Bagging Booting Stacking

鈴木瑞人
東京大学大学院　新領域創成科学研究科
メディカル情報生命専攻
博士課程1年

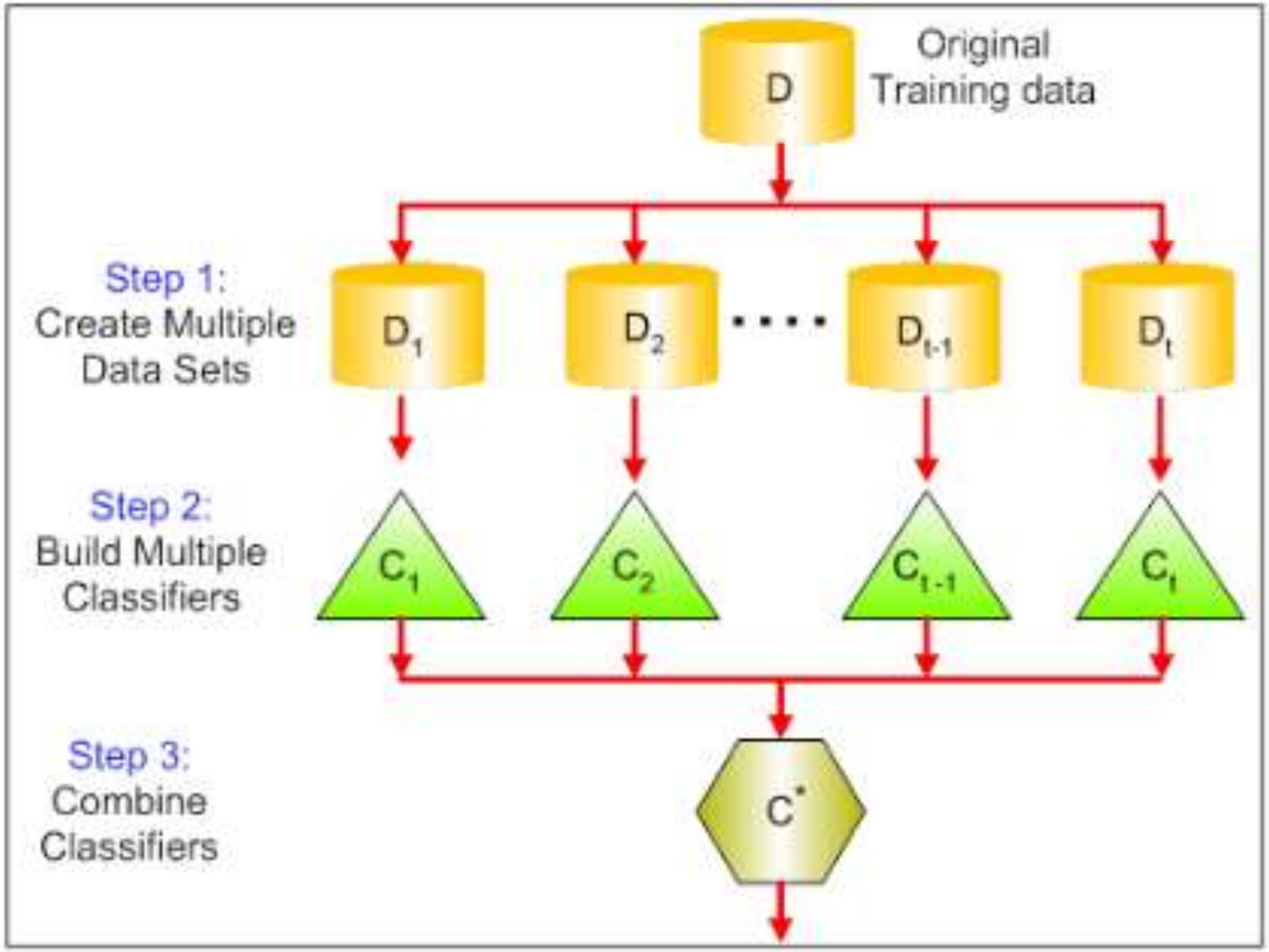
Bagging, Boosting, Stackingとは?

