Practicum 1

Josiah Parry

2020-01-02

(0 pts) Download the data set Glass Identification Database along with its explanation. Note that the data file does not contain header names; you may wish to add those. The description of each column can be found in the data set explanation. This assignment must be completed within an R Markdown Notebook.

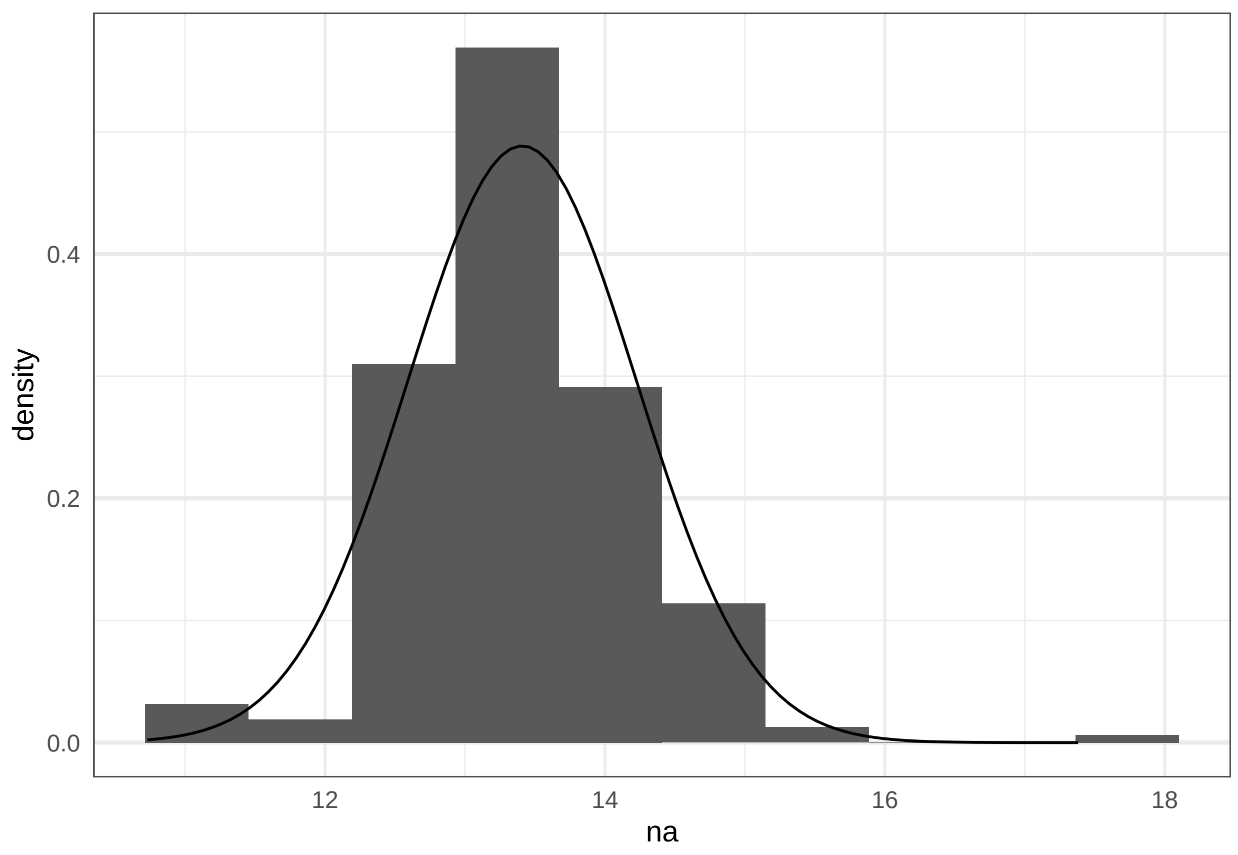
(0 pts) Explore the data set as you see fit and that allows you to get a sense of the data and get comfortable with it.

```r
library(tidyverse)
library(rsample)
library(recipes)
colnames <- c("id", "ri", "na", "mg", "al", "si", "k", "ca", "ba", "fe", "type")

glass <- read_csv("data/glass.data", col_names = colnames)

skimr::skim_to_list(glass) %>%
  pluck(1) %>%
  knitr::kable()
```

variable	missing	complete	n	mean	sd	p0	p25	p50	p75	p100	hist
al	0	214	214	1.44	0.5	0.29	1.19	1.36	1.63	3.5	
ba	0	214	214	0.18	0.5	0	0	0	0	3.15	
ca	0	214	214	8.96	1.42	5.43	8.24	8.6	9.17	16.19	
fe	0	214	214	0.057	0.097	0	0	0	0.1	0.51	
id	0	214	214	107.5	61.92	1	54.25	107.5	160.75	214	
k	0	214	214	0.5	0.65	0	0.12	0.56	0.61	6.21	
mg	0	214	214	2.68	1.44	0	2.11	3.48	3.6	4.49	
na	0	214	214	13.41	0.82	10.73	12.91	13.3	13.83	17.38	
ri	0	214	214	1.52	0.003	1.51	1.52	1.52	1.52	1.53	
si	0	214	214	72.65	0.77	69.81	72.28	72.79	73.09	75.41	
type	0	214	214	2.78	2.1	1	1	2	3	7	

