Why do we standardize the validation set according to the training set's mean and standard deviation?

Abra Brisbin

1/29/2021

```
library(dplyr)
library(ggformula)
```

Why do we standardize the validation (test) set according to the training set's mean and standard deviation (sd)?

When we make the training/validation set split randomly (though there are reasons why we don't always do this), then typically the mean and sd will be similar in the two groups. When this happens, you'll get similar results regardless of whether you scale the validation set according to the original mean and sd of the training set (correctly) or the validation set (incorrectly).

The reason why it's important to scale the validation set based on the training set mean and sd is most apparent when the validation set happens to have a different mean than the training set. So, let's simulate a test and training set with different means. Then we can see what goes wrong when you scale the validation set according to its own mean and sd.

Simulating the raw (unscaled) data

This data is simulated to have the same standard deviation ($\sigma = 2$) for each of the predictor variables $x1$ and $x2$, because this is the case where standardization isn't really necessary– both variables are already weighted equally, based on their standard deviations, so we should get the same predicted classifications regardless of whether we scale the data or not.

```
set.seed(111)
train = cbind(rnorm(100, 2, 2), rnorm(100, 3, 2))
test = cbind(rnorm(15, 3, 2), rnorm(15, 5, 2))
combo = data.frame(rbind(train, test))
names(combo) = c("x1", "x2")
combo <- combo %>%
  mutate(group = c(rep("train",100), rep("validation",15)))
```

Add a column of random numbers to facilitate creating random classifications.

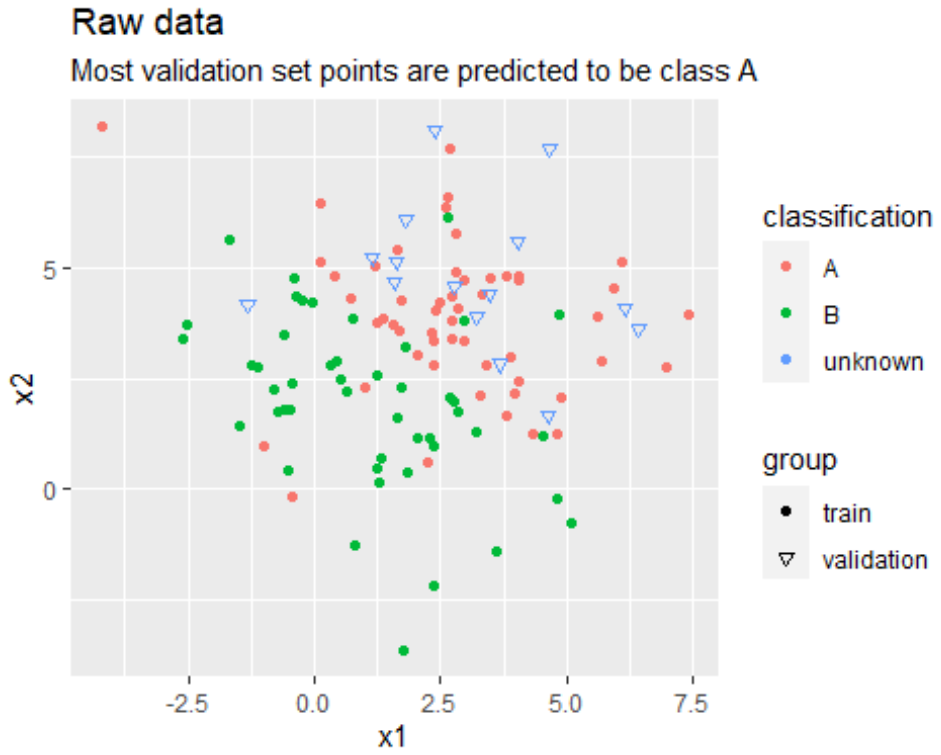
```
set.seed(111)
combo <- combo %>%
  mutate(random = runif(115))
```

```
combo <- combo %>%
  mutate(classification = case_when(group == "validation" ~ "unknown",
                                    x1 + x2 > 5 & random < .9 ~ "A",
                                    x1 + x2 <= 5 & random < .1 ~ "A",
                                    TRUE ~ "B"))
```

