Week 4 Homework - Modeling

Question 9.1

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function prcomp for PCA. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

Answer:

The regression model based on the Principle components **did achieve a better BIC than the orginal model (670 vs 681), with 6 instead of 15 predictors. The new model predicted a crime rate of 1388.926.** I handled the scale problem by building the linear regression model against the scaled/centered and rotated data tranformed by the PCA model and added the unscaled dependent variable to the PCA output. I then transformed the "new" data into PCA form by using the predict function against the PCA model, and used the outcome as the input for the final prediction. Doing this, the prediction came in the scale of the original dependent variable without having to "Unscale" the original predictor values.

I used the cumulative proportion metric of the PCA summary, along with the plot of variance explanation to land on 5 Principle components as the "most bang for the buck". The BIC stat confirms this.

Ok, so now for the hard part. I need to express the new model in terms of the original factors. The formula as I know it is:

$a_j = \sum_{k=1}^{L} b_k v_{jk}$

- a is the coefficient for the original factor
- b_k is the coefficient for the kth PC in the LM model
- v_{jk} is the rotation value in the PCA model for each factor in each PC.

This will need to be pulled together with For loops (See Supporting Code).

Here is the final list of coefficients (for 5 PC's), which would need to be applied to the scaled data. This table also shows the scale and cener values used on the original data:

Factor	Coefficient	Scale	Center
M	60.79	1.26	13.86
So	37.85	0.48	0.34
Ed	19.95	1.12	10.56
Po1	117.34	2.97	8.50
Po2	111.45	2.80	8.02
LF	76.25	0.04	0.56
M.F	108.13	2.95	98.30
Pop	58.88	38.07	36.62
NW	98.07	10.28	10.11
U1	2.87	0.02	0.10
U2	32.35	0.84	3.40
Wealth	35.93	964.91	5253.83
Ineq	22.10	3.99	19.40
Prob	-34.64	0.02	0.05
Time	27.21	7.09	26.60

Supporting code for Question 9.1

Load the data and the test values.

In [285]:
```
### Load the data and the test values.

crimdata=read.delim("http://www.statsci.org/data/general/uscrime.txt")

### remove the dependent data for PCA analysis

crimdata2 = crimdata[,1:15]
head(crimdata2,2)
new.values = data.frame(
    M = 14.0,
    So = 0,
    Ed = 10.0,
    Po1 = 12.0,
    Po2 = 15.5,
    LF = 0.640,
    M.F = 94.0,
    Pop = 150,
    NW = 1.1,
    U1 = 0.120,
    U2 = 3.6,
    Wealth = 3200,
    Ineq = 20.1,
    Prob = 0.04,
    Time = 39.0)
```

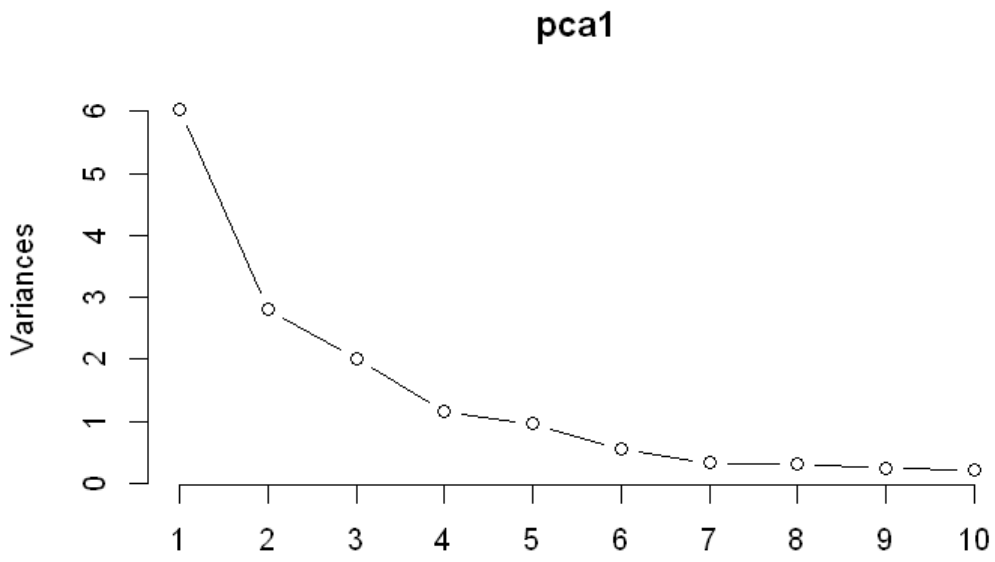
M	So	Ed	Po1	Po2	LF	M.F	Pop	NW	U1	U2	Wealth	Ineq	Prob	Time
15.1	1	9.1	5.8	5.6	0.510	95.0	33	30.1	0.108	4.1	3940	26.1	0.084602	26.2011
14.3	0	11.3	10.3	9.5	0.583	101.2	13	10.2	0.096	3.6	5570	19.4	0.029599	25.2999

