# SVM Class Exercises and Examples

Zach Adair

MSDS 600

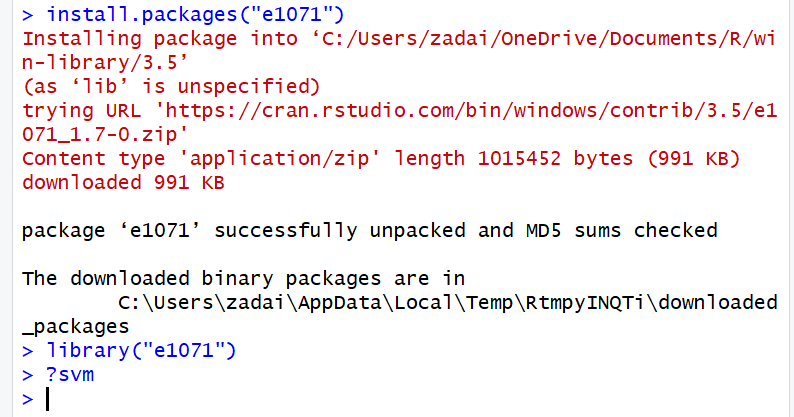
11/19/2018

## Objective

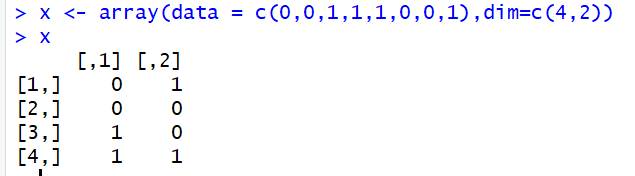
Go through the instructions in the pdf of the XOR Classification Problem and learn about Support Vector Machines and how they operate in R studio. After, see if there are opportunities to use other data sets and compare how those other examples compare to the original example.

## From the Original Instructions

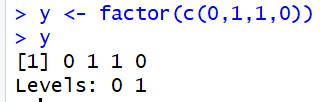
1. Install the e1071 package. Then set up the XOR problem as follows



1. Create a 4,2 array with the input values



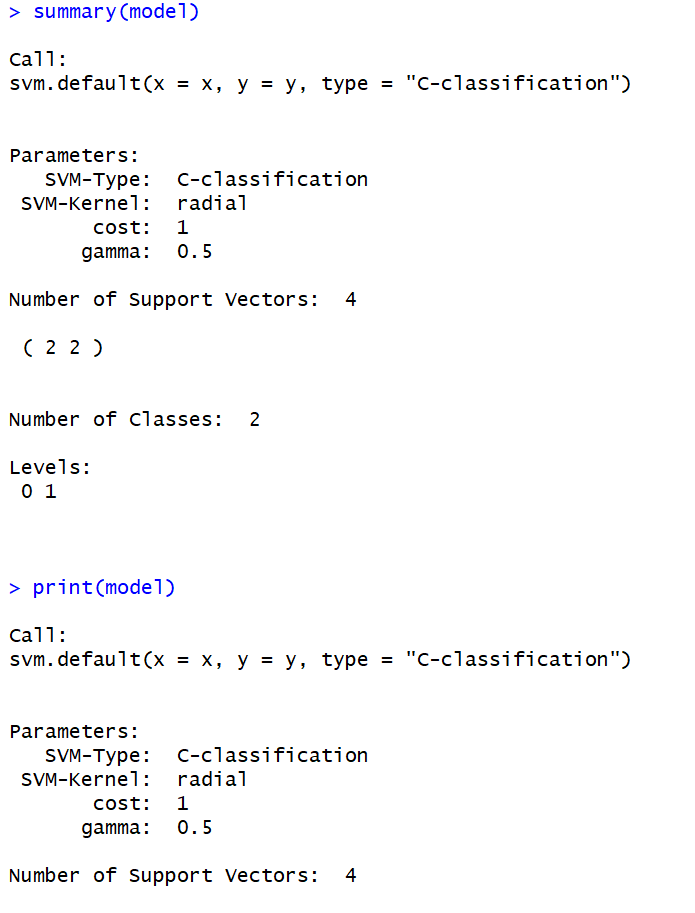
1. Next create a vector of factors (outputs for the input values):



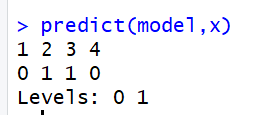
1. Next, create the SVM Model:



