HW4

Your Name Goes Here

Details

Due Date

Please commit your submission for this assignment by 5:00 PM Saturday Oct 19.

Grading

20% of your grade on this assignment is for completion. A quick pass will be made to ensure that you've made a reasonable attempt at all problems.

Some of the problems will be graded more carefully for correctness. In grading these problems, an emphasis will be placed on full explanations of your thought process. You usually won't need to write more than a few sentences for any given problem, but you should write complete sentences! Understanding and explaining the reasons behind your decisions is more important than making the "correct" decision.

Solutions to all problems will be provided.

Collaboration

You are allowed to work with others on this assignment, but you must complete and submit your own write up. You should not copy large blocks of code or written text from another student.

Sources

You may refer to class notes, our textbook, Wikipedia, etc.. All sources you refer to must be cited in the space I have provided at the end of this problem set.

In particular, you may find the following resources to be valuable:

- Courses assigned on DataCamp
- Example R code from class
- Cheat sheets and resources linked from [http://www.evanlray.com/stat340_f2019/resources.html]

Load Packages

The following R code loads packages needed in this assignment.

```r
library(readr)
library(dplyr)
library(ggplot2)
library(caret)
```

Conceptual Problems

If you prefer, you can write out your answers to the conceptual problems by hand and then either turn in a physical copy or scan and commit them to GitHub.

Problem 1: Adapted from ISLR Exercise 2.7

The table below provides a training data set containing six observations, two predictors (explanatory variables), and one qualitative response variable (knit the document to view the table).

Obs. Index	X_1	X_2	Y
1	3	0	Red
2	2	0	Green
3	0	3	Red
4	1	2	Green
5	-1	1	Green
6	1	0	Red

Suppose we wish to use this data set to make a prediction for Y when $X_1 = 0$ and $X_2 = 0$ using K nearest neighbors.

(a) Compute the Euclidean distance between each observation and the test point, $(0, 0)$.

Obs. Index	X_1	X_2	Y	Distance from Test Point
1	3	0	Red	$\sqrt{3^2 + 0^2} = 3$
2	2	0	Green	$\sqrt{2^2 + 0^2} = 2$
3	0	3	Red	$\sqrt{0^2 + 3^2} = 3$
4	1	2	Green	$\sqrt{1^2 + 2^2} \approx 2.236$
5	-1	1	Green	$\sqrt{(-1)^2 + 1^2} \approx 1.414$
6	1	0	Red	$\sqrt{1^2 + 0^2} = 1$

(b) What is our prediction ($\hat{Y}$) with $K = 1$? Why?

The single nearest neighbor to the test point is observation number 6, which is in the Red class. Our prediction is $\hat{Y}$ = Red (the most prevalent class among the neighbors).

(c) What is our prediction with $K = 3$? Why?

The 3 nearest neighbors to the test point are observation numbers 6, 5, and 2, which are the Red, Green, and Green classes respectively. Our prediction is $\hat{Y}$ = Green (the most prevalent class among the neighbors).

(d) If the Bayes decision boundary is highly nonlinear, then would we expect the best value for K to be large or small? Why?

We would expect the best value for K to be small for a highly nonlinear decision boundary. When only a small number of nearest neighbors are used for prediction, the predicted class can change rapidly as the explanatory variables change, allowing the method to capture a nonlinear decision boundary.

Problem 2: Adapted from ISLR Exercise 4.4

When the number of features p is large, there tends to be a deterioration in the performance of KNN and other *local* approaches that perform prediction using only observations that are *near* the test observation for which a prediction must be made. This phenomenon is known are the *curse of dimensionality*, and it ties into the fact that non-parametric approaches often perform poorly when p is large. We will now investigate this "curse".

(a) Suppose that we have a set of observations, each with measurements on $p = 1$ feature, X. We assume that X is uniformly (evenly) distributed on the interval $[0, 1]$. Associated with each observation is a response value Y. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with $X = 0.6$, we will use observations in the range $[0.55, 0.65]$. We will make a suitable adjustment at the boundary of the range of values for X; for example, to predict the response for a test observation with $X = 0.03$, we will use observations in the range $[0.00, 0.10]$. On average, what fraction of the available observations will we use to make the prediction? You do not need to prove that you answer is correct formally; an informal/intuitive justification is fine.

If the explanatory variable X is uniformly distributed on the interval $[0, 1]$, we would expect about 10% (or a proportion 0.1) of the observations to fall within 10% of the range of X that is closest to the test point.

