hw_5

Sabrina Lem

5/13/2022

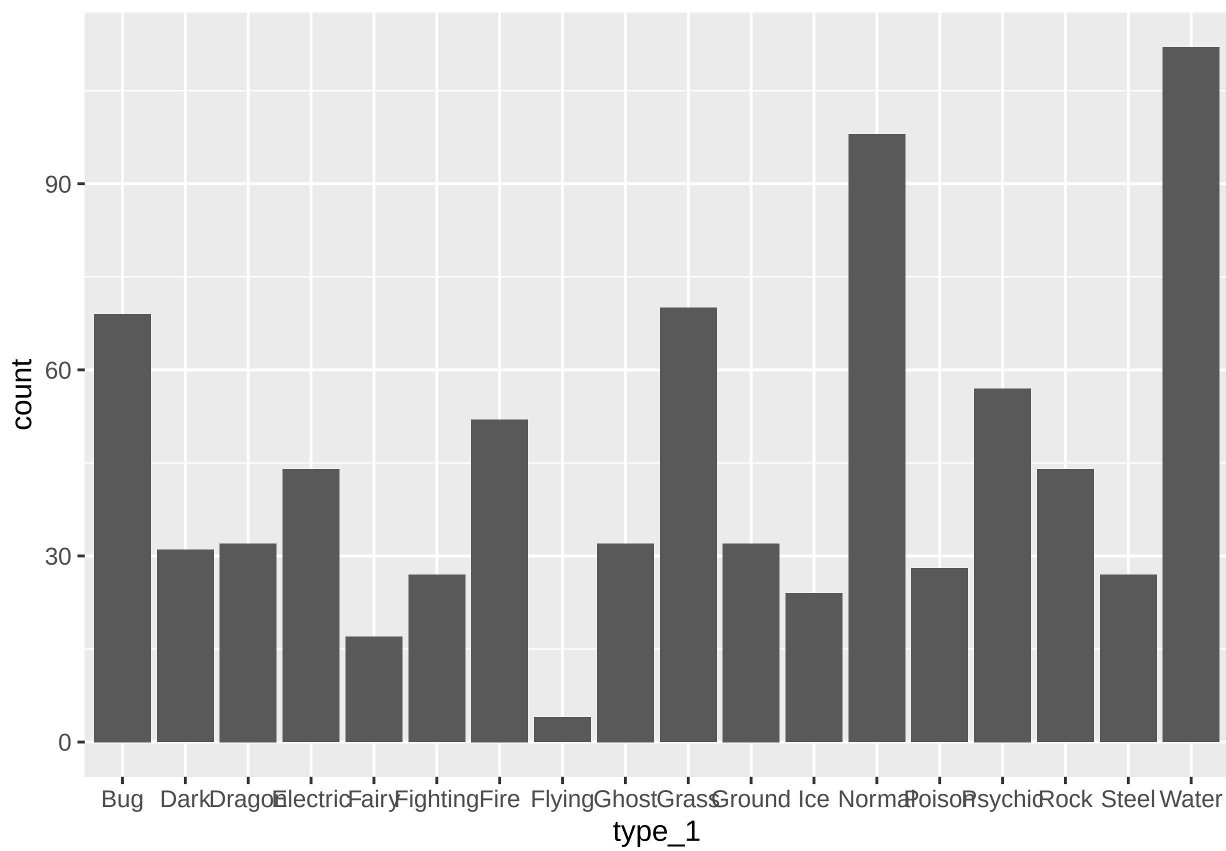
Exercise 1:

```
pokemon_clean <- janitor::clean_names(data)
```

clean names helps us identify variables easier in R. Before clean names some of the variables were labeled with two words. Clean names connects these words with underscores, so we are able to call them in the script.

Exercise 2:

```
plot <- ggplot(data=pokemon_clean, aes(x = type_1)) +
  geom_bar()
plot
```



```r
pokemon <- pokemon_clean %>% filter(type_1 == "Bug"|type_1 == "Fire"|
                                     type_1 == "Grass"|type_1 == "Normal"|
                                     type_1 == "Water"|type_1 == "Psychic")
pokemon$type_1 = factor(pokemon$type_1)
pokemon$legendary = factor(pokemon$legendary)
```

There are 18 different classes in the type_1 outcome variable. Flying has a lot fewer Pokemon than any of the other types.
Exercise 3:

```r
set.seed(1027)
pokemon_split <- initial_split(pokemon, prop= 0.7, strata = "type_1")

pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)

dim(pokemon_train)[1]/nrow(pokemon)
```

```
## [1] 0.6943231
```

```r
dim(pokemon_test)[1]/nrow(pokemon)
```

```
## [1] 0.3056769
```

```r
pokemon_folds <- vfold_cv(pokemon_train, strata = "type_1", v = 5)
```

