# HW 8

**Enter your name and EID here: Erik Mercado, emm4376**

**You will submit this homework assignment as a pdf file on Gradescope.**

*For all questions, include the R commands/functions that you used to find your answer (show R chunk). Answers without supporting code will not receive credit. Write full sentences to describe your findings.*

We will use the following packages. If you get an error loading one of these packages you may need to install it with `install.packages()`.

```
# Load packages
library(tidyverse)
library(kknn)
library(plotROC)
library(tidymodels)
```

---

We will revisit the Pokemon dataset for this homework.

## Question 1: (2 pts)

Let's re-download the data to start from fresh and recode the variable `Legendary`:

```
# Download data from GitHub
pokemon <- read_csv("https://raw.githubusercontent.com/laylaguyot/datasets/main/pokemon.csv") %>%
    mutate(Legendary = factor(ifelse(Legendary, "Legendary",
                                     "Not Legendary"))) %>%
    mutate(Legendary = fct_relevel(Legendary, "Not Legendary"))
## (NOTE: Tidymodels requires classification outcome to be a factor)

# Take a look
head(pokemon)
```

```
## # A tibble: 6 x 13
##   Number Name   Type1 Type2 Total    HP Attack Defense SpAtk SpDef Speed Gener~1
##    <dbl> <chr>  <chr> <chr> <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1      1 Bulba~ Grass Pois~   318    45     49      49    65    65    45       1
## 2      2 Ivysa~ Grass Pois~   405    60     62      63    80    80    60       1
## 3      3 Venus~ Grass Pois~   525    80     82      83   100   100    80       1
## 4      3 Venus~ Grass Pois~   625    80    100     123   122   120    80       1
## 5      4 Charm~ Fire  <NA>    309    39     52      43    60    50    65       1
## 6      5 Charm~ Fire  <NA>    405    58     64      58    80    65    80       1
## # ... with 1 more variable: Legendary <fct>, and abbreviated variable name
## #   1: Generation
```

In the last assignment, you tried linear and logistic regression and (hopefully) found that these two models had a similar performance. It turns out that the logistic regression model fitted to the complete dataset had an AUC = 0.8581. Let's see how a logistic regression would be able to predict the `Legendary` status of "new" pokemons using a 10-fold cross-validation:

```r
## Make this example reproducible by setting a seed
set.seed(322)

## Create the recipe
rec <- pokemon %>%
    recipe(Legendary ~ Attack + HP)

## Create the model
model <- logistic_reg() %>%
    set_engine("glm") %>%
    set_mode("classification")

## Create the workflow
wf <- workflow() %>%
    add_recipe(rec) %>%
    add_model(model)

## Create 10 folds from the dataset
folds <- vfold_cv(pokemon, v = 10)

## Run cross validation with the model
res <- fit_resamples(wf, resamples = folds)

## Show performance metrics
res %>%
    collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.916    10  0.0115 Preprocessor1_Model1
## 2 roc_auc  binary     0.867    10  0.0264 Preprocessor1_Model1
```

How does the average AUC presented here compare to the AUC of our `pokemon_log` model trained on the entire data? What does it indicate about the logistic regression model?

**The new average AUC is higher than the one from the 'pokemon_log' model that we trained with the entire model. This means that the new model will perform better than the old one.**

---

## Question 2: (3 pts)

Another classifier we can consider to predict `Legendary` status from `HP` and `Attack` is using the k-nearest neighbors (kNN). Fit the kNN model with 5 nearest neighbors and save the results to an object called `pokemon_kNN`. How does this model make a prediction for each pokemon (i.e., what output do we get when using the function `predict()`)?

```r
## Create the recipe
knn_rec <- pokemon |>
  recipe(Legendary ~  Attack + HP)

## Create the model
knn_model <- nearest_neighbor(neighbors = 5) %>%
    set_engine("kknn") %>%
    set_mode("classification")

## Create workflow
knn_wf <- workflow() |>
    add_recipe(knn_rec) |>
    add_model(knn_model)

## Fit the model on the full training dataset using the `fit()` function
pokemon_knn <- knn_wf |>
  fit(data = pokemon)
```

