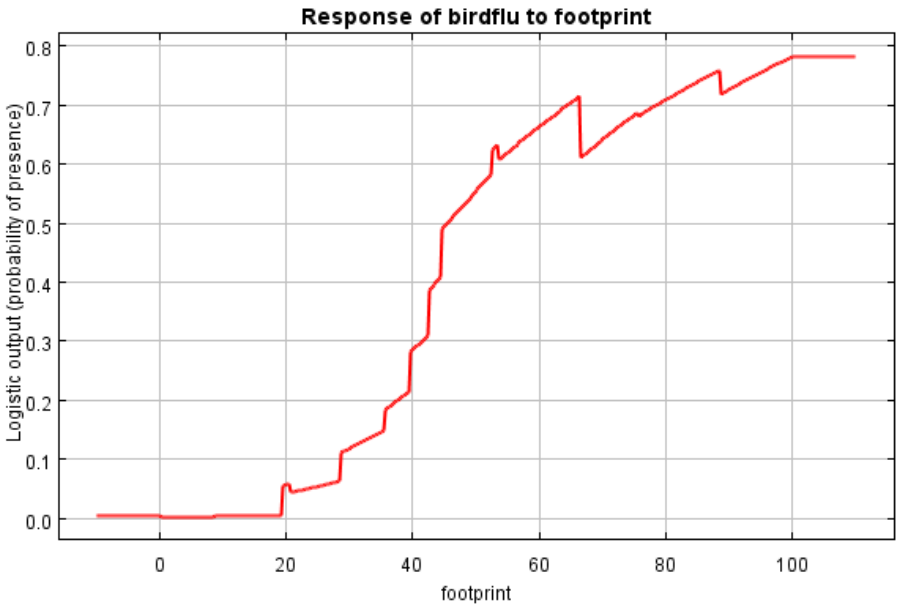
Response curves: human footprint index

Marginal response curves



The curves show how the logistic prediction changes as the environmental variable is varied, keeping all other environmental variables at their average sample value.

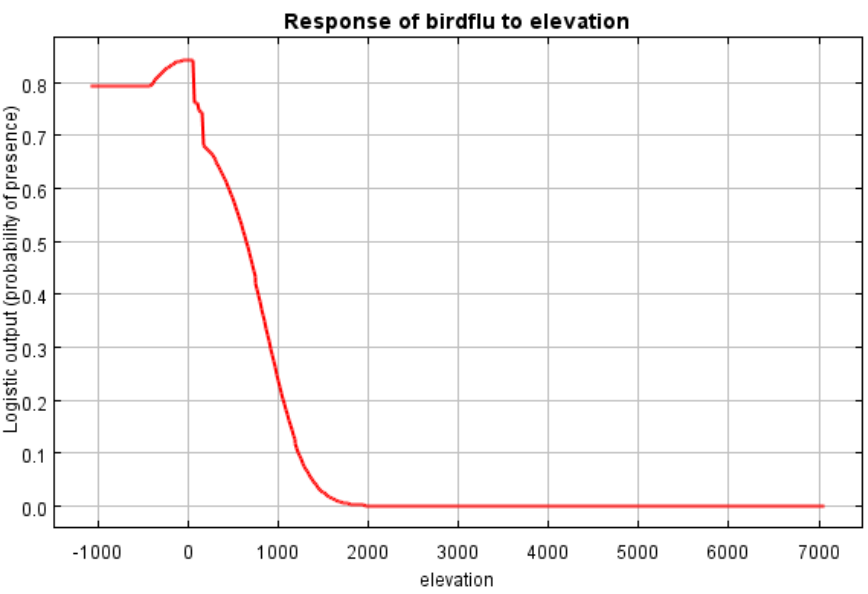
Maxent model



These plots reflect the dependence of predicted suitability both on the selected variable and on dependencies induced by correlations between the selected variable and other variables.

