Stacking for Regression

Example: Boston Housing Prices

Predicting the median value of owner-occupied homes in neighborhoods around Boston, based on recorded characteristics of those neighborhoods.

```
library(readr)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(purrr)
library(glmnet)
library(caret)

# read in data
Boston <- read_csv("http://www.evanlray.com/data/mass/Boston.csv")

# Initial train/test split ("estimation"/test) and cross-validation folds
set.seed(63770)
tt_inds <- caret::createDataPartition(Boston$medv, p = 0.8)
train_set <- Boston %>% slice(tt_inds[[1]])
test_set <- Boston %>% slice(-tt_inds[[1]])

crossval_val_fold_inds <- caret::createFolds(
  y = train_set$medv, # response variable as a vector
  k = 10 # number of folds for cross-validation
)

get_complementary_inds <- function(x) {
  return(seq_len(nrow(train_set))[-x])
}
crossval_train_fold_inds <- map(crossval_val_fold_inds, get_complementary_inds)
```

Individual Methods

Linear Regression

```
lm_fit <- train(
  form = medv ~ .,
  data = train_set,
  method = "lm", # method for fit
  trControl = trainControl(method = "cv", # evaluate method performance via cross-validation
    number = 10, # number of folds for cross-validation
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all", # return information from cross-validation
    savePredictions = TRUE) # return validation set predictions from cross-validation
)

lm_fit$results
```

```
##   intercept     RMSE  Rsquared      MAE  RMSESD RsquaredSD    MAESD
## 1      TRUE 4.811094 0.7295385 3.442972 1.35204  0.1159406 0.737564
```

KNN

```r
knn_fit <- train(
  form = medv ~ .,
  data = train_set,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "cv",
    number = 10,
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all",
    savePredictions = TRUE),
  tuneGrid = data.frame(k = 1:20)
)

knn_fit$results
```

```
##     k     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1   1 4.302124 0.7734696 2.784063 1.937430  0.1690466 0.7366880
## 2   2 4.479520 0.7566818 2.905509 1.829320  0.1617163 0.7439624
## 3   3 4.437936 0.7718712 2.849816 1.809025  0.1449745 0.7471309
## 4   4 4.526359 0.7619782 2.857381 1.672480  0.1349338 0.7513697
## 5   5 4.738040 0.7418541 2.945441 1.610703  0.1388212 0.7713081
## 6   6 4.883579 0.7287600 3.027654 1.464551  0.1341844 0.6947846
## 7   7 4.894280 0.7293593 3.072536 1.475694  0.1345581 0.6800884
## 8   8 4.768478 0.7426477 3.049146 1.372125  0.1299861 0.5880398
## 9   9 4.737729 0.7470452 3.051570 1.326056  0.1197310 0.5619318
## 10 10 4.726088 0.7526682 3.062019 1.264614  0.1159238 0.5213294
## 11 11 4.691792 0.7584863 3.072702 1.272412  0.1190761 0.5278557
## 12 12 4.715812 0.7576409 3.100684 1.270655  0.1214070 0.5119859
## 13 13 4.716232 0.7574298 3.104043 1.261080  0.1229788 0.5029439
## 14 14 4.731370 0.7550899 3.113542 1.277869  0.1279896 0.5071200
## 15 15 4.742244 0.7550562 3.125198 1.310155  0.1310548 0.5289046
## 16 16 4.778555 0.7522025 3.154427 1.310425  0.1302173 0.5340460
## 17 17 4.825442 0.7501275 3.185001 1.332016  0.1314060 0.5393511
## 18 18 4.854797 0.7500211 3.228256 1.351186  0.1307268 0.5506677
## 19 19 4.902809 0.7465536 3.265701 1.351885  0.1282640 0.5326113
## 20 20 4.948886 0.7432701 3.287049 1.371115  0.1305545 0.5334139
```

Trees

```
rpart_fit <- train(
  form = medv ~ .,
  data = train_set,
  method = "rpart",
  trControl = trainControl(method = "cv",
    number = 10,
    index = crossval_train_fold_inds, # I'm specifying which folds to use, for consistency across methods
    indexOut = crossval_val_fold_inds, # I'm specifying which folds to use, for consistency across methods
    returnResamp = "all",
    savePredictions = TRUE),
  tuneGrid = data.frame(cp = seq(from = 0, to = 1, length = 20))
)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
rpart_fit$results
```

```
##            cp     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1  0.00000000 4.613606 0.7545507 3.116270 1.271684 0.10504781 0.5412488
## 2  0.05263158 5.444849 0.6600556 3.923264 1.202235 0.10425816 0.6688393
## 3  0.10526316 5.956504 0.5979661 4.439363 1.242146 0.11119482 0.7842615
## 4  0.15789474 5.956504 0.5979661 4.439363 1.242146 0.11119482 0.7842615
## 5  0.21052632 7.199021 0.4018550 5.422556 1.222995 0.08523368 0.7795902
## 6  0.26315789 7.199021 0.4018550 5.422556 1.222995 0.08523368 0.7795902
## 7  0.31578947 7.199021 0.4018550 5.422556 1.222995 0.08523368 0.7795902
## 8  0.36842105 7.199021 0.4018550 5.422556 1.222995 0.08523368 0.7795902
## 9  0.42105263 7.199021 0.4018550 5.422556 1.222995 0.08523368 0.7795902
## 10 0.47368421 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 11 0.52631579 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 12 0.57894737 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 13 0.63157895 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 14 0.68421053 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 15 0.73684211 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 16 0.78947368 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 17 0.84210526 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 18 0.89473684 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 19 0.94736842 9.150400       NaN 6.677802 1.191345         NA 0.6703375
## 20 1.00000000 9.150400       NaN 6.677802 1.191345         NA 0.6703375
```

