Zukünftige Risikogebiete von Zoonosen

Einbindung ökologischer Zusammenhänge in die Verbreitungsmodellierung

„Zoonose RISKTOOL“

Dokumentation mit Beispieldaten

Stand: 20. August 2015

Forschungskennzahl:

01KI1302

biogeografie
uni bayreuth

Prof. Dr. Carl Beierkuhnlein

Dr. Anja Jaeschke

Nils Tjaden

Dr. Stephanie Thomas

Lehrstuhl Biogeografie

Universität Bayreuth

Voraussetzungen

Installation von R

Das Risktool basiert auf der freien Programmiersprache R. Grundkenntnisse im Umgang mir R werden vorausgesetzt. Es empfiehlt sich, jeweils die aktuellste Version von R über das *Comprehensive R Archive Network* (CRAN, https://cran.r-project.org/) zu beziehen und zu installieren. Für Debian-basierte Linux-Systeme bietet CRAN eigene Paketquellen an, die gegenüber den systemeigenen Paketquellen oft aktuellere Versionen bieten. Siehe https://cran.r-project.org/bin/linux/ubuntu/README für weitere Details zur verwendung dieser Paketquellen.

IDE installieren

Zum komfortableren Umgang mit R empfiehlt es sich, zusätzlich zu R selbst eine Integrierte Entwicklungsumgebung zu installieren. Je nach Betriebssystem steht eine unterschiedliche Auswahl zur Verfügung, unter anderem RStudio (alle Systeme), Tinn-R (Windows), RKWard (KDE /Linux) und StatET (R-Plugin für eclipse, Multiplatform).

RISKTOOL herunterladen

Das Risktool kann frei über die RISKTOOL-Website bezogen werden. Dazu im Downloadbereich der Website die Datei `RISKTOOL_example.zip` herunterladen und an einem beliebigen Ort entpacken. Das dabei entstehende Verzeichnis RISKTOOL_example wird von diesem Tutorium als Arbeitsverzeichnis genutzt. Das Unterverzeichnis data enthält bereits einen Beispieldatensatz. Sämtlicher R-Code ist im Unterverzeichnis code zu finden und in der in diesem Tutorium verwendeten Reihenfolge nummeriert. Der Beispielcode ist umfangreich kommentiert und enthält zahlreiche Zusatzinformationen, die hier aufgrund der limitierten Zeilenlänge nur schwer unterzubringen gewesen wären. Beispielcode und Tutorium sind so konzipiert, dass sie einander ergänzen. Es empfiehlt sich daher, den Beispielcode Zeile für Zeile auszuführen und parallel den Erklärungen im Tutorium zu folgen.

Maxent installieren

Soll neben den in R integrierten Modellalgorithmen auch das verbreitete Maxent verwendet werden, so muss dies als externe Software heruntergeladen werden. Dazu wird die aktuelle version der Software von der Website der Princeton-University heruntergeladen und entpackt. Entscheidend ist dabei die im Download enthaltene datei `maxent.jar`, die am besten direkt im Arbeitsverzeichnis (hier: `…/RISKTOOL_example/`) platziert wird. Diese Datei muss ggf. als „ausführbar“ markiert werden! Außerdem wird ein Java Runtime Environment (JRE) benötigt. Für weitere Informationen zu Maxent siehe z.B. A Brief Tutorial on Maxent (.doc).

Tutorium

Wir werden nun die einzelnen Teilskripte des Beispielpaketes der Reihe nach durchgehen.

Teilskript 0: Installation von Zusatzpaketen und Grundeinstellung

In diesem Teilskript werden zunächst benötigte Zusatzpakete für R installiert. Hierzu ist eine Internetverbingung erforderlich. Es hadelt sich hier im Einzelnen um:

- `downloader` – wird hier zum Herunterladen von Beispieldaten benötigt
- `sp, rgdal, raster` – zur Verarbeitung von Geodaten
- `ncfd4` und `ncdf4.helpers` – zum Einlesen von Klimadaten im NetCDF-Format
- `biomod2` – zur Artverbreitungsmodellierung
- `hier.part` – zur Variablenselektion für die Artverbreitungsmodellierung
- `vegan` – die hierin enthaltene Funktion `decostand` wird zur Normierung von Daten verwendet

```
install.packages(c("downloader", "sp", "rgdal", "raster", "biomod2",
"hier.part", "ncdf4", "ncdf4.helpers", "vegan"), dependencies=TRUE)
```

Hinweis: Auf Linux-Systemen können zusätzliche externe Abhängigkeiten bestehen, wenn Pakete aus dem Quellcode kompiliert werden muessen. Auf Debian-basierten Systemen (Debian, Ubuntu, Linux Mint ...) empfiehlt es sich vorher zu ueberpruefen ob "build-essential" installiert ist und dies ggf. nachzuholen (sudo apt-get install build-essential).