This can be justified formally as follows:

Suppose that $X \sim \text{Unif}(0, 1)$ and a and b are the endpoints of the interval we are considering, defining a neighborhood of the test point. Since the interval $[a, b]$ contains the 10% of the range of values for X that are closest to the test observation, we must have $b - a = 0.1$.

Note that the probability density function of X is $f(x) = \begin{cases} 1 & \text{if } x\ in[0,1] \\ 0 & \text{otherwise} \end{cases}$.

Then we can calculate the expected proportion of observations falling in the interval $[a, b]$ as:

$$\begin{aligned} P(X \in [a,b]) &= \int_a^b f(x)dx \\ &= \int_a^b 1dx \\ &= b - a \\ &= 0.1 \end{aligned}$$

(b) Now suppose that we have a set of observations, each with measurements on $p = 2$ features, X_1 and X_2. We assume that (X_1, X_2) are uniformly distributed on the unit square $[0, 1] \times [0, 1]$. We wish to predict a test observation's response using only observations that are within 10% of the range of X_1 *and* within 10% of the range of X_2 closest to that test observation. For instance, in order to predict the response for a test observation with $X_1 = 0.6$ and $X_2 = 0.35$, we will use observations in the range $[0, 55, 0.65]$ for X_1 and $[0.30, 0.40]$ for X_2. On average, what fraction of the available observations will we use to make the prediction?

Now, we are imagining that the possible values of our test points are the unit square, and our neighborhood of the test point is a square with side lengths 0.1. So the area of the neighborhood is $.1^2 = 0.01$. We expect the proportion of observations falling within the test neighborhood to be about 0.01.

More formally, the situation above is represented by the following set up:

$X_1 \sim \text{Unif}(0, 1)$

$X_1 \sim \text{Unif}(0, 1)$

X_1 is independent of X_2.

Also, we now have an interval $[a_1, b_1]$ for values of X_1 and an interval $[a_2, b_2]$ for values of X_2, where $b_1 - a_1 = b_2 - a_2 = 0.1$.

Now,

$$\begin{aligned} P(X_1 \in [a_1, b_1] \text{ and } X_2 \in [a_2, b_2]) &= P(X_1 \in [a_1, b_1]) \times P(X_2 \in [a_2, b_2]) \\ &= 0.1^2 = 0.01 \end{aligned}$$

(c) Now suppose that we have a set of observations on $p = 100$ features. Again, the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10% of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?

Following the pattern established in parts (a) and (b), we see that the proportion of observations falling within the specified neighborhood of the test point is 0.1^100 if $p = 100$.

(d) Using your answers to parts (a) - (c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

We have seen that when p is large, the proportion of training set observations falling near a test observation is very small. If the features were uniformly distributed and we had 100 features, we would need 10^{101} (a huge number!) observations to get to a place where we'd expect to find even a single observation that was within 10% of the range of possible values for all features simultaneously. This is a problem for methods like KNN because those methods base their predictions entirely on the response variable value for "neighboring" points, in the hope that the response variable for those neighboring points will be similar to the response variable for the test set. If the closest neighbors are too far away, that hope may be misplaced.

(e) Now suppose that we wish to make a prediction for a test observation by creating a p-dimensional hypercube centered around the test observation that contains, on average, 10% of the training observations. For each of $p = 1$, $p = 2$, and $p = 100$, what is the length of each side of the hypercube? That is, how long does each side of the hypercube have to be in order for the cube to contain 10% of the training set observations, on average? Comment on your answer.

Note: a hypercube is a generalization of a cube to an arbitrary number of dimensions. When $p = 1$*, a hypercube is simply a line segment; when* $p = 2$ *it is a square; when* $p = 3$ *it is what we usually refer to as a cube; and when* $p = 100$ *it is a 100-dimensional "cube"*

Call the length of one side of our hypercube l; this is unknown that we want to find. We do know that the proportion of observations falling in our hypercube should be 0.1. Building on the work we did in parts a through c, for p features we should have $l^p = 0.1$, or $l = 0.1^{1/p}$.

For p = 1, we find $l = 0.1$.

For p = 2, we find $l = 0.316$.

For p = 100, we find $l = 0.977$.

In order for our neighborhood to include about 10% of the observations, we need the interval side length to be 0.1 for p = 1, as we saw in part (a).

For $p = 2$, our hypercube sides must be of length 0.316; we have to include about 1/3 of the range of values for each feature in order for our final "neighborhood" to contain about 10% of the training set observations.

