Omkar A. Katta February 25, 2022

Imputing Categorical Variables with SVM

PPHA 30545 Guillaume A. Pouliot

1 Primer on SVM

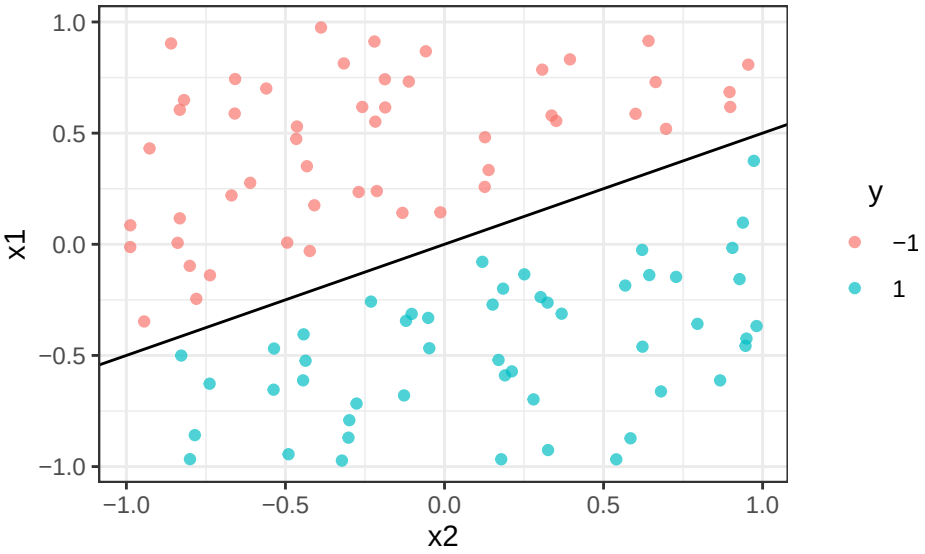
By the end of this primer, you should be able to:

- use `e1071::svm()` to fit SVM classifiers in `R`,
- compute the classification error using the output of `svm()`, and
- change the kernel and tuning parameter of SVMs.

We'll cover two examples: a simple example with linearly separable data and a slightly more complicated example that isn't linearly separable.

1.1 Example 1

Consider two features, called `x1` and `x2`, and a binary outcome, called `y`, that are visualized as follows:



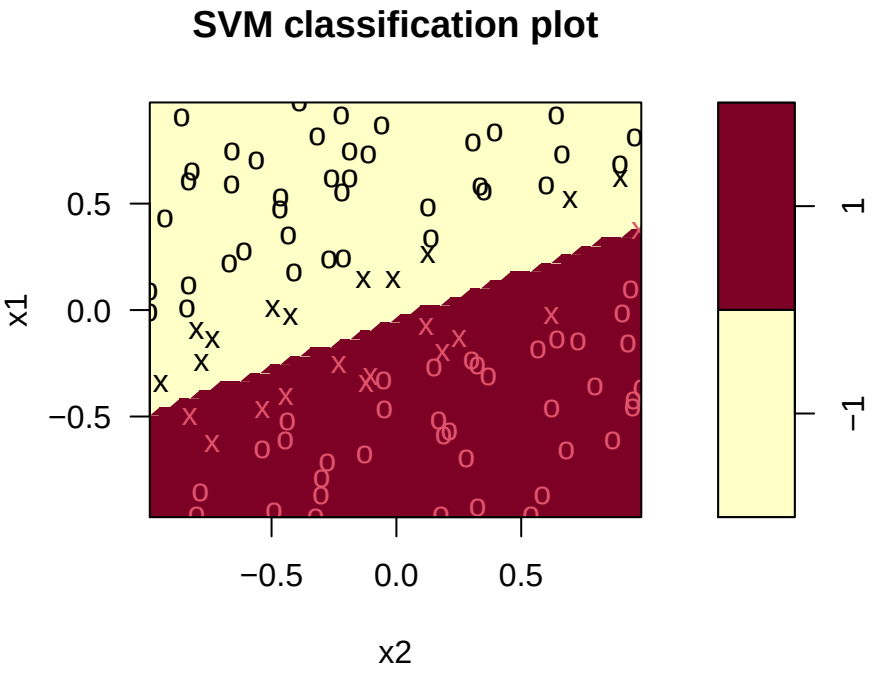
The data is linearly separable; after all, we are able to draw a 2-dimensional hyperplane (i.e., a line) that cleanly separates the +1 observations from the -1 observations. Let's fit a hyperplane that perfectly separates these points into two regions. We'll use the `e1071` package to fit SVMs in `R`; it is the same package that the authors of our textbook use (James et al., 2013). The workhorse function is `svm`.

```r
# install.packages("e1071") # Install the package if you don't already have it.
library(e1071)
# read the documentation: help(svm)
svmfit <- svm(as.factor(y) ~ ., # the dot => all columns except y are regressors
              data = df,
              kernel = "linear",
              scale = FALSE)
```