Anschließend wird das Arbeitsverzeichnis („working directory“) für das Modellierungsprojekt angegeben. Dies muss individuell an die lokale Verzeichnisstruktur angepasst werden und muss im Falle dieses Tutoriums auf das Verzeichnis `RISKTOOL_example` zeigen. Alle relativen Pfadangaben in R werden von nun an dieses Arbeitsverzeichnis als Wurzel haben.

```
setwd("C:/Users/Max\ Mustermann/Projekte/RISKTOOL_example/")
```

Hinweis: Der unter Windows zur Trennung von Unterverzeichnissen verwendete Backslash (\) muss in den auf Unixoiden Systemen üblichen normalen Schrägstrich (/) geändert werden. Leerzeichen in Pfaden sollten vermieden werden, wenn unvermeidbar wird ihnen ein Backslash zur Maskierung vorangestellt („\ „).

Abschließend wird das Konfigurationsskript `risktool-settings.r` aufgerufen. Dieses Skript, das wie alle anderen im Unterverzeichnis „code“ liegen muss, erfüllt folgende Aufgaben:

- Laden der zuvor installierten Zusatzpakete
- Laden der in `risktool-functions.r` bereitgestellten Risktool-Funktionen
- Anlage der benötigten Verzeichnisstruktur unterhalb von `RISKTOOL_example`, sofern diese nicht bereits existiert
- Definition Namen der in der Verbreitungsmodellierung verwendeten Klimavariablen

- Definition des Namens der in der Verbreitungsmodellierung untersuchten Art
- Definition des geografischen Referenzsystems der Vorkommensdaten für die Verbreitungsmodellierung.

Zur Durchführung dieses Tutoriums sind keine Änderungen am Konfigurationsskript erforderlich.

```
source("./code/risktool-settings.r")
```

Teilskript 1: Herunterladen und Aufbereitung der für dieses Tutorium benötigten Beispieldaten

Download von Daten für die Verbreitungsmodellierung

Wir werden hier beispielhaft mit der Mückenart *Aedes japonicus* in ihrem Herkunftsgebiet Japan arbeiten. Verbreitungsdaten für diese Art wurden bereits mit dem Beispielcode heruntergeladen. Sie befinden sich im Unterverzeichnis `./daten/presences/` in der Datei `Ae_japonicus.csv`.

Weiterhin werden Klimadaten für gegenwärtige („baseline“) Klimabedingungen benötigt. Wir verwenden hier den vielfach genutzten „Bioclim“ Datensatz, der unter worldclim.com kostenlos zur Verfügung steht. Bitte beachten: Diese Daten dürfen nur für nichtkommerzielle Zwecke verwendet und nicht ohne Einverständnis der Anbieter weiterverbreitet werden (für Details siehe Anbieterwebsite). Das Herunterladen und entpacken der Daten an den dafür vorgesehenen Ort kann wie folgt automatisiert werden:

```
download.file(url =
"http://biogeo.ucdavis.edu/data/climate/worldclim/1_4/grid/cur/bio_5m_esri.zip",
              destfile=paste0(path.envir.baseline, "bio_5m_esri.zip"),
              method="internal", mode="w")
unzip(zipfile=paste0(path.envir.baseline, "bio_5m_esri.zip"),
      exdir=path.envir.baseline)
```

Für Zukunftsprojektionen benötigen wir außerdem Klimadaten für zukünftige („future“) Klimabedingungen. Wir verwenden dazu in unserem Beispiel Daten des Klimamodells HadGEM2-AO für das IPCC-Szenario RCP 4.5 und den Zeitabschnitt 2031-2050. Diese können ebenfalls über worldclim bezogen werden (Details siehe Anbieterwebsite):

```
download.file(url =
"http://biogeo.ucdavis.edu/data/climate/cmip5/5m/hd45bi50.zip",
              destfile=paste0(path.envir.future, "hd45bi50.zip"),
              method="internal", mode="w")
unzip(zipfile=paste0(path.envir.future, "hd45bi50.zip"),
      exdir=path.envir.future)
```

Da alle diese Daten eine globale Abdeckung haben, wir uns aber nur für Japan interessieren, möchten wir die Daten auf unser Untersuchungsgebiet zuschneiden, um bei der Modellierung Rechenzeit zu sparen. Dazu laden wir ein Shapefile mit den Umrissen Japans herunter. Auch diese diese Daten dürfen nur für nicht-kommerzielle Zwecke verwendet werden, für Details siehe die im Download enthaltene Datei `read_me.pdf` und die Anbieterwebsite.

```
download.file(url="http://biogeo.ucdavis.edu/data/gadm2/shp/JPN_adm.zip",
              destfile=paste0(path.misc, "JPN_adm.zip"),
              method="internal", mode="w")
unzip(zipfile=paste0(path.misc, "JPN_adm.zip"), exdir=path.misc)
```

Download von Klimadaten in täglicher Auflösung für über die Verbreitungsmodellierung hinausgehende, auf experimentellen Daten beruhende Risikoabschätzung.

