STAT 4710: Final Exam

Name

Release Date: 12/15/22 at 9am; Due Date: 12/16/22 at 9pm

Contents

Instructions

Materials

The allowed materials are as stated on the Syllabus:

> "Students may consult all course materials, including course textbooks, for all assignments and assessments. For programming-based assignments (homeworks and exams), students may also consult the internet (e.g. Stack Overflow) for help with general programming tasks (e.g. how to add a dashed line to a plot). Students may not search the internet for help with specific questions or specific datasets on any homework or exam. In particular, students may not use solutions to problems that may be available online and/or from past iterations of the course."

Collaboration

The collaboration policy is as stated on the Syllabus:

> "Students are prohibited from collaborating on the quizzes or exams."

Writeup

Use this document as a starting point for your writeup, adding your R code using code chunks and adding your text answers using **bold text**. Consult the preparing reports guide for guidance on compilation, creation of figures and tables, and presentation quality. In particular, if the instructions ask you to "print a table", you should use `kable`. If the instructions ask you to "print a tibble", you should not use `kable` and instead print the tibble directly.

Programming

The `tidyverse` paradigm for data visualization, manipulation, and wrangling is required. No points will be awarded for code written in base R.

We will need to use the following `R` packages:

```
library(kableExtra)                    # for printing tables
library(ggcorrplot)                    # for correlation plots
library(randomForest)                  # for fitting random forests
library(gbm)                           # for fitting boosted models
library(keras)                         # for fitting deep learning models
library(stat471)                       # for class-specific functions
library(naniar)                        # for handling missing values
library(lubridate)                     # for dealing with dates
library(tidyverse)                     # for everything else
```

Grading

The point value for each problem sub-part is indicated. Additionally, the presentation quality of the solution for each problem (as exemplified by the guidelines in Section 4 of the preparing reports guide will be evaluated on a per-problem basis (e.g. in this homework, there are three problems). There are 100 points possible on this midterm, 85 of which are for correctness and 15 of which are for presentation.

Submission

Please compile your writeup to PDF and submit to Gradescope.

Non-Fatal Firearm-Related Injuries

Firearms are a major source of injury and death in the United States. The Firearm Injury Surveillance Study conducted by the Centers for Disease Control gathers information on nonfatal firearm-related injuries (i.e., injuries associated with powder-charged guns) and all nonfatal BB and pellet gun-related injuries from 1993 through 2013. The data, made available online by the Inter-university Consortium for Political and Social Research, are available to you as `firearm-data.csv`. Information is available on injury diagnosis, firearm type, use of drugs or alcohol, criminal incident, and locale of the incident. Demographic information includes age, sex, and race of the injured person. Descriptions of each variable in the data are available in `codebook.pdf`. *The primary analysis goal for this exam is to build predictive models for whether a victim of a nonfatal firearm-related injury is hospitalized.*

1 Wrangling (26 points for correctness, 3 points for presentation)

1.1 Import (1 point)

1. (1 point) Read the dataset into a tibble called `firearm_data_raw`. Print this tibble. At this stage, do not convert the character variables to factors.

1.2 Handling missing data and redundant variables (8 points)

1. (3 points) Some variables are missing in a substantial fraction of rows, and should therefore be removed from the data. A convenient function summarizing the proportion of rows for which each variable is missing is `miss_var_summary()` from the `naniar` package. Using `miss_var_summary()`, obtain a list of variables for which over 20% of rows are missing, storing it in a vector named `high_missing_vars`. Print this vector.

2. (2 points) Certain groups of variables carry similar information. For example, `BDYPT` is the "primary body part affected" while `BDYPTG_C` is the "primary body part affected - grouped." The latter contains the same information as the former, but is somewhat coarser. Having categorical variables with too many levels slows down predictive model training, so we might want to store information at a coarser level to avoid this. By consulting the list of variables in `codebook.pdf` (starting on p. 12), make a list of variables called `redundant_vars` for which grouped versions are available. Additionally, include all age-related variables that are not the numeric age in `redundant_vars`. (Hint: `redundant_vars` should have nine entries.)

3. (3 points) Below is a list of three more variables that are not useful for prediction:

```
# other variables to remove
other_vars_to_remove <- c("RCASE", "WT_C", "PSU")
```

   Create a tibble called `firearm_data_dropped` by removing the variables in `high_missing_vars`, `redundant_vars`, and `other_vars_to_remove` from `firearm_data_raw` and then dropping any rows containing missing values. Print `firearm_data_dropped`.

4. (3 points) In the steps above, we first removed some variables (columns), including those with large fractions of missing observations, and then removed observations (rows) with any missing variables. If we had reversed the order of these steps, would we have ended up with more, fewer, or the same number of observations in `firearm_data_dropped`? Why is the approach taken here more sensible?

