Appendix 10: Code for case study 2: Model averaging applied to the distribution analysis of short-finned eel

Dormann et al.

28 Feb 2018

Contents

1 Introduction

In this case study we will use data on the real distribution of short-finned eel (*Anguilla australis*) in New Zealand as an example. The data are provided in the R-package **dismo**, already split into a 1000-rows training and a 500-rows test data set (Anguilla_train and Anguilla_test, with environmental data in Anguilla_grids).

The response is called Angaus, and there is a range of predictors (SegSumT, SegSumT, SegTSeas, SegLowFlow, DSDist, DSMaxSlope, USAvgT, USRainDays, USSlope, USNative). We omit DSDam and Method for simplicity. For explanation of these variables please see the help file for Anguilla_train (?Anguilla_train).

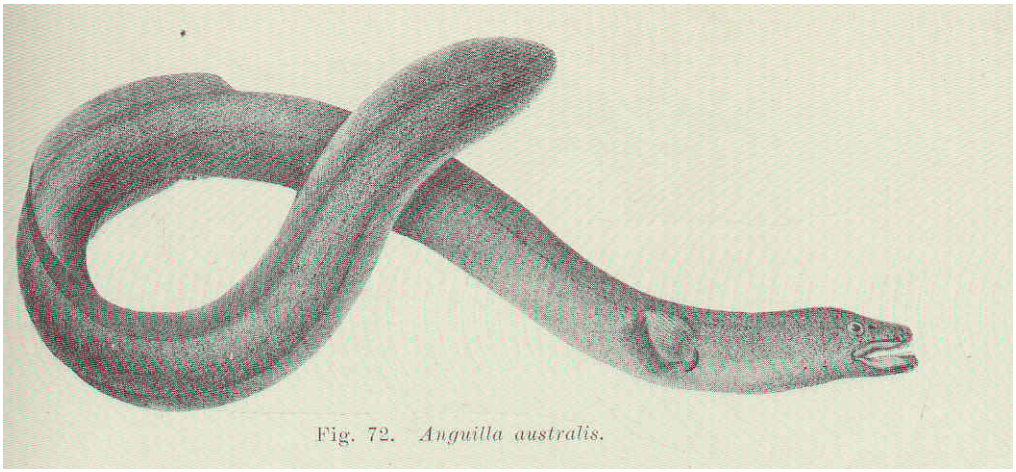


Figure 1: alt text

In this example study, we *omit* several steps that we would usually would perform (or consider to perform):

1. standardisation of predictors;
2. resolving collinearity of predictors (which is moderate, with $r_{\max} = 0.64$);
3. include the model-selection procedure in all approaches (for GLM), i.e. not refitting the model selected in the beginning, but rather see the stepwise selection of variables as part of the GLM-approach;
4. spatial block-cross valdiation to reduce optimism on goodness-of-fit statistics;
5. analysis of spatial autocorrelation.

We thus implore the reader *not* to take this as an exemplatory analysis, but rather as an illustration of model averaging techniques in the context of species distribution analyses.

2 Setting up the data and running six different models

We run different and common modeling approaches for presence-absence data, plus a variation on the GLM:

1. Generalised Linear Model (glm) with backward-stepwise selection based on AIC; the full model contained all pairwise interactions and a selection of quadratic effects);
2. GLM selected down by BIC from the previous model (glmBIC);
3. Generalised Additive Model (gam) with cubic regression splines and an extra penalty to be able to remove irrelevant predictors out of the model;
4. randomForest (rf), with default settings;
5. a neural network (nn) of size 11 and a decay of 0.1;
6. a support vector machine classification (svm) of type "C" with radial kernel.

```
# data:
library(dismo)
# models:
library(e1071) # for sum
library(MASS) # for stepAIC
library(mgcv) # for gam
library(nnet) # for nnet
library(randomForest)
# model averaging:
library(EBMAforecast)
```

```r
data(Anguilla_train)
data(Anguilla_test)
data(Anguilla_grids)

fglm <- stepAIC(glm(Angaus ~ (SegSumT + SegSumT + SegTSeas + SegLowFlow + DSDist +
      DSMaxSlope + USAvgT + USRainDays + USSlope + USNative)^2 + I(DSDist^2) +
      I(USAvgT^2), family=binomial, data=Anguilla_train, control=list(maxit=500)), trace=F)
# We did some cheating here, using GAM to select variables for which to include a quadratic term
summary(fglm) # still a very large model
```

```
Call:
glm(formula = Angaus ~ SegSumT + SegTSeas + SegLowFlow + DSDist +
    DSMaxSlope + USAvgT + USRainDays + USSlope + USNative + I(USAvgT^2) +
    SegSumT:DSDist + SegSumT:USSlope + SegTSeas:DSDist + SegTSeas:USRainDays +
    SegTSeas:USSlope + SegTSeas:USNative + DSDist:DSMaxSlope +
    DSDist:USAvgT + DSDist:USSlope + DSDist:USNative, family = binomial,
    data = Anguilla_train, control = list(maxit = 500))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.73496  -0.58923  -0.16645  -0.00034   3.15421

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
(Intercept)        -0.1660190  3.1442080  -0.053 0.957890
SegSumT             0.0018520  0.1747375   0.011 0.991544
SegTSeas           -0.7366229  0.3328321  -2.213 0.026884 *
SegLowFlow          2.1903928  0.6372309   3.437 0.000587 ***
DSDist             -0.2514218  0.0507992  -4.949 7.45e-07 ***
DSMaxSlope         -0.1800507  0.0760822  -2.367 0.017956 *
USAvgT             -0.2915087  0.3275688  -0.890 0.373511
USRainDays         -1.4012026  0.3499911  -4.004 6.24e-05 ***
USSlope            -0.4033205  0.2193277  -1.839 0.065931 .
USNative            0.5103062  0.7778306   0.656 0.511783
I(USAvgT^2)        -0.2274882  0.1110110  -2.049 0.040439 *
SegSumT:DSDist      0.0130017  0.0027636   4.705 2.54e-06 ***
SegSumT:USSlope     0.0185880  0.0122446   1.518 0.129001
SegTSeas:DSDist     0.0089527  0.0031740   2.821 0.004793 **
SegTSeas:USRainDays 0.4959986  0.2222928   2.231 0.025662 *
SegTSeas:USSlope    0.0426801  0.0232800   1.833 0.066753 .
SegTSeas:USNative  -1.0482887  0.4706341  -2.227 0.025921 *
DSDist:DSMaxSlope   0.0027490  0.0012596   2.183 0.029072 *
DSDist:USAvgT       0.0100774  0.0046839   2.151 0.031438 *
DSDist:USSlope      0.0013957  0.0005628   2.480 0.013144 *
DSDist:USNative    -0.0143756  0.0100441  -1.431 0.152360
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1006.3  on 999  degrees of freedom
Residual deviance:  664.1  on 979  degrees of freedom
AIC: 706.1
```

```
Number of Fisher Scoring iterations: 9
```

```
fglmBIC <- stepAIC(fglm, k=log(min(table(Anguilla_train$Angaus))), trace=F)
summary(fglmBIC)
```

```

Call:
glm(formula = Angaus ~ SegSumT + SegTSeas + SegLowFlow + DSDist +
    DSMaxSlope + USRainDays + USNative + I(USAvgT^2) + SegSumT:DSDist +
    SegTSeas:DSDist + SegTSeas:USRainDays + DSDist:DSMaxSlope,
    family = binomial, data = Anguilla_train, control = list(maxit = 500))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6326  -0.6010  -0.1833  -0.0014   3.2010

Coefficients:
                     Estimate Std. Error z value Pr(>|z|)
(Intercept)        -4.1407341  1.7897164  -2.314 0.020688 *
SegSumT             0.2420816  0.0959866   2.522 0.011668 *
SegTSeas           -0.6836268  0.2648430  -2.581 0.009844 **
SegLowFlow          1.5996324  0.5409227   2.957 0.003104 **
DSDist             -0.2200433  0.0466989  -4.712 2.45e-06 ***
DSMaxSlope         -0.2220805  0.0658824  -3.371 0.000749 ***
USRainDays         -1.3965688  0.3186924  -4.382 1.17e-05 ***
USNative           -0.8742864  0.2967712  -2.946 0.003219 **
I(USAvgT^2)        -0.2109691  0.0863629  -2.443 0.014573 *
SegSumT:DSDist      0.0117354  0.0025859   4.538 5.67e-06 ***
SegTSeas:DSDist     0.0102503  0.0031048   3.301 0.000962 ***
SegTSeas:USRainDays 0.5031607  0.1985775   2.534 0.011282 *
DSDist:DSMaxSlope   0.0035174  0.0008037   4.376 1.21e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1006.33  on 999  degrees of freedom
Residual deviance:  678.56  on 987  degrees of freedom
AIC: 704.56

Number of Fisher Scoring iterations: 8
```

```
fgam <- gam(Angaus ~ s(SegSumT, bs="cr") + s(SegSumT, bs="cr") + s(SegTSeas, bs="cr") +
      s(SegLowFlow, bs="cr") + s(DSDist, bs="cr") + s(DSMaxSlope, bs="cr") + s(USAvgT, bs="cr") +
      s(USRainDays, bs="cr") + s(USSlope, bs="cr") + s(USNative, bs="cr"), select=T,
      family=binomial, data=Anguilla_train)
summary(fgam)
```

```

Family: binomial
Link function: logit

Formula:
Angaus ~ s(SegSumT, bs = "cr") + s(SegSumT, bs = "cr") + s(SegTSeas,
```

```
    bs = "cr") + s(SegLowFlow, bs = "cr") + s(DSDist, bs = "cr") +
    s(DSMaxSlope, bs = "cr") + s(USAvgT, bs = "cr") + s(USRainDays,
    bs = "cr") + s(USSlope, bs = "cr") + s(USNative, bs = "cr")

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.0398     0.3615  -8.408   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                  edf Ref.df Chi.sq  p-value
s(SegSumT)    2.796e+00      9 36.534 3.92e-10 ***
s(SegTSeas)   1.891e+00      9  7.298 0.011931 *
s(SegLowFlow) 1.479e+00      9 11.821 0.000190 ***
s(DSDist)     8.426e+00      9 15.986 0.044190 *
s(DSMaxSlope) 1.003e+00      9  5.847 0.004879 **
s(USAvgT)     7.046e+00      9 24.085 0.000478 ***
s(USRainDays) 1.962e+00      9 18.907 1.30e-05 ***
s(USSlope)    2.743e-05      9  0.000 0.527023
s(USNative)   3.244e+00      9 13.127 0.001966 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.312   Deviance explained = 34.6%
UBRE = -0.28432  Scale est. = 1         n = 1000
```

```
#plot(fgam)

frf <- randomForest(as.factor(Angaus) ~ SegSumT + SegSumT + SegTSeas + SegLowFlow + DSDist +
                DSMaxSlope + USAvgT + USRainDays + USSlope + USNative, data=Anguilla_train)
frf
```

```
Call:
 randomForest(formula = as.factor(Angaus) ~ SegSumT + SegSumT +      SegTSeas + SegLowFlow + DSDist + DS
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 15.5%
Confusion matrix:
    0  1 class.error
0 753 45  0.05639098
1 110 92  0.54455446
```

```
fnn <- nnet(Angaus ~ SegSumT + SegSumT + SegTSeas + SegLowFlow + DSDist + DSMaxSlope + USAvgT +
      USRainDays + USSlope + USNative, data=Anguilla_train, size=11, maxit=500, decay=0.1, trace=F)
fnn
```

```
a 9-11-1 network with 122 weights
inputs: SegSumT SegTSeas SegLowFlow DSDist DSMaxSlope USAvgT USRainDays USSlope USNative
output(s): Angaus
options were - decay=0.1
```

```
fsvm <- svm(x=as.matrix(Anguilla_train[,c(3:12)]), y=as.factor(Anguilla_train$Angaus),
            type="C-classification", probability=TRUE, kernel="radial")
fsvm
```

```
Call:
svm.default(x = as.matrix(Anguilla_train[, c(3:12)]), y = as.factor(Anguilla_train$Angaus),
    type = "C-classification", kernel = "radial", probability = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.1

Number of Support Vectors:  452
```