We know from the plot that a linear kernel will suffice to separate the observations, so we set `kernel = "linear"`. By default, `svm` will rescale the variables, so we have to set `scale = FALSE` if we don't want this behaviour.

One of the coolest abilities of `svm` is that for 2-dimensional feature space, it can plot the results:

```r
plot(svmfit, data = df)
```



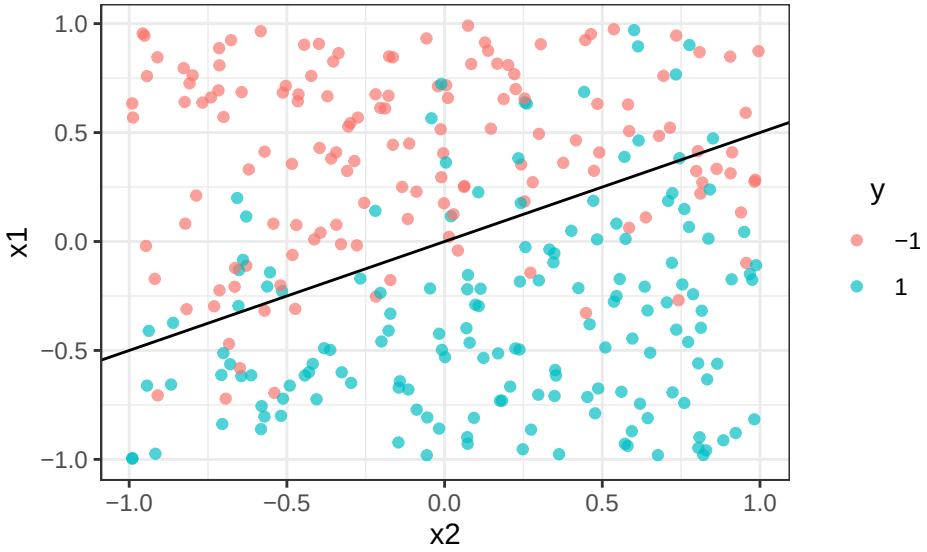
Notice that all the black points are on one side of the hyperplane and the red points are on the other. Thus, our classification error is 0, i.e., we perfectly classify our data.

Exercise 1.1. Install and load the `e1071` package. Using the documentation of the `svm` and `plot.svm` (run `help(svm)` and `help(plot.svm)`), answer the following questions:

a. What are the four kernels supported by `svm`?
b. What is the default value of the `cost` parameter? (The `cost` is the tuning parameter that controls how much you are willing to tolerate violations of the margins.)
c. In the plot above, what are the X's? Hint: look up the `svSymbol` in `help(plot.svm)`.

