alpha
sample_n()

R button
sticker

# Introduction to Classification and Training Data

**k-nearest neighbors** (or **knn**) is an introductory supervised machine learning algorithm, most commonly used as a classification algorithm.

**Classification** - *def* - prediction, but specifically for a response variable that is categorical.

For example, for a data set with SLU students, we might be interested in predicting whether or not each student graduates in four years (so the response has two categories: graduates in 4 years or doesn't). We might want to classify this response based on various student characteristics like anticipated major, GPA, standardized test scores, etc. knn can also be used to predict a quantitative response, but we'll focus on categorical responses throughout this section.

If you've had STAT 213, you might try to draw some parallels to knn and classification using logistic regression. Note, however, that logistic regression required the response to have **two** levels while knn can classify a response variable that has **two or more** levels.

To introduce this, we will be using `pokemon_full.csv` data. Pokemon have different Types: we will use `Type` as a categorical response that we are interested in predicting. For simplicity, we will only use Pokemon's primary type and we will only use 4 different types:

```
set.seed(1119)
library(tidyverse)
library(here)
pokemon <- read_csv(here("other", "pokemon_full.csv")) %>%
  filter(Type %in% c("Steel", "Dark", "Fire", "Ice"))
```

**Overall Goal**: classify Pokemon into types as "accurately" as possible using their other characteristics. (like stats, height, weight, etc.)

### Training and Test Data

In order to develop our knn model (note that we still haven't discussed what knn actually is yet!), we first need to discuss terms that applies to almost **all** predictive/classification modeling: training and test data.

**training data** - *def* - ~~data~~ used to train a model

- ~~70-80% of~~ subset of the full data set

≈70-80% of the observations in the full data set.

**test data** - *def* - subset of the full data used to test a fitted model's predictive ability.

- ~20-30% of the observations in the full data set.

Why do we need to do this division? Using the full data set for both training a model and testing that model is "cheating:" the model will perform better than it should because we are using each observation twice: once for fitting and once for testing. Having a separate test data set that wasn't used to fit the model gives the model a more "fair" test, as these observations are supposed to be new data that the model hasn't yet seen.

The following code uses that `sample_n()` function to randomly select 15 observations to be in the training data set. `anti_join()` then makes a test data set without the 15 pokemon in the training data set. (Note that our training data set would almost always be larger than 15, but we are using a small number here so that we can more clearly see how knn works in the next section).

```
train_sample <- pokemon %>%
  sample_n(15)
test_sample <- anti_join(pokemon, train_sample)
```

The ideas of a training data set and test data set are pervasive in predictive and classification models, including models not related to knn. Note that we are going to do this method because it's the simplest: if you wanted to take this a step further, you'd repeat the training and test process 5 or 10 times, using what's known as **k-fold cross-validation.**

#### Exercises

1. Explain what `anti_join()` joins on when by isn't specified and why not specifying a by argument works for this example. You may need to use `?anti_join` to answer this question

# Introduction to K-Nearest-Neighbors

The purpose of this handout is to introduce **k-nearest neighbors** (or **knn**), an introductory supervised machine learning algorithm, most commonly used to predict a categorical response variable with two or more categories.

