Example of how to use XGBoost optimized with SERA

Use case

In this markdown two use cases are provided where three variants are taken into consideration:

- XGBoost with SERA optimization where weights are obtained automatically;
- XGBoost with SERA optimization where weights are obtained with domain knowledge;
- XGBoost with Squared Error as objective.

The first use case uses a number of intervals $I = 1000$, while the second $I = 10$.

Let us start.

Load required packages

```
library(ModelOptimizationIR)
library(xgboost)
library(dplyr)
library(IRon)
library(scam)
```

Load data and perform random partition

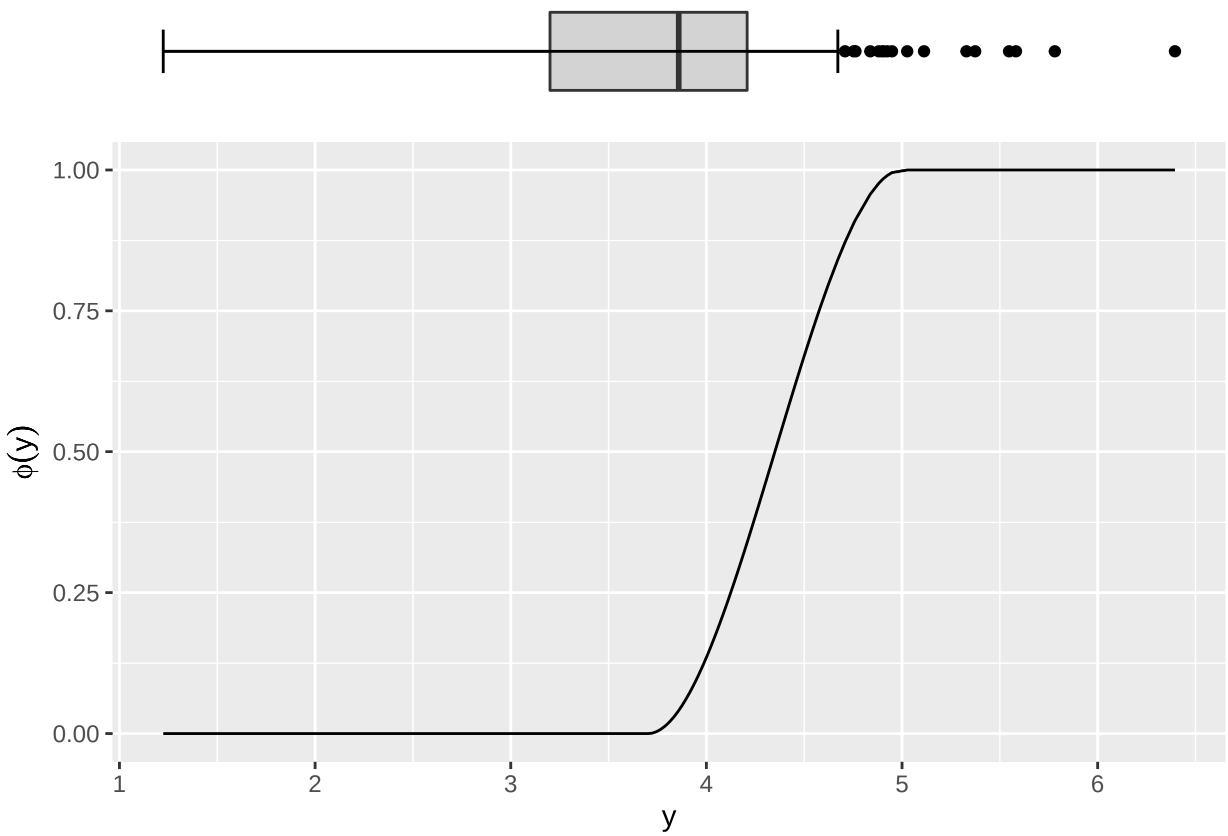
```
data("NO2Emissions")
n <- nrow(NO2Emissions)
s <- sample(1:n, size = n*0.8)

formula <- LNO2 ~ .
train <- NO2Emissions %>% dplyr::slice(s)
test <- NO2Emissions %>% dplyr::slice(-s)
```

Extract weights using domain knowledge

```
y <- train$LNO2
control.points <- matrix(c(1.1, 0, 0, 3.7, 0, 0, 5, 1, 0), byrow = TRUE, ncol=3) # Extreme points above

ph.ctrl <- phi.control(y = y, method="range", control.pts = control.points)
phi <- phi(y = y, phi.parms = ph.ctrl)
```

Plot with the relevance function

```
phiPlot(ds=y, phi.parms = ph.ctrl)
```



Add some hyper-parameters for modeling

```
params <- list(max_depth=7, eta=10^{-1}, gamma=10^{-2})
```