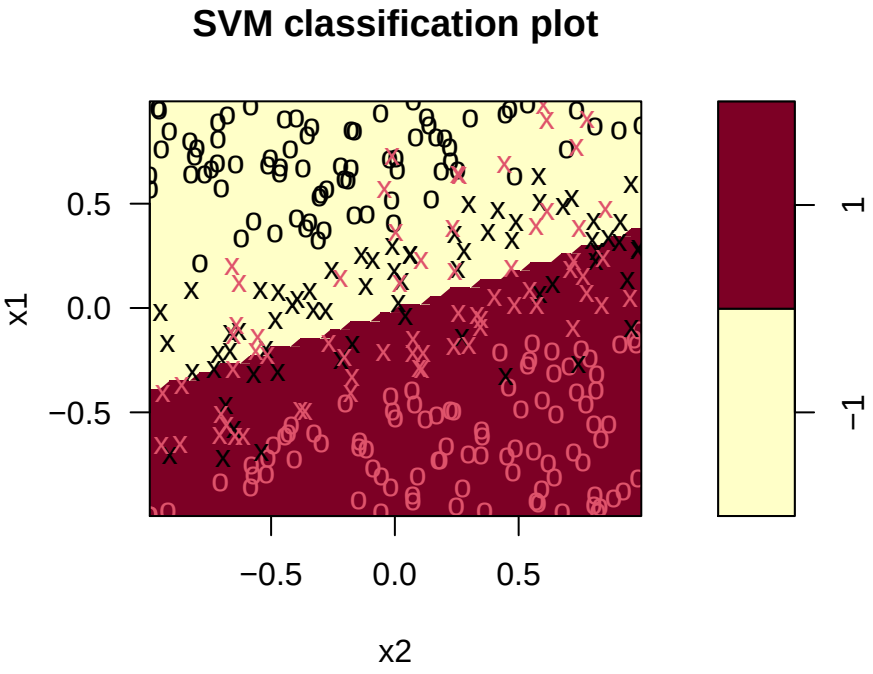
1.2 Example 2

Now suppose we have data that looks like this:



Is the data linearly separable? No, not quite. Let's see what happens when we try and fit an SVM classifier as above on this data:

```
svmfit_naive <- svm(as.factor(y) ~ .,
                    data = df_noisy,
                    kernel = "linear",
                    scale = FALSE)
plot(svmfit_naive, data = df_noisy)
```

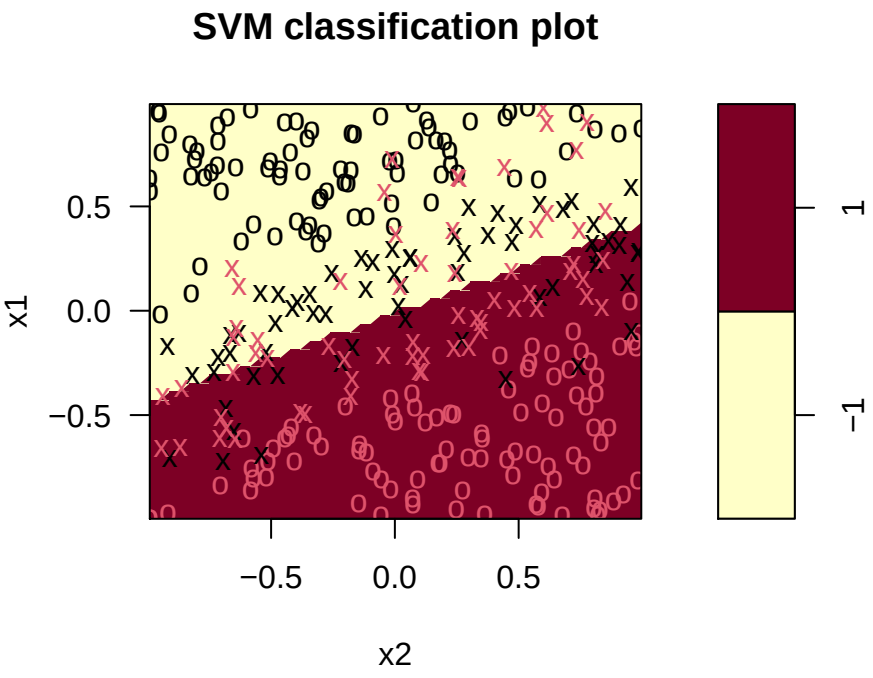


Inevitably, we misclassify the points near the separating hyperplane. Let's quantify the classification error. Using `predict`, we can obtain the predicted values of the SVM on the data we used to fit the model. Then, we can compare the predictions with the truth:

```
naive_prediction <- predict(svmfit_naive, newdata = df_noisy[, c("x1", "x2")])
naive_error <- sum(naive_prediction != df_noisy$y) / length(naive_prediction)
print(naive_error)
#> [1] 0.17
```