1. To display some information about the created model you can enter these two commands:

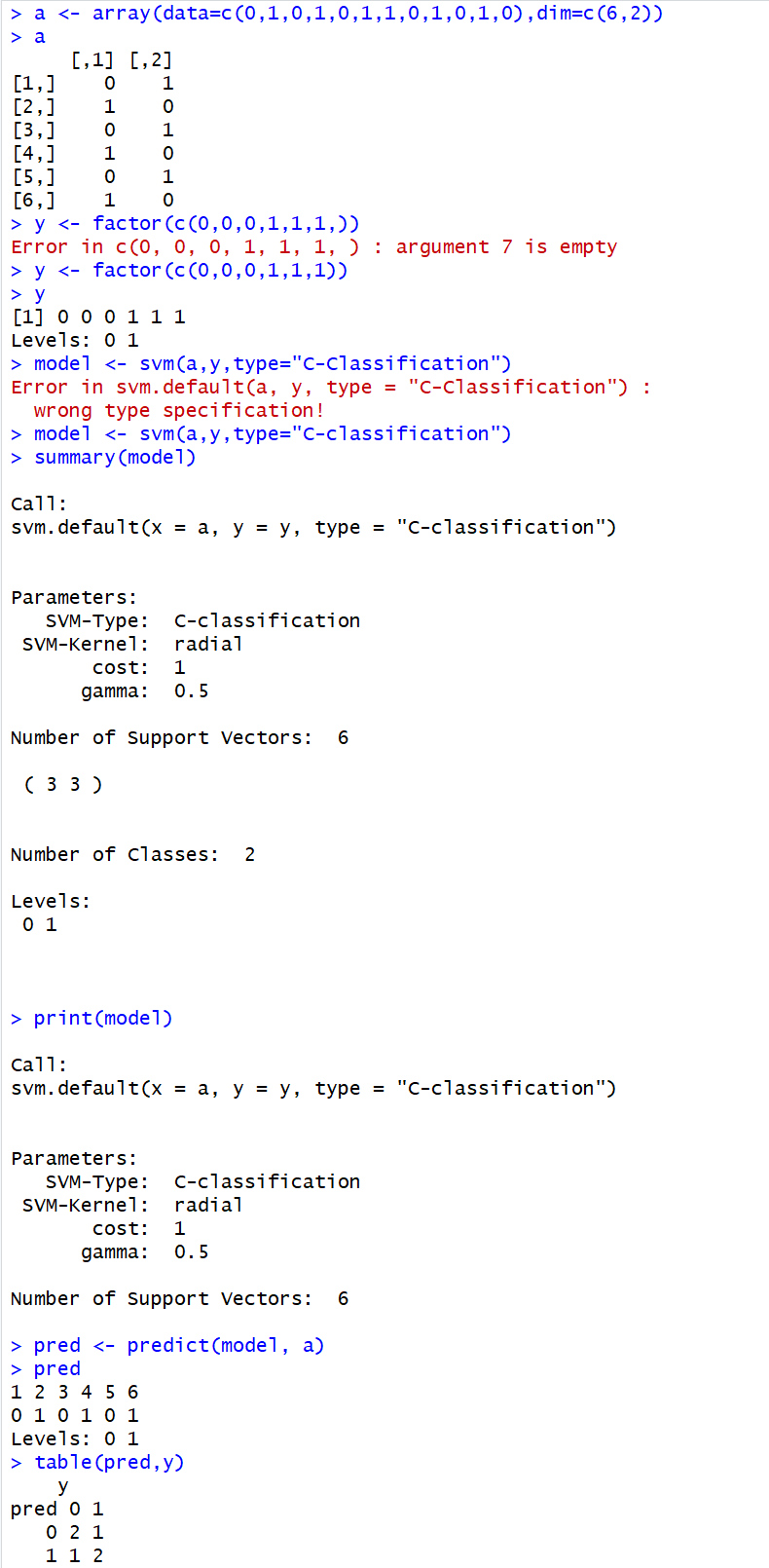


1. TO see the SVM model predictions for the input values enter the following:



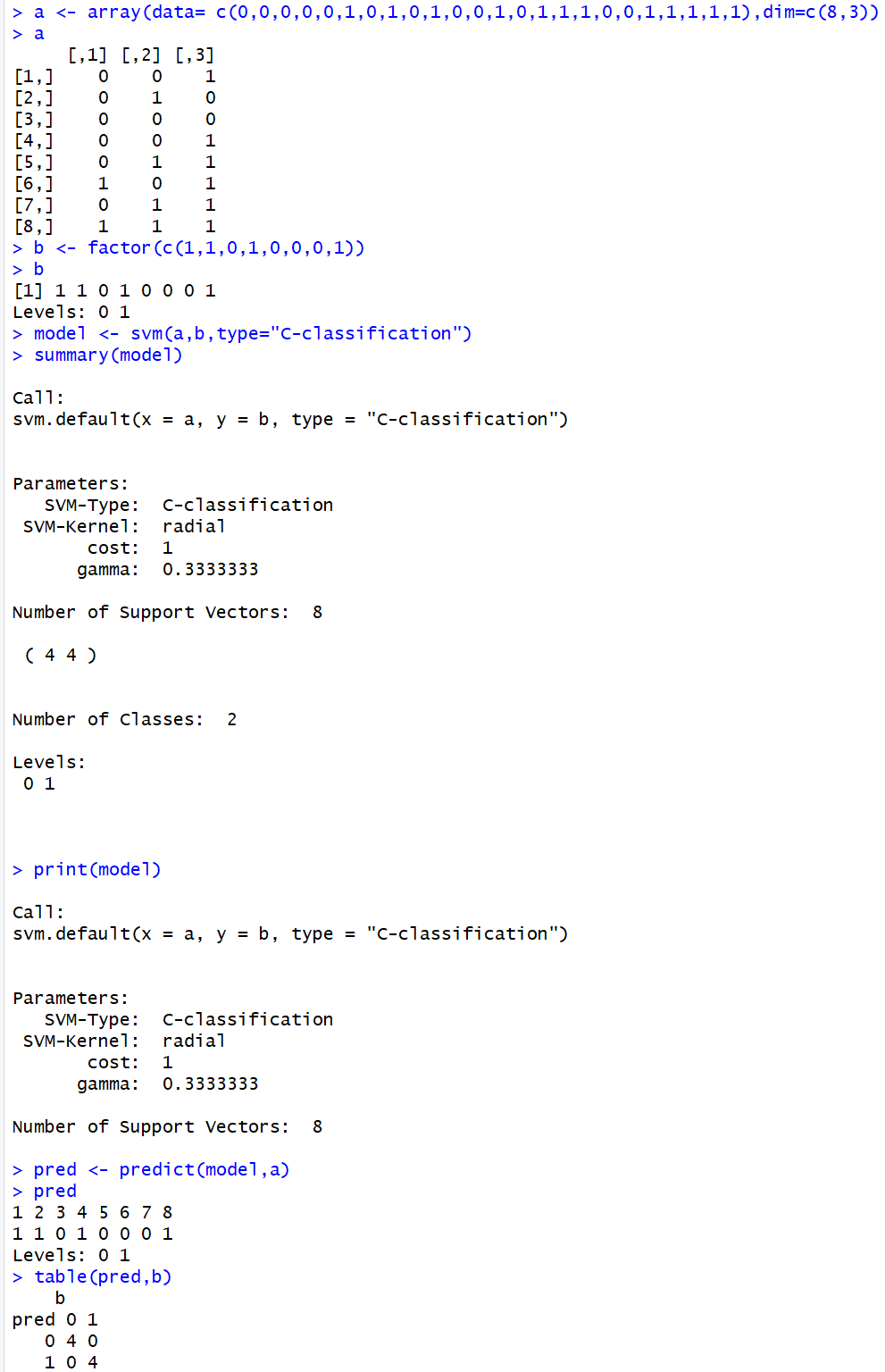
1. Create a variation of the simple learning problem and see how the results change.

