Stacking for Regression

Introduction

Ensembles for Classification

Last class, we introduced stacking in a classification context.

- We have multiple "stage 1" or "component" classifiers
- We combine predictions from these stage 1 classifiers
  - Majority vote
  - Average predicted class probabilities
  - Stacking: feed predicted class probabilities from stage 1 models into a new stage 2 model. Lets us weight models according to (cross-validated) performance on training set.
- If component classifiers generate predictions that are not highly correlated, ensembles can often improve overall classification error rates.
  - Example where 3 independent classifiers have error rate 0.3, majority vote has error rate 0.216
  - Benefit from building ensemble disappears if component model predictions are identical/highly correlated

Ensembles for Regression

$\hat{Y}_i$ is a number.

Consider an ensemble that takes the average of predictions $\widehat{Y}_i^{(1)}$, $\widehat{Y}_i^{(2)}$, $\widehat{Y}_i^{(3)}$ from 3 stage 1 models:

$\widehat{Y}_i^{(ensemble)} = \frac{1}{3}\widehat{Y}_i^{(1)} + \frac{1}{3}\widehat{Y}_i^{(2)} + \frac{1}{3}\widehat{Y}_i^{(3)}$

Toy Scenario: Component models are independent, bias 0, same variance

Suppose each of our component regression models generates predictions with the following characteristics (stated in terms of the first model):

- On average, $\widehat{Y}_i^{(1)} = Y_i$ (so predictions have bias 0).

$$E\left[\widehat{Y}_i^{(1)}\right] = Y_i$$

- Variance of predictions is σ^2 (same variance for all models).

$$Var\left[\widehat{Y}_i^{(1)}\right] = \sigma^2$$

- So each component model has expected test set MSE = $\sigma^2 + \text{Var}(\varepsilon)$

$$\begin{aligned}\text{MSE} &= \text{Bias}(\widehat{Y}_i^{(1)})^2 + \text{Var}(\widehat{Y}_i^{(1)}) + \text{Var}(\varepsilon) \\ &= 0^2 + \sigma^2 + \text{Var}(\varepsilon)\end{aligned}$$

Then...

- On average, $\widehat{Y}_i^{(ensemble)} = Y_i$ (ensemble predictions also have bias 0).

$$
\begin{aligned}
E\left[\widehat{Y}_i^{(ensemble)}\right] &= E\left[\frac{1}{3}\widehat{Y}_i^{(1)} + \frac{1}{3}\widehat{Y}_i^{(2)} + \frac{1}{3}\widehat{Y}_i^{(3)}\right] \\
&= \frac{1}{3}E\left[\widehat{Y}_i^{(1)}\right] + \frac{1}{3}E\left[\widehat{Y}_i^{(2)}\right] + \frac{1}{3}E\left[\widehat{Y}_i^{(3)}\right] \\
&= \frac{1}{3}Y_i + \frac{1}{3}Y_i + \frac{1}{3}Y_i \\
&= Y_i
\end{aligned}
$$

- Ensemble predictions have variance $\frac{1}{3}\sigma^2$

$$
\begin{aligned}
\mathrm{Var}(\widehat{Y}_i^{(ensemble)}) &= \mathrm{Var}\left(\frac{1}{3}\widehat{Y}_i^{(1)} + \frac{1}{3}\widehat{Y}_i^{(2)} + \frac{1}{3}\widehat{Y}_i^{(3)}\right) \\
&= \frac{1}{9}\mathrm{Var}(\widehat{Y}_i^{(1)}) + \frac{1}{9}\mathrm{Var}(\widehat{Y}_i^{(2)}) + \frac{1}{9}\mathrm{Var}(\widehat{Y}_i^{(3)}) \\
&= \frac{1}{9}\sigma^2 + \frac{1}{9}\sigma^2 + \frac{1}{9}\sigma^2 \\
&= \frac{1}{3}\sigma^2
\end{aligned}
$$

- So the ensemble model has expected test set MSE $= \frac{1}{3}\sigma^2 + \mathrm{Var}(\varepsilon)$

$$
\begin{aligned}
\mathrm{MSE} &= \mathrm{Bias}(\widehat{Y}_i^{(ensemble)})^2 + \mathrm{Var}(\widehat{Y}_i^{(ensemble)}) + \mathrm{Var}(\varepsilon) \\
&= 0^2 + \frac{1}{3}\sigma^2 + \mathrm{Var}(\varepsilon)
\end{aligned}
$$

Comments:

- Combining predictions from independent (or not-too-highly-correlated) methods reduces variance, and so overall expected test set MSE
- If the methods are highly correlated, less beneficial to combine them
  - Extreme case: correlation 1, ensemble predictions are same as component model predictions
- Combining predictions could also help bias if the methods are biased in different directions (some predict too high, some too low)
  - in general this effect will be small
- There's nothing we can do about $\mathrm{Var}(\varepsilon)$ in a regression problem, just like we can never improve beyond the Bayes error rate in a classification problem.