The misclassification error is 17%. We can do better by using a soft margin and allow some observations to be within the margins of the hyperplane. This is controlled by the `cost` parameter. Let's try increasing it to 5:

```
svmfit_soft <- svm(as.factor(y) ~ .,
                   data = df_noisy,
                   cost = 5,
                   kernel = "linear",
                   scale = FALSE)
plot(svmfit_soft, data = df_noisy)
```




```
soft_prediction <- predict(svmfit_soft, newdata = df_noisy[, c("x1", "x2")])
soft_error <- sum(soft_prediction != df_noisy$y) / length(soft_prediction)
print(soft_error)
#> [1] 0.1633333
```

Our classification error is now 16%, which is smaller than before! Is 5 the best choice for the `cost`? Well, what we care about as aspiring machine learning experts is not the in-sample classification error;

it is the out-of-sample error. We want to choose the SVM model that minimizes the test error. This means that we'll need to do some cross-validation to choose our tuning parameter. Rather than do this here, we'll return to cross-validation when studying the empirical application of SVMs (see Exercise 2.2).

In the following empirical application, you'll be training and using your own SVMs. The code above will be a useful reference for you, so you may find yourself flipping back to it.

2 Empirical Application

You are tasked with answering the following question: *does having flexible work hours impact whether people vote?* At your disposal are two datasets called `vote`[1] and `work`[2]:

```
vote_df <- read.csv("vote.csv")
work_df <- read.csv("work.csv")
```

The two datasets share the following *core variables*:

- `prtage`: age
- `pesex`: sex
- `ptdtrace`: race
- `pehspnon`: Hispanic or Non-Hispanic status
- `prcitshp`: U.S. citizenship status
- `peeduca`: highest level of schooling

The `vote_df` dataset has a synthetic binary variable, which is appropriately called `vote`, that indicates the voting status of each individual in the data. Meanwhile, the `work_df` dataset has its own synthetic binary variable called `work` that indicates whether individuals have flexible work schedules.

Individuals in one dataset are presumably different than the individuals in the other dataset. As a result, for any individual in our data, we will either know their voting status or their work schedule, but we cannot know both simultaneously. Hence, we have a missing data problem.

The plan to overcome this challenge will be as follows:

1. Train a support vector machine classifier on `work_df` that uses the demographic variables to forecast whether someone has flexible work hours.
2. Apply the SVM classifier from the previous step to `vote_df` to predict whether the people in this dataset have flexible work hours.
3. Regress voting status on the predictions obtained in the previous step.

[1] This semi-synthetic dataset was created for pedagogical reasons, and any results using this dataset are not credible reflections of voting status. This dataset was based on the CPS Voting and Registration Supplement (U.S. Census Bureau & U.S. Bureau of Labor Statistics, 2004b).

[2] This semi-synthetic dataset was created for pedagogical reasons, and any results using this dataset are not credible reflections of work schedules. This dataset was based on the CPS Work Schedules Supplement (U.S. Census Bureau & U.S. Bureau of Labor Statistics, 2004a).

But before anything else, we must clean and explore our data a bit.

2.1 Clean the Data

Notice that all the variables in our dataset except for `prtage` are categorical, meaning that they take discrete values as opposed continuous values. When working with categorical variables, `e1071::svm()` prefers them to be factors[3]. However, these variables are `character` variables. To confirm this, we can look at the structure of our datasets and identify the `chr` abbreviation for the variable class:

```
apply(vote_df, 2, class)
#>      prtage       pesex    ptdtrace    pehspnon    prcitshp     peeduca
#> "character" "character" "character" "character" "character" "character"
#>        vote
#> "character"
apply(work_df, 2, class)
#>      prtage       pesex    ptdtrace    pehspnon    prcitshp     peeduca
#> "character" "character" "character" "character" "character" "character"
#>        work
#> "character"
```

Our focus for the moment will be to convert these variables into factors. The easiest way might be to use `as.factor()`:

```
vote_demo <- vote_df
work_demo <- work_df
vote_demo$prcitshp <- as.factor(vote_demo$prcitshp)
work_demo$prcitshp <- as.factor(work_demo$prcitshp)
```

Let's confirm that the citizenship status variable is indeed a factor using `str()`:

```
str(vote_demo$prcitshp)
#>  Factor w/ 4 levels "FOREIGN BORN, U.S. CITIZEN BY",..: 4 4 1 4 4 4 4 4 4 4 ...
str(work_demo$prcitshp)
#>  Factor w/ 5 levels "FOREIGN BORN, NOT A CITIZEN OF",..: 5 5 5 5 5 5 5 5 5 5 ...
```