1.3 Further cleaning (14 points)

1. (3 points) The four columns `FA_GSW`, `FA_NGSW`, `BB_GSW`, and `BB_NGSW` carry two pieces of information: whether the gun used was a firearm (`FA`) or a BB gun (`BB`) and whether the wound is a gunshot wound (`GSW`) or another type of wound (`NGSW`). Add two new columns to `firearm_data_dropped` called `GUN_TYPE` and `WOUND_TYPE`, which take values `Firearm` or `BB gun` and `Gunshot wound` or `Other wound`, respectively. Therefore, `GUN_TYPE` and `WOUND_TYPE` should each have two unique values. Then, remove the columns `FA_GSW`, `FA_NGSW`, `BB_GSW`, and `BB_NGSW` from `firearm_data_dropped`. Call the resulting tibble `firearm_data`.

2. (3 points) The categorical features `DIAG` and `ONTHEJOB` have levels that are somewhat rare. This can create issues, such as certain levels not showing up in the training data at all. To avoid this, first create a vector `rare_diagnoses` of levels of `DIAG` that have at most 10 observations in the data. Print `rare_diagnoses`. Then, modify the `DIAG` variable to combine all individual categories in `rare_diagnoses` into one new category called `other` in `DIAG`, as opposed to having unique variables for each. For `ONTHEJOB`, absorb the category `(3) Act duty military` into the category `(1) Yes`, so that `ONTHEJOB` has just unique values: `(1) Yes` and `(2) No`.

3. (2 points) The variables `INJDT_C` and `TRDATE` represent dates when the injury and treatment occurred, respectively. While these dates themselves may not have much predictive power, it is possible that the number of days a person waited to get treated after their injury might. Using the package `lubridate`, create a new variable called `DAYS_TO_TREATMENT` representing the number of days after the injury that the treatment occurred. For example, a person who got treated the same day as their injury would have `DAYS_TO_TREATMENT = 0`. Remove `INJDT_C` and `TRDATE` from the tibble. (Hint: For two date-time objects `date1` and `date2`, the code `as.numeric(date2 - date1, "days")` calculates the number of days elapsed from `date1` to `date2`. See the `lubridate` cheat sheet for how to convert a string to a date-time object.)

4. (2 points) How any people have `DAYS_TO_TREATMENT < 0`? Remove these from the dataset, as these are likely data entry errors.

5. (2 points) All character features need to be in the form of factors for `gbm()` to work properly. Therefore, convert all character features to factors. (Hint: Use `mutate_if()`.)

6. (2 points) Create the response variable `HOSP`, a numeric variable taking the value 1 if the emergency department disposition of the case (`DISP`) is equal to `(4) Hospitalized`, and 0 otherwise. Remove the variable `DISP` from the tibble. Print the resulting `firearm_data` tibble.

2 Exploration (19 points for correctness, 5 points for presentation)

NOTE: If you were unable to complete the data wrangling portion of the exam, please post privately on Ed Discussion and the teaching staff will add the correctly cleaned `firearm_data` to your RStudio Cloud project. You will still receive points for partial progress on the data wrangling portion. *However, 5 points will be deducted from your grade.*

2.1 Basic data information (5 points)

1. (3 points) How many observations are in `firearm_data`? What percentage of observations were removed during the data cleaning process? How many features are in `firearm_data`? For the latter question, please exclude the response variable and count each categorical feature as one feature regardless of the number of levels.

2. (2 points) What percentage of the individuals in these data were hospitalized? Comment on the degree of class imbalance in these data.

2.2 Feature relationships (10 points)

1. (4 points) Construct and plot the correlation matrix for the features using `mixed_cor()` from the `stat471` package in conjunction with `ggcorrplot()` from the `ggcorrplot` package, specifying `hc.order = TRUE` and `tl.cex = 7` to reorder the features and resize the axis labels. Comment on three trends you see in the correlation matrix, and how you would explain them. (Note: This code may take 10 seconds or more to run. After you have run the code, run the `save` command provided for you to save the correlation matrix to file. Since this code chunk has `eval = FALSE`, it will not be re-run during compilation. Instead, the saved correlation matrix will be read from file using the next code chunk, in which you will need to uncomment the `load` command.)

```
# create correlation matrix
# TBD

# save the correlation matrix to file
save(corr_mat, file = "corr_mat.rda")
```

```
# plot the correlation matrix
# TBD
```

2. (3 points) Create a line plot to track the number of gunshot wounds and non-gunshot wounds from firearms (excluding BB guns) as a function of year. Use faceting to stack the panels for gunshot and non-gunshot wounds on top of one another. Discuss the trends that emerge, and provide a possible historical explanation, given that Bill Clinton was famously tough on crime (including guns) in the 1990s, while George W. Bush (elected in 2000) had more relaxed gun control policies.

3. (3 points) Create a histogram of `DAYS_TO_TREATMENT`, restricting the data first to `DAYS_TO_TREATMENT <= 10`. Choose histogram bins so that each bin represents a unique value of `DAYS_TO_TREATMENT`, and

put the y axis on a logarithmic scale. Discuss the trends that emerge, and how you would explain them. What percentage of individuals sought treatment on the same day as their injury?

2.3 Feature-response relationships (4 points)

1. (2 points) Create a bar chart showing how the hospitalization rate depends on the primary body part affected by the injury, restricting your graph to `DAYS_TO_TREATMENT <= 5` for better visualization. Discuss and interpret your findings.
2. (2 points) Create a bar chart showing how the hospitalization rate depends on the number of days between injury and treatment. Discuss and interpret your findings.

3 Modeling (24 points for correctness, 3 points for presentation)

3.1 Data split (2 points)

1. Split your data into training and test sets based on the `YEAR` variable. Define the training set as all observations in years up to and including 1999, and the test set as all observations in years starting from 2000. What percentage of observations are in the training test? What are one benefit and one drawback of having the training data be a smaller fraction of the whole dataset than usual?

3.2 Baseline models (2 points)

We will use two models as baselines: the logistic regression model containing only an intercept and the logistic regression model containing all features.

1. (2 points) Add code to train these models below; store the resulting fit objects in `int_only_fit` and `log_reg_fit`. (Note that the enclosing `system.time()` commands are to time the code for a comparison in a later problem. When you are done training these models, run the `save` command provided for you. Note that the code chunk header includes `eval = FALSE`, which will prevent the code chunk from being re-run during compilation.)

```
int_only_time <- system.time({
  # add code to train intercept-only model here
  # TBD
})["elapsed"]

log_reg_time <- system.time({
  # add code to train logistic regression model here
  # TBD
})["elapsed"]

save(list = c("int_only_time", "int_only_fit", "log_reg_time", "log_reg_fit"),
     file = "baseline_models.rda")
```

3.3 Tree-based models (8 points)

3.3.1 Random forests (4 points)

1. (2 points) Train random forest models with 250 trees each, for `mtry = 2, 4, 8`. Store the fitted models in `rf_fit_2`, `rf_fit_4`, and `rf_fit_8`. (This code may take up to about a minute to run. When you are done training these models, run the `save` command provided for you. Note that the code chunk header includes `eval = FALSE`, which will prevent the code chunk from being re-run during compilation.)

```
# random forest
B <- 250
rf_time <- system.time({
  set.seed(1)
  # train random forest with mtry = 2
  # TBD

  set.seed(1)
  # train random forest with mtry = 4
  # TBD

  set.seed(1)
  # train random forest with mtry = 8
  # TBD

})["elapsed"]

save(list = c("rf_time", "rf_fit_2", "rf_fit_4", "rf_fit_8"),
     file = "rf_models.rda")
```

2. (2 points) Plot the out-of-bag errors for each of the random forest fits as a function of the number of trees. Out of the three random forests, which gives the lowest OOB error? Assign the corresponding fit object to the variable `rf_fit`.

3.3.2 Boosting (4 points)

1. (2 points) Train boosting models with up to 1000 trees each, for interaction depths 1, 2, and 3. Use a shrinkage parameter of 0.1, and a 20% validation set approach rather than a cross-validation approach to save time. Store the fitted models in `gbm_fit_1`, `gbm_fit_2`, and `gbm_fit_3`. (This code may take up to about a minute to run. Hint: You'll need to set `cv.folds` and `train.fraction` in `gbm()` appropriately to implement the validation set approach. When you are done training these models, run the `save` command provided for you. Note that the code chunk header includes `eval = FALSE`, which will prevent the code chunk from being re-run during compilation.)

```
gbm_time <- system.time({
  set.seed(1)
  # train boosting model with interaction depth = 1
  # TBD

  set.seed(1)
  # train boosting model with interaction depth = 2
  # TBD

  set.seed(1)
  # train boosting model with interaction depth = 3
  # TBD

})["elapsed"]

save(list = c("gbm_time", "gbm_fit_1", "gbm_fit_2", "gbm_fit_3"),
     file = "gbm_models.rda")
```

2. (2 points) Store the validation errors of all three boosting models in a tibble called `valid_errors`, with columns `ntree`, `valid_err`, and `depth`. Based on this tibble, plot the validation errors for each of the boosting model fits as a function of the number of trees. Out of the three boosting models, which gives

the lowest validation error, and for which number of trees? Determine this by appropriately sorting `valid_errors` and printing the first row (no need to use `kable`). Assign the corresponding fit object to the variable `gbm_fit`. (Hint: The validation error is contained in a field of the `gbm` fit object called `valid.error`.)

3.4 Neural network models (12 points)

We cannot specify formula objects to `keras`'s `fit()` function. Instead, we must specify a feature matrix `x` and a response vector `y`. The feature matrix needs to have all the factor variables expanded as dummy variables, and columns normalized so that they have maximum value 1. The response vector needs to be "one-hot" coded. These transformations are given to you (uncomment the code below once `firearm_train` is created):

```
# x_train <- model.matrix(factor(HOSP) ~ . - 1,
#                         data = firearm_train %>%
#                           mutate_if(is.numeric, function(col)(col/max(col))))
# y_train <- to_categorical(y = firearm_train$HOSP, num_classes = 2)
```

1. (2 points) Calculate the number of features (after dummy coding) by extracting the appropriate dimension of `x_train` defined above. Store this number in a variable called `input_length`. What is this number of features? By comparing it to the number of features before dummy coding (which you calculated before), how many levels (on average) does each categorical variable have? You can ignore the presence of non-categorical variables for this calculation.

2. (2 points) Define three neural network models, each with one hidden layer, with dropout rate 0.5 and ReLU activation. The only difference between the models is the number of units in the hidden layer: try 32, 64, and 128 units. Compile the models to use cross-entropy loss and RMSprop optimizers. Call the models `model_nn_32`, `model_nn_64`, and `model_nn_128`. Note that the definition and compilation of a model can be constructed as one continuous sequence of pipes. (Hint: Specify the input shape in the first layer via `input_shape = c(input_length)`.)

3. (3 points) Print a summary of the largest of the three models. How many total weights does it have? How many weights (including bias) are associated with each neuron at the output layer, and why?

4. (3 points) Train each of the three neural network models for 10 epochs, batch size 128, using 20% of the data for validation. How many total SGD steps were taken when training one of these models, and how do you arrive at that number? (When you are done training these models, run the `save` commands provided for you. Note that the code chunk header includes `eval = FALSE`, which will prevent the code chunk from being re-run during compilation.)

```
nn_time <- system.time({
  set.seed(1)
  # train neural network model with 32 hidden units
  # TBD

  set.seed(1)
  # train neural network model with 64 hidden units
  # TBD

  set.seed(1)
  # train neural network model with 128 hidden units
  # TBD

})["elapsed"]

# save the run time
save(list = c("nn_time"), file = "nn_time.rda")
```

```
# save the models
save_model_hdf5(model_nn_32, "model_nn_32.h5")
save_model_hdf5(model_nn_64, "model_nn_64.h5")
save_model_hdf5(model_nn_128, "model_nn_128.h5")
```

5. (2 points) Compute the validation misclassification error for each of the three trained models, and print these in a table. Find the model with the lowest validation error, and assign it to `model_nn`.

4 Evaluation (8 points for correctness, 1 point for presentation)

1. (2 points) Compute the test misclassification error for all five of the predictive models fit in the previous section (the two baseline models `int_only_fit` and `log_reg_fit`, the best random forest model `rf_fit`, the best boosting model `gbm_fit`, and the best neural network model `model_nn`).

```
# intercept-only
# TBD

# logistic regression (you may get a warning here, which you can ignore)
# TBD

# random forest
# TBD

# boosting
# TBD

# neural network model (uncomment code below when firearm_test is created)
# x_test <- model.matrix(factor(HOSP) ~ .,    # preprocess features as before
#                        data = firearm_test %>%
#                          mutate_if(is.numeric, function(col)(col/max(col))))
# TBD
```

2. (2 points) Print a table displaying the five methods compared, their runtimes for training and tuning, and their test misclassification error to three significant digits. Discuss the relative performances and computational speeds of the methods, and identify the best-performing method and its misclassification rate. Which method strikes a reasonable balance between predictive performance and speed?

3. (4 points) Discuss the contribution of interactions to predictive performance on this data set. Do methods incorporating interactions perform better than methods that do not? Why might this be the case? Give a concrete example (though this example need not be supported by evidence from the data.)

5 Interpretation (8 points for correctness, 3 points for presentation)

1. (3 points) Create two tables of the five most important features for the chosen random forest and boosting models as well as their Gini-based feature importance measures, one table for each model. Points will not be deducted for not cleaning up the variable names in the tables. (Hint: For random forests, the variable importances are stored in the `importance` field of the fit object.)

2. (5 points) Which variables appear in the top five for both boosting and random forests? Produce the boosting partial dependence plots for these variables. Discuss each of these plots, and whether the trends displayed make sense to you and why. Why might one or more of these plots not make sense? Points will not be deducted for not cleaning up the variable names in the plots. (Hint: Increase the

figure width to decrease labels overlap. A small degree of label overlap in these plots is hard to avoid, and will not be penalized.)