**Example 1**



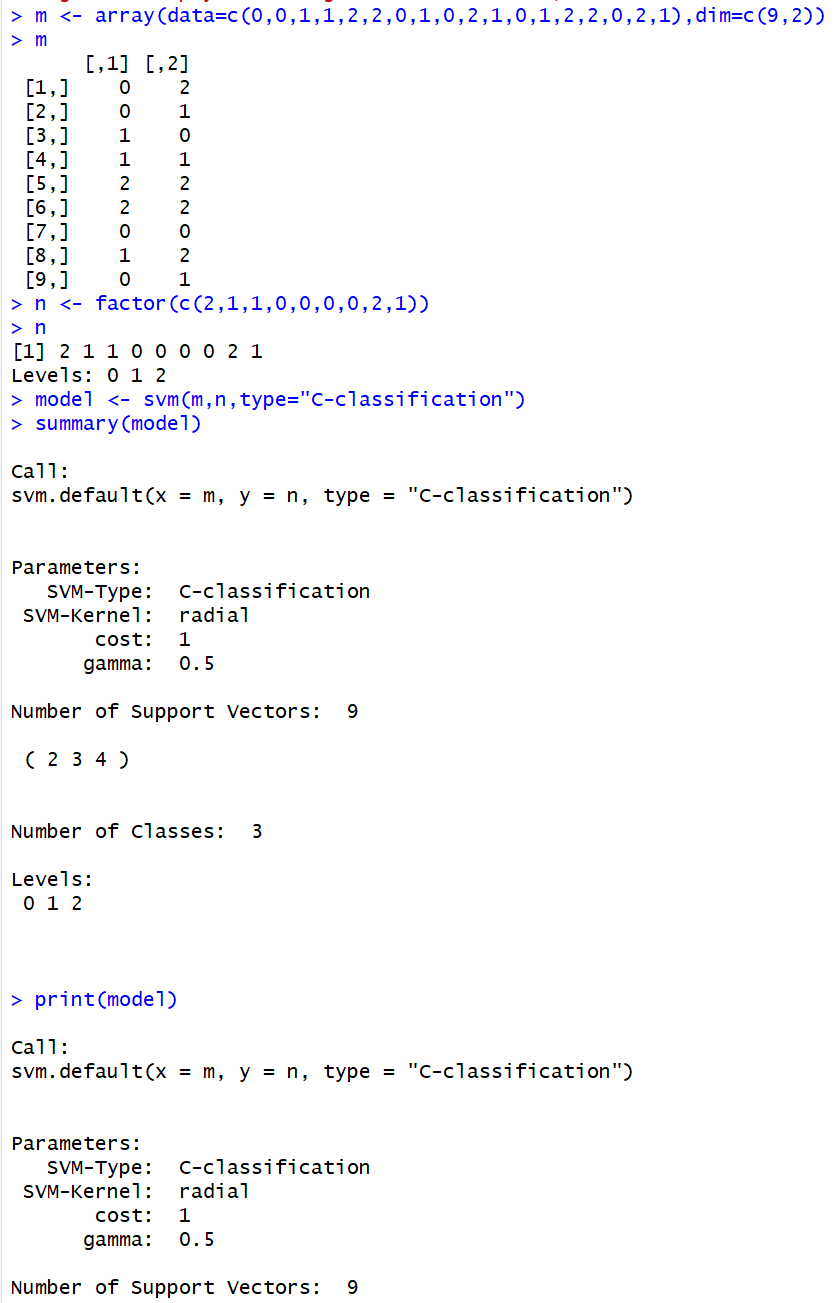
For my first example on my own, I added two more to the vector to see the model would look like lengthened out with more entries. For my y vector, I simply just put in the first three outputs as 0 and the last two outputs as 1. When the prediction model was setup and run though the results alternated 0,1,0,1,0,1; similar to what each output in the first column of the input vector displayed.

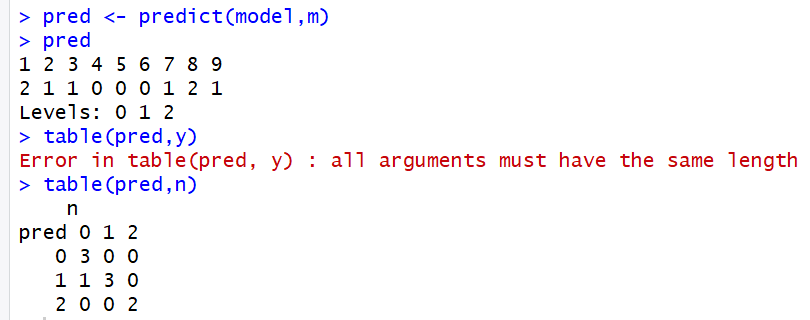
**Example 2**



The change I made with this example is I made the vector 8x3, changing the gamma to 0.333. The y vector I set up to immolate the Exclusive or rules with 0 and 1, anticipating that the prediction model would do the same. Once the prediction model was setup and ran it ended up predicting exactly what I thought it would from what I set in the b vector