Graph the raw data

Notice that most of the data points in the validation set are predicted to belong to class A.

```
combo %>%
  gf_point(x2 ~ x1, shape =~ group, col =~ classification) %>%
  gf_refine(scale_shape_manual(values = c(19, 6))) %>%
  gf_labs(title = "Raw data",
          subtitle = "Most validation set points are predicted to be class
A")
```



Correctly scaling based on mean and sd of training set

Let's scale the data correctly: both the training and validation sets will be scaled based on the original mean and sd of the training set.

```
train_stats <- combo %>%
  filter(group == "train") %>%
  summarise(train_mean_x1 = mean(x1),
            train_mean_x2 = mean(x2),
```

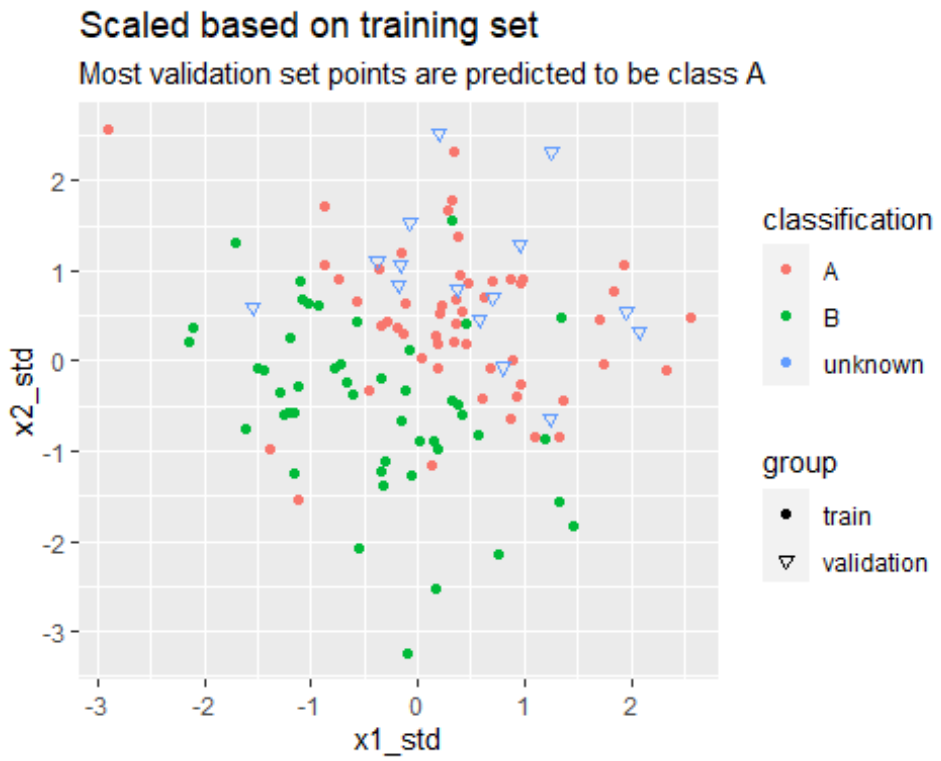
```
            train_sd_x1   = sd(x1),
            train_sd_x2   = sd(x2))
```

To make it more clear what's happening, I'm doing the scaling by explicitly subtracting the mean and dividing by the standard deviation. (This is what the `scale` function does behind the scenes.)

```
combo <- combo %>%
  mutate(x1_std = (x1-train_stats$train_mean_x1)/train_stats$train_sd_x1,
         x2_std = (x2-train_stats$train_mean_x2)/train_stats$train_sd_x2)
```

Here's what it looks like after we scale all the data points (training and validation) by the training set mean and sd. The whole data set is shifted to be centered at (0, 0), and the vertical and horizontal spread have changed, because we rescaled to have a standard deviation of 1. But the validation set points are still in the same positions relative to the training set. This version still predicts that most of the validation set belongs to class A.

```
combo %>%
  gf_point(x2_std ~ x1_std, shape =~ group, col =~ classification) %>%
  gf_refine(scale_shape_manual(values = c(19, 6))) %>%
  gf_labs(title = "Scaled based on training set",
          subtitle = "Most validation set points are predicted to be class
A")
```