https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/

Bagging

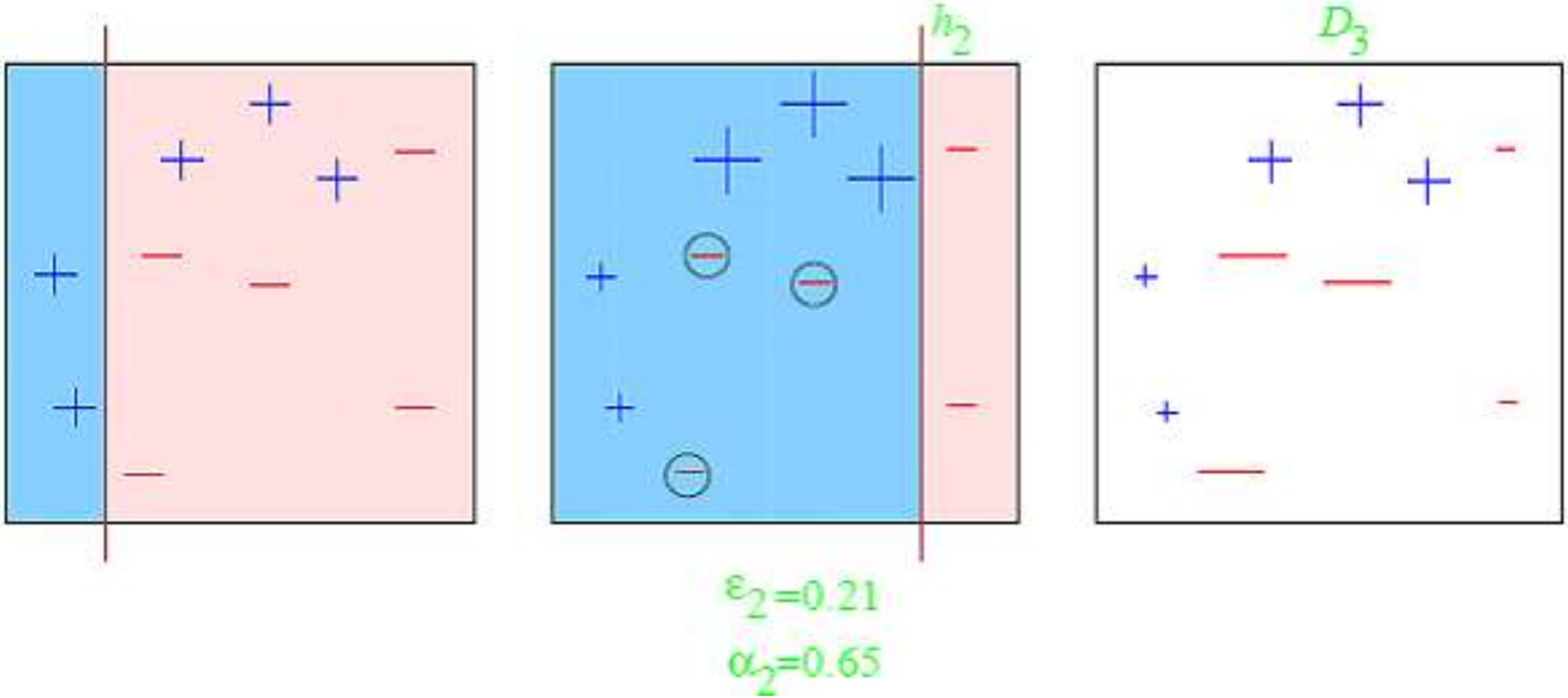
- （ブートストラップ集約）がアンサンブル方法です。まず、トレーニングデータセットのランダムサンプル（訓練データセットの部分集合）を作成。
- そこで、各サンプルの分類器を構築。
- 最後に、これらの複数の分類器の結果は、平均値または過半数の投票を使用して合成。

Bagging


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D1
D2
Dt-1
Dt
Step 2:
Build Multiple
Classifiers
C1
C2
Ct-1
Ct
Step 3:
Combine
Classifiers
C*