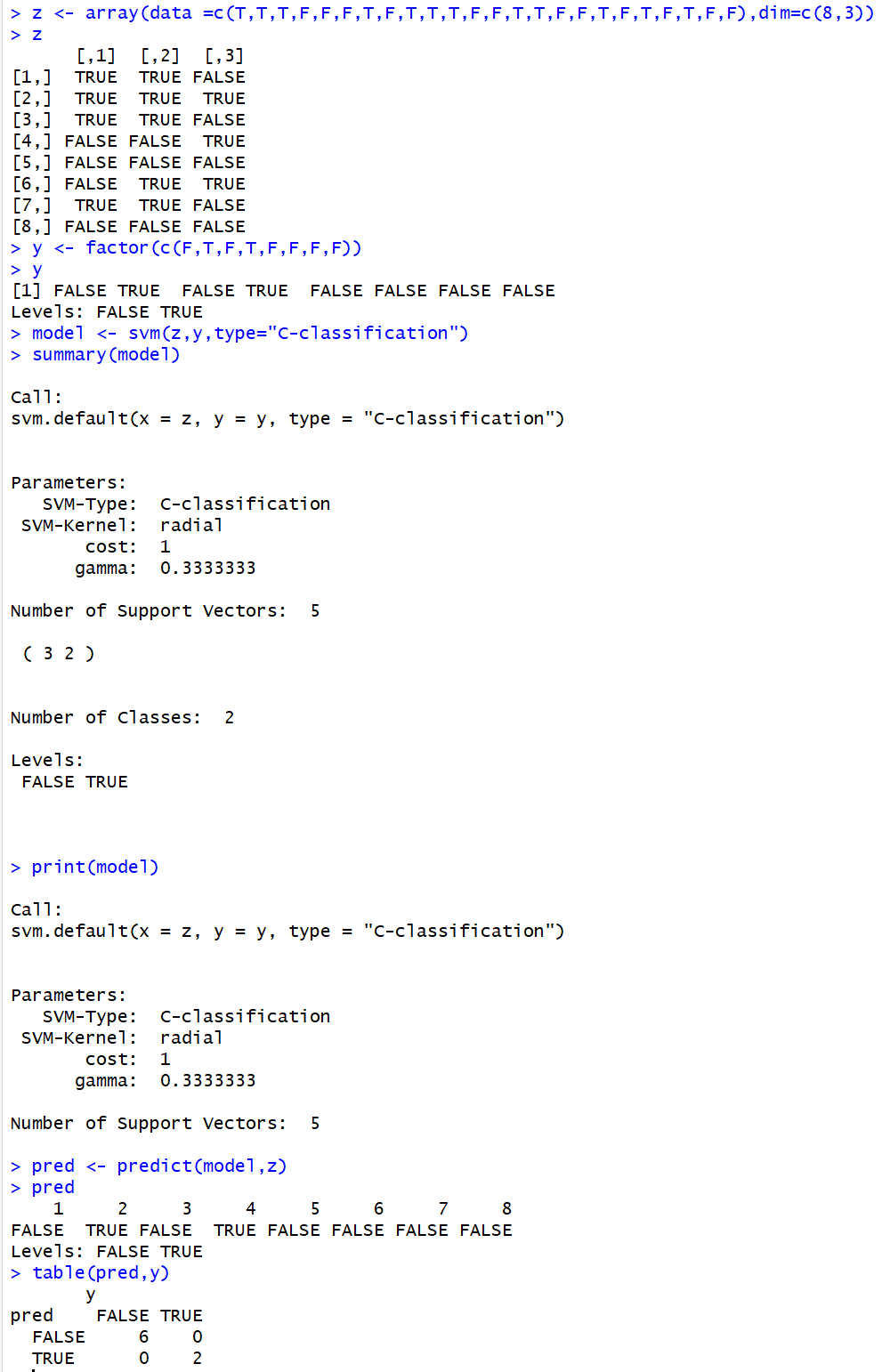
**Example 3**





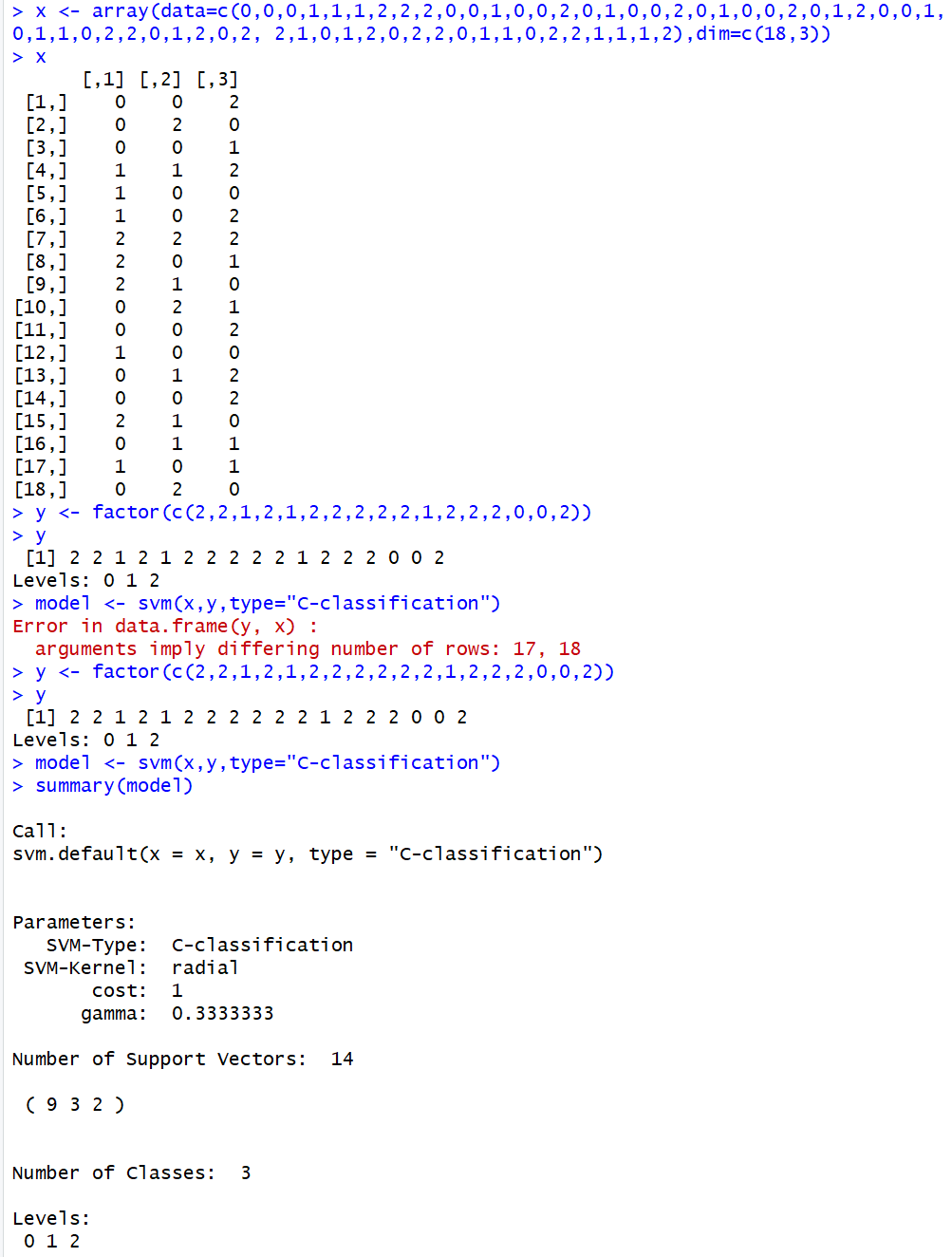
The change I made with the 3rd example is that I added a 3rd class to my vector to see any kind of change that occurs. As a result, nothing changed and each output was dependent on which of the three inputs were used. Besides a small change in the prediction results compared to the y vector the results didn’t have much variation.

