Practicum 2 Part I

Package Loading and Data reading

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(rsample)
library(recipes)
```

```
##
## Attaching package: 'recipes'

## The following object is masked from 'package:stringr':
##
##     fixed

## The following object is masked from 'package:stats':
##
##     step
```

```
colnames <- c("age", "work_class", "fnlwgt", "education", "education_num",
              "marital_status", "occupation", "relationship", "race", "sex",
              "capital_gain", "capital_loss", "hours_per_week", "native_country", "inc_group")


census <- read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",
                   col_names = colnames) %>%
  mutate_if(is.character, as.factor)
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   work_class = col_character(),
##   fnlwgt = col_double(),
##   education = col_character(),
##   education_num = col_double(),
##   marital_status = col_character(),
```

```
##   occupation = col_character(),
##   relationship = col_character(),
##   race = col_character(),
##   sex = col_character(),
##   capital_gain = col_double(),
##   capital_loss = col_double(),
##   hours_per_week = col_double(),
##   native_country = col_character(),
##   inc_group = col_character()
## )
```

3. Data partitioning

I split the data using `rsample`.

> Note that I am stratifying the sample by country because without this the Rmd wouldn't knit properly even though the code runs from top to bottom without a hitch.

```
# split data
init_split <- initial_split(census, .8, strata = native_country)
census_train <- training(init_split)
census_test <- testing(init_split)
```

4. Model Fit

First I define the formula to prevent excessive text. Next I define the preprocessing step in the recipe. Following, I apply the preprocessing steps to both the training and testing data.

```
#------------------------------- define formula -------------------------------#
fit_form <- formula(inc_group ~ education + native_country + work_class + race + age + sex + native_cou

# make recipe and discretize age
fit_rec <- recipe(fit_form, census_train) %>%
  step_discretize(age, num_breaks = 5) %>%
  prep()

# apply preprocessing
baked_train <- bake(fit_rec, census_train)
baked_test <- bake(fit_rec, census_test)
```

In this next code chunk I fit all four models as requested.

```
#------------------------------------------------------------------------------#
#                                 Train Models                                 #
#------------------------------------------------------------------------------#
#---------------------------------- klaR fit ----------------------------------#
klar_nb <- klaR::NaiveBayes(fit_form, data = baked_train)

#------------------------------ naivebayes fit ------------------------------#
nb_nb <- naivebayes::naive_bayes(fit_form, data = baked_train)
```

```
## Warning: naive_bayes(): Feature education - zero probabilities are present.
## Consider Laplace smoothing.

## Warning: naive_bayes(): Feature native_country - zero probabilities are
## present. Consider Laplace smoothing.

## Warning: naive_bayes(): Feature work_class - zero probabilities are
## present. Consider Laplace smoothing.

## Warning: naive_bayes(): Feature age - zero probabilities are present.
## Consider Laplace smoothing.
```

```
#--------------------------------- e1071 fit ----------------------------------#
e1071_nb <- e1071::naiveBayes(fit_form, data = baked_train)

#--------------------------------- logit model --------------------------------#
logit_fit <- glm(fit_form, family = "binomial", data = baked_train)
```

In this next section I define a function that will predict the class using each of the above four models. It then takes a majority vote. That vote result is what is used in the ensemble prediction. I include the `Mode` function definiiton within the `pred_ensemble()` function definition. This is **bad practice** but alas, I am not making a package. The next important part of the function definition is to create a tibble which contains all of the prediction.

```
#------------------------------- ensemble model -------------------------------#
# There is no definition of what kind of ensembling to use. For that reason
# I will use a majority vote method

pred_ensemble <- function(new_data) {

  klar_preds <- predict(klar_nb, new_data)[[1]]
  nb_preds <- predict(nb_nb, new_data)
  e1071_preds <- predict(e1071_nb, new_data)
  logit_preds <- ifelse(predict(logit_fit, new_data, type = "response")  < .5, "<=50K", ">50K")

  # mode function for calculating mode
  Mode <- function(x) {
    ux <- unique(x)
    ux[which.max(tabulate(match(x, ux)))]
  }


  tibble(klar = klar_preds,
         nb = nb_preds,
         e1071 = e1071_preds,
         logit = logit_preds) %>%
    mutate_all(as.character) %>%
    # create a vector of all of the prediction results per row
    mutate(ensemble_vote = pmap(list(klar, nb, e1071, logit), c),
           # find the modal value
           ensemble_vote = map_chr(ensemble_vote, Mode))

}
```

Next, I apply the ensemble voting method to the testing set.