Test set predictions from each of the 3 methods above:

```
lm_preds <- predict(lm_fit, newdata = test_set)
sqrt(mean((test_set$medv - lm_preds)^2))
```

```
## [1] 4.35759
```

```
knn_preds <- predict(knn_fit, newdata = test_set)
sqrt(mean((test_set$medv - knn_preds)^2))
```

```
## [1] 4.808452
```

```
rpart_preds <- predict(rpart_fit, newdata = test_set)
sqrt(mean((test_set$medv - rpart_preds)^2))
```

```
## [1] 3.608844
```

Ensemble Methods

Mean of Predictions from Stage 1 Methods

```
lm_preds <- predict(lm_fit, newdata = test_set)
knn_preds <- predict(knn_fit, newdata = test_set)
rpart_preds <- predict(rpart_fit, newdata = test_set)

mean_pred <- (lm_preds + knn_preds + rpart_preds) / 3

sqrt(mean((test_set$medv - mean_pred)^2))
```

```
## [1] 3.037933
```

Stacking: Fit a model to combine predictions from component models

Process:

Estimation:

1. Get cross-validated predictions for each "stage 1" or "component" model
2. Create a new data set where the explanatory variables are the cross-validated predictions from the component models
3. Fit a "stage 2" model to predict the response based on the component model predictions

Prediction for test set:

4. For each component model, re-fit to the full training data set and make predictions for the test set
5. Create a new data set where the explanatory variables are the test set predictions from the component models
6. Predict using the stage 2 model fit from step 3 and the data frame created in step 5.

Stacking via Linear Model, no intercept

```r
# Step 1: Validation-fold predictions from component models
lm_val_pred <- lm_fit$pred %>%
  arrange(rowIndex) %>%
  pull(pred)

knn_val_pred <- knn_fit$pred %>%
  filter(k == knn_fit$bestTune$k) %>%
  arrange(rowIndex) %>%
  pull(pred)

rpart_val_pred <- rpart_fit$pred %>%
  filter(cp == rpart_fit$bestTune$cp) %>%
  arrange(rowIndex) %>%
  pull(pred)

# Step 2: data set with validation-set component model predictions as explanatory variables
train_set <- train_set %>%
  mutate(
    lm_pred = lm_val_pred,
    knn_pred = knn_val_pred,
    rpart_pred = rpart_val_pred
  )

# Step 3: fit model using component model predictions as explanatory variables
# Here, a linear model without intercept (via lm directly because caret::train
# doesn't let you fit a model without intercept without more work).
stacking_fit <- lm(medv ~ 0 + lm_pred + knn_pred + rpart_pred, data = train_set)
coef(stacking_fit)
```

```
##    lm_pred   knn_pred rpart_pred
##  0.2484498  0.3998825  0.3618082
```

```r
# Step 4 (both cross-validation and refitting to the full training set were already done
# as part of obtaining lm_fit, knn_fit, and rpart_fit above)
lm_test_pred <- predict(lm_fit, newdata = test_set)
knn_test_pred <- predict(knn_fit, newdata = test_set)
rpart_test_pred <- predict(rpart_fit, newdata = test_set)

# Step 5: Assemble data frame of test set predictions from each component model
stacking_test_x <- data.frame(
  lm_pred = lm_test_pred,
  knn_pred = knn_test_pred,
  rpart_pred = rpart_test_pred
)

# Step 6: Stacked model predictions
stacking_preds <- predict(stacking_fit, stacking_test_x)

# Calculate error rate
sqrt(mean((test_set$medv - stacking_preds)^2))
```

```
## [1] 3.037996
```

Stacking via Ridge Regression

- We could also use other methods for the second stage model.

```r
# Step 1: Validation-fold predictions from component models
lm_val_pred <- lm_fit$pred %>%
  arrange(rowIndex) %>%
  pull(pred)

knn_val_pred <- knn_fit$pred %>%
  filter(k == knn_fit$bestTune$k) %>%
  arrange(rowIndex) %>%
  pull(pred)

rpart_val_pred <- rpart_fit$pred %>%
  filter(cp == rpart_fit$bestTune$cp) %>%
  arrange(rowIndex) %>%
  pull(pred)

# Step 2: data set with validation-set component model predictions as explanatory variables
train_set <- train_set %>%
  mutate(
    lm_pred = lm_val_pred,
    knn_pred = knn_val_pred,
    rpart_pred = rpart_val_pred
  )

# Step 3: fit model using component model predictions as explanatory variables
stacking_fit <- train(
  form = medv ~ lm_pred + knn_pred + rpart_pred,
  data = train_set,
  method = "glmnet",
  tuneLength = 10)
coef(stacking_fit$finalModel, stacking_fit$bestTune$lambda) %>% t()
```

```
## 1 x 4 sparse Matrix of class "dgCMatrix"
##    (Intercept)   lm_pred  knn_pred rpart_pred
## 1    0.6657554 0.2909121 0.3519403  0.3339393
```

```r
# Step 4 (both cross-validation and refitting to the full training set were already done
# as part of obtaining lm_fit, knn_fit, and rpart_fit above)
lm_test_pred <- predict(lm_fit, newdata = test_set)
knn_test_pred <- predict(knn_fit, newdata = test_set)
rpart_test_pred <- predict(rpart_fit, newdata = test_set)

# Step 5: Assemble data frame of test set predictions from each component model
stacking_test_x <- data.frame(
  lm_pred = lm_test_pred,
  knn_pred = knn_test_pred,
  rpart_pred = rpart_test_pred
)

# Step 6: Stacked model predictions
stacking_preds <- predict(stacking_fit, stacking_test_x)

# Calculate error rate
sqrt(mean((test_set$medv - stacking_preds)^2))
```

```
## [1] 3.045026
```