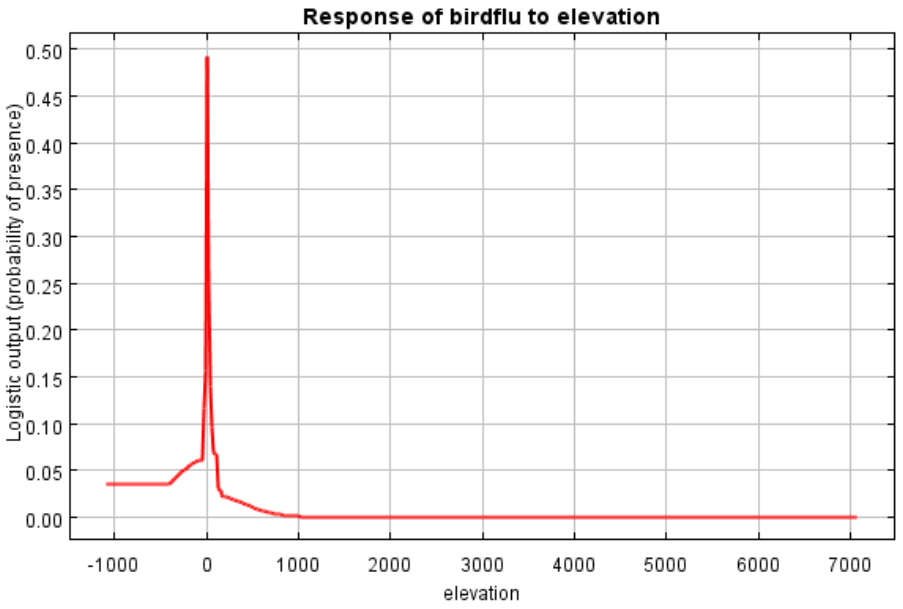
Response curves: elevation

Marginal response curves



The curves show how the logistic prediction changes as the environmental variable is varied, keeping all other environmental variables at their average sample value.

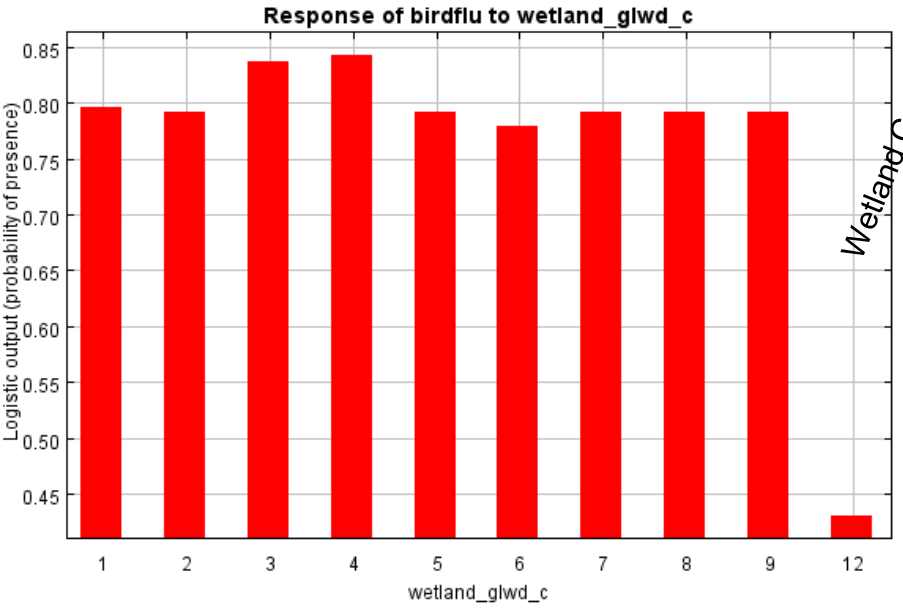
Maxent model



These plots reflect the dependence of predicted suitability both on the selected variable and on dependencies induced by correlations between the selected variable and other variables.

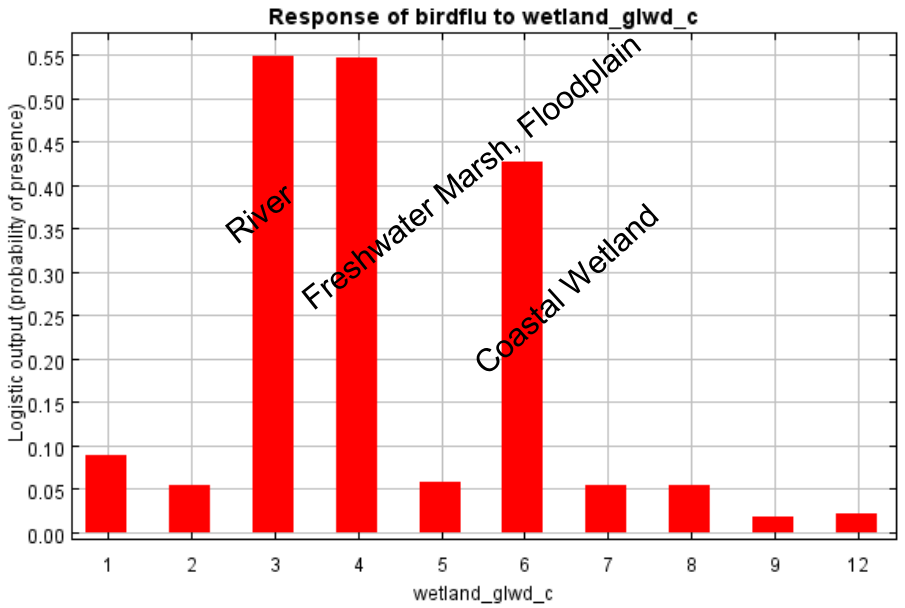
Response curves: wetland

Marginal response curves



The curves show how the logistic prediction changes as the environmental variable is varied, keeping all other environmental variables at their average sample value.

Maxent model



These plots reflect the dependence of predicted suitability both on the selected variable and on dependencies induced by correlations between the selected variable and other variables.