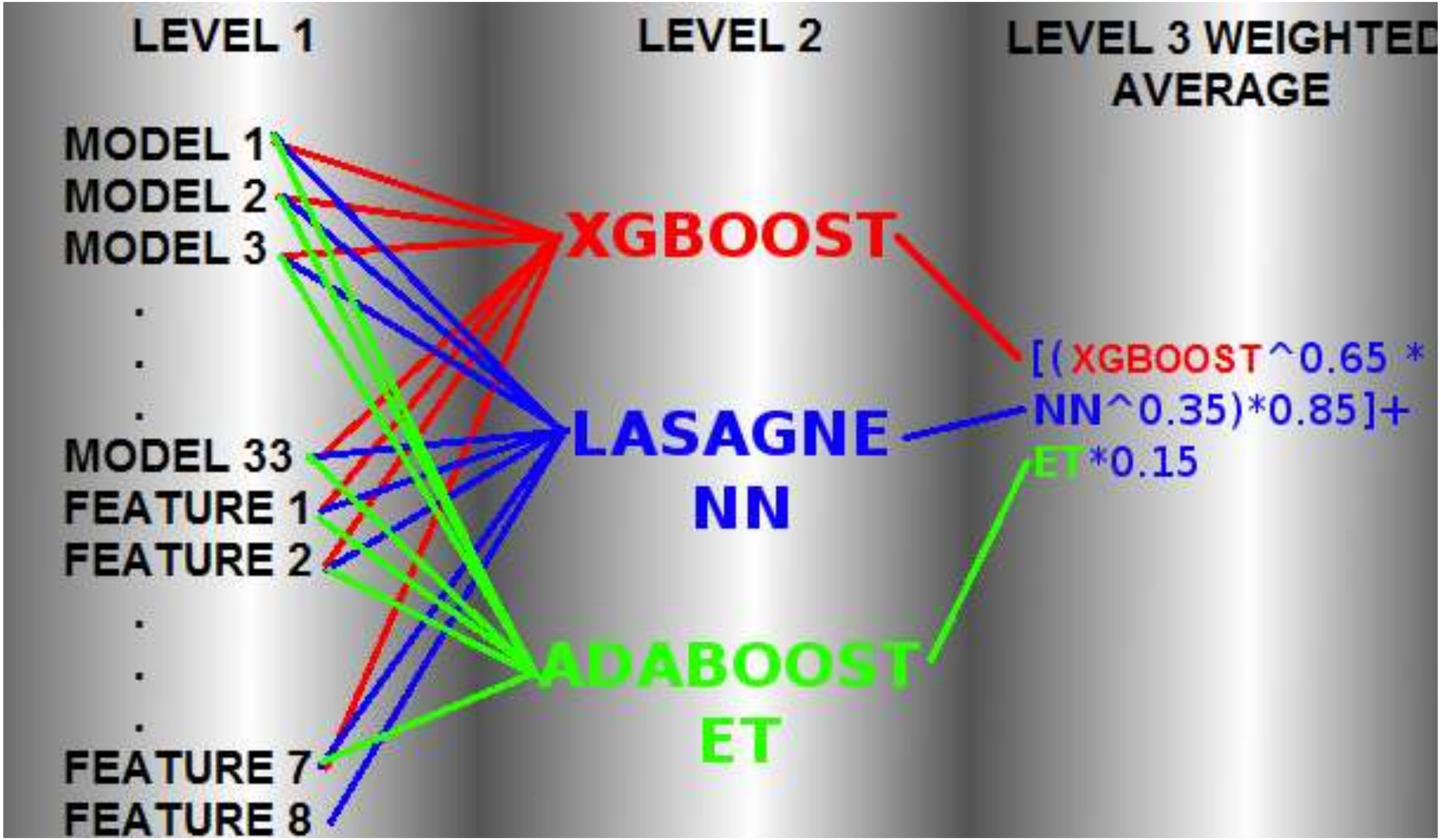
Boosting

- Boostingは初めの学習はすべてのデータを用いて行う。
- 一方でそれに続く学習は、その前に行った学習のパフォーマンスに基づいた訓練データセットに基づいて行われる。
- オリジナルデータセットを分類し、各観測対象に等しい重みを与えることから始まる。
- 前の学習機によって不正確に予測された観測対象はより高い重みを与えられる。
- 正確さの限界か、あらかじめ与えられたモデル数の限界に達するまで繰り返しプロセスは実行される。
- BoostingはBaggingより予測精度が高いことが示されている。しかし、過学習の傾向もある。

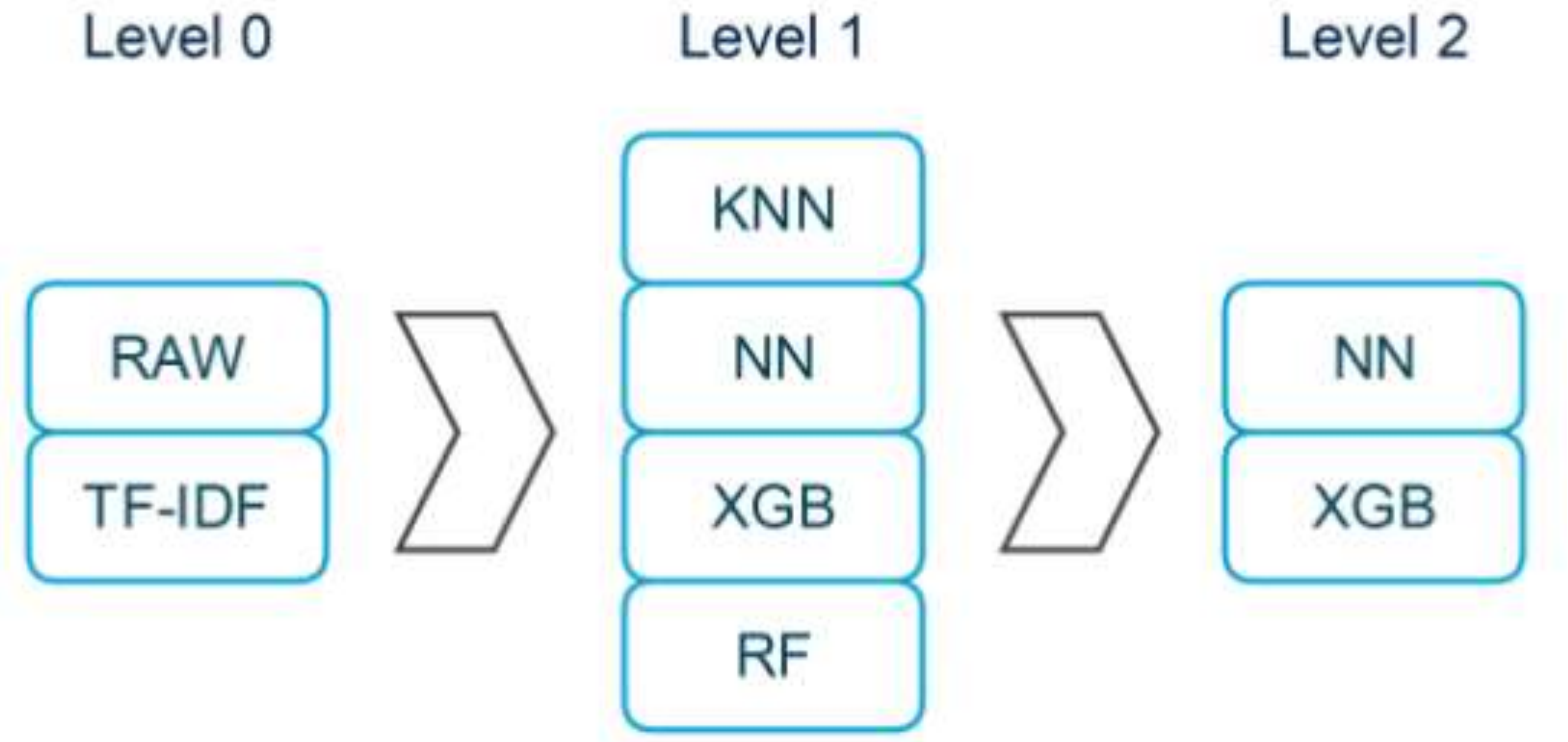


h_2
D_3
ε_2=0.21
α_2=0.65

Stacking



LEVEL 1
LEVEL 2
LEVEL 3 WEIGHTED AVERAGE
MODEL 1
MODEL 2
MODEL 3
MODEL 33
FEATURE 1
FEATURE 2
FEATURE 7
FEATURE 8
XGBOOST
LASAGNE NN
ADABOOST ET
[(XGBOOST^0.65 *
NN^0.35)*0.85]+
ET*0.15