The primary type variable is not binary. There are 6 possible outcomes for the primary type. Thus there when split in the k-folds, the randomization may not lead to representative subgroups. Stratification helps to ensure that the splits and thus the subgroups will represent the distribution of the different primary types.
Exercise 4:

```r
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk +
                           attack + speed + defense +
                           hp + sp_def, pokemon_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_interact(~ starts_with("legendary"):generation)
```

Exercise 5:

```r
elas_spec <-multinom_reg(penalty = tune(), mixture = tune() ) %>%
  set_engine("glmnet")
elas_wkfl <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(elas_spec)

penalty_grid <- grid_regular(penalty(range = c(-5,5)), mixture(range=c(0,1)), levels = 10)
```

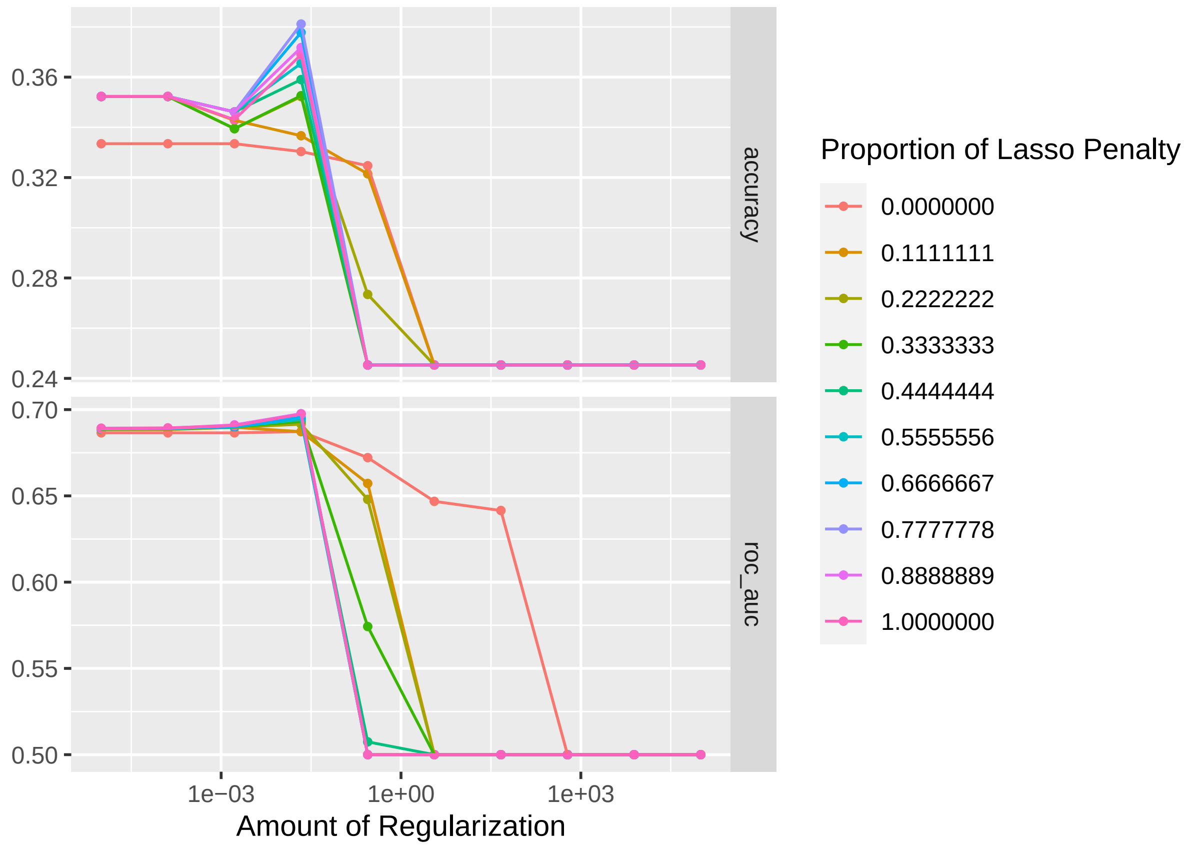
There are 10 levels and five folds. Thus for each training fold, (4 training folds) there will be 10 models. Thus there will be 40 models in total.

Exercise 6:

```r
tune_res <- tune_grid(
  elas_wkfl,
  resamples = pokemon_folds,
  grid = penalty_grid
)
tune_res
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 4
##   splits           id    .metrics           .notes
##   <list>           <chr> <list>             <list>
## 1 <split [252/66]> Fold1 <tibble [200 x 6]> <tibble [0 x 3]>
## 2 <split [253/65]> Fold2 <tibble [200 x 6]> <tibble [0 x 3]>
## 3 <split [253/65]> Fold3 <tibble [200 x 6]> <tibble [0 x 3]>
## 4 <split [256/62]> Fold4 <tibble [200 x 6]> <tibble [0 x 3]>
## 5 <split [258/60]> Fold5 <tibble [200 x 6]> <tibble [0 x 3]>
```

```r
autoplot(tune_res)
```



Smaller values of of penalty and mixture have higher and better accuracy and ROC AUC.