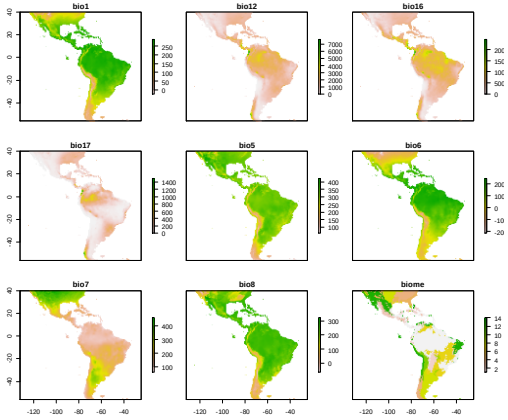
Analysis of variable contributions to HPAI (H5N1) occurrences using Maxent

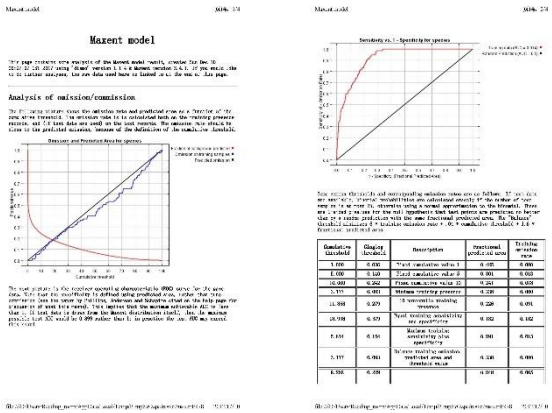
Variable	Percent contribution
footprint	64.6
prec_ann_sd	12.3
elevation	7.7
t_ann_q_sd	2.6
prec_ann	2.5
t_ann_min	2.2
t_ann_mean	2.0
ecosystem_c	2.0
wetland_glwd_c	1.8
aspect100	1.1
t_ann_max	0.6
landcover_c	0.4
t_ann_d_range	0.1
slope	0.1

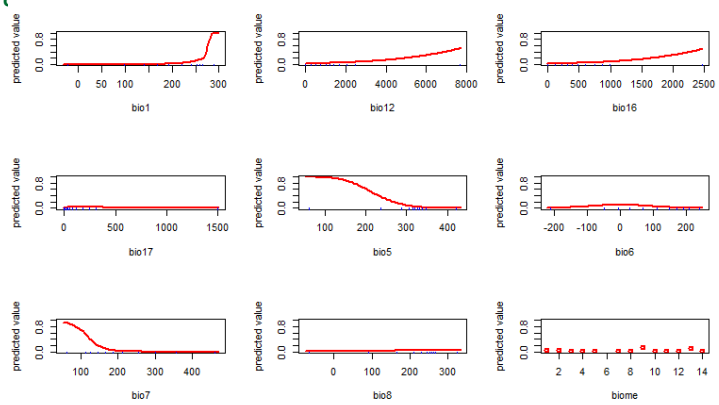
Li, X. H., H. D. Tian, M. Heiner, and D. M. Li. 2011. Global Occurrence and Spread of Highly Pathogenic Avian Influenza Virus of the Subtype H5N1. Avian Diseases 55:21-28.

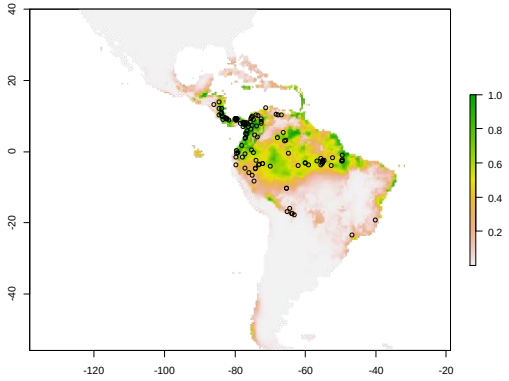
Maxent in R

```
Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre-9')
library(dismo); library(rJava)

 # get predictor variables
  fnames <- list.files(path=paste(system.file(package="dismo"), '/ex', sep=''),
                       pattern='grd', full.names=TRUE )
  predictors <- stack(fnames)
  plot(predictors)

  # file with presence points
  occurence <- paste(system.file(package="dismo"), '/ex/bradypus.csv', sep='')
  occ <- read.table(occurence, header=TRUE, sep=',')[,-1]

  # witholding a 20% sample for testing
  fold <- kfold(occ, k=5)
  occtest <- occ[fold == 1, ];   occtrain <- occ[fold != 1, ]

  # fit model, biome is a categorical variable
  me <- maxent(predictors, occtrain, factors='biome')

  # see the maxent results in a browser:
  me

  # plot showing importance of each va
  plot(me)

  # response curves
  response(me)

  # predict to entire dataset
  r <- predict(me, predictors)

  plot(r)
  points(occ)
```