**This model make predictions for each Pokemon by comparing the legendary status of the Pokemon that are nearest to the the Pokemon who's legendary status we are trying to predict.**

---

## Question 3: (3 pts)

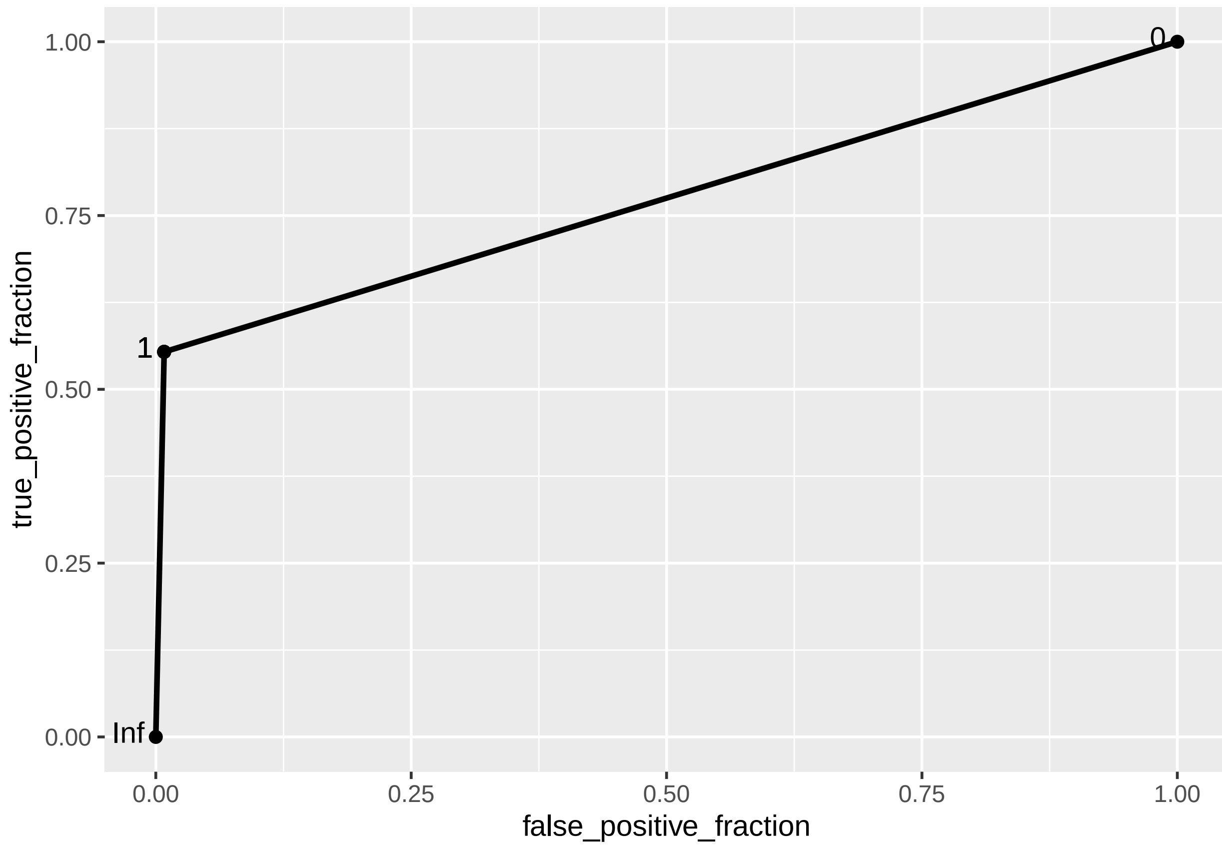
Use the `pokemon_kNN` model to build a ROC curve for the model using `geom_roc()`. NOTE: In order to use `geom_roc()` you will need to convert the `Legendary` variable into a numeric variable where 0 = "Not Legendary" and 1 = "Legendary".

```r
## Create the data for doing model predictions
dat_model <- rec %>%
    prep(pokemon) %>%
    bake(new_data = NULL)

## Make model predictions using `dat_model`; convert Legendary variable to 0/1; make ROC curve plot
pokemon_knn |>
  extract_fit_parsnip()
```

```
## parsnip model object
##
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(5,     data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.06875
## Best kernel: optimal
## Best k: 5
```

```r
dat_model |>
  mutate(Legendary = ifelse(Legendary == "Legendary", 1, 0)) |>
```

```r
  mutate(predictions = ifelse(predict(pokemon_knn, new_data = pokemon) == "Legendary", 1, 0)) |>
  ggplot(aes(d = Legendary, m = predictions[,".pred_class"])) +
  geom_roc()
```