Projizierte Temperaturdaten in täglicher Auflösung müssen manuell heruntergeladen werden. Diese sind fuer reine Verbreitungsmodellierung eines Vektors NICHT erforderlich. Wir werden in diesem Beispiel der Einfachheit halber nur mit Mitteltemperaturen (nicht Minimum oder Maximum) arbeiten, deren Verarbeitung erfolgt aber analog zu dem hier Dargestellten.

Wir verwenden hier Daten des Klimamodells MPI-ESM-LR, wieder für das Klimaszenario RCP45 und den Zeitabschnitt 2031-2050. Diese Daten können vom Deutschen Klimarechenzentrum (DKRZ) heruntergeladen werden (es ist jeweils die Anlage eines Benutzeraccounts nötig):

- aktuelle Daten ueber den DKRZ-Datenknoten der Earth System Grid Foundation (ESGF) http://esgf-data.dkrz.de/
- alternativ archivierte Daten über das World Data Center for Climate Data http://cera-www.dkrz.de/ (Link zum Direkteinstieg)

Herguntergeladen werden müssen die Folgenden Dateien:

- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20300101-20391231.nc
- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20400101-20491231.nc
- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20500101-20591231.nc

Hinweis: Es wird erwartet, dass diese im Unterverzeichnis
`".../RISKTOOL_example/data/environment/daily_temperature/"`
abgelegt werden, andernfalls muss der Pfad für path.tas in risktool-settings.r angepasst werden.

Hinweis: Diese Daten haben eine verhältnismaessig geringe Räumliche Auflösung. Daten mit höherer Aufloesung stellen z.B. die regionalen CORDEX-Projekte zur Verfügung (http://www.cordex.org/) Die Verarbeitung der Daten erfolgt analog zu den hier verwendeten, benötigt aber unter Umstaenden erheblich mehr Festplattenspeicher und Rechenzeit.

Zuschneiden der Klimadaten für die Verbreitungsmodellierung auf das Untersuchungsgebiet

Zunächst wird das zum Zuschnitt verwendete Shapefile mit den Umrissen Japans geladen:

```
cropshape <- readOGR(dsn=substr(path.misc, start=1, stop=nchar(path.misc)-1),
                     layer="JPN_adm0" )
```

Der Zuschnitt erfolgt dann mittels der Risktool-Funktion `rt.clip`:

```
# baseline-Klimadaten zuschneiden
for(varname in 1:19){
  dat <- raster(paste0(path.envir.baseline, "bio/bio_", varname))
  dat.crop <- rt.clip(data=dat, cropshape=cropshape)
  writeRaster(x=dat.crop,
              filename=paste0(path.envir.baseline.cropped,
                              "bio_", varname, ".tif"))
}
# Zukunfts-Klimadaten zuschneiden
for(varname in 1:19){
  dat <- raster(paste0(path.envir.future, "/hd45bi50", varname, ".tif"))
  dat.crop <- rt.clip(data=dat, cropshape=cropshape)
  writeRaster(x=dat.crop,
              filename=paste0(path.envir.future.cropped,
              "bio_", varname, ".tif"))
```

Die fertig zugeschnittenen Daten finden sich dann in den entsprechenden Unterverzeichnissen:

- `./data/environment/bioclim_baseline/cropped/`
- `./data/environment/bioclim_future/cropped/`

Teilskript 2: Variablenselektion für Verbreitungsmodellierung

Zunächst werden die im CSV-Format vorliegenden Vorkommenstdaten eingelesen und in ein georeferenziertes SpatialPoints Datenobjekt umgewandelt:

```
pres <- read.csv(file=paste0(path.speciesdata, "Ae_japonicus_Japan.csv"),
                 header=TRUE, sep=";", dec=".")
summary(pres) # Zusammenfassung der Daten
# Extrahieren der Spalten "longitude" und "latitude" (d.h. die Koordinaten)
pres <- pres[,c("longitude", "latitude")]
# Entferne Zellen, die NA (d.h. keine Information) enthalten:
pres <- na.omit(pres)
# Konvertierung in SpatialPoints Objekt mit der vorgegebenen
# Projektion (proj4)
pres.sp <- SpatialPoints(pres, proj4string=proj4.pres)
```

Zur Variablenselektion benötigen wir für jede Rasterzelle unser Umweltdaten die Information ob die Art in dieser Zelle gefunden wurde. Wir wandeln daher die unregelmäßig verteilten Punkt-Verbreitungsdaten in ein entsprechendes Raster um:

```
# einen Umwelt-Layer als Referenz einladen
env.tmp <- raster(paste0(path.envir.baseline.cropped, envlayers[1], ".tif"))
pres.grd <- rasterize(pres.sp, env.tmp)
pres.grd <- as(pres.grd, ("SpatialGridDataFrame"))
```