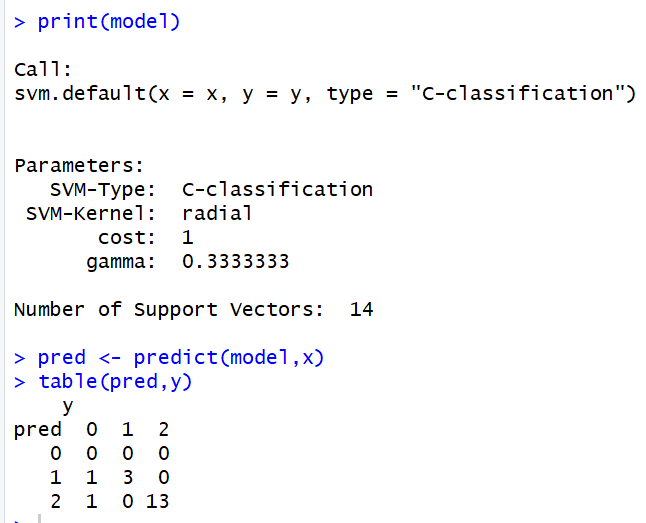
**Example 4**



In example 4, I tried creating a prediction model with True/False instead of with numeric classes. I tested it to see if they would relate to each other even with the subtle difference. The change made no effect on the example and it ended up outputting similar to Example 2 because of how the vector was set up with three inputs but only two classes. I created the y vector in the same rules as the Exclusive or, where the class with the odd amount of inputs would be what the output would become. With how I created the z variable, it was over-balanced with a lot of vector inputs having 2 True values and only 1 False value. And as a result, the y vector and the predictor outputted overwhelming as False.