Example of Ensembles for Regression: 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)


# Function to calculate error rate
calc_rmse <- function(observed, predicted) {
  sqrt(mean((observed - predicted)^2))
}
```

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 5.025714 0.7185674 3.521468 0.6805193 0.06412781 0.4152867
```

KNN

```
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 4.931673 0.7390619 3.000359 1.152091 0.11258010 0.5131336
## 2   2 4.408710 0.7799870 2.849250 1.267178 0.10906674 0.5410545
## 3   3 4.347795 0.7962749 2.881047 1.137002 0.08980069 0.5372020
## 4   4 4.663994 0.7660576 2.982270 1.110708 0.09381608 0.5309543
## 5   5 4.800501 0.7542518 3.032455 1.125649 0.09658700 0.5279068
## 6   6 4.791768 0.7564159 3.047812 1.091507 0.09228818 0.4396901
## 7   7 4.748684 0.7625673 3.033776 1.142471 0.09799070 0.5000469
## 8   8 4.707296 0.7699082 3.014360 1.095811 0.09535442 0.4463654
## 9   9 4.680848 0.7754724 3.027455 1.107982 0.09883212 0.4571527
## 10 10 4.770322 0.7722160 3.076885 1.068084 0.09434895 0.4258845
## 11 11 4.824141 0.7696020 3.089801 1.036094 0.09075463 0.4230247
## 12 12 4.882281 0.7646528 3.120073 1.085633 0.09497085 0.4393558
## 13 13 4.915189 0.7607095 3.140820 1.081353 0.09747934 0.4523821
## 14 14 4.930706 0.7621762 3.174795 1.139793 0.10081988 0.5381322
## 15 15 4.977595 0.7594625 3.232771 1.138999 0.10429540 0.5505091
## 16 16 4.972870 0.7643463 3.227529 1.140101 0.10686991 0.5548220
## 17 17 5.006084 0.7651308 3.244611 1.119445 0.10416654 0.5439144
## 18 18 5.065394 0.7578510 3.266501 1.120979 0.10678777 0.5532262
## 19 19 5.106508 0.7546957 3.269572 1.098079 0.10612742 0.5477386
## 20 20 5.163214 0.7481105 3.319266 1.112726 0.10875469 0.5807145
```

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),
  tuneLength = 10
)
```

```
## 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  0.007668559 4.774046 0.7403614 3.241857 0.9290802 0.10303863 0.5303755
## 2  0.007721165 4.776139 0.7402864 3.249523 0.9296645 0.10303935 0.5313649
## 3  0.008398386 4.816355 0.7366519 3.301891 0.8837101 0.09958789 0.4894321
## 4  0.017684397 5.086834 0.7093609 3.447708 1.0495644 0.10353241 0.6188988
## 5  0.024924681 5.263665 0.6913274 3.627819 0.9203274 0.09647193 0.5290841
## 6  0.033259667 5.495427 0.6613844 3.822806 0.9435124 0.11306652 0.4482429
## 7  0.037240395 5.520659 0.6596508 3.868258 0.9619152 0.11285111 0.4852382
## 8  0.083128169 6.325936 0.5655603 4.640580 0.9048618 0.12519307 0.5852019
## 9  0.155311532 6.835044 0.4844858 5.065129 1.0353002 0.16921796 0.5865178
## 10 0.442311909 8.502411 0.3062601 6.091474 1.1919389 0.07822153 0.8560560
```

Test set predictions from each of the 3 methods above:

```r
lm_preds <- predict(lm_fit, newdata = test_set)
calc_rmse(test_set$medv, lm_preds)
```

```
## [1] 4.086116
```

```r
knn_preds <- predict(knn_fit, newdata = test_set)
calc_rmse(test_set$medv, knn_preds)
```

```
## [1] 3.098053
```

```r
rpart_preds <- predict(rpart_fit, newdata = test_set)
calc_rmse(test_set$medv, rpart_preds)
```

```
## [1] 3.488305
```

Ensemble Methods

Mean of Predictions from Stage 1 Methods