(5 pts) Create a histogram of the Na column and overlay a normal curve; visually determine whether the data is normally distributed. You may use the code from this tutorial.

```r
# plotting hist over fr
ggplot(glass, aes(na)) +
  geom_histogram(aes(y = ..density..), bins = 10) +
  stat_function(fun = dnorm,
                args = list(mean = mean(glass$na),
                            sd = sd(glass$na)))
```



> (5 pts) Does the k-NN algorithm require normally distributed data or is it a non-parametric method? Comment on your findings.

k-nearest neighbors is a non parametric model that does not require normally distributed data. There is only one hyper-parameter in this model and that is k. We determine k and no parameters are discovered. One should note that distances can be affected by the magnitude of the variables and such should rescaled in some manner.

> (10 pts) After removing the ID column (column 1), normalize the numeric columns, except the last one, using z-score standardization. The last column is the glass type and so it is excluded.

> (10 pts) The data set is sorted, so creating a validation data set requires random selection of elements. Create a stratified sample where you randomly select 40% of each of the cases for each glass type to be part of the validation data set. The remaining cases will form the training data set.

knn is not a trained model. This question is misleading. There is no "validation" of a knn model. A knn model holds all data in memory and is the reason why it is not recommended at large scale. One must recalculate distances with each new observation.

In the training recipe we keep both the `type` and `id` columns for later use and validation.

```r
# stratified sample
init_split <- initial_split(
  glass,
  prop = .6,
  strata = type
)

# extract the training set
train_df <- training(init_split)

# prepping recipe
train_rec <- recipe(type ~., data = train_df) %>%
  # center and scale - i.e. "z-score normalization"
  recipes::step_center(all_numeric(), -type, -id) %>%
  step_scale(all_numeric(), -type, -id) %>%
  prep()
```

> (20 pts) Implement the k-NN algorithm in R (do not use an implementation of k-NN from a package) and use your algorithm with a k=6 to predict the glass type for the following two cases:

First, I create a function that calculates the euclidian distance for all observations.

```r
tidy_dist <- function(df, id_col) {

  mdist <- select(df, -{{ id_col }}) %>%
    dist(diag = TRUE, upper =FALSE) %>%
    as.matrix(labels = TRUE)


  ret <- as.data.frame(mdist) %>%
    setNames(pull(df, id)) %>%
    mutate(id = pull(df, id)) %>%
    gather(item2, distance, -id) %>%
    as_tibble()

  # remove upper triangle
  ret[!upper.tri(mdist),] %>%
    # remove diag
    filter(distance != 0)

}
```

Next, we need to apply the normalization steps to the data.

```r
baked_train <- bake(train_rec, train_df)
```

Then I calculate distances for all observations using this new function.

```r
glass_dist <- tidy_dist(baked_train, id)
```

Then, to identify which type each observation actually belongs to I join it back to the original data set using the `id` column. I select only the `id` and `type` columns for simplicity and data reduction. Following, I identify the 6 observations with the smalled euclidian distance then assign a predicted class based on majority vote.

```r
# define mode function
mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

```r
preds <- left_join(glass_dist,
          select(glass, id, type)) %>%
  group_by(item2) %>%
  top_n(-6, distance) %>%
  summarise(class = mode(type)) %>%
  mutate(item2 = as.integer(item2)) %>%
  left_join(select(glass, id, type),
            by = c("item2" = "id"))
```

```
## Joining, by = "id"
```

I visualize the confusion matrix with some help from `janitor`.

```r
janitor::tabyl(preds, class, type)
```

```
##  class  1  2 3 5 6  7
##      1 23  0 0 0 0  0
##      2 12 44 0 0 0  0
##      3  0  8 8 0 0  0
##      5  0  1 0 3 0  0
##      6  0  0 3 4 2  0
##      7  0  0 0 3 3 15
```

I calculate the accuracy below.

```r
mean(preds$class == preds$type)
```

```
## [1] 0.7364341
```

> Use the whole normalized data set for this; not just the training data set. Note that you need to normalize the values of the new cases the same way as you normalized the original data.

In order to make predictions with the below observations I create a function to do so.

