Homework Assignment #6

Nick Noel, Liz Villa, and Cadee Pinkerton

Due May 12, 2023

Instructions

You should submit either two or three files:

1. You should write your solutions to the Applied Problem in this R Markdown file and submit the (.Rmd) file.
2. You should knit the final solution file to pdf and submit the pdf. If you are having trouble getting code chunks to run, add `eval = FALSE` to the chunks that do not run. If you are having trouble getting R Studio to play nice with your LaTeX distribution, I will begrudgingly accept an HTML file instead.
3. Solutions to the Key Terms and Conceptual Problem can be submitted in a separate Word or pdf file or included in the same files as your solutions to the Applied Problem.

This homework assignment is worth a total of **35 points**.

Key Terms (13 pts)

Read Chapters 8 (Sections 8.1-8.2.3), 10 (Sections 10.1-10.4, 10.6-10.7), and 12 (Sections 12.1-12.4) of Introduction to Statistical Learning, Second Edition. Based on your reading, answer the following questions.

1. How do you determine whether a node in a tree is a *terminal node* or an *internal node*?

Answer: A node in a tree is a terminal if it is the region at the end referred to as the leaf of the tree while the internal node is the point in which the decision is split referred to as the branch of the tree.

2. Briefly explain what is meant by the term *recursive binary splitting*.

Answer: Recursive binary splitting is the process of starting at the top of the tree with all data and splitting the data based on some condition into two groups (binary), into smaller and smaller subsets of the original data.

3. Explain the similarity between *cost-complexity pruning* and the *lasso* model.

Answer: The similarity between cost-complexity pruning and lasso model is how they control for complexity, in lasso models it was controlling the impact that a variable can have on the data, in lasso it pays attention to colinearity while in cost-complexity it pays attention to the number of terminal nodes, punishing models with larger numbers of splits on the data.

4. Write out the equations for two different measures of *node purity* (or node impurity). Why are these measures preferred over simpler accuracy/inaccuracy measures?

Answer:

$$GiniIndex: G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

$$Entropy: D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

These measures are preferred because classification error is not great when looking at growing trees and these methods look at total variance across classes as an alternative measure that provides a more substancial basis.

5. Do decision trees perform better when there is a linear boundary or a highly nonlinear boundary? Why?

Answer: Decision trees perform better when there are highly nonlinear boundaries based on the nature of how trees are split. The binary operator cuts into linear relationships making for worse overall classification. Overall, linear data is better suited for linear regression and nonlinear bounds make for more accurate decision trees.

6. Describe the basic procedure in *bagging*. Could we use *bagging* with models that aren't trees (linear models, generative models, etc.)? Why or why not?

Answer: The basic procedure of bagging is to take advantage of bootstrapping to create many trees that all have different values and then averaging them out in order to reduce the variance of the forest. Each tree has high variance and low bias but when combined a not terrible model can be produced. Bagging can be used in other modeling techniques but it is most effective on random trees since other models generally have lower variance and higher bias.

7. Why might we use the *out-of-bag* prediction error to estimate the test error rate instead of using cross-validation? Describe how to compute it.

Answer: In order to compute out-of-bag prediction error, since each bagged tree takes on average 2/3 of the data the remaining 1/3, out of bag data, will be estimated based on the trees resulting in B/3 predictions which can be averaged for linear estimates or use a majority vote for classification data. This is a valid estimate of the data and thus we can use it over cross validation.

8. Explain how to read the bar graph in Figure 8.9.

Answer: Figure 8.9 describes how important variables are based on splitting the data. The variable, Thai, is the most important which means that in almost every tree it was always the initial split, and the importance continues as you move up the graph. More important variables mean they are going to be seen in earlier splits.

9. When would we expect *random forests* to improve over a "regular" bagged trees model and why?

Answer: Random Forests would be better than bagged tree models as it "decorrelates" the data. The issue with bagged trees is the fact that every single tree will look similar if there is a predictor vairable that is highly correlated resulting in every tree looking roughly similar. Random forest ensures that sometimes that variable is not available leading to findings about new variables being important.

10. Briefly explain how *boosted trees* are grown.

Answer: Boosted trees are grown sequentially meaning the next tree learns based on the previous one. This relies on residuals, the residuals of the first tree are applied to the second one and the residuals are updated once again. Overall, instead of fitting to the response, Y, we are fitting to reduce our residuals.

11. Briefly explain what is meant by the terms *hidden unit* and *hidden layer*.

Answer: The hidden layer is the component of the neural network that is made up of multiple hidden units. Hidden units are what are used to apply functions and weighted sums to the data to produce outputs.

12. What is an *activation*? Give two examples of activation functions.

Answer: An activation is the function applied to the data to transform it. Two examples are the sigmoid and ReLu functions which are respectively

$$g(z) = \frac{e^z}{1 + e^z}$$

$$g(z) = z_+ = \begin{cases} 0 & z < 0 \\ z & otherwise \end{cases}$$

13. The machine learning community refers to *weights* and *bias* terms in neural networks. In statistical language, what do these terms correspond to?