```r
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

calc_rmse(test_set$medv, mean_pred)
```

```
## [1] 2.799591
```

Stacking: Fit a model to combine predicted class membership probabilities

- Some methods might be better than others; we should give them more weight.
- We can use training set performance to determine how much to weight them.
- We must cross-validate: otherwise, we'd give too much weight to models that overfit the training data

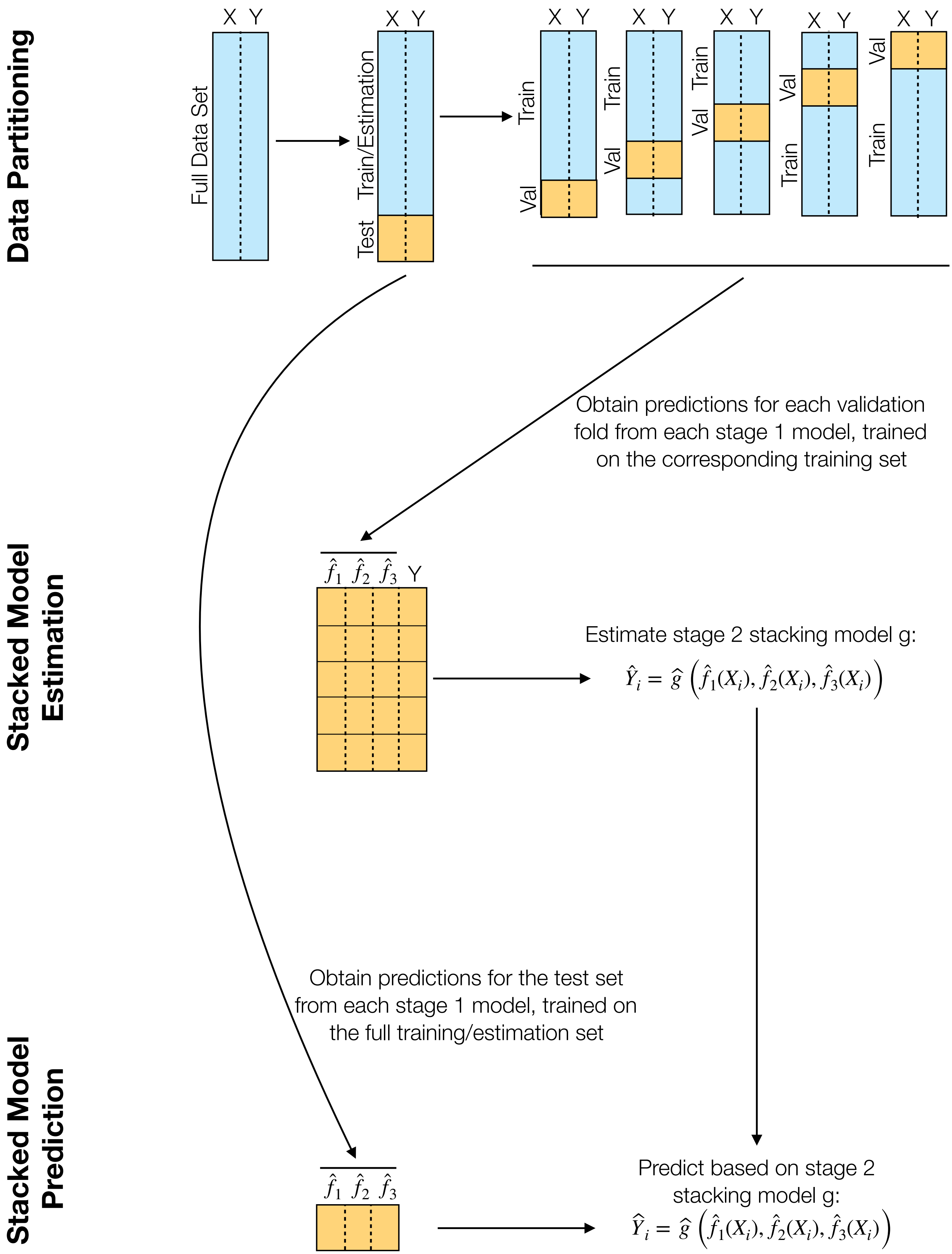
Process:

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.


Data Partitioning
Stacked Model Estimation
Stacked Model Prediction
X Y
Full Data Set
Train/Estimation
Test
Train
Val
Obtain predictions for each validation fold from each stage 1 model, trained on the corresponding training set
$\hat{f}_1$ $\hat{f}_2$ $\hat{f}_3$ Y
Estimate stage 2 stacking model g:
$\hat{Y}_i = \hat{g}\left(\hat{f}_1(X_i), \hat{f}_2(X_i), \hat{f}_3(X_i)\right)$
Obtain predictions for the test set from each stage 1 model, trained on the full training/estimation set
$\hat{f}_1$ $\hat{f}_2$ $\hat{f}_3$
Predict based on stage 2 stacking model g:
$\hat{Y}_i = \hat{g}\left(\hat{f}_1(X_i), \hat{f}_2(X_i), \hat{f}_3(X_i)\right)$

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
##  0.1837451  0.5104439  0.3309975
```

```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
calc_rmse(test_set$medv, stacking_preds)
```

```
## [1] 2.713425
```

Stacking via Ridge Regression

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

```
# 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   -1.288193 0.2511953 0.4850011  0.3361018
```

```
# 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
calc_rmse(test_set$medv, stacking_preds)
```

```
## [1] 2.691994
```