3 Assessing the fit on the test data

To do so we use the likelihood of the test data, given the prediction of the models fitted to the train data. To save space, we define a little helper function to compute the log-likelihood for us. This we then apply to the six models from above.

```
lltest <- function(pred, obs){
    # compute the likelihood of observing the test data, given the predictions
    sum(dbinom(obs, size=1, prob=pred, log=T))
}

pGLM <- predict(fglm, type="response", newdata=Anguilla_test)
pGLMbic <- predict(fglmBIC, type="response", newdata=Anguilla_test)
pGAM <- predict(fgam, type="response", newdata=Anguilla_test)
prf <- predict(frf, type="prob", newdata=Anguilla_test)[,2]
pnn <- predict(fnn, type="raw", newdata=Anguilla_test)
psvm <- attr(predict(fsvm, newdata=Anguilla_test[, 2:11], probability=TRUE), "probabilities")[,2]
# note: column "Site" missing in test data!

(ellTest <- sapply(list(pGLM, pGLMbic, pGAM, prf, pnn, psvm), lltest, obs=Anguilla_test$Angaus))
```

```
[1] -197.4759 -197.7348 -193.3972 -189.2895 -187.9738 -215.0021
```

Larger values indicate a better fit to the test data, indicating randomForest as best model, followed by GAM and GLM-AIC. SVM is substantially worse, possibly indicating that our model tuning choices weren't ideal.

Next, we put all predictions to test data into one matrix, making some of the computations later easier.

```
allpreds.test <- cbind(pGLM, pGLMbic, pGAM, prf, pnn, psvm)
```

We are now set to explore the various model averaging options. Note that AIC, BIC, Mallows Cp, WAIC require log-likelihoods of the model fits and ranks of models, which we do not have for randomForest, neural net and support vector machines. Hence these procedures will not be used.

4 Equal weights: $1/M$

Equal weights means we can simply average across the predictions.

```
pred.1overM.mean <- rowMeans(allpreds.test)
lltest(pred.1overM.mean, obs=Anguilla_test$Angaus) # [1] -185.2922
```

```
[1] -185.7434
```

So the median is slightly better (in line with suggestions by Bates & Granger 1969).

5 Median of predictions

```
pred.median <- apply(allpreds.test, 1, median)
lltest(pred.median, obs=Anguilla_test$Angaus) # [1] -183.6075
```

```
[1] -182.9127
```

It may be interesting to know, which models contributed to these 500 predicted points. It could be that only two were ever choosen.

```
# first find out which are 3rd and 4th best, then make it a vector and then table it all
# (note that because always 2 models form the median, we need to divide by 1000 for the
# 500 data points)
table(as.vector(apply(allpreds.test, 1, function(x) order(x)[3:4])))/1000
```

```
    1     2     3     4     5     6
0.176 0.237 0.212 0.162 0.124 0.089
```

6 Leave-one-out cross-validation

In LOO, each data point is omitted in turn and models are fitted on the reduced data set, to then predict to the omitted data point. Since we have 1000 data points, this takes a while (hours), as we have to re-fit 6 models 1000 times. (Note that we take the GLM model structure for granted and do not re-run the model selection for each LOO again. This should be considered in order to cross-validate also the flexibility that led to the final model structure.)

```
if (!("App_case2_looAll.Rdata" %in% dir())){
  looAll <- matrix(NA, nrow=nrow(Anguilla_train), ncol=6)
  for (i in 1:nrow(Anguilla_train)){
    fm.loo <- lapply(list(fglm, fglmBIC, fgam, frf, fnn),
                     function(x) update(x, .~., data=Anguilla_train[-i, ]))
    fm.loo.svm <- svm(x=as.matrix(Anguilla_train[-i, c(3:12)]),y=as.factor(Anguilla_train$Angaus[-i]),
            type="C-classification", probability=TRUE, kernel="radial") # no update function for svm
    looAll[i, 1:3] <- suppressWarnings(sapply(fm.loo[1:3], function(x)
                          predict(x, newdata=Anguilla_train[i, ,drop=F], type="response")))
    looAll[i, 4] <- predict(fm.loo[[4]], newdata=Anguilla_train[i, ,drop=F], type="prob")[,2]
    looAll[i, 5] <- predict(fm.loo[[5]], newdata=Anguilla_train[i, ,drop=F], type="raw")
    looAll[i, 6] <- attr(predict(fm.loo.svm, newdata=Anguilla_train[i, 3:12 ,drop=F],
                                 probability=TRUE), "probabilities")[,2]
  }
  save(looAll, file="App_case2_looAll.Rdata") ################## good idea ###############
```

```
} else {
  load(file="App_case2_looAll.Rdata")
}
```

Note that we can use the object looAll again for Jackknife Model Averaging. It contains the predictions of the re-fitted models to the omitted data point.

```
# now choose a criterion for evaluation, e.g. RMSE:
LooRMSE <- apply(looAll, 2, function(x) sqrt(mean((x-Anguilla_train$Angaus)^2)))
# turn into weights:
(weightsLOO <- (exp(-1*(LooRMSE-min(LooRMSE))))/sum(exp(-1*(LooRMSE-min(LooRMSE)))))
```

```
[1] 0.1681401 0.1683243 0.1664732 0.1685105 0.1646118 0.1639401
```

Now use these weights for combining predictions to test data:

```
lltest(allpreds.test %*% weightsLOO, obs=Anguilla_test$Angaus)
```

```
[1] -185.6939
```