Tuning Maxent

```
library(ENMeval)
### Simulated data environmental covariates
library(raster)
set.seed(1)
r1 <- raster(matrix(nrow=50, ncol=50, data=runif(10000, 0, 25)))
r2 <- raster(matrix(nrow=50, ncol=50, data=rep(1:100, each=100), byrow=TRUE))
r3 <- raster(matrix(nrow=50, ncol=50, data=rep(1:100, each=100)))
r4 <- raster(matrix(nrow=50, ncol=50, data=c(rep(1,1000),rep(2,500)),byrow=TRUE))
values(r4) <- as.factor(values(r4))
env <- stack(r1,r2,r3,r4)

### Simulate occurrence localities
nocc <- 50
x <- (rpois(nocc, 2) + abs(rnorm(nocc)))/11
y <- runif(nocc, 0, .99)
occ <- cbind(x,y)

### This call gives the results loaded below
enmeval_results <- ENMevaluate(occ, env, method="block", n.bg=500,
                      RMvalues = seq(0.5, 4, 0.5),
                      fc = c("L", "LQ", "H", "LQH", "LQHP", "LQHPT"))

data(enmeval_results)
enmeval_results

### See table of evaluation metrics (see next slide)
enmeval_results@results

### Plot prediction with lowest AICc
plot(enmeval_results@predictions[[which (enmeval_results@results$delta.AICc == 0) ]])
points(enmeval_results@occ.pts, pch=21, bg=enmeval_results@occ.grp)
```

Methods in Ecology and Evolution

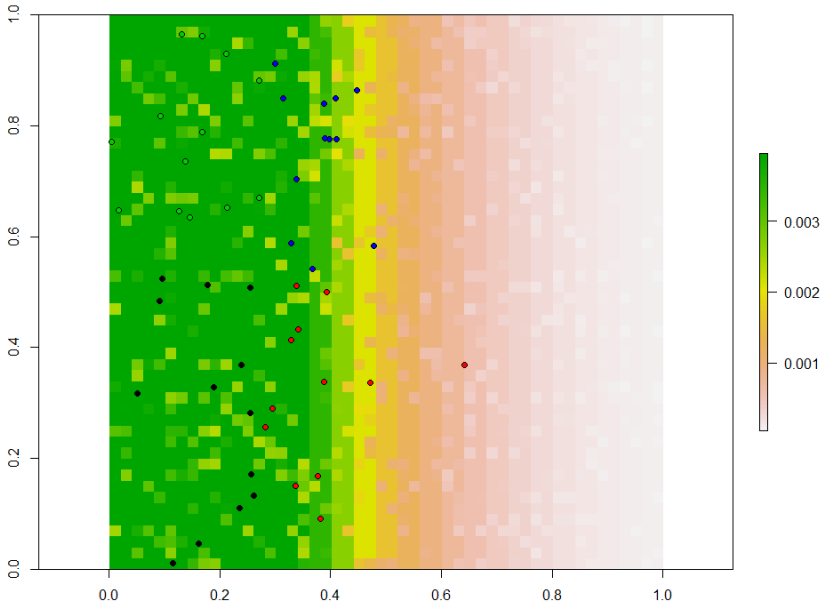
Methods in Ecology and Evolution 2014, 5, 1198–1205 doi: 10.1111/2041-210X.12261

APPLICATION

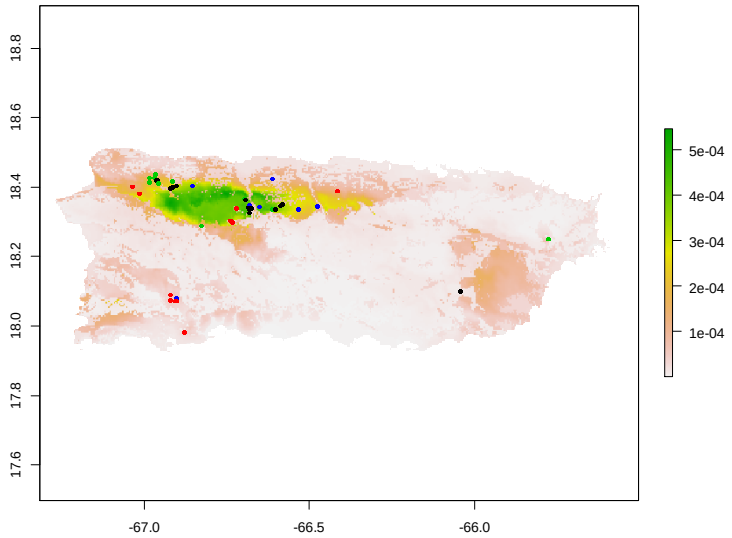
ENMeval: An R package for conducting spatially independent evaluations and estimating optimal model complexity for MAXENT ecological niche models