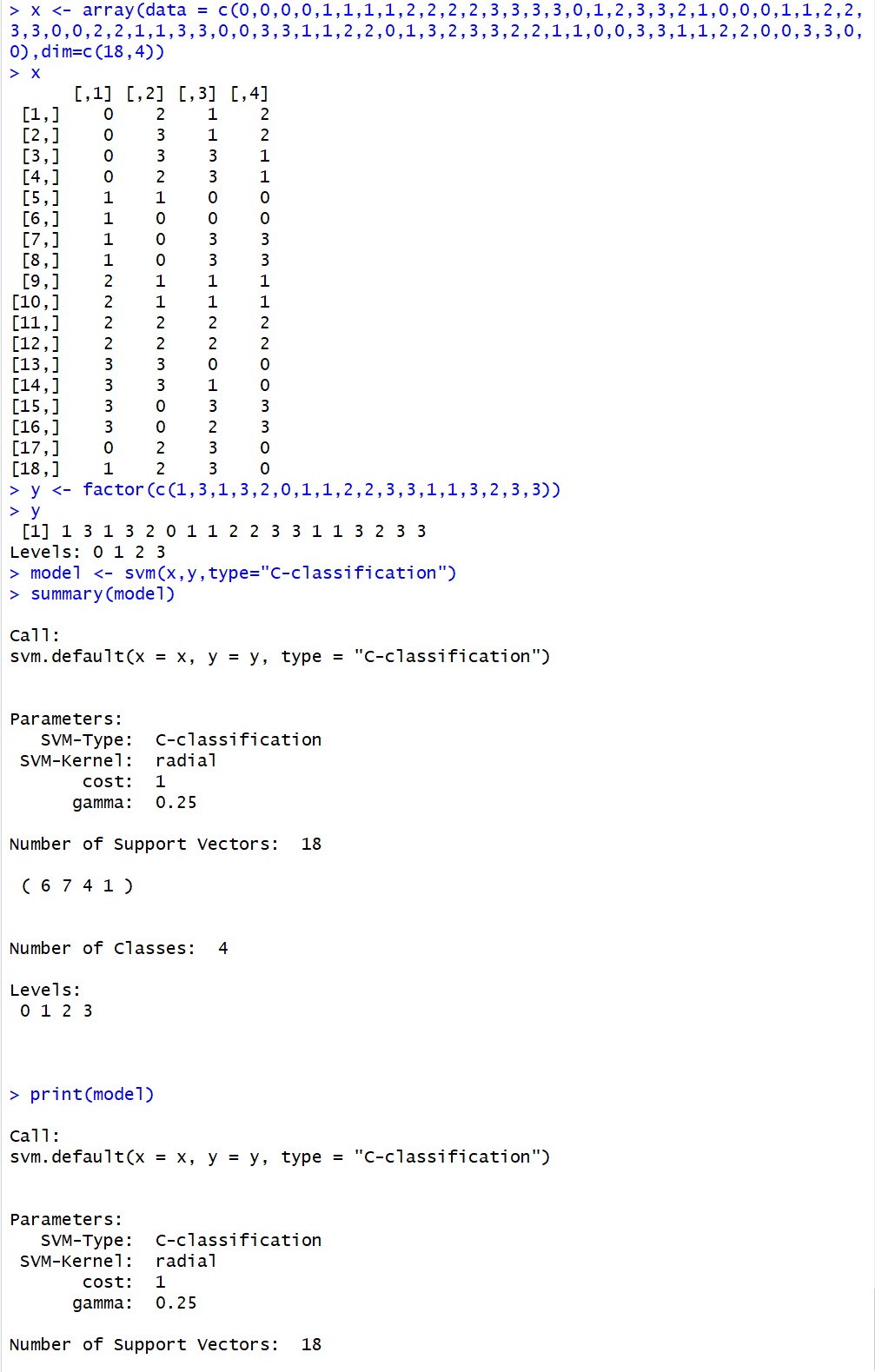
**Example 5**

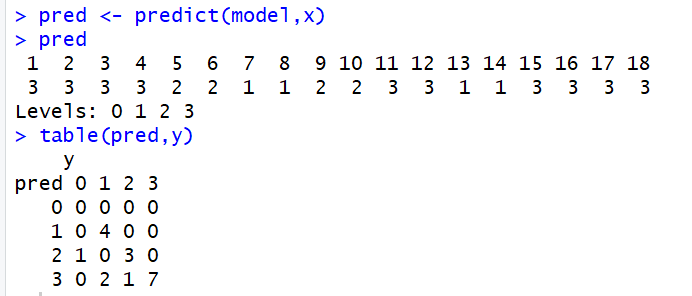




In example 5, I created a vector with three classes and within the vector each row has three inputs, adding another vector column and extra output possible when predicted. It was also a bigger vector than the previous five and it was interesting to see when the prediction model laid out the variety shown over our 18 different inputs and three output options that I didn’t expect. Interesting to see when predicted how different the prediction of model and the y vector are so different.

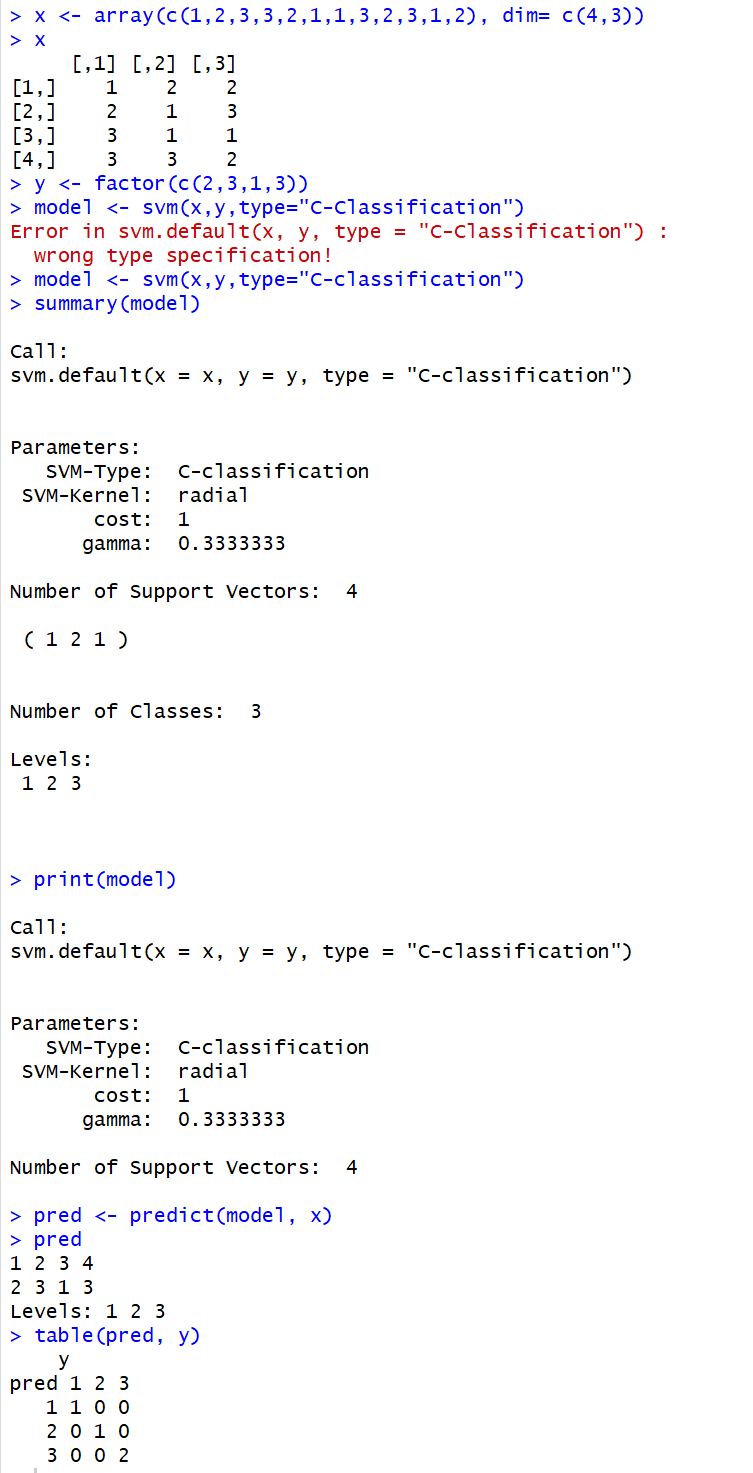
**Example 6**





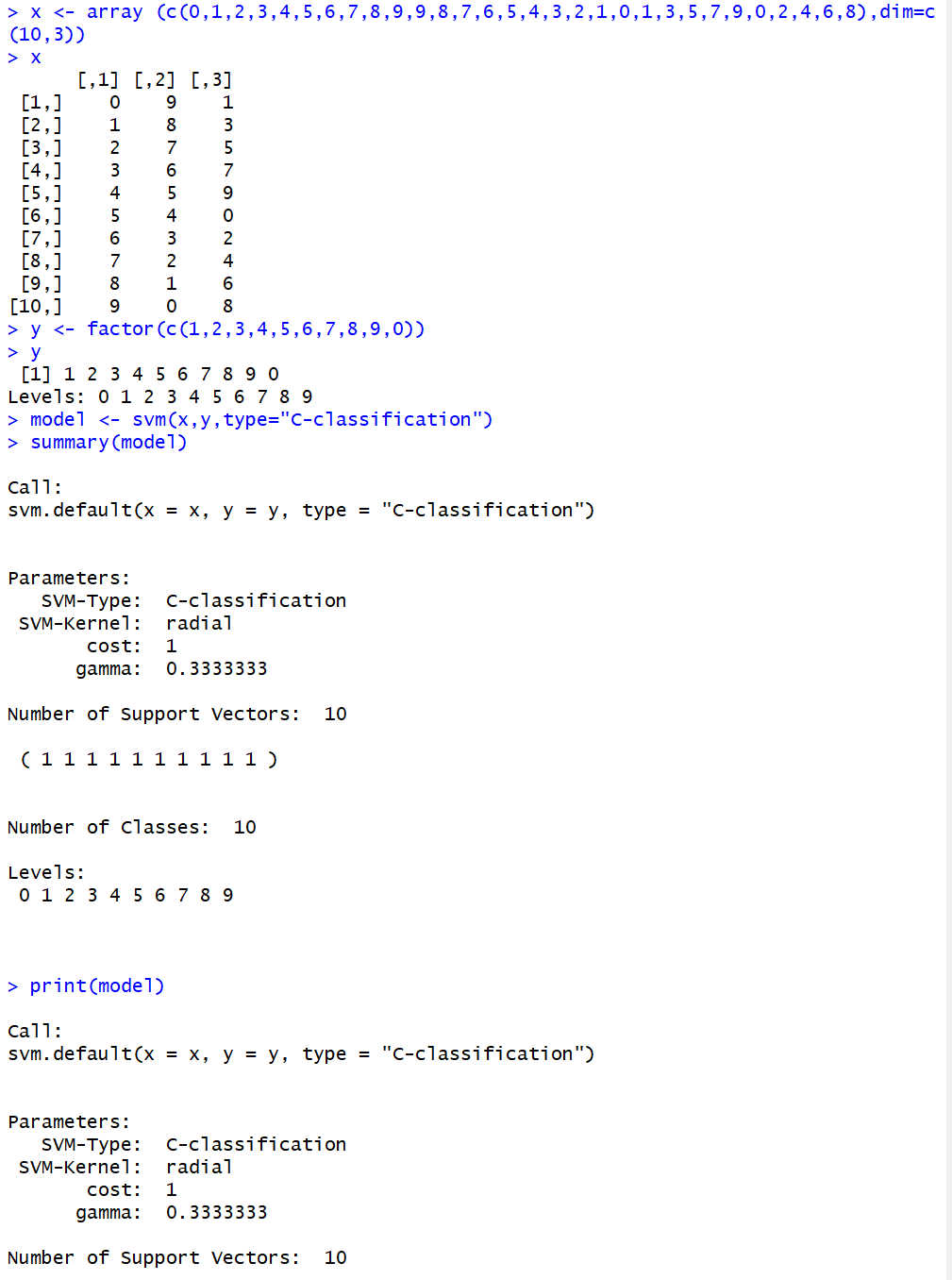
For this example, I brought in an extra class and input compared to every other example. I wanted to see the prediction output when adding more possible outputs, while also creating a bigger vector, x. The predictions look to be much less predictable with the bigger vector and more outputs; looking at the y output vector and the result of the prediction, the predictability is much more difficult, which is what I anticipated from this example.

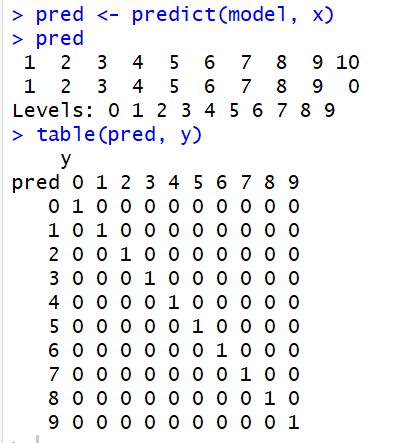
**Example 7**



For example 7, I used three classes and three inputs per vector to produce my outputs. The outputs were designed to populate for whichever in the input had the more values or if each input had a different value the highest of the three inputs would become the output. When predicted based on the model set up, the output transpired as such too.

