Omkar A. Katta February 28, 2021

Imputing Categorical Variables with SVM

PPHA 30545 Guillaume A. Pouliot

1 Primer on SVM

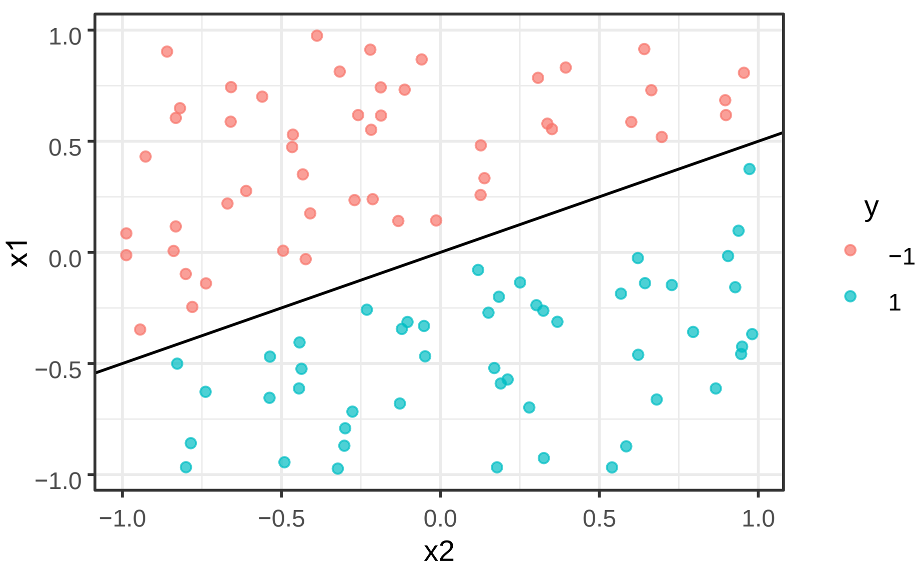
By the end of this primer, you should be able to:

- use `sklearn.svm.SVC()` to fit SVM classifiers in `Python`,
- compute the classification error, and
- change the kernel and tuning parameter of SVCs.

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 svm package from `sklearn` to fit SVCs. The workhorse function is `svc`.

```
import numpy as np
from sklearn import datasets
# pip install scikit-learn # Install the package if you don't already have it.
from sklearn.svm import SVC

iris = datasets.load_iris()
# Take the first two features only (Sepal Length and Sepal Width)
# Limiting are dataset to 2 classes (Setosa, Versicolor)
X = iris.data[:100, :2]
y = iris.target[:100]

model = SVC(C = 0.1, kernel = 'linear').fit(X,y)
```

We know from the plot that a linear kernel will suffice to separate the observations, so we set `kernel = "linear"`. Read the documentation: scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

We can plot the results of `SVC` for 2-dimensional feature space using the commands below:

```
plot_svm(model, X, y, 'Sepal Length', 'Sepal Width')
```

