ECS256 - Final project

Olga Prilepova, Christopher Patton, Alexander Rumbaugh,
John Chen, Thomas Provan

March 12, 2014

Contents

1 Problem 1

As suggested by the hint, in order to find the bias at $t = 0.5$, we're going to be examining the value of β as n goes to ∞. The idea behind this is that the proposed model will converge to some β as the number of samples increases, to the point that it fits as closely as possible to the actual function. Of course, it isn't possible for the model to fit precisely, because the actual relationship isn't linear like the model is, so there will be some bias in the resulting model.

The first thing to consider is model construction. The simplest way to construct the model would be to use a least-squares method, so in the construction we would minimize the following with respect to $g(X)$, which is the set of all possible functions for our model.

$$E[(EY - g(X))^2] \tag{1}$$

For this problem, $g(X) = \beta X$ for all possible β, and the actual distribution of EY is $X^{0.75}$, so we can rewrite this to minimizing the following with respect to β

$$E[(X^{0.75} - \beta X)^2] \tag{2}$$

To actually find this value, we will first make a new variable $Q = (X^{0.75} - \beta X)^2$, and then use the continuous case of iterated expectations along X (eq 5.33) to find EQ

$$EQ = \int_{-\infty}^{\infty} f_X(t)\ E(Q|X = t)\ dt \tag{3}$$

$$E(Q|X = t) = E[(t^{0.75} - \beta t)^2] = (t^{0.75} - \beta t)^2 \tag{4}$$

X is $U(0, 1)$, so $f_X(t)$ is 1 between the values of 0 and 1, and is zero elsewhere. With this and the value of $E(Q|X = t)$, we can rewrite EQ as the following.

$$EQ = \int_0^1 (t^{0.75} - \beta t)^2\ dt \tag{5}$$

From here it's a simple matter to minimize with respect to β. First, finish computing the integral to get the final function of β to minimize.

$$EQ = \frac{1}{3}(\beta^2 - \frac{24}{11}\beta + \frac{6}{5}) \tag{6}$$

And then take the derivative of the function with respect to β.

$$\frac{d\,EQ}{d\beta} = \frac{2}{3}\beta - \frac{8}{11} \tag{7}$$

And finally set that equal to zero and solve for β, which yields $\beta = \frac{12}{11}$ as the value that minimizes our function.

Now we can move on and compute the bias. The bias is simply $\hat{m}_{Y;X}(t) - m_{Y;X}(t)$, which for the selected β and $t = 0.5$ is

$$Bias(0.5) = \beta 0.5 - 0.5^{0.75} \approx \boxed{\text{-0.0491}} \tag{8}$$

2 Problem 2.a

For convenience, we give the call forms of the important routines:

- `prsm(y,x,k=0.01,predacc=ar2,crit="max",printdel=F)`
- prediction accuracy criteria: `ar2(y, x)`, `aiclogit(y, x)`

Our implementation of `prsm()`[1] honors the program's specification. We first calculated the PAC ("prediction accuracy criterion") with all predictor variables. Next, we removed the first attribute from the predictor variables and recalculated the PAC with the new parsimony. If the PAC has improved, or has only degraded marginally, we accept it as our new target PAC and the set of attributes as our new parsimony. We repeat this procedure for the next attribute, using the modified parsmony. This process is repeated for all attributes. For continuous regression, we used the adjusted R^2 for the PAC; for logstic regression, we used the AIC.

One of our goals in this project was to validate this greedy approach to reducing parsimony. Let PAC* be our target prediction accuracy criterion. Using AIC in logistic regression, the goal of `prsm()` is to find a smallest subset of the predictor variables with minimal PAC value, but no greater than $(1+k)\cdot \text{PAC}^*$. (The goal is analogous for continuous regession with adjusted R^2, except we maximize this value.)

It's easy to see that `prsm()` will not find the optimal parsimony in general. Suppose we start with predictor variables $\{X^{(1)}, X^{(2)}, X^{(3)}, X^{(4)}, X^{(5)}\}$ and response variable Y. Suppose at some point we remove $X^{(2)}$ from the parsimony, since `aiclogit`$(Y, \{X^{(1)}, X^{(3)}, X^{(4)}, X^{(5)}\}) < (1+k)\cdot \text{PAC}^*$. If the optimal parsiomny includes $X^{(2)}$, but excludes variables we would delete later on, then we won't ever reach it. We affectively cut off a branch of computation that would have lead to a better, if not globablly optimal solution. Intuitively, we see that the order in which the attributes are successively removed determines the resulting PAC value and parsimony.

To address this apparent problem, we implemented an exhaustive search version, called `prsmpwr()`[2], which minimizes the PAC (or maximizes in the case of continuous regression) over all subsets of the attributes. We iterate over the power set of the predictor variables from smallest to largest. We accept the new parsimony if the PAC is less than $(1+k)\cdot \text{PAC}^*$; however, we only accept a new PAC* if it is strictly less than the old target.

We tested this method against prsm() over many of the data sets we discuss in proceeding sections. Obviously, this wouldn't be feasible for higher dimensional data. If there are p predictor variables, than there are $2^p - 1$ subsets of predictor variables. Nevertheless, our results for the lower dimensional data were encouraging. The following run results were identical for both the parsimony and optimal PAC:

- In the pima data set [1], classifying diabetes patient, predicting age.
- In California house data set [3], predicting median house value, population, and household size.
- In the Abalone data [5], predicting the number of rings, the total weight, and classifying speciment as male.

In fact, the only instance in which `prsm()` had a sub-optimal solution was when we ran logistic regression to classify an abalone as an infant.

```
  prsm() PAC=2150.704
         parsimony=c(‘is_male‘, ‘length‘, ‘viscerra_weight‘, ‘rings‘)

prsmpwr() PAC=2139.02
         parsimony=c(‘is_male‘, ‘length‘, ‘whole_weight‘, ‘viscerra_weight‘, ‘rings‘)
```

[1] See `Parsimony.R` in the project folder. The code is also listed in the appendix of this document.
[2] The code is listed in the appendix.

3 Problem 2.b

First, the results requested from the simulated data in tabular form. The variables listed in the table are the variables that were selected by the specified method for each run.

		prsm(k=0.01)	prsm(k=0.05)	sig test
n=100	Run 1	X_1, X_2, X_3, X_9	X_1, X_2, X_3	X_1, X_2, X_3
	Run 2	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3
	Run 3	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3
n=1000	Run 1	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4
	Run 2	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3
	Run 3	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3
n=10000	Run 1	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4
	Run 2	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4
	Run 3	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4, X_9
n=100000	Run 1	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4
	Run 2	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4, X_9
	Run 3	X_1, X_2, X_3	X_1, X_2, X_3	X_1, X_2, X_3, X_4, X_9

As the table shows, each method consistently keeps the most important 3 X_i's in all of our test runs. The smaller k value seems to be less willing to discard variables at low n values, indicated by it selecting the irrelevant X9 variable, but at higher n values it seems to consistently discard the unimportant variables. The sig test, on the other hand, appears to become more willing to accept irrelevant X's as we increase n. However, it is also worth noting that it starts consistently picking out the relevant, but weakly influencing variable X4, which the parsimony methods seems to always discard at high values of n, and rarely pick up at low values.