First use case $I = 1000$

Define steps to discretize SERA and calculate weights

```
I = 1000 # Default value
steps <- seq(0, 1, 1/I)
sigma <- sigma(phis = phi, steps = steps)
```

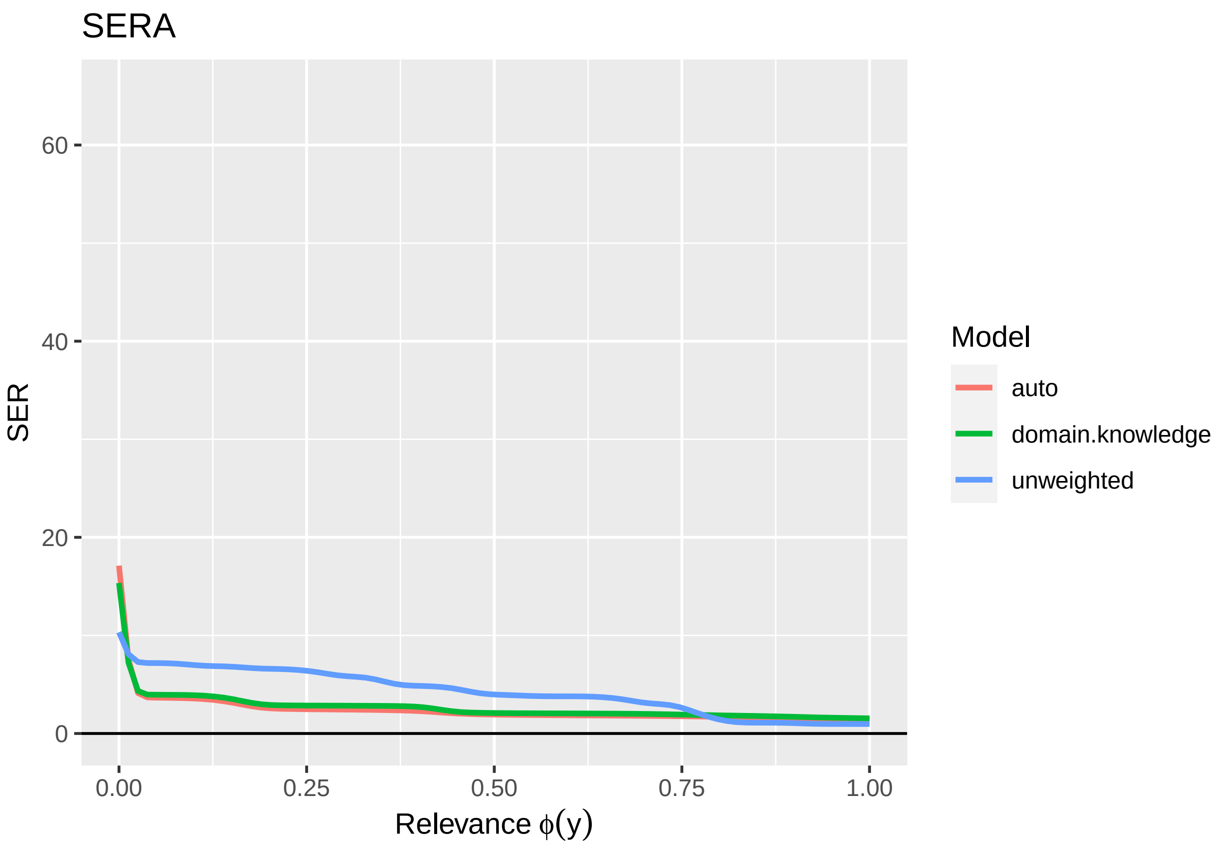
Train the models and save results

```
model_domain_knowledge <- XGBoost.sera(formula = formula, train = train, test = test, sigma = sigma, nro

model_auto <- XGBoost.sera(formula = formula, train = train, test = test, nrounds = 250, parameters = pa

model_unweighted <- wf.XGBoost(formula = formula, train = train, test = test, nrounds = 250, params = pa

df <- tibble(
        trues = model_domain_knowledge$trues,
        `auto` = model_auto$preds,
        `domain knowledge` = model_domain_knowledge$preds,
        `unweighted` = model_unweighted$preds
```

```
)
```

Evaluate SERA

```
sera(trues=df$trues, preds=dplyr::select(df, -trues), ph = ph.ctrl, pl=TRUE)
```




```
##            auto domain.knowledge       unweighted
##        2.331837         2.563095         4.234552
```

Evaluate MSE

```
df %>%
  summarise(across(!matches("trues"), ~mean((trues - .x)^2)))
```

```
## # A tibble: 1 x 3
##    auto `domain knowledge` unweighted
##   <dbl>              <dbl>      <dbl>
## 1 0.654              0.576      0.197
```