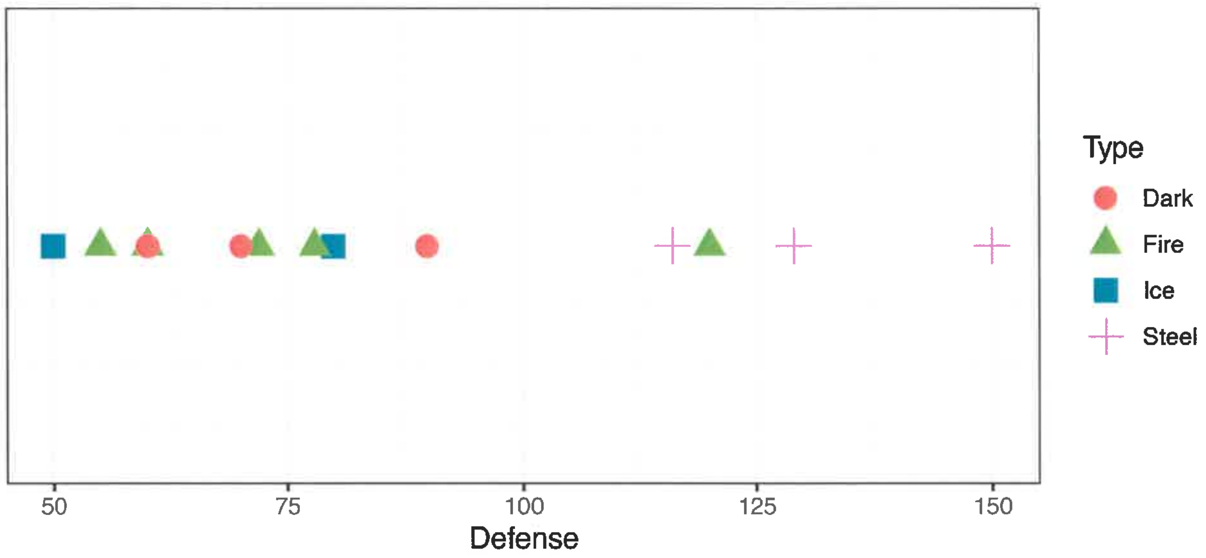
To introduce this, we will be using Pokemon data (the last time many of you will see Pokemon data!). Pokemon have different Types: we will use `Type` as a categorical response that we are interested in predicting. For simplicity, we will only use Pokemon's primary type and we will only use **4** different types:

Table 1: Table continues below

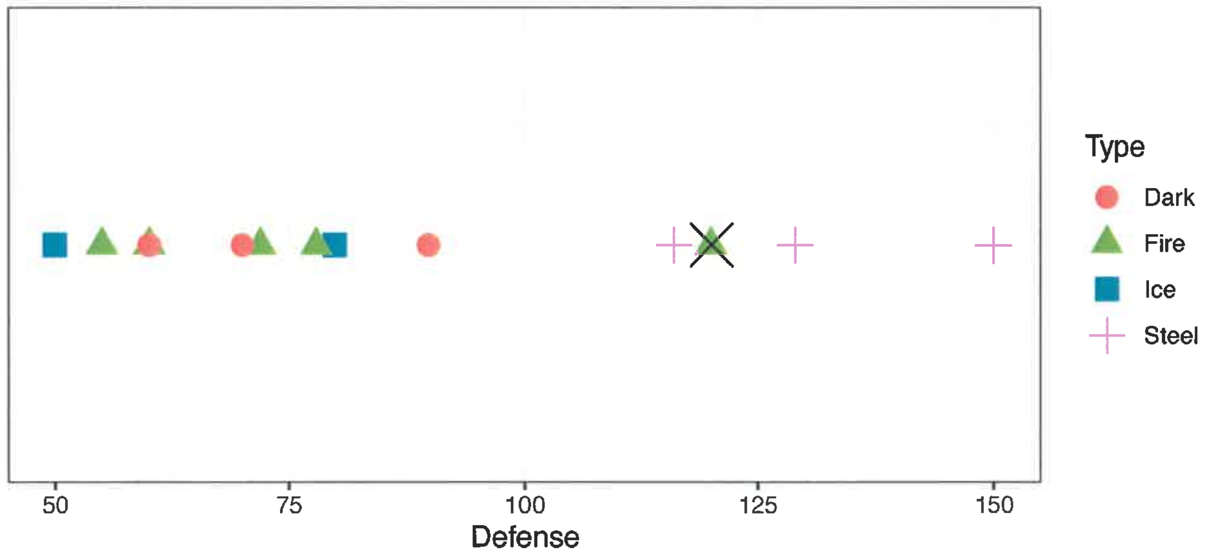
| ...1 | Name | Type | HP | Attack | Defense | Speed | SpAtk | SpDef |
|---|---|---|---|---|---|---|---|---|
| 491 | Darkrai | Dark | 70 | 90 | 90 | 125 | 135 | 90 |
| 136 | Flareon | Fire | 65 | 130 | 60 | 65 | 95 | 110 |
| 571 | Zoroark | Dark | 60 | 105 | 60 | 105 | 120 | 60 |
| 221 | Piloswine | Ice | 100 | 100 | 80 | 50 | 60 | 60 |
| 668 | Pyroar | Fire | 86 | 68 | 72 | 106 | 109 | 66 |
| 262 | Mightyena | Dark | 70 | 90 | 70 | 70 | 60 | 60 |