However, this analysis felt insufficient to us. Three runs isn't nearly enough to see any patterns, and some results may be pure chance! For example, notice how the only completely irrelevant value it chose was X_9. Each of these trials were independent from each other, so the methods should have been no more likely to select X_9 from any of the five irrelevant variables. This indicated to us that this is not enough trials to show us anything about the data. To investigate further, we ran each analysis method 100 times on distinct samples of our data, and then compiled the following tables. The value in the table is the proportion of our 100 trials that each method selected the particular variable X_i.

prsm(k=0.01)	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
N = 100	1	1	1	0.24	0.11	0.14	0.21	0.22	0.26	0.28
N = 1000	1	1	1	0.08	0	0	0	0	0	0
N = 10000	1	1	1	0	0	0	0	0	0	0
N = 100000	1	1	1	0	0	0	0	0	0	0
N = 1M	1	1	1	0	0	0	0	0	0	0

prsm(k=0.05)	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
N = 100	1	1	0.99	0.1	0.02	0.05	0.04	0.03	0.07	0.02
N = 1000	1	1	1	0	0	0	0	0	0	0
N = 10000	1	1	1	0	0	0	0	0	0	0
N = 100000	1	1	1	0	0	0	0	0	0	0
N = 1M	1	1	1	0	0	0	0	0	0	0

Sig Test	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
N = 100	1	1	1	0.14	0.03	0.05	0.05	0.03	0.09	0.04
N = 1000	1	1	1	0.31	0.02	0.05	0.05	0.05	0.02	0.04
N = 10000	1	1	1	1	0.04	0.01	0.07	0.07	0.03	0.06
N = 100000	1	1	1	1	0.35	0.06	0.09	0.03	0.05	0.03
N = 1M	1	1	1	1	1	0.05	0.03	0.08	0.02	0.03

This gives us a much more complete picture of the situation. As we suggested earlier, it appears that all of the methods very easily find that X_1, X_2, and X_3 are important to the model, and select them in every simulated data set. (with one exception) This is still an uninteresting conclusion, but now it's much more solidly demonstrated than from just three dataruns. More interesting is the behavior of the other columns, which we previously could not say much about, since we only had the three simulations.

First, we will examine the behavior of methods on the two weakly influencing variables, X_4 and X_5. The **prsm** methods seem to be more likely to pick up on these weakly influencing variables at low n values, but discard them completely at higher n values. Additionally, it seems as if setting lower tolerance values increases **prsm**'s ability to pick up on these weakly influencing values. However, this only seems to take place at reasonable n. At $n = 100$, **prsm** seems to lose the weakly influencing variables regardless of k value. We did test $k = 0.005$ briefly to explore this. At $n = 100$, **prsm** didn't select the weak values any more often than the irrelevant variables, but at $n = 1000$ it selected the weak variable X_4 significantly more often than the irrelevant variables. As with other k values, however, this proportion dropped as n increased. The significance test, on the otherhand, started to pick up the weakly influencing variables as n increased. For the highest n, the significance test always selected the weakly relevant variables.

Now we will look at the second group, the irrelevant variables. **prsm** very quickly discarded all of the irrelevant variables as n increased. For both k values we looked at (and the smaller k we examined briefly), **prsm** selected none of the irrelevant variables by $n = 1000$. Significance testing, however, never fully stopped selecting irrelevant variables. Even at our highest n, the significance test still selected irrelevant variables as part of the predictor, and with roughly the same probability as it had at lower n. Increasing n seems to have no impact on significance test's likelyhood to select one of the irrelevant variables.

Between the two, this allows us to draw some interesting contrast between the two models. The **prsm** models very quickly zero in on the most critical variables for creating it's estimate and remove irrelevant variables with a comparatively low sample size, but can lose subtle variables with small impacts on the distribution. In contrast, significance testing will consistently find relevant variables, getting better at finding and including subtle effects as n increases, but at the cost of being unable to fully exclude irrelevant variables at any point.

Perhaps this suggests another model, where we use these two models to cover each others weaknesses. **prsm** is very good at finding the most relelvant variables, and seems to be able to easily point out the most

important variables for a model to consider. But if we need to consider finer aspects, perhaps we could use the Sig Test method (or something similar and a little more sophisticates) to aid in finding subtle variables to add to the **prsm**'s list.

4 Problem 2.c

4.1 Abalone, [5] (continuous and logistic, small p, small n)

The Abalone data set [3] is fairly small in the number of observations and dimensions. The only variable that wasn't numeric was Sex. We changed the nominal Sex variable from F,M,I (Male, Female or Infant) into is.male and is.infant (each 0-1 valued). For logistic regression we used is.infant variable as the response variable.

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	is.male, length, viscerra.weight, rings	is.male, shell.weight	length, diameter, whole.weight, viscerra.weight, rings
AIC	2150.704	2303.834	3318.3

The AIC reported after using our parsimony function is lower for k=0.01, and the number of explanatory variables remaining is smaller than the number of significant explanatory variables according to significance testing performed. This is one of those cases where parsimony function works better than significance testing in identifying a smaller subset of important explanatory variables.

Shell Weight was chosen as the response variable for continuous case analysis.

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	whole.weight, shucked.weight	whole.weight	diameter, height, whole.weight, shucked.weight, viscerra.weight, rings
Adjusted RSquared	0.9440963	0.9126831	0.953487

Our parsimony function performs almost as well as the significance testing approach and it is using only 2 explanatory variables. It works very well for this small and very straightforward data set.

4.2 Median House Values [3] (continuous, small p, large n)

Our next data set explores house value throughout California derived from 1990 census data [4]. The state is divided up into 20640 geographic blocks with average population 1425.

We are interested in how the following variables affect median house value: median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude.

Here is R's summary output of the data:

```
Median.House.Value Median.Income       Median.Age
 Min.   : 14999     Min.   : 0.4999   Min.   : 1.00
 1st Qu.:119600     1st Qu.: 2.5634   1st Qu.:18.00
 Median :179700     Median : 3.5348   Median :29.00
 Mean   :206856     Mean   : 3.8707   Mean   :28.64
```

[3] See `abaloneAnalysis.R` and `abaloneAnalysisLog.txt` in the project folder. `Parsimony.R` has to be loaded first.
[4] See `cadata.R` and `plotmap.R` in the project folder. `Parsimony.R` has to be loaded first

```
 3rd Qu.:264725     3rd Qu.: 4.7432   3rd Qu.:37.00
 Max.   :500001     Max.   :15.0001   Max.   :52.00
Total.Rooms    Total.Bedrooms     Population
 Min.   :    2   Min.   :   1.0   Min.   :    3
 1st Qu.: 1448   1st Qu.: 295.0   1st Qu.:  787
 Median : 2127   Median : 435.0   Median : 1166
 Mean   : 2636   Mean   : 537.9   Mean   : 1425
 3rd Qu.: 3148   3rd Qu.: 647.0   3rd Qu.: 1725
 Max.   :39320   Max.   :6445.0   Max.   :35682
 Households        Latitude       Longitude
 Min.   :   1.0   Min.   :32.54   Min.   :-124.3
 1st Qu.: 280.0   1st Qu.:33.93   1st Qu.:-121.8
 Median : 409.0   Median :34.26   Median :-118.5
 Mean   : 499.5   Mean   :35.63   Mean   :-119.6
 3rd Qu.: 605.0   3rd Qu.:37.71   3rd Qu.:-118.0
 Max.   :6082.0   Max.   :41.95   Max.   :-114.3
```

Next we use our Parsimony package on the data at various k levels. Here are the calls and output:

```
 > prsm(df[,1],df[,2:9],k=0.01, predacc=ar2, crit='max', printdel=T)
full outcome =  0.6369649
deleted         Total.Rooms
new outcome  =  0.6350863
deleted         Total.Bedrooms
new outcome  =  0.6321316
[1] "Median.Income" "Median.Age"    "Population"    "Households"    "Latitude"      "Longitude"
```

```
> prsm(df[,1],df[,2:9],k=0.05, predacc=ar2, crit='max', printdel=T)
full outcome =  0.6369649
deleted         Median.Age
new outcome  =  0.6243571
deleted         Total.Rooms
new outcome  =  0.6218261
deleted         Total.Bedrooms
new outcome  =  0.6198323
[1] "Median.Income" "Population"    "Households"    "Latitude"      "Longitude"
```

By loosening our k value, the model selector decides to remove Median Age in the latter run. It is interesting to note that longitude and latitude remain in the model, which we will address later.

Next we want to compare our Parsimony package to the significance testing approach.

