Homework 1 Key

# Conceptual Problems

## Problem 1

Consider the supervised learning setup. Recall, we have observations and a model given by

We’ll consider doing a **regression task**. Let’s assume that the errors (’s) are independent and identically distributed, following a Normal distribution with mean zero and variance . Mathematically, we write this as

We’ll assume that the predictors, , are fixed, known values.

Suppose we apply the KNN model with a pre-specified value of and a fixed point of interest. We have an interest in the properties of from this model.

1. Compute the variance of (we wrote down a form for this quantity in the week 2 notes).

Hint: Recall from previous coursework - If we have independent observations (call them ) and are interested in the variability of a mean (call it ), we know

In the second set of notes we wrote the solution for as

As this is an average of iid terms, we can apply a similar result to that of ,

1. Using your answer from (a), what happens to the variance of our estimate as we increase ? What happens as we decrease ? Explain why this makes sense.

As we increase our variance decreases since is in the denominator. As we decrease , the variance likewise increases. This makes sense because we should have less variability if we include more ‘neighbors’ and more variablity for including fewer.

1. Suppose we run two separate KNN regressions on the same data set. One model uses and one model uses . For each model we compute the training MSE and the test MSE on an independent test set.
   1. Which model () will have lower training MSE? Explain.

* The model that includes fewer neighbors should have a lower training MSE. This is because the model is more flexible and able to train closely to the data in the training set. The more neighbors we include, the less flexible the model becomes as we borrow across more observations.
  1. Do we know which one will have lower test MSE? Explain.
* We don’t know which will have lower test MSE! This is problem dependent. Usually, there is some optimal that we can estimate using CV, bootstrap resampling, or a validation set.

## Book Problems

Complete the following problems from the Introduction to Statistical Learning with R book (I’m not sure if the problems are in the same order in the python book so use the R book to identify which problems to do).

Section 2.4

* Book Problem 1
  1. Flexible since a lot of observations with few predictors will allow us to estimate most functions well
  2. Inflexible as there likely aren’t enough observations to estimate a complicated function
  3. Flexible to obtain a more reasonable fit to the complicated structure
  4. Inflexible as when we have higher variance this implies we need more observations to see the signal and learn with our model.
* Book Problem 2
  1. Inference as we want to understand the relationship. A regression task with and
  2. Prediction as we aren’t looking at relationships but mainly if we have success or failure. A classification task with and
  3. Prediction as we want to get the percent change. A regression task with and
* Book Problem 5

Flexible approaches can better fit more complicated structures and nonlinear type relationships. Less flexible methods are better when you have less data generally.

* Book Problem 7. Note: For the KNN classification model, the predicted classification is given by the class the occurs the most for the K neighbors.
  + - For observations 1, the distance is
    - For observations 2, the distance is
    - For observations 3, the distance is
    - For observations 4, the distance is
    - For observations 5, the distance is
    - For observations 6, the distance is
  1. When , we use the closest point for prediction. This is observation 5. Therefore, we would predict Green.
  2. When , we would use the three closest points for prediction. These are observations 2, 5, and 6. We have two red and one green in these three so we would predict Red.
  3. The optimal value for would likely be small since we would need our decision boundary to be more variable.

# Implementation Problems

## Problem 2

Consider the Boston housing data from the ISLR2 package (a similar package exists in python). Suppose we want to build a prediction model for the medv variable. We did this in our notes using the lstat variable as our predictor.

1. Split the data into a training and test set using a 70/30 split. Using SRSWOR.

library(ISLR2)  
library(caret)  
library(tidyverse)  
Boston <- as\_tibble(Boston) #not needed, but I like how tibbles print!  
#Use the caret package to split the data  
set.seed(50)  
index <- createDataPartition(Boston$medv,  
 p = 0.7,  
 list = FALSE,  
 times = 1)  
  
train <- Boston[index, ]  
test <- Boston[-index, ]