Let's do this also for AUC (and exp(AUC)) and log-likelihood:

```
library(verification)
looAUC <- apply(looAll, 2, function(x) roc.area(pred=x, obs=Anguilla_train$Angaus)$A)
ellbern <- function(x) -sum(dbinom(Anguilla_train$Angaus, size=1, prob=x, log=T))
# negative ell for minimisation
LOOell <- apply(looAll, 2, ellbern)
# turn into weights:
(weightsLOOauc <- (exp(-1*(looAUC-min(looAUC))))/sum(exp(-1*(looAUC - min(looAUC)))))
```

```
[1] 0.1632329 0.1634690 0.1673113 0.1643863 0.1718355 0.1697651
```

```
(weightsLOOexpauc <- (exp(-1 * (exp(looAUC) - min(exp(looAUC))))) /
    sum(exp(-1 * (exp(looAUC) - min(exp(looAUC))))))
```

```
[1] 0.1587183 0.1592615 0.1681375 0.1613741 0.1786688 0.1738398
```

```
(weightsLOOell <- (exp(-1 * (LOOell-min(LOOell)))) / sum(exp(-1 * (LOOell - min(LOOell)))))
```

```
[1] 2.076482e-01 7.903210e-01 2.114310e-16 2.030841e-03 4.488990e-24 1.408956e-31
```

```
lltest(allpreds.test %*% weightsLOOauc, obs=Anguilla_test$Angaus)
```

```
[1] -185.7827
```

```
lltest(allpreds.test %*% weightsLOOexpauc, obs=Anguilla_test$Angaus)
```

```
[1] -185.838
```

```
lltest(allpreds.test %*% weightsLOOell, obs=Anguilla_test$Angaus)
```

```
[1] -196.9886
```

7 Bayesian Model Averaging using Expectation Maximisation (BMA-EM)

Montgomery et al. (2012, and pers. comm.) suggest to split the training data (e.g. 50/50), fit on one half, predict to the other, and use these predictions (along with the true values) to calibrate the ensemble. Using the fits to the full training set will lead to giving overfitting models most weight.

```
set.seed(1)
trainsplit <- sample(rep(c(T, F), 500))
train1 <- Anguilla_train[trainsplit, ]
train2 <- Anguilla_train[!trainsplit, ]

ebmafglm <- update(fglm, .~., data=train1)
ebmafglmBIC <- update(fglmBIC, .~., data=train1)
ebmafgam <- update(fgam, .~., data=train1, sp=fgam$sp)
#update gam without re-fitting the penalised terms!
ebmafrf <- update(frf, .~., data=train1)
ebmafnn <- update(fnn, .~., data=train1)
ebmafsvm <- svm(x=as.matrix(train1[,c(3:12)]), y=as.factor(train1$Angaus),
                type="C-classification", probability=TRUE, kernel="radial")

ebmapreds2 <- cbind("GLM"=predict(ebmafglm, newdata=train2, type="response"),
                    "GLMbic"=predict(ebmafglmBIC, newdata=train2, type="response"),
                    "GAM"=predict(ebmafgam, newdata=train2, type="response"),
                    "RF"=predict(ebmafrf, newdata=train2, type="prob")[,2],
                    "ANN"=as.numeric(predict(ebmafnn, newdata=train2, type="raw")),
                    "SVM"=attr(predict(ebmafsvm, probability=TRUE, newdata=train2[, 3:12]),
                               "probabilities")[,2])
ebmaY <- train2$Angaus

ebmafits.noised <- ebmapreds2
ebmafits.noised[,4] <- ebmafits.noised[,4] + 1E-4 # EMBAcalibrate cannot handle 0s or 1s!!
# use predictions from model fitted to ALL data; ebmafits are only to get the model weights
allpreds.noised <- allpreds.test
allpreds.noised[,4] <- allpreds.noised[,4] + 1E-4 # maybe use 1/ntrees instead!
EBMAdata <- makeForecastData(.predCalibration=ebmafits.noised, .outcomeCalibration=ebmaY,
            .predTest=allpreds.noised, .outcomeTest=Anguilla_test$Angaus,
            .modelNames=c("GLMAIC", "GLMBIC", "GAM", "RF", "ANN", "SVM"))
EBMAfit <- calibrateEnsemble(EBMAdata, model="logit", exp=3)
EBMAfit@modelWeights
```

```
    GLMAIC     GLMBIC        GAM         RF        ANN        SVM
0.27210698 0.08120859 0.35761652 0.28906791 0.00000000 0.00000000
```

```
lltest(EBMAfit@predTest[,1,1], obs=Anguilla_test$Angaus)
```

```
[1] -185.1237
```

8 Naive bootstrap

Here we re-run the models on a bootstrap of the data and count how often a model comes out best. "Best" is quantified by RMSE.

```
Nboots <- 1000
bestCounter <- rep(0, 6)
for (i in 1:Nboots){
  bsdata <- Anguilla_train[sample(nrow(Anguilla_train), nrow(Anguilla_train), replace=T),]
  bsfglm <- update(fglm, .~., data=bsdata)
  bsfglmBIC <- update(fglmBIC, .~., data=bsdata)
  bsfgam <- update(fgam, .~., data=bsdata, sp=fgam$sp)
```

```
  bsfrf <- update(frf, .~., data=bsdata)
  bsfnn <- update(fnn, .~., data=bsdata)
  bsfsvm <- svm(x=as.matrix(bsdata[,c(3:12)]), y=as.factor(bsdata$Angaus),
                type="C-classification", probability=TRUE, kernel="radial")

  # fits:
  bsfits <- cbind("GLM"=predict(bsfglm, type="response"),
                  "GLMbic"=predict(bsfglmBIC,type="response"),
                  "GAM"=predict(bsfgam, type="response"),
                  "RF"=predict(bsfrf, type="prob")[,2],
                  "ANN"=as.numeric(predict(bsfnn, type="raw")),
                  "SVM"=attr(predict(bsfsvm, newdata=bsdata[, 3:12], probability=TRUE),
                             "probabilities")[,2]  )
  # RMSE:
  bsfitsRMSE <- sqrt(colMeans((matrix(bsdata$Angaus, nrow=nrow(bsdata), ncol=6, byrow=F) -
                               bsfits)^2))
  bestCounter[which.min(bsfitsRMSE)] <- bestCounter[which.min(bsfitsRMSE)] + 1
  #cat(i, " ")
}

(weightsboot <- bestCounter/sum(bestCounter))
```

```
[1] 0 0 0 1 0 0
```

```
weightedPredsBoot <- allpreds.test %*% weightsboot
lltest(weightedPredsBoot, obs=Anguilla_test$Angaus)
```