```r
classify_type <- function(new_data) {

  bind_rows(
    # pre-process new data
    bake(train_rec, new_data),
    # pre-process glass
    bake(train_rec, glass)
  ) %>%
    # calc dist
    tidy_dist(id) %>%
    # join glass back on for type
    left_join(select(glass, id, type), by = "id") %>%
    # group for counts
    group_by(item2) %>%
    # identify 6 neighbors
    top_n(-6, distance)  %>%
    # identify the pred clasz
    summarise(class = mode(type)) %>%
    # identify only the new dat
    filter(item2 == 999) %>%
    # pull out the underlying vector
    pull(class)

}
```

> RI = 1.51721 | 12.53 | 3.48 | 1.39 | 73.39 | 0.60 | 8.55 | 0.00 | Fe = 0.08 RI = 1.4893 | 12.71 | 1.85 | 1.81 | 72.62 | 0.52 | 10.01 | 0.00 | Fe = 0.03

I create a new data frame with the above data. I assign the `id` value to `999` so I can identify these later.

```r
new_dat <- tribble(
  ~"ri", ~"na", ~"mg", ~"al", ~"si", ~"k", ~"ca", ~"ba", ~"fe",
  1.51721, 12.53, 3.48, 1.39, 73.39, 0.60, 8.55, 0.00, 0.08,
  1.4893 , 12.71 , 1.85 , 1.81 , 72.62 , 0.52 , 10.01 , 0.00 ,0.03
) %>%
  mutate(id = 999)
```

I create the predictions using purrr. We must iterate because if we add other observations it creates an inacurate distance measure for the single observation.

```r
map_dbl(1:2, ~{
  classify_type(
    slice(new_dat, .x))
  }
)
```

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

> (10 pts) Apply the knn function from the class package with k=6 and redo the cases from Question (7). Compare your answers.

Below I use the `class::knn()` classifier to generate predictions.

```r
library(class)

baked_test <- testing(init_split) %>%
  bake(train_rec, .)

class_preds <- knn(
  train = select(baked_train, -id, -type),
  test = select(baked_test, -type, -id),
  cl = pull(baked_train, type),
  k = 6)
```

I define my own testing set and perform my own predictions below. Note that the preprocessing occurs within the `classify_type()` function.

```r
testing_set <- testing(init_split) %>%
  mutate(id = 999)

my_preds <- map_dbl(1:nrow(testing_set), ~{
  classify_type(
    slice(testing_set, .x))
  }
)
```

I calculate the accuracy of each classifier.

```r
mean(class_preds == baked_test$type)
```

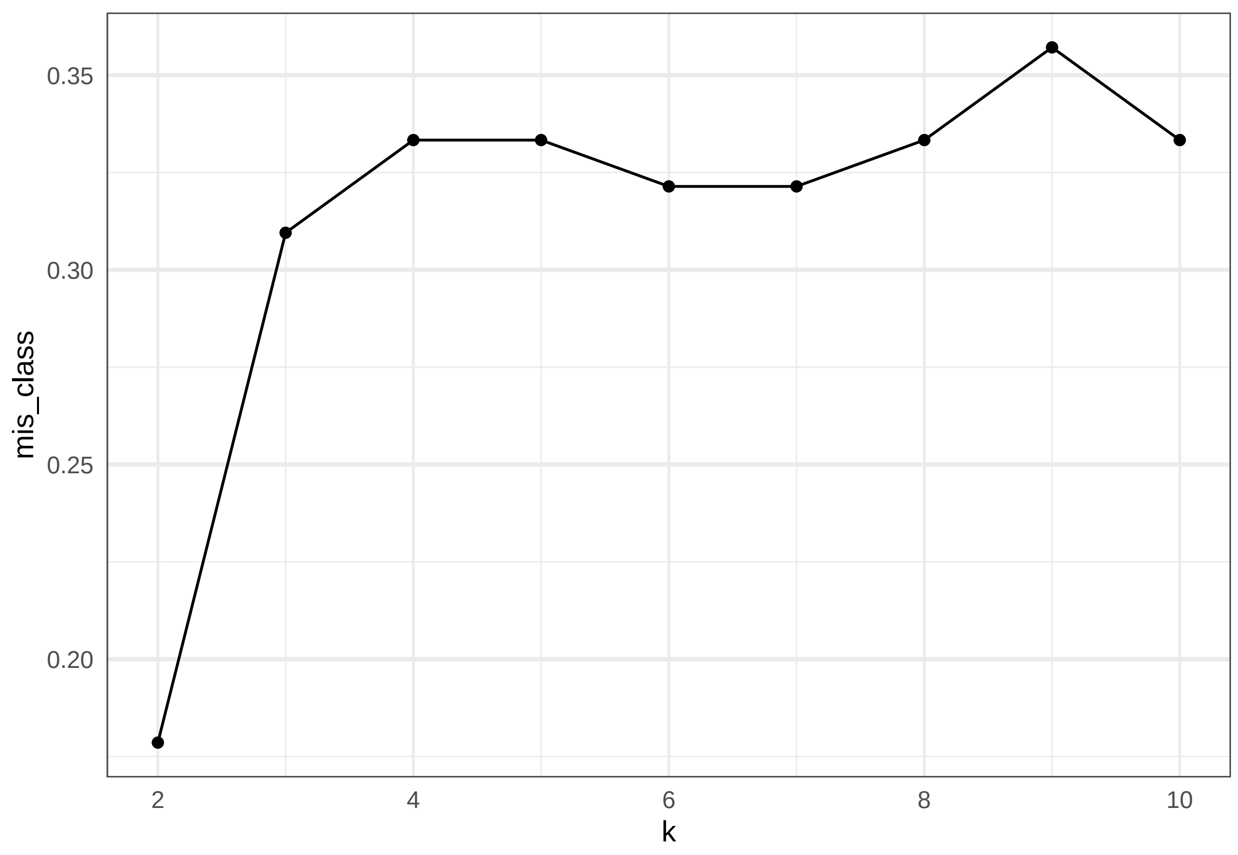
```
## [1] 0.6904762
```