MY SOLUTION'S STACKING SCHEMA

```r
install.packages("mlbench",quiet=T, dependencies=T)
install.packages("caret",quiet=T, dependencies=T)
install.packages("caretEnsemble",quiet=T, dependencies=T)
```

```r
# Load libraries
library(mlbench)
library(caret)
library(caretEnsemble)

# Load the dataset
data(Ionosphere)
str(Ionosphere)
dataset <- Ionosphere
dataset <- dataset[,-2]
dataset$V1 <- as.numeric(as.character(dataset$V1))
```

```
> str(Ionosphere)
'data.frame':   351 obs. of  35 variables:
 $ V1   : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 1 2 2 ...
 $ V2   : Factor w/ 1 level "0": 1 1 1 1 1 1 1 1 1 1 ...
 $ V3   : num  0.995 1 1 1 1 ...
 $ V4   : num  -0.0589 -0.1883 -0.0336 -0.4516 -0.024 ...
 $ V5   : num  0.852 0.93 1 1 0.941 ...
 $ V6   : num  0.02306 -0.36156 0.00485 1 0.06531 ...
 $ V7   : num  0.834 -0.109 1 0.712 0.921 ...
 $ V8   : num  -0.377 -0.936 -0.121 -1 -0.233 ...
 $ V9   : num  1 1 0.89 0 0.772 ...
 $ V10  : num  0.0376 -0.0455 0.012 0 -0.164 ...
 $ V11  : num  0.852 0.509 0.731 0 0.528 ...
 $ V12  : num  -0.1776 -0.6774 0.0535 0 -0.2028 ...
 $ V13  : num  0.598 0.344 0.854 0 0.564 ...
 $ V14  : num  -0.44945 -0.69707 0.00827 0 -0.00712 ...
 $ V15  : num  0.605 -0.517 0.546 -1 0.344 ...
 $ V16  : num  -0.38223 -0.97515 0.00299 0.14516 -0.27457 ...
 $ V17  : num  0.844 0.055 0.838 0.541 0.529 ...
 $ V18  : num  -0.385 -0.622 -0.136 -0.393 -0.218 ...
 $ V19  : num  0.582 0.331 0.755 -1 0.451 ...
 $ V20  : num  -0.3219 -1 -0.0854 -0.5447 -0.1781 ...
 $ V21  : num  0.5697 -0.1315 0.7089 -0.6997 0.0598 ...
 $ V22  : num  -0.297 -0.453 -0.275 1 -0.356 ...
 $ V23  : num  0.3695 -0.1806 0.4339 0 0.0231 ...
 $ V24  : num  -0.474 -0.357 -0.121 0 -0.529 ...
 $ V25  : num  0.5681 -0.2033 0.5753 1 0.0329 ...
 $ V26  : num  -0.512 -0.266 -0.402 0.907 -0.652 ...
 $ V27  : num  0.411 -0.205 0.59 0.516 0.133 ...
 $ V28  : num  -0.462 -0.184 -0.221 1 -0.532 ...
 $ V29  : num  0.2127 -0.1904 0.431 1 0.0243 ...
 $ V30  : num  -0.341 -0.116 -0.174 -0.201 -0.622 ...
 $ V31  : num  0.4227 -0.1663 0.6044 0.2568 -0.0571 ...
 $ V32  : num  -0.5449 -0.0629 -0.2418 1 -0.5957 ...
 $ V33  : num  0.1864 -0.1374 0.5605 -0.3238 -0.0461 ...
 $ V34  : num  -0.453 -0.0245 -0.3824 1 -0.657 ...
 $ Class: Factor w/ 2 levels "bad","good": 2 1 2 1 2 1 2 1 2 1 ...
```

head(dataset)

```
> head(dataset)
  V1      V3       V4       V5       V6       V7       V8      V9      V10     V11
1  1 0.99539 -0.05889  0.85243  0.02306  0.83398 -0.37708 1.00000  0.03760 0.85243
2  1 1.00000 -0.18829  0.93035 -0.36156 -0.10868 -0.93597 1.00000 -0.04549 0.50874
3  1 1.00000 -0.03365  1.00000  0.00485  1.00000 -0.12062 0.88965  0.01198 0.73082
4  1 1.00000 -0.45161  1.00000  1.00000  0.71216 -1.00000 0.00000  0.00000 0.00000
5  1 1.00000 -0.02401  0.94140  0.06531  0.92106 -0.23255 0.77152 -0.16399 0.52798
6  1 0.02337 -0.00592 -0.09924 -0.11949 -0.00763 -0.11824 0.14706  0.06637 0.03786
       V12     V13      V14      V15      V16      V17      V18      V19      V20
1 -0.17755 0.59755 -0.44945  0.60536 -0.38223  0.84356 -0.38542  0.58212 -0.32192
2 -0.67743 0.34432 -0.69707 -0.51685 -0.97515  0.05499 -0.62237  0.33109 -1.00000
3  0.05346 0.85443  0.00827  0.54591  0.00299  0.83775 -0.13644  0.75535 -0.08540
4  0.00000 0.00000  0.00000 -1.00000  0.14516  0.54094 -0.39330 -1.00000 -0.54467
5 -0.20275 0.56409 -0.00712  0.34395 -0.27457  0.52940 -0.21780  0.45107 -0.17813
6 -0.06302 0.00000  0.00000 -0.04572 -0.15540 -0.00343 -0.10196 -0.11575 -0.05414
       V21      V22      V23      V24      V25      V26      V27      V28      V29
1  0.56971 -0.29674  0.36946 -0.47357  0.56811 -0.51171  0.41078 -0.46168  0.21266
2 -0.13151 -0.45300 -0.18056 -0.35734 -0.20332 -0.26569 -0.20468 -0.18401 -0.19040
3  0.70887 -0.27502  0.43385 -0.12062  0.57528 -0.40220  0.58984 -0.22145  0.43100
4 -0.69975  1.00000  0.00000  0.00000  1.00000  0.90695  0.51613  1.00000  1.00000
5  0.05982 -0.35575  0.02309 -0.52879  0.03286 -0.65158  0.13290 -0.53206  0.02431
6  0.01838  0.03669  0.01519  0.00888  0.03513 -0.01535 -0.03240  0.09223 -0.07859
       V30      V31      V32      V33      V34 Class
1 -0.34090  0.42267 -0.54487  0.18641 -0.45300  good
2 -0.11593 -0.16626 -0.06288 -0.13738 -0.02447   bad
3 -0.17365  0.60436 -0.24180  0.56045 -0.38238  good
4 -0.20099  0.25682  1.00000 -0.32382  1.00000   bad
5 -0.62197 -0.05707 -0.59573 -0.04608 -0.65697  good
6  0.00732  0.00000  0.00000 -0.00039  0.12011   bad
```