1. Consider three separate simple linear regression models: one using age as the predictor, one using rm as the predictor, and one using ptratio as the predictor.
   * Compare these models on the training data set only using 10 fold CV and RMSE as your metric. Note: CV isn’t needed to tune a hyperparameter here but we can still use it to choose between our three candidate SLR models!
   * With your best model, fit it using the entire training data set.

tc <- trainControl(method = "cv",  
 number = 10)  
mod1 <- train(medv ~ age,   
 data = train,  
 method = "lm",  
 trControl = tc)  
mod2 <- train(medv ~ rm,   
 data = train,  
 method = "lm",  
 trControl = tc)  
mod3 <- train(medv ~ ptratio,   
 data = train,  
 method = "lm",  
 trControl = tc)  
mod1

Linear Regression   
  
356 samples  
 1 predictor  
  
No pre-processing  
Resampling: Cross-Validated (10 fold)   
Summary of sample sizes: 320, 320, 320, 320, 323, 321, ...   
Resampling results:  
  
 RMSE Rsquared MAE   
 8.735775 0.1442729 6.173765  
  
Tuning parameter 'intercept' was held constant at a value of TRUE

mod2

Linear Regression   
  
356 samples  
 1 predictor  
  
No pre-processing  
Resampling: Cross-Validated (10 fold)   
Summary of sample sizes: 321, 320, 321, 320, 319, 322, ...   
Resampling results:  
  
 RMSE Rsquared MAE   
 6.813814 0.4874654 4.647741  
  
Tuning parameter 'intercept' was held constant at a value of TRUE

mod3

Linear Regression   
  
356 samples  
 1 predictor  
  
No pre-processing  
Resampling: Cross-Validated (10 fold)   
Summary of sample sizes: 321, 320, 320, 320, 320, 321, ...   
Resampling results:  
  
 RMSE Rsquared MAE   
 8.019405 0.269002 5.846958  
  
Tuning parameter 'intercept' was held constant at a value of TRUE

Model 2 with rm as the predictor has the lowest CV error on the training set. This is actually contained in the mod2$finalModel element but we can refit that model to the entire data set manually if we want.

mod2\_fit <- train(medv ~ rm,   
 data = train,  
 method = "lm",  
 trControl = trainControl(method = "none"))  
mod2\_fit$finalModel

Call:  
lm(formula = .outcome ~ ., data = dat)  
  
Coefficients:  
(Intercept) rm   
 -32.965 8.849

#or just use  
mod2$finalModel

Call:  
lm(formula = .outcome ~ ., data = dat)  
  
Coefficients:  
(Intercept) rm   
 -32.965 8.849

1. Consider three separate KNN models, each using one of the three predictors above.
   * Find the optimal for each of these models using 10 fold CV on the training data only
   * Compare the CV error for the three tuned models and select a best model.
   * With your best model, fit it using the entire training data set.

mod1\_knn <- train(medv ~ age,   
 data = train,  
 method = "knn",  
 trControl = tc,   
 tuneGrid = data.frame(k = 1:60))  
mod2\_knn <- train(medv ~ rm,   
 data = train,  
 method = "knn",  
 trControl = tc,   
 tuneGrid = data.frame(k = 1:60))  
mod3\_knn <- train(medv ~ ptratio,   
 data = train,  
 method = "knn",  
 trControl = tc,   
 tuneGrid = data.frame(k = 1:60))  
mod1\_knn$results |>  
 filter(k == mod1\_knn$bestTune$k)

k RMSE Rsquared MAE RMSESD RsquaredSD MAESD  
1 34 8.733989 0.1528798 6.108134 1.141589 0.1314991 0.734765

mod2\_knn$results |>  
 filter(k == mod2\_knn$bestTune$k)

k RMSE Rsquared MAE RMSESD RsquaredSD MAESD  
1 21 6.252258 0.5430292 4.46823 1.150762 0.07312772 0.6603687

mod3\_knn$results |>  
 filter(k == mod3\_knn$bestTune$k)

k RMSE Rsquared MAE RMSESD RsquaredSD MAESD  
1 18 7.674413 0.35587 5.506717 2.019486 0.2066331 1.190259

The overall best model on the training set uses rm as a predictor and has $k=$21. This model is actually contained in the `mod2\_knn$finalModel` object but we can refit it explicitly.

mod2\_knn\_fit <- train(medv ~ rm,   
 data = train,  
 method = "knn",  
 trControl = trainControl(method = "none"),  
 tuneGrid = mod2\_knn$bestTune)  
mod2\_knn\_fit$finalModel

21-nearest neighbor regression model

#or just use  
mod2\_knn$finalModel

21-nearest neighbor regression model

1. Predict on your test set using both of your ‘best’ models. Compute the test set RMSEs. Which model is the overall best?