Here is the summary of the coefficients of our lm() call in R:

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -3.594e+06  6.254e+04 -57.468  < 2e-16 ***
Median.Income   4.025e+04  3.351e+02 120.123  < 2e-16 ***
Median.Age      1.156e+03  4.317e+01  26.787  < 2e-16 ***
Total.Rooms    -8.182e+00  7.881e-01 -10.381  < 2e-16 ***
Total.Bedrooms  1.134e+02  6.902e+00  16.432  < 2e-16 ***
Population     -3.854e+01  1.079e+00 -35.716  < 2e-16 ***
Households      4.831e+01  7.515e+00   6.429 1.32e-10 ***
Latitude       -4.258e+04  6.733e+02 -63.240  < 2e-16 ***
Longitude      -4.282e+04  7.130e+02 -60.061  < 2e-16 ***
```

The significance testing approach deletes no predictors. This data set has a large size (20640) so it makes sense that the predictors are all deemed significant.

Here is a summary of the comparison of methods:

Method	Parsimony (k=0.01)	Parsimony (k=0.05)	Significance Testing
Columns Deleted	Total Rooms Total Bedrooms	Total Rooms Total Bedrooms Median Age	None
Adjusted RSquared	0.6321316	0.6218261	0.6369649

Before discussing conclusions about the data, we address latitude and longitude. Looking at the full summary above, latitude and longitude both have negative coefficient values. This says if we travel east or north, on average house value decreases. However a real estate expert would think you are crazy hearing that statement.

The problem arises with using coordinate data in a linear regression model. Without any interaction terms we are assuming the statement above. We would risk overfitting if we attempted to make a complex formula involving coordinates.

A block's location is clearly important. We can see what blocks are close to one another, and proximity to oceans or mountains. Using location data effectively can lead to powerful models, however our model is not set up to do this. But we can plot our data atop a map which we will show shortly.

We remove latitude and longitude and use the predictors kept by our Parsimony package. Here is the summary of the regression:

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   -32165.268   2167.358  -14.84   <2e-16 ***
Median.Income  43094.918    284.263  151.60   <2e-16 ***
Median.Age      2000.544     45.080   44.38   <2e-16 ***
Population       -43.045      1.127  -38.20   <2e-16 ***
Households       152.700      3.344   45.66   <2e-16 ***
```

First we see that income positively affects house value which is unsurprising. If people have more income, they can afford more expensive houses.

Next looking at age, we see it also positively affects house value. One would expect older people to have more money and this reflected in the high co-variance between these predictors.

Moving onto population, we see it negatively affects house value. With this data set each block has approximately similar geographic size, so blocks with higher population can be assumed to be denser. From our model we can conclude less dense areas have higher housing value.

Finally we look at number of households which positively affects house value. This is a bit surprising since population had a negative coefficient. One possible explanation is that our model is detecting large suburban areas which are all houses. Areas with more apartment complexes would fewer number of households and also lower housing value.

Parsimony simplified our model down to four predictors. While we lost the data about rooms, we still have a model we can clearly understand.

Now we make use of our coordinate data using R's ggmap library. First we looked at a map of California as a whole, then zoomed into the Bay Area and the Greater Los Angeles area as seen in Figure 1.

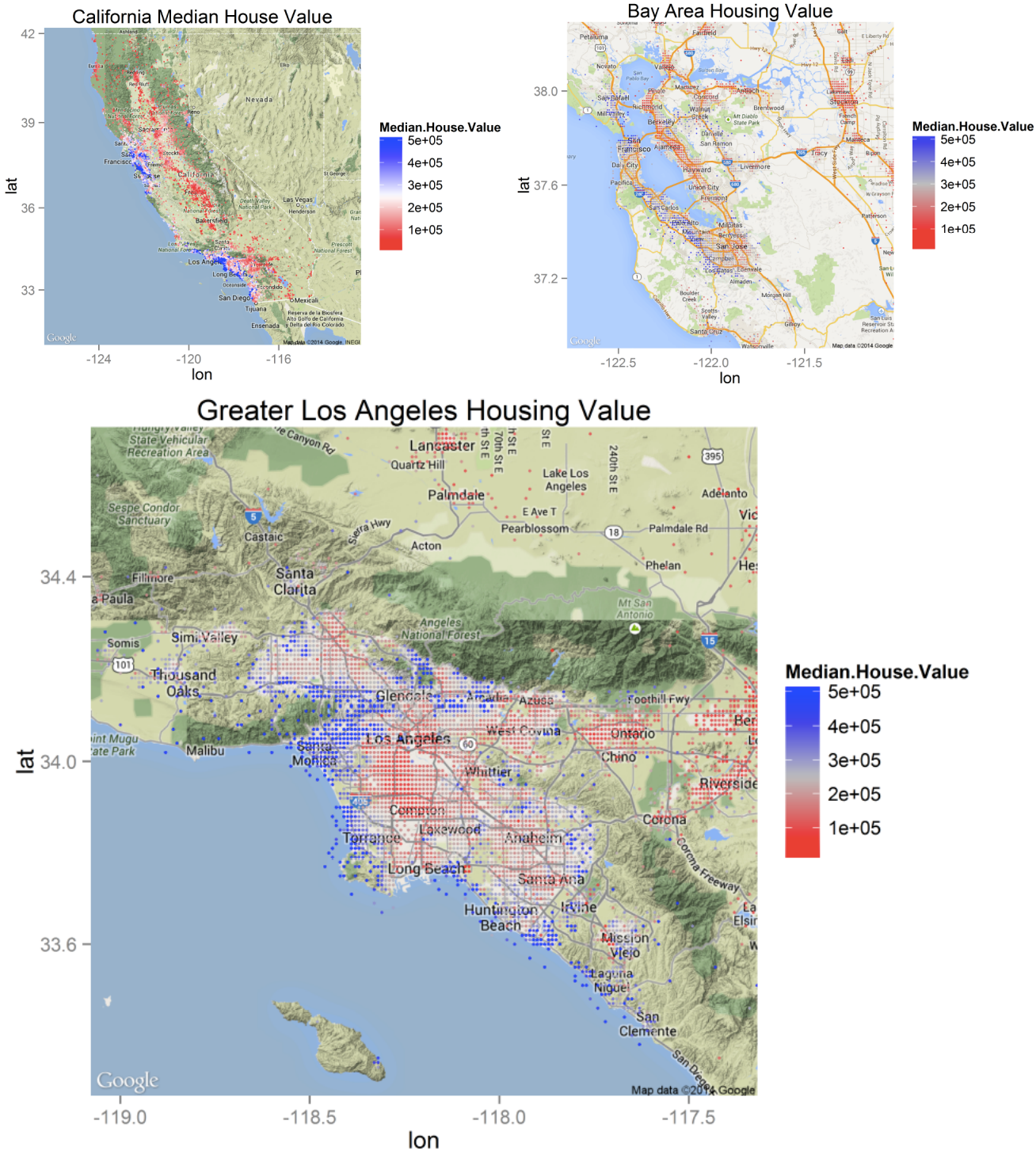

Figure 1

The visualization helps us see how housing value is clustered in the state. As one might expect, rural areas generally have lower housing value than metropolitan areas. Within the metropolitan areas there are visible divisions as well.

4.3 Space Shuttle [4] (logistic, small p, large n)

Next we looked at space shuttle data [5] taken from the UCI Machine Learning Repository. The goal was to predict the class given 8 predictors, with 43500 data points. Our logistic regression predicted whether or not the class was of type 1. The description of the data set was very poor, however it matched our criterion for dimension, data points, and regression type. Because of the lack of sufficient description, few conclusions can be drawn from the data, but we can still test our Parsimony package.

Here is the summary of results using Parsimony:

Method	Parsimony (k=0.01)	Parsimony (k=0.05)	Significance Testing
Columns Deleted	V1,V3,V4,V6,V9	V1,V2,V3,V4,V6,V9	V4
AIC	8575.803	8728.906	8475.2

We also tested the various methods against a validation training set. We looked at the conditional accuracy of our model, the probability that it correctly identified members of a given class, and the true positive and negative rates of our model.

Here is a summary of the results:

Method	Parsimony (k=0.01)	Parsimony (k=0.05)	Significance Testing
$P(\hat{x} == 1 \mid x == 1)$	0.9862345	0.9861474	0.9864959
$P(\hat{x} == 0 \mid x == 0)$	0.9285242	0.9285242	0.9169424
$P(x == 1 \mid \hat{x} == 1)$	0.981276	0.9812744	0.9783135
$P(x == 0 \mid \hat{x} == 0)$	0.9466937	0.9463744	0.9470267

The results for the different methods are very close and it is hard to detect much difference in the accuracy levels. There were slightly fewer false positives and there was a slightly better detection of negatives for the Parsimony models. By simplifying the model, there is a possibility of increased accuracy during validation runs.