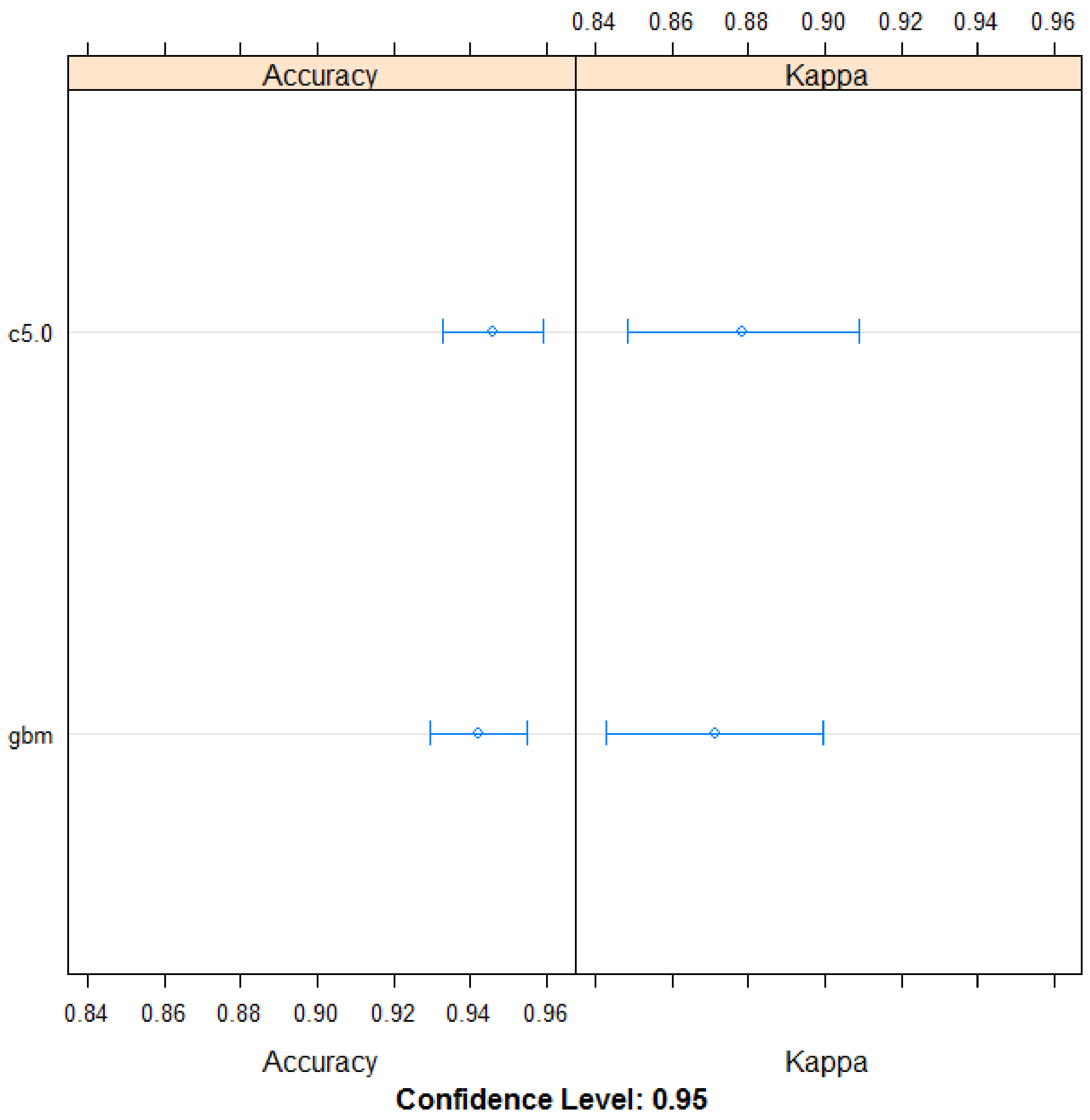
1. Boosting Algorithms

- Boostingアルゴリズム例
- C5.0
- Stochastic Gradient Boosting

```r
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
# C5.0
set.seed(seed)
fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric, trControl=control)
# Stochastic Gradient Boosting
set.seed(seed)
fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric, trControl=control, verbose=FALSE)
# 結果の要約
boosting_results <- resamples(list(c5.0=fit.c50, gbm=fit.gbm))
summary(boosting_results)
dotplot(boosting_results)
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)
seed <- 7
metric <- "Accuracy"
# C5.0
set.seed(seed)
fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric,
trControl=control)
# Stochastic Gradient Boosting
set.seed(seed)
fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric,
trControl=control, verbose=FALSE)
# 結果の要約
boosting_results <- resamples(list(c5.0=fit.c50, gbm=fit.gbm))
summary(boosting_results)
dotplot(boosting_results)
```

```
> summary(boosting_results)

Call:
summary.resamples(object = boosting_results)

Models: c5.0, gbm
Number of resamples: 30

Accuracy
       Min.  1st Qu. Median   Mean 3rd Qu. Max. NA's
c5.0 0.8824  0.9143 0.9437 0.9458  0.9714    1    0
gbm  0.8824  0.9143 0.9429 0.9420  0.9714    1    0

Kappa
       Min.  1st Qu. Median   Mean 3rd Qu. Max. NA's
c5.0 0.7244  0.8135 0.8745 0.8786  0.9378    1    0
gbm  0.7323  0.8135 0.8734 0.8713  0.9372    1    0
```

