USASK Data Science Bootcamp T7:Multi-Task Prediction Using Stacking Algorithms (MTPS)

TA: Kyle Gardiner Tutor: Xiaowen Cao

June 14, 2023. (11:00am-11:30am)

This document will outline how to utilize Multi-Task Prediction Using Stacking Algorithms (MTPS) in R to simultaneously predict multiple outcomes based on revised stacking algorithms, which allows the intergration of information from predictions of individual models.

First we will load in the necessary packages for this tutorial.

```
rm(list = ls(all = TRUE))  # removes objects from the environment to start fresh

# install.packages('MTPS')
library(MTPS)  # loads in the MTPS functions and contains the 'HIV' data used in this document
```

Data

The data used in this tutorial is from HIV Drug Resistance Database, which is included in the MTPS package. After loading the data, there will be 2 objects in the environment: YY and XX. YY contains the resistance of five Nucleoside RT Inhibitor (NRTI) drugs which represent the multivariate outcomes. The 5 NRTI drugs are: Lamivudine (3TC), Abacavir(ABC), Zidovudine (AZT), Stavudine (D4T), Didanosine (DDI). XX contains the mutation variables that will be used as predictors.

```
data(HIV)  # add the 'HIV' dataset to the environment

`?`(HIV  # prints dataframe information in 'help' console
)

head(XX[, 1:10], 10)  # displays the first 10 rows and columns
```

```
##    X.4S X.6D X.6E X.6K X.8I X.8V X.11K X.11R X.20K X.20R
## 1     0    0    0    0    0    0     0     0     0     0
## 3     0    0    0    0    0    0     0     0     0     0
## 4     0    0    0    0    0    0     0     1     0     0
## 6     0    0    0    0    0    0     0     0     0     0
## 7     0    0    0    0    0    0     0     0     0     0
## 8     0    0    0    0    0    0     0     0     0     0
## 9     0    0    0    0    0    0     0     0     0     0
## 10    0    0    0    0    0    0     1     1     0     0
## 11    0    0    0    0    0    0     0     0     0     1
## 12    0    0    0    0    0    0     0     0     0     0
```

```r
head(YY, 10)  # displays the first 10 rows of YY
```

```
##           ABC       3TC         AZT        D4T         DDI
## 1  0.63346846 2.3010300  0.59106461 0.14612804  0.07918125
## 3  0.04139269 0.1461280  1.44715803 0.00000000 -0.09691001
## 4  0.17609126 0.2552725  0.85125835 0.07918125  0.04139269
## 6  0.85125835 2.3010300 -0.09691001 0.11394335  0.27875360
## 7  0.71600334 0.4065402  0.82607480 0.73239376  0.72427587
## 8  0.77085201 2.3010300  0.27875360 0.11394335  0.23044892
## 9  0.79934055 2.3010300  0.71600334 0.14612804  0.27875360
## 10 0.75587486 2.3010300 -0.52287875 0.07918125  0.30103000
## 11 0.65321251 0.7923917  3.00000000 0.38021124  0.20411998
## 12 0.69019608 0.5185139  2.24797327 0.46239800  0.20411998
```

Model Fitting and Prediction

Revised Stacking Algorithm for Continuous Outcome

The HIV data set is used as the example. To illustrate how to fit the model and make predictions, we first split the HIV data into 2 parts, the training data and testing data.

```r
set.seed(12345)
xmat <- as.matrix(XX)
ymat <- as.matrix(YY)
nobs <- nrow(xmat)
training.id <- sample(seq_len(nobs), size = 0.8 * nobs)
y.train <- ymat[training.id, ]
y.test <- ymat[-training.id, ]
x.train <- xmat[training.id, ]
x.test <- xmat[-training.id, ]
```

The `r multiFit` function fits individual models for each outcome separately, referring the non-stacking algorithm. The following code fits generalized linear models with ridge regression regularization on each outcome.

```r
# no stacking
fit.mult <- multiFit(xmat = x.train, ymat = y.train, method = glmnet.ridge,
    family = "gaussian")
```

To set up the stacking algorithm, A list of algorithms for step one and step two need to be specified. A full list of the algorithms can be found using `list.learners()`.

