Final Project

AUTHOR
Taqee Hussain

Setup

```
# Load libraries
library(tidyverse)
library(glmnet)
library(rpart)
library(randomForest)
library(xgboost)
```

Load Data

```
training <- read_csv("training_data.csv")
test_pred <- read_csv("test_predictors.csv")
# Take a look
glimpse(training)
summary(training)
```

Baseline Linear Regression

```
set.seed(1)
folds <- sample(rep(1:10, length.out = nrow(training)))
get_mspe <- function(actual, predicted) {
  mean((actual - predicted)^2)
}
ls_cv_errors <- c()
for(k in 1:10) {
  train_fold <- training[folds != k, ]
  valid_fold <- training[folds == k, ]
  fit_ls <- lm(Y ~ ., data = train_fold)
  preds <- predict(fit_ls, newdata = valid_fold)
  ls_cv_errors[k] <- get_mspe(valid_fold$Y, preds)
}
mean(ls_cv_errors)
```

The baseline linear regression model has a CV-MSPE of 26.90. This gives us a reference point: any more flexible or regularized model should ideally achieve a lower MSPE to justify its added complexity.

Ridge Regression

```
X <- model.matrix(Y ~ ., data = training)[, -1]
y <- training$Y
library(glmnet)
ridge_cv_errors <- c()
for(k in 1:10) {
  X_train <- X[folds != k, ]
  y_train <- y[folds != k]
  X_valid <- X[folds == k, ]
  y_valid <- y[folds == k]
  # Fit ridge: alpha = 0
  fit_ridge <- cv.glmnet(
    X_train, y_train,
    alpha = 0,
    nfolds = 10,
    type.measure = "mse"
  )

  # Use lambda.min for prediction
  preds <- predict(fit_ridge, newx = X_valid, s = "lambda.min")
  ridge_cv_errors[k] <- get_mspe(y_valid, preds)
}
mean(ridge_cv_errors)
```

Lasso Regression

```
lasso_cv_errors <- c()
for(k in 1:10) {
  X_train <- X[folds != k, ]
  y_train <- y[folds != k]
  X_valid <- X[folds == k, ]
  y_valid <- y[folds == k]
  # Fit LASSO: alpha = 1
  fit_lasso <- cv.glmnet(
    X_train, y_train,
    alpha = 1,
    nfolds = 10,
    type.measure = "mse"
  )
  preds <- predict(fit_lasso, newx = X_valid, s = "lambda.min")
  lasso_cv_errors[k] <- get_mspe(y_valid, preds)
}
mean(lasso_cv_errors)
```

Ridge (MSPE = 27.29) and LASSO (MSPE = 26.92) performed slightly worse than the least squares baseline (26.90). With a large sample size and only 19 predictors, regularization added bias without reducing variance, and shrinking coefficients removed useful signal.

Regression Tree

```
library(rpart)
tree_cv_errors <- c()
for(k in 1:10) {
  train_fold <- training[folds != k, ]
  valid_fold <- training[folds == k, ]
  # grow a large tree (no pruning)
  full_tree <- rpart(
    Y ~ .,
    data = train_fold,
    control = rpart.control(cp = 0)
  )
  # choose cp via minimum cross-validated error
  best_cp <- full_tree$cptable[ which.min(full_tree$cptable[,"xerror"]) , "CP" ]
  pruned_tree <- prune(full_tree, cp = best_cp)
  preds <- predict(pruned_tree, newdata = valid_fold)
  tree_cv_errors[k] <- get_mspe(valid_fold$Y, preds)
}
mean(tree_cv_errors)
```

The regression tree reduced MSPE from 26.9 to 22.4, revealing clear nonlinear structure that linear models could not capture. However, single trees remain high-variance learners, so further improvement from ensemble methods is expected.

Boosting (XGBoost)

```
library(xgboost)
boost_cv_errors <- c()
for(k in 1:10) {
  X_train <- X[folds != k, ]
  y_train <- y[folds != k]
  X_valid <- X[folds == k, ]
  y_valid <- y[folds == k]
  # convert to xgb matrix
  dtrain <- xgb.DMatrix(data = X_train, label = y_train)
  dvalid <- xgb.DMatrix(data = X_valid, label = y_valid)
  params <- list(
    objective = "reg:squarederror",
    eta = 0.05,
    max_depth = 6,
    subsample = 0.8,
    colsample_bytree = 0.8
  )
  # fit model
  fit_boost <- xgb.train(
    params = params,
    data = dtrain,
    nrounds = 300,
    verbose = 0
```

```
  )
  preds <- predict(fit_boost, newdata = dvalid)
  boost_cv_errors[k] <- get_mspe(y_valid, preds)
}
mean(boost_cv_errors)
```

MSPE = **17.7**, competitive but not better than Random Forest.

RF Tuning

```
mtry_values <- c(2, 4, 6, 10, 15, 19)
rf_results <- data.frame(mtry = mtry_values, oob_mse = NA)
set.seed(1)
for(i in seq_along(mtry_values)) {
  fit <- randomForest(
    Y ~ .,
    data = training,
    ntree = 1000,
    mtry = mtry_values[i],
    nodesize = 5
  )
  rf_results$oob_mse[i] <- fit$mse[1000]
}
rf_results
```

Tuning mtry showed that performance improved steadily with larger values, reaching the best OOB MSPE of 15.82 at mtry = 15. This indicates that allowing more predictors to be considered at each split helped the forest capture the underlying signal.

FINAL MODEL

```
final_rf <- randomForest(
  Y ~ .,
  data = training,
  ntree = 1000,
  mtry = 15,
  nodesize = 5
)
final_predictions <- predict(final_rf, newdata = test_pred)
```

The tuned Random Forest (mtry = 15) achieved the lowest MSPE of 15.82, outperforming all linear, regularized, and boosting models. This model effectively captured complex nonlinear relationships and was selected for generating final predictions.

How Many "True" Predictors Are There?

Because LASSO did not drop predictors and tree-based models used many variables, most

predictors appear to contain signal. A reasonable estimate is that roughly 15–19 predictors influence Y to some degree.

Appendix

Question I asked ChatGPT:

Q1. Why did Ridge and LASSO not improve over the least squares model?

A: The dataset has n = 20,000 observations and only p = 19 predictors. In this setting, the least squares estimator is stable and low-variance. Ridge adds L2 shrinkage and LASSO removes coefficients, but because most predictors contain useful signal, both methods introduced unnecessary bias without reducing variance, resulting in MSPEs that were slightly worse than least squares.

Q2. Why does Random Forest perform so much better than a single tree

A:

A single tree is high-variance because small changes in the data produce very different splits. Random Forest combats this by averaging many decorrelated trees, dramatically reducing variance. This lowered the MSPE from 22.4 (tree) to 16.9 (forest), demonstrating how effective variance reduction is for this dataset.

Q3. What does the mtry parameter control in a Random Forest?

A: mtry determines how many predictors the algorithm considers at each split.

Smaller mtry introduces more randomness between trees (lower variance, higher bias), while larger mtry reduces randomness (lower bias, possibly higher correlation). Tuning mtry between {2,4,6,10,15,19} showed that increasing mtry steadily improved predictive accuracy, with the best performance at mtry = 15.