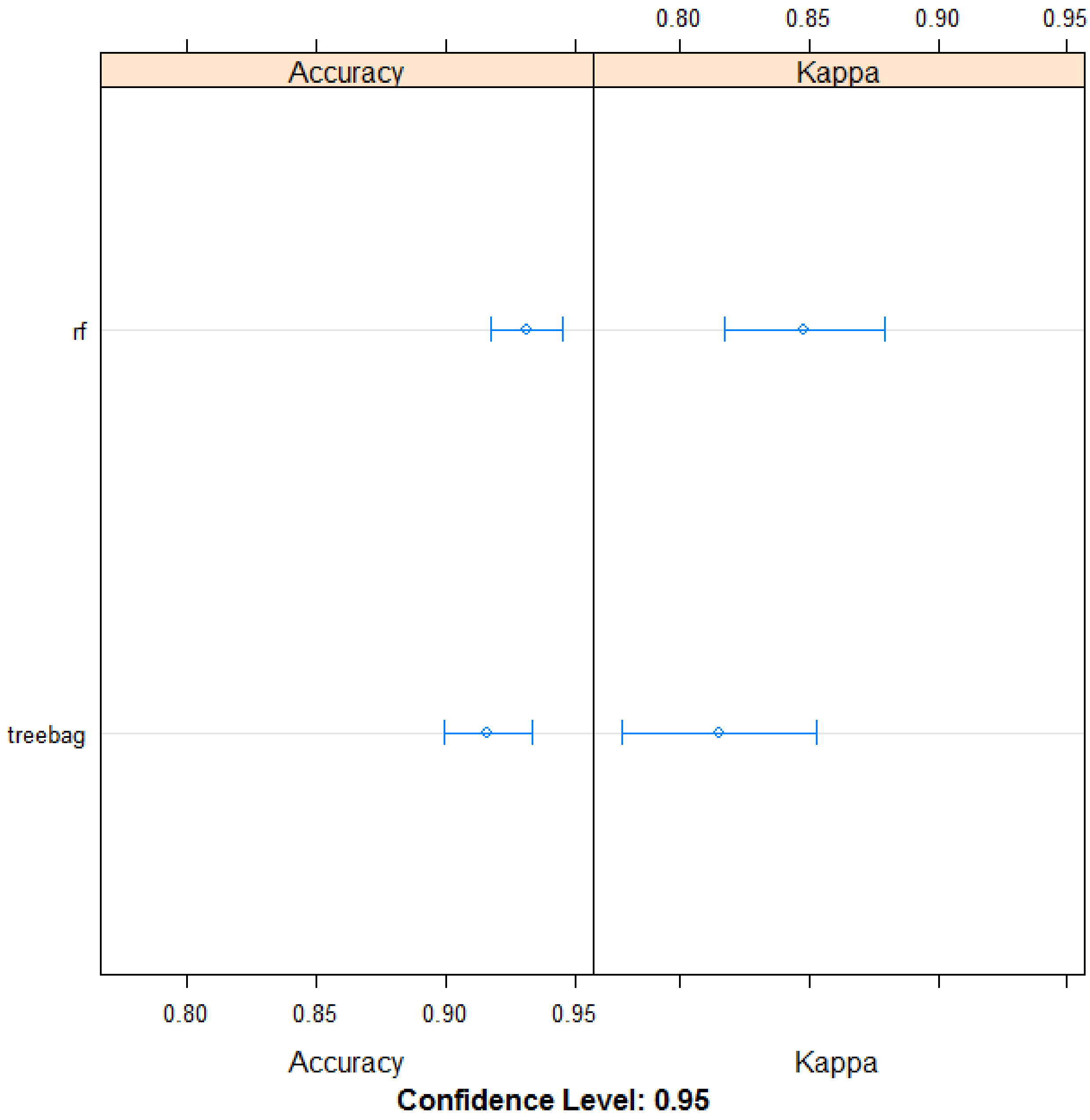

Accuracy
Kappa
c5.0
gbm
0.84
0.86
0.88
0.90
0.92
0.94
0.96
Confidence Level: 0.95

2. Bagging Algorithms

- Baggingアルゴリズム例
- Bagged CART
- Random Forest

```
# Example of Bagging algorithms
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
# Bagged CART
set.seed(seed)
fit.treebag <- train(Class~., data=dataset, method="treebag", metric=metric, trControl=control)
# Random Forest
set.seed(seed)
fit.rf <- train(Class~., data=dataset, method="rf", metric=metric, trControl=control)
# summarize results
bagging_results <- resamples(list(treebag=fit.treebag, rf=fit.rf))
summary(bagging_results)
dotplot(bagging_results)
```

```
> # summarize results
> bagging_results <- resamples(list(treebag=fit.treebag, rf=fit.rf))
> summary(bagging_results)

Call:
summary.resamples(object = bagging_results)

Models: treebag, rf
Number of resamples: 30

Accuracy
          Min.  1st Qu. Median   Mean 3rd Qu. Max. NA's
treebag 0.8235  0.8857 0.9155 0.9164  0.9444    1    0
rf      0.8571  0.9124 0.9155 0.9316  0.9444    1    0

Kappa
          Min.  1st Qu. Median   Mean 3rd Qu. Max. NA's
treebag 0.5984  0.7430 0.8118 0.8155  0.8796    1    0
rf      0.6628  0.8063 0.8179 0.8484  0.8796    1    0
```



Accuracy
Kappa
rf
treebag
0.80
0.85
0.90
0.95
Accuracy
Kappa
Confidence Level: 0.95