For continuous outcome we need to specify `family = "gaussian"`. We also use `cv` and `residual` argument to specify whether we want to use Cross-Validation Stacking (CVS) or Residual Stacking (RS) or their combination. The default value for `cv` is "FALSE" and for `residual` is "TRUE", referring to the residual stacking algorithm. The following code fits models using Standard Stacking algorithm (SS), Cross-Validation Stacking (CVS), Residual Stacking (RS) and Cross-Validation Residual Stacking (CVRS) with `r MTPS` function.

In the example, generalized linear models with ridge regression regularization are used to fit models in the step one and tree models are applied in the step two for each outcome.

```
# Standard Stacking
fit.ss <- MTPS(xmat = x.train, ymat = y.train, family = "gaussian",
    cv = FALSE, residual = FALSE, method.step1 = glmnet.ridge,
    method.step2 = rpart1)
# Cross-Validation Stacking
fit.cv <- MTPS(xmat = x.train, ymat = y.train, family = "gaussian",
    cv = TRUE, residual = FALSE, method.step1 = glmnet.ridge,
    method.step2 = rpart1)
# Residual Stacking
fit.rs <- MTPS(xmat = x.train, ymat = y.train, family = "gaussian",
    cv = FALSE, residual = TRUE, method.step1 = glmnet.ridge,
    method.step2 = rpart1)
# Cross-Validation Residual Stacking
fit.cvrs <- MTPS(xmat = x.train, ymat = y.train, family = "gaussian",
    cv = TRUE, residual = TRUE, method.step1 = glmnet.ridge,
    method.step2 = rpart1)
```

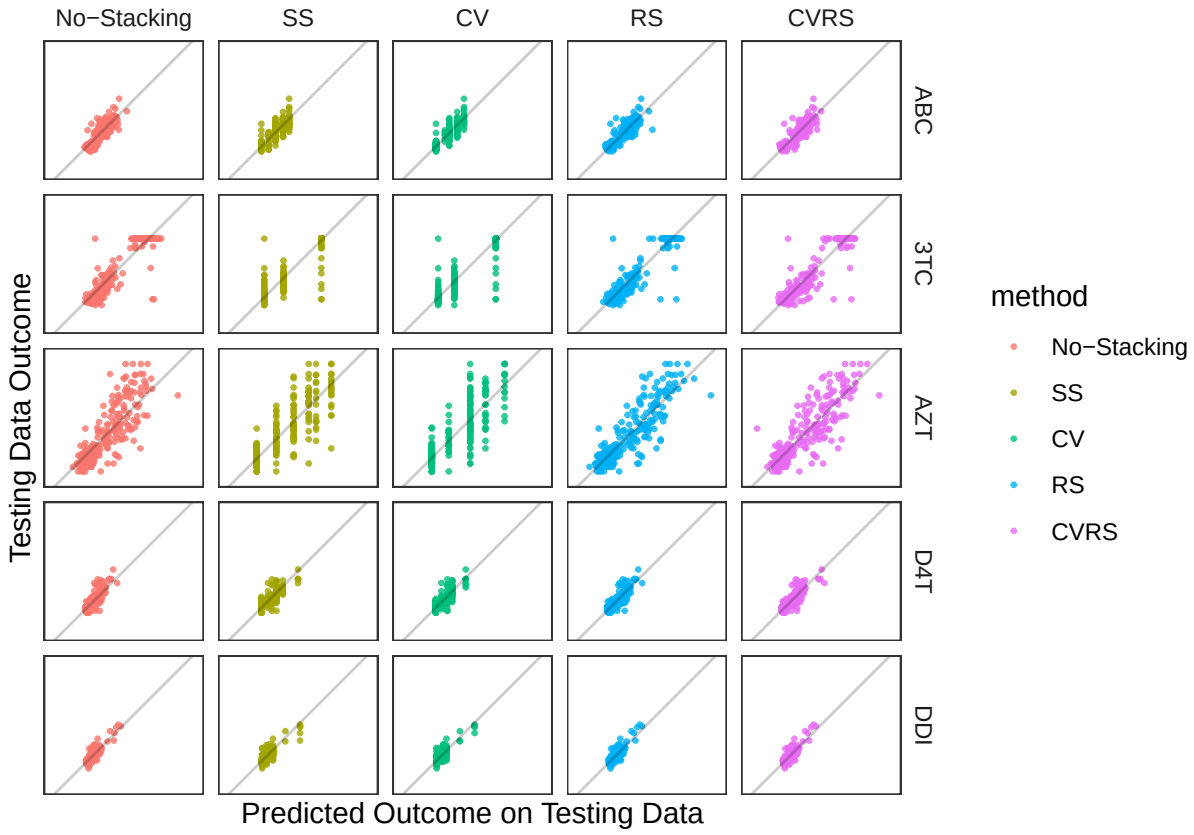
The `predict()` function returns the predicted value for the outcome on the testing data.

```
# no stacking
pred.mult <- predict(fit.mult, x.test)
# Standard Stacking
pred.ss <- predict(fit.ss, x.test)
# Cross-Validation Stacking
pred.cv <- predict(fit.cv, x.test)
# Residual Stacking
pred.rs <- predict(fit.rs, x.test)
# Cross-Validation Residual Stacking
pred.cvrs <- predict(fit.cvrs, x.test)
```