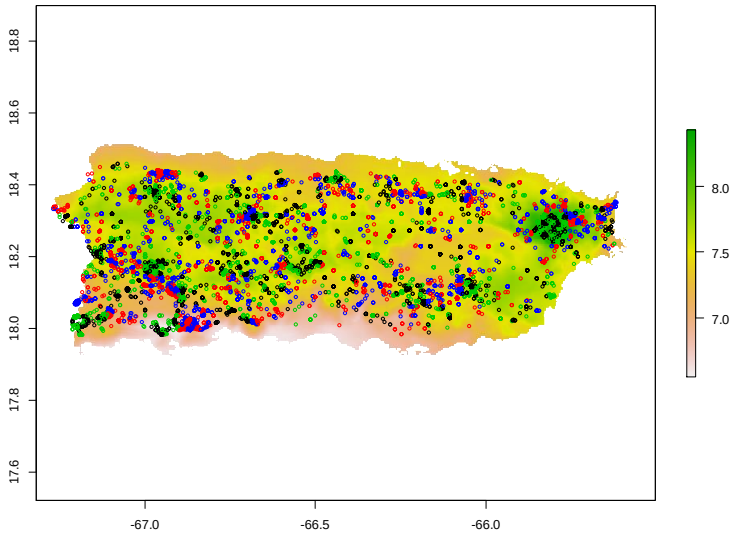
Robert Muscarella[1]*, Peter J. Galante[2], Mariano Soley-Guardia[2,3], Robert A. Boria[2], Jamie M. Kass[2,3], María Uriarte[1] and Robert P. Anderson[2,3,4]

[1]*Department of Ecology, Evolution and Environmental Biology, Columbia University, 1200 Amsterdam Ave., New York, NY 10027, USA;* [2]*Department of Biology, City College of the City University of New York, 160 Convent Ave., New York, NY 10031, USA;* [3]*Graduate Center of the City University of New York, 365 5th Ave., New York, NY 10016, USA; and* [4]*Division of Vertebrate Zoology (Mammalogy), American Museum of Natural History, Central Park West & 79th Street, New York, NY 10024, USA*



Feature class (FC) and regularization multiplier (RM)





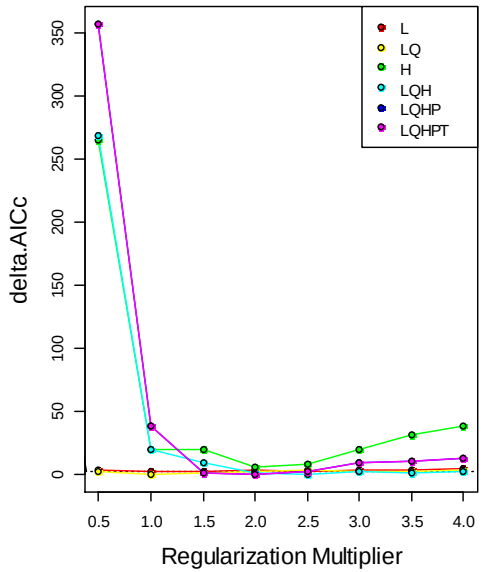
settings	features	rm	full.AUC	Mean.AUC	Var.AUC	Mean.AUC. DIFF	Var.AUC. DIFF	Mean. OR10	Var.OR10	Mean. ORmin	Var.ORmin	nparm	AICc	delta.AICc
L_0.5	L	0.5	0.75	0.7551	0.0523	0.0697	0.0210	0.1250	0.0625	0.0625	0.0156	4	754.218	25.730
LQ_0.5	LQ	0.5	0.8078	0.7972	0.0036	0.0190	0.0025	0.1218	0.0212	0.0208	0.0017	6	729.476	0.988
H_0.5	H	0.5	0.8313	0.6642	0.0435	0.1080	0.0117	0.2740	0.0410	0.1010	0.0151	18	861.367	132.879
LQH_0.5	LQH	0.5	0.832	0.6780	0.0480	0.0982	0.0086	0.2163	0.0373	0.0801	0.0086	17	856.283	127.795
LQHP_0.5	LQHP	0.5	0.8395	0.6489	0.0317	0.1301	0.0089	0.2740	0.0410	0.0994	0.0138	22	879.770	151.282
LQHPT_0.5	LQHPT	0.5	0.8446	0.7447	0.0071	0.1022	0.0155	0.2580	0.0202	0.0994	0.0138	17	845.904	117.416
L_1	L	1	0.7494	0.7529	0.0519	0.0705	0.0208	0.1042	0.0434	0.0625	0.0156	4	754.159	25.671
LQ_1	LQ	1	0.8018	0.7994	0.0095	0.0297	0.0039	0.1042	0.0249	0.0208	0.0017	6	731.289	2.801
H_1	H	1	0.8245	0.7085	0.0395	0.0886	0.0074	0.2981	0.0120	0.0609	0.0063	22	789.126	60.638
LQH_1	LQH	1	0.827	0.7407	0.0039	0.0912	0.0027	0.2388	0.0113	0.0833	0.0139	15	753.104	24.616
LQHP_1	LQHP	1	0.8306	0.7430	0.0033	0.0893	0.0029	0.2772	0.0164	0.1026	0.0111	17	759.155	30.667
LQHPT_1	LQHPT	1	0.8358	0.7351	0.0074	0.1042	0.0069	0.3189	0.0110	0.1026	0.0111	17	756.900	28.412
L_1.5	L	1.5	0.7488	0.7527	0.0502	0.0693	0.0198	0.1042	0.0434	0.0625	0.0156	4	754.142	25.654
LQ_1.5	LQ	1.5	0.7964	0.7931	0.0188	0.0422	0.0070	0.1250	0.0394	0.0417	0.0069	7	735.941	7.453
H_1.5	H	1.5	0.8185	0.7803	0.0012	0.0389	0.0034	0.2196	0.0053	0.0208	0.0017	15	755.492	27.004
LQH_1.5	LQH	1.5	0.8172	0.7667	0.0029	0.0543	0.0048	0.1603	0.0086	0.0625	0.0156	10	738.777	10.289
LQHP_1.5	LQHP	1.5	0.8193	0.7628	0.0012	0.0597	0.0035	0.2404	0.0041	0.0417	0.0069	14	751.206	22.718
LQHPT_1.5	LQHPT	1.5	0.8282	0.7646	0.0012	0.0600	0.0026	0.2788	0.0051	0.0417	0.0069	17	761.828	33.340