Answer: Weights would refer to the coefficients of a variable while bias would be related to the intercept of a model. Generally neural networks are a collection of these models to create better ones.

14. In convolutional neural networks, what is the purpose of a *convolution layer*? What is the purpose of a *pooling layer*? The purpose of a convolution layer is to determine which features are present in an image. The purpose of a pooling layer is to reduce the size of an image by summarizing it.

15. Briefly explain how and why *data augmentation* is performed in image classification. Data augmentation is performed by distorting the images in the training set slightly so they are different but still recognizable to humans and it serves to increase the size of the training set and helps prevent overfitting.

16. Section 10.6 poses the question: "Should we discard all our older tools, and use deep learning on every problem with data?" Answer this question and explain your reasoning. No, if a simpler technique gives similar results, which it can and does in the example, then we should choose that over deep learning since we are not really losing anything by not using deep learning techniques and instead gaining better interpretability, and reducing computational cost.

17. In the example in Section 10.6, the authors first used a lasso model to perform variable selection, then fit a linear model with the predictors selected. How did they use their final model to answer *inference* questions? Why did they do it that way?

Their final model could be used to answer the inference questions about which variables affect salary with an estimate of how much. They did it this way in order to simplify their model as much as possible while still maintaining accuracy.

18. Briefly explain how *gradient descent* methods work. Why is estimating the gradient relatively straightforward when (scaled) MSE is used as the objective function $R(\theta)$?

Answer: Gradient descent is an algorithm used for finding a minimum. It works by applying some function, getting new values, and then running the function again, it is straightforward when using MSE since we are just looking at derivatives.

19. What is a principal component *score* vector? How are the associated *loadings* computed?

Answer: The loadings are the eigenvectors of the covariance matrix. The principal component score vectors are linear combinations of the loadings multiplied by the predictors. The score vectors explain the variation in the predictors. z_1 is the direction in p-dimensional space that explains the most variation in the x_j's. z_2 is the direction orthogonal to z_1 that explains the most remaining variation in the x_j's. This pattern continues for the rest of the score vectors.

20. The book mentions that eigen decomposition can be used to obtain the loadings (in practice most algorithms use a related technique, singular value decomposition, which is faster). What do the eigenvalues of the covariance matrix correspond to? What do the eigenvectors correspond to?

The eigenvalues correspond to the magnitude of the eigenvector. The eigenvectors correspond to the loadings of the principal components. The loading vector defines a direction in feature space along which the data vary the most.

21. What is a scree plot? Why is it useful? A scree plot is a plot that shows the proportion of variance explained by the principal components. It is useful because it allows us to see the point where additional principal components are no longer significantly contributing to the proportion of variance explained thereby helping us choose the appropriate number of principal components.

22. What is the difference between data *missing at random* and *missing not at random* (or "not missing at random")? Is it appropriate to impute data that is missing at random? What about data that is not missing at random? Data missing at random is missing because of reasons not relating to the data or it's method of collection. Data missing not at random is missing because of some characteristic of the data or how it was collected. It is appropriate to impute data that is missing at random but it is not appropriate to impute data that is not missing at random.

23. Give two application areas in which clustering techniques may be useful. You may use the ones identified in the book or describe your own. Clustering techniques may be useful in medical applications such as finding subtypes of cancers based on certain characteristics. It would also be useful for marketing applications to find groups of people who would be receptive to certain kinds of advertising or a type of product.

24. Why is it important to run k-means clustering from many different random initializations? It is important to do so in order to increase our chances of finding the solution at convergence that is the global optimum.

25. Briefly explain how to read a *dendrogram*.

A dendrogram is read by referring to the y-axis which represents dissimilarity and seeing where observations or "leaves" merge in respects to the y-axis which indicates their similarity to one another and they become "branches" which then continue to merge with other leaves or branches.

26. Many of the techniques we learned in this class have been rather "automatic" in the sense that there are fairly well-defined rules for selecting the "best" model and all we need "human judgment" for is determining which of several models "close enough to the best" we should actually use. This does not work for cluster analysis. Give at least two reasons why cluster analysis requires us to think a lot harder before settling on a "best" model.

Two reasons why cluster analysis requires us to think harder before settling on a best model are because measuring the quality of a model is not spelled out in the sense we have no measures such as accuracy or MSE. Another reason is due to the assumptions we have to make when looking at clustering, some assumptions can lead to durastically different models making it more difficult to identify a "best" model.

Conceptual Problems

Conceptual Problem 1 (3 pts)

Consider the random variables $Y_1, Y_2, \ldots, Y_n$ to be independent (but not identically distributed) random variables with pmf given by

$$P(Y_i = m | X = x) = \begin{cases} f_m(x_i), m = 0, 1, \ldots, 9 \\ 0, \text{otherwise} \end{cases}$$

where $f_m(x_i)$ are a collection of $m + 1$ functions of predictor variables X evaluated for observation i. The joint pmf of the Y_i's is then:

$$P(Y_1 = m_1, Y_2 = m_2, \ldots, Y_n = m_n) = \prod_{i=1}^{n} \prod_{m=0}^{9} (f_m(x_i))^{y_{im}}$$

where $y_i m$ is an indicator variable taking the value 1 if $Y_i = m$ and 0 otherwise.