These variables are indeed factors, but `vote_demo$prcitshp` has 4 levels whereas `work_demo$prcitship` has 5. Ideally, we want each core variable, including `prcitshp`, between our two datasets to have the same exact structure. So, let's try again:

```
prcitshp_unique <- unique(c(vote_df$prcitshp, work_df$prcitshp))
vote_df$prcitshp <- factor(vote_df$prcitshp, levels = prcitshp_unique)
work_df$prcitshp <- factor(work_df$prcitshp, levels = prcitshp_unique)
str(vote_df$prcitshp)
#>  Factor w/ 5 levels "NATIVE, BORN IN THE UNITED",..: 1 1 2 1 1 1 1 1 1 1 ...
```

[3]Consult Hadley Wikham's R for Data Science to learn about factors (Wikham & Grolemund, n.d.).

```
str(work_df$prcitshp)
#>  Factor w/ 5 levels "NATIVE, BORN IN THE UNITED",..: 1 1 1 1 1 1 1 1 1 1 ...
```

By specifying the unique values of `prcitship` across both datasets as the levels, we ensure that `vote_df$prcitshp` and `work_df$prcitshp` are indeed of the same structure.

Exercise 2.1. Finish cleaning the datasets by (a) repeating the cleaning process above for all the categorical core variables in `vote` and `work`, not just `prcitshp`, and (b) converting the `prtage` from a `character` variable to an `integer` variable with the `as.integer` function. Ensure that the core variables have the same structure between the `vote` and `work` datasets.

2.2 Train SVM with Cross-Validation

Now that our datasets are set up, we can now train a support vector machine classifier. There are many choices to make when fitting the SVM: the cost and the kernel. To determine which cost and kernel to use, let's use 5-fold cross-validation. Here is an example:

```
cv_svm <- function(k, data, ...) {
  # randomly assign each observation to a fold ---
  initialization <- rep(seq_len(k), nrow(data))
  shuffle <- sample(seq_len(nrow(data)), nrow(data))
  fold_label <- initialization[shuffle]
  # compute the error for each validation set ---
  error <- vector("double", k)
  for (i in seq_len(k)) {
    hold_out <- fold_label == i
    train <- data[!hold_out, ] # create training set
    test <- data[hold_out, ] # create validation set
    # fit the candidate SVM on the training set
    svm_kfold <- svm(work ~ .,
                     data = train,
                     ...)
    predict_kfold <- predict(svm_kfold,
                             newdata = test[, !(names(test) %in% c("work"))])
    # compute classification error
    error[i] <- sum(predict_kfold != test[, "work"]) / length(predict_kfold)
  }
  # compute the mean error across the validation sets ---
  mean(error)
}

cost_values <- c(1, 5)
```

```
kernel_values <- c("linear", "sigmoid")
models <- expand.grid(cost = cost_values, kernel = kernel_values)
models$error <- NA_real_
for (cost_candidate in cost_values) {
  for (kernel_candidate in kernel_values) {
    # run 5-fold cross-validation for each model
    fold_error <- cv_svm(k = 5,
                         data = work_df,
                         scale = FALSE,
                         cost = cost_candidate,
                         kernel = kernel_candidate)
    # store the cross-validation error in a data frame
    models[models$cost == cost_candidate &
           models$kernel == kernel_candidate, "error"] <- fold_error
  }
}
print(models)
#>   cost  kernel  error
#> 1    1  linear 0.1388
#> 2    5  linear 0.1402
#> 3    1 sigmoid 0.3944
#> 4    5 sigmoid 0.3944
(cv_cost <- models[which.min(models$error), "cost"])
#> [1] 1
(cv_kernel <- models[which.min(models$error), "kernel"])
#> [1] linear
#> Levels: linear sigmoid
(cv_error <- models[which.min(models$error), "error"])
#> [1] 0.1388
```

Let's break down this code. I define a function called `cv_svm`. This function arbitrarily assigns each observation in our dataset to a group, with `k` being the total number of groups. Then, it iterates through the different groups. In each iteration, it chooses one of the groups to be the validation set and the rest of the groups to be the training set. It fits the candidate model in question on the training set. Then, it applies the fitted model to the validation set and computes the classification error – we will have one classification error for each model. Finally, it will average the classification errors and return the cross-validation classification error rate for that candidate model.