| Generation | Legendary | height | weight | base_experience |
|---|---|---|---|---|
| 4 | TRUE | 15 | 505 | 270 |
| 1 | FALSE | 9 | 250 | 184 |
| 5 | FALSE | 16 | 811 | 179 |
| 2 | FALSE | 11 | 558 | 158 |
| 6 | FALSE | 15 | 815 | 177 |
| 3 | FALSE | 10 | 370 | 147 |

**knn with k = 1 and 1 Predictor** Suppose that we have just **15** pokemon in our data set. We want to predict `Type` from just one predictor, Defense. Below is a plot that shows the defenses of the 15 pokemon in our data set, and has points coloured by Type and with different shapes for Type.



We see from the plot that `Steel` type Pokemon tend to have pretty high defense values. Now suppose that we want to predict the `Type` for a new Pokemon not in the original data set, Dialga. We know that Dialga has a `Defense` stat of 120: the plot below shows Dialga marked with a large black X.



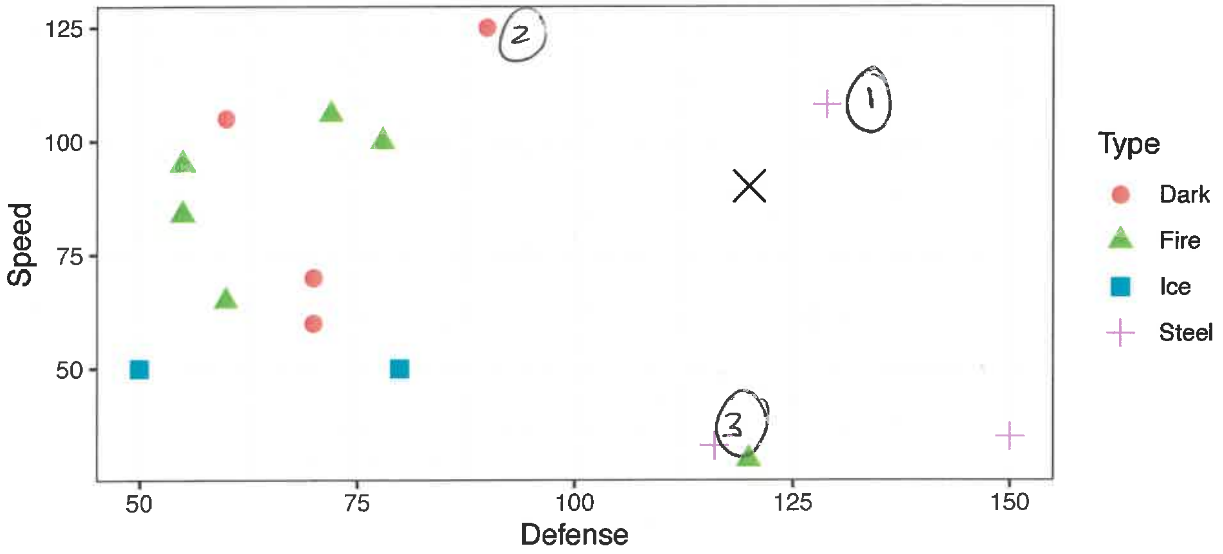
What would the prediction for Dialga be? Why? For k=1, Fire; it's the single "nearest neighbor."