Prove that maximizing the joint pmf is equivalent to minimizing the cross-entropy given in Equation (10.14). (HINT: take the logarithm and use properties of logs.)

If we let $P(Y_1 = m_1, Y_2 = m_2, \ldots, Y_n = m_n) = L(m)$ when we take the log of $L(m)$ equation we get

$log(L(m)) = \sum_{i=1}^{n} \sum_{m=0}^{9} log((f_m(x_i))^{y_{im}})$

then using the properties of log we get

$log(L(m)) = \sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} log(f_m(x_i))$

so when we maximize this equation, we are minimizing equation 10.14 for cross-entropy which is

$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} log(f_m(x_i))$,

since the logarithmic function is a monotonically increasing function.

Applied Problems

Applied Problem 1 (19 pts total)

Using the `Default` dataset, we wish to predict whether a customer will default on their debt.

```
library(ISLR2) # for Default dataset
library(tidyverse)
library(tidymodels)
library(tidyclust)
library(dendextend)
library(GGally)
library(xgboost)
```

Part a (Code: 1 pt)

Divide the Default dataset into a training set of 8000 observations (80% of the data) and a test set containing the remaining 2000 observations.

```
set.seed(437)
default_split <- initial_split(Default, prop = 0.80)
default_train <- training(default_split)
default_test <- testing(default_split)
```

Part b (Code: 3 pts; Explanation: 2 pts)

Fit a k-means clustering model on the training set using the three predictors (`student`, `balance`, and `income`). Either pick a reasonably small (3-4) number of clusters or tune the number of clusters.

Describe the general characteristics of each of the clusters identified. Supplement your description by producing a scatterplot of income and balance showing each cluster in a different color.

```
kmeans_model <- k_means(num_clusters = tune()) %>%
  set_args(nstart = 20)

kmeans_recipe_fv <- recipe(~ student + balance + income, data = default_train) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors()) # deal with different variances

kmeans_wflow_fv <- workflow() %>%
  add_model(kmeans_model) %>%
  add_recipe(kmeans_recipe_fv)
```

```
set.seed(1002)
fv_kfold_tidy <- vfold_cv(default_train, v = 5, repeats = 1)
```