```
pred_ensemble(baked_test)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1546

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1877

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2088

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3154

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3466

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3532

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3622

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3708

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4289

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4491

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4503

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4654

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5309

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5816

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5820
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6270
```

```
## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```

```
## # A tibble: 6,512 x 5
##    klar  nb    e1071 logit ensemble_vote
##    <chr> <chr> <chr> <chr> <chr>
##  1 <=50K <=50K <=50K <=50K <=50K
##  2 >50K  >50K  >50K  >50K  >50K
##  3 <=50K <=50K <=50K <=50K <=50K
##  4 <=50K <=50K <=50K <=50K <=50K
##  5 <=50K <=50K <=50K <=50K <=50K
##  6 <=50K <=50K <=50K <=50K <=50K
##  7 <=50K <=50K <=50K <=50K <=50K
##  8 <=50K <=50K <=50K <=50K <=50K
##  9 <=50K <=50K <=50K <=50K <=50K
## 10 <=50K <=50K <=50K <=50K <=50K
## # ... with 6,502 more rows
```

Given the example values, I create a tibble and then apply the pre-processing recipe. Then, I create use the ensemble method function to get all prediction results.

```
#-------------------------------------------------------------------------------#
#                                    Predict                                    #
#-------------------------------------------------------------------------------#
#------------------------ white female federal worker ------------------------#
rare_human <- tibble(
  education = "Doctorate",
  age = 35,
  sex = "Female",
  native_country = "Portugal",
  work_class = "Local-gov",
  race = "White"
) %>%
  mutate_if(is.character, as.factor) %>%
  bake(fit_rec, .)


pred_ensemble(rare_human)
```

```
## # A tibble: 1 x 5
##   klar  nb    e1071 logit ensemble_vote
##   <chr> <chr> <chr> <chr> <chr>
## 1 >50K  >50K  >50K  <=50K >50K
```

```
# this woman makes less than 50k
```

The results of this prediction indicate that this individual absolutely does not make more than 50 thousand dollars annually. I mean, what local government work does?

Finally, it is always imperative to evaluate model performance. I create predictions for all models using the ensemble function. Then, I add the original y values to the predicted data set which will be used as the `truth`. Following, I manually create all of the confusion matrixes and calculate the accuracy.

```
#-------------------------------------------------------------------------------#
#                                Model Evaluation                               #
#-------------------------------------------------------------------------------#
#------------------------------ create predictions -----------------------------#
all_preds <- pred_ensemble(baked_test) %>%
  mutate(truth = baked_test$inc_group)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1546

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1877

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2088

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3154

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3466

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3532

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3622

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3708

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4289

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4491

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4503

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4654

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5309
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5816

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5820

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6270

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```

```
#----------------------------- confusion matrices -----------------------------#
janitor::tabyl(all_preds, klar, truth)
```

```
##   klar <=50K >50K
##  <=50K  4582  880
##   >50K   387  663
```

```
janitor::tabyl(all_preds, nb, truth)
```

```
##     nb <=50K >50K
##  <=50K  4582  880
##   >50K   387  663
```

```
janitor::tabyl(all_preds, e1071, truth)
```

```
##  e1071 <=50K >50K
##  <=50K  4582  880
##   >50K   387  663
```

```
janitor::tabyl(all_preds, logit, truth)
```

```
##  logit <=50K >50K
##  <=50K  4681  945
##   >50K   288  598
```

```
#----------------------------- accuracy measures ------------------------------#
pivot_longer(all_preds, cols = -truth, "model") %>%
  mutate(same = truth == value) %>%
  group_by(model) %>%
  summarise(accuracy = mean(same))
```

```
## # A tibble: 5 x 2
##   model         accuracy
##   <chr>            <dbl>
## 1 e1071            0.805
## 2 ensemble_vote    0.805
## 3 klar             0.805
## 4 logit            0.811
## 5 nb               0.805
```

What is beautiful about this is that it is very clear that all of the naive bayes models are well implemented because they return the exact same results for the confusion matrixes and accuracy.