**knn with k > 1 and One Predictor** But, we might not necessarily want to predict the response value based on the single nearest neighbor. Dialga is also near many purple plus signs: should those factor in at all? We can extend knn to different values for k. For example, $k = 3$ looks at the 3 nearest neighbors, and assigns a prediction as the category that appears the **most** among those 3 nearest neighbors.

What would the prediction for Dialga be? Why?

Steel: for k=3, 2 are Steel, 1 is Fire so Steel is majority.

**knn with k > 1 and More Than One Predictor** We can increase the number of predictors in our knn model as well. We can generally include as many predictors as we would like, but visualizing becomes challenging with more than 2 predictors and nearly impossible with more than 3 predictors. For the case of two predictors, suppose that we want to use Defense and Speed as our predictors for Type. Dialga, the Pokemon we want to predict for, is again marked with a large black X.



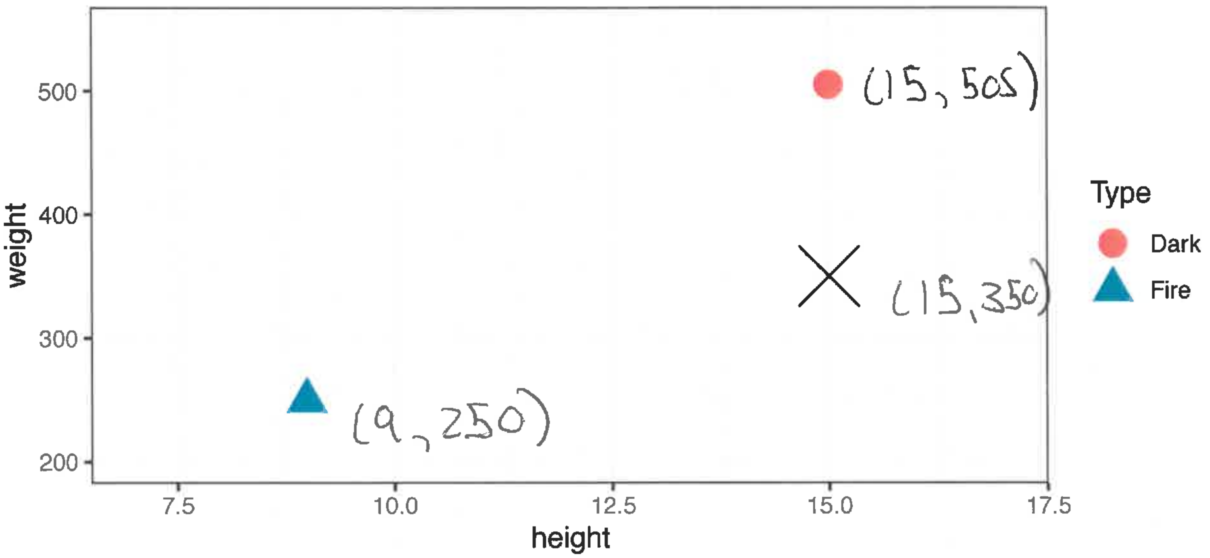
For $k = 1$, what **Type** would you predict for Dialga?

Steel - closest neighbor

For $k = 3$, what **Type** would you predict for Dialga?

Steel still: 2 Steel and 1 Dark so Steel is majority.

**Scaling Predictor Variables before Using knn** In general, we want to scale any quantitative predictors when using knn because it relies on distances between points in its predictions. This is easiest to see with an example. Suppose, in our Pokemon example, we just have 2 observations in our data set: a Dark Type pokemon with a height of 15 centimeters and a weight of 505 pounds, and a Fire Type Pokemon with a height of 9 centimeters and a weight of 250 pounds.