In diesem Raster wurden alle Pixel, in denen die Art nicht vorkommt, auf "NA" (Not Available, nicht verfügbar) gesetzt. Die Pixel, in denen die Art vorkommt, haben den gleichen Wert wie das zuvor geladene Umweltraster an dieser Stelle hatte. Da wir nur die Information benoetigen, dass die Art vorkommt, ändern wir diesen Wert in "1" und die NAs in "0".

```
pres.grd@data$layer[which(!is.na(pres.grd@data$layer))] <- 1
pres.grd@data$layer[which(is.na(pres.grd@data$layer))] <- 0
```

Für die Variablenselektion benötigen wir nur einen Vektor mit 1/0 in der korrekten Reihenfolge. Diesen erhält man durch einfache Extraktion des Datenstrangs aus dem Raster.

```
pres.vector <- pres.grd@data$layer
```

Nun können wir die eigentlichen Umweltdaten einlesen. Die geografische Lage der einzelnen Rasterzellen ist hier noch nicht von Bedeutung, wir können daher erneut auf eine einfache Extraktion der Datenstränge zurückgreifen und diese zu einer einzigen großen Tabelle (data.frame) vereinen.

```
# Einlesen der Datenreihe der ersten Variable
env.selection <- c(readGDAL(paste(path.envir.baseline.cropped, envlayers[1],
                                  ".tif", sep=""))@data$band1)
# Konvertierung des Vektors in einen data.frame
env.selection <- data.frame(env.selection)
# Anfügen der Daten der übrigen Umweltvariablen mittels einer for-Schleife
for(layer in envlayers[2:length(envlayers)]){
  # Einlesen der Daten vom jeweils naechsten Layer im Datensatz
  dat.temp <- readGDAL(paste(path.envir.baseline.cropped, layer,
                             ".tif", sep=""))@data$band1
  # ... und Anfügen dieser an den bestehenden dataframe
  env.selection <- cbind(env.selection, dat.temp)
}
# Ergaenzen von geeigneten Namen für die einzelnen Spalten mit names()
names(env.selection) <- envlayers
```

Gebiete (Rasterzellen/Pixel), in denen keine Klimadaten vorliegen (z.B. Ozeane) enthalten "NA" statt Daten. Diese werden hier verworfen – sowohl im im Vektor mit den Vorkommen als auch in der Tabelle mit den Umweltvariablen:

```
pres.vector <- subset(pres.vector, !is.na(env.selection)[,1])
env.selection <- subset(env.selection, !is.na(env.selection)[,1])
```

Es folgt die eigentliche Variablenauswahl mittels hierarchischer Partitionierung. Dabei können systembedingt maximal 12 Variablen berücksichtigt werden! Eine auf Expertenwissen beruhende Vor-Auswahl ist daher nötig. Wir haben uns hier nach reiflicher Überlegung entschieden, uns auf die folgenden Variablen zu beschränken:

- bio_1
- bio_5 und 6
- bio_8 bis 11
- bio_14
- bio_16 bis 18

```
varimportance <- hier.part(pres.vector,
                           env.selection[,c(1, 5:6, 8:11, 14, 16:18)],
                           fam = "binomial",
                           gof = "logLik" )
```

Diese Berechnungen können durchaus eine geraume Zeit in Anspruch nehmen. Anschließend können wir uns den prozentualen Anteil der einzelnen Variablen an der erklaerten Varianz ausgeben lassen:

```
varimportance$I.perc
```

Als wichtigste Variablen im Beispiel stellen sich dabei heraus:

- bio_1
- bio_6
- bio_10
- bio_11

Diese 4 Variablen werden wir in die folgende Artverbreitungsmodellierung einfließen lassen.

Teilskript 3: Artverbreitungsmodellierung

Vorbereitungen

Zunächst lesen wir die zuvor ausgewählten Umweltdaten ein, dieses Mal mit der vollen geografischen Information als „RasterStack“ (gewissermaßen ein „Stapel“ mehrer Raster-Layer)

```
# Variablen festlegen
envir.names <- c("bio_1", "bio_6", "bio_10", "bio_11")
# Einlesen der Umweltdaten für aktuelle Klimabedingungen in ein "RasterStack"
envir.baseline <- stack(paste0(path.envir.baseline.cropped,
                               envir.names, ".tif"))
# Einlesen der Umweltdaten fuer zukünftige Klimabedingungen
envir.future <- stack(paste0(path.envir.future.cropped,
                             envir.names, ".tif"))
```

Erneutes Einlesen der Verbreitungsdaten und ggf. Entfernen von fehlenden Daten:

```
presences <- read.csv(paste0(path.speciesdata, "Ae_japonicus_Japan.csv"),
                      header=TRUE, sep=";", dec=".")
presences <- presences[,c("longitude", "latitude")]
presences <- na.omit(presences)
```