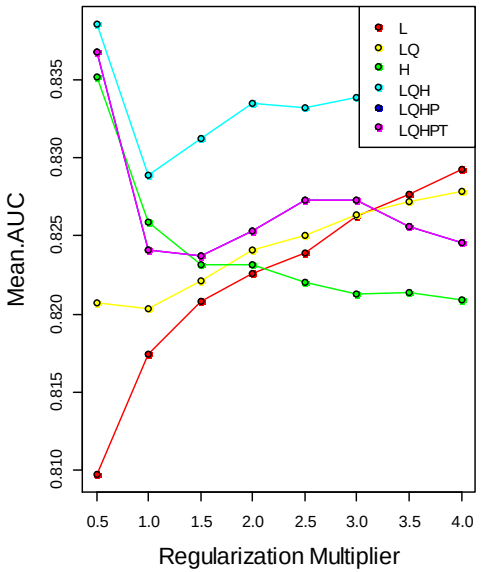
Feature class (FC) and regularization multiplier (RM)

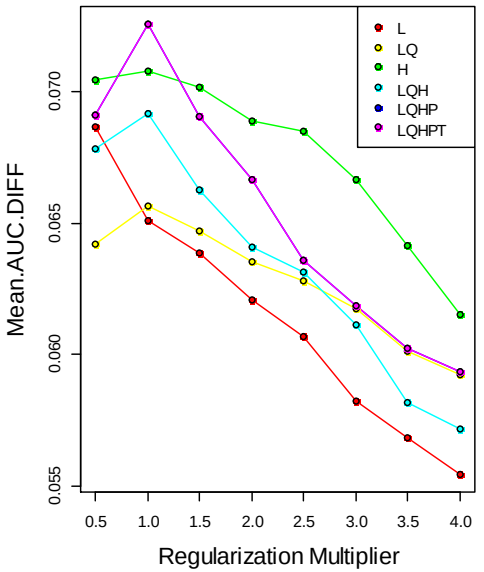
```
res <- ENMevaluate(occ, env, method = "block", n.bg = 500,
                   RMvalues = seq(0.5, 4, 0.5),
                   fc = c("L", "LQ", "H", "LQH", "LQHP", "LQHPT"))
```

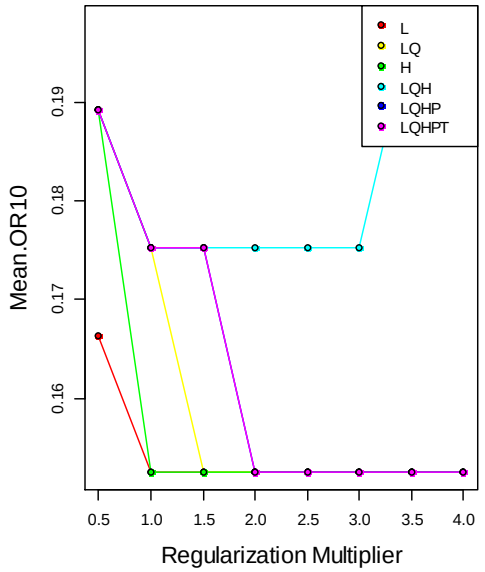
```
# Plot delta.AICc for different settings
par(mfrow=c(2,3))
eval.plot(res@results)
eval.plot(res@results, 'Mean.AUC')
eval.plot(res@results, 'Mean.AUC.DIFF')
eval.plot(res@results, 'Mean.OR10')
eval.plot(res@results, 'Mean.ORmin')

# How similar (Schoener's D) were the AICc-selected models?
res@overlap[aicmods, aicmods]

# Note that using the results table for plotting enables creativity via the full range of R plotting options. For example:
# Let's see how AUCtrain values compare with delta.AICc, depending on feature classes and regularization multiplier settings...
plot(res@results$Mean.AUC, res@results$delta.AICc, bg=res@results$features, pch=21, cex= res@results$rm/2, ylim=c(0,30))
legend("topright", legend=unique(res@results$features), pt.bg=res@results$features, pch=21)
mtext("Circle size proportional to regularization multiplier value")
```