After defining this function, I consider two different values of the cost. For each cost value, I consider two different kernels. Hence, I consider 4 models in total. Iterating across the different models, I can use the `cv_svm()` function to compute its cross-validation error. Finally, I can decide which cost and

kernel to go with by choosing the model with the smallest cross-validation error rate.

In this case, an SVM with a linear kernel and a cost of 1 minimizes the 5-fold cross-validation error rate among the models considered. Note that if you run the above code chunk, it may take a few minutes.

Exercise 2.2. Consider three values of the cost: 1, 5, and 10. Consider two kernels: "linear" and "sigmoid." Report the cross-validation error rates of all 6 SVM models. Then, pick the cost and kernel that minimizes the 5-fold cross-validation. Use this model for the rest of the exercises.

Now that we have chosen a kernel and cost, let's fit the SVM model on `work` and compute the classification error.

```
svmfit <- svm(work ~ ., data = work_df, scale = FALSE,
              cost = cv_cost, kernel = cv_kernel)
predict_svmfit <- predict(svmfit,
                          newdata = work_df[, !(names(work_df) %in% c("work"))])
(error <- sum(predict_svmfit != work_df[, "work"]) / length(predict_svmfit))
#> [1] 0.134
```

Exercise 2.3. What is the classification error of the model that you decided from Exercise 2.2 when fitting it to `work_df`?

2.3 Impute Work Schedule

With the SVM model that we fit on `work_df`, we'll now impute the work schedules using the core variables of `vote_df`.

```
impute_work <- predict(svmfit,
                       newdata = vote_df[, !(names(vote_df) %in% c("vote"))])
```

Exercise 2.4. Replicate the above code chunk for the model that you fit in Exercise 2.3. The result should be the imputed work schedules.

2.4 Regress Voting Status on Imputed Work Schedules

The last step of our analysis is to regress voting status on imputed work schedules. Unlike `svm()`, the `lm()` function isn't particularly amenable to factor variables as the outcome variable and the main variable of interest. So let's first convert the variables of interest into numeric variables before feeding them into `lm()`.

Also, since the data is semi-synthetic, I know which regressors are most important and which regressors aren't. In particular, sex and a second-degree polynomial of age were used to construct the voting status. Using this priveleged knowledge (which we would never have in practice), we can come up with an estimate for the relationship between flexible work schedules and voter turnout.

```r
work_numeric <- as.numeric(impute_work == "flexible")
vote_numeric <- as.numeric(vote_df$vote == "vote")
reg <- lm(vote_numeric ~ work_numeric + poly(prtage, 2) + pesex, data = vote_df)
stargazer::stargazer(reg, single.row = TRUE, header = FALSE)
```

Table 1:

	Dependent variable:
	vote_numeric
work_numeric	0.320*** (0.017)
poly(prtage, 2)1	−16.319*** (0.597)
poly(prtage, 2)2	1.754*** (0.334)
pesexMALE	0.015 (0.009)
Constant	0.344*** (0.011)
Observations	5,000
R^2	0.567
Adjusted R^2	0.567
Residual Std. Error	0.329 (df = 4995)
F Statistic	1,634.503*** (df = 4; 4995)
Note:	*p<0.1; **p<0.05; ***p<0.01

```r
work_vote_relationship <- coef(reg)["work_numeric"]
```

It turns out there is a positive and statistically significant relationship between having flexible work schedules and voting. Note that this data is semi-synthetic, so these results shouldn't be treated as credible.

Exercise 2.5. Regress the voting status on imputed work schedules. Use age, squared age, and sex as regressors in addition to the imputed work schedule. Be sure to convert the variables to an appropriate format. Interpret and discuss the results.