Wir könnten nun die Vorkommensdaten direkt in ein SpatialPoints-Objekt umwandeln und mit der Modellierung beginnnen. Wir möchten aber sicher gehen, dass wir keine Duplikate (mehrere Vorkommenspunkte innerhalb einer Rasterzelle der Umweltdaten) in unseren Daten haben. Wir gehen daher den Umweg über vorgeschaltete eine Rasterisierung der Vorkommensdaten.

```
presences.raster <- rasterize(presences, envir.baseline[[1]])
presences.raster[!is.na(presences.raster)] <- 1
presences.points <- as(presences.raster, "SpatialPointsDataFrame")
```

Wir verwenden nun die biomod2-eigene Funktion `BIOMOD_FormatingData`, um unsere Eingangsdaten für die Modelierung zu Formatieren:

```
myBiomodData <- BIOMOD_FormatingData(resp.var = presences.points,
                                     expl.var = envir.baseline,
                                     resp.name = speciesname,
                                     PA.nb.rep = 1,
                                     PA.nb.absences = 580,
                                     PA.strategy = "random")
```

Nun werden die Modellierungseinstellungen getätigt. Im Normalfall können einfach die Standardeinstellungen von biomod2 verwendet werden:

```
myBiomodOption <- BIOMOD_ModelingOptions()
```

Bei Verwendung von MAXENT müssen die Einstellungen jedoch angepasst werden. Erstens muss der Dateipfad zu maxent.jar definiert werden – wir gehen hier davn aus, dass die Datei direkt in unserem Arbeitsverzeichnis liegt. Zweitens macht es Sinn, den für Java erlaubten maximalen Arbeitsspeicher festzulegen, der unter einigen Systemen standardmäßig auf 64MB limitiert ist. Wir erhöhen ihn hier auf 512MB. Die weiteren hier gemachten Angaben sind wiederum Standardeinstellungen:

```
myBiomodOption <- BIOMOD_ModelingOptions(
  MAXENT = list( path_to_maxent.jar = getwd(),  # Pfadangabe zu maxent.jar
                 memory_allocated = 512,        # Arbeitspeicher f. Java
                 maximumiterations = 200,
                 visible = FALSE,
                 linear = TRUE,
                 quadratic = TRUE,
                 product = TRUE,
                 threshold = TRUE,
                 hinge = TRUE,
                 lq2lqptthreshold = 80,
                 l2lqthreshold = 10,
                 hingethreshold = 15,
                 beta_threshold = -1,
                 beta_categorical = -1,
                 beta_lqp = -1,
                 beta_hinge = -1,
                 defaultprevalence = 0.5))
```

Modellierung gegenwärtige Klimabedingungen

Wir führen jetzt die eigentliche Verbreitungsmodellierung durch. Dazu verwenden wir die Modellalgorithmen Maxent, Generalized Linear Model, Generalized Boosted Regression Model und Random Forest. Die einzelnen Parameter sind im R-Skript genauer erklärt.

```
mymodels <- BIOMOD_Modeling( myBiomodData,
                             models = c("MAXENT", "GLM", "GBM", "RF"),
                             models.options = myBiomodOption,
                             NbRunEval=1,
                             DataSplit=70,
                             Prevalence=NULL,
                             VarImport=3,
                             models.eval.meth = c("ROC", "TSS", "KAPPA"),
                             SaveObj = TRUE,
                             do.full.models=FALSE,
                             modeling.id=speciesname,
                             rescal.all.models = FALSE)
```

Sicherung des gesamten Arbeitsbereiches (work space) mit save.image als R.Data, kann mit load("XY.RData") wieder in R eingelesen werden:

```
save.image(paste0(path.data, "workspace-backup_", speciesname, ".RData"))
```

Wir können uns nun eine Übersicht über die Modellierungsergebnisse verschaffen:

```
mymodels # Zusammenfassung der Modellierung
get_evaluations(mymodels) # Modellevaluierung
get_variables_importance(mymodels) # „Wichtigkeit" der einzelnen Variablen
```

Zu diesem Zeitpunkt sind unsere erstellten Modell nur einige in mymodels gespeicherte Formeln. Wenn wir sie als Karte graphisch darstellen wollen, müssen wir diese Formeln auf die Werte unserer Umweltdaten-Raster anwenden. Man spricht hier von einer „Projektion". Dazu verwenden wir die biomo2-Funktion `BIOMOD_Projection` (Die einzelnen Parameter sind im R-Skript genauer erklärt, siehe auch: Hilfeseite zu BIOMOD_Projection).

```
myproj.baseline <- BIOMOD_Projection(modeling.output = mymodels,
                                     new.env = envir.baseline,
                                     proj.name = "baseline",
                                     selected.models = 'all',
                                     binary.meth = 'ROC',
                                     compress = F,
                                     build.clamping.mask = F,
                                     output.format = '.img')
# Zusammenfassung der Gegenwarts-Projektion:
myproj.baseline
# Darstellung der Ergebnisse der einzelnen Modellalgorithmen als Karte:
plot(myproj.baseline)
```