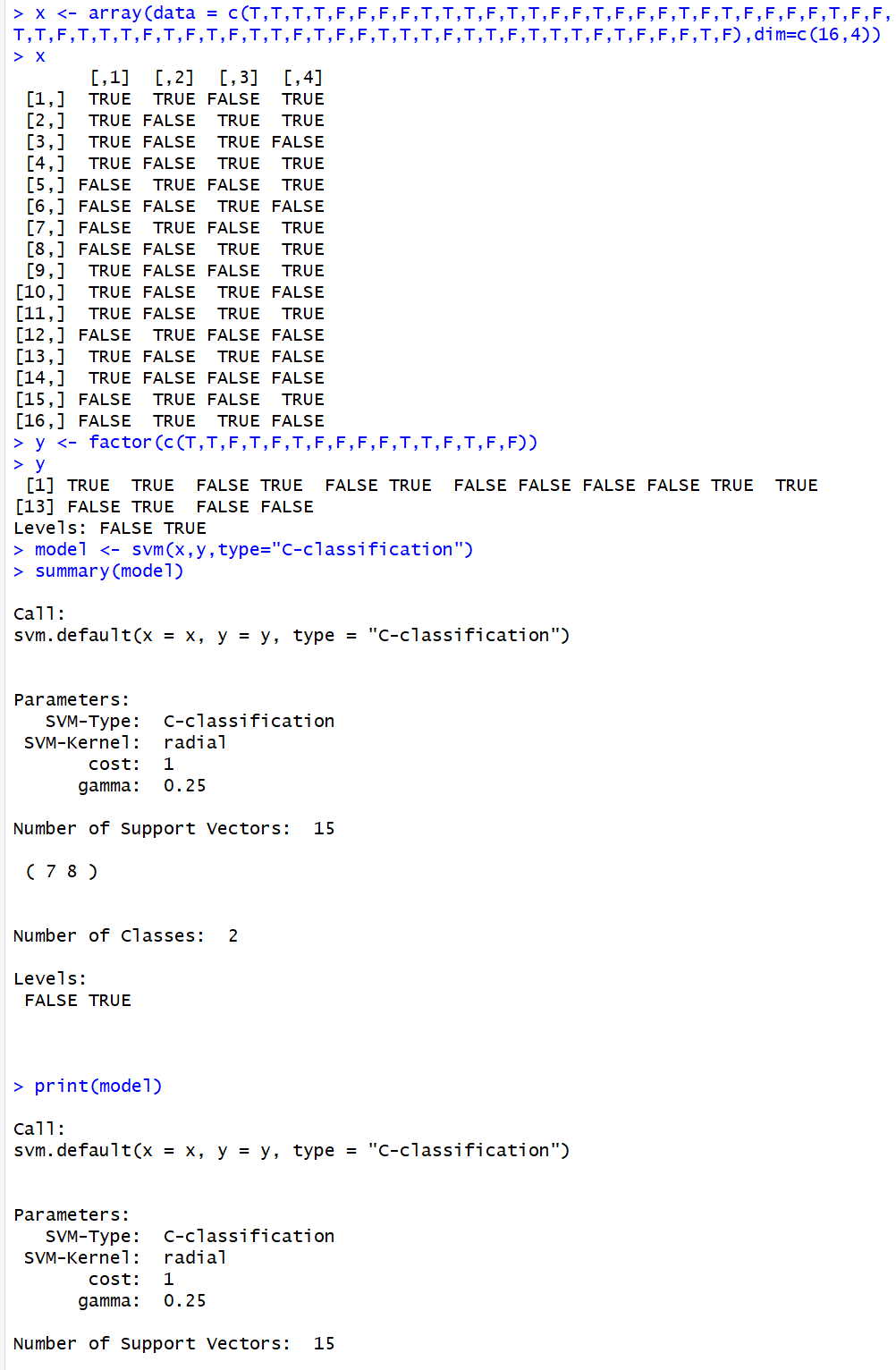
**Example 8**

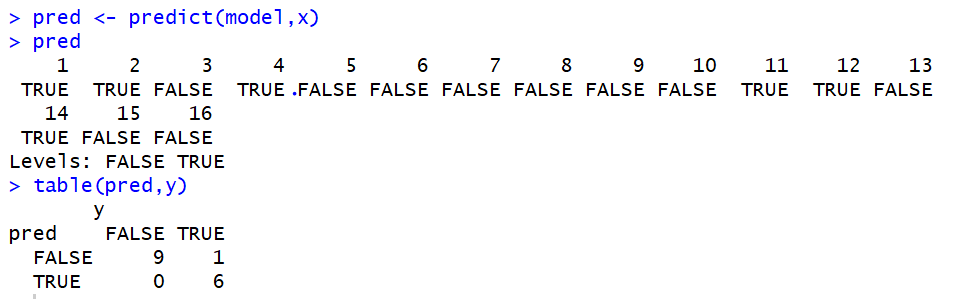




In example 8, I tried creating a vector with more inputs in hopes of making a prediction model with some hopefully more unpredictable results because of the amount of classes and the different inputs. The vector outputted a new value for each of the 10 inputs, which based on how my y was set up is not surprising to see happen.

**Example 9**





For example 9, I tried creating an example of Exclusive or, where there are 4 input values for every one output value. My input options are True/False, so I figured I would create something where the predictability on my end would be harder to guess. From my y vector and my prediction results ended up differing which means my notion about the vector being harder to predict were warranted.

## Conclusion from the Examples Created

For my first look into SVM (Support Vector Machines) I found this exercise quite interesting. None of the vectors I created were very complex but the results of the prediction models varied much more than I assumed they would. I originally anticipated that what I put in my 2nd variable, usually y, would be what the output would be every time. I found out quickly that wasn’t the case and it wasn’t the case for the majority of the examples I created myself. My next thought was that it depended on the class and what each vector had in it. Which held true more often but wasn’t exactly how some of the predictions resulted. One thing I found interesting was the addition of adding more than 2 inputs into the vectors. Whenever more classes were added I tend to find my example models were a little more unpredictable than when I was making vectors with 2 inputs. I believe the result of this has something to do with the gamma value, and how whenever a vector had more inputs in the rows the gamma value would be divided out depending on what that number is. I also thought adding more variable classes would do the same thing, but this notion was wrong. If I had a vector with more than 2 classes but each vector row only had 2 inputs, than the gamma value would stay at 0.5 and it was no less or more predictable than if I had just two classes.