On the plot is also given a Pokemon that we wish to predict the Type of, marked with a black X. Upon visual inspection, with k = 1, it looks like we would classify this pokemon as Dark. However, because the units of weight and height are on very different scales, if we actually find the distances between the X and the two pokemon in our data set. . . .

~~distance~~ $= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

$\text{distance}_{\overset{\cancel{\text{fire}}}{\text{dark}},\ \cancel{\text{dark}}\,X} = \sqrt{(15-15)^2+(505-350)^2}$

$= 155$ units

$\text{distance}_{\text{fire},X} = \sqrt{(\overset{9}{\cancel{15}}-\overset{15}{\cancel{9}})^2+(250-350)^2}$

$= \sqrt{36+10000}$

$= 100.2$ ish units

knn model says △ Fire is closer?

To get around this issue, it is customary to scale all quantitative predictors before applying knn. One method of doing this is applying

$$scaled_x = \frac{x - min(x)}{max(x) - min(x)}$$

applied to all predictors, puts all on same scale: between 0 and 1.

For example, scaling **weight** for the 15 original pokemon:

```
library(pander)
train_sample %>% slice(1:3) %>% select(weight) %>%
  pander()
```

| weight |
|---|
| 505 |
| 250 |
| 811 |

e.g. height = c(100, 160, 200)

$$\text{height scaled}_1 = \frac{100-100}{200-100} = 0$$

$$\text{hight\_s}_2 = \frac{160-100}{200-100} = 0.6$$

$$\text{height\_s}_3 = \frac{200-100}{200-100} = 1.$$

puts all weights between 0 and 1:

```
train_sample %>% mutate(weight_s = (weight - min(weight)) /
                          (max(weight) - min(weight))) %>%
  select(weight_s) %>%
  slice(1:3) %>%
  pander()
```

| weight_s |
|---|
| 0.1874 |
| 0.0835 |
| 0.312 |

If we do the same with **height**, then the variables will contribute more equally to the distance metric used in knn.

The code below scales all numeric variables in a data set, using the `across()` function. `across()` applies a tranformation to every column in a data set that satisfies the condition given in the `where` argument. You would probably be able to take a data science with `R` class every semester of your college career and still come "across" new super-useful functions!

```
library(pander)
train_sample %>%
  mutate(across(where(is.numeric), ~ (.x - min(.x)) /
                                    (max(.x) - min(.x)))) %>%
  slice(1:3) %>% select(2, 3, 4, 5) %>%
  pander()
```

| Name | Type | HP | Attack |
|---|---|---|---|
| Darkrai | Dark | 0.4167 | 0.4444 |
| Flareon | Fire | 0.3333 | 0.8889 |
| Zoroark | Dark | 0.25 | 0.6111 |