For $p = 100$, our hypercube sides must be of length 0.977; we have to include about 98% of the range of values for each feature in order for our final "neighborhood" to contain about 10% of the training set observations.

The side length for $p = 100$ is so large that I would not expect the resulting predictions to be very meaningful.

Applied Problems

Problem 3: Human Development Index, redux

In the first homework assignment, we considered polynomial regression models for the relationship between per capital gross national income (`gni_per_cap`, the explanatory variable) and the Human Development Index (`hdi`, the response variable). These data are from Statistics: Data and Models, 4th edition, by De Veaux et al. Recall that none of our polynomial fits were very successful. Here we will consider using K nearest neighbors (KNN) regression instead.

The following R code reads in data including the HDI and gross national income per capita for 187 countries as of 2012, does a train/test split of the data, and creates an initial plot of the training set data including a degree 2 polynomial fit and a degree 3 polynomial fit, neither of which is very satisfying.

```r
country_data <- read_csv("https://mhc-stat140-2017.github.io/data/sdm4/HDI_2012.csv")
```

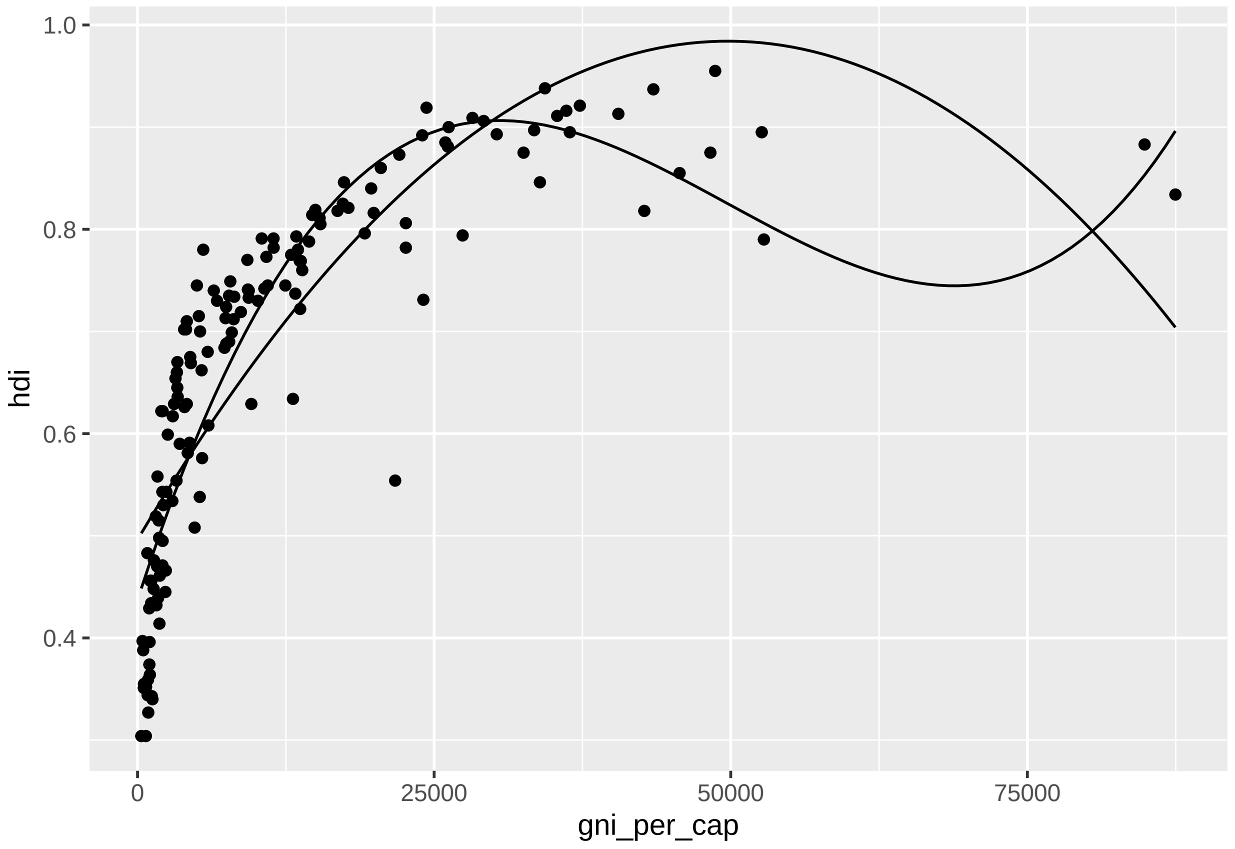
```
## Parsed with column specification:
## cols(
##   `HDI rank` = col_double(),
##   Country = col_character(),
##   Abbreviation = col_character(),
##   HDI = col_double(),
##   `Life Expectancy` = col_double(),
##   `Mean School yrs 2010` = col_double(),
##   `Exp school yrs 2011` = col_double(),
##   `GNI/cap2012` = col_double(),
##   `GNI rank-HDIrank` = col_double(),
##   `NonIncome HDI 2012` = col_double(),
##   Type = col_character()
## )
```

```r
names(country_data) <- c("hdi_rank", "country", "country_abbr", "hdi", "life_expectancy", "mean_school_
country_data <- as.data.frame(country_data)

set.seed(97466)
train_inds <- caret::createDataPartition(country_data$hdi, p = 0.8)
train_country_data <- country_data %>% dplyr::slice(train_inds[[1]])
test_country_data <- country_data %>% dplyr::slice(-train_inds[[1]])

predict_poly <- function(x, degree) {
  fit_poly <- lm(hdi ~ poly(gni_per_cap, degree = degree), data = train_country_data)
  predict(fit_poly, data.frame(gni_per_cap = x))
}
```

```r
ggplot(data = train_country_data, mapping = aes(x = gni_per_cap, y = hdi)) +
  geom_point() +
  stat_function(fun = predict_poly, args = list(degree = 2), color = "black") +
  stat_function(fun = predict_poly, args = list(degree = 3), color = "black")
```