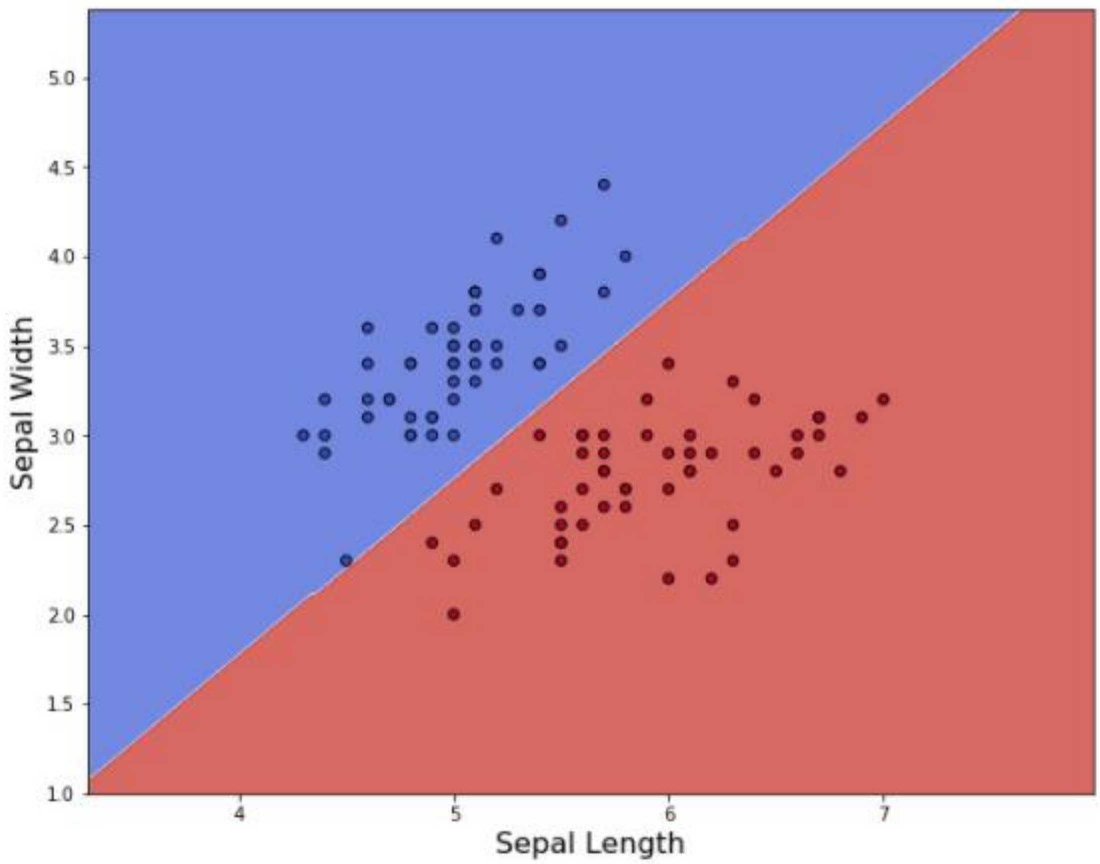
However, the plot_svm() function is a custom function created by us. You do no need to memorize or understand how it works. However, feel free to have a look at the code below. Make sure to run the function below before calling the plot_svm() above.

```
def plot_svm(model, X, y, x_label=None, y_label=None):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
      np.arange(y_min, y_max, .02))
    Z = model.predict(np.c_[xx.ravel(),
      yy.ravel()]).reshape(xx.shape)

    plt.figure(figsize=(10,8))
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm,      alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap =    plt.cm.coolwarm,
s=30, edgecolors='k')
```

```
plt.xlabel(x_label, fontsize= 16)
plt.ylabel(y_label, fontsize= 16)
```

Running plot_svm() will generate the following plot:



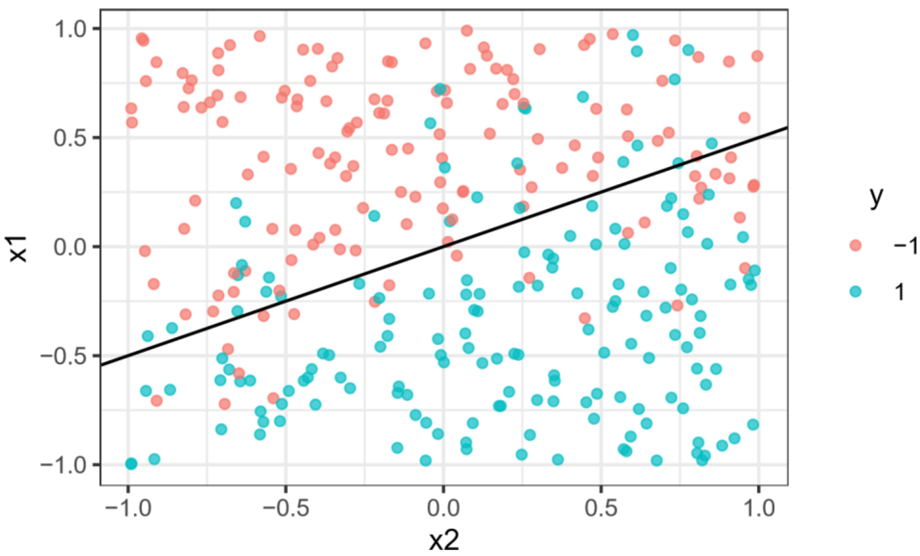
Notice that all the red points (Versicolor class) are on one side of the hyperplane and the blue points (Setosa class) are on the other. Thus, our classification error is 0, i.e., we perfectly classify our data.