To visualize the above prediction, we plot the predicted value versus the actual outcomes of testing data for above algorithms.

```
library(ggplot2)
library(reshape2)
n.test <- nrow(x.test)
ctn.plot.data_matrix <- cbind(rbind(pred.mult, pred.ss, pred.cv,
    pred.rs, pred.cvrs), y.test[rep(seq_len(n.test), 5), ])
ctn.plot.data <- data.frame(rep(c("No-Stacking", "SS", "CV",
    "RS", "CVRS"), each = n.test), ctn.plot.data_matrix)
colnames(ctn.plot.data) <- c("method", paste0(rep("pred.", ncol(y.test)),
    colnames(y.test)), colnames(y.test))
dm1 <- melt(ctn.plot.data[, c("method", "ABC", "3TC", "AZT",
    "D4T", "DDI")], id = c("method"))
dm2 <- melt(ctn.plot.data[, c("method", "pred.ABC", "pred.3TC",
    "pred.AZT", "pred.D4T", "pred.DDI")], id = c("method"))
ctn.plot.data <- cbind(dm1, dm2[, -1])
colnames(ctn.plot.data) <- c("method", "Y", "yVal", "predict",
    "predictVal")
ctn.plot.data$method <- factor(ctn.plot.data$method, unique(as.character(ctn.plot.data$method)))
ctn.plot.data$yVal <- as.numeric(ctn.plot.data$yVal)
ctn.plot.data$predictVal <- as.numeric(ctn.plot.data$predictVal)
ggplot(ctn.plot.data) + geom_point(aes(x = predictVal, y = yVal,
```

```
    color = method), size = 0.5, alpha = 0.8) + geom_abline(slope = 1,
    alpha = 0.2) + coord_fixed() + ylab("Testing Data Outcome") +
    xlab("Predicted Outcome on Testing Data") + scale_x_discrete(breaks = NULL) +
    scale_y_discrete(breaks = NULL) + theme_bw() + theme(axis.text = element_blank(),
    strip.placement = "outside", strip.background = element_blank()) +
    facet_grid(Y ~ method)
```