Exercise 7:

```r
best_penalty <- select_best(tune_res, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215   0.889 Preprocessor1_Model084
```

```r
elas_final <- finalize_workflow(elas_wkfl, best_penalty)

elas_final_fit <- fit(elas_final, data = pokemon_train)

aug <- augment(elas_final_fit, new_data = pokemon_test)
aug
```

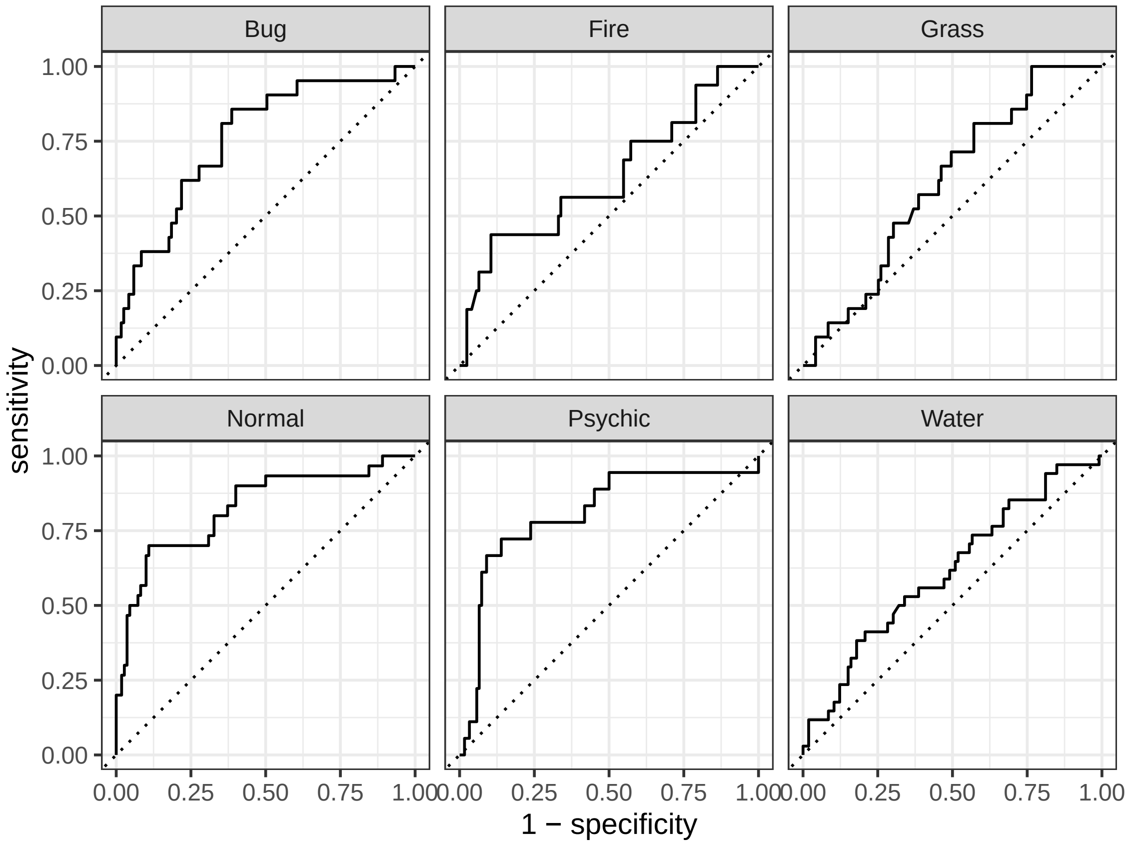
```
## # A tibble: 140 x 20
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <fct>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      9 Blastois~ Water  <NA>     630    79    103     120    135    115    78
##  2     14 Kakuna    Bug    Poison   205    45     25      50     25     25    35
##  3     18 PidgeotM~ Normal Flying   579    83     80      80    135     80   121
##  4     19 Rattata   Normal <NA>     253    30     56      35     25     35    72
##  5     20 Raticate  Normal <NA>     413    55     81      60     50     70    97
##  6     22 Fearow    Normal Flying   442    65     90      65     61     61   100
##  7     37 Vulpix    Fire   <NA>     299    38     41      40     50     65    65
##  8     39 Jigglypu~ Normal Fairy    270   115     45      20     45     25    20
##  9     40 Wigglytu~ Normal Fairy    435   140     70      45     85     50    45
## 10     48 Venonat   Bug    Poison   305    60     55      50     40     55    45
## # ... with 130 more rows, and 9 more variables: generation <dbl>,
## #   legendary <fct>, .pred_class <fct>, .pred_Bug <dbl>, .pred_Fire <dbl>,
## #   .pred_Grass <dbl>, .pred_Normal <dbl>, .pred_Psychic <dbl>,
## #   .pred_Water <dbl>
```