Second use case $I = 10$

Define steps to discretize SERA and calculate weights

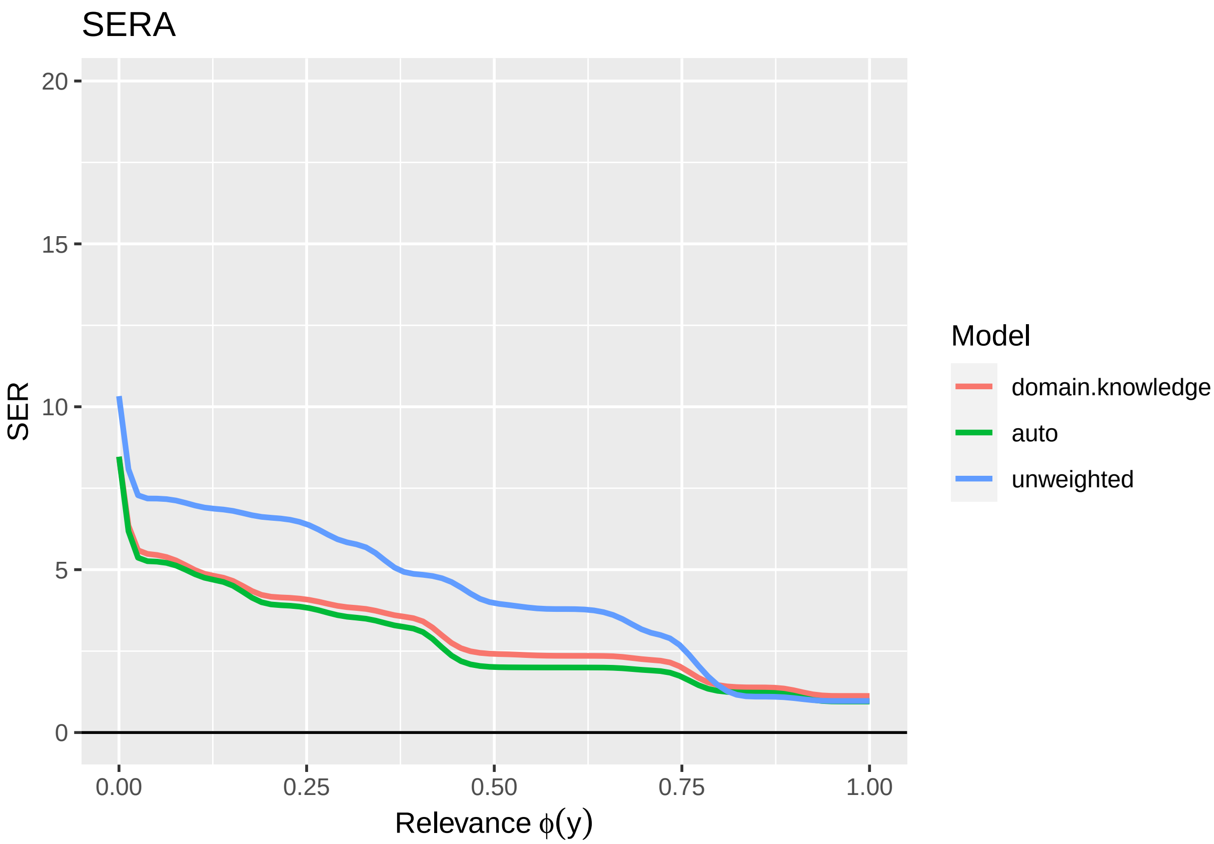
```
I = 10 # Default value
steps <- seq(0, 1, 1/I)
sigma <- sigma(phis = phi, steps = steps)
```

Train the models and save results

```
model_domain_knowledge <- XGBoost.sera(formula = formula, train = train, test = test, sigma = sigma, nro

model_auto <- XGBoost.sera(formula = formula, train = train, test = test, nrounds = 250, parameters = pa

model_unweighted <- wf.XGBoost(formula = formula, train = train, test = test, nrounds = 250, params = pa

df <- tibble(
        trues = model_domain_knowledge$trues,
        `domain knowledge` = model_domain_knowledge$preds,
        `auto` = model_auto$preds,
        `unweighted` = model_unweighted$preds
)
```

Evaluate SERA

```
sera(trues=df$trues, preds=dplyr::select(df, -trues), ph = ph.ctrl, pl=TRUE)
```




```
## domain.knowledge             auto       unweighted
##         2.989148         2.727815         4.234552
```

Evaluate MSE

```
df %>%
  summarise(across(!matches("trues"), ~mean((trues - .x)^2)))
```

```
## # A tibble: 1 x 3
##   `domain knowledge`  auto unweighted
##                <dbl> <dbl>      <dbl>
## 1              0.186 0.196      0.197
```

Better generalization was achieved, but at the cost of worst predictive focus on extreme values.