(a) Select K for K nearest neighbors regression.

Use cross-validation to select a value of K for a KNN regression fit to these data. Make sure you are summarizing performance of validation set predictions with a measure of performance for regression. You should try enough values of K that you can see a dip and then subsequent rise in the cross-validated scores across values of K.

```r
val_folds <- caret::createFolds(train_country_data$hdi, k = 10)
k_vals <- c(1:10, 25, 50, 75)
results <- expand.grid(
    fold_ind = seq_len(10),
    k = k_vals,
    val_mse = NA
  )

for(i in seq_len(10)) {
  train_data <- train_country_data %>% dplyr::slice(-val_folds[[i]])
  val_data <- train_country_data %>% dplyr::slice(val_folds[[i]])

  for(k in k_vals) {
    knn_fit <- train(
      hdi ~ gni_per_cap,
      data = train_data,
      method = "knn",
      preProcess = "scale",
      trControl = trainControl(method = "none"),
      tuneGrid = data.frame(k = k)
    )

    # get predicted values
    y_hats <- predict(knn_fit, newdata = val_data)

    # mean squared error
    save_ind <- which(results$fold_ind == i & results$k == k)
    results$val_mse[save_ind] <- mean((y_hats - val_data$hdi)^2)
  }
}
results %>%
  group_by(k) %>%
  summarize(mean(val_mse))
```

```
## # A tibble: 13 x 2
##         k `mean(val_mse)`
##     <dbl>           <dbl>
##  1      1         0.00689
##  2      2         0.00515
##  3      3         0.00447
##  4      4         0.00416
##  5      5         0.00401
##  6      6         0.00396
##  7      7         0.00359
##  8      8         0.00343
##  9      9         0.00328
## 10     10         0.00329
## 11     25         0.00341
## 12     50         0.00605
## 13     75         0.0113
```