Exercise 1.1. Import the `sklearn.svm` package. Using the documentation of the `svc (link given above)`, answer the following questions:

a. What are the kernels supported by `svc in sklearn`?
b. What is the default value of the `Regularization` parameter C?
c. What can you explain about the degree parameter?

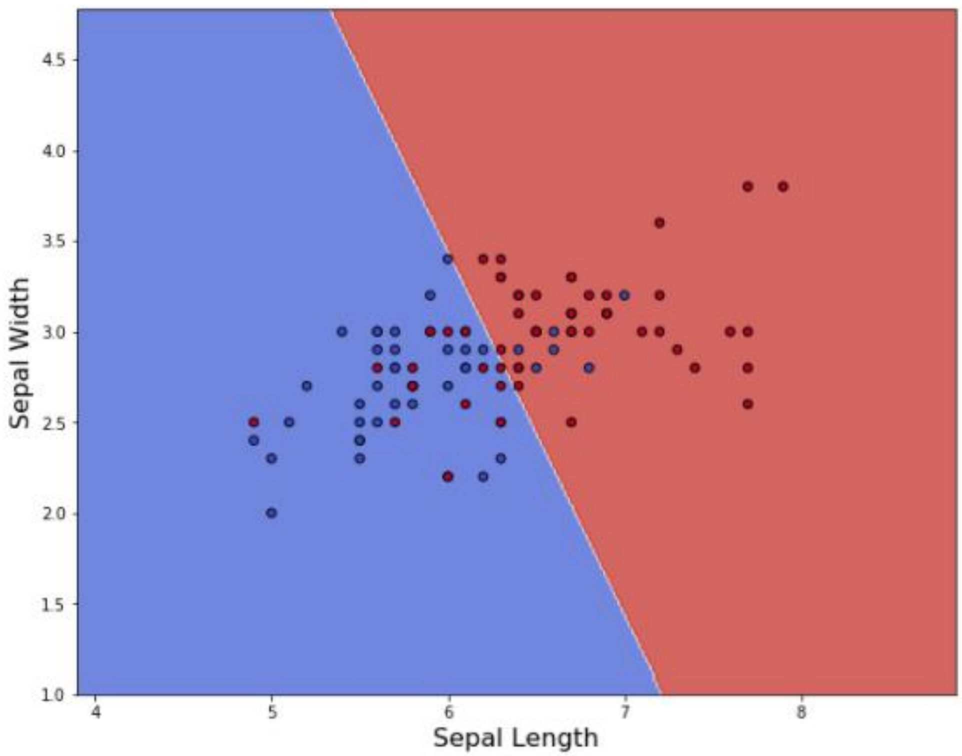
1.2 Example 2

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:

```
X = iris.data[50:150, :2]
y = iris.target[50:150]
model = SVC(C = 0.1, kernel='linear').fit(X,y)
plot_svm(model, X, y, 'Sepal Length', 'Sepal Width')
```

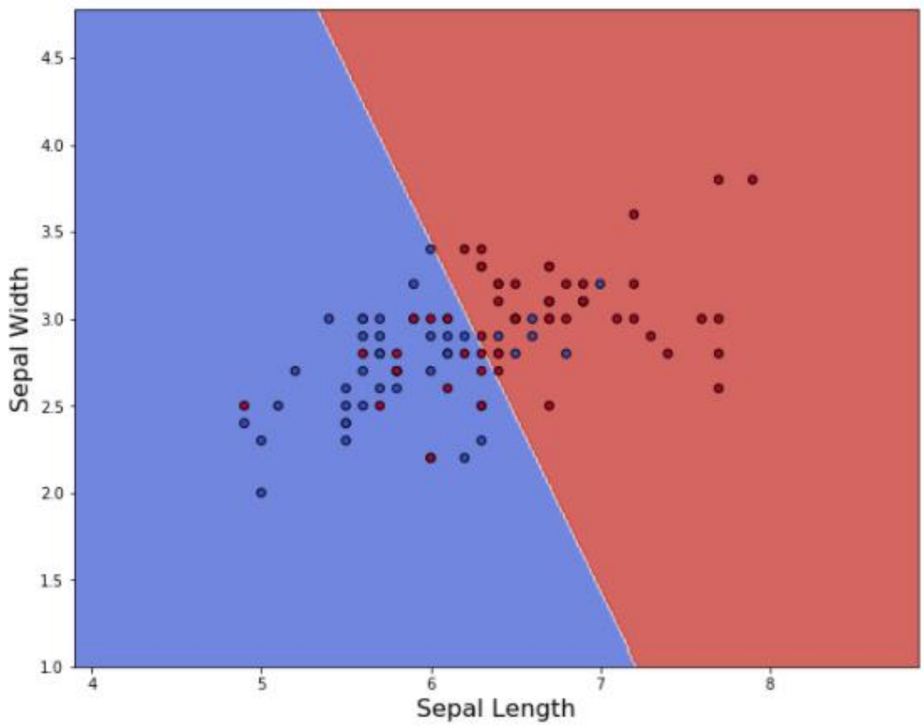


Inevitably, we misclassify the points near the separating hyperplane. Let's quantify the classification error. Using `score`, we can obtain the mean accuracy of the SVM on the data we used to fit the model:

```
print(1 - model.score(X, y))
# 0.3
```

The misclassification error is 30%. 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 `regularization` parameter C.

```
model = SVC(C = 1, kernel='linear').fit(X,y)
plot_svm(model, X, y, 'Sepal Length', 'Sepal Width')
```




```
print(1 - model.score(X, y))
# 0.27
```

Our classification error is now 27%, which is smaller than before! Is 1 the best choice for `C`? 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]:

```
# import data
vote_df = pd.read_csv("vote.csv")
work_df = pd.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, 2004a).

[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, 2004b).

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. SVC() in Sklearn only accepts numerical values for training. However, all these, except prtage, are `object` variables.

```
vote_df.dtypes

# prtage        int64
# pesex        object
# ptdtrace     object
# pehspnon     object
# prcitshp     object
# peeduca      object
# vote         object
```

```
work_df.dtypes

# prtage        int64
# pesex        object
# ptdtrace     object
# pehspnon     object
# prcitshp     object
# peeduca      object
# work         object
```

Our focus for the moment will be to convert these variables into some sort of integer forms. One way to do this is to use encoding technique. We'll first convert our target variables to binary forms by mapping them to 0s and 1s:

```
# Map labels to 0 and 1
work_mapper = {'flexible': 1, "not flexible": 0}
work_df['work'] = work_df['work'].replace(work_mapper)
```

We'll convert the categorical variables through an encoding technique called 'One-hot encoding' by converting it to multiple binary columns using pandas get_dummies. Here's how we do it with citizenship status variable 'prcitshp':

```
work_df = pd.get_dummies(work_df, columns=['prcitshp'])
work_df.columns
# Index(['prtage', 'pesex', 'ptdtrace', 'pehspnon', 'peeduca',
'work',
#         'prcitshp_FOREIGN BORN, NOT A CITIZEN OF',
#         'prcitshp_FOREIGN BORN, U.S. CITIZEN BY',
#         'prcitshp_NATIVE, BORN ABROAD OF',
#         'prcitshp_NATIVE, BORN IN PUERTO RICO OR',
#         'prcitshp_NATIVE, BORN IN THE UNITED'],
#        dtype='object')
```

However, `vote_df['prcitshp']` has 4 levels whereas `work_df['prcitship']` has 5. Ideally, we want each core variable, including `prcitshp`, between our two datasets to have the same exact structure. So we'll add a column of 0s for the fifth category after applying get_dummies to `vote_df['prcitshp']`:

```
# 'FOREIGN BORN, NOT A CITIZEN OF' category is missing in
# vote_df['prcitshp']
vote_df = pd.get_dummies(vote_df, columns=['peeduca'])
vote_df["prcitshp_FOREIGN BORN, NOT A CITIZEN OF"] = 0
```

```
work_df = pd.get_dummies(work_df, columns=['ptdtrace'])
vote_df = pd.get_dummies(vote_df, columns=['ptdtrace'])

# categories missing in vote_df and work_df
# vote_df - '4 or 5 Races'
# work_df - 'W-B-AI'
# work_df - 'W-A-HP'
# work_df - 'Black-Asian'
```

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 variables in `vote` and `work`, not just `prcitshp`, and (b) converting the `target` variable of vote to binary values 0 and 1 as done for work. 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 C and kernel to use, let's use 5-fold cross-validation. Here is an example:

```
X = work_df.drop('work', axis = 1)
y = work_df['work']

# Define the set of parameters to form combinations in grid search
parameters = {'kernel':('linear', 'sigmoid'),
              'C':[0.1, 1, 10]}

# Run each combination using GridSearch CV
# By specifying cv = 5, GridSearchCV will run
# 5-fold CV on each combination
from sklearn.model_selection import GridSearchCV
svc = SVC()
clf = GridSearchCV(svc, parameters, cv = 5)
clf.fit(X, y)

# Print out the mean_score for each combination
# GridSearchCV uses accuracy scores by default
for params, mean_score, rank in zip(clf.cv_results_["params"],
clf.cv_results_["mean_test_score"],
clf.cv_results_["rank_test_score"]):
    print(params, mean_score, rank)
```

Let's break down this code. We define a dictionary of parameters. We then use GridSearchCV to iterate through all possible combinations of kernel and Regularization parameter C. Within each iteration, it cross-validates and further iterates to choose 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 accuracy score. Finally, it will average the accuracy score from each step of cross-validation and return the cross-validation mean accuracy score for that candidate model.

In the parameters we have considered 3 different values of C and 2 for kernels. Hence, we consider 6 models in total. Finally, we can decide which C and kernel to go with by choosing the model with the highest accuracy score.

In this case, an SVM with a linear kernel and a cost of 0.1 maximizes the 5-fold cross-validation accuracy score among the models considered. Note that if you run the above code chunk, it may take a few minutes.

Exercise 2.2. Consider 4 values of the C: 0.1, 1, 5 and 10. Consider three kernels: "linear", "poly" and "sigmoid." Report the cross-validation error rates of all 12 SVM models. Then, pick and report the C and kernel that maximizes the 5-fold cross-validation. Use this model for the rest of the exercises.

Exercise 2.3. What is the accuracy score 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 = clf.predict(vote_X)
```

Exercise 2.4. Replicate the above code chunk [that predicts work status in the voting data] for the model that you fit in Exercise 2.3. The result should be the imputed work schedules and answer if accuracy score is applicable or not in this case. If not, explain why.

2.4 Regress Voting Status on Imputed Work Schedules

The last step of our analysis is to regress voting status on imputed work schedules. 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 privileged 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.

```
import statsmodels.formula.api as smf
result = smf.ols('vote ~  impute_work + prtage + np.power(prtage,
2) + pesex_FEMALE', data = vote_df).fit()
print(result.summary())
work_vote_relationship = result.params[1]
#0.3355
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   vote   R-squared:                       0.570
Model:                            OLS   Adj. R-squared:                  0.570
Method:                 Least Squares   F-statistic:                     1655.
Date:                Fri, 12 Mar 2021   Prob (F-statistic):               0.00
Time:                        00:46:45   Log-Likelihood:                -1512.8
No. Observations:                5000   AIC:                             3036.
Df Residuals:                    4995   BIC:                             3068.
Df Model:                           4
Covariance Type:            nonrobust
=========================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Intercept               1.0793      0.046     23.518      0.000       0.989       1.169
impute_work             0.3355      0.017     19.750      0.000       0.302       0.369
prtage                 -0.0198      0.002    -12.890      0.000      -0.023      -0.017
np.power(prtage, 2)  7.208e-05   1.43e-05      5.046      0.000    4.41e-05       0.000
pesex_FEMALE           -0.0151      0.009     -1.626      0.104      -0.033       0.003
==============================================================================
Omnibus:                      191.265   Durbin-Watson:                   1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              413.479
Skew:                          -0.245   Prob(JB):                     1.64e-90
Kurtosis:                       4.321   Cond. No.                     3.01e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.01e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

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:

$$M = \frac{1}{1-2b}\left(1 - \frac{(1-b)b}{a} - \frac{(1-b)b}{1-a}\right), \text{ where}$$

$a = \mathbb{P}$ *(impute_work ==``flexible''), and* $b = \mathbb{P}$*(impute_work is incorrectly labeled)*

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

```
def compute_M(a,b):
    return 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.

```
a = sum(impute_work)/(impute_work.size)
#0.548
b = 1- 0.8585999999999998
#0.141
M = compute_M(a,b)
#0.711
```

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

```
work_vote_bias_correction = work_vote_relationship / M
# 0.472
```

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?

References

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: Voting and registration supplement* [Data set]. https://www.nber.org/research/data/currentpopulation-survey-cps-supplements-voting-and-registration

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

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