Incorrectly scaling the test set based on the test set's mean and sd

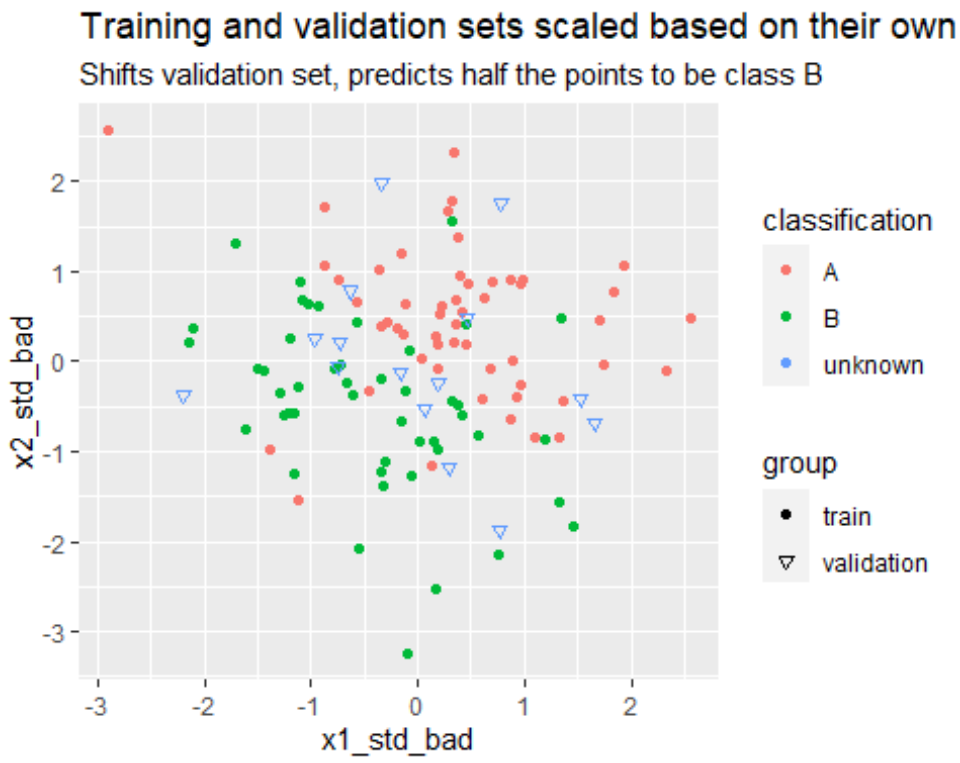
Now let's scale the data incorrectly: the training set will be scaled based on its mean and sd, and the validation set will be based on the validation set's mean and sd.

```
test_stats <- combo %>%
  filter(group == "validation") %>%
  summarise(test_mean_x1 = mean(x1),
            test_mean_x2 = mean(x2),
            test_sd_x1   = sd(x1),
            test_sd_x2   = sd(x2))

combo <- combo %>%
  mutate(x1_std_bad = case_when(group == "train" ~ x1_std,
                                TRUE ~ (x1-
test_stats$test_mean_x1)/test_stats$test_sd_x1),
         x2_std_bad = case_when(group == "train" ~ x2_std,
                                TRUE ~ (x2-
test_stats$test_mean_x2)/test_stats$test_sd_x2))
```

Here's what it looks like if we incorrectly scale the validation set according to the validation set's mean and sd. The training and validation sets have been shifted separately, so they are each centered at (0, 0). This changes the position of the validation set relative to the training set. As a result, about half the validation set is now incorrectly predicted to belong to class B.

```
combo %>%
  gf_point(x2_std_bad ~ x1_std_bad, shape =~ group, col =~ classification)
%>%
  gf_refine(scale_shape_manual(values = c(19, 6))) %>%
  gf_labs(title = "Training and validation sets scaled based on their own
mean and sd",
          subtitle = "Shifts validation set, predicts half the points to be
class B")
```


Training and validation sets scaled based on their own
Shifts validation set, predicts half the points to be class B
x2_std_bad
x1_std_bad
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
classification
A
B
unknown
group
train
validation