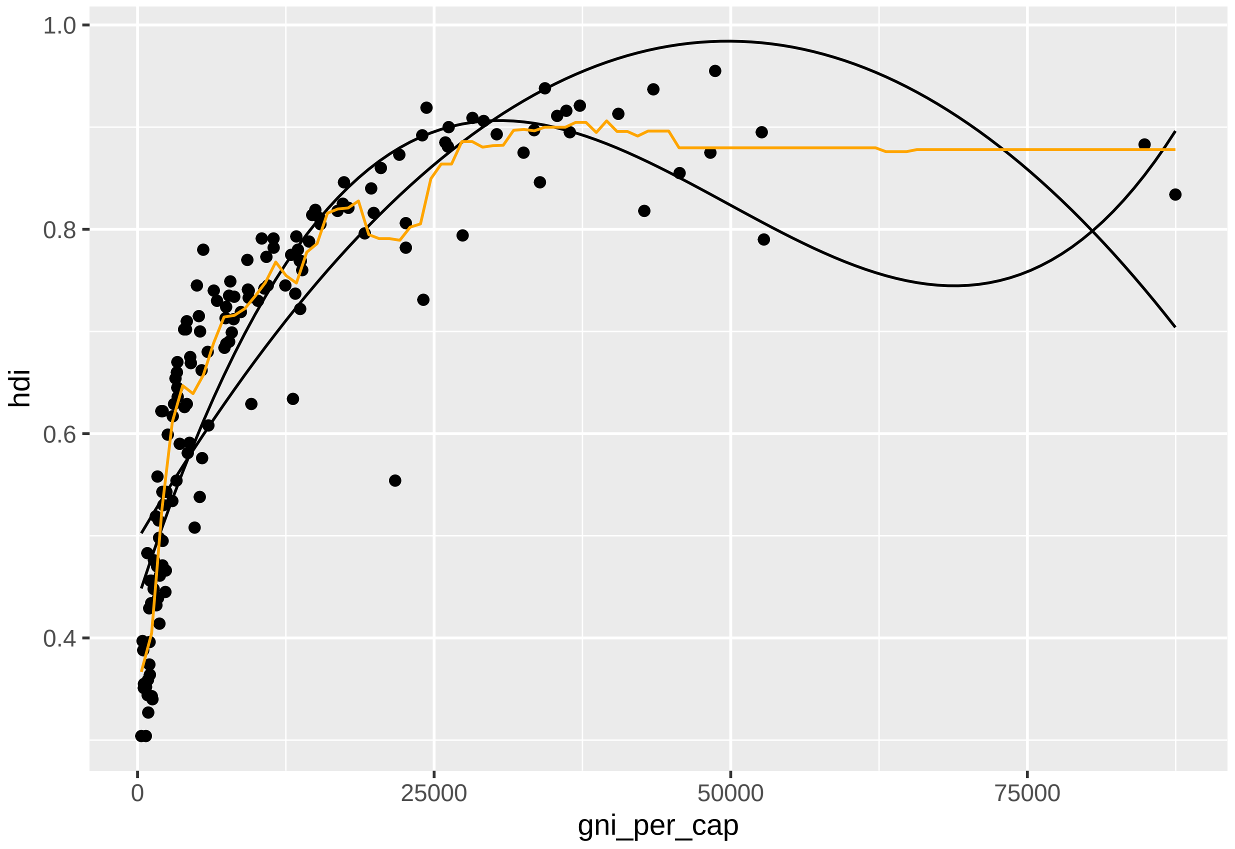
Based on the results above, I would choose k = 8.

(b) Add a plot of your estimated KNN regression model to the plot below.

```
knn_fit <- train(
  hdi ~ gni_per_cap,
  data = train_country_data,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = 8)
)

predict_knn <- function(x) {
  predict(knn_fit, newdata = data.frame(gni_per_cap = x))
}

ggplot(data = train_country_data, mapping = aes(x = gni_per_cap, y = hdi)) +
  geom_point() +
  stat_function(fun = predict_poly, args = list(degree = 2), color = "black") +
  stat_function(fun = predict_poly, args = list(degree = 3), color = "black") +
  stat_function(fun = predict_knn, color = "orange")
```



(c) Obtain a measure of test set performance for your selected KNN model and for degree 2 and degree 3 polynomial regression models. Each of these models should be fit to the full training data set.

```
# knn
mean((test_country_data$hdi - predict(knn_fit, newdata = test_country_data))^2)
```

```
## [1] 0.001670139
```

```
fit_poly <- lm(hdi ~ poly(gni_per_cap, degree = 2), data = train_country_data)
mean((test_country_data$hdi - predict(fit_poly, newdata = test_country_data))^2)
```

```
## [1] 0.005043657
```

```
fit_poly <- lm(hdi ~ poly(gni_per_cap, degree = 3), data = train_country_data)
mean((test_country_data$hdi - predict(fit_poly, newdata = test_country_data))^2)
```

```
## [1] 0.003213821
```

The KNN model has better test set performance than the polynomial regression models.

Problem 4: Adapted from ISLR Example 4.11

In this problem, you will develop a model to predict whether a given car gets high or low gas mileage using the `Auto` data set that is provided with the ISLR package. The code below loads the data and creates a new variable called `mpg_category` that is either "high" if the miles per gallon rating for that car is higher than the median or "low" if the miles per gallon rating for that car is less than or equal to the median. It then gets a train/test split of the data and further subdivisions of the training set into 10 folds for cross-validation.

```
library(ISLR)

Auto <- ISLR::Auto %>%
  mutate(
    mpg_category = ifelse(mpg > median(mpg), "high", "low") %>% factor()
  )

head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                        name mpg_category
## 1 chevrolet chevelle malibu          low
## 2         buick skylark 320          low
## 3        plymouth satellite          low
## 4             amc rebel sst          low
## 5               ford torino          low
## 6          ford galaxie 500          low
```

```
set.seed(87053)
train_inds <- caret::createDataPartition(Auto$mpg_category, p = 0.8)
Auto_train <- Auto %>% dplyr::slice(train_inds[[1]])
Auto_test <- Auto %>% dplyr::slice(-train_inds[[1]])
val_folds <- caret::createFolds(Auto_train$mpg_category, k = 10)
```

(a) Select K for K nearest neighbors classification.