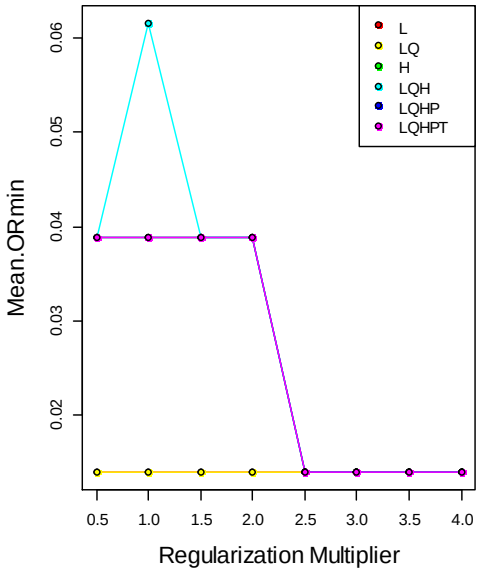
Feature class (FC) and regularization multiplier (RM)

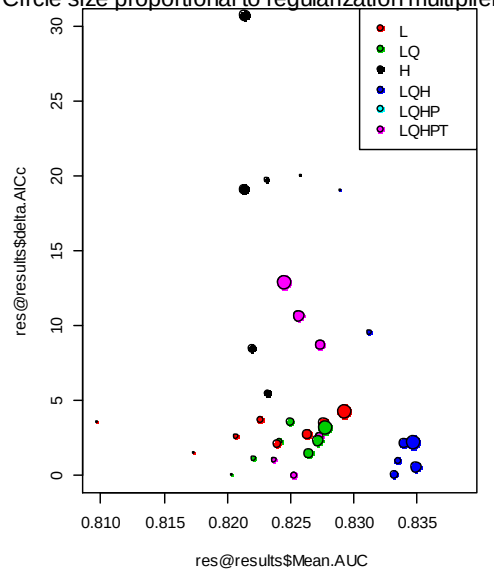












MaxEnt vs. MaxLike

MaxEnt generates an index of relative habitat suitability.

MaxLike, a newly introduced maximum-likelihood technique, directly estimates the probability of occurrence using presence-only data.

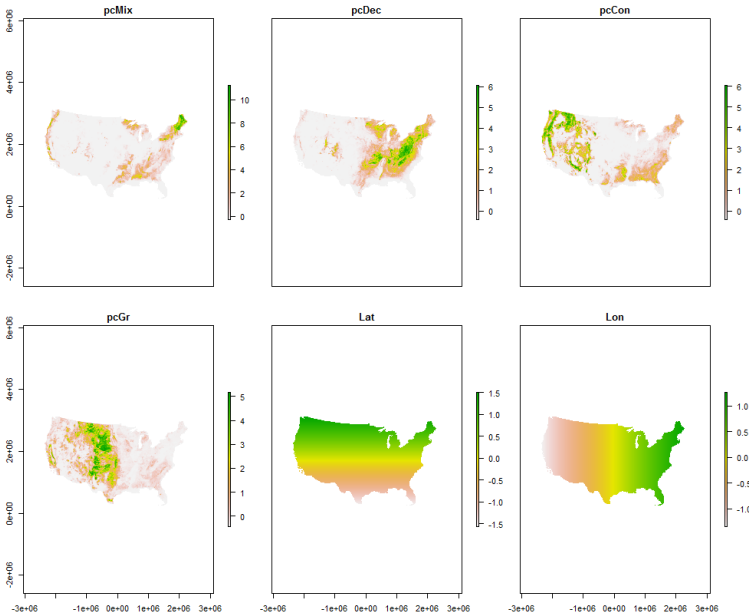
The performance of MaxLike exceeds that of MaxEnt.

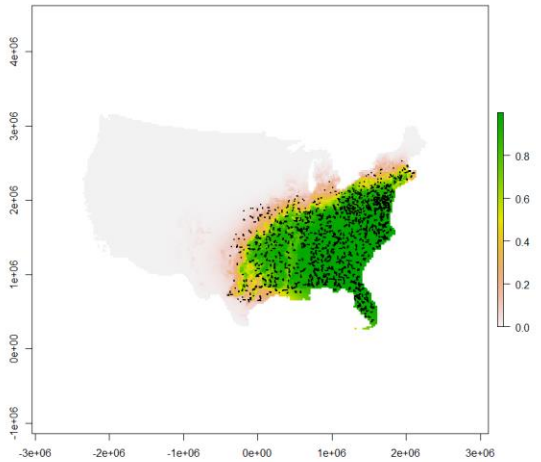
Package “maxlike” is available.