Revised Stacking Algorithm for Binary Outcome

The MTPS package can fit models and make predictions when the outcome is binary. We use the HIV data to illustrate the usage. First we need convert the continuous outcome in the HIV data into binary outcome. The HIV database website recommended cutoff values to convert IC50 ratios to binary outcomes are 3TC = 3, ABC = 2, AZT = 3, D4T = 1.5, and DDI = 1.5. Then, using similar approach the data is split into training data and testing data.

```
# https://hivdb.stanford.edu/pages/published_analysis/genophenoPNAS2006/CUTOFFS/drug.cutoffs
# cutoff value to be used to define drug resistant
cutoffs <- c(2, 3, 3, 1.5, 1.5)
ymat.bin <- ymat
xmat.bin <- xmat
for (ii in 1:5) ymat.bin[, ii] <- (10^ymat[, ii] < cutoffs[ii])
y.train.bin <- ymat.bin[training.id, ]
y.test.bin <- ymat.bin[-training.id, ]
```

```
x.train.bin <- xmat.bin[training.id, ]
x.test.bin <- xmat.bin[-training.id, ]
```

For binary outcomes, the residual type can be `deviance`, `pearson` or `raw` for Deviance residual, Pearson residual or raw residual, respectively. The default value for the `residual` is "deviance". The `resid.std` argument determines whether or not to standardized the residuals and the default value is "FALSE".

Note that for Residual Stacking algorithm, the step 2 method must be able to fit continuous outcomes. Though we are making predictions on binary data, the step 2 in residual stacking is to make predictions on the residuals Therefore, we need use a method that can fit models for continuous outcomes.

The following code fit models using the Residual Stacking algorithms with standardized Pearson residual. The step one uses tree method and step two uses linear regression method.

```
fit.prs.std <- MTPS(xmat = x.train.bin, ymat = y.train.bin, family = "binomial",
    cv = FALSE, residual = TRUE, method.step1 = rpart1, method.step2 = lm1,
    resid.type = "pearson", resid.std = TRUE)
pred.prs.std <- predict(fit.prs.std, x.test.bin)
```

To see how the model performs we print the confusion matrices of each outcome on the testing data. 0.5 is used as the threshold for the prediction.

```
for (yy in 1:ncol(y.test.bin)) {
    print(colnames(y.test.bin)[yy])
    print(table((pred.prs.std[, yy] > 0.5) * 1, y.test.bin[,
        yy]))
}
```

```
## [1] "ABC"
##
##       0   1
##   0 139   5
##   1  10  96
## [1] "3TC"
##
##       0   1
##   0 131   8
##   1  10 101
## [1] "AZT"
##
##       0   1
##   0 110  15
##   1   2 123
## [1] "D4T"
##
##       0   1
##   0  99  17
##   1   9 125
## [1] "DDI"
##
##       0   1
##   0 114  26
##   1   6 104
```

Revised Stacking Algorithm for Mix Outcome

For illustration purpose, the first three columns of the outcome in the HIV data and last two columns of the binary outcome HIV data are combined for the mix outcome example.

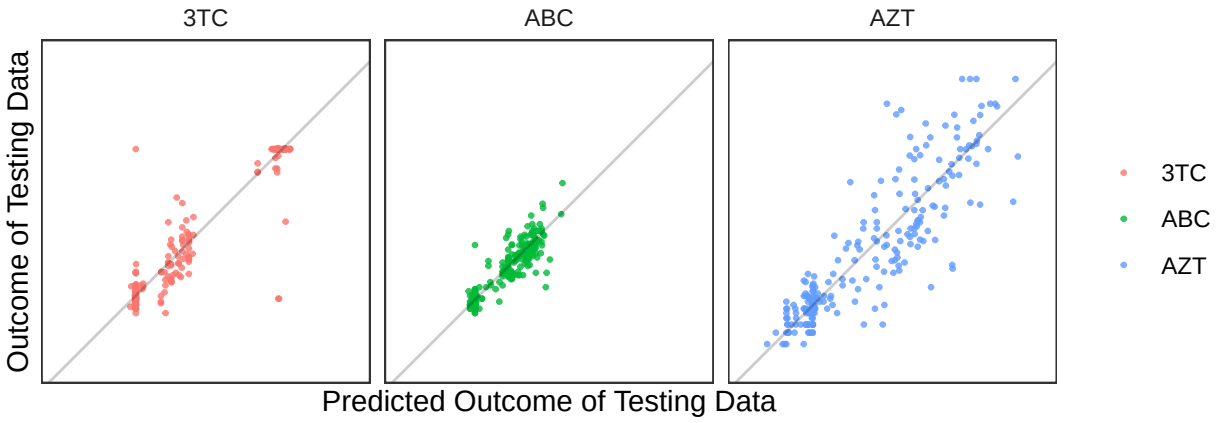
```
ymat.mix <- cbind(ymat[, 1:3], ymat.bin[, 4:5])
xmat.mix <- xmat
y.train.mix <- ymat.mix[training.id, ]
y.test.mix <- ymat.mix[-training.id, ]
x.train.mix <- xmat.mix[training.id, ]
x.test.mix <- xmat.mix[-training.id, ]
```

