Data Science for Public Policy

Aaron R. Williams - Georgetown University

PPOL 670 | Assignment 06
Supervised Machine Learning

Due Date: Wednesday, March 30th at 6:29 PM.

Deliverable:

1. An .Rmd file with your R code
2. The resulting project .html file
3. The URL of a private Git repository.

Note: You must use a private Git repository for this assignment. Since you are all working on the same assignment, and GitHub repositories are public by default, this is an academic integrity issue. Thus, you absolutely must **only push your work to a private repository**. Please reach out with any questions or concerns.

Grading Rubric

Please show your work! It is easier to give partial credit when a computational mistake is made if formulas are fully specified and substitutions are correctly made.

- [0.5 point] Create a private, well-managed GitHub repository, including an appropriate `.gitignore`, and an informative `README.md` file. You should add at least one commit for each question. Points will be reduced for infrequent commits or unclear commit messages.
- [0.5 point] Write a clean and well-composed `.Rmd file`, including separate named code chunks for each task required below. The resulting `.html` file should show code and results, but hide unnecessary warnings and messages.
- [1 point] Exercise 01
- [1 point] Exercise 02
- [1 point] Exercise 03
- [1 point] Exercise 04
- [2 points] Exercise 05

Points: 7 points

Plagiarism on homework or projects will be dealt with to the full extent allowed by Georgetown policy (see http://honorcouncil.georgetown.edu).

Setup

Create a new folder with a new R project (`.Rproj`) and R Markdown file (`.Rmd`). Then create a new **private GitHub repository**. Add awunderground and ncstabile17 (section 01) or alenastern and joshrrosen (section 02) and your partner (if working with a partner) to your repository.

Exercise 01 (1 point)

Calculate the mean square error (MSE), root mean square error (RMSE), and mean absolute error (MAE) for the following data "by hand". You can add scanned answers with `knitr::include_graphics()` or you can use inline LaTeX equations with R Markdown. How do RMSE and MAE handle outlier predictions differently?

true_value	predicted_value
1	2
2	2
3	1
4	8
5	4

Exercise 02 (1 point)

The following data come from a binary classification problem.

true_value	predicted_value
0	0
0	0
0	1
0	0
0	0
1	1
1	0
1	0
1	1
1	1

Using the above data, calculate the following "by hand" and show your work:

1. A confusion matrix
2. Accuracy
3. Precision
4. Recall/Sensitivity

You can scan your paper answer and add it with `knitr::include_graphics()` or you can use Markdown tables. Do not use `yardstick::conf_mat()` or `caret::confusionMatrix()`.

Exercise 03 (1 point)

The following data come from a multiclass classification problem.

true_value	predicted_value
compliance	compliance
compliance	compliance
compliance	compliance
compliance	risk of noncompliance
compliance	compliance
compliance	noncompliance
compliance	compliance
compliance	compliance
compliance	compliance
compliance	risk of noncompliance
risk of noncompliance	risk of noncompliance
risk of noncompliance	noncompliance
noncompliance	noncompliance
noncompliance	compliance
noncompliance	noncompliance

Using the above data, calculate the following "by hand" and show your work:

1. A confusion matrix
2. Accuracy
3. Misclassification rate

You can scan your paper answer and add it with `knitr::include_graphics()` or you can use Markdown tables. Do not use `yardstick::conf_mat()` or `caret::confusionMatrix()`.

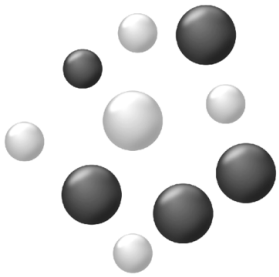
Exercise 04 (1 point)

Consider a population where it is known that 0.49 of observations have a value of `0` and 0.51 of observations have a value of `1`. Approximately what accuracy can be achieved by simply guessing the same value for all observations? What number should you predict?

Consider a population where it is known that 0.99 of observations have a value of `0` and 0.01 of observations have a value of `1`. Approximately what accuracy can be achieved by simply guessing the same value for all observations? What number should you predict?

Explain why it is important to consider context when comparing calculated accuracy in different supervised machine learning tasks?

Exercise 05 (2 points)



`marbles.csv` contains a new simple random sample from the population of marbles that generated the first machine learning example in class #7.