Exercise 8:

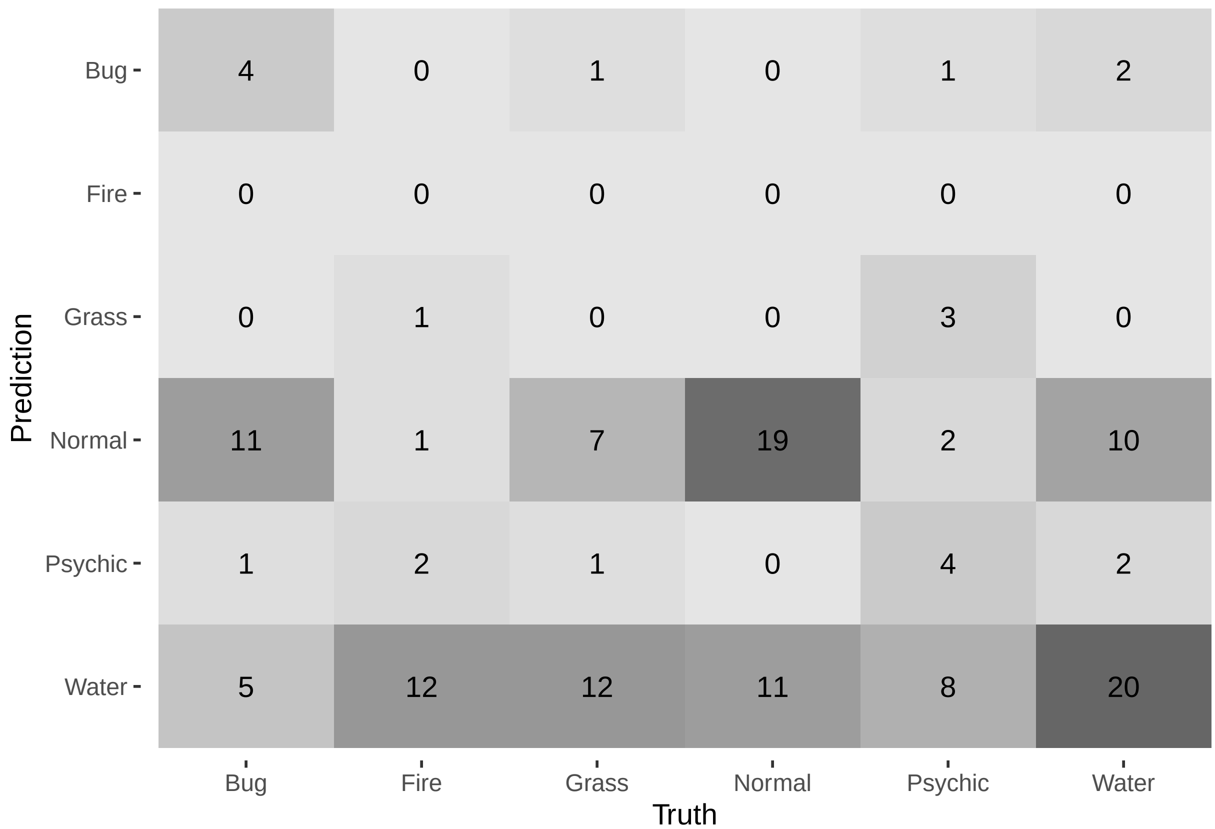
```r
aug %>% roc_auc(truth = type_1, estimate =c(
  .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,
  .pred_Psychic, .pred_Water))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.706
```

```r
aug %>% roc_curve(truth = type_1, estimate =c(
  .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal,
  .pred_Psychic, .pred_Water)) %>%
  autoplot()
```




```
aug %>% conf_mat(truth = type_1, estimate =.pred_class) %>%
  autoplot(type = "heatmap")
```



Bug, Psychic, and Normal primary types have the best ROC AUC curves. While Fire, Grass, and Water have ROC AUC curves that cover far less area. The model was not extremely accurate. Some variables were predicted better than others. Looking at the diagonals in the confusion matrix, we can see that there are a range of True Positive values. Water and Normal had high true positive counts of 20 and 19, respectively. However, Grass and Fire both had 0 True Positive counts. Psychic and Bug both had a TP count of 4, which is also not great. So overall, the model did not do very well.