---

### Question 4: (4 pts)

Perform a 10-fold cross-validation with the `pokemon_kNN` model to get an unbiased estimate of the AUC of the model

```r
## Run 10-fold cross validation and print out performance metrics
knn_folds <- vfold_cv(pokemon, v = 10)

## Fit the model to the different folds of the data
knn_res <- fit_resamples(wf, resamples = knn_folds)

## Show performance metrics
knn_res %>%
    collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
```

```
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.919    10  0.0115 Preprocessor1_Model1
## 2 roc_auc  binary     0.858    10  0.0213 Preprocessor1_Model1
```

How does the AUC compare to the logistic regression model when predicting `Legendary` status on "new" data? What does it indicate about our k-NN model?

**The AUC for the k-NN model is lower than the AUC for the logistic regression model. This indicates that the k-NN model is less accurate at predicting the right outcome.**

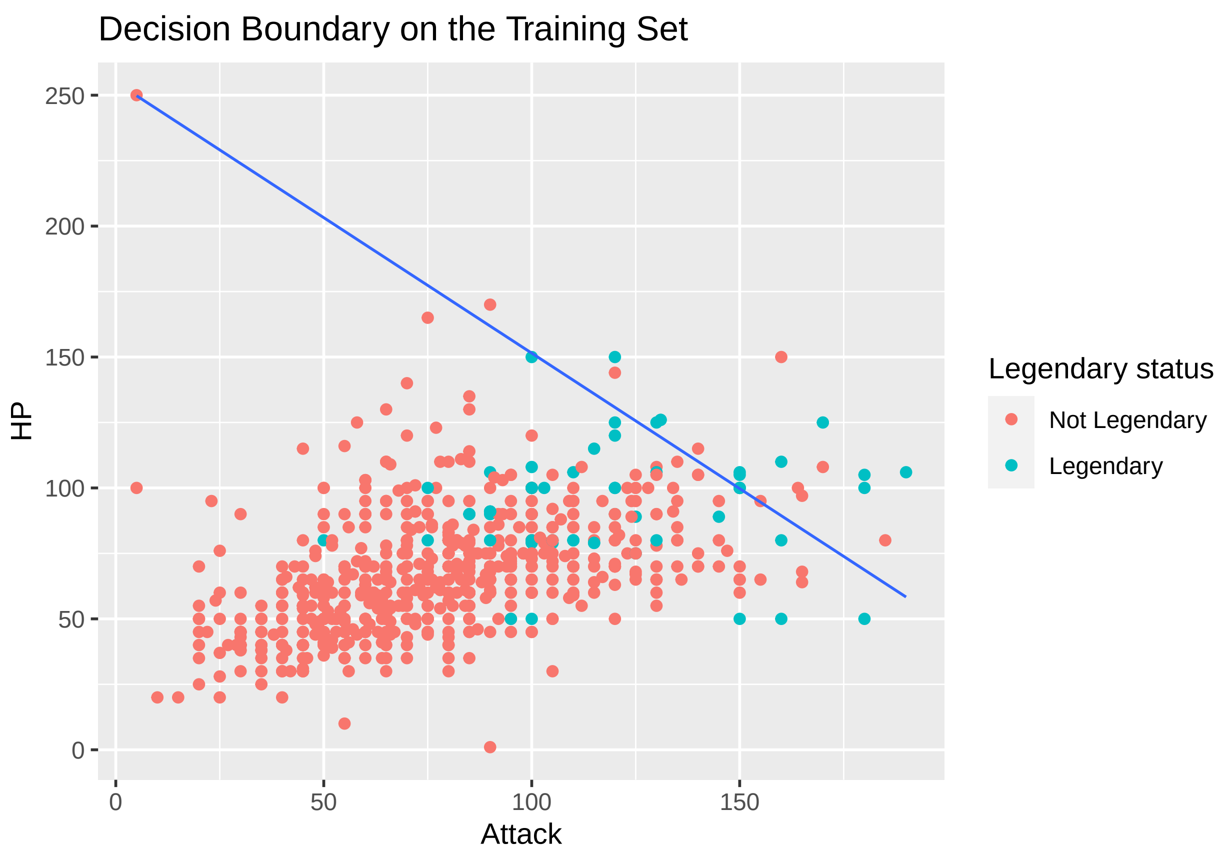
---

## Question 5: (3 pts)

Let's focus on the `pokemon_kNN` model trained on a random 9/10 of the data and then tested on the remaining 1/10. We plot the decision boundary: the blue boundary classifies points inside of it as *Legendary* and points outside as *Not Legendary*. Describe where the false positive cases and the false negative cases are in the plot (indicate if they are inside/outside the decision boundary and what they mean).

```r
# Make this example reproducible by setting a seed
set.seed(322)

# Split data into train and test sets
pokemon_split <- initial_split(pokemon, prop = 0.9)
train <- training(pokemon_split)
test <- testing(pokemon_split)

# Fit the model on the train data
knn_rec <- train %>%
    recipe(Legendary ~ Attack + HP)
knn_model <- nearest_neighbor(neighbors = 5) %>%
    set_engine("kknn") %>%
    set_mode("classification")
knn_wf <- workflow() %>%
    add_recipe(knn_rec) %>%
    add_model(knn_model)
pokemon_kNN <- wf %>%
    fit(data = train)

# Make a grid for the graph to layout the contour geom
grid <- tibble(expand.grid(Attack = seq(min(pokemon$Attack),
                                        max(pokemon$Attack),
                                        length.out = 100),
                           HP = seq(min(pokemon$HP),
                                    max(pokemon$HP),
                                    length.out = 100)))

## Make predictions on this grid
pgrid <- pokemon_kNN %>%
    extract_fit_parsnip() %>%
    augment(new_data = grid) %>%
    mutate(p = `.pred_Legendary`)
```

```r
# Use this grid to predict legendary status
pgrid %>%
  ggplot(aes(Attack, HP)) +
  # Only display data in the train set
  geom_point(aes(Attack, HP, color = Legendary),
             data = train) +
  # Draw the decision boundary
  geom_contour(aes(z = p), breaks = 0.5) +
  # Labels
  labs(title = "Decision Boundary on the Training Set",
       color = "Legendary status")
```



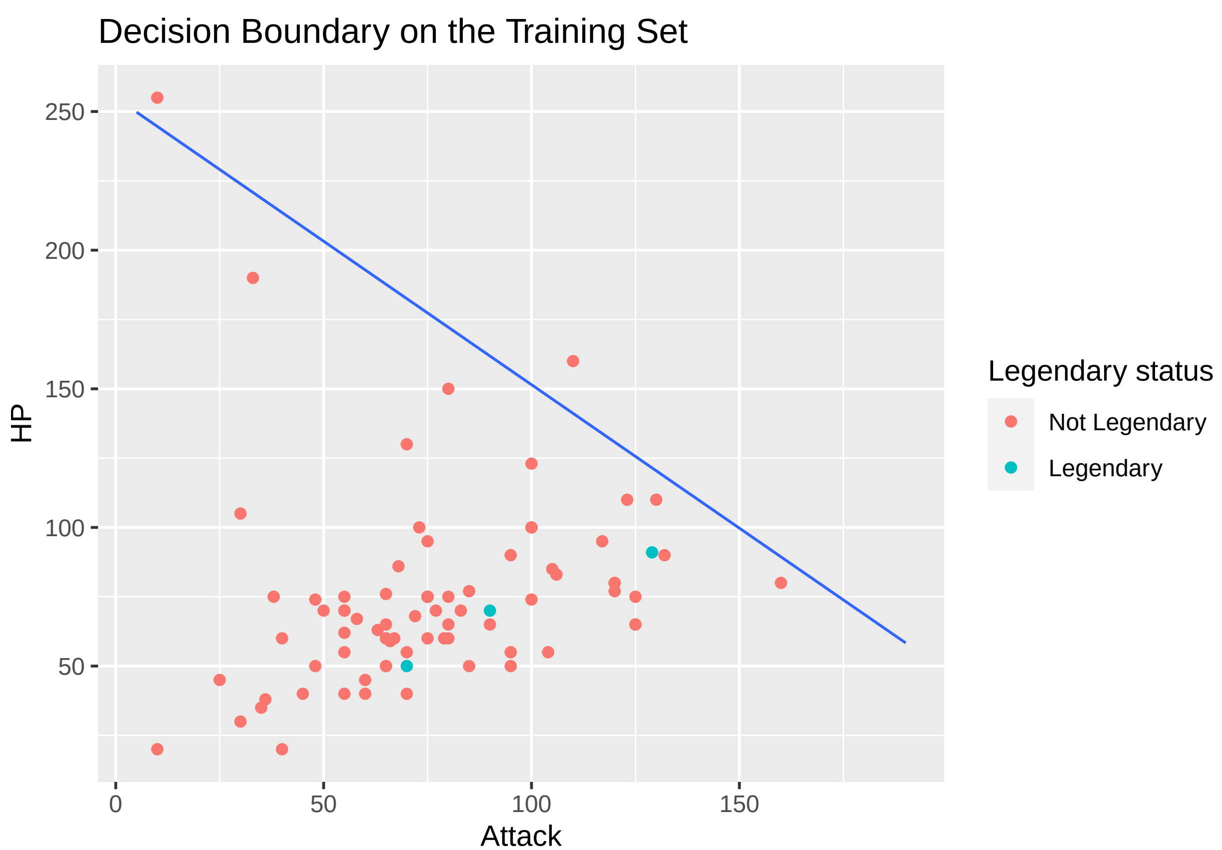

**The majority of the false negatives are located right around the outside the edge of the boundary and the majority of false positives are located right around the edge on the inside of the boundary. This is the case for these individual observations because they have more of their opposite "legendary status" surrounding them than their own "legendary status."**

---

### Question 6: (3 pts)

Now, represent the same decision boundary but with the test set. *Hint: use the last piece of the code from the previous question.*

```r
# your code goes below (make sure to edit comment)
pgrid %>%
  ggplot(aes(Attack, HP)) +
  # Only display data in the train set
  geom_point(aes(Attack, HP, color = Legendary),
             data = test) +
  # Draw the decision boundary
  geom_contour(aes(z = p), breaks = 0.5) +
  # Labels
  labs(title = "Decision Boundary on the Training Set",
       color = "Legendary status")
```




Comparing how the decision boundary performs on the training set versus the test set, describe why the kNN model might not perform very well on the test set.

**The reason why the kNN model might not perform very well on the test set is because the actual legendary were predicted to be outside the decision berrrier.**

---

## Formatting: (2 pts)

Comment your code, write full sentences, and knit your file!

```
##                                                                                                  sysname
##                                                                                                 "Darwin"
##                                                                                                  release
##                                                                                                 "22.4.0"
##                                                                                                  version
## "Darwin Kernel Version 22.4.0: Mon Mar  6 21:00:17 PST 2023; root:xnu-8796.101.5~3/RELEASE_X86_64"
##                                                                                                 nodename
##                                                                                     "Eriks-MBP-2424.lan"
##                                                                                                  machine
##                                                                                                 "x86_64"
##                                                                                                    login
##                                                                                                   "root"
##                                                                                                     user
##                                                                                                   "erik"
##                                                                                           effective_user
##                                                                                                   "erik"
```