Gegenwärtige Klimabedingungen: Ensemble

Anstatt die Ergebnisse der einzelnen Modellierungsalgorithmen separat zu betrachten, können wir aus ihnen auch ein Modell-Ensemble erstellen:

```
mymodels.ensemble <- BIOMOD_EnsembleModeling(modeling.output = mymodels,
                                             chosen.models = 'all',
                                             em.by='all',
                                             eval.metric = c('ROC'),
                                             prob.mean = T,
                                             prob.cv = T,
                                             prob.ci = T,
                                             prob.ci.alpha = 0.05,
                                             prob.median = T,
                                             committee.averaging = T,
                                             prob.mean.weight = T,
                                             prob.mean.weight.decay = 'proportional' )
# Ausgabe einer Zusammenfassung der Ensemble-Modellierung Gegenwart
mymodels.ensemble
# Ausgabe der Modellevaluierung
get_evaluations(mymodels.ensemble)
```

Dieses Ensemble können wir nun wie zuvor auch die einzelnen Modelle auf unsere Umweltdaten projizieren. Anstelle `BIOMOD_Projection` von nutzen wir hier jedoch die Funktion `BIOMOD_EnsembleForecasting`.

```
myproj.baseline.ensemble <- BIOMOD_EnsembleForecasting(
                                        EM.output = mymodels.ensemble,
                                        projection.output = myproj.baseline)
# Ausgabe einer Zusammenfassung der Ensemble-Modellierung Zukunft
myproj.baseline.ensemble
```

Das hierbei erstellte Datenobjekt mit den Ergebnissen der Projektion enthält zahlreiche Informationen die für eine Reine Kartendarstellung nicht relevant sind. Wir nutzen die Funktion `get_predictions` um nur die für uns wichtigen geografischen Daten zu daraus zu extrahieren. Interessant ist insbesondere der Layer mit `EMmeanBy-ROC_mergedAlgo_mergedRun_mergedData`, der die gemeinsam von allen Modellalgorithmen ermittelte klimatische Eignung des Untersuchungsgebietes für unsere Art beinhaltet. Nach einer ersten Begutachtung der Ergebnisse in R speichern wir diese als GeoTiff Dateien zur weiteren Verwendung in externen Programmen.

```
# Extraktion der Geodaten aus der Projektion
myproj.baseline.ensemble.geo <- get_predictions(myproj.baseline.ensemble)
# Plot der Modellierungsergebnisse
plot(myproj.baseline.ensemble.geo)
# Plot der mittleren vorhergesagten Suitability ueber alle Modellalgorithmen
plot(myproj.baseline.ensemble.geo$testspecies_EMmeanByROC_mergedAlgo_mergedRun
_mergedData)
# Ergebnisse abspeichern:
writeRaster(x=myproj.baseline.ensemble.geo,
            filename=paste0(path.output, speciesname, "_ensemble_baseline"),
            bylayer=TRUE, suffix="names", format="GTiff", overwrite=TRUE)
```

Zukunftsprojektionen: Einzelmodelle

„Vorhersagen“ für zukünftige Klimabedingungen erfolgen auf die gleiche Art und Weise wie die Projektion der Modelle auf die Klimadaten für Gegenwärtige Bedingungen, einzig die Klimadaten werden mit denen für zukünftige Bedingungen ersetzt:

```
myproj.future <- BIOMOD_Projection(modeling.output = mymodels,
                                   new.env = envir.future,
                                   proj.name = 'future',
                                   selected.models = 'all',
                                   binary.meth = 'ROC',
                                   compress = F,
                                   build.clamping.mask = F,
                                   output.format = '.img')
# Zusammenfassung der Zukunfts-Projektion
myproj.future
```

```
# Darstellung der Zukunfts-Projektion  der einzelnen Modell-Algorithmen
plot(myproj.future)
```

Zukunftsprojektionen: Ensemble

Auch hier wird ganz analog zur Gegenwartsprojektion des Ensembles verfahren:

```
myproj.future.ensemble <- BIOMOD_EnsembleForecasting(
                                    EM.output = mymodels.ensemble,
                                    projection.output = myproj.future)
# Zusammenfassung
myproj.future.ensemble
# Extraktion der Geodaten aus der Projektion
myproj.future.ensemble.geo <- get_predictions(myproj.future.ensemble)
# Plot der Modellierungsergebnisse
plot(myproj.future.ensemble.geo)
# Plot der mittleren vorhergesagten Suitability ueber alle Modellalgorithmen
plot(myproj.future.ensemble.geo$testspecies_EMmeanByROC_mergedAlgo_mergedRun_m
ergedData)
# Ergebnisse abspeichern:
writeRaster(x=myproj.future.ensemble.geo,
            filename=paste0(path.output, speciesname, "_ensemble_future"),
            bylayer=TRUE, suffix="names", format="GTiff", overwrite=TRUE)
```

Teilskript 4: Überlebensfähigkeit von Vektoren basierend auf Experimentellen Daten