```r
mean(my_preds == baked_test$type)
```

```
## [1] 0.9404762
```

(10 pts) Create a plot of k (x-axis) from 2 to 10 versus error rate (percentage of incorrect classifications) using ggplot.

```r
k_preds <- map_dbl(2:10, ~{
  class_preds <- knn(
    train = select(baked_train, -id, -type),
    test = select(baked_test, -type, -id),
    cl = pull(baked_train, type),
    k = .x)

  mean(class_preds == baked_test$type)

})

tibble(
  k = 2:10,
  mis_class = 1 - k_preds
  ) %>%
  ggplot(aes(k, mis_class)) +
  geom_line() +
  geom_point()
```



(10 pts) Produce a cross-table confusion matrix showing the accuracy of the classification using knn from the class package with k = 6.

```
table(class_preds, baked_test$type)
```

```
##
## class_preds  1  2  3  5  6  7
##           1 26  2  3  0  0  2
##           2  8 18  3  1  0  1
##           3  1  0  0  0  0  0
##           5  0  3  0  2  1  0
##           6  0  0  0  0  2  0
##           7  0  0  0  0  1 10
```

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
```

```
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] class_7.3-15    recipes_0.1.7   rsample_0.0.5   forcats_0.4.0
##  [5] stringr_1.4.0   dplyr_0.8.3     purrr_0.3.3     readr_1.3.1
##  [9] tidyr_1.0.0     tibble_2.1.3    ggplot2_3.2.0   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1         lubridate_1.7.4    lattice_0.20-38
##  [4] listenv_0.7.0      assertthat_0.2.1   zeallot_0.1.0
##  [7] digest_0.6.20      ipred_0.9-9        R6_2.4.0
## [10] cellranger_1.1.0   backports_1.1.4    evaluate_0.14
## [13] highr_0.8          httr_1.4.0         pillar_1.4.2
## [16] rlang_0.4.2        lazyeval_0.2.2     readxl_1.3.1
## [19] rstudioapi_0.10    furrr_0.1.0        rpart_4.1-15
## [22] Matrix_1.2-17      rmarkdown_1.16     labeling_0.3
## [25] splines_3.6.1      gower_0.2.1        munsell_0.5.0
## [28] broom_0.5.2        janitor_1.2.0      compiler_3.6.1
## [31] modelr_0.1.5       xfun_0.10          pkgconfig_2.0.2
## [34] globals_0.12.4     htmltools_0.4.0    nnet_7.3-12
## [37] tidyselect_0.2.5   prodlim_2018.04.18 codetools_0.2-16
## [40] future_1.14.0      crayon_1.3.4       withr_2.1.2
## [43] MASS_7.3-51.4      grid_3.6.1         nlme_3.1-140
## [46] jsonlite_1.6       gtable_0.3.0       lifecycle_0.1.0
## [49] magrittr_1.5       scales_1.0.0       cli_1.1.0
## [52] stringi_1.4.3      timeDate_3043.102  skimr_1.0.7
## [55] xml2_1.2.2         ellipsis_0.3.0     generics_0.0.2
## [58] vctrs_0.2.0        lava_1.6.6         tools_3.6.1
## [61] glue_1.3.1         hms_0.5.0          parallel_3.6.1
## [64] survival_2.44-1.1  yaml_2.2.0         colorspace_1.4-1
## [67] rvest_0.3.4        knitr_1.25         haven_2.2.0
```