Fitzpatrick, M. C., N. J. Gotelli, and A. M. Ellison. 2013. MaxEnt versus MaxLike: empirical comparisons with ant species distributions. Ecosphere **4**:art55.
http://www.esajournals.org/doi/abs/10.1890/ES13-00066.1

R code for maxlike

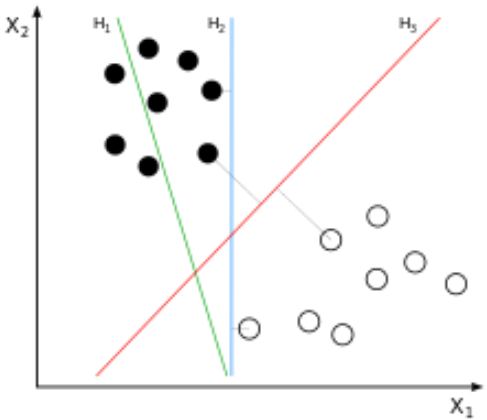
```
library(maxlike)
# Carolina Wren data used in Royle et. al (2012)
data(carw)

# Covert data.frame to a list of rasters
rl <- lapply(carw.data$raster.data, function(x) {
  m <- matrix(x, nrow=carw.data$dim[1], ncol=carw.data$dim[2], byrow=TRUE)
  r <- raster(m)
  extent(r) <- carw.data$ext
  r
})

# Create a raster stack and add layer names
rs <- stack(rl[[1]], rl[[2]], rl[[3]], rl[[4]], rl[[5]], rl[[6]])
names(rs) <- names(carw.data$raster.data)

plot(rs)

# Fit a model
fm <- maxlike(~pcMix + I(pcMix^2) + pcDec + I(pcDec^2)+ pcCon +
              I(pcCon^2) + pcGr + I(pcGr^2) +
              Lat + I(Lat^2) + Lon + I(Lon^2), rs, carw.data$xy1,
            method="BFGS", removeDuplicates=TRUE, savedata=TRUE)

summary(fm)
confint(fm)
AIC(fm)
logLik(fm)

# Produce species distribution map (ie, expected probability of occurrence)
psi.hat <- predict(fm) # Will warn if savedata=FALSE
plot(psi.hat)
points(carw.data$xy1, pch=16, cex=0.1)
```





Support vector machine (SVM)

- A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks.
- Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.
- The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes.
- The SVM is only directly applicable for two-class tasks. For multi-class tasks, several binary algorithm have to be applied.



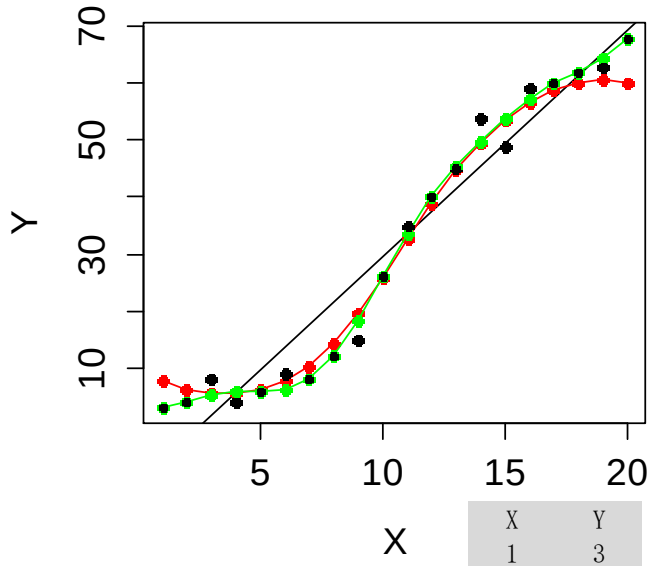
Vapnik, V., and A. Lerner, 1963. Pattern recognition using generalized portrait method. *Automation and Remote Control*, **24**, 774–780.

The algorithm implemented by support vector machines is a nonlinear generalization of the Generalized Portrait algorithm).

Support vector machine (SVM)

http://www.svm-tutorial.com/2014/10/support-vector-regression-r/

```
# Plot the data, Create a linear regression model
X = 1:20;  Y = c(3,4,8,4,6,9,8,12,15,26,35,40,45,54,49,59,60,62,63,68)
data = data.frame(X, Y)
plot(data, pch=16); abline(lm(Y ~ X, data))

# Support vector machine (SVM)
library(e1071) # svm()
model <- svm(Y ~ X , data)
predictedY <- predict(model, data)
points(data$X, predictedY, col = "red", pch=16)
lines(  data$X, predictedY, col = "red", pch=16)

# Select the best parameters for the SVM, perform a grid search
# we performed an epsilon-regression, it took a default value of 0.1.
# There is also a cost parameter which we can change to avoid overfitting.
tuneResult <- tune(svm, Y ~ X,  data = data,
                   ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)))
print(tuneResult); plot(tuneResult)
# Improved prediction
tunedModel <- tuneResult$best.model
tunedModelY <- predict(tunedModel, data)
points(data$X, tunedModelY, col = "green", pch=16)
lines  (data$X, tunedModelY, col = "green")
```



X	Y
1	3
2	4
3	8
4	4
5	6
6	9
7	8
8	12
9	15
10	26
11	35
12	40
13	45
14	54
15	49
16	59
17	60
18	62
19	63
20	68

Assignment

General objectives: learn random forest.

- Develop a dataset to perform:
  - random forest Y - X1, X2, X3, etc.
- Describe the dataset, define the type of analysis (multiple regression, or logistic regression, or discriminant analysis), demonstrate the weight of variables, plot the partial effect of variables, describe the overall model fit.