Wir verwenden nun die anfangs manuell heruntergeladenen Klimadaten. Die 3 Dateien

- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20300101-20391231.nc
- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20400101-20491231.nc
- tas_day_MPI-ESM-LR_rcp45_r1i1p1_20500101-20591231.nc

enthalten in täglicher Auflösung die vom Klimamodell MPI-ESM-LR für das Klimaszenario RCP4.5 und die Zeitspanne 2030-2050 projizierten Tagesmitteltemperaturen. Auch hier handelt es sich um Rasterdaten, allerdings in einer deutlich gröberen räumlichen Auflösung als die zuvor verwendeten Bioclim-Daten (die im Gegenzug aber zeitlich sehr viel niedriger aufgelöst sind). Wir müssen nun die 3 Dateien zusammenfügen und auf den Untersuchungszeitraum (2031-2050) und das Untersuchungsgebiet Japan zuschneiden. Außerdem werden wir langjährige Mittelwerte der Tagestemperaturen für jeden Tag im Jahr berechnen. Dazu sind einige Vorbereitungen nötig:

```
# Variablennamen für Mitteltemperatur in ntcdf-Dateien
varname<-"tas"
# Shapefile fuer den Zuschnitt der Daten auf Japan (erneut) laden
cropshape <- readOGR(dsn=substr(path.misc, start=1, stop=nchar(path.misc)-1),
                     layer="JPN_adm0" )
# Vollständige Pfade zu Klimadaten-Dateien einlesen
filenames.tas <- dir(path.tas, pattern=".nc", full.names=TRUE)
#Ueberpruefen:
filenames.tas
```

Anschließend verwenden wir die Risktool-Funktion `rt.extract365` um die Daten automatisiert wie oben beschrieben aufzubereiten. Dies kann insbesondere bei räumlich höher aufgelösten Daten einige Zeit in Anspruch nehmen.

```
rt.extract365(infiles = filenames.tas,
              startyear = 2031,
              endyear = 2050,
              path.outfiles = paste0(path.tas, "2031-2050_means_cropped/"),
              cropshape = cropshape,
              cropmode = "crop",
              fname.prefix="tas_day",
              fun=mean,
              varname=varname,
              overwrite=FALSE
              )
```

Im Unterverzeichnis

`…/data/environment/daily_temperature/2031-2050_means_cropped/`

findet sich nun für jeden Tag im Jahr eine Tif-Datei mit der langjährigen Mitteltemperatur für unseren Beobachtungszeitraum.

Nehmen wir nun zu Demonstrationszwecken an, Experimente hätten gezeigt dass unser Vektor bei Temperaturen von mehr als 30°C sofort stirbt. (Die tatsächlich experimentell gefundene Temperatur-Untergrenze fuer das Überleben von Ae. japoicus-Eiern lässt sich mit den hier verwendeten, räumlich grob aufgelösten Daten leider nicht darstellen.

Wir legen also eine Grenztemperatur von 30°C als Obergrenze für das Überleben unseres Vektors fest. Dabei gilt zu beachten, dass Temperaturdaten von Klimatologen oft in "°C x 10" gespeichert werden. Dies ist auch bei unseren Beispieldaten der Fall, weshalb wir unseren Grenzwert mit 10 multiplizieren:

```
threshold <- 30 * 10
```

Als Eingangsdaten legen wir nun die zuvor erstellten Langjährigen Tagesmittel fest:

```
filenames.temperatures <- dir(paste0(path.tas, "2031-2050_means_cropped/"),
                              pattern=".tif$", full.names=TRUE)
```

Die Risktool-Funktion `rt.threshold` erlaubt es uns nun, für jede Rasterzelle die Anzahl an Tagen im Jahr herauszufinden, bei denen die Temperatur über 30°C liegt.

```
hotdays.number <-  rt.threshold(filenames=filenames.temperatures, thresh-
old=threshold, thresholdside="above")
plot(hotdays.number)
plot(cropshape, add=TRUE)
```

Dies können wir nun in eine einfach ja/nein-Information umwandeln: Ist die Anzahl an Tagen mit einer Temperatur von mehr als 30°C größer als 0, kann laut unseren (imaginären) experimentellen Ergebnissen der Vektor nicht überleben.

```
# Rasterzellen mit Überleben auf 0, ohne Ueberleben auf 1 setzen
survives <- hotdays.number
survives[which(hotdays.number[]>0)] <- 0
survives[which(hotdays.number[]==0)] <- 1
```

Mit dieser Information können wir unser zuvor erstelltes Artverbreitungsmodell verfeinern, indem wir Gebiete die „sicher“ kein Überleben zulassen herausfiltern. Dazu lesen wir die Ergebnisse des Verbreitungsmodells (hier: Ensemble für 2031-2050) wieder ein:

```
sdm <- raster(paste0(path.output,
"testspeies_ensemble_future_testspecies_EMmeanByROC_mergedAlgo_mergedRun_mergedData.tif"))
plot(sdm)
```

