Exercise 1:

```
##########################################################################
################
#   1. Execute the code above.
#     Based on the results, rank the models from "most underfit" to "most overfit".

#install.packages("kernlab")
library(kernlab)
data("spam")
tibble::as.tibble(spam)

is.factor(spam$type)
levels(spam$type)

set.seed(42)
# spam_idx = sample(nrow(spam), round(nrow(spam) / 2))
spam_idx = sample(nrow(spam), 1000)
spam_trn = spam[spam_idx, ]
spam_tst = spam[-spam_idx, ]

fit_caps = glm(type ~ capitalTotal,
               data = spam_trn, family = binomial)
fit_selected = glm(type ~ edu + money + capitalTotal + charDollar,
                   data = spam_trn, family = binomial)
fit_additive = glm(type ~ .,
                   data = spam_trn, family = binomial)
fit_over = glm(type ~ capitalTotal * (.),
               data = spam_trn, family = binomial, maxit = 50)

# training misclassification rate
mean(ifelse(predict(fit_caps) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_selected) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_additive) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_over) > 0, "spam", "nonspam") != spam_trn$type)

library(boot)
```

```
set.seed(1)
cv.glm(spam_trn, fit_caps, K = 5)$delta[1]
cv.glm(spam_trn, fit_selected, K = 5)$delta[1]
cv.glm(spam_trn, fit_additive, K = 5)$delta[1]
cv.glm(spam_trn, fit_over, K = 5)$delta[1]


# most underfit to most overfit:

# fit_caps
# fit_selected
# fit_over
# fit_additive


##############################################################################
################
#   2. Re-run the code above with 100 folds and a different seed. Does your conclusion
change?

#install.packages("kernlab")
library(kernlab)
data("spam")
tibble::as.tibble(spam)

is.factor(spam$type)
levels(spam$type)

set.seed(42)
# spam_idx = sample(nrow(spam), round(nrow(spam) / 2))
spam_idx = sample(nrow(spam), 1000)
spam_trn = spam[spam_idx, ]
spam_tst = spam[-spam_idx, ]

fit_caps = glm(type ~ capitalTotal,
               data = spam_trn, family = binomial)
fit_selected = glm(type ~ edu + money + capitalTotal + charDollar,
                   data = spam_trn, family = binomial)
fit_additive = glm(type ~ .,
```

```
               data = spam_trn, family = binomial)
fit_over = glm(type ~ capitalTotal * (.),
            data = spam_trn, family = binomial, maxit = 50)

# training misclassification rate
mean(ifelse(predict(fit_caps) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_selected) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_additive) > 0, "spam", "nonspam") != spam_trn$type)
mean(ifelse(predict(fit_over) > 0, "spam", "nonspam") != spam_trn$type)

library(boot)
set.seed(10)
cv.glm(spam_trn, fit_caps, K = 100)$delta[1]
cv.glm(spam_trn, fit_selected, K = 100)$delta[1]
cv.glm(spam_trn, fit_additive, K = 100)$delta[1]
cv.glm(spam_trn, fit_over, K = 100)$delta[1]

# the results are still in the same order of most overfit to most underfit

##########################################################################
################

make_conf_mat = function(predicted, actual) {
  table(predicted = predicted, actual = actual)
}

#additive
spam_add_pred = ifelse(predict(fit_additive, spam_tst) > 0,
                  "spam",
                  "nonspam")
spam_add_pred = ifelse(predict(fit_additive, spam_tst, type = "response") > 0.5,
                  "spam",
                  "nonspam")
#caps
spam_caps_pred = ifelse(predict(fit_caps, spam_tst) > 0,
                  "spam",
                  "nonspam")
spam_caps_pred = ifelse(predict(fit_caps, spam_tst, type = "response") > 0.5,
                  "spam",
```

```
                  "nonspam")

#selective
spam_sel_pred = ifelse(predict(fit_selected, spam_tst) > 0,
                  "spam",
                  "nonspam")
spam_sel_pred = ifelse(predict(fit_selected, spam_tst, type = "response") > 0.5,
                  "spam",
                  "nonspam")

#over
spam_over_pred = ifelse(predict(fit_over, spam_tst) > 0,
                  "spam",
                  "nonspam")
spam_over_pred = ifelse(predict(fit_over, spam_tst, type = "response") > 0.5,
                  "spam",
                  "nonspam")

(conf_mat_50 = make_conf_mat(predicted = spam_add_pred, actual = spam_tst$type))
(conf_mat_50 = make_conf_mat(predicted = spam_caps_pred, actual =
spam_tst$type))
(conf_mat_50 = make_conf_mat(predicted = spam_sel_pred, actual = spam_tst$type))
(conf_mat_50 = make_conf_mat(predicted = spam_over_pred, actual = spam_tst$type))

table(spam_tst$type) / nrow(spam_tst)
```

3

The third model is the best model for predicting the classification of spam vs nonspam emails. When running a 5 fold cross validation we found that models one, two, and 4 all had a prediction error of at least 10% or more. Our third model had a prediction error of 6.8% which isn't fantastic however it was the best of the bunch.

Some of the errors that occured are worse than others. It is better for a model to be specific as opposed to sensitive. This is because a specific model will weigh true positives against false negatives instead of weighing true negatives over false positives. What this means is, a more specific model will base its accuracy off of the nonspam emails classified as nonspam divided by the nonspam emails classified as spam.

Exercise 2:

##
1. Use the bank data and create a train / test split.
Based on the results, rank the models from "most underfit" to "most overfit".

```
set.seed(10)
num_obs = nrow(spam)

train_index = sample(num_obs, size = trunc(0.50 * num_obs))
train_data = Ames[train_index, ]
test_data = Ames[-train_index, ]
```

##
2. Run any logistic regression you like with 10-fold cross-validation in order to predict the yes/no variable (y).

```
set.seed(10)
train.control <- trainControl(method = "cv", number = 10)
model <- train(type$spam, data = spam, method = "lm",
            trControl = train.control)
print(model)


tuned_parameters = {'C': [1, 50, 20]}
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
#cv : number of folds, n_jobs: # of cpus
_mykNN = LogisticRegression()
mykNN = GridSearchCV(_mykNN, tuned_parameters, cv=10,
                    scoring='accuracy',
                    verbose=10,
                    n_jobs=2)
mykNN.fit(X_train_std_sk, y_train)
mykNN.best_params_
```

```
In [32]: mykNN.best_params_

Out[32]: {'C': 1}
```

##
#################
4. Create a confusion matrix of your preferred model, evaluated against your test data.

```
make_conf_mat = function(predicted, actual) {
  table(predicted = predicted, actual = actual)
}

fit = glm(type ~ .,data = train_data, family = binomial)

spam_test_pred = ifelse(predict(fit, spam_tst) > 0,
                     "spam",
                     "nonspam")
spam_train_pred = ifelse(predict(fit, spam_tst, type = "response") > 0.5,
                     "spam",
                     "nonspam")
(conf_mat_50 = make_conf_mat(predicted = train_data, actual = spam_tst$type))
```