**Question 2.2**

The files credit\_card\_data.txt (without headers) and credit\_card\_data-headers.txt (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the “Credit Approval Data Set” from the UCI Machine Learning Repository (<https://archive.ics.uci.edu/ml/datasets/Credit+Approval>) without the categorical variables and without data points that have missing values.

1. Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set. (Don’t worry about test/validation data yet; we’ll cover that topic soon.)

Notes on ksvm

* You can use scaled=TRUE to get ksvm to scale the data as part of calculating a classifier.
* The term λ we used in the SVM lesson to trade off the two components of correctness and margin is called C in ksvm. One of the challenges of this homework is to find a value of C that works well; for many values of C, almost all predictions will be “yes” or almost all predictions will be “no”.
* ksvm does not directly return the coefficients a0 and a1…am. Instead, you need to do the last step of the calculation yourself. Here’s an example of the steps to take (assuming your data is stored in a matrix called data):[[1]](#footnote-1)

The intercept “a0” is at point 0.08010652. The equation for the classifier is f(x) = a0 + ∑(i=1 to 10) [ ai \* xi ]. The model’s accuracy came out to 0.863706 or 86.4%.

Code included in ‘2dot2SVM.R’

1. You are welcome, but not required, to try other (nonlinear) kernels as well; we’re not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

I attempted this with rbfdot and polydot. Changing the kpar and kernal values didn’t seem to result in any differing model accuracy. This may be due to the size of the dataset.

Code included in ‘2dot2SVMAltKern.R’

1. Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Don’t forget to scale the data (scale=TRUE in kknn).

# Notes on kknn

* You need to be a little careful. If you give it the whole data set to find the closest points to i, it’ll use i itself (which is in the data set) as one of the nearest neighbors. A helpful feature of R is the index –i, which means “all indices except i”. For example, data[-i,] is all the data except for the ith data point. For our data file where the first 10 columns are predictors and the 11th column is the response, data[-i,11] is the response for all but the ith data point, and data[-i,1:10] are the predictors for all but the ith data point.

(There are other, easier ways to get around this problem, but I want you to get practice doing some basic data manipulation and extraction, and maybe some looping too.)

* **Note** that kknn will read the responses as continuous, and return the fraction of the k closest responses that are 1 (rather than the most common response, 1 or 0).

I found that the highest accuracy values for my implementation of kknn happened around k=42. It seems to accurately reproduce the values of the R1 column within an accuracy of ~87%.

Code included in ‘2dot2kNN.R’

1. [↑](#footnote-ref-1)