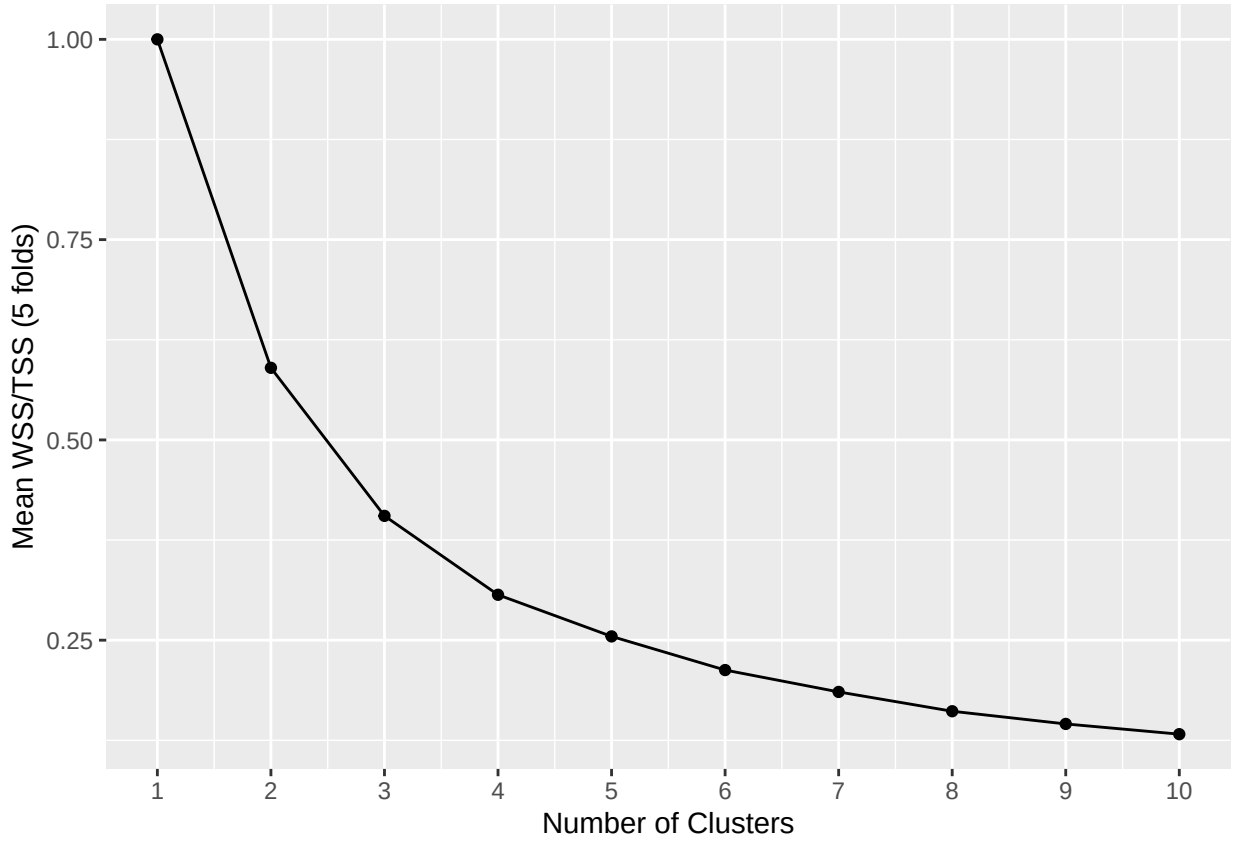
```r
# grid is now expected to be a tibble or data frame instead of a list of named parameters
nclusters_grid <- data.frame(num_clusters = seq(1, 10))

kmeans_tuned_fv <- tune_cluster(kmeans_wflow_fv,
                                resamples = fv_kfold_tidy,
                                metrics = cluster_metric_set(sse_total,
                                                             sse_within_total,
                                                             sse_ratio),
                                grid = nclusters_grid)
```

```
## ! Fold1: preprocessor 1/1, model 8/10: Quick-TRANSfer stage steps exceeded maximu...
## ! Fold3: preprocessor 1/1, model 5/10: Quick-TRANSfer stage steps exceeded maximu...
## ! Fold4: preprocessor 1/1, model 7/10: Quick-TRANSfer stage steps exceeded maximu...
## ! Fold5: preprocessor 1/1, model 4/10: Quick-TRANSfer stage steps exceeded maximu...
## ! Fold5: preprocessor 1/1, model 8/10: Quick-TRANSfer stage steps exceeded maximu...
## ! Fold5: preprocessor 1/1, model 10/10: did not converge in 10 iterations
```

```r
tuned_metrics <- collect_metrics(kmeans_tuned_fv)
```

```r
tuned_metrics %>% filter(.metric == "sse_ratio") %>%
  ggplot(aes(x = num_clusters, y = mean)) +
  geom_point() +
  geom_line() +
  labs(x = "Number of Clusters", y = "Mean WSS/TSS (5 folds)") +
  scale_x_continuous(breaks = seq(1, 10))
```



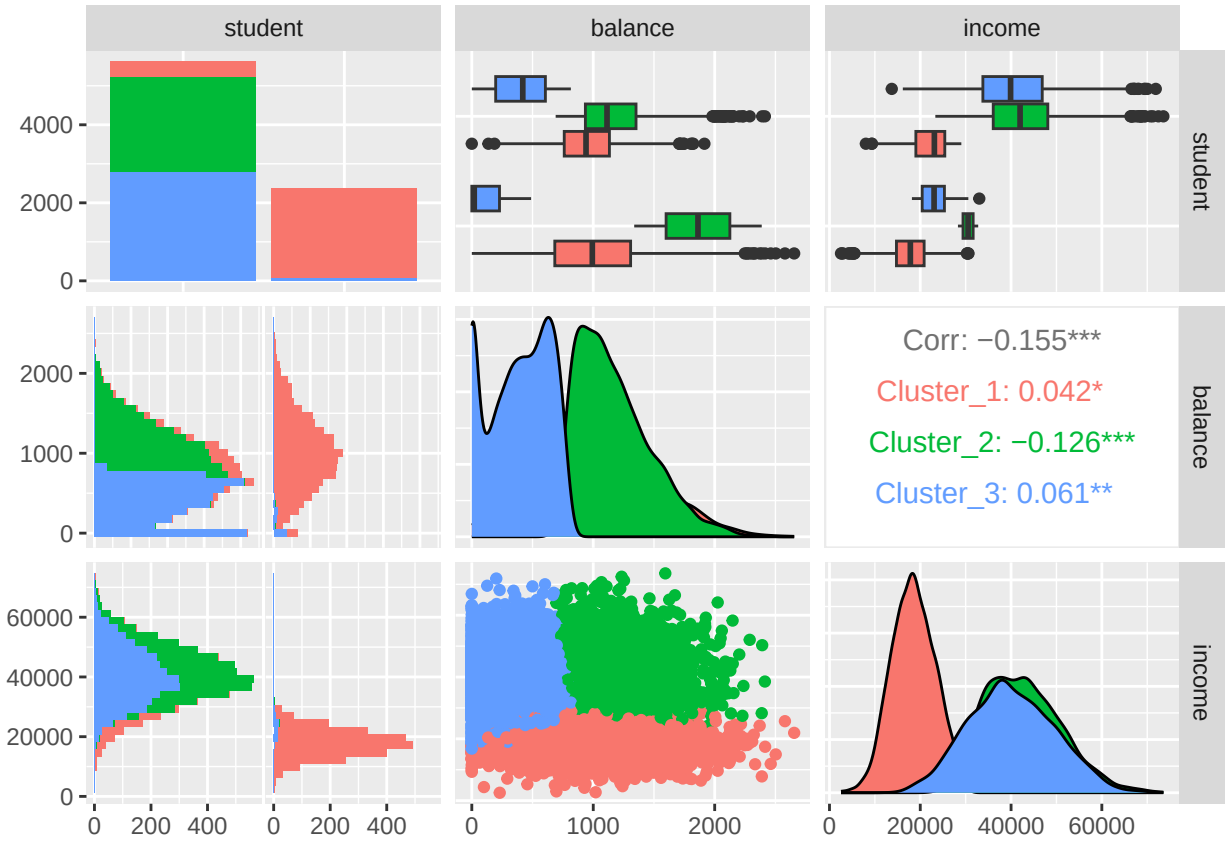

```
kmeans_fv_3clusters <- kmeans_wflow_fv %>%
  finalize_workflow_tidyclust(parameters = list(num_clusters = 3))
```

```
set.seed(437)
# always reset the seed before you re-fit, just in case something weird happens

kmeans_fv_fit3 <- kmeans_fv_3clusters %>%
  fit(data = default_train)
```

```
assignments3 <- bind_cols(
  default_train,
  kmeans_fv_fit3 %>% extract_cluster_assignment())
```

```
ggpairs(assignments3, columns = c("student", "balance", "income"),
        aes(color = .cluster))
```



Red Cluster - Low income, evenly dispersed balance. Green Cluster - Higher balance, high income. Blue Cluster - Low balance, high income.

Part b (Code: 3 pts)

Fit a random forest model (using either `randomForest` or `ranger`) on the training set. Use cross-validation to determine whether `mtry` should be set to 1, 2, or 3.

```
rfC_model <- rand_forest(mode = "classification", engine = "ranger") %>%
  set_args(seed = 437,
           importance = "permutation",
           mtry = tune())

rfC_recipe <- recipe(
  default ~ student + balance + income, # response ~ predictors
  data = default_train
) %>% step_dummy(all_nominal_predictors())

rfC_wflow <- workflow() %>%
  add_model(rfC_model) %>%
  add_recipe(rfC_recipe)
```

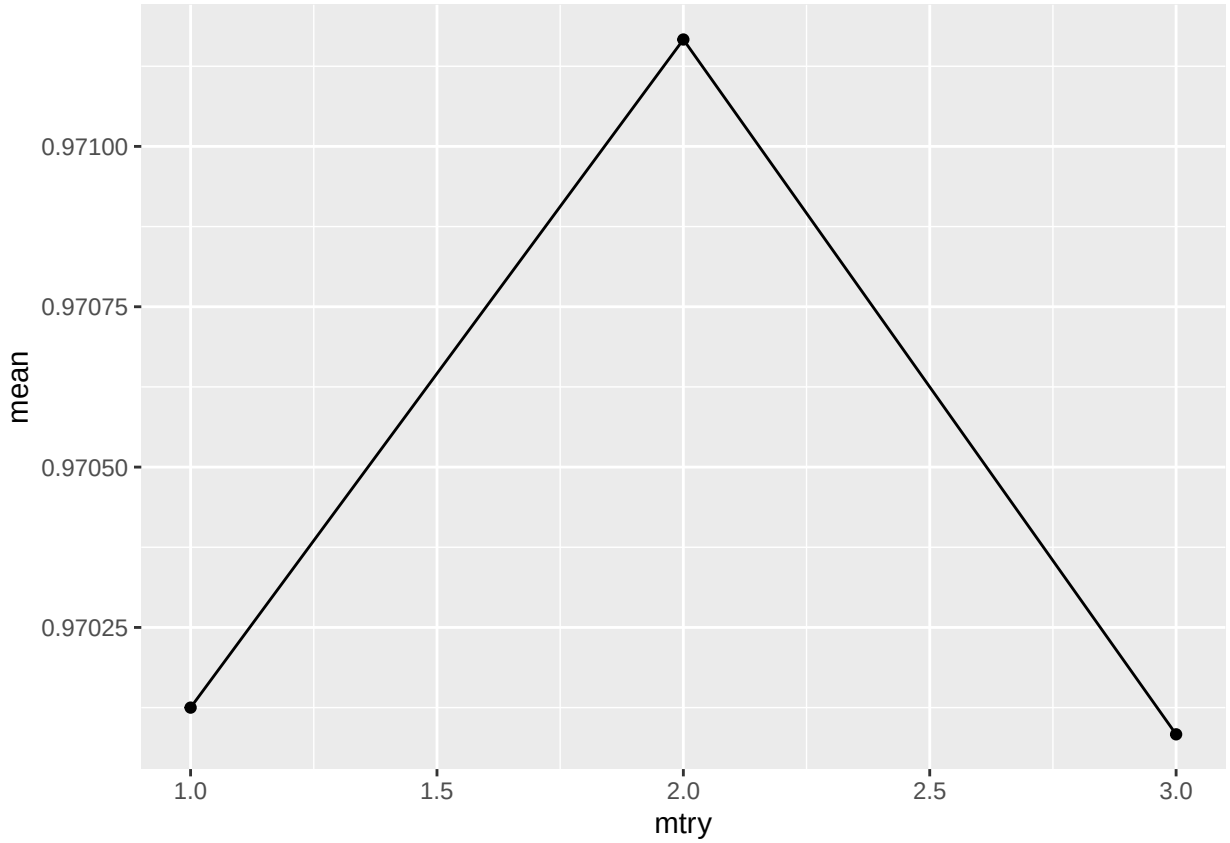
```
set.seed(437)
default_kfold <- vfold_cv(default_train, v = 5, repeats = 3)


n_predictors <- sum(rfC_recipe$var_info$role == "predictor")
manual_grid <- expand.grid(mtry = seq(1, n_predictors))
```

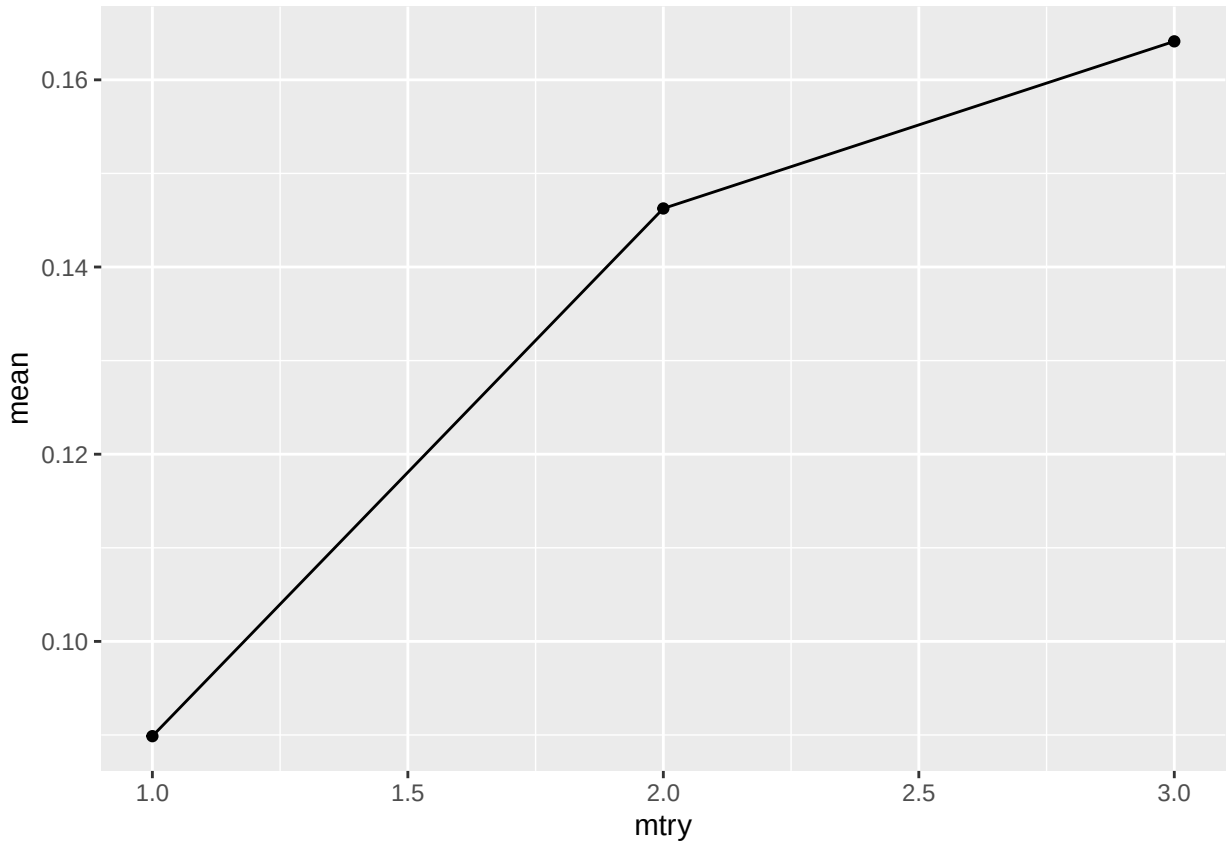
```
rfC_tune1 <- tune_grid(rfC_model,
                       rfC_recipe,
                       resamples = default_kfold,
                       metrics = metric_set(accuracy, mn_log_loss),
                       grid = manual_grid)
```

```
## Warning: package 'ranger' was built under R version 4.1.3
```

```
rfC_tune1 %>%
  collect_metrics() %>%
  filter(.metric == "accuracy") %>%
  ggplot(mapping = aes(x = mtry, y = mean)) + geom_point() + geom_line()
```




```
rfC_tune1 %>%
  collect_metrics() %>%
  filter(.metric == "mn_log_loss") %>%
  ggplot(mapping = aes(x = mtry, y = mean)) + geom_point() + geom_line()
```




```
rfC_best <- select_best(
  rfC_tune1,
  metric = "mn_log_loss",
  mtry
)
```

```
rfC_wflow_final <- finalize_workflow(rfC_wflow, parameters = rfC_best)

rfC_fit <- fit(rfC_wflow_final, data = default_train)
rfC_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_dummy()
##
## -- Model -----------------------------------------------------------------------
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~1L,      x), seed = ~437, importanc
##
## Type:                             Probability estimation
```

```
## Number of trees:                  500
## Sample size:                      8000
## Number of independent variables:  3
## Mtry:                             1
## Target node size:                 10
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.02263424
```

Part c (Code: 3 pts)

Fit a gradient-boosted trees model on the training set (using either `gbm` or `xgboost`). Use a learning rate of $\lambda = 0.01$ and an interaction depth of $d = 2$. You can either tune the number of trees or use a reasonably large number.

```
xgboostC_model <- boost_tree(mode = "classification", engine = "xgboost",
                             trees = tune(), tree_depth = 2,
                             learn_rate = 0.01)

xgboostC_wflow <- workflow() %>%
  add_model(xgboostC_model) %>%
  add_recipe(rfC_recipe)
```

```
set.seed(1486)
xgboostC_tune <- tune_grid(xgboostC_model,
                       rfC_recipe,
                       resamples = default_kfold,
                       metrics = metric_set(accuracy, mn_log_loss),
                       grid = grid_latin_hypercube(
                         trees(), size = 20)) # search over 20 possible combinations of the three parame
```

```
xgboostC_best <- select_by_one_std_err(
  xgboostC_tune,
  metric = "mn_log_loss", trees)
```

```
xgboostC_wflow_final <- finalize_workflow(xgboostC_wflow, parameters = xgboostC_best)

xgboostC_fit <- fit(xgboostC_wflow_final, data = default_train)
xgboostC_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_dummy()
##
## -- Model -----------------------------------------------------------------------
## ##### xgb.Booster
## raw: 517.1 Kb
## call:
##   xgboost::xgb.train(params = list(eta = 0.01, max_depth = 2, gamma = 0,
##     colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
##     subsample = 1), data = x$data, nrounds = 591L, watchlist = x$watchlist,
```

```
##     verbose = 0, nthread = 1, objective = "binary:logistic")
## params (as set within xgb.train):
##   eta = "0.01", max_depth = "2", gamma = "0", colsample_bytree = "1", colsample_bynode = "1", min_ch
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 3
## niter: 591
## nfeatures : 3
## evaluation_log:
##     iter training_logloss
##        1       0.68406250
##        2       0.67515761
## ---
##      590       0.07535177
##      591       0.07534010
```

Part d (Code: 3 pts)