Use cross-validation to select a value of K for a KNN classification fit to these data. Make sure you are summarizing performance of validation set predictions with a measure of performance for classification. You should try enough values of K that you can see a dip and then subsequent rise in the cross-validated scores across values of K.

```
k_vals <- c(1:10, 25, 50, 75, 100, 150, 200, 250, 300)
results <- expand.grid(
    fold_ind = seq_len(10),
    k = k_vals,
    val_class_error = NA
  )

for(i in seq_len(10)) {
  train_data <- Auto_train %>% dplyr::slice(-val_folds[[i]])
  val_data <- Auto_train %>% dplyr::slice(val_folds[[i]])
  for(k in k_vals) {
    knn_fit <- train(
      mpg_category ~ cylinders + displacement + horsepower + weight + acceleration + year + origin,
      data = train_data,
      method = "knn",
      preProcess = "scale",
      trControl = trainControl(method = "none"),
      tuneGrid = data.frame(k = k)
    )

    # get predicted values
    y_hats <- predict(knn_fit, newdata = val_data, type = "raw")

    # classification error rate
    save_ind <- which(results$fold_ind == i & results$k == k)
    results$val_class_error[save_ind] <- mean(y_hats != val_data$mpg_category)
  }
}
```

```
## Warning in knn3Train(train = structure(c(4.72929024810445,
## 4.72929024810445, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.71251699564226,
## 4.71251699564226, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.7070065729838,
## 4.7070065729838, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.74160916699445,
## 4.74160916699445, : k = 300 exceeds number 284 of patterns

## Warning in knn3Train(train = structure(c(4.74917702265313,
## 4.74917702265313, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.75731903459528,
## 4.75731903459528, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.67604776196149,
## 4.67604776196149, : k = 300 exceeds number 282 of patterns

## Warning in knn3Train(train = structure(c(4.7158437332718,
## 4.7158437332718, : k = 300 exceeds number 283 of patterns

## Warning in knn3Train(train = structure(c(4.77505010883335,
## 4.77505010883335, : k = 300 exceeds number 283 of patterns

## Warning in knn3Train(train = structure(c(4.78424907316163,
## 4.78424907316163, : k = 300 exceeds number 284 of patterns
```

```
results %>%
  group_by(k) %>%
  summarize(mean(val_class_error))
```

```
## # A tibble: 18 x 2
##        k `mean(val_class_error)`
##    <dbl>                   <dbl>
##  1     1                  0.111
##  2     2                  0.137
##  3     3                  0.0987
##  4     4                  0.108
##  5     5                  0.0890
##  6     6                  0.0987
##  7     7                  0.0988
##  8     8                  0.0827
##  9     9                  0.0986
## 10    10                  0.0954
## 11    25                  0.0985
## 12    50                  0.102
## 13    75                  0.102
## 14   100                  0.0889
## 15   150                  0.108
## 16   200                  0.118
## 17   250                  0.165
## 18   300                  0.562
```

Based on the results above I would select k = 8. However, it appears that there is a wide range of values for k that would work about as well.

(b) Obtain a measure of test set performance for your selected KNN model and for a null model that always predicts "high" (which is by construction tied for the most commonly occurring class in the training data set).

```
knn_fit <- train(
  mpg_category ~ cylinders + displacement + horsepower + weight + acceleration + year + origin,
  data = Auto_train,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = 8)
)

mean(Auto_test$mpg_category != predict(knn_fit, newdata = Auto_test))
```

```
## [1] 0.05128205
```

```
mean(Auto_test$mpg_category != "high")
```

```
## [1] 0.5
```

KNN has an error rate of about 0.05, much better than the error rate of 0.5 if we always predict "high".

Collaboration and Sources

If you worked with any other students on this assignment, please list their names here.

If you referred to any sources (including our text book), please list them here. No need to get into formal citation formats, just list the name of the book(s) you used or provide a link to any online resources you used.