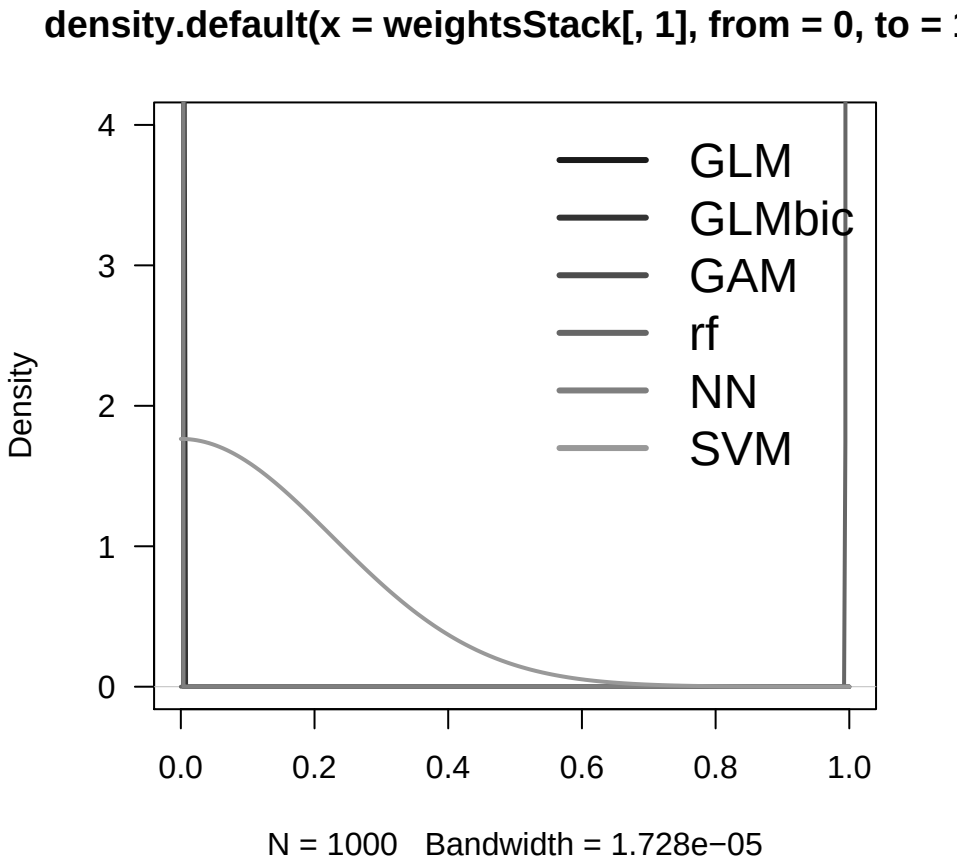
```
[1] -187.2434
```

9 Stacking

Again, this is more time-consuming, as we want to run the stacking repeatedly.

```
stacking <- function(test.preds, test.obs){
    # this function computes the optimal weight for a single train/test split;
    # from the models fitted to the training data it uses the predictions to the test;
    # then it optimises the weight vector across the models for combining these
    # predictions to the observed data in the test;
    # trick 1: each weight is between 0 and 1: w <- exp(-w)
    # trick 2: weights sum to 1: w <- w/sum(w)
    #
    # weights are weights for each model, between -infty and +infty!
    # preds are predictions from each of the models

    if (NCOL(test.preds) >= length(test.obs)) stop("Increase the test set! More models
                                                    than test points.")

    # now do an internal splitting into "folds" data sets:
    weightsopt <- function(ww){
        # function to compute RMSE on test data
        w <- c(1, exp(ww)); w <- w/sum(w)
        # w all in (0,1) SIMON; set weight1 always to 1, other weights are scaled accordingly
        # (this leads to a tiny dependence of optimal weights on whether model1 is any good or
```

```
        # utter rubbish; see by moving the 1 to the end instead -> 3rd digit changes)
        pred <- as.vector(test.preds %*% w)
        return(sqrt(mean((pred - test.obs)^2))) #RMSE
    }

    ops <- optim(par=runif(NCOL(test.preds)-1), weightsopt, method="BFGS")
    if (ops$convergence != 0) stop("Optimisation not converged!")
    round(c(1, exp(ops$par))/sum(c(1, exp(ops$par))), 4)
}

Nstack <- 1000
weightsStack <- matrix(NA, ncol=6, nrow=Nstack)
colnames(weightsStack) <- c("GLM", "GLMbic", "GAM", "rf", "NN", "SVM")
i = 0
while (i < Nstack){
  trainsplit <- sample(rep(c(T, F), 500))
  train1 <- Anguilla_train[trainsplit, ]
  train2 <- Anguilla_train[!trainsplit, ]
  stackfglm <- update(fglm, .~., data=train1)
  stackfglmBIC <- update(fglmBIC, .~., data=train1)
  stackfgam <- update(fgam, .~., data=train1, sp=fgam$sp)
  stackfrf <- update(frf, .~., data=train1)
  stackfnn <- update(fnn, .~., data=train1)
  stackfsvm <- svm(x=as.matrix(train1[,c(3:12)]), y=as.factor(train1$Angaus),
                   type="C-classification", probability=TRUE, kernel="radial")
  # predict to train2:
  stackpreds2 <- cbind("GLM"=predict(ebmafglm, newdata=train2, type="response"),
                  "GLMbic"=predict(ebmafglmBIC, newdata=train2, type="response"),
                  "GAM"=predict(ebmafgam, newdata=train2, type="response"),
                  "RF"=predict(ebmafrf, newdata=train2, type="prob")[,2],
                  "ANN"=as.numeric(predict(ebmafnn, newdata=train2, type="raw")),
                  "SVM"=attr(predict(ebmafsvm, probability=TRUE, newdata=train2[, 3:12]),
                             "probabilities")[,2])
  optres <- try(stacking(test.preds=stackpreds2, test.obs=train2$Angaus))
  if (inherits(optres, "try-error"))  next;
  i = i + 1
  weightsStack[i,] <- optres
  rm(trainsplit, train1, train2, stackfglm, stackfglmBIC, stackfgam, stackfrf, stackfnn,
     stacksvm, stackpreds2)
  #print(i)
}
# visualise, if you want:
plot(density(weightsStack[,1], from=0, to=1), las=1, lwd=2, ylim=c(0, 4), xlim=c(0,1),
     col="grey10")
for (j in 2:6) lines(density(weightsStack[,j], from=0, to=1), lwd=2, col=paste0("grey", j*10))
legend("topright", col=paste0("grey", (1:6)*10), lwd=3, legend=colnames(weightsStack),
       cex=1.5, bty="n")
```




```
(weightsStacking <- colSums(weightsStack)/sum(weightsStack))
```

```
         GLM       GLMbic          GAM           rf           NN          SVM
3.440004e-05 1.350001e-04 1.144001e-04 9.996110e-01 1.052001e-04 0.000000e+00
```

```
weightedPredsStack <- allpreds.test %*% weightsStacking
lltest(weightedPredsStack, obs=Anguilla_test$Angaus)
```

```
[1] -187.2322
```

10 Jackknife model averaging

Here we recycle the jackknife-matrix produced in the leave-one-out crossvalidation. Quality criterion is RMSE.

```
weightsopt <- function(ww, J){
    # function to compute RMSE on test data
    w <- c(1, exp(ww)); w <- w/sum(w)
    Jpred <- J %*% w
    return(sqrt(mean((Jpred - Anguilla_test$Angaus)^2)))
}
J <- looAll
ops <- optim(par=runif(NCOL(J)-1), weightsopt, method="BFGS", control=list(maxit=5000), J=J)
if (ops$convergence != 0) stop("Not converged!")
round(weightsJMA <- c(1, exp(ops$par))/sum(c(1, exp(ops$par))),3)
```

```
[1] 0 0 0 0 0 1
```

```
weightedPredsJMA <- allpreds.test %*% weightsJMA
lltest(weightedPredsJMA, obs=Anguilla_test$Angaus)
```

```
[1] -215.5822
```

11 Bates-Granger

Based on prediction-error of the different models, thus again requiring a splitting of the training data to avoid undue favouring of overfitting models. We can recycle the results from EMBA for that.

```
set.seed(1) # same as in EBMA
trainsplit <- sample(rep(c(T, F), 500))
train1 <- Anguilla_train[trainsplit, ]
train2 <- Anguilla_train[!trainsplit, ]
BGresids <- matrix(train2$Angaus, nrow=nrow(train2), ncol=6, byrow=F) - ebmapreds2
Sigma <- cov(BGresids)
ones <- rep(1, 6)
(weightsBG <- solve(t(ones) %*% solve(Sigma) %*% ones)%*%ones%*%solve(Sigma)  )
```

```
            GLM    GLMbic       GAM        RF          ANN        SVM
[1,] 0.02222779 0.3464185 0.3972994 0.4940725 -0.09143414 -0.1685841
```

```
# weights may become negative!
weightedPredsBG <- allpreds.test %*% t(weightsBG)
weightedPredsBG <- ifelse(weightedPredsBG<0, sort(weightedPredsBG[weightedPredsBG>0])[1],
                          weightedPredsBG)  # ensure positive predictions
lltest(weightedPredsBG, obs=Anguilla_test$Angaus)
```

```
[1] -188.628
```

12 Cos-squared model weights

Works with correlation in predictions, thus no splitting of the train data required.

```
csweights <- function(R, eps=1E-6, maxit=50, verbose=FALSE){
  # implements Garthwaite & Mubwandarikwa's cos-square scheme (their appendix)
  # eps and maxit are chosen without much testing; not converging within 20 iterations
  # doesn't actually mean that something is wrong; mostly it works much faster, though,
  # i.e. within only a few iterations.
  require(expm)
  D1 <- diag(rep(2, NCOL(R)))
  D2 <- diag(NCOL(R))
  counter = 0
  while (any( abs(diag(D1) - diag(D2)) > eps)){
      ED <- eigen(D1 %*% R %*% D1)
      Q <- ED$vectors
      Lambda <- diag(ED$values)
      ## test:
      #Q %*% Lambda %*% solve(Q) # fine
      Lambda12 <- sqrtm(Lambda)
      E <- solve(D1) %*% Q %*% Lambda12 %*% solve(Q)
      D2 <- D1
```

```
      D1 <- diag( diag(Re(E)))
      counter <- counter + 1
      if (verbose) cat(counter, " ")
      if (counter >= maxit){
        warning("Maximum number of iterations reached without convergence!")
        break
      }
  }
  w <- diag(D2)^2 / sum(diag(D2)^2)
  return(w)
}
R <- cor(allpreds.test)
(weightsCS <- csweights(R, verbose=F, maxit=50))
```

```
[1] 0.1179181 0.1000021 0.1838883 0.1868688 0.1951257 0.2161970
```

```
weightedPredsCS <- allpreds.test %*% weightsCS
lltest(weightedPredsCS, obs=Anguilla_test$Angaus)
```

```
[1] -185.9797
```