4.4 Census, 1994 [2] (continuous and logistic, large p, large n)

Here we are analyzing Census data [6] which contains 32561 observations, 13 variable vectors and is based on the 1994 Census. There are several categorical variables in this data set. In order to accommodate regression modeling each categorical vector was split into 0-1 valued vectors for each category. According to the website from which the data set was taken the goal was to predict whether a given person would be making over 50K a year. Our parsimony function at k=0.1 eliminated all but one explanatory variable, which turned out to be "capital gain".

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.269e-01  2.334e-03   97.19   <2e-16 ***
capital.gain 1.293e-05  3.128e-07   41.34   <2e-16 ***
---
```

This makes sense. If we were to look at only one column in order to predict if a person is making over 50K a year, the capital gain variable would be most informative. At k=0.01 there are many more explanatory variables that enhance the predictions: "age", "education.num", "Exec.managerial", "Not.in.family", "Own.child", "Unmarried", "Wife", "capital.gain", "capital.loss", "hours.per.week".

[5] See `shuttle.R` in the project folder. `Parsimony.R` has to be loaded first.

[6] See `censusAnalysis.R` and `censusAnalysisLog.txt` in the project folder. `Parsimony.R` has to be loaded first

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     -3.507e-01  1.284e-02 -27.323   <2e-16 ***
age              3.349e-03  1.602e-04  20.910   <2e-16 ***
education.num    4.195e-02  7.820e-04  53.650   <2e-16 ***
Exec.managerial  1.259e-01  6.036e-03  20.861   <2e-16 ***
Not.in.family   -2.774e-01  4.918e-03 -56.396   <2e-16 ***
Own.child       -2.439e-01  6.692e-03 -36.448   <2e-16 ***
Unmarried       -2.826e-01  6.702e-03 -42.167   <2e-16 ***
Wife             8.008e-02  9.369e-03   8.547   <2e-16 ***
capital.gain     8.576e-06  2.655e-07  32.304   <2e-16 ***
capital.loss     1.004e-04  4.836e-06  20.768   <2e-16 ***
hours.per.week   3.472e-03  1.665e-04  20.858   <2e-16 ***
---
```

Another interesting variable that can be of interest for predictions is "Sex", which can be only "Male" or "Female" in the scope of this census. With k=0.01 the following explanatory variables aren't eliminated: "Widowed", "Craft.repair", "Farming.fishing", "Handlers.cleaners", "Transport.moving", "Not.in.family", "Other.relative", "Own.child", "Unmarried", "Wife". Some of them make perfect sense, such as the "Wife" variable. If somebody is identified as a wife in 1994 Census it is very likely that the person was female. "Craft.repair", "Farming.fishing", and "Transport.moving" variables make sense too, and we expect to see a strong negative correlation between those jobs and being female in the 1994.

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	"Widowed", "Craft.repair", "Farming.fishing", "Handlers.cleaners", "Transport.moving", "Not.in.family", "Other.relative", "Own.child", "Unmarried", "Wife"	"Craft.repair", "Not.in.family", "Other.relative", "Own.child", "Unmarried", "Wife"	34 reported significant
AIC	20500.69	22383.32	19140

```
Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)      -9.11547    0.99672  -9.145   <2e-16 ***
Widowed           1.44049    0.09539  15.102   <2e-16 ***
Craft.repair     -2.49589    0.08198 -30.445   <2e-16 ***
Farming.fishing  -2.63170    0.15941 -16.509   <2e-16 ***
Handlers.cleaners -2.06744    0.09489 -21.787   <2e-16 ***
Transport.moving -2.61252    0.13222 -19.759   <2e-16 ***
Not.in.family     9.25812    0.99701   9.286   <2e-16 ***
Other.relative    9.26290    0.99924   9.270   <2e-16 ***
Own.child         9.26914    0.99719   9.295   <2e-16 ***
Unmarried        10.71474    0.99781  10.738   <2e-16 ***
Wife             16.18651    1.22313  13.234   <2e-16 ***
---
```

Fitting the linear regression model to the data works as predicted, which supports our intuitive expectations of explanatory variable behavior. Naturally, the remaining variables are significant, however with large number of observations most variables are expected to be significant.

With the continuously defined variable an interesting one to predict was "Age". After running our parsimony function with k=0.01, the variables that explain the variance in "Age" are "Self.emp.not.inc", "Assoc.acdm", "Never.married", "Widowed", "Own.child", "hours.per.week", "salary".