1. Divide the marbles data set into a training set with 80% of observations and a testing set 20% of observations. Set the seed to `20200229` before sampling.
2. Use `count()` and `library(ggplot2)` to develop and justify an intuitive/mental model for predicting black marbles.
3. Construct a custom function that takes a vector of sizes and returns a vector of predicted colors. Apply it to the testing data. The R4DS chapter on functions is helpful.
4. Construct a custom function that takes `y` and `y_hat` that returns calculated accuracy and a confusion matrix. Until now, we have only returned one object from a custom function. Use `list()` inside of `return()` to return more than one object. Apply it to the data from part 3. Do not use `yardstick::conf_mat()` or `caret::confusionMatrix()`.
5. Using the same testing and training data, estimate a decision tree/CART model with functions from `library(parnsip)`. Use the "rpart" engine.
6. Does the decision tree/CART model generate the same predictions on the testing data as the model from part 2? Why or why not?

Stretch (5 points)

Part 01 (2 points)

The following example includes a simulated data set about the presence of rat burrows in alleys and proximity to the nearest jumbo slice pizza restaurant. (rats.R is on Canvas)

- `rat_burrow` 1 if burrow present, 0 if no burrow present
- `pizza_proximity` Proximity in miles from the alley to the nearest jumbo slice pizza restaurant

The goal is to estimate a K-Nearest Neighbors model "by hand" with three different Ks where your outcome variable is `rat_burrow` and your predictor variable is `pizza_proximity`. "By hand" means using a paper and pencil or writing code from scratch that applies the algorithm. Do not use `library(tidymodels)`, `library(caret)`, or any other machine learning packages. Run the following code chunk to create three resamples of the data.

Note: *running the code out-of-order will change the observations included in reach resample. Run the entire code chunk for consistent results.*

```
set.seed(20200302)

# input the data
rats <- tribble(
  ~rat_burrow, ~pizza_proximity,
  1, 0.01,
  1, 0.05,
  1, 0.08,
  0, 0.1,
  0, 0.12,
  1, 0.2,
  1, 0.3,
  1, 0.5,
  1, 0.75,
  0, 0.9,
  1, 1,
  0, 1.2,
  0, 2.2,
  0, 2.3,
  0, 2.5,
  1, 3,
  0, 3.5,
  0, 4,
  0, 5,
  0, 7
) %>%
  mutate(rat_burrow = factor(rat_burrow))

# split into training and testing data
split <- initial_split(rats, prop = 0.75)
rats_training <- training(split)
rats_testing <- testing(split)

rats_k1 <- vfold_cv(data = rats_training,
                    v = 3)
```

```
rats_k3 <- vfold_cv(data = rats_training,
                    v = 3)

rats_kn <- vfold_cv(data = rats_training,
                    v = 3)
```

Extract the analysis data and assessment data from the first resample in `rats_k1`, `rats_k3`, and `rats_kn` with `analysis()` and `assessment()`. **Hint:** You can access the first resample for the first problem with `rats_k1$splits[[1]]`. The observations should slightly differ in each resample.

- Calculate `y_hat` for the assessment data "by hand" in the first resample of `rats_k1` with KNN and $k = 1$.
- Calculate `y_hat` for the assessment data "by hand" in the first resample of `rats_k3` with KNN and $k = 3$.
- Calculate `y_hat` for the assessment data "by hand" in the first resample of `rats_kn` with KNN and $k = n$.

Note: Only make the calculations for the first resample. This is to save time!

You can write non-`library(tidymodels)` code or arithmetic to come up with $\hat{y}$. In each case, add `y_hat` to the assessment data using `bind_cols()`. Include the data frame in your R Markdown document using `knitr::kable()`. Calculate accuracy and a confusion matrix using your function from the marbles exercise.

Which model was easiest to estimate computationally and why? Which model was toughest to estimate computationally and why?

Part 02 (1.5 points)

The algorithm that creates decision trees and regression trees picks the split in a predictor that minimizes the heterogeneity of the resulting nodes. Heterogeneity is measured by mean square error for regression trees and gini index for classification trees. Lower values for the gini index mean the algorithm is successfully separating the classes with the split.

Consider the function `calc_mse()` in `mse-example.R` that was shared in class. Write a similar function called `calc_gini_2()` with arguments `left` and `right` that calculates the gini index (also known as gini impurity) for splits in a decision tree model for binary classification. You can assume that the values of the outcome variable will be dummy encoded (`0` or `1`).

Let k be the k^{th} class and K be the number of classes, then

$$\text{Gini index}_{left} = 1 - \sum_{k=1}^{K} (p_{k,left})^2$$

$$\text{Gini index}_{right} = 1 - \sum_{k=1}^{K} (p_{k,right})^2$$

$$\text{Gini index}_{total} = \frac{|S_{left}|}{|S|}\text{Gini index}_{left} + \frac{|S_{right}|}{|S|}\text{Gini index}_{right}$$

where $|S|$ is the number of observations, $|S_{left}|$ is the number of observations in the left node, and $|S_{right}|$ is the number of observations in the right node.

In this case, $k \in \{1, 2\}$ and $K = 2$ since we have a binary classification problem with `0` and `1` as the only outcomes.

Note, the maximum gini index in binary classification for a node is 0.5.

Your function should return results identical to the following:

```
calc_gini_2(left = c(0, 0, 0), right = c(1, 1, 1))
```

```
## $gini_left
## [1] 0
##
## $gini_right
## [1] 0
##
## $gini_combined
## [1] 0
```

```
calc_gini_2(left = c(0, 0, 1, 1), right = c(0, 0, 1, 1))
```

```
## $gini_left
## [1] 0.5
##
## $gini_right
## [1] 0.5
##
## $gini_combined
## [1] 0.5
```

```
dt_analysis <- analysis(rats_k1$splits[[3]]) %>%
  arrange(pizza_proximity)

calc_gini_2(left = dt_analysis$rat_burrow[1:5], right = dt_analysis$rat_burrow[6:10])
```

```
## $gini_left
## [1] 0.48
##
## $gini_right
## [1] 0
##
## $gini_combined
## [1] 0.24
```

Finally, run the following code to evaluate all of the non-empty splits, or all possible splits of the data between the right and left node where there is at least one observation in each node. What value of `pizza_proximity` is a good value to split on for a decision tree based on this small analysis data set? You should state an explicit value like `pizza_proximity = 8.9`.

```
map_dbl(
  .x = 1:9,
  .f = ~calc_gini_2(
    left = dt_analysis$rat_burrow[1:.x],
    right = dt_analysis$rat_burrow[(.x + 1):10]
  )$gini_combined
)
```

Part 03 (1.5 points)

Write a new function called `calc_gini_k()` with arguments `left` and `right` that calculates the gini index for splits in a decision tree model for multi-class classification.

The formula for gini index is the same as above except now k can be any positive integer. Note that the maximum possible value of the gini index increases with additional classes in multi-class classification.

Hint: `prop.table(table(x))` is a quick way to calculate proportions of classes from a vector.

Test to ensure that you get the same results as with `calc_gini_2()`:

```
calc_gini_k(left = c(0, 0, 0), right = c(1, 1, 1))
```

```
## $gini_left
## [1] 0
##
## $gini_right
## [1] 0
##
## $gini_combined
## [1] 0
```

```
calc_gini_k(left = c(0, 0, 1, 1), right = c(0, 0, 1, 1))
```

```
## $gini_left
## [1] 0.5
##
## $gini_right
## [1] 0.5
##
## $gini_combined
## [1] 0.5
```

```
calc_gini_k(left = dt_analysis$rat_burrow[1:5], right = dt_analysis$rat_burrow[6:10])
```

```
## $gini_left
## [1] 0.48
##
## $gini_right
## [1] 0
##
## $gini_combined
## [1] 0.24
```

Finally, use `calc_gini_k()` to evaluate all potential non-empty splits with `map_dbl()` (like above) for the following data where `x1` is the predictor and `y` is the outcome.

```
set.seed(20220318)

data <- tibble(
  x1 = runif(100, min = 0, max = 1),
  x2 = runif(100, min = 0, max = 1),
  x1_prob = runif(100, min = 0, max = 1),
  x2_prob = runif(100, min = 0, max = 1)
) %>%
```

```
  mutate(
    y = case_when(
      x1_prob > x1 & x2_prob > x2 ~ "a",
      x1_prob < x1 & x2_prob > x2 ~ "b",
      x1_prob > x1 & x2_prob < x2 ~ "c",
      x1_prob < x1 & x2_prob < x2 ~ "d",
    )
  ) %>%
  mutate(y = factor(y)) %>%
  select(x1, y)
```

For reference, here one potential split:

```
calc_gini_k(left = data$y[1:5], right = data$y[6:100])
```

```
## $gini_left
## [1] 0.56
##
## $gini_right
## [1] 0.7445983
##
## $gini_combined
## [1] 0.7353684
```