I work at General Motors (GM) analyzing 12V battery data to identify potential vehicle issues. Since all vehicle electronics draw power from the 12V battery while the vehicle is off, the battery often acts as a "canary in the coal mine" for software problems in off-power mode – such as control units not shutting down properly, or features activating unexpectedly.

To detect these problems, we analyze signals that vehicles transmit to GM's servers (telemetry) along with diagnostic trouble codes (DTCs). GM also collects approximately real-world usage data from employees who drive company vehicles in their daily lives. We then classify whether a vehicle's battery is draining excessively and determine if the issue is a previously known problem or a new one requiring deeper engineering investigation.

Some predictors that we use in our classification work are:

1. Telemetry signals sent by the vehicle
2. Software versions or modules running on the vehicle
3. Whether the driver is a known "super user" who may have non-standard equipment installed
4. Presence of diagnostic trouble codes (DTCs).

Part A

Note: This is my first time programming in R, so I relied heavily on Google and Stack Overflow; however, all code is original. As recommended all code is saved in a single file `Homework1.R`. To further organize the code, I placed all analysis for this section in `Problem2_PartA`.

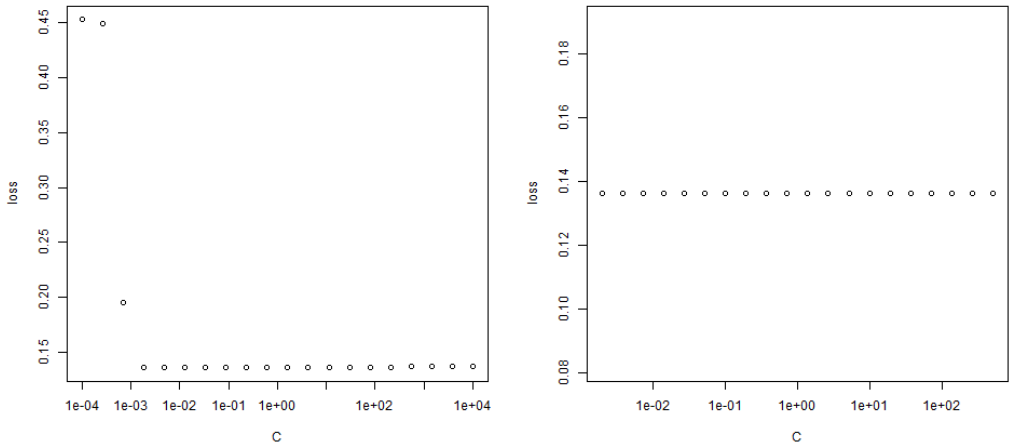
Methodology

Since I did not know an appropriate value for C, I decided to do an initial domain search plotting loss against a range of values. To reduce boilerplate through the rest of the code, I defined a `model_generator` function defining the parameters laid out in the assignment. Then using `sapply` as a for loop generated the loss for a range of C values using the `svm_plot_range` (again to reduce boilerplate).

The `caret` was used to create a confusion matrix for the chosen model to reveal additional information about the accuracy of the classifier.

Results

The domain search shows a wide range C that the analysis is insensitive to the choice in parameter with a constant error floor from $C = [2 \times 10^{-3}, 5 \times 10^{2}]$.



Since there is negligible difference over this range, I will choose to use $C = 10$ for the remaining calculations.

Using the code provided in the assignment I exported the coefficients to show the equation for this classifier:

$$\begin{aligned} 0 = 8.16 \times 10^{-2} - 9.03 \times 10^{-4} A_1 - 7.89 \times 10^{-4} A_2 - 1.70 \times 10^{-3} A_3 + 2.61 \times 10^{-3} A_8 \\ + 1.01 \times 10^{0} A_9 - 2.84 \times 10^{-3} A_{10} - 1.57 \times 10^{-4} A_{11} - 3.93 \times 10^{-4} A_{12} \\ - 1.28 \times 10^{-3} A_{14} + 1.06 \times 10^{-1} A_{15} \end{aligned}$$

The confusion matrix for this classifier is:

		Actual	
		0	**1**
Predicted	**0**	**286**	**17**
	1	**72**	**279**

The accuracy for the classifier is 86.3%.

Discussion of results

The stability across a wide range of hyperparameters points to the classifier not being overfit, which we would generally assume because there are so few parameters relative to the number of datapoints.

The small scale of coefficients except for A_9 and A_{15} shows the other factors are relatively insignificant, this was verified because limiting to these two factors still generates an error of 86.3%.

We can see that the model is much more likely to misclassify defaulters as safe risks, this points to needing shift the model to be more risk averse.

Part B

Note: This is my first time programming in R, so I relied heavily on Google and Stack Overflow; however, all code is original. As recommended all code is saved in a single file `Homework1.R`. To further organize the code, I placed all analysis for this section in `Problem2_PartB`.

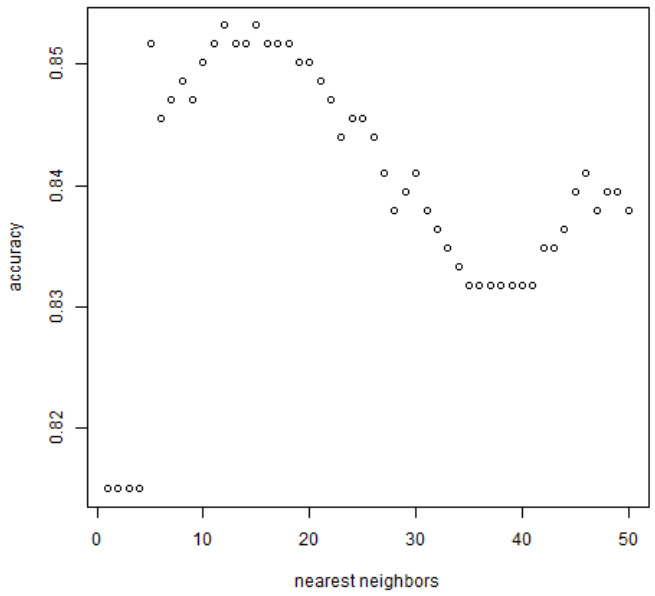
Methodology

Similar to the strategy used in Part A, I created a boilerplate function `knn_leave_one_out_no_test_set_iteration` that fits a KNN for n neighbors while excluding the i^{th} point (using the indexing technique recommended). To find the accuracy of the n nearest neighbors KNN, I used `sapply` to exclude it from training while using it for inference in the `knn_leave_one_out_no_test_set` function. The individual nearest neighbor function was used to scan a range of possible neighbors with the results shown below.

The `caret` was used to create a confusion matrix for the chosen model to reveal additional information about the accuracy of the classifier.

Results

This showed that the best number of neighbors to use is 12 with an accuracy of 85.3%, although all the classifiers performed similarly with a relatively large jump in accuracy for 5 or more neighbors.



The confusion matrix for this classifier is:

		Actual	
		0	**1**
Predicted	**0**	**308**	**46**
	1	**50**	**250**

Discussion of results

The KNN classifier is relatively stable across a wide range of parameters and for the k=12 classifier being the highest accuracy is a good sign the model was not overfit to the training data. Additionally, even though the KNN model had slightly worse accuracy (1%), it produced fewer false positives which we were told were more costly than false negatives in this application meaning it may be the "better" model.