13 Model-based model combinations

Again we can recycle the test data model fits (from the EBMA-code above) for this, thereby avoiding overfitting.

```
mbmcpreds2 <- ebmapreds2
colnames(mbmcpreds2) <- colnames(weightsStack)
# this is a simple linear model:
summary(mbmcGLM <- glm(train2$Angaus ~ ., data=as.data.frame(mbmcpreds2), family=binomial))
```

```

Call:
glm(formula = train2$Angaus ~ ., family = binomial, data = as.data.frame(mbmcpreds2))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0766  -0.4665  -0.2506  -0.2086   2.5557

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.79862    3.76718  -1.805  0.07112 .
GLM          0.98351    2.28862   0.430  0.66739
GLMbic       1.88472    2.28474   0.825  0.40942
GAM          3.34978    1.31571   2.546  0.01090 *
rf           3.15508    0.99302   3.177  0.00149 **
NN          -0.02485    0.92515  -0.027  0.97857
SVM          3.52080    4.33424   0.812  0.41661
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 513.96  on 499  degrees of freedom
```

```
Residual deviance: 334.92  on 493  degrees of freedom
AIC: 348.92

Number of Fisher Scoring iterations: 5
```

```
summary(mbmcGAM <- gam(train2$Angaus ~ s(GLM) + s(GLMbic) + s(GAM) + s(rf) + s(NN)  +s(SVM),
                       family=binomial, method="ML", select=F, data=as.data.frame(mbmcpreds2)))
```

```

Family: binomial
Link function: logit

Formula:
train2$Angaus ~ s(GLM) + s(GLMbic) + s(GAM) + s(rf) + s(NN) +
    s(SVM)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -2.2195     0.2223  -9.984   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
            edf Ref.df Chi.sq p-value
s(GLM)    1.000  1.000  0.100 0.75231
s(GLMbic) 1.000  1.000  0.454 0.50036
s(GAM)    2.387  3.046 12.440 0.00648 **
s(rf)     1.000  1.000  8.439 0.00367 **
s(NN)     1.000  1.000  0.195 0.65840
s(SVM)    1.000  1.000  0.005 0.94201
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =   0.35   Deviance explained = 36.6%
-ML = 166.97  Scale est. = 1         n = 500
```

```
(mbmcRF <- randomForest(as.factor(train2$Angaus) ~ ., data=as.data.frame(mbmcpreds2)))
```

```

Call:
 randomForest(formula = as.factor(train2$Angaus) ~ ., data = as.data.frame(mbmcpreds2))
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 17.6%
Confusion matrix:
    0  1 class.error
0 366 29  0.07341772
1  59 46  0.56190476
```

```
# now supra-predict to test data:
colnames(allpreds.test) <- colnames(mbmcpreds2)
lltest(predict(mbmcGLM, newdata=as.data.frame(allpreds.test), type="response"),
       obs=Anguilla_test$Angaus)
```

```
[1] -266.3385
```

```
lltest(predict(mbmcGAM, newdata=as.data.frame(allpreds.test), type="response"),
       obs=Anguilla_test$Angaus)
```

```
[1] -197.814
```

```
lltest(predict(mbmcRF, newdata=as.data.frame(allpreds.test), type="prob")[,2],
       obs=Anguilla_test$Angaus)
```

```
[1] -201.3224
```

14 Summary of results so far

Across the model-averaging approaches illustrated here, equal-weights (or $1/M$ if you like) performed extremely well. Even if on a new data set it may be outperformed by one of the others, here it had the edge over all the more sophisticated and computer-intensive approaches. The league table looks like this:

Table 1: Table 1. Log-likelihoods on test data for all approaches, from best to worst. * indicates an individual model, all others are model averaging approaches.

Approach	log-likelihood
equal weight (median)	-182.84
LOO (RMSE)	-184.82
equal weight (mean)	-184.86
cos-squared	-185.02
BMA-EM	-185.24
stacking	-186.82
randomForst	-186.83
bootstrap	-186.83
Bates-Granger	-188.45
GAM	-193.40
ANN	-194.28
GLM.AIC	-197.48
GLM.BIC	-197.73
MBMC (GAM)	-198.23
MBMC (randomForest)	-200.20
SVM	-214.68
jackknife	-214.68
MBMC (GLM)	-268.52

15 Prediction distributions on new data points

Let us, entirely for illustrative purposes, predict to the data point with most extreme temperature, plus a bit (SegSumT=19.7). Since SegSumT is not particularly correlated with any climate-related variable in the data, we set all other data points to their median value.

```
round(cor(Anguilla_train[, -c(1,2,13,14)]), 3)
```

```
           SegSumT SegTSeas SegLowFlow DSDist DSMaxSlope USAvgT USRainDays USSlope USNative  DSDam
SegSumT      1.000    0.255     -0.013 -0.356     -0.403 -0.017      0.042  -0.219   -0.344 -0.227
```

```
SegTSeas      0.255    1.000     -0.047 -0.644     -0.186  0.411      0.352  -0.239   -0.172 -0.547
SegLowFlow   -0.013   -0.047      1.000  0.048     -0.092 -0.421      0.108   0.161    0.064  0.020
DSDist       -0.356   -0.644      0.048  1.000      0.297 -0.238     -0.272   0.134    0.157  0.584
DSMaxSlope   -0.403   -0.186     -0.092  0.297      1.000 -0.066     -0.043   0.338    0.357  0.283
USAvgT       -0.017    0.411     -0.421 -0.238     -0.066  1.000      0.014  -0.581   -0.357 -0.230
USRainDays    0.042    0.352      0.108 -0.272     -0.043  0.014      1.000   0.102    0.223 -0.213
USSlope      -0.219   -0.239      0.161  0.134      0.338 -0.581      0.102   1.000    0.640  0.076
USNative     -0.344   -0.172      0.064  0.157      0.357 -0.357      0.223   0.640    1.000  0.131
DSDam        -0.227   -0.547      0.020  0.584      0.283 -0.230     -0.213   0.076    0.131  1.000
```

```
newdatum <- cbind.data.frame(SegSumT=19.7, t(apply(Anguilla_train[, -c(1,2,3,13,14)], 2, median,
                                                   na.rm=T)))
```

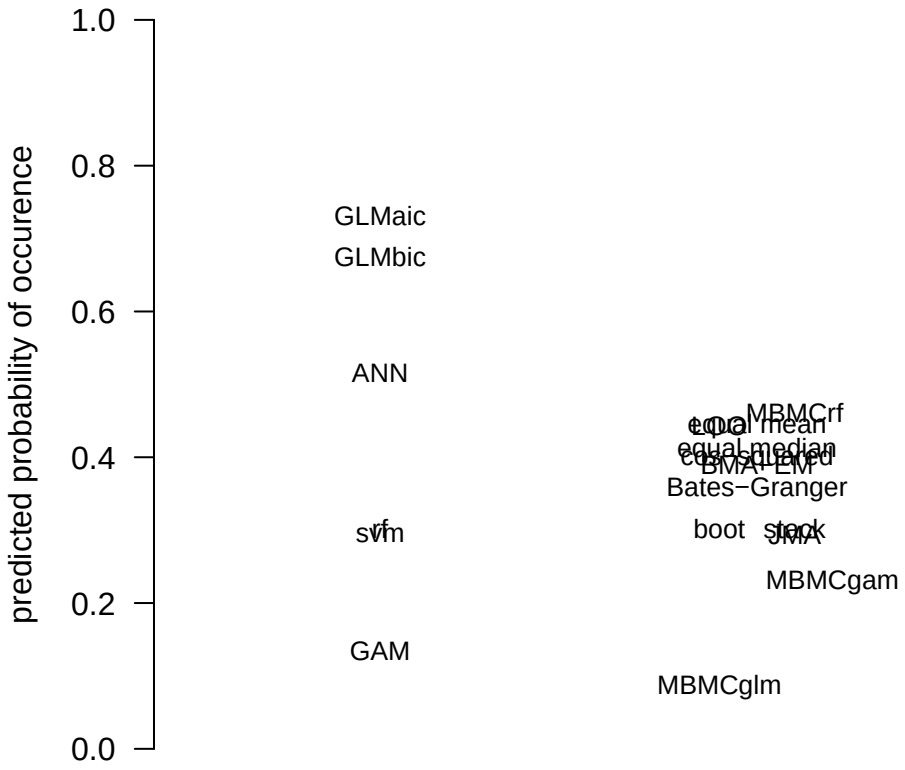
We next compute the single model's point predictions:

```
pred4new.glmAIC <- predict(fglm, newdata=newdatum, type="response")
pred4new.glmBIC <- predict(fglmBIC, newdata=newdatum, type="response")
pred4new.gam <- predict(fgam, newdata=newdatum, type="response")
pred4new.rf <- predict(frf, newdata=newdatum, type="prob")[,2]
pred4new.nn <- predict(fnn, newdata=newdatum, type="raw")
pred4new.svm <- attr(predict(fsvm, newdata=newdatum, probability=TRUE), "probabilities")[,2]
# put them together for convenience:
pred4new.singles <- c(pred4new.glmAIC, pred4new.glmBIC, pred4new.gam, pred4new.rf,
                      pred4new.nn, pred4new.svm)
names(pred4new.singles) <- colnames(mbmcpreds2)
```

We can now compute model-average predictions, in the same sequence as above.

```
## equal weight:
pred4newMAequalmean <- mean(pred4new.singles)
pred4newMAequalmedian <- median(pred4new.singles)
## LOO
pred4newLOOrmse <- pred4new.singles %*% weightsLOO
## BMA
pred4newEBMA <- pred4new.singles %*% EBMAfit@modelWeights
## bootstrap
pred4newBoot <- pred4new.singles %*% weightsboot
## stacking
pred4newStack <- pred4new.singles %*% weightsStacking
## JMA
pred4newJMA <- pred4new.singles %*% weightsJMA
## Bates-Granger
pred4newBG <- pred4new.singles %*% t(weightsBG)
## cos-squared
pred4newCS <- pred4new.singles %*% (weightsCS)
## MBMCglm
pred4newMBMCglm <- predict(mbmcGLM, newdata=data.frame(t(pred4new.singles)), type="response")
## MBMCgam
pred4newMBMCgam <- predict(mbmcGAM, newdata=data.frame(t(pred4new.singles)), type="response")
## MBMCrf
pred4newMBMCrf <- predict(mbmcRF, newdata=data.frame(t(pred4new.singles)), type="prob")[2]

# Depict point predictions:
par(mar=c(1,5,1,1))
plot(c(0,3), c(0,1), ylim=c(0,1), las=1, xlim=c(0.5,3), type="n",
     ylab="predicted probability of occurence", xlab="", axes=F)
```

```
text( c(rep(1, 6), c(2,2,1.9,2,1.9, 2.1,2.1,2,2,1.9, 2.2, 2.1)),
      c(pred4new.glmAIC, pred4new.glmBIC, pred4new.gam, pred4new.rf, pred4new.nn,
        pred4new.svm,   pred4newMAequalmean, pred4newMAequalmedian,pred4newLOOrmse,
        pred4newEBMA, pred4newBoot, pred4newStack, pred4newJMA, pred4newBG, pred4newCS,
        pred4newMBMCglm, pred4newMBMCgam, pred4newMBMCrf),
     c("GLMaic", "GLMbic", "GAM", "rf", "ANN", "svm", "equal mean", "equal median", "LOO",
       "BMA-EM", "boot",  "stack", "JMA", "Bates-Granger", "cos-squared", "MBMCglm",
       "MBMCgam", "MBMCrf"), cex=0.8)
axis(side=2, las=1)
```



So all but one model averages constrain the extreme predictions of GLMs and GAM. When using model-based model combinations, intercept and/or coefficients for the models may be negative, leading to predictions *outside* those of the individuals (as here the case for MBMCglm).

To get uncertainty bounds for each model and thus the their prediction distribution, we bootstrap the models. We don't have to bootstrap GLM and GAM, since they provide error bounds for their point prediction. Since we want to use them as individual points, rather than intervals, we draw from them as as if they were bootstrapped.

```
if (!("App_case2_bootpreds.Rdata" %in% dir())){
  NBOOTSTRAPS <- 1000
  # bootstrap models for RF, ANN and SVM:
  frf.boots <- fnn.boots <- fsvm.boots <- list() # list of bootstrapped models
  for (b in 1:NBOOTSTRAPS){# rerun the non-parametric models; takes about 1 sec per bootstrap!
    bsdata <- Anguilla_train[sample(1000, 1000, replace=T), ]
    frf.boots[[b]] <- randomForest(as.factor(Angaus) ~ SegSumT + SegSumT + SegTSeas +
                        SegLowFlow + DSDist + DSMaxSlope + USAvgT + USRainDays + USSlope +
                          USNative, data=bsdata)
    fnn.boots[[b]] <- nnet(Angaus ~ SegSumT + SegSumT + SegTSeas + SegLowFlow +
                       DSDist + DSMaxSlope + USAvgT + USRainDays + USSlope +
```

```
                        USNative, data=bsdata, size=11, maxit=500, decay=0.1, trace=F)
    fsvm.boots[[b]] <- svm(x=as.matrix(bsdata[,c(3:12)]), y=as.factor(bsdata$Angaus),
                           type="C-classification", probability=TRUE, kernel="radial")
    rm(bsdata)
    #cat(b, " ")
  }
  save(frf.boots, fnn.boots, fsvm.boots, file="App_case2_bootpreds.Rdata")
} else {
  load(file="App_case2_bootpreds.Rdata")
  NBOOTSTRAPS <- 1000
}
```

```
load(file="App_case2_bootpreds.Rdata")
NBOOTSTRAPS <- 1000
```

Now draw from each model's prediction distribution:

```
NBOOTSTRAPS <- 1000
# from GLMs and GAM:
pred4newBS.glmAIC <- plogis(rnorm(NBOOTSTRAPS, mean=predict(fglm, newdata=newdatum,
                se.fit=T)$fit, sd=predict(fglm, newdata=newdatum, se.fit=T)$se.fit))
pred4newBS.glmBIC <- plogis(rnorm(NBOOTSTRAPS, mean=predict(fglmBIC, newdata=newdatum,
                se.fit=T)$fit, sd=predict(fglmBIC, newdata=newdatum, se.fit=T)$se.fit))
pred4newBS.gam <- plogis(rnorm(NBOOTSTRAPS, mean=predict(fgam, newdata=newdatum,
                se.fit=T)$fit, sd=predict(fgam, newdata=newdatum, se.fit=T)$se.fit))
# for the non-likelihood models:
pred4newBS.rf <- pred4newBS.nn <- pred4newBS.svm <- numeric(NBOOTSTRAPS)
for (b in 1:NBOOTSTRAPS){
    pred4newBS.rf[b] <- predict(frf.boots[[b]], newdata=newdatum, type="prob")[,2]
    pred4newBS.nn[b] <- predict(fnn.boots[[b]], newdata=newdatum, type="raw")
    pred4newBS.svm[b] <- attr(predict(fsvm.boots[[b]], newdata=newdatum, probability=TRUE),
                              "probabilities")[,2]
}
# put all predictions together:
pred4newBS.singles <- cbind(pred4newBS.glmAIC, pred4newBS.glmBIC, pred4newBS.gam,
                            pred4newBS.rf, pred4newBS.nn, pred4newBS.svm)
colnames(pred4newBS.singles) <- colnames(mbmcpreds2)
```

Here it becomes briefly slightly ambiguous. We have no *a priori* reason to believe that the predictions are correlated, in principle. Since they were fitted to the same bootstrapping, they may well be (not in this particular case, though: we checked). But the predictions *conditional on the data* should be uncorrelated. It may be worth considering shuffling the bootstrapped predictions within a method to break any spurious correlation arising from the fit on the same bootstrap data.

First, we compute the variance of the model averaged prediction, $var(\widetilde{Y})$, based on the bootstraps:

```
## equal weights, mean:
weightsEqual <- rep(1/6, 6)
var(pred4newBS.singles %*% weightsEqual)
```

```
             [,1]
[1,] 0.002599964
```

```
# which is the same as:
var(apply(pred4newBS.singles, 1, mean))
```

```
[1] 0.002599964
```

```
## equal weights, median:
var(apply(pred4newBS.singles, 1, median))
```

```
[1] 0.009453086
```

```
## LOO:
var(pred4newBS.singles %*% weightsLOO)
```

```
             [,1]
[1,] 0.002556839
```

```
## EMBA:
var(pred4newBS.singles %*% EBMAfit@modelWeights)
```

```
             [,1]
[1,] 0.001723225
```

```
## boot:
var(pred4newBS.singles %*% weightsboot)
```

```
             [,1]
[1,] 0.006139323
```

```
## stack:
var(pred4newBS.singles %*% weightsStacking)
```

```
             [,1]
[1,] 0.006135053
```

```
## JMA:
var(pred4newBS.singles %*% weightsJMA)
```

```
            [,1]
[1,] 0.02551818
```

```
## BG:
var(pred4newBS.singles %*% t(weightsBG))
```

```
             [,1]
[1,] 0.003355864
```

```
## CS:
var(pred4newBS.singles %*% weightsCS)
```

```
             [,1]
[1,] 0.003576009
```

```
## MBMBglm:
var(predict(mbmcGLM, newdata=as.data.frame(pred4newBS.singles), type="response"))
```

```
[1] 0.01097245
```

```
## MBMBgam:
var(predict(mbmcGAM, newdata=as.data.frame(pred4newBS.singles), type="response"))
```

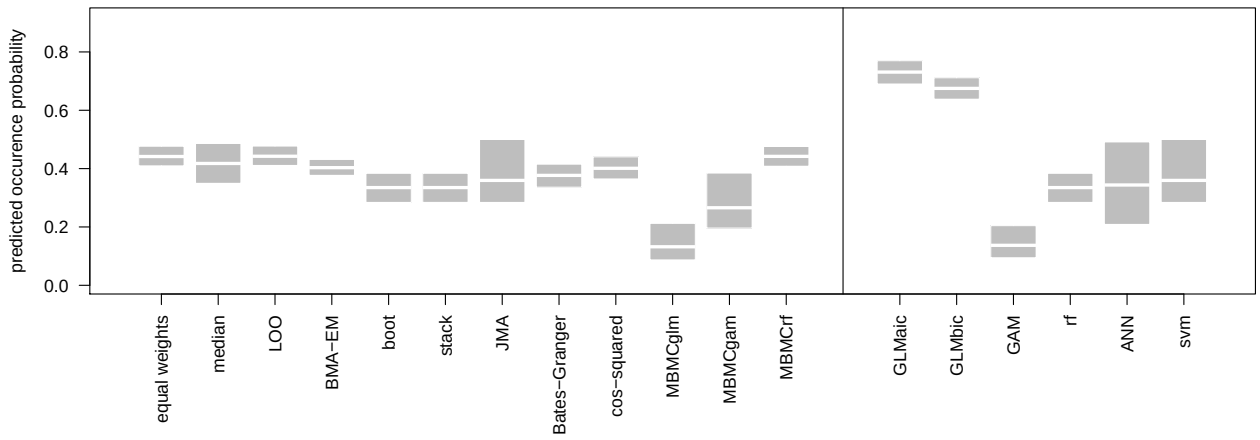
```
[1] 0.01576325
```

```
## MBMBgam:
var(predict(mbmcRF, newdata=as.data.frame(pred4newBS.singles), type="prob")[,2])
```

```
[1] 0.004191907
```

These estimates of the variance of averaged predictions are relatively close together, apart from equal-median, JMA and two of the MBMCs. We can visualise predictions and their variability as boxplots, alternatively.

```
par(mar=c(8,4,1,1))
boxplot(cbind(pred4newBS.singles %*% weightsEqual,
              apply(pred4newBS.singles, 1, median),
              pred4newBS.singles %*% weightsLOO,
              pred4newBS.singles %*% EBMAfit@modelWeights,
              pred4newBS.singles %*% weightsboot,
              pred4newBS.singles %*% weightsStacking,
              pred4newBS.singles %*% weightsJMA,
              pred4newBS.singles %*% t(weightsBG),
              pred4newBS.singles %*% weightsCS,
              predict(mbmcGLM, newdata=as.data.frame(pred4newBS.singles), type="response"),
              predict(mbmcGAM, newdata=as.data.frame(pred4newBS.singles), type="response"),
              predict(mbmcRF, newdata=as.data.frame(pred4newBS.singles), type="prob")[,2],
              pred4newBS.singles),
        border="white", col="grey", las=2, ylab="predicted occurence probability",
        names=c("equal weights", "median", "LOO", "BMA-EM", "boot","stack", "JMA", "Bates-Granger",
           "cos-squared","MBMCglm", "MBMCgam", "MBMCrf","GLMaic", "GLMbic", "GAM", "rf",
           "ANN", "svm"), at=c(1:12, 14:19), horizontal=F)
abline(v=13)
```

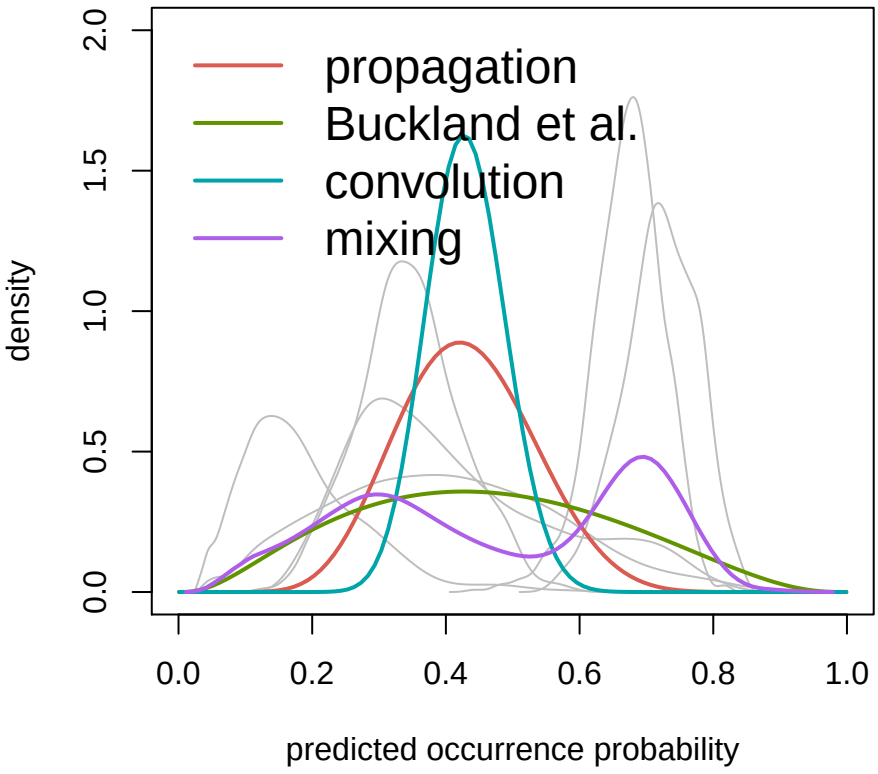


16 Quantifying model-averaged prediction uncertainty: confidence distributions of the prediction

```
predictMA <- function(Preds, weights, type, PredsSE=NULL, N=1E5, PredsBS=NULL){
  # function to compute confidence distributions from averaged predictions
  # if used for non-normal data, all data have to be provided AT THE LINK SCALE!

  if (length(Preds) != length(weights)) stop("The number of models in Preds is
                                        different from the number of weights.")

  if (type == "propagation"){
    if (is.null(PredsBS)) stop("Please provide matrix with bootstrap estimates
                               for Preds from each model (at least 500)!")
    # compute model misspecification bias as deviation of each prediction from the weighted mean:
```

```
    biasBS <- (PredsBS - matrix(PredsBS %*% weights, ncol=NCOL(PredsBS), nrow=NROW(PredsBS)))
    # we estimate model misspecification bias gamma as the mean bias (colMeans(bias)):
    gamma_m <- colMeans(biasBS)   # mean for each run of the bootstrap; similar to original data!
    # estimate variance of prediction from bootstrap:
    varY_m <- apply(PredsBS, 2, var) # from across the bootstraps
    # the substitution for the covariance term in eqn 16 is:
    undertheroot <- tcrossprod(varY_m + gamma_m^2)
    diag(undertheroot) <- 0
    # now fill in eqn 16, using gamma_m from bootstrapped data!:
    varYtilde <- sum(weights^2 * (varY_m + gamma_m^2)) + sum(weights^2 *
                                      cov2cor(cov(PredsBS))*sqrt(undertheroot))
    out <- function(x) dnorm(x, mean=mean(PredsBS %*% weights), sd=sqrt(varYtilde))
  }

  if (type == "Buckland"){
    vars <- PredsSE^2
    gamma <- Preds - Preds %*% weights # model misspecification bias
    varBuckland <- sum(weights*sqrt(vars + gamma^2))^2 # var according to Buckland et al.
    out <- function(x) dnorm(x, mean=mean(Preds %*% weights), sd=sqrt(varBuckland))
  }

  if (type == "mixing"){
    warning("You requested mixing. At present, this function assumes your predictions are
            AT THE LINK SCALE and normally distributed.")
    mix <- unlist(sapply(seq_along(Preds), function(i) rnorm(round(N*weights)[i],
                                                  mean=Preds[i], sd=PredsSE[i])))
    dx <- density(mix, n=N/20)
    out <- approxfun(dx)
  }

  if (type == "convolution"){
    warning("You requested a convolution. At present, this function assumes your
            predictions are normally distributed.")
    if (is.null(PredsSE)) stop("For convolution, please provide estimates for the
                    prediction standard error of each prediction, akin to preds).")
    meanconv <- Preds %*% weights
    sdconv <- sqrt(sum(weights^2*PredsSE^2))
    out <- function(x) dnorm(x, mean=meanconv, sd=sdconv)
  }
  return(out)
}
# Let's assume equal weights (but any other weights can be used here):
weights <- rep(1/6, 6)
# now prepare the data accordingly; note NOT to use the predictions above, since they are
# at the response scale, and we want the link-scale!
newdatum <- cbind.data.frame(SegSumT=19.7, t(apply(Anguilla_train[, -c(1,2,3,13,14)], 2,
                                                   median, na.rm=T)))
# from GLMs and GAM:
pred4newBS.glmAIC <- rnorm(NBOOTSTRAPS, mean=predict(fglm, newdata=newdatum, se.fit=T)$fit,
                           sd=predict(fglm, newdata=newdatum, se.fit=T)$se.fit)
pred4newBS.glmBIC <- rnorm(NBOOTSTRAPS, mean=predict(fglmBIC, newdata=newdatum, se.fit=T)$fit,
                           sd=predict(fglmBIC, newdata=newdatum, se.fit=T)$se.fit)
pred4newBS.gam <- rnorm(NBOOTSTRAPS, mean=predict(fgam, newdata=newdatum, se.fit=T)$fit,
```

```
                        sd=predict(fgam, newdata=newdatum, se.fit=T)$se.fit)
# for the non-likelihood models (transform to logit link scale!):
pred4newBS.rf <- pred4newBS.nn <- pred4newBS.svm <- numeric(NBOOTSTRAPS)
for (b in 1:NBOOTSTRAPS){
    pred4newBS.rf[b] <- qlogis(predict(frf.boots[[b]], newdata=newdatum, type="prob")[,2])
    pred4newBS.nn[b] <- qlogis(predict(fnn.boots[[b]], newdata=newdatum, type="raw"))
    pred4newBS.svm[b] <- qlogis(attr(predict(fsvm.boots[[b]], newdata=newdatum,
                                             probability=TRUE), "probabilities")[,2])
}
# put all predictions together:
pred4newBS.singles <- cbind(pred4newBS.glmAIC, pred4newBS.glmBIC, pred4newBS.gam,
                            pred4newBS.rf, pred4newBS.nn, pred4newBS.svm)
#head(pred4newBS.singles)

# collect point predictions from original models:
Preds <- c(predict(fglm, newdata=newdatum), predict(fglmBIC, newdata=newdatum),
           predict(fgam, newdata=newdatum), qlogis(predict(frf, newdata=newdatum,
           type="prob")[,2]), qlogis(predict(fnn, newdata=newdatum, type="raw")),
           qlogis(attr(predict(fsvm, newdata=newdatum, probability=TRUE),
                       "probabilities")[,2]))
# collect SE for these point predictions (from original model or bootstraps):
PredsSE <- apply(pred4newBS.singles, 2, sd)
# choose a sequence along which to plot the confidence distributions (at the response scale):
xseq <- seq(0.001, 0.999, len=100)

plot(1:2, 1:2, type="n", xlim=c(0,1), ylim=c(0,2), xlab="predicted occurrence probability",
     ylab="density")
for (i in 1:6){
  ds <- density(pred4newBS.singles[,i])
  lines(plogis(ds$x), ds$y, col="grey")
}

propagPred.fun <- predictMA(Preds, weights, PredsSE=PredsSE, type="propagation",
                            PredsBS=pred4newBS.singles)
lines(xseq, propagPred.fun(qlogis(xseq)), type="l", col="#DA5B51", lwd=2)

BuckPred.fun <- predictMA(Preds, weights, PredsSE=PredsSE, type="Buckland")
lines(xseq, BuckPred.fun(qlogis(xseq)), type="l", col="#629400", lwd=2)

convPred.fun <- predictMA(Preds, weights, type="convolution", PredsSE=PredsSE)
lines(xseq, convPred.fun(qlogis(xseq)), type="l", col="#00A5AA", lwd=2)

mixPred.fun <- predictMA(Preds, weights, PredsSE=PredsSE, type="mixing")
lines(xseq, mixPred.fun(qlogis(xseq)), type="l", col="#AE5EE9", lwd=2)

legend("topleft", bty="n", legend=c("propagation", "Buckland et al.", "convolution",
       "mixing"), lwd=2, col=c("#DA5B51", "#629400", "#00A5AA", "#AE5EE9"), cex=1.5)
```



16.1 Consensus - as in unanimous

Some publications propose using the "consensus" of models. In the context of binary response variables (and only there it has been proposed and seems plausible), this requires a thresholding of predictions into binary predictions: yes/no. It should be obvious that any such thresholding is (a) unnecessary and (b) incurs loss of information. In the context of stacking of probability predictions, Calabrese et al. (2014) additionally have shown that it will inevitably be suboptimal to the mixing of predictions shown above.

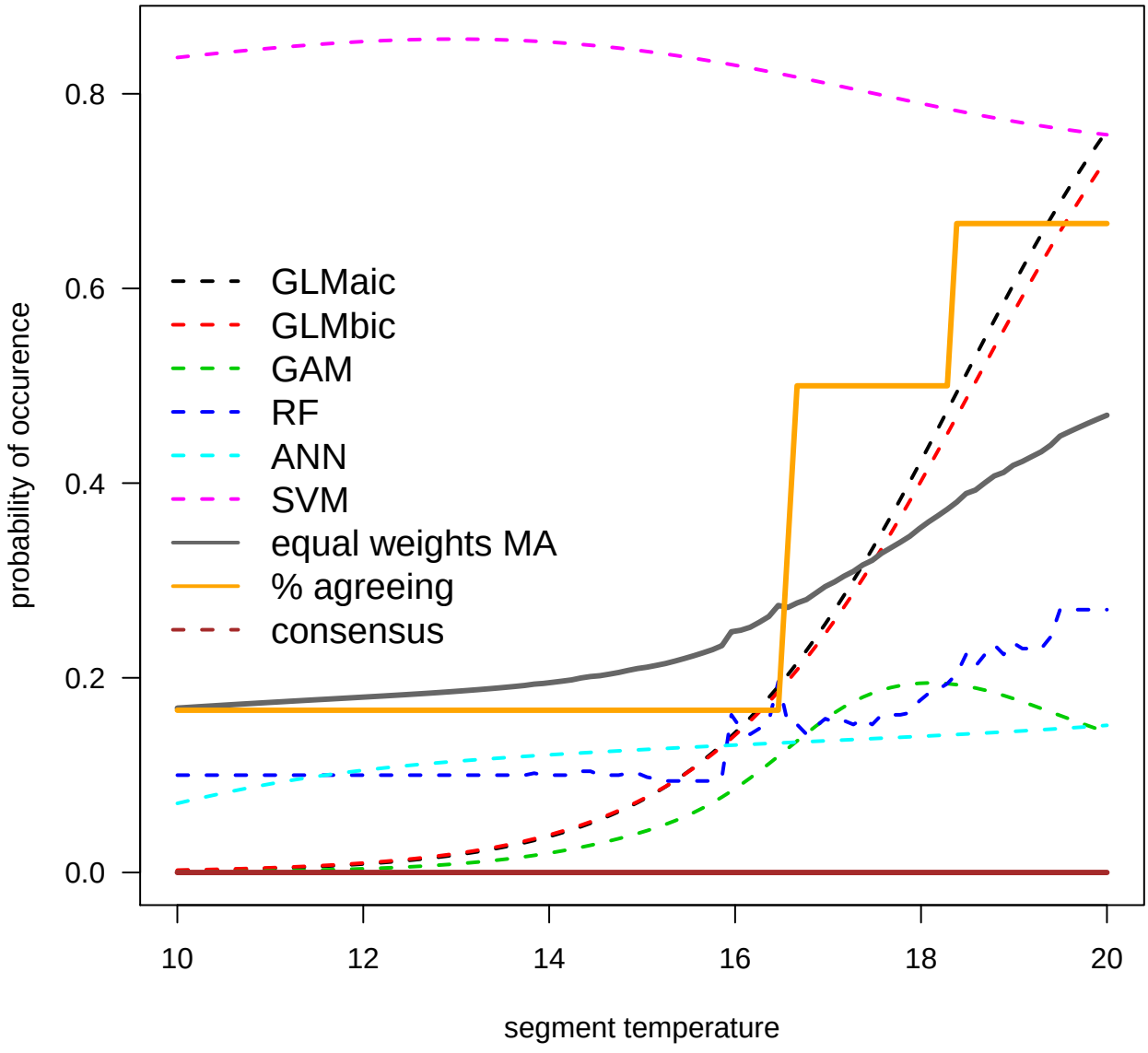
To illustrate the effect such a thresholded consensus prediction would have, we threshold predictions into 0/1 along a gradient of SegSumT and then compute the consensus prediction compared to the averaged prediction.

```
newSegSumT <- seq(10, 20, len=100)
newdats2 <- cbind.data.frame(SegSumT=newSegSumT, t(apply(Anguilla_train[, -c(1,2,3,13,14)],
                                                         2, median, na.rm=T)))
(prev=mean(Anguilla_train$Angaus)) # the prevalence of the species in this data set
```

```
[1] 0.202
```

```
temppreds <- cbind("GLM"=predict(ebmafglm, newdata=newdats2, type="response"),
                   "GLMbic"=predict(ebmafglmBIC, newdata=newdats2, type="response"),
                   "GAM"=predict(ebmafgam, newdata=newdats2, type="response"),
                   "RF"=predict(ebmafrf, newdata=newdats2, type="prob")[,2],
                   "ANN"=as.numeric(predict(ebmafnn, newdata=newdats2, type="raw")),
                   "SVM"=attr(predict(ebmafsvm, probability=TRUE, newdata=newdats2),
                              "probabilities")[,2])
temppreds.thresh <- apply(temppreds, 2, function(x) ifelse(x > prev, 1, 0))
RS <- rowSums(temppreds.thresh)
consensus <- ifelse(RS == 6, 1, 0)
# visualise:
```

```
matplot(newSegSumT, temppreds, type="l", lwd=2, lty=2, las=1, ylab="probability of occurence",
        xlab="segment temperature")
lines(newSegSumT, rowMeans(temppreds), cex=2, type="l", lwd=3, col="grey40")
legend("left", bty="n", col=c(1:6, "grey40", "orange", "brown"), legend=c("GLMaic", "GLMbic",
          "GAM", "RF", "ANN", "SVM", "equal weights MA", "% agreeing", "consensus"), cex=1.25,
          lty=c(rep(2, 6), 1,1), lwd=2)
lines(newSegSumT, RS/6, cex=2, type="l", lwd=3, col="orange")
lines(newSegSumT, consensus, cex=2, type="l", lwd=3, col="brown")
```



It seems obvious that neither the proportion above the threshold nor the consensus is a reasonable way to average models.