When the outcome variable is a combination of continuous and binary outcomes, it is necessary to specify each outcome type in the `r family` argument. For example, we fit the model using Residual Stacking algorithm. The first step uses LASSO and the second step uses tree method.

```
fit.mix.rs <- MTPS(xmat = x.train.mix, ymat = y.train.mix, family = c("gaussian",
    "gaussian", "gaussian", "binomial", "binomial"), cv = FALSE,
    residual = TRUE, method.step1 = glmnet.lasso, method.step2 = rpart1)
pred.mix.rs <- predict(fit.mix.rs, x.test.mix)
```

The following code draw the scatter plot of predicted value versus the actual outcomes on the first three columns of testing data, and the two confusion matrices for the last two columns of the testing data using 0.5 as the cutoff value.

```
n.test <- nrow(x.test)
mix.plot.data <- cbind(rep(colnames(y.test.mix)[1:3], each = nrow(y.test.mix)),
    rbind(cbind(pred.mix.rs[, 1], y.test.mix[, 1]), cbind(pred.mix.rs[,
        2], y.test.mix[, 2]), cbind(pred.mix.rs[, 3], y.test.mix[,
        3])))
colnames(mix.plot.data) <- c("Y", "predict", "outcome")
mix.plot.data <- as.data.frame(mix.plot.data)
mix.plot.data$predict <- as.numeric(as.character(mix.plot.data$predict))
mix.plot.data$outcome <- as.numeric(as.character(mix.plot.data$outcome))
ggplot(mix.plot.data) + geom_point(aes(x = predict, y = outcome,
    color = Y), size = 0.5, alpha = 0.8) + ylab("Outcome of Testing Data") +
    xlab("Predicted Outcome of Testing Data") + scale_x_discrete(breaks = NULL) +
    scale_y_discrete(breaks = NULL) + geom_abline(slope = 1,
    alpha = 0.2) + coord_fixed() + theme_bw() + theme(legend.title = element_blank(),
    axis.text = element_blank(), strip.placement = "outside",
    strip.background = element_blank()) + facet_grid(~Y)
```




```
for (yy in 4:5) {
    print(colnames(y.test.mix)[yy])
    print(table((pred.mix.rs[, yy] > 0.5) * 1, y.test.mix[, yy]))
}
```

```
## [1] "D4T"
##
##       0   1
##   0 100  18
##   1   8 124
## [1] "DDI"
##
##       0   1
##   0 113  20
##   1   7 110
```

This concludes our tutorial on how to implement MTPS in R to simultaneously predict multiple outcomes. We have shown that MTPS is capable of handling continuous, binary, and mixed outcomes. We have also shown that users can specify the type of stacking used: no stacking, residual stacking, cross-validation stacking, or cross-validation residual stacking. Moreover, MTPS has the flexibility to specify which base learners you wish to use for steps 1 and 2 of the stacking algorithm.