2.5 Bias Correction

Since we imputed the work schedules, there are likely to be some forecasts that are incorrect. To correct for the attenuation bias, we will need to divide our estimate by the following:

$$M = \frac{1}{1-2b}\left(1 - \frac{(1-b)b}{a} - \frac{(1-b)b}{1-a}\right), \text{ where}$$
$$a = \mathbb{P}(\texttt{impute\_work == ``flexible''}), \text{ and}$$
$$b = \mathbb{P}(\texttt{impute\_work} \text{ is incorrectly labeled}).$$

We can write a simple function to compute M as follows:

```r
compute_M <- function(a, b) {
    1 / (1 - 2 * b) * (1 - (1 - b) * b / a - (1 - b) * b / (1 - a))
}
```

Now, all we need is a and b. To compute a, we find the proportion of imputed work schedules that flexible. To compute b, we find use the cross-validation error rate from Exercise 2.2.

```r
(a <- sum(impute_work == "flexible") / length(impute_work))
#> [1] 0.5488
(b <- cv_error)
#> [1] 0.1388
(M <- compute_M(a, b))
#> [1] 0.7160345
```

Finally, we divide our naive answer by M to get our bias-corrected result:

```r
(work_vote_bias_correction <- work_vote_relationship / M)
#> work_numeric
#>    0.4465501
```

Exercise 2.6. Correct for the attenuation bias in your results from Exercise 2.5. Is the bias-corrected version larger or smaller? Does the bias-correction change your results from earlier?

2.6 Creating Bootstrap Confidence Intervals

To obtain confidence intervals, we can bootstrap the data by following these steps:

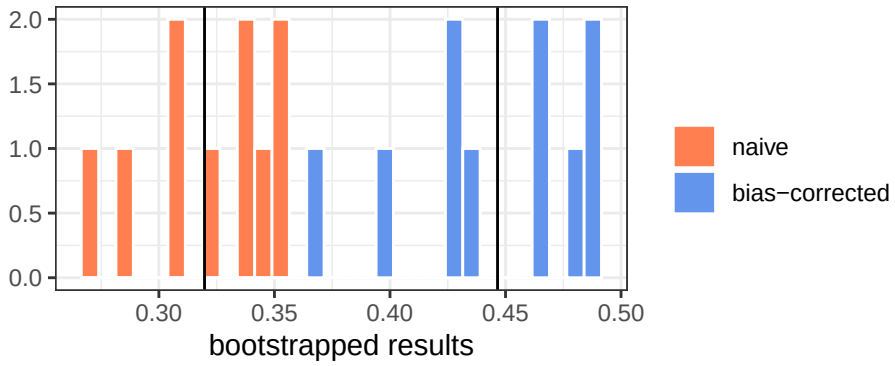
1. Obtain a bootstrap sample of the `work` dataset.
2. Train the SVM on the bootstrap sample.
3. Impute the work schedules using the `vote` dataset.
4. Bootstrap the `vote` dataset *with* the imputed work schedules.
5. Regress voting status on imputed work schedules using the bootstrap sample.
6. Correct for attenuation bias.

These 6 steps create one bootstrapped estimate. We can repeat this process as many times as we'd like to create a bootstrapped distribution of estimates. We then use these estimates to create a 95% confidence interval. Below, we demonstrate how to create 5 bootstrapped estimates and visualize the results:

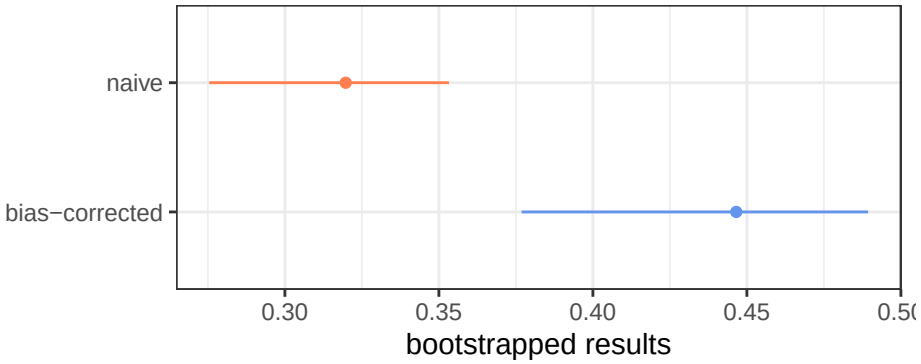
```r
num_bootstrap <- 10
bootstrap_work_vote <- vector("double", num_bootstrap)
bootstrap_work_vote_corrected <- vector("double", num_bootstrap)
for (i in seq_len(num_bootstrap)) {
  bootstrap_index <- sample(nrow(work_df), nrow(work_df), replace = TRUE)
```

```
  work_df_bootstrap <- work_df[bootstrap_index, ]
  svm_bootstrap <- svm(work ~ .,
                       data = work_df_bootstrap,
                       scale = FALSE,
                       cost = cv_cost,
                       kernel = cv_kernel)
  impute_bootstrap <- predict(svm_bootstrap,
                              newdata = vote_df[!(names(vote_df) %in% c("vote"))])
  a_bootstrap <- sum(impute_bootstrap == "flexible") / length(impute_bootstrap)
  b_bootstrap <- cv_svm(k = 5,
                        data = work_df_bootstrap,
                        scale = FALSE,
                        cost = cv_cost,
                        kernel = cv_kernel)
  impute_bootstrap_numeric <- as.numeric(impute_bootstrap == "flexible")
  bootstrap_index <- sample(nrow(vote_df), nrow(vote_df), replace = TRUE)
  vote_df_bootstrap <- vote_df[bootstrap_index, ]
  vote_bootstrap <- vote_numeric[bootstrap_index]
  work_bootstrap <- impute_bootstrap_numeric[bootstrap_index]
  reg_bootstrap <- lm(vote_bootstrap ~ work_bootstrap + poly(prtage, 2) + pesex,
                      data = vote_df_bootstrap)
  bootstrap_work_vote[i] <- coef(reg_bootstrap)["work_bootstrap"]
  M_bootstrap <- compute_M(a_bootstrap, b_bootstrap)
  bootstrap_work_vote_corrected[i] <- bootstrap_work_vote[i] / M_bootstrap
}
```

```
ggplot() +
  geom_histogram(aes(x = bootstrap_work_vote, fill = "0"), color = "white") +
  geom_histogram(aes(x = bootstrap_work_vote_corrected, fill = "1"), color = "white") +
  scale_fill_manual(labels = c("naive", "bias-corrected"),
                    values = c("coral", "cornflowerblue"),
                    name = "") +
  geom_vline(xintercept = work_vote_relationship) +
  geom_vline(xintercept = work_vote_bias_correction) +
  xlab("bootstrapped results") +
  ylab("")
```




```
# lower/upper bound of 95% confidence interval
naive_ci <- quantile(bootstrap_work_vote, prob = c(0.025, 0.975))
corrected_ci <- quantile(bootstrap_work_vote_corrected, prob = c(0.025, 0.975))
ggplot() +
  geom_segment(aes(x = naive_ci["2.5%"], xend = naive_ci["97.5%"],
                   y = "naive", yend = "naive", color = "0")) +
  geom_point(aes(x = work_vote_relationship, y = "naive", color = "0")) +
  geom_segment(aes(x = corrected_ci["2.5%"], xend = corrected_ci["97.5%"],
                   y = "bias-corrected", yend = "bias-corrected", color = "1")) +
  geom_point(aes(x = work_vote_bias_correction, y = "bias-corrected", color = "1")) +
  scale_color_manual(labels = c("naive", "bias-corrected"),
                     values = c("coral", "cornflowerblue")) +
  xlab("bootstrapped results") +
  ylab("") +
  theme(legend.position = "none")
```




```
# standard errors
sd(bootstrap_work_vote)
#> [1] 0.02824501
sd(bootstrap_work_vote_corrected)
#> [1] 0.04004298
```

Exercise 2.7. Create 95% confidence intervals via 50 bootstrap samples. Visualize the naive and bias-corrected estimates from Exercises 2.5 and 2.6 along with their confidence intervals. Are the bootstrapped standard errors larger or smaller than the ones suggested by the regression output in Exercise 2.5? Note: if you use the code above as a reference, it may take a few minutes for the code to finish.

References

10 James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

U.S. Census Bureau, & U.S. Bureau of Labor Statistics (Eds.). (2004a). *Current population survey: Work schedules and work at home supplement* [Data set]. https://www.nber.org/research/data/current-population-survey-cps-supplements-work-schedules

U.S. Census Bureau, & U.S. Bureau of Labor Statistics (Eds.). (2004b). *Current population survey: Voting and registration supplement* [Data set]. https://www.nber.org/research/data/current-population-survey-cps-supplements-voting-and-registration

Wikham, H., & Grolemund, G. (n.d.). *R for data science*. https://r4ds.had.co.nz/index.html