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       44.927608   0.230145 195.215  < 2e-16 ***
Self.emp.not.inc   4.313808   0.224466  19.218  < 2e-16 ***
Assoc.acdm        -1.475503   0.335395  -4.399 1.09e-05 ***
Never.married    -10.851233   0.154146 -70.396  < 2e-16 ***
Widowed           16.070178   0.354766  45.298  < 2e-16 ***
Own.child         -8.221388   0.194804 -42.203  < 2e-16 ***
hours.per.week    -0.073794   0.005135 -14.371  < 2e-16 ***
salary             2.904975   0.150546  19.296  < 2e-16 ***
---
```

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	"Self.emp.not.inc", "Assoc.acdm", "Never.married", "Widowed", "Own.child", "hours.per.week", "salary"	"Never.married", "Widowed", "Own.child"	Many reported significant
Adjusted RSquared	0.3763604	0.3601	0.428

The significance testing approach has a larger subset of explanatory variables than the parsimony function has selected and a slightly higher significance, which unfortunately is still not very high.

Since many of the explanatory variables are categorical and age is not correlated in any straightforward way with the explanatory variables the model cannot be a perfect fit as seen in Figures 2, 3. In fact there are many observations that have a very big impact on the model as Cook's distance plot shows (Figure 4).

Since there are many more observations than we can visually assess, we decided to split the original data set in training(90%) and test(10%) subsets and to fit the train data but report the predicted versus actual values in the test data (Figure 5). Now it is more clear that the age is clustered around 24, 32 and 40-45. The explanatory variables used aren't very good predictors for the person's age, unfortunately.

In order to consider the contribution of explanatory variables to the variance of age we sorted the 7 variables in order of their absolute correlation value with age. Then we adapted parsimony function to only work with this set and in the provided order so that we can produce and visualize the contribution of each variable in the most to least significance order as seen in Figure 6.

In all of the performed analyses the country of origin doesn't seem to play much of a role, or, perhaps, the country data is too sparse to be a good predictor of the outcome variables. Also it is interesting to point out that education isn't highly correlated with the variables of interest when all of the explanatory variables are considered, since it didn't make it into the final models at k=0.01. It would be very interesting to compare these results with a more modern census information.