Stacking

- 第1層
- Linear Discriminate Analysis (LDA)
- Classification and Regression Trees (CART)
- Logistic Regression (via Generalized Linear Model or GLM)
- k-Nearest Neighbors (kNN)
- Support Vector Machine with a Radial Basis Kernel Function (SVM)
- 第2層(第3層はないのでどちらかでよい)
- glm
- randomforest

caretEnsembleパッケージによるstacking

library(caretEnsemble)

Stacking algorithms例

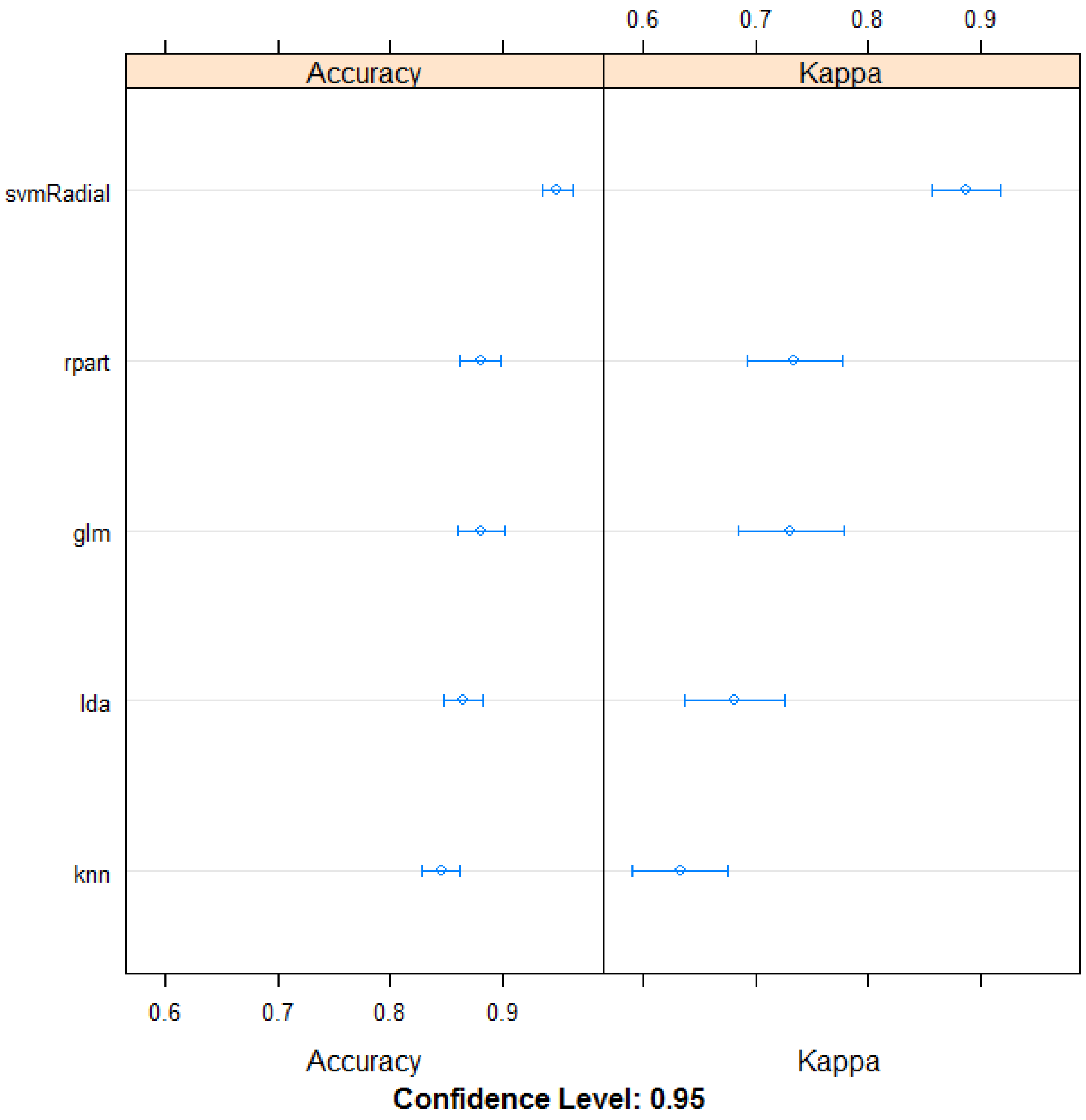
control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)

algorithmList <- c('lda', 'rpart', 'glm', 'knn', 'svmRadial')

```
set.seed(seed)
models <- caretList(Class~., data=dataset, trControl=control, methodList=algorithmList)
results <- resamples(models)
```

```
summary(results)
dotplot(results)
```

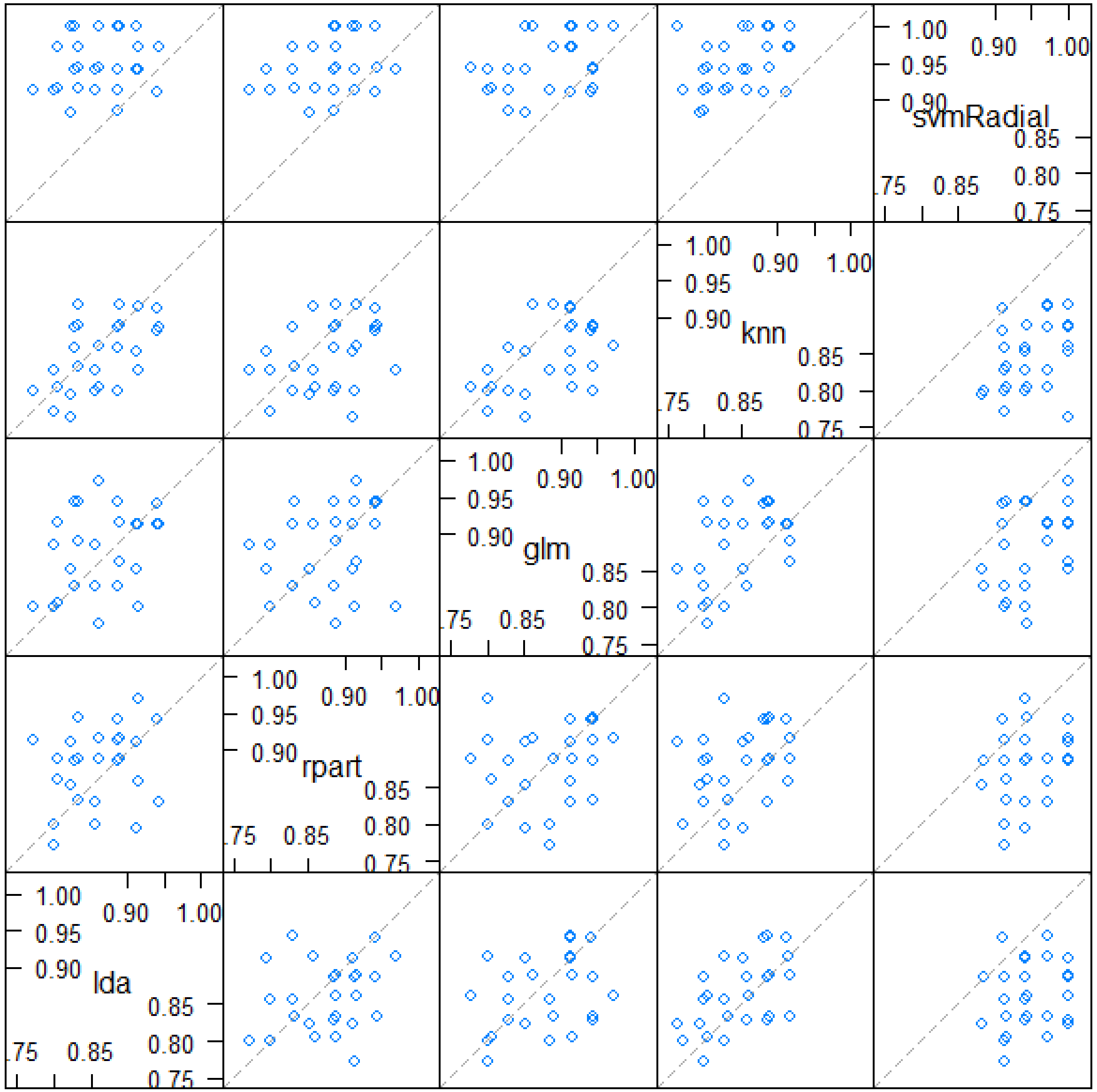
```
> summary(results)

Call:
summary.resamples(object = results)

Models: lda, rpart, glm, knn, svmRadial
Number of resamples: 30

Accuracy
            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
lda       0.7714  0.8286 0.8611 0.8645  0.9060 0.9429    0
rpart     0.7714  0.8540 0.8873 0.8803  0.9143 0.9714    0
glm       0.7778  0.8286 0.8873 0.8803  0.9167 0.9722    0
knn       0.7647  0.8056 0.8431 0.8451  0.8857 0.9167    0
svmRadial 0.8824  0.9143 0.9429 0.9486  0.9722 1.0000    0

Kappa
            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
lda       0.4595  0.5746 0.6810 0.6809  0.7852 0.8712    0
rpart     0.4776  0.6619 0.7490 0.7346  0.8127 0.9378    0
glm       0.5017  0.6144 0.7572 0.7316  0.8163 0.9388    0
knn       0.3929  0.5364 0.6353 0.6327  0.7330 0.8099    0
svmRadial 0.7244  0.8144 0.8778 0.8873  0.9405 1.0000    0
```


Accuracy
Kappa
svmRadial
rpart
glm
lda
knn
0.6
0.7
0.8
0.9
0.6
0.7
0.8
0.9
Accuracy
Kappa
Confidence Level: 0.95

# 結果の相関をみる
modelCor(results)
splom(results)

```
> dotplot(results)
> modelCor(results)
                lda     rpart       glm       knn svmRadial
lda       1.0000000 0.2515454 0.2970731 0.5013524 0.1790091
rpart     0.2515454 1.0000000 0.1749923 0.2823324 0.2938493
glm       0.2970731 0.1749923 1.0000000 0.5172239 0.4270294
knn       0.5013524 0.2823324 0.5172239 1.0000000 0.3973669
svmRadial 0.1790091 0.2938493 0.4270294 0.3973669 1.0000000
> splom(results)
```

Accuracy



Scatter Plot Matrix

```
# glmでstacking
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)
set.seed(seed)
stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=stackControl)
print(stack.glm)
```

```
> print(stack.glm)
A glm ensemble of 2 base models: lda, rpart, glm, knn, svmRadial

Ensemble results:
Generalized Linear Model

1053 samples
   5 predictor
   2 classes: 'bad', 'good'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 948, 947, 948, 947, 949, 948, ...
Resampling results:

  Accuracy   Kappa
  0.9534673  0.8989228
```

```r
# random forestでstacking
set.seed(seed)
stack.rf <- caretStack(models, method="rf", metric="Accuracy", trControl=stackControl)
print(stack.rf)
```

```
> print(stack.rf)
A rf ensemble of 2 base models: lda, rpart, glm, knn, svmRadial

Ensemble results:
Random Forest

1053 samples
   5 predictor
   2 classes: 'bad', 'good'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 948, 947, 948, 947, 949, 948, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  2     0.9623173  0.9178031
  3     0.9585226  0.9095254
  5     0.9563124  0.9047619

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 2.
```

出典

http://machinelearningmastery.com/machine-learning-ensembles-with-r/