Fit a neural network on the training set. Either follow the instructions in Textbook Exercise 10.10.7 or adapt the example code in the "Single Hidden Layer Neural Network" activity.

```
neuralnetC_model <- mlp(mode = "classification", engine = "keras",
                        hidden_units = tune(),
                        dropout = tune(),
                        epochs = 25,
                        activation = "relu") %>%
  set_args(seeds = c(1, 2, 3)) # we need to set 3 seeds
# let's tune the dropout parameter instead

neuralnetC_recipe <- recipe(
  default ~ student + balance + income,
  data = default_train
) %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors())
# no nominal predictors here so won't do anything

neuralnetC_wflow <- workflow() %>%
  add_model(neuralnetC_model) %>%
  add_recipe(neuralnetC_recipe)
```

```
extract_parameter_set_dials(neuralnetC_model) %>%
  pull("object")
```

```
## [[1]]
## # Hidden Units (quantitative)
## Range: [1, 10]
##
## [[2]]
## Dropout Rate (quantitative)
## Range: [0, 1)
```

```
set.seed(437)
defaultnn_kfold <- vfold_cv(default_train, v = 5, repeats = 1)


neuralnetC_tune <- tune_grid(neuralnetC_model,
                      neuralnetC_recipe,
                      resamples = defaultnn_kfold,
                      metrics = metric_set(mn_log_loss),
                      grid = grid_regular(hidden_units(range = c(16, 32)),
                                          dropout(range = c(0, 0.1)),
                                          levels = 2)
)
```

```
## Warning: package 'keras' was built under R version 4.1.3
```

```
## Warning: package 'magrittr' was built under R version 4.1.3
```

```
collect_metrics(neuralnetC_tune)
```

```
## # A tibble: 4 x 8
##   hidden_units dropout .metric     .estimator  mean     n std_err .config
##          <int>   <dbl> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
## 1           16     0   mn_log_loss binary     0.103     5  0.0135 Preprocessor1~
## 2           32     0   mn_log_loss binary     0.106     5  0.0144 Preprocessor1~
## 3           16     0.1 mn_log_loss binary     0.110     5  0.0150 Preprocessor1~
## 4           32     0.1 mn_log_loss binary     0.120     5  0.0203 Preprocessor1~
```

```
neuralnetC_best <- select_by_one_std_err(
  neuralnetC_tune,
  metric = "mn_log_loss",
  hidden_units, desc(dropout)
)
neuralnetC_best
```

```
## # A tibble: 1 x 10
##   hidden_units dropout .metric  .esti~1  mean     n std_err .config .best .bound
##          <int>   <dbl> <chr>    <chr>   <dbl> <int>   <dbl> <chr>   <dbl>  <dbl>
## 1           16     0.1 mn_log_~ binary  0.110     5  0.0150 Prepro~ 0.103  0.116
## # ... with abbreviated variable name 1: .estimator
```

```
neuralnetC_wflow_final <- finalize_workflow(neuralnetC_wflow,
                                            parameters = neuralnetC_best)
```

```
neuralnetC_fit <- fit(neuralnetC_wflow_final, data = default_train)
neuralnetC_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: mlp()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_YeoJohnson()
## * step_normalize()
## * step_dummy()
##
```

```
## -- Model ---------------------------------------------------------------------
## Model: "sequential_20"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_50 (Dense)                   (None, 16)                      64
##  dense_51 (Dense)                   (None, 16)                      272
##  dropout_10 (Dropout)               (None, 16)                      0
##  dense_52 (Dense)                   (None, 2)                       34
## =========================================================================
## Total params: 370
## Trainable params: 370
## Non-trainable params: 0
## _________________________________________________________________________
```

Part e (Code: 2 pts; Explanation: 2 pts)

For each of the models in parts (b)-(d), predict on the test set and obtain the confusion matrix. Which model is making the best predictions? Why?

```
rfC_predict <- augment(rfC_fit, new_data = default_test)

confusion_mat <- rfC_predict %>% conf_mat(default,.pred_class)
confusion_mat
```

```
##           Truth
## Prediction   No  Yes
##        No  1929   54
##        Yes    3   14
```

```
summary(confusion_mat)
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.972
##  2 kap                  binary         0.320
##  3 sens                 binary         0.998
##  4 spec                 binary         0.206
##  5 ppv                  binary         0.973
##  6 npv                  binary         0.824
##  7 mcc                  binary         0.403
##  8 j_index              binary         0.204
##  9 bal_accuracy         binary         0.602
## 10 detection_prevalence binary         0.992
## 11 precision            binary         0.973
## 12 recall               binary         0.998
## 13 f_meas               binary         0.985
```

```
xgboostC_predict <- augment(xgboostC_fit, new_data = default_test)

confusion_mat <- xgboostC_predict %>% conf_mat(default,.pred_class)
confusion_mat
```

```
##           Truth
## Prediction   No  Yes
##        No  1925   49
```

```
##        Yes    7   19
```

```
summary(confusion_mat)
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.972
##  2 kap                  binary         0.393
##  3 sens                 binary         0.996
##  4 spec                 binary         0.279
##  5 ppv                  binary         0.975
##  6 npv                  binary         0.731
##  7 mcc                  binary         0.441
##  8 j_index              binary         0.276
##  9 bal_accuracy         binary         0.638
## 10 detection_prevalence binary         0.987
## 11 precision            binary         0.975
## 12 recall               binary         0.996
## 13 f_meas               binary         0.986
```

```
predictions_neuralnetC <- broom::augment(neuralnetC_fit, new_data = default_test)
mn_log_loss(predictions_neuralnetC, truth = default, .pred_No,
            event_level = "first")
```

```
## # A tibble: 1 x 3
##   .metric     .estimator .estimate
##   <chr>       <chr>          <dbl>
## 1 mn_log_loss binary         0.128
```

```
confusion_mat <- predictions_neuralnetC %>% conf_mat(default,.pred_class)
confusion_mat
```

```
##           Truth
## Prediction   No  Yes
##        No  1925   50
##        Yes    7   18
```

```
summary(confusion_mat)
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.972
##  2 kap                  binary         0.376
##  3 sens                 binary         0.996
##  4 spec                 binary         0.265
##  5 ppv                  binary         0.975
##  6 npv                  binary         0.720
##  7 mcc                  binary         0.426
##  8 j_index              binary         0.261
##  9 bal_accuracy         binary         0.631
## 10 detection_prevalence binary         0.988
## 11 precision            binary         0.975
## 12 recall               binary         0.996
## 13 f_meas               binary         0.985
```

The neural net is making the best predictions because it has an accuracy of about 0.973. The gradient-boosted tree has a similar accuracy at 0.972.