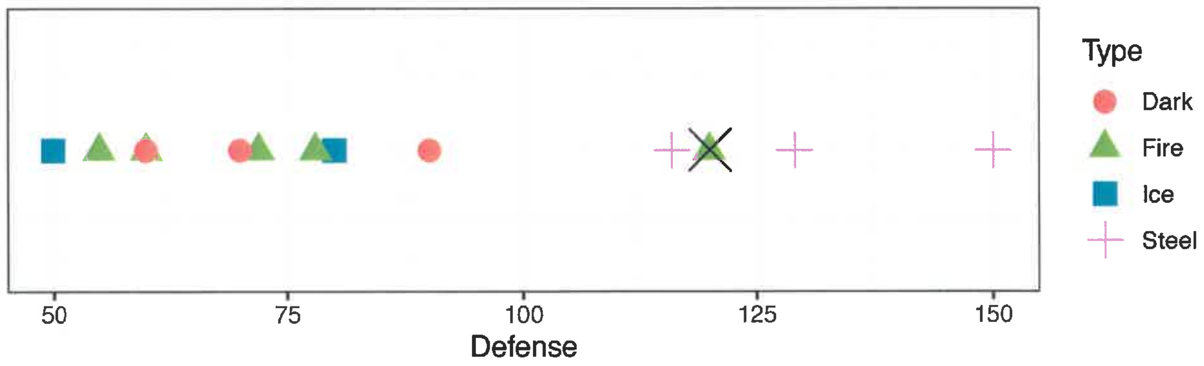
**Exercises**

1. In this tiny example, the actual (height, weight) coordinates of the Fire pokemon are (9, 250), the actual coordinates of the Dark pokemon are (15, 505), and the actual coordinates of the test pokemon are (15, 350). We mentioned that, visually, the pokemon looks "closer" to the Dark type pokemon. Verify that this is **not** actually the case by computing the actual distances numerically.

completed on page 4.

2. After scaling according to the formula in this section, the coordinates (height, weight) of the Fire pokemon are (0, 0) and the coordinates of the Dark pokemon are (1, 1). (Since there are only two observations, the formula doesn't give any output between 0 and 1 for this tiny example). The scaled coordinates for the test pokemon are (1, 0.39). Verify that, after scaling, the test pokemon is "closer" to the Dark type pokemon by numerically computing distances.

3. Consider again the example with 15 pokemon in the training data set and a single predictor, `Defense`.



With k = 2, there is a tie between Fire and Steel. Come up with a way in which you might break ties in a knn algorithm.

4. Explain what knn would use as a prediction for all test observations if k equals the number of observations in the training data set.

* Copy

# Evaluating knn Models

Some things we haven't yet discussed include:

- how should we choose which predictors to explore for knn models?
- how will we evaluate what makes a model "good"?
- how will we choose a value of $k$?

We will tackle each of these questions, to varying degrees, in this section.

## Choosing Predictors

We will continue to use the scaled version of the pokemon data set for this handout. This time, we will have **75** pokemon in our training data set and we are only looking at the `Steel`, `Dark`, `Fire`, and `Ice` types.

```
set.seed(1119)
library(tidyverse)
library(pander)
library(here)
pokemon <- read_csv(here("other", "pokemon_full.csv")) %>%
  filter(Type %in% c("Steel", "Dark", "Fire", "Ice")) %>%
  mutate(across(where(is.numeric), ~ (.x - min(.x)) /
                                (max(.x) - min(.x))))
train_sample <- pokemon %>%
  sample_n(75)

test_sample <- anti_join(pokemon, train_sample)

train_sample %>% head() %>% slice(1:3) %>% pander()
```

Table 1: Table continues below

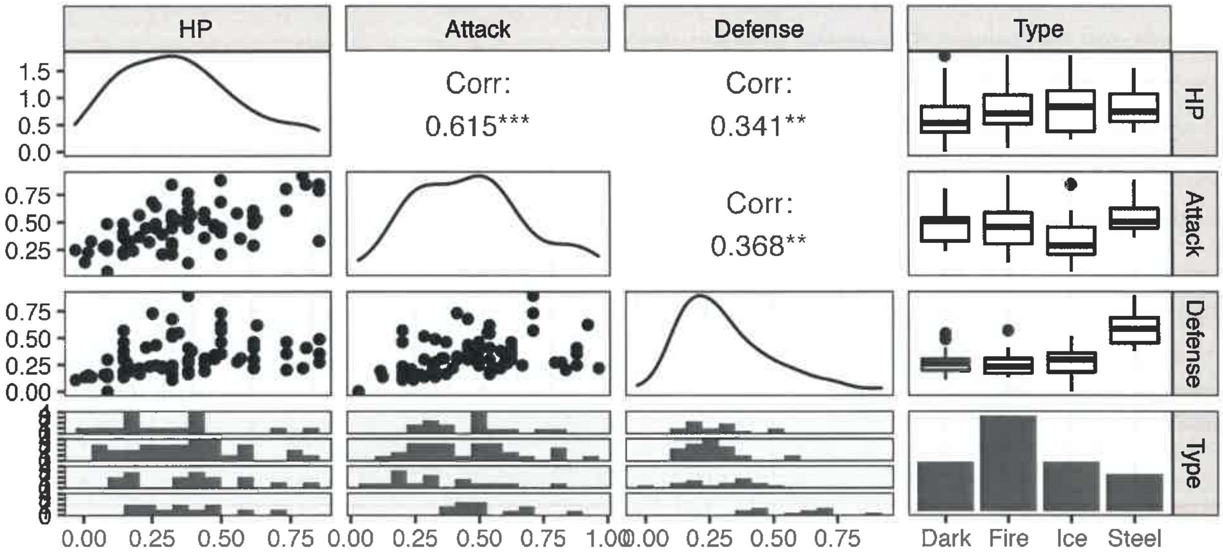
| ...1 | Name | Type | HP | Attack | Defense | Speed | SpAtk | SpDef |
|---|---|---|---|---|---|---|---|---|
| 0.6792 | Darkrai | Dark | 0.3846 | 0.5238 | 0.4054 | 0.9906 | 0.8889 | 0.3529 |
| 0.1841 | Flareon | Fire | 0.3297 | 0.8413 | 0.2432 | 0.4245 | 0.5926 | 0.4706 |
| 0.7908 | Zoroark | Dark | 0.2747 | 0.6429 | 0.2432 | 0.8019 | 0.7778 | 0.1765 |

| Generation | Legendary | height | weight | base_experience |
|---|---|---|---|---|
| 0.6 | TRUE | 0.1444 | 0.07244 | 0.8594 |
| 0 | FALSE | 0.07778 | 0.03505 | 0.5234 |
| 0.8 | FALSE | 0.1556 | 0.1173 | 0.5039 |

There are many candidate predictors in this data set: `HP`, `Attack`, `Defense`, ..., all the way up to `base_experience`. How should we determine which predictors to include in our model?

Much of this will be trial and error by evaluating different models with a criterion that we will talk about in the next section. However, it is always helpful to explore the data set with graphics to get us to a good starting point. A **scatterplot matrix** is a useful exploratory tool. The following is a scatterplot matrix with the response variable, `Type`, and just three candidate predictors, `HP`, `Attack`, and `Defense`, created with the `GGally` ("g-g-ally") package.

```
## install.packages("GGally")
library(GGally)
ggpairs(data = train_sample, columns = c(4, 5, 6, 3),
        lower = list(combo = wrap(ggally_facethist, bins = 15)))
```



The `lower` argument changes the number of bins in the faceted histograms in the bottom row. You can mostly ignore this.

The `columns` argument is important: it allows you to specify which columns you want to look at. I prefer putting the response, `Type` (column ~~3~~ 4) in the last slot.

We can examine this to see which variables seem to have a relationship with `Type`. Where would we want to look for this? The last column or the first row.

It looks like a HP, Attack, and Defense might all be useful predictors.

What's given on the diagonal of the scatterplot matrix?

A density plot or a bar plot

L think "smoothed histogram."

Which variables might we want to include as predictors in a knn model?

→ all 3 of these, probably. still need to explore others.

### Evaluating a knn Model with the Test Data

One definition of "good" in the classification context is a model that has a high proportion of correct predictions in the test data set. This should make some intuitive sense, as we would hope that a "good" model correctly classifies most `Dark` pokemon as `Dark`, most `Fire` pokemon as `Fire`, etc.

In order to examine the performance of a particular model, we'll create a **confusion matrix** that shows the results of the model's classification on observations in the test data set. Note that in STAT 213, we didn't call this a confusion matrix; we instead called this a classification table.

**confusion matrix** - *def* - a matrix/table showing correct/incorrect classification

actual value

| | | A | B | C |
|---|---|---|---|---|
| predicted | A | 10 | 0 | 2 |
| predicted | B | 2 | 3 | 1 |
| | C | 0 | 0 | 9 |

2 C's were predicted A's

10 A's were predicted A's.

Let's fit a knn model and see the resulting confusion matrix with a training sample of **75** pokemon. Don't worry about the code to fit the model yet, but note that we've already scaled the predictor variables to be between 0 and 1 in a previous `R` chunk.

```
library(tidyverse)
set.seed(11232021) ## run this line so that you get the same
## results as I do!

train_sample_2 <- pokemon %>%
  sample_n(75)
test_sample_2 <- anti_join(pokemon, train_sample_2)

library(class)

## create a data frame that only has the predictors
## that we will use
train_small <- train_sample_2 %>% select(HP, Attack, Defense)
test_small <- test_sample_2 %>% select(HP, Attack, Defense)

## put our response variable into a vector
train_cat <- train_sample_2$Type
test_cat <- test_sample_2$Type

## fit the knn model with 9 nearest neighbors
knn_mod <- knn(train = train_small, test = test_small,
               cl = train_cat, k = 9)
knn_mod

##  [1] Fire  Fire  Fire  Fire  Dark  Fire  Fire  Fire  Fire  Fire  Fire
## [12] Fire  Fire  Fire  Fire  Fire  Fire  Fire  Steel Fire  Dark  Steel
## [23] Steel Fire  Steel Fire  Steel Fire  Fire  Fire  Fire  Fire  Fire
## [34] Fire  Fire  Fire  Fire  Fire  Ice   Fire  Steel Fire  Steel Fire
## [45] Dark
## Levels: Dark Fire Ice Steel
```

The output of `knn_mod` gives the predicted categories for the **test** sample. We can compare the predictions from the knn model with the actual pokemon `Type`s in the test sample with `table()`, which makes a confusion matrix:

```
table(knn_mod, test_cat) %>% pander()
```

| | Dark | Fire | Ice | Steel |
|---|---|---|---|---|
| **Dark** | 0 | 2 | 1 | 0 |
| **Fire** | 9 | 15 | 7 | 3 |
| **Ice** | 1 | 0 | 0 | 0 |
| **Steel** | 1 | 0 | 2 | 4 |

The columns of the confusion matrix give the actual Pokemon types in the test data `test_cat` while the rows give the predicted types from our knn model.

Interpret the value in the top-left of the confusion matrix.

0 Dark Type pokemon were actually classified as Dark

Interpret the value in the row with **Dark** and the column with **Ice**.

1 Ice Pokemon was predicted (incorrectly) to be Dark.

One common metric used to assess overall model performance is the model's **classification rate**, which is computed as the number of correct classifications divided by the total number of observations in the test data set.

Calculate our classification rate:

45 = total number in test sample.

$$\frac{0+15+0+4}{45} = \frac{19}{45} = 0.422.$$

The following is code that can automatically calculate the classification rate:

```
tab <- table(knn_mod, test_cat)
sum(diag(tab)) / sum(tab)
```

A baseline classification rate to compare to is a model that just classifies everything in the test data set as the most common `Type` in the training data set. In this case, the most common `Type` in the training data set is `Fire`, and there were 17 Fire pokemon in the test data set (out of 45 total Pokemon). So, a baseline model with no predictors has a classification rate of:

"no predictors": $\frac{17}{45} = 0.405$, barely worse than our model!

bias variance trade-off in all predictive models/algorithms.

## Choosing k

We will choose $k$, the number of neighbors considered, using a bit of trial and error as well. However, we will discuss the relative advantages of smaller and larger k values. Which value is "best" is entirely dependent on the data at hand!

What are some advantages for making k smaller?

small k: low bias, high variance

(sensitive to small changes in the training data and has a tendency to pick up on "noise")

What are some advantages for making k larger?

large k: high bias, low variance

(may not accurately predict observations in the test data and often oversimplifies the model)

We will now use R to practice fitting these knn models on this Pokemon data set.