#slr model  
sqrt(mean((test$medv - predict(mod2\_fit, test))^2))

[1] 5.61531

# or use caret  
postResample(predict(mod2\_fit$finalModel, test), test$medv)

RMSE Rsquared MAE   
5.6153098 0.6005255 4.0565172

#knn model  
sqrt(mean((test$medv - predict(mod2\_knn\_fit, newdata = test))^2))

[1] 5.326581

postResample(predict(mod2\_knn\_fit, test), test$medv)

RMSE Rsquared MAE   
5.3265813 0.6392572 3.9285806

The best model in terms of test error is the knn model! Of course, we’d likely want to have more than one predictor considered in each model. We’ll get there!

## Problem 3

It is important to really understand cross-validation as it is used in many situation. In this problem we’ll do our own basic cross-validation.

Consider the iris data set built into R. This data set can be read into python via the sklearn package using the following code:

from sklearn import datasets  
  
# Load the dataset  
iris = datasets.load\_iris()

1. Randomly split the data into five distinct folds.

indices <- sample(1:nrow(iris), nrow(iris), replace = FALSE)  
fold1 <- iris[indices[1:30], ]  
fold2 <- iris[indices[31:60], ]  
fold3 <- iris[indices[61:90], ]  
fold4 <- iris[indices[91:120], ]  
fold5 <- iris[indices[121:150], ]

1. Using the first four folds, fit an SLR model with Sepal.Length as the predictor and Petal.Length as the response. Predict on the fifth fold and compute the MSE on those predictions.

one\_two\_three\_four\_fit <- lm(Petal.Length ~ Sepal.Length,   
 data = rbind(fold1,   
 fold2,  
 fold3,  
 fold4))  
fifth\_MSE <- mean((fold5$Petal.Length-predict(one\_two\_three\_four\_fit, newdata = fold5))^2)

1. Repeat this process, using the fourth fold as the test fold. Then again, with the third fold as the test fold, etc.

one\_two\_three\_five\_fit <- lm(Petal.Length ~ Sepal.Length,   
 data = rbind(fold1,   
 fold2,  
 fold3,  
 fold5))  
fourth\_MSE <- mean((fold4$Petal.Length-predict(one\_two\_three\_five\_fit, newdata = fold4))^2)  
  
one\_two\_four\_five\_fit <- lm(Petal.Length ~ Sepal.Length,   
 data = rbind(fold1,   
 fold2,  
 fold4,  
 fold5))  
third\_MSE <- mean((fold3$Petal.Length-predict(one\_two\_four\_five\_fit, newdata = fold3))^2)  
  
one\_three\_four\_five\_fit <- lm(Petal.Length ~ Sepal.Length,   
 data = rbind(fold1,   
 fold3,  
 fold4,  
 fold5))  
second\_MSE <- mean((fold2$Petal.Length-predict(one\_three\_four\_five\_fit, newdata = fold2))^2)  
  
two\_three\_four\_five\_fit <- lm(Petal.Length ~ Sepal.Length,   
 data = rbind(fold2,   
 fold3,  
 fold4,  
 fold5))  
first\_MSE <- mean((fold1$Petal.Length-predict(two\_three\_four\_five\_fit, newdata = fold1))^2)

1. Combine the MSE across the folds (averaging them is fine here) to create a CV error.

#CV error  
mean(c(first\_MSE, second\_MSE, third\_MSE, fourth\_MSE, fifth\_MSE))

[1] 0.760811