Question 2.1
Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

Response
I have worked in the renewable energy industry as a data scientist. One project that I have worked on which utilized classification involved creating a model to predict which homes are likely candidates for photovoltaic (solar) panel installation. Some predictors that determine the likelihood of PV installation include: household income, type of home (detached, townhouse, etc.), age of occupants, geographic location, orientation (cardinal direction) of the roof of the home, proximity to other PV installations, and average sunshine duration (how sunny the area is where is the home is located). Based upon these factors, a classification model can fairly accurately predict the likelihood of a future PV installation for a given home.

Question 2.2.1
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.

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.)

(Please note that my R code for this question is contained in the file 'homework_1_Q2.2.R', and If you wish to run the code, you will need to change line 8 of my R code to your local directory which contains the credit card application data, referred to in this document as *cc_data*.)

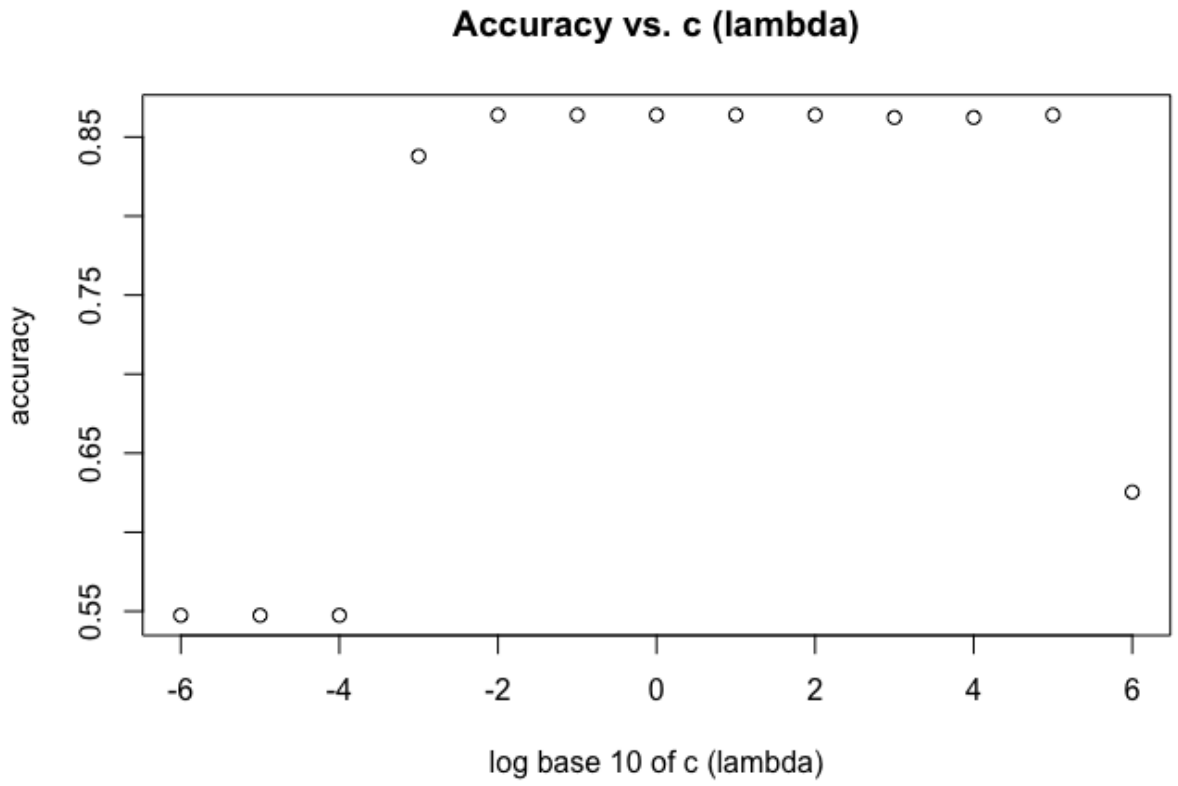
Response
I first read in the credit card approval data and saved it in a matrix called *cc_data*. Using the R support vector machine function called *ksvm*, I then determined the optimal function and *lambda* value, referred to as '*c*' rather than lambda in R's *ksvm* function, to predict the binary outcome of a credit card application. Data scaling was taken care of within the *ksvm* function, by setting *scaled = TRUE*.

My plan was to first determine the correct order of magnitude for the value c, and then, after determining the correct order of magnitude of *c*, more finely adjust the value the value of *c*, so that I could find the optimal equation of the classifier.

I determined the correct order of magnitude of c, by running the *ksvm* function in a for loop in R, and tested the following values of *c*: $c = \{10^{-6}, 10^{-5}, 10^{-4}, \ldots, 10^{6}\}$. Within the for loop I checked the accuracy of the predictions for each tested value of c. I did this by applying the coefficients generated through the *ksvm* model for each given value of c to the 10 independent variables in

each of the 654 data points in *cc_data* using R's *predict* function. I then determined the accuracy of correct predictions by comparing the predicted results with the actual results (the dependent variable in *cc_data*), in order to determine the percentage of correct results predicted for each of the tested values of *c*.

I expected to find a wide range of accuracies since I tested such a wide range of values of for *c*. However, I instead found that a very wide range of values of *c* produce an identical, or very similar rate of accuracy. This rendered my plan of fine tuning the values of *c* impossible, since I was not able to sufficiently narrow down the order of magnitude of *c* which would produce the most accurate model. The accuracy results for the values of *c* which I tested may be seen in the figure below (note that the x axis of the plot has been scaled to log base of 10) :



As may be observed in the plot, all tested values of *c* in the range $c = 10^{-2}$ to $c = 10^{5}$, resulted in a model with nearly identical accuracy, and additionally $c = 10^{-3}$ resulted in a similar level of accuracy.

I generated a matrix in R of the values of *c* which result in the most accurate model. The results of that matrix may be observed in the following table:

c (lambda)	prediction_accuracy
0.01	0.863914373
0.1	0.863914373
1	0.863914373
10	0.863914373
100	0.863914373
1.00E+05	0.863914373

As may be observed in the table, there are in fact *6* different values of *c* which produce an identically accurate matrix.

Since there is no single order of magnitude of *c* which results in a most accurate model, I next moved on to analyzing the results of the model, rather than trying to find the optimal value of *c*, since the results indicated that there likely was no singular value of *c* significantly more accurate than the rest. For each value of *c* that I tested, I recorded the coefficients generated by the *ksvm* function. I next computed the mean absolute value of the coefficients and also found the percentage of approved applications for each tested value of *c*. The results of this analysis may be seen in the following table:

c (lambda)	accuracy	a0	a1	a2	a3	a8	a9	a10	a11	a12	a14	a15	mean_abs_val_a	percent_approved
1.00E-06	0.547400612	0.999042974	-2.16E-06	6.87E-05	9.37E-05	0.000180291	0.000412707	-0.000241631	0.000233403	-1.00E-05	-3.29E-05	0.000101125	0.000137663	0
1.00E-05	0.547400612	0.990201405	-4.32E-05	0.000654135	0.000877164	0.001795607	0.00412707	-0.00239691	0.002332092	-6.01E-05	-0.0004761	0.001014968	0.001377733	0
1.00E-04	0.547400612	0.902125462	-0.000431956	0.006541594	0.00874008	0.017976241	0.041270703	-0.023961745	0.023320186	-0.000801735	-0.0044803	0.010223971	0.013774848	0
0.001	0.837920489	0.222615545	-0.002159778	0.03233817	0.046612449	0.111223162	0.375305335	-0.202026081	0.169560847	-0.004923501	-0.0252103	0.081189766	0.105054935	0.376146789
0.01	0.863914373	-0.081988535	-0.000150074	-0.001481829	0.001408313	0.007286389	0.991647004	-0.004466124	0.00714829	-0.000546839	-0.0016931	0.105482427	0.112131035	0.536697248
0.1	0.863914373	-0.081552264	-0.001160898	-0.0006366	-0.001520968	0.003202064	1.004133872	-0.003377367	0.000242862	-0.000474702	-0.0011932	0.106445053	0.112238758	0.536697248
1	0.863914373	-0.08148382	-0.001102664	-0.000898054	-0.001607456	0.00290417	1.004736346	-0.002985211	-0.000203518	-0.00055048	-0.0012519	0.10644046	0.112268028	0.536697248
10	0.863914373	-0.081575595	-0.000903367	-0.000789104	-0.001697213	0.002611363	1.005022141	-0.002836302	-0.000156929	-0.000392596	-0.0012784	0.106438717	0.112212617	0.536697248
100	0.863914373	-0.081584922	-0.001006535	-0.001172905	-0.001626197	0.00300642	1.004940564	-0.002825943	0.00026003	-0.000534955	-0.0012284	0.1063634	0.112296532	0.536697248
1000	0.862385321	-0.070178708	-0.000214919	0.000709779	0.001164517	0.000567302	0.998719209	-0.000503791	0.000715543	-0.000913008	0.00079697	0.001006235	0.100531127	0.535168196
10000	0.862385321	-0.070461037	0.000893617	0.001612572	-0.000341592	0.004211421	1.001452752	0.002357364	0.005275787	0.000280366	0.00250978	0.003592044	0.10225273	0.535168196
1.00E+05	0.863914373	-0.080544514	-0.004117738	-0.086896089	0.12971526	-0.083744032	0.988381368	0.031253888	-0.055666972	-0.037281856	0.02194074	0.018521785	0.145751973	0.536697248
1.00E+06	0.625382263	0.12811677	-0.828347146	-0.22172162	-0.330178203	0.282548814	0.575073141	0.6143978	0.260777397	-0.594304221	-1.1175369	0.933683312	0.575856857	0.472477064

As can be observed in the table, $c = 10^{-6}$, $c = 10^{-5}$, *and* $c = 10^{-4}$ all result in a model which predicted the approval of *0%* of the applications. Thus, we can clearly see that these values of *c* should be rejected as candidates that for the choice of *c* that we will use in our model.

If we look at the percent of approved applications for each of the 6 most accurate values of *c*, $c = 10^{-2}$, $c = 10^{-1}$, $c = 10^{0}$, $c = 10^{1}$, $c = 10^{2}$, and $c = 10^{5}$, it interesting to note that they all result in a model with an identical approval rate of approximately *.537*.

We may next go on to analyze the coefficients generated by the *ksvm* function for our tested values of *c*. We shall only concentrate on the 6 values of *c* which resulted in the most accurate model, as these values are the most compelling candidates for our choice of an optimal value of *c* in our model. For the values $c = 10^{-2}$, $c = 10^{-1}$, $c = 10^{0}$, $c = 10^{1}$, and $c = 10^{2}$ we may note that the mean absolute value of our coefficients is identical to three decimal places, with a rounded value of $c = .112$. However, for the choice $c = 10^{5}$, we note that the mean absolute value of our coefficients is $c = .146$. This indicates that our model considers the independent variables in *cc_data* to be more significant in determining the classification of the credit card applications when we let $c = 10^{5}$ than in our other tested values of *c*.

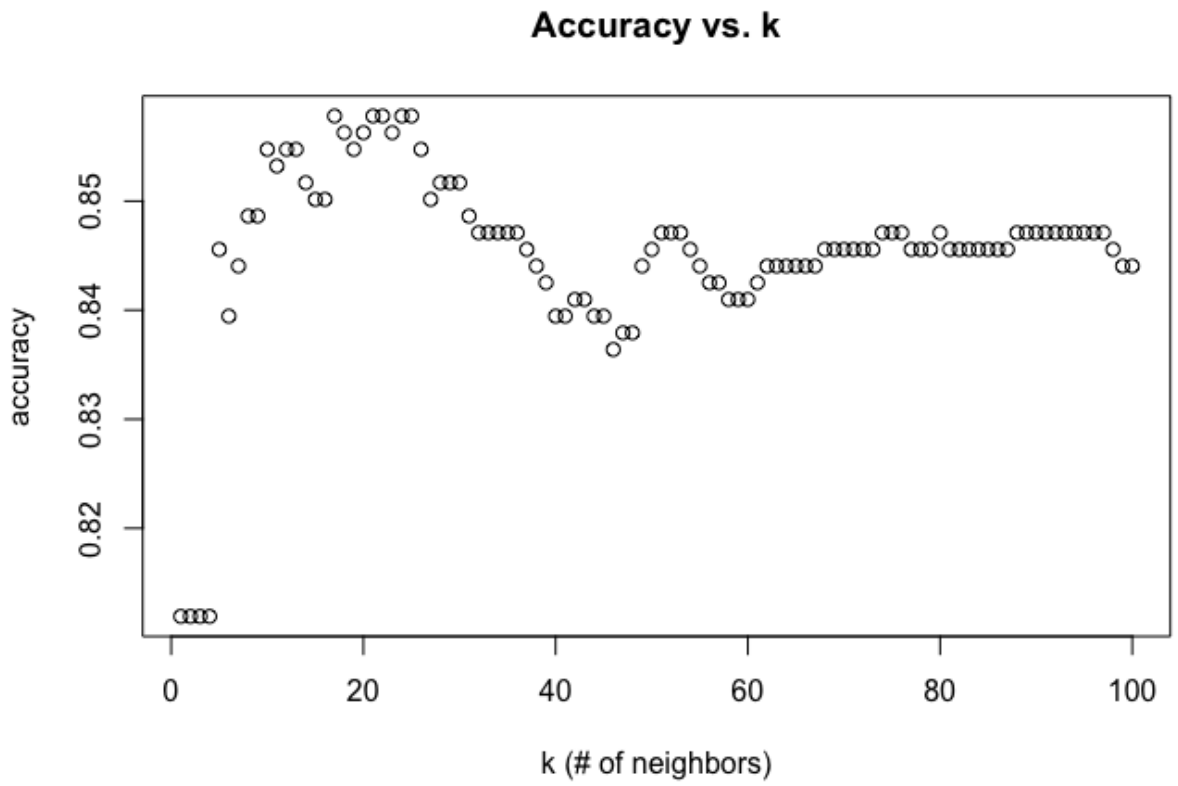
Since we see that very similar models and are produced for $c = 10^{-2}$, $c = 10^{-1}$, $c = 10^{0}$, $c = 10^{1}$, $c = 10^{2}$, and $c = 10^{5}$, we must look at the underlying role that c plays in our model. The *ksvm* model seeks to maximize the margins between the classes, while simultaneously minimizing the number of incorrect classifications. *lamba*, referred to here as *c*, is a regularization parameter in the support vector machine model, which penalizes misclassifications. As lambda decreases, misclassifications are more strongly penalized, and thus less incorrect classifications are allowed, which leads to narrower margins. As *lamda* increases, more misclassifications are allowed, and thus larger margins are produced. Generally, this will lead to a tradeoff between accuracy and maximization of margins. However, in our model we surprisingly find no trade off in accuracy for the values of *c* in the set $c = \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{5}\}$. Thus, our optimal choice of *c* is the value which seeks to maximize the margins while simultaneously minimizing the error. We may therefore choose $c = 10^{5}$ as our regularization parameter, as this value produces a model with the largest margin without suffering any loss in classification accuracy. $c = 10^{5}$ is also a compelling choice for our value of *c*, since it places more importance upon the values of the independent variables in *cc_data* than the other tested choices of *c*, as demonstrated by the mean absolute value of the coefficients generated by the model. The coefficients generated by setting $c = 10^{5}$ may be observed within the table above.

Question 2.2.3
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).

(Please note that my R code for this question is contained in the file 'homework_1_Q2.2.R', and If you wish to run the code, you will need to change line 8 of my R code to your local directory which contains the credit card application data, referred to in this document as *cc_data*.)

Response
I first read in the credit card approval data and saved it in a matrix called *cc_data*. I then used the k-nearest neighbors algorithm to predict the binary outcome of a credit card application, by means of implementing the *kknn* function in R. Data scaling was taken care of within the *kknn* function, by setting *scale = TRUE*.

Using nested for loops, I generated *100* different k nearest neighbor models using every value in the set $k = \{1, 2, 3, \ldots, 100\}$, where *k* is defined as the number of neighbors. I then tested the accuracy of each of the *100* models, by applying the model to each of the *10* independent variables in each of the 654 data points in cc_data. I specified that each data point *i* being tested should specifically be excluded from being counted as one of the "neighbors", so that the predicted outcome would not be tainted by self-counting. The R function *kknn* returns the fraction of k nearest neighbors that are 1. Since *kknn* returns a continuous a response, and we are attempting to generate a binary classification, I rounded the output to the nearest whole number (either *0* or *1*). Finally, I determined the accuracy of the predictions generated by the model for every value of *k*, by comparing the predicted results with the actual results (the dependent variable in *cc_data*), in order to determine the percentage of correct results for each of the tested values of *k*. The results of the test may be observed in the following figure:



As may be observed in the figure, for every value of *k* in the set {1, 2, 3, … ,100}, the model produced resulted in an accuracy between *.81* and *.86*. Thus, we can see that there is not a significant variance in the accuracy of each model.

In order to determine the optimal value of *k* for use in our model, I found the value(s) of *k* which resulted in the highest accuracy. This resulted in the determination that there are 5 separate values of *k* which each produce a model that results in an identical maximum accuracy.

k (# nearest neighbors)	prediction_accuracy
17	0.857798165
21	0.857798165
22	0.857798165
24	0.857798165
25	0.857798165

For each of the values *k = 17*, *k = 21*, *k = 22*, *k = 24*, and *k = 25* we may observe that a model, with an identical, and maximum accuracy, of .8578 (rounded to four decimal places) is produced. Thus, we may select any of these five values of *k* to use in our model, and for the purposes posed in this question, each of these 5 questions will produce an equally good value of *k*, with regard to the accuracy of the model that is generated.

The results of the model for these *k* values may be seen by running the attached R code, and are not presented in this paper, since it is neither practical nor useful to present 654 predictions here for each of 5 different models.