Aufgrund der unterschiedlichen räumlichen Auflösung der jeweiligen Ausgangsdaten müssen wir die beiden Datensätze bezöglich dessen aneinanderanpassen. Um keinen Informationsverlust durch Downsampling des Verbreitungsmodells zu haben, wenden wir ein einfaches Upsampling auf die neuen „Überlebensdaten“ an:

```
survives.res <- resample(survives, sdm, method="ngb")
```

Wenn wir nun die Werte der beiden Raster miteinander multiplizieren, werden aufgrund der 1/0-Codierung der Überlebensdaten die Werte des Verbreitungsmodells in Gebieten ohne Überleben auf 0 gesetzt; in allen anderen gebieten bleiben sie unverändert.

```
# Multiplikation der beiden Raster
sdm.survival <- sdm*survives.res
# Ergebnisausgabe
plot(sdm.survival)
# Als TIF zur weiteren Verwendung in externen Programmen abspeichern
writeRaster(sdm.survival, paste0(path.output, "sdm_survival.tif"))
```

Teilskript 5: Temperaturabhängigkeit der Extrinsischen Inkubationsperiode

Nehmen wir zu Demonstrationszwecken an, ein Experiment haette einen einfachen linearen Zusammenhang zwischen der Umgebungstemperatur (in Grad Celsius) und der Extrinsischen Inkubationsperiode (EIP, in Tagen) eines von unserem Beispielvektor uebertragenen Virus festgestellt: EIP=45-Temperatur

```
# Beispieldaten erzeugen
eip.experiment <- data.frame(Temperatur=15:40, EIP=45-(15:40))
eip.experiment
plot(eip.experiment)
```

Das bedeutet, dass bei höheren Umgebungstemperaturen der Virus schneller übertragen werden kann als bei niedrigen (Körpertemperatur der Mücke hängt von der Umgebungstempertur ab, da endotherm.
Wir verwenden hier erneut die im vorigen Abschnitt erstellten Langjährigen Tagesmittel und die Risktool-Funktion rt.consdaysdata um sie entsprechend zu formatieren:

```
filenames.temperatures <- dir(paste0(path.tas, "2031-2050_means_cropped/"),
                              pattern=".tif$", full.names=TRUE)
dat <- rt.consdaysdata(filenames.temperatures)
```

Wir können nun die Risktool-Funktion `rt.consdays` verwenden, um die maximale Anzahl aufeinanderfolgender Tage bei einer gegebenen Temperatur herauszufinden (hier beispielhaft: >= 28 Grad, angegeben als 280):

```
days28 <- rt.consdays(data=dat, referencevalue=280, startday=1, endday=365)
days28 # angabe für jede Rasterzelle
# Umwandlung in geographische Daten
days28.geo <- dat             # Kopieren der Geoinformation der Eingangsdaten
# Überschreiben der Eingangsdaten mit den Ausgabedaten
days28.geo@data <- data.frame(days=days28)
# SpatialGridDataFrame wegen besserer Plot-Funktion in Raster umwandeln
days28.geo <- raster(days28.geo)
plot(days28.geo)  # Plotten
```

Interessanter für eine Risikoabschätzung ist es jedoch, für jede Rasterzelle die kürzest mögliche EIP herauszufinden. Dazu rufen wir `rt.consdays` indirekt durch `rt.eip_duration` auf:

```
eip <- rt.eip_duration(data=dat,
                       eips=eip.experiment$EIP,
                       temperatures=eip.experiment$Temperatur*10,
                       startday=1, endday=365)
plot(eip)
```

Auch diese Ergebnisse können wir zur Verfeinerung des Verbreitungsmodells bzw. zur Erstellung eines Risiko-Indexes verwenden. Da der Wertebereich des Verbreitungsmodells

intern den 0-1000 beträgt, können wir ihn durch einfache Division mit 1000 auf Werte zwischen 0 und 1 normieren.

```
sdm.norm <- sdm/1000
```

Da bei der EIP umgekeht kleinere Wete eine größere Gefahr bedeuten, verwenden wir hier den Kehrwert der EIP, den wir dann ebenfalls auf Werte zwischen 0 und 1 normieren. Wir verwenden dazu die Risktool-Funktion `rt.transformEIP`, die optional gleich die Anpassung der Auflösung an das Raster des Verbreitungsmodells vornimmt:

```
eip.norm <- rt.transformEIP(eip, refraster=sdm)
```

Eine Multiplikation der beiden Raster liefert uns einen einfachen Risikoindex:

```
risk <- sdm.norm*eip.norm
# Vergleichsplot
par(mfrow=c(2,1))
plot(sdm.norm, main = "SDM")
plot(risk, main = "SDM * EIP")
par(mfrow=c(1,1))
# Als TIF zur weiteren Verwendung in externen Programmen abspeichern
writeRaster(risk, paste0(path.output, "risk-index.tif"))
```