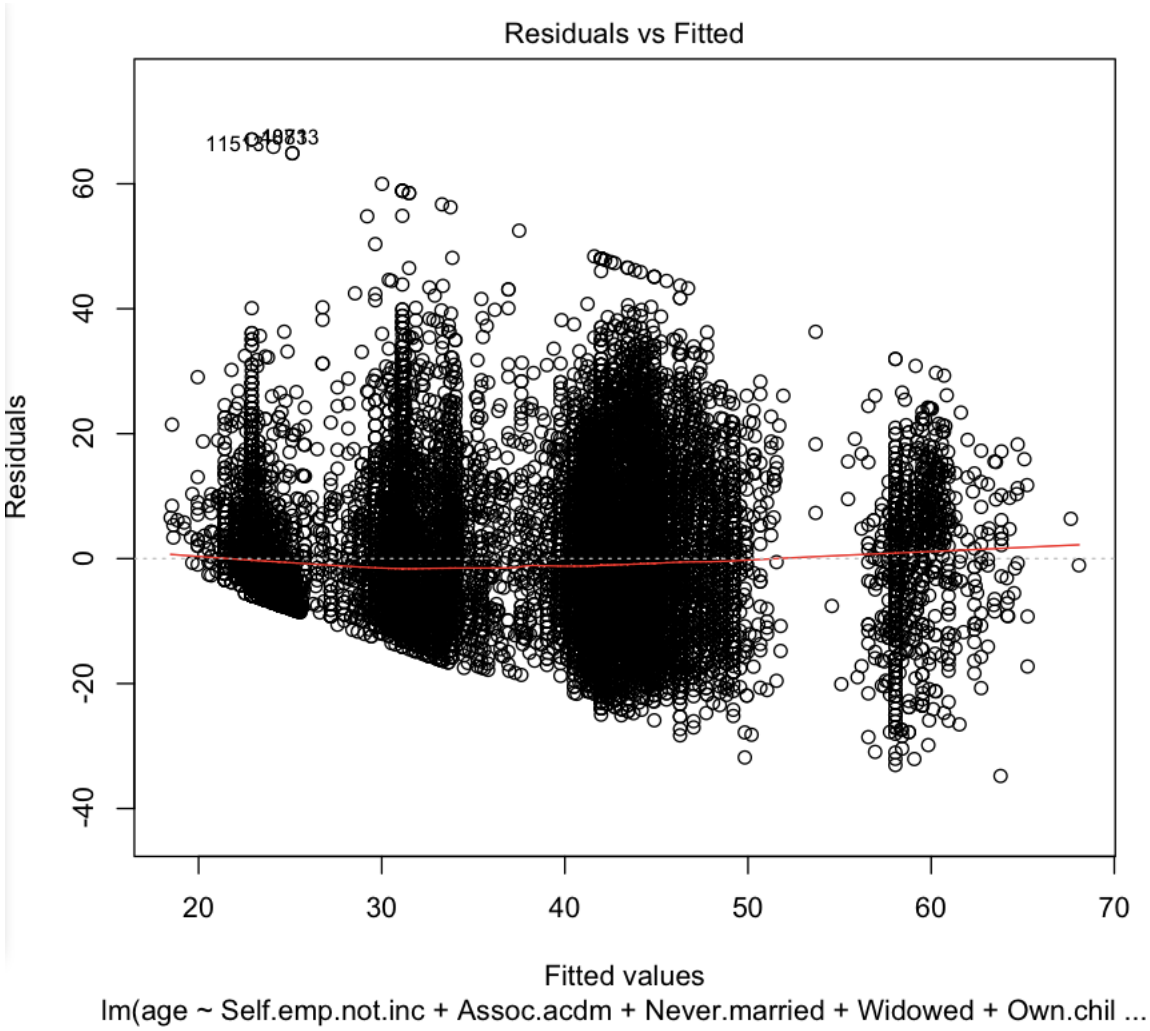
Residuals vs Fitted
Residuals
Fitted values
lm(age ~ Self.emp.not.inc + Assoc.acdm + Never.married + Widowed + Own.chil ...
-40
-20
0
20
40
60
20
30
40
50
60
70

Figure 2

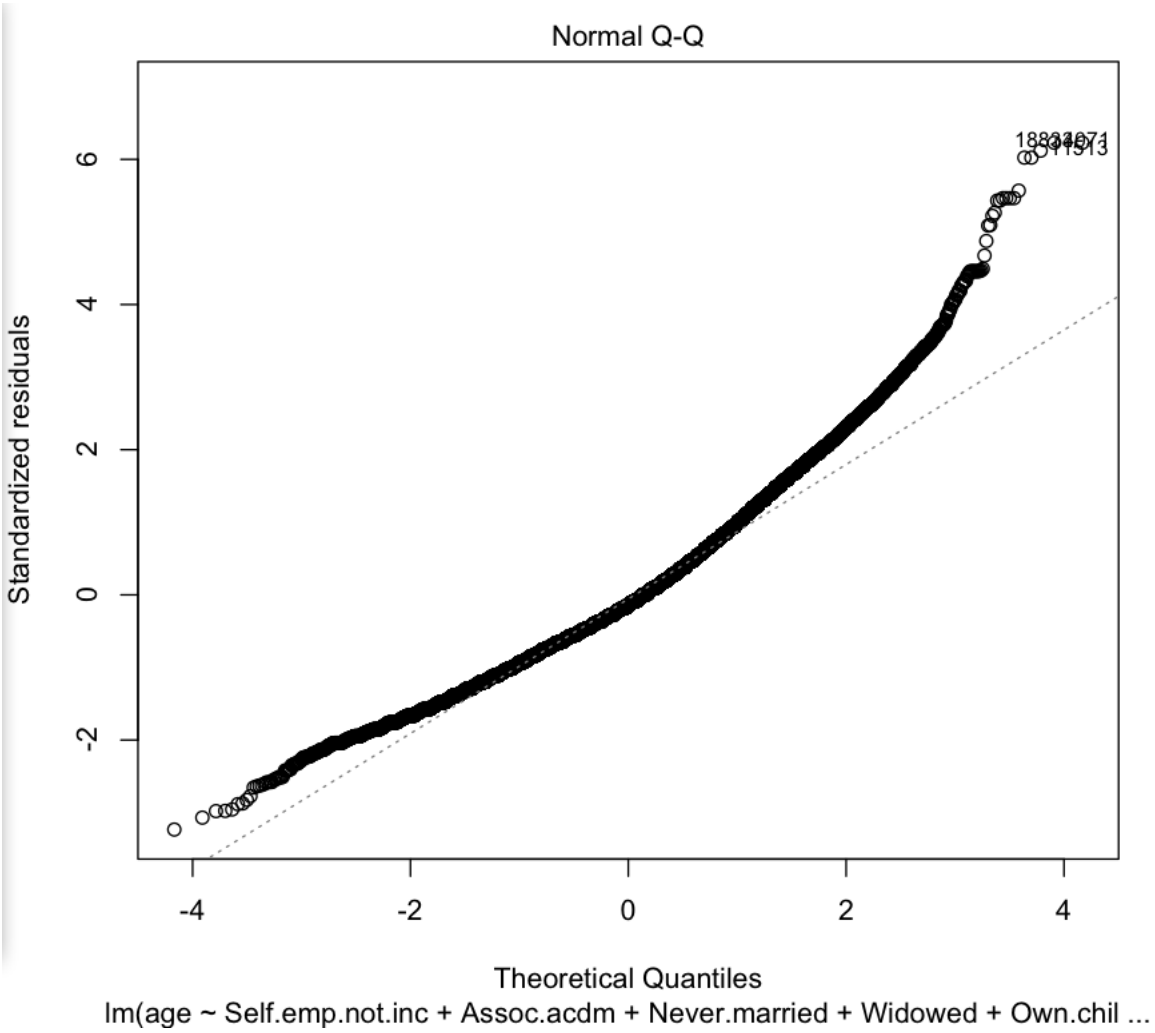
Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(age ~ Self.emp.not.inc + Assoc.acdm + Never.married + Widowed + Own.chil ...

Figure 3

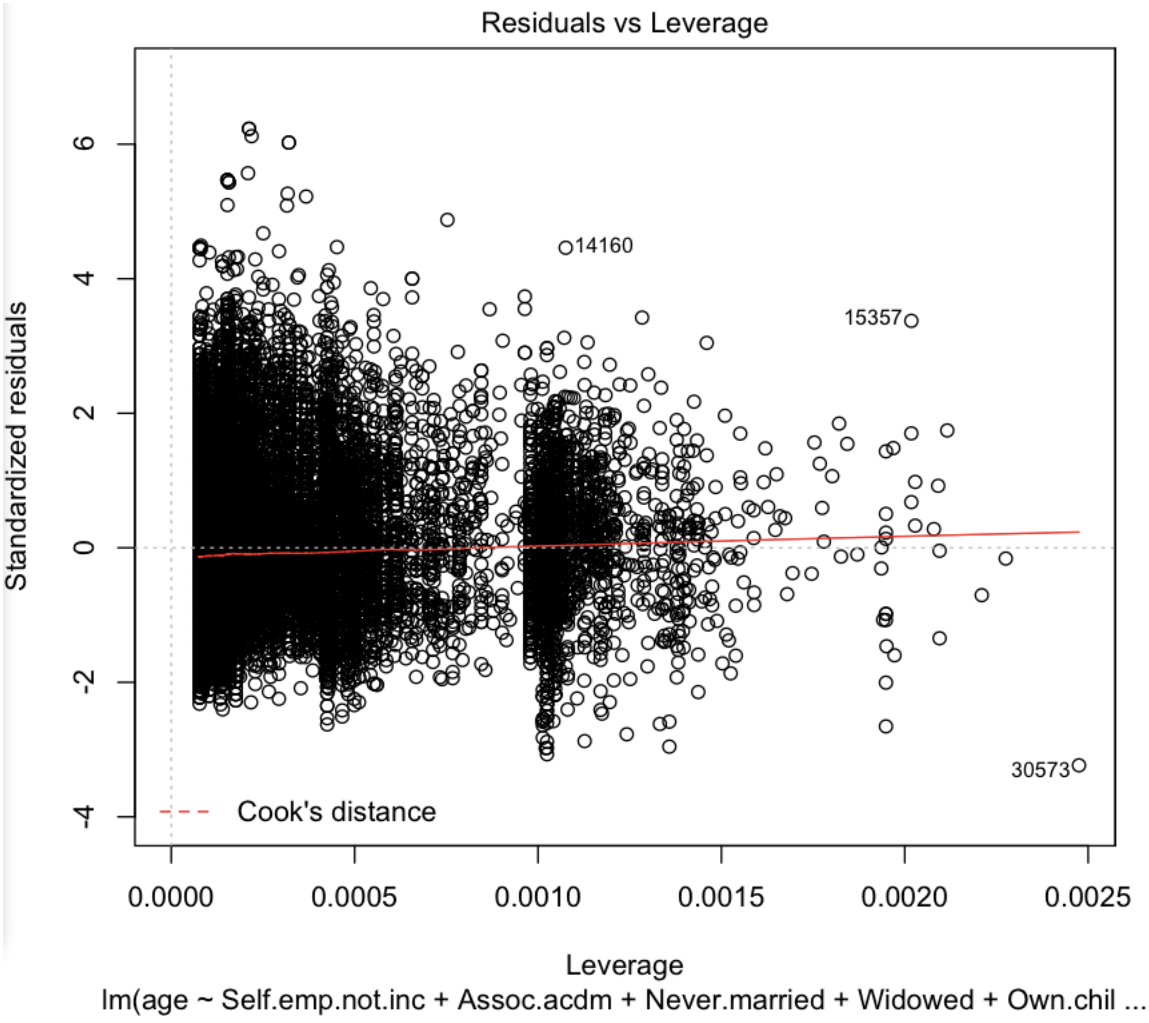
Residuals vs Leverage
Standardized residuals
6
4
2
0
-2
-4
14160
15357
30573
Cook's distance
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
Leverage
lm(age ~ Self.emp.not.inc + Assoc.acdm + Never.married + Widowed + Own.chil ...

Figure 4

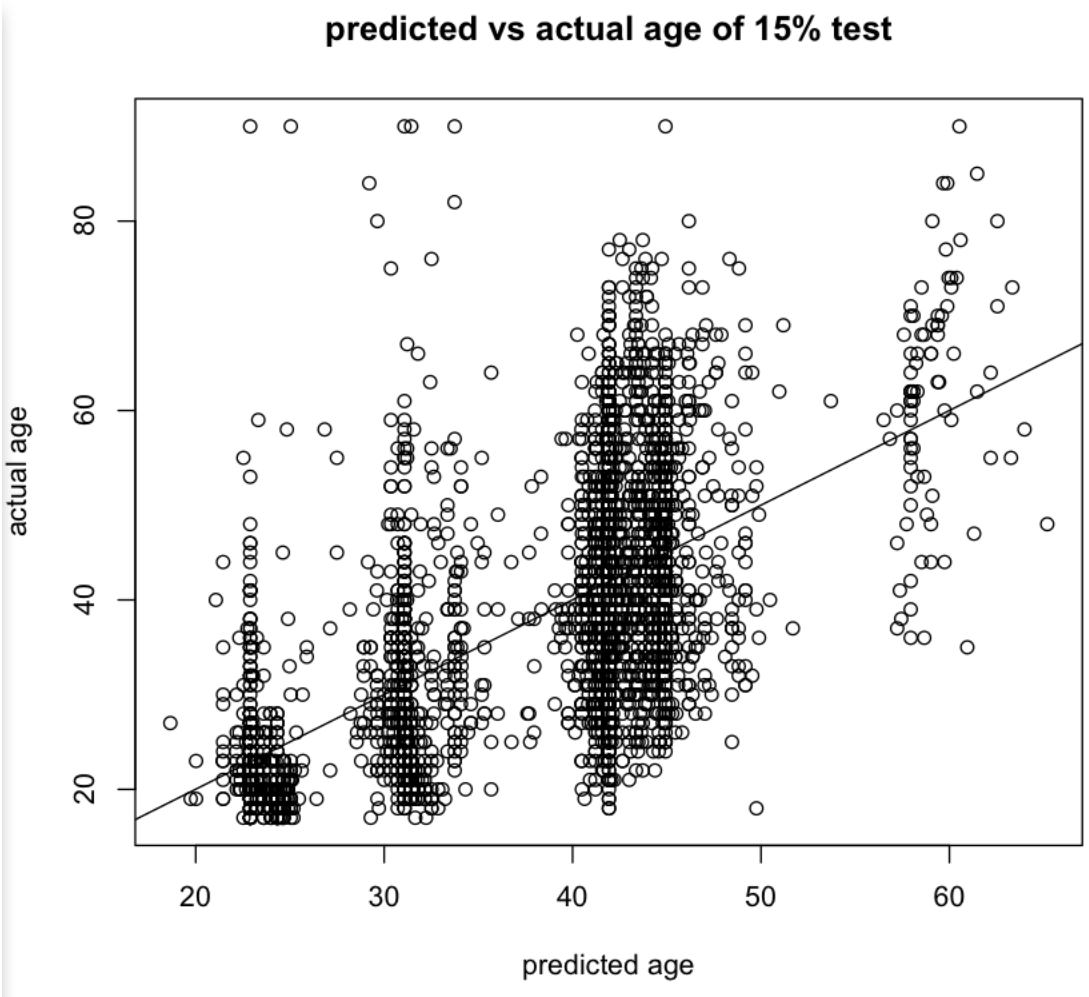
predicted vs actual age of 15% test
actual age
predicted age
20
40
60
80
20
30
40
50
60

Figure 5

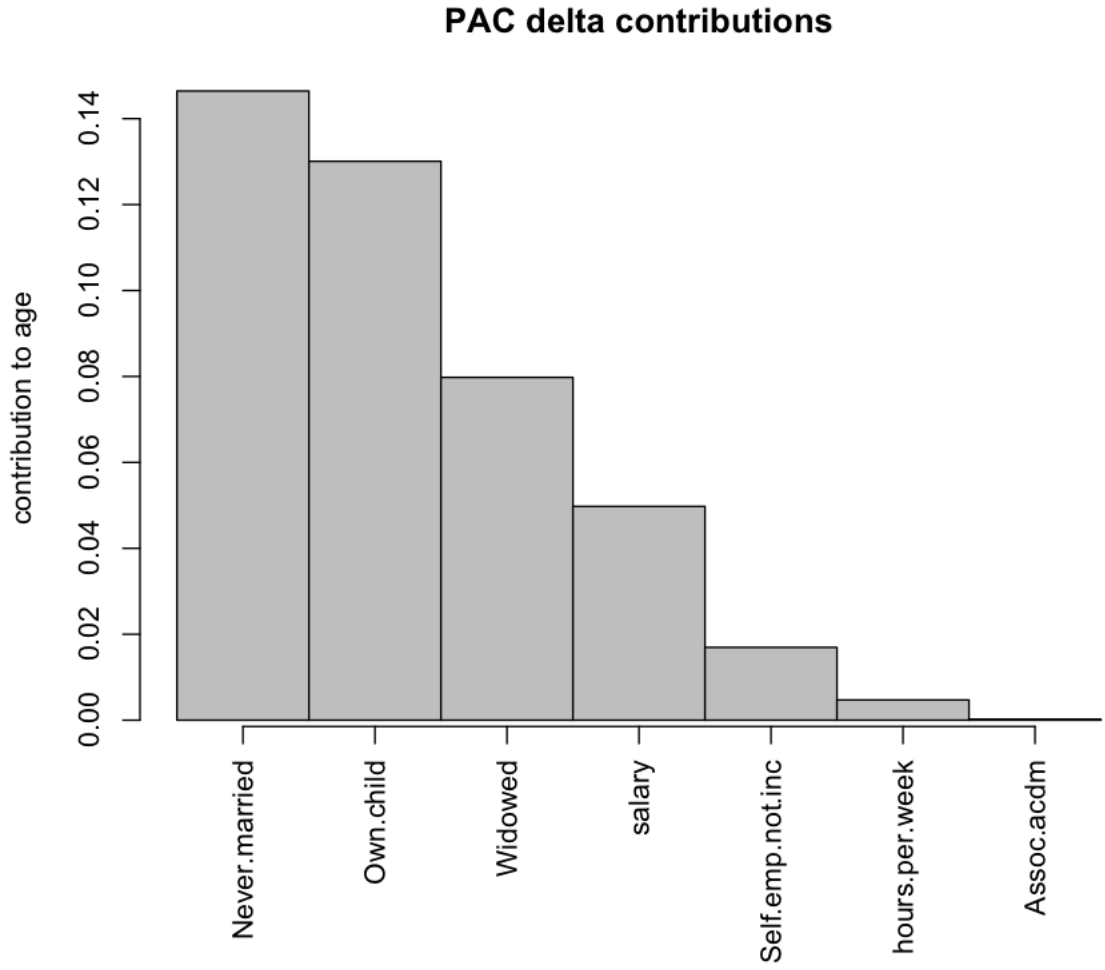
PAC delta contributions
contribution to age
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Never.married
Own.child
Widowed
salary
Self.emp.not.inc
hours.per.week
Assoc.acdm

Figure 6

4.5 Automobile Data Set [6] (continuous and logistic, large p, small n)

For the small n (n <1000), large p (p >15) case, we will use the automobile data [7] set found on the UCI Machine Learning Repository. This data set contains 193 instances of automobile, with 25 categories of classification for each one.

We chose the price of each vehicle as our continuous response variable. Calling **prsm()** gives us a list of the most accurate predictors of a vehicle's price. The variables have been listed in the table below, along with the results of significance testing :

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	ohcv, twelve-cylinders, engine.size, stroke, compression.ratio, peak.rpm	engine.size	bmw, dodge, 'mercedes-benz', mitsubishi, plymouth, porsche, saab, std, front, wheel.base, length, width, height, curb.weight, dohc, ohc, engine.size, peak.rpm
AIC	0.8676842	0.7888274	0.9308

The results of the running **prsm()** tells us that a vehicle's price is ultimately determined by its underlying components' ability to perform - especially that of the engine. This is unsurprising, as the engine and transmission of an automobile are generally its most expensive components. Running significance tests yielded a larger list of predictor variables. Significance testing shows that larger automobiles as well as luxury brands like BMW, Mitsubishi, and Porsche tend to command higher prices.

```
Summary of Significance Testing:

                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     -4.234e+04  1.125e+04  -3.764 0.000229 ***
bmw              9.290e+03  8.611e+02  10.788  < 2e-16 ***
dodge           -1.504e+03  8.532e+02  -1.762 0.079785 .
'mercedes-benz'  6.644e+03  1.003e+03   6.625 4.17e-10 ***
mitsubishi      -2.628e+03  7.331e+02  -3.585 0.000438 ***
plymouth        -1.628e+03  8.881e+02  -1.833 0.068485 .
porsche          4.053e+03  2.238e+03   1.811 0.071936 .
saab             2.413e+03  1.028e+03   2.347 0.020043 *
std             -1.109e+03  5.129e+02  -2.162 0.031973 *
front           -1.275e+04  2.663e+03  -4.785 3.63e-06 ***
wheel.base       1.141e+02  7.390e+01   1.544 0.124355
length          -7.918e+01  4.225e+01  -1.874 0.062586 .
width            7.652e+02  2.029e+02   3.772 0.000222 ***
height          -1.377e+02  1.164e+02  -1.183 0.238332
curb.weight      3.781e+00  1.118e+00   3.381 0.000890 ***
dohc             1.569e+03  8.067e+02   1.944 0.053451 .
ohc              8.531e+02  4.575e+02   1.865 0.063911 .
engine.size      7.733e+01  1.035e+01   7.470 3.74e-12 ***
peak.rpm         1.522e+00  3.938e-01   3.864 0.000157 ***
---
Residual standard error: 2128 on 174 degrees of freedom
Multiple R-squared:  0.9373,Adjusted R-squared:  0.9308
F-statistic: 144.5 on 18 and 174 DF,  p-value: < 2.2e-16
```

[7] See `car_analysis.R` in the project folder. `Parsimony.R` has to be loaded first

For the logistic regression case, we seek to find most accurate predictors when it comes to determining how safe a car is. In the original data set, each car is assigned a safety rating ranging from -3 to 3. This range of values was converted into a discreet indicator variable: 1 if the car is relatively safe, having scored 1 or higher on the original scale, and 0 otherwise.

prsm() determined the most accurate predictor variables as follows:

Method	Parsimony (k = 0.01)	Parsimony (k = 0.05)	Significance Testing
Columns Retained	saab, toyota, volkswagen, turbo, two-doors, hatchback, sedan, 4wd, rwd, rear, wheel.base, length, width, height, curb.weight, l, ohc, ohcf ,ohcv, five-cylinders, four-cylinders, three-cylinders, twelve-cylinders, engine.size, 2bbl, idi, mfi, mpfi, spdi, bore, stroke, compression.ratio, horsepower, peak.rpm, city.mpg, highway.mpg	saab, toyota, volkswagen, turbo, two-doors, hatchback, sedan, 4wd, rwd, rear, wheel.base, length, width, height, curb.weight, l, ohc, ohcf ,ohcv, five-cylinders, four-cylinders, three-cylinders, twelve-cylinders, engine.size, 2bbl, idi, mfi, mpfi, spdi, bore, stroke, compression.ratio, horsepower, peak.rpm, city.mpg, highway.mpg	audi, saab, volkswagen, diesel, std, four-doors, 4wd, fwd, 1bbl
AIC	74	74	130.24

Interestingly, changing k from 0.01 to 0.05 did nothing to the resulting set of predictor variables. It would appear that **prsm()** is not well suited toward classifying data sets with large **p** and small **n**. Fortunately, significance testing yielded far more usable results. Significance tests indicate that a car with four doors, front wheel drive, or was manufactured by Volkswagen tends to be higher safety rating, as rated by insurers.

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    2.5122     1.1216   2.240  0.02510 *
audi          20.3574  2027.3521   0.010  0.99199
saab          17.7446  1985.9220   0.009  0.99287
volkswagen     1.8112     0.9634   1.880  0.06011 .
diesel        -2.0155     1.2716  -1.585  0.11297
std           -0.4196     1.0765  -0.390  0.69668
‘four-doors‘  -5.9725     1.1293  -5.288 1.23e-07 ***
‘4wd‘         -0.1377     2.1849  -0.063  0.94976
fwd            3.3028     1.1093   2.977  0.00291 **
‘1bbl‘        -4.4965     1.4035  -3.204  0.00136 **
---
Null deviance: 266.06  on 192  degrees of freedom
Residual deviance: 110.24  on 183  degrees of freedom
AIC: 130.24
```

5 Problem 2.d

See `Leave.R` for the code.

Our results are below. The only potentially interesting difference is that this PAC discards one additional variable, Genetics or Pedigree. (depending on where the label came from) This was the biggest loss the Leave-One-Out PAC took, but it still cost less than a percent of accuracy. Perhaps it was just outside of the k-value range for **aiclogit** to throw it out, or this PAC is slightly compressed compared to **aiclogit**.

```
> leaveTest()
full outcome =  0.7682292
deleted         blood_pressure
new outcome  =  0.7669271
deleted         skin_thickness
new outcome  =  0.7669271
deleted         insulin
new outcome  =  0.7695312
deleted         pedigree
new outcome  =  0.7630208
deleted         age
new outcome  =  0.7630208
[1] "pregnancies"     "plasma"          "body_mass_index"
```

References

[1] Instances of diabetes in the Pima Indians. $p = 9, n = 769$ Stored locally in **pima.csv**. Downloaded from `https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/`

[2] Adult Data Set, compiled from Census of 1994. $p = 14 \rightarrow 110$ after factor variables expanded into their own 0-1 vectors, $n = 32561$ Stored locally in **census.csv**. Downloaded from `https://archive.ics.uci.edu/ml/datasets/Adult`

[3] California Housing Data. $p = 8, n = 20640$. Stored locally in **cadata.csv**. Downloaded from `http://lib.stat.cmu.edu/datasets/houses.zip`

[4] Shuttle Data. $p = 9, n = 58000 \rightarrow 43500$ for training, 14500 for testing. Stored locally in **shuttle_training.csv** and **shuttle_testing.csv**. Downloaded from `https://archive.ics.uci.edu/ml/datasets/Statlog+%28Shuttle%29`

[5] Abalone Data Set. $p = 8, n = 4177$. Stored locally in **abalone.csv**. Nominal Sex variable changed from F,M,I (Male, Female or Infant) into is.male and is.infant (each 0-1 valued). Downloaded from `https://archive.ics.uci.edu/ml/datasets/Abalone`

[6] Automobile Data Set. $p = 26 \rightarrow 70$ after factor variables expanded into their own 0-1 vectors, $n = 205 \rightarrow 194$ after NA removal. Stored locally in **imports-85.csv**. Downloaded from `https://archive.ics.uci.edu/ml/datasets/Automobile`

Appendices

Our implementation of of `prsm()`, `ar2()`, and `aiclogit()`.

```
# Perform linear regression on the response variable 'y' and a subset of the
# predictor variables. 'y' is a vector and 'x' is a data.frame. The columns
# Return the predictor accuracy criterion (PAC) value. (Calls lm().)
```

```
ar2 <- function(y, x)
{
  a <- summary(lm(y ~ ., data=x))
  return (a$adj.r.squared)
}

# Same signature as ar2(), but logistic linear regression. 'y' is an indicator
# random variable. (Calls glm().)
aiclogit <- function(y, x)
{
  a = glm(formula = y ~ ., data=x, family = binomial, control=list(maxit=500))
  return (a$aic)
}

# Reduce the parsimony of a data set for predicting the response variable 'y'.
# 'x' is either a data.frame or matrix with N samples and R attributes.
# 'predacc' is a function with inputs 'y' and a subset of 'x' which returns
# a predictor error criterion value. if 'crit' is "min", then we minimize
# the PAC; if 'crit' is "max", then we maximize the PAC. Return a vector of
# column names corresponding to the new parsimony for 'y'.
prsm <- function(y, x, k=0.01, predacc=ar2, crit="max", printdel=F)
{
  if (is.matrix(x))
  {
    x <- data.frame(x)
  }
  orig_cols <- colnames(x)
  cols <- orig_cols
  pac <- predacc(y, x)
  if (printdel)
  {
    cat("full␣outcome␣=␣", pac, "\n")
  }

  for (col in orig_cols)
  {
    new_cols <- setdiff(cols, col)
    new_pac <- predacc(y, subset(x, select=new_cols))
    if (crit == "max" & (new_pac >= pac | new_pac >= (1-k)*pac)) # ar2() case
    {
      cols <- new_cols
      pac <- new_pac
      if (printdel)
      {
        cat("deleted␣␣␣␣␣␣␣␣␣", col, "\n")
        cat("new␣outcome␣␣␣=␣", pac, "\n")
      }
    }
    else if ( crit == "min" & (new_pac <= pac | new_pac <= (1+k)*pac )) # aiclogit() case
    {
      cols <- new_cols
      pac <- new_pac
      if (printdel)
      {
        cat("deleted␣␣␣␣␣␣␣␣␣", col, "\n")
        cat("new␣outcome␣␣␣=␣", pac, "\n")
      }
    }
  }
```

```
  return (cols)
}
```

Our implementation of an exhaustive parsimony minimizer (`prsmpwr()`).

```
# Reduce parsimony, exhaustively trying all combinations of attributes. To do this,
# we maximize/minimize the PAC over all subsets of the columns, in order of the
# size of the subsets.
prsmpwr <- function(y, x, k=0.01, predacc=ar2, crit="max", printdel=F)
{
  if (is.matrix(x))
  {
    x <- data.frame(x)
  }

  cols <- colnames(x)
  pac <- predacc(y, x)

  if (printdel)
  {
    cat("full outcome = ", pac, "\n")
  }

  for (new_cols in powerset(cols))
  {
    new_pac <- predacc(y, subset(x, select=new_cols))
    if (crit == "max" & (new_pac >= pac | new_pac >= (1-k)*pac)) # ar2() case
    {
      cols <- new_cols
      if (new_pac >= pac)
      {
        pac <- new_pac
      }
      if (printdel)
      {
        cat("new outcome  = ", new_pac, "\n")
      }
    }
    else if (crit == "min" & (new_pac <= pac | new_pac <= (1+k)*pac)) # aiclogit() case
    {
      cols <- new_cols
      if (new_pac <= pac)
      {
        pac <- new_pac
      }
      if (printdel)
      {
        cat("new outcome  = ", new_pac, "\n")
      }
    }
  }
  return (cols)
}

# Enumerate all k-length subsets of S. Called by powerset().
kset <- function(S, k, i, current, e)
{
  if (length(current) == k)
  {
```

```
    e$sets <- append(e$sets, list(c(current)))
    return()
  }

  if (i == length(S) + 1)
  {
    return()
  }

  current <- append(current, S[i])
  kset(S, k, i+1, current, e)

  current <- current[!current==S[i]]
  kset(S, k, i+1, current, e)
}

# Generate the power set of S, ordered by decreasing
# subset size.
powerset <- function(S)
{
  e <- new.env()
  e$sets <- list(c(S))
  for (k in (length(S)-1) : 2)
  {
    kset(S, k, 1, c(), e)
  }
  for (s in S)
  {
    e$sets <- append(e$sets, c(s))
  }
  return (e$sets)
}
```

Implementation of **leave1out01()**

```
# A 'leave one out' pac for a categorizing system with Y = 0 or 1
leave1out01 <- function(Y, X)
{

        n <- length(Y)

        # perform the prediction! res[i] = 1 --> correct prediction, 0 otherwise
        res <- sapply(1:n, function(x) (leave1outHelper(Y, X, x)))

        # sum(res) --> # of correct predictions
        return(sum(res) / n)

}


leave1outHelper <- function(Y, X, i)
{

        X1 <- as.matrix(X)

        y <- NULL
        x <- NULL

        n <- length(Y)
```

```
25
26
27          # Remove i for the predictor
28          if(i > 1){
29                  y <- Y[1:(i-1)]
30                  x <- X1[1:(i-1),]
31          }
32          if(i < n){
33                  #print(i)
34                  #print(n)
35                  #print(dim(X))
36                  y <- c(y, Y[(i+1):n])
37                  x <- rbind(x, X1[(i+1):n,])
38          }
39
40          # Fit Y & X without i
41          fit <- glm(y ~ x, family = binomial)
42
43          # Calculate the prediction
44          left <- c(1, X1[i,]) # Add a value for the intercept to multiply against
45          gt <- Y[i]
46
47          predict <- sum(left * fit["coefficients"][[1]])
48
49          # Return 1 if correct, 0 otherwise
50          if(predict > 0.5){
51                  if(gt == 1){
52                          return(1)
53                  }
54                  return(0)
55          }
56
57          if(gt == 0)
58          {
59                  return(1)
60          }
61          return(0)
62  }
```