Gambrel 6501x HW 1

```
rm(list = ls())

if (Sys.info()[[1]] == "Windows") {
  setwd("E:/GoogleDrive/edx/6501x/week1")
}

pacman::p_load(dplyr, tidyr, magrittr, kernlab, kknn, readr, purrr)
```

Q1

My company makes practice management software for physician groups. Frequently, they want to know the expected amount of patient no-shows so that they can efficiently schedule patients' and providers' time. To predict whether a scheduled patient shows up as scheduled, we might use:

1. Prior patient visits (have they no-showed in the past?)
2. Patient access to transportation (do they own a vehicle or rely on public transportation / rides from others)
3. How long ago was the appointment made
4. Is this a regular checkup or a sick visit
5. Time of schedule appointment

Q2.1

```
cc <- read_tsv("credit_card_data-headers.txt")

cc_matrix <- cc %>% as.matrix()

best_combo <- ''
best_val <- 0
best_params <- c()

sv <- function(c_val, kern) {
  model <- ksvm(x = cc_matrix[, 1:10], y = as.factor(cc_matrix[, 11]), type = "C-svc",
                kernel = kern, C = c_val, scaled = T)

  a <- colSums(cc_matrix[model@SVindex, 1:10] * model@coef[[1]])
  a0 <- sum(a * cc_matrix[1, 1:10]) - model@b
  pred <- predict(model, cc_matrix[, 1:10])
  correct = sum(pred == cc_matrix[, 11]) / nrow(cc_matrix)

  print(paste0("C val: ", c_val, ". Kern: ", kern, ". Proportion correct: ", correct))

  if (correct > best_val) {
    best_val <<- correct
    best_combo <<- paste(c_val, kern, sep = ', ')
    best_params <<- c(a, a0)
  }
```

```
}

cross2(
  c(.0001, .001, 0.01, 0.1, 1, 10, 100),
  c("vanilladot", "splinedot", "rbfdot", "anovadot")
) %>%
    map(lift(sv))
```

```
print(best_combo)
```

```
## [1] "10, splinedot"
```

I tested several kernels and C-values and found that a spline kernel with C = 10 was the best predictor of the data. The equation was:

$y = -852694 + 0.271 * A1 + 549.21 * A2 - 51.655 * A3 - 9.52 * A8 - 0.667 * A9 + 3.153 * A10 - 10.414 * A11 + 5.738 * A12 - 4305.13 * A14 + 5911.14 * A15$

This classified 97.86% of the responses correctly.

Q2.2

I trained a series of KNN models by splitting the data into a training and testing set.

```
# testing K choices
for (k_choice in 2:10){

    knn <- kknn(R1~., cc[1:500, ], cc[500:654, ], k = k_choice, scale = T)
    pred <- round(fitted(knn))

  correct = sum(pred == cc[500:654, 11]) / 155
  print(paste0("k = ", k_choice, ": ", correct))
}
```

```
## [1] "k = 2: 0.793548387096774"
## [1] "k = 3: 0.793548387096774"
## [1] "k = 4: 0.793548387096774"
## [1] "k = 5: 0.838709677419355"
## [1] "k = 6: 0.832258064516129"
## [1] "k = 7: 0.838709677419355"
## [1] "k = 8: 0.832258064516129"
## [1] "k = 9: 0.825806451612903"
## [1] "k = 10: 0.832258064516129"
```

From these results, I would choose K = 5 as the best parameter choice. It had good fit with minimal marginal improvement from adding additional clusters. This combination correctly classified 83.87% of the observations.

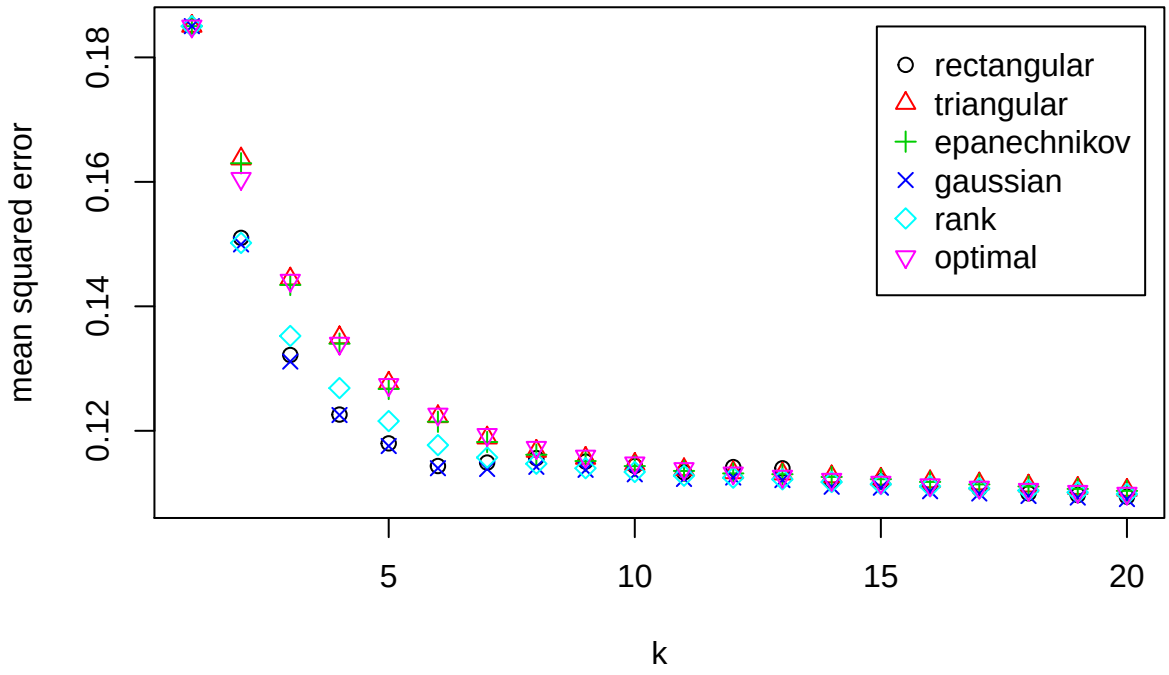
Q3.a

```
trained <- train.kknn(R1 ~ ., cc, kmax = 20,
                      kernel = c("rectangular", "triangular", "epanechnikov",
                                 "gaussian", "rank", "optimal"),
```

```
                      distance = 2)
trained$best.parameters
```

```
## $kernel
## [1] "gaussian"
##
## $k
## [1] 20
```

```
plot(trained)
```



The `kknn` package's cross-validation optimizer selected a gaussian kernel with 20 clusters as the clustering method that minimized squared errors. However, since adding extra clusters might cause overfitting, it is worth checking the marginal improvement of these higher K values. Viewing the error plot, it appears that a model with K = 6 and a gaussian kernel will have almost identical classification performance.

Q3.b

I split my data into 3 parts: a 60% training set, a 20% validation set, and a 20% test set, in order to accurately estimate out-of-sample performance. I then re-run my support vector checks of C and kernel choices.

```
index <- 1:654
train <- sample(index, size = round(654*.6)) %>% sort()
val <- sample(index[-train], size = round(654 * .2)) %>% sort()
test <- index[-c(train, val)]
```

```
best_combo <- ''
best_val <- 0
best_params <- c()

sv_val <- function(c_val, kern) {
  model <- ksvm(x = cc_matrix[train, 1:10], y = as.factor(cc_matrix[train, 11]), type = "C-svc",
                kernel = kern, C = c_val, scaled = T)

  a <- colSums(cc_matrix[model@SVindex, 1:10] * model@coef[[1]])
  a0 <- sum(a * cc_matrix[1, 1:10]) - model@b
  pred <- predict(model, cc_matrix[val, 1:10])
  correct = sum(pred == cc_matrix[val , 11]) / nrow(cc_matrix[val, ])

  print(paste0("C val: ", c_val, ". Kern: ", kern, ". Proportion correct: ", correct))

  if (correct > best_val) {
    best_val <<- correct
    best_combo <<- paste(c_val, kern, sep = ', ')
    best_params <<- c(a, a0)
  }
}


cross2(
  c(.0001, .001, 0.01, 0.1, 1, 10, 100),
  c("vanilladot", "rbfdot", "anovadot")
) %>%
  map(lift(safely(sv_val)))
```

```
print(best_combo)
```

```
## [1] "0.01, vanilladot"
```

```
print(best_params)
```

```
##            A1            A2            A3            A8            A9
## -7.963425e-02  2.896641e+00  4.636778e-01  7.179174e-01  1.757988e-01
##           A10           A11           A12           A14           A15
## -2.092815e-01  9.262195e-01 -6.980796e-02 -8.501275e+00  1.211972e+02
##
## -1.626037e+03
```

The best combination this time around was a C value of 1 using the ANOVA RBF kernel. In order to estimate the model's accuracy, I apply these results to the testing data:

```
knn <- kknn(R1~., cc[train, ], cc[val, ], k = 6, scale = T)
pred <- round(fitted(knn))

model <- ksvm(x = cc_matrix[train, 1:10], y = as.factor(cc_matrix[train, 11]), type = "C-svc",
              kernel = 'anovadot', C = 1, scaled = T)
```

```
##  Setting default kernel parameters
```

```
a <- colSums(cc_matrix[model@SVindex, 1:10] * model@coef[[1]])
a0 <- sum(a * cc_matrix[1, 1:10]) - model@b
pred <- predict(model, cc_matrix[test, 1:10])
```

```r
correct = sum(pred == cc_matrix[test , 11]) / nrow(cc_matrix[test, ])
print(correct)
```

```
## [1] 0.7938931
```

This estimate of performance is much lower than what was found in Question 2.1. This is to be expected - the earlier model used the full dataset without accounting for sample error. This newer result is much closer to what we would expect the model to achieve with new data.