Question 10.1

Using the same crime data set as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and

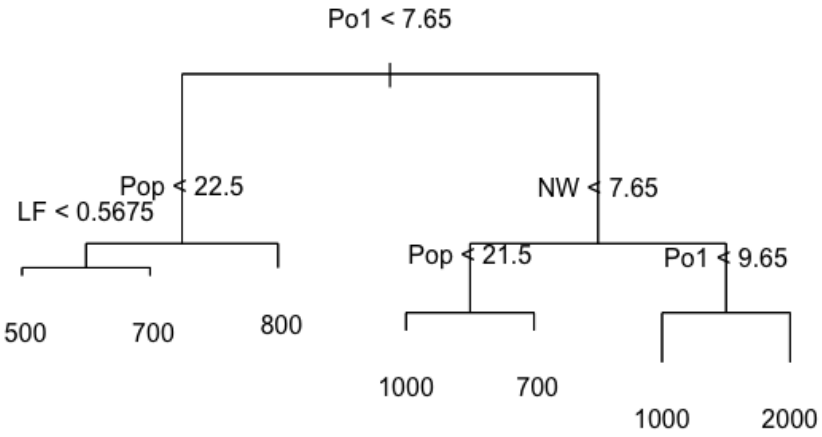
(b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

(Please note that my R code for this question is contained in the R files 'homework_7_Q10.1A.R' and 'homework_7_Q10.1B.R', and if you wish to run the code, you will need to change line 5 of my R code to your local directory which contains the 'uscrime.txt' data)

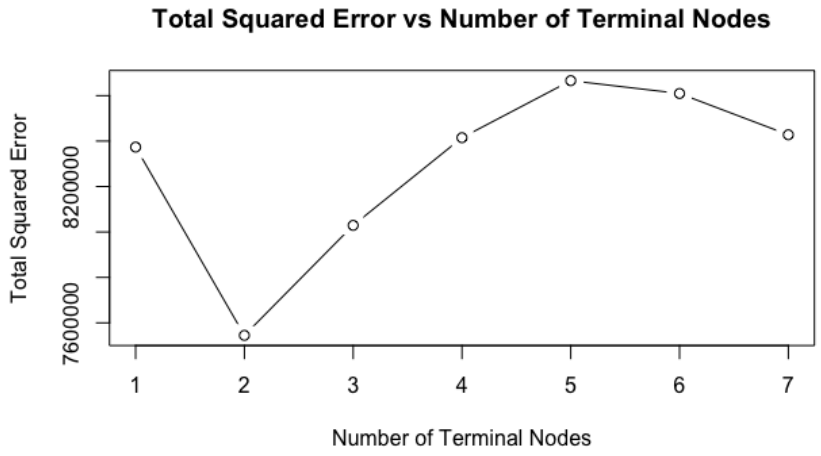
Response (Part a)

To begin solving this question, I first read in the data *uscrime.txt* and saved it as a variable called *crime_data*. *crime_data* contains 47 data points, 15 predictor variables and 1 response variable. I used the built-in R function *tree* to create a regression tree to train a model to predict the output variable *Crime* based upon the *15* predictor variables.

I first created a *regression tree* using the built-in *tree* function, using the *crime_data* data set as the only function input. This returned the following result:



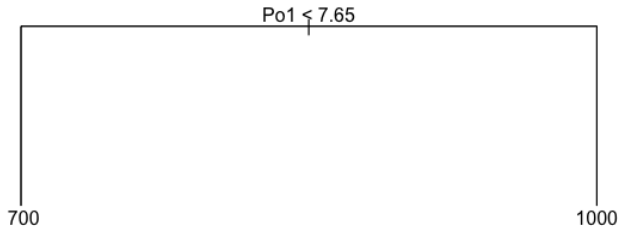
As may be observed in the figure above, the *tree* function created a regression tree with 7 terminal nodes. This is an unreasonably large number of nodes given that the *crime_data* data set contains only *47* data points. This means that on average, there would be fewer than *7* data points contained within each terminal node (leaf). This is far too few points to create a regression model. This is reflected in the large squared error associated with this model. The squared error vs the number of terminal nodes may be observed in the following figure:



As we can note in the figure above, we see the smallest squared error associated with the case in which there are two terminal nodes. The choice of using two terminal nodes to create regression tree which will be applied a data set with just *47* data points is much more reasonable than the choice of *7* terminal nodes, given the small size of the *crime_data* data set.

By choosing to include just two nodes in our regression tree, we will have an average of *23.5* data points in each terminal node. While this number of data points is less than ideal for creating a regression model, it is far better than the case of having *7* terminal nodes, which was selected by the *tree* function. Thus, from this point forward I chose strictly to focus on the case in which the regression tree has just *1* split and *2* terminal nodes.

In order to create a regression tree with just *1* split, I used the built-in R function *prune.tree* and applied this to the regression tree illustrated in the first figure above. The results my seen in the following diagram:



As we can see in the figure above, there is just one split based upon the predictive variable Po1. This indicates that *Po1* is likely one of the most important variables in forecasting the value of *Crime.* All of the data points in which the value of *Po1* is less than *7.65* are included in the first leaf, while all of the data points in which the value of *Po1* is greater

than or equal to *7.65* are included in the second leaf. Splitting the data based on this value of *Po1* divides the data as evenly as possible; we see *23* data points contained in the first leaf and *24* data points contained in the second leaf.

My next step in analyzing this problem was to build a separate regression model for each of the terminal nodes shown in the figure above. To begin, I simply used R's built-in *lm* function and created a regression model for each subset of data using all of the predictor variables as factors in each regression model. After creating the regression models using all predictor variables as factors in the model, I next selected the predictor variables which were most significant by removing all predictor variables which had a *p-value* greater than *0.1*. This left me with the following two models. The regression model in the first leaf is given by:

$$Crime = 820 + 9.5(Ed) + 11.4(Pop) - 3164.1(Prob) - 12.1(Time)$$

And the formula for the regression model in the second leaf is given by:

$$Crime = -4271.1 + 72.8(Ineq) + 312(Ed) + 26.5(Time)$$

Given that the variables *Ed* and *Time* appear in both of the above two linear models, it is fair to assume that these predictors are likely good estimators of the response variable *Crime.* Additionally, the variable *Po1* is very likely significant in forecasting *Crime*, given that R's *tree* function selected to split the *crime_data* data set into subsets based on this factor, even though we do not see it in either of the regression equations above.

In order to analyze the performance of the above two regression models, I implemented cross validation, since testing the performance of a model on the same of data which it was trained on will likely lead to an overly optimistic estimation of the performance the model will exhibit when applied to other independent data sets. I implemented cross validation via R's built-in *cv.lm* function. I found that the regression model in the first leaf had an R^2 value of *0.448* and the regression model in the second leaf had an adjusted R^2 value of *0.394*. Together, they had a weighted average cross validated R^2 value of *0.421*. While this is not an extremely high R^2 value, it still indicates that the models account for *42.1%* of the variability exhibited by the response variable *Crime.* In real world applications, this model would likely be said to have reasonable forecasting capabilities.

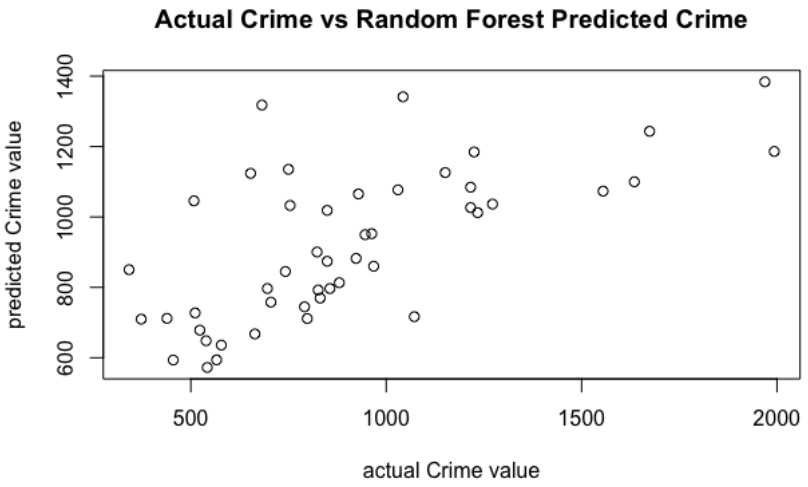
It should perhaps be noted that a method such as a regression tree, which splits the data set into smaller subsets, is likely not the most ideal algorithm to apply to the *crime_data* data set, given that it only contains *47* data points. Regression tends to work best on relatively large data sets, and given that the regression models in the first and second leaves only contain *24* and *23* data points respectively, we would perhaps see improved performance if the data set had not been split into such small subsets.

Response (Part b)

For this question, I used the *random forest* algorithm to create a model to predict the response variable *Crime.* To begin, I first read in the data *uscrime.txt* and saved it is a variable called *us_crime*. I used the built-in R function *randomForest* to create the random forest model.

A random forest is an ensemble learning algorithm which may be used for both classification and regression. Random forest algorithms work by creating a multitude of decision, and then returning the mean prediction of all of the trees (in the case of regression), or the mode prediction of all trees (in the case of classification). Since we are attempting to predict a continuous output in this problem, I of course created a random forest which implements regression. Random forests implement *feature bagging* also known as the *random subspace method.* This is done to reduce correlation among the trees contained within the model. *Feature bagging* in the *random forest* algorithm works by randomly selecting a small subset of features at each *branch* (*split*) in the tree, and then choosing the best factor from amongst this subset to branch on. If *feature bagging* were not implemented, then the factors which are the strongest predictors of the response variable would likely be branched on in many of the trees, thus leading to a high correlation amongst the trees. By ensuring that the *trees* within the *random forest* are not strongly correlated with one another, *random forests* help to reduce overfitting. The number of factors to be contained within the random subset selected at each branch is generally equal to *n/3* rounded down, where *n* is equal to the number of features. Since the *crime_data* data set contains *15* predictor variables, I chose to include *5* factors in the random sample at each branch when constructing the *random forest* model.

In the figure below, I have plotted the predicted Crime values generated by the random forest model I created against the true Crime values:



As may be inferred by the figure above, the random forest model does a fairly good job of predicting the value of Crime, except in the cases of the data points which had quite large

values for Crime. In these cases, the random forest model consistently under-predicted the correct value. This could be due to the fact that these instances are simply anomalous, and thus, difficult to predict accurately.

When analyzing the performance of the model illustrated in the figure above we find that it has an R^2 value of approximately *0.431,* meaning that it accounts for *43.1%* of the variance contained within the model. Although this number may appear slightly low, it still means that a significant portion of the variance is accounted for and that the model therefore likely has acceptable forecasting capabilities.

However, before making conclusions about the performance of the model, we must remember that if we train and test a model on the same set of data, we will likely see an overly optimistic estimation of the performance that the model will exhibit when applied to independent data sets. Thus, we must cross validate the data. Since the *crime_data* data set contains only *47* data points, *k-fold cross validation* is the most appropriate method of cross validation. I chose to implement a special form of *k-fold cross validation* known as *leave one out* validation, in which all but one data point are used for training and the remaining data points are used for validation in each iteration. This ensures that variance in model performance arising from which specific points are chosen to be in the training and validation sets, is reduced as much as possible, which is an important consideration in this case since we only have *47* data points. After implementing cross validation, I found that the model had an R^2 value of approximately *0.425*. It is interesting to note that the cross-validated R^2 value is so similar to the non-cross validated R^2 value. This is because the random forest algorithm inherently reduces overfitting, which is generally the reason that we see a discrepancy in measured performance between cross validated and non-cross validated model performance estimations.

Question 10.2

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression 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 logistic regression, 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 logistic regression model can fairly accurately predict the likelihood of a future PV installation for a given home.

Question 10.3.1

Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine- learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the *glm* function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use *family=binomial(link="logit")* in your *glm* function call.

(Please note that my R code for this question is contained in the R file 'homework_7_Q10.3.R', and if you wish to run the code, you will need to change line 5 of my R code to your local directory which contains the 'germancredit.txt' data)

Response

To begin my analysis of this question, I first read in the *germancredit.txt* data set and saved it as a variable called *gc_data*. The goal of this problem is to predict a binary variable, based upon other categorical variables. *gc_data* contains *20* categorical predictor variables and *1* response variable. The binary response variable that we are trying to predict represents a response of "good" with the number *1* and represents a response of "bad" with the number *2*. In order to be able to utilize R's built-in *glm* function, I first redefined "bad" to be represented with the number *0*, so that we would have a standard binary variable in which responses are represented by the numbers *0* and *1*.

I next split the data into training and validation subsets. I used *70%* of the data for training and the remaining *30%* of the data for validation. Since the *gc_data* data set contains 1000 data points, this means that the training subset contains *700* data points and the validation subset contains *300* data points. Since this is a sufficiently large data set, *k-fold cross validation* is not necessary here.

I next began training a model using the training subset. I began by using R's built-in *glm* function to predict the "good" and "bad" response variable in the *gc_data* data set using all *20* of the predictor variables. My initial model, which included all factors as predictor variables, had an *Akaike Information Criterion* (*AIC*) value of *681*. After my initial training of the model I determined which variables were important and which variables were unimportant based upon their p-value, and I removed all variables which had a p-value greater than *0.1* for every category, meaning that if a variable had a *p-value* less than *0.1* for any individual category, I kept the entire variable. After this second round of training, I found that my model had an *AIC* value of 679. I repeated the process of removing the insignificant variables once more and after this round of removing unimportant variables, I was left with only significant variables, meaning that every variable had at least one category with a *p-value* of less than *0.1*. After this round of training, the model had an *AIC* value of *678*, again a small improvement.

While it may appear as though I had already removed as many insignificant predictor variables as possible, since all predictor variables had at least one category with a *p-value*

less than *0.1,* it is important to note that there were certain categories within the predictor variables which had *p-values* greater than *0.1* , meaning that they were likely insignificant. In order to try to further narrow down my model and remove insignificant categories I next created binary (yes or no) variables for each category of the variables which had a *p-value* less than *0.1* . I appended the new binary categorical variables to the *gc_data* training subset and then used the *glm* function once more to create a logistic regression model which included these newly appended binary variables as predictors. After this round of training I now removed all variables which had a *p-value* greater than *0.1* and trained the model once more using only the variables determined to be important. After this final round of training, I found that all variables still included in the model had a *p-value* less than *0.1,* meaning that all predictor variables were significant.

After creating a model which contained only significant variables, I next moved on to validation. I created the binary categorical variables for the validation subset based upon my determination of which categories were most important in the training phase. I then tested the logistic regression model determined to be optimal in the training phase on the on the validation subset of data.

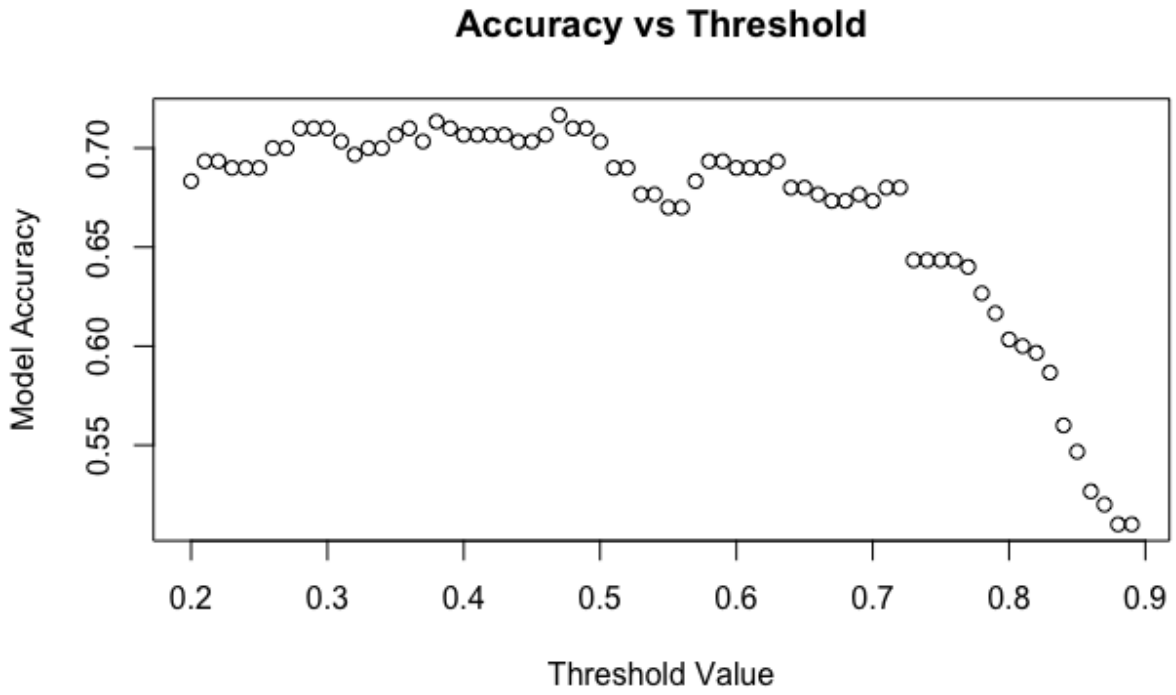
When R's *glm* function is used in logistic regression, it returns a probability between *0* and *1*. However, in this we simply want to make a prediction of whether an applicant is a good credit risk or not, and we are therefore not interested in a probability. Therefore, in order to make a prediction of whether or not someone is a good credit risk, I assigned a value of *1* to all predictions greater than the threshold of *0.5* and I assigned a value of *0* to all predictions less than *0.5*. Remember, as defined above, *1* is good and *0* is bad. The results of the prediction may be seen in the confusion matrix below:

		Actual Response	
		0	**1**
Predicted	**0**	31	23
Response	**1**	64	182

In the confusion matrix above, the top left entry (*31*) represents true negative (*0* was the predicted response and *0* was also the actual response), the top right entry (*23*) represents false negative (0 was the predicted response and *1* was the actual response), the bottom left entry (*64*) represents false positive (1 was the predicted response and 0 was the actual response) and finally, the bottom right entry represents true positive (1 was the predicted response and 1 was also the actual response). As may be inferred from the above confusion matrix, there were *213* correct predictions and *87* incorrect predictions. This means that the correct prediction was made exactly *71%* of the time.

Predicting the correct response *71%* of the time is a good result, but I wanted to see if this result could perhaps be improved. In order to try improve the accuracy of the model, I experimented with the threshold for determining whether the prediction should be

classified as *0* or *1*. I tried setting the threshold to all values in the range (*0.20, 0.21, … 0.90*). The results of this accuracy testing may be seen in the figure below:



After examining the results of experimenting with different threshold values, I determined that the most accurate model was produced when the threshold was equal to *0.47* with an accuracy of *71.3%,* a marginally better result, and an accurate model.

Question 10.3.2

Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

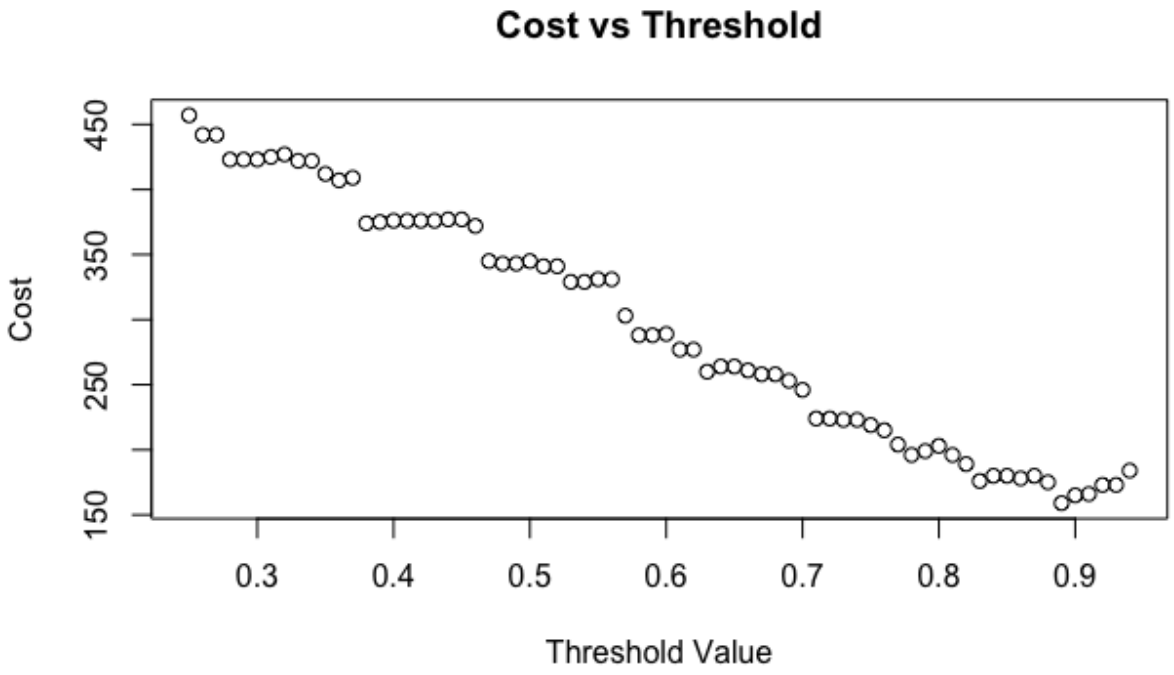
(Please note that my R code for this question is contained in the R file 'homework_7_Q10.3.R', and if you wish to run the code, you will need to change line 5 of my R code to your local directory which contains the 'germancredit.txt' data)

Response

Question 10.3.2 is an example of an optimization problem. We are told that incorrectly labeling a bad customer as good a good customer (a false positive) is 5 times worse (or more costly) than incorrectly labeling a good customer as bad (a false negative). This means that we can define our cost function as follows:

cost = 5(false positive) + 1*(false negative)*

Our goal in this question is to minimize the cost in the above equation by changing the threshold value. In order to do this, I tested all threshold values in the range (*0.25, 0.26, … 0.95*). The cost associated with each of these threshold values may be seen in the figure below:



As may be confirmed by the figure above, the cost is minimized when the threshold is set equal to *0.89,* with a cost of $c = 159$. The confusion matrix associated with setting the threshold equal to *0.89* may be observed below:

		Actual Response	
		0	**1**
Predicted	**0**	94	179
Response	**1**	1	26

Not surprisingly, we may note from the figure above that many points (*179*) are false negatives while only a single point is a false positive. This is because the penalty associated with predicting a false positive is *5* times greater than the penalty associated with predicting a false negative. Therefore, it is no surprise that the cost is minimized by setting the threshold to such a large value, which predicts that many more customers are bad credit risks than good credit risks (*273* negative prediction versus *27* positive predictions). Overall, the accuracy associated with setting the threshold equal to *0.89* is equal to just *40%*, while, as demonstrated in question 10.3.2, we can find a maximum

accuracy of *71.3%* associated with setting the threshold equal to *0.47*. This serves as evidence that the best model is not always the model which produces the highest level of accuracy. In this, case a model with diminished accuracy in fact has the lowest cost.