Run the PCA model and analyze results

In [286]:
```
### Run the PCA model and plot out the variances
pca1 = prcomp(crimdata2,scale=TRUE)

options(repr.plot.width=6, repr.plot.height=4)
plot(pca1,type="l") ## looks like 5 or 6 Principal components will be ideal.
summary(pca1)
pca1 ## Inferring from the rotation matrix, PC1 is mostly influenced by Wealth and Ineq, while PC2 is M.F and Pop
```

```
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7
Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377 0.56729
Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688 0.02145
Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996 0.92142
                           PC8     PC9    PC10    PC11    PC12    PC13   PC14
Standard deviation     0.55444 0.48493 0.44708 0.41915 0.35804 0.26333 0.2418
Proportion of Variance 0.02049 0.01568 0.01333 0.01171 0.00855 0.00462 0.0039
Cumulative Proportion  0.94191 0.95759 0.97091 0.98263 0.99117 0.99579 0.9997
                          PC15
Standard deviation     0.06793
Proportion of Variance 0.00031
Cumulative Proportion  1.00000

Standard deviations (1, .., p=15):
 [1] 2.45335539 1.67387187 1.41596057 1.07805742 0.97892746 0.74377006
 [7] 0.56729065 0.55443780 0.48492813 0.44708045 0.41914843 0.35803646
[13] 0.26332811 0.24180109 0.06792764

Rotation (n x k) = (15 x 15):
                PC1         PC2           PC3         PC4         PC5
M      -0.30371194  0.06280357  0.1724199946 -0.02035537 -0.35832737
So     -0.33088129 -0.15837219  0.0155433104  0.29247181 -0.12061130
Ed      0.33962148  0.21461152  0.0677396249  0.07974375 -0.02442839
Po1     0.30863412 -0.26981761  0.0506458161  0.33325059 -0.23527680
Po2     0.31099285 -0.26396300  0.0530651173  0.35192809 -0.20473383
LF      0.17617757  0.31943042  0.2715301768 -0.14326529 -0.39407588
M.F     0.11638221  0.39434428 -0.2031621598  0.01048029 -0.57877443
Pop     0.11307836 -0.46723456  0.0770210971 -0.03210513 -0.08317034
NW     -0.29358647 -0.22801119  0.0788156621  0.23925971 -0.36079387
U1      0.04050137  0.00807439 -0.6590290980 -0.18279096 -0.13136873
U2      0.01812228 -0.27971336 -0.5785006293 -0.06889312 -0.13499487
Wealth  0.37970331 -0.07718862  0.0100647664  0.11781752  0.01167683
Ineq   -0.36579778 -0.02752240 -0.0002944563 -0.08066612 -0.21672823
Prob   -0.25888661  0.15831708 -0.1176726436  0.49303389  0.16562829
Time   -0.02062867 -0.38014836  0.2235664632 -0.54059002 -0.14764767
                PC6         PC7         PC8         PC9        PC10        PC11
M      -0.449132706 -0.15707378 -0.55367691  0.15474793 -0.01443093  0.39446657
So     -0.100500743  0.19649727  0.22734157 -0.65599872  0.06141452  0.23397868
Ed     -0.008571367 -0.23943629 -0.14644678 -0.44326978  0.51887452 -0.11821954
Po1    -0.095776709  0.08011735  0.04613156  0.19425472 -0.14320978 -0.13042001
Po2    -0.119524780  0.09518288  0.03168720  0.19512072 -0.05929780 -0.13885912
LF      0.504234275 -0.15931612  0.25513777  0.14393498  0.03077073  0.38532827
M.F    -0.074501901  0.15548197 -0.05507254 -0.24378252 -0.35323357 -0.28029732
Pop     0.547098563  0.09046187 -0.59078221 -0.20244830 -0.03970718  0.05849643
NW      0.051219538 -0.31154195  0.20432828  0.18984178  0.49201966 -0.20695666
U1      0.017385981 -0.17354115 -0.20206312  0.02069349  0.22765278 -0.17857891
U2      0.048155286 -0.07526787  0.24369650  0.05576010 -0.04750100  0.47021842
Wealth -0.154683104 -0.14859424  0.08630649 -0.23196695 -0.11219383  0.31955631
Ineq    0.272027031  0.37483032  0.07184018 -0.02494384 -0.01390576 -0.18278697
Prob    0.283535996 -0.56159383 -0.08598908 -0.05306898 -0.42530006 -0.08978385
Time   -0.148203050 -0.44199877  0.19507812 -0.23551363 -0.29264326 -0.26363121
               PC12        PC13        PC14          PC15
M       0.16580189 -0.05142365  0.04901705  0.0051398012
So     -0.05753357 -0.29368483 -0.29364512  0.0084369230
Ed      0.47786536  0.19441949  0.03964277 -0.0280052040
Po1     0.22611207 -0.18592255 -0.09490151 -0.6894155129
Po2     0.19088461 -0.13454940 -0.08259642  0.7200270100
LF      0.02705134 -0.27742957 -0.15385625  0.0336823193
M.F    -0.23925913  0.31624667 -0.04125321  0.0097922075
Pop    -0.18350385  0.12651689 -0.05326383  0.0001496323
NW     -0.36671707  0.22901695  0.13227774 -0.0370783671
U1     -0.09314897 -0.59039450 -0.02335942  0.0111359325
U2      0.28440496  0.43292853 -0.03985736  0.0073618948
Wealth -0.32172821 -0.14077972  0.70031840 -0.0025685109
Ineq    0.43762828 -0.12181090  0.59279037  0.0177570357
```

```
Prob     0.15567100 -0.03547596  0.04761011  0.0293376260
Time     0.13536989 -0.05738113 -0.04488401  0.0376754405
```



Load test data, predict and test quality of fit

In [287]:
```
### load new values into the principle component tranforms

pca.predvals = predict(pca1,newdata = new.values)

### build a table with the principle components and add the dependedn variable to the mix

pcatable = as.data.frame(pca1$x)
pcatable$Crime = crimdata$Crime

## build a linear regression model against 5 principal components

lm.fit = lm(Crime~PC1+PC2+PC3+PC4+PC5,data=pcatable)

## determine quality of fit against the original model in question 8.2

predict(lm.fit,newdata=as.data.frame(pca.predvals), interval = "prediction") ## new model predicts 1388.93

BIC (lm.fit) ## with PC5, the BIC was 670.65 - BIC on original model was 681
```

	fit	lwr	upr
1	1388.926	861.1796	1916.672

670.654067058524

Find the model as it relates to the original factors

In [288]:

```
L=5 ##Set the number of PC's in the model
factor.count = length(crimdata2) ## Original Factor Count

##Initialize the lists used
coeffs = c()
scl = c()
ctr = c()

## Calculate the sum of b*v for each factor/PC combination used.
## We will also capture the scale and center values for each factor used in the model
for(i in 1:factor.count){
    sum.sub.factors = 0
    for(x in 1:L){
        b=lm.fit$coefficients[x+1]
        v=pca1$rotation[i,x]
        sub.factor = b*v
        sum.sub.factors = sum.sub.factors + sub.factor
        }
    fac.ctr = pca1$center[i]
    fac.scl = pca1$scale[i]
    names(sum.sub.factors) = row.names(pca1$rotation)[i]
    coeffs = round(append(coeffs,sum.sub.factors),2)
    scl = append(scl,fac.scl)
    ctr = append(ctr,fac.ctr)
}

## Build the final table
Coefficient = coeffs
Coeff.Table = (as.data.frame(Coefficient))
Coeff.Table$Scale = round(scl,2)
Coeff.Table$Center = round(ctr,2)
Coeff.Table
```

	Coefficient	Scale	Center
M	60.79	1.26	13.86
So	37.85	0.48	0.34
Ed	19.95	1.12	10.56
Po1	117.34	2.97	8.50
Po2	111.45	2.80	8.02
LF	76.25	0.04	0.56
M.F	108.13	2.95	98.30
Pop	58.88	38.07	36.62
NW	98.07	10.28	10.11
U1	2.87	0.02	0.10
U2	32.35	0.84	3.40
Wealth	35.93	964.91	5253.83
Ineq	22.10	3.99	19.40
Prob	-34.64	0.02	0.05
Time	27.21	7.09	26.60

Question 10.1

Using the same crime data set uscrime.txt 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).

Answer:

Using the model packages for both tree and rpart, I got predictions in the 700-800 range. I don't really trust these models as accurate because of the relatively limited permutations. (each of these models had 7 splits). Using the original linear regression formulas as a benchmark, and the "optimized" PCA model as a comparative target, I can conclude that the tree models are good for "back of the napkin" predictions only, The RandomForest model has promise, in that it returns many results which I can average. In either case, however, the predictive value in a case such as the Crime Study is limited because the dependent variable is not very categorical in nature.

Some basic analysis:

- The 'tree' model only leveraged 4 of the predictors,("Po1" "Pop" "LF" "NW"), which seems oversimplified. Predicted a crime rate of **724.**
- the 'rpart' model assigned the importance of each variable, which I liked (different from the'tree' model, incidentally) and presents the value of **886** as the mean value of the node in which the predicted data landed. I would not interpret this numerically, but rather in T-Shirt Sizes (S,M,L,XL) in terms of crime rate, where the model placec it at a "Large".
- the "RandomForest" has the benefit of providing a range of predictions over 500 trees, and more or less supports the t-Shirt sized predictions of the 'rpart' model, predicting a **919. with seed = 1**
- In general, the tree models predicted lower crime rates for the new data than the linear regression models. Based on the complex relationships involved, I would tend to trust the regression models a bit more in this case.

Supporting code for Question 10.1

Load the Libraries

In [289]:
```
### Load the Libraries
library(tree)
library(randomForest)
library(rpart)
```

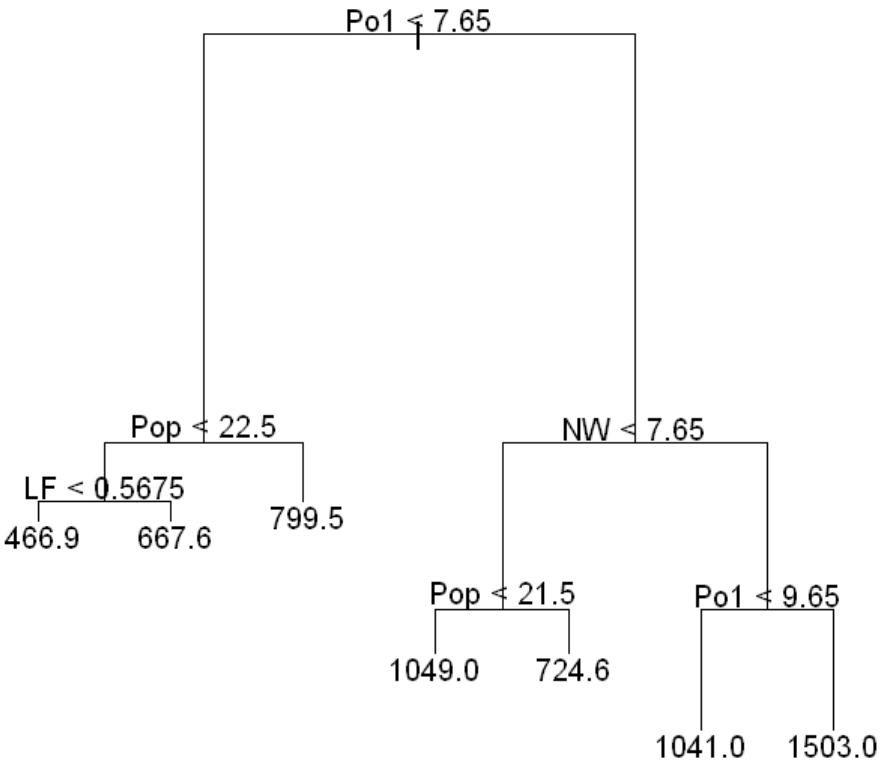
Load and analyze the 'tree' model

In [290]:
```
### Load and analyze the 'tree' model
tree.fit=tree(Crime~.,data=crimdata)
```

In [291]:
```
summary(tree.fit)
predict(tree.fit,newdata=new.values)
options(repr.plot.width=6, repr.plot.height=6)
plot(tree.fit)
text(tree.fit)
```

```
Regression tree:
tree(formula = Crime ~ ., data = crimdata)
Variables actually used in tree construction:
[1] "Po1" "Pop" "LF"  "NW"
Number of terminal nodes:  7
Residual mean deviance:  47390 = 1896000 / 40
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-573.900  -98.300   -1.545    0.000  110.600  490.100
```

1: 724.6



Load and analyze the 'rpart' model

In [292]:
```
### Load and analyze the 'rpart' model
rpt.tree = rpart(Crime~.,data=crimdata)
predict(rpt.tree,newdata = new.values)
```

1: 886.9

In [293]:
```
summary(rpt.tree)
```

```
Call:
rpart(formula = Crime ~ ., data = crimdata)
  n= 47

          CP nsplit rel error   xerror      xstd
1 0.36296293      0 1.0000000 1.089241 0.2782844
2 0.14814320      1 0.6370371 1.217533 0.2988880
3 0.05173165      2 0.4888939 1.051039 0.2309394
4 0.01000000      3 0.4371622 1.016544 0.2311003

Variable importance
   Po1    Po2 Wealth   Ineq   Prob      M     NW    Pop   Time     Ed     LF
    17     17     11     11     10     10      9      5      4      4      1
    So
     1

Node number 1: 47 observations,    complexity param=0.3629629
  mean=905.0851, MSE=146402.7
  left son=2 (23 obs) right son=3 (24 obs)
  Primary splits:
      Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
      Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
      Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
      NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
      Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
  Surrogate splits:
      Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
      Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
      Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
      M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
      Ineq   < 17.15     to the right, agree=0.745, adj=0.478, (0 split)

Node number 2: 23 observations,    complexity param=0.05173165
  mean=669.6087, MSE=33880.15
  left son=4 (12 obs) right son=5 (11 obs)
  Primary splits:
      Pop < 22.5      to the left,  improve=0.4568043, (0 missing)
      M   < 14.5      to the left,  improve=0.3931567, (0 missing)
      NW  < 5.4       to the left,  improve=0.3184074, (0 missing)
      Po1 < 5.75      to the left,  improve=0.2310098, (0 missing)
      U1  < 0.093     to the right, improve=0.2119062, (0 missing)
  Surrogate splits:
      NW   < 5.4       to the left,  agree=0.826, adj=0.636, (0 split)
      M    < 14.5      to the left,  agree=0.783, adj=0.545, (0 split)
      Time < 22.30055  to the left,  agree=0.783, adj=0.545, (0 split)
      So   < 0.5       to the left,  agree=0.739, adj=0.455, (0 split)
      Ed   < 10.85     to the right, agree=0.739, adj=0.455, (0 split)

Node number 3: 24 observations,    complexity param=0.1481432
  mean=1130.75, MSE=150173.4
  left son=6 (10 obs) right son=7 (14 obs)
  Primary splits:
      NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
      M    < 13.05     to the left,  improve=0.2714159, (0 missing)
      Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
      M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
      Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
  Surrogate splits:
      Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
      Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
      Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
      Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
      LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)

Node number 4: 12 observations
  mean=550.5, MSE=20317.58

Node number 5: 11 observations
  mean=799.5455, MSE=16315.52

Node number 6: 10 observations
  mean=886.9, MSE=55757.49
```

```
Node number 7: 14 observations
  mean=1304.929, MSE=144801.8
```

Load and Analyze the 'randomForest' model

In [294]:
```
### Load and Analyze the 'randomForest' model
set.seed(1)
rf.fit = randomForest(Crime~.,crimdata)
```

In [295]:
```
round(mean(predict(rf.fit,data=new.values)),2)
```

919.46

In [296]:
```
rf.fit
```

```
Call:
 randomForest(formula = Crime ~ ., data = crimdata)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 5

          Mean of squared residuals: 87461.6
                    % Var explained: 40.26
```

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.

Answer:

Logistic regression can be used to determine probability of malignancy in tumors, or, frankly, a wide variety of medical diagnostics based on the presence of appropriate symptoms and lifestyle descriptors. Based on the mix of predictors, we can determine the probability of 1= Having the disease, or 0 = Not having the disease. Models will be tuned based on the cost of errors (type 1 = False Positive, type 2 = False Negative)

Question 10.3.1

Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german (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 (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.

Answer:

In using Logistic Regression, R "dummifies" factor variables to assign a coeffecient to each factor value (each value affects the prediction by a diffeernt amount.) Each coefficient is shown in the table below. The software output is shown below in the supporting code, along with a residual deviance table that shows the reduction of deviance by each variable.

to measure quality of fit I generated an ROC Curve and measured the **AUC at 80.14** which means that there is a 80% probability that a Bad credit risk will be classified as such. I'm sure this can be improved, but I was not able to do so by removing variables, etc.

Table of Coefficients

Factor	Coefficient	Factor	Coefficient
(Intercept)	0.949934729	Employment_StatusA74	-0.744732682
CA_StatusA12	-0.320493451	Employment_StatusA75	-0.157826685
CA_StatusA13	-0.69800201	Installment_Rate	0.360662161
CA_StatusA14	-1.827148361	Personal_StatA92	0.056027162
Duration	0.030042334	Personal_StatA93	-0.598615373
CredhistA31	0.257881222	Personal_StatA94	-0.059919403
CredhistA32	-0.724622688	Other_DebtorsA102	0.517136813
CredhistA33	-1.0770272	Other_DebtorsA103	-0.955649205
CredhistA34	-1.46173135	Resident_Since	-0.106880417
PurposeA41	-1.651355266	PropertyA122	0.13380115
PurposeA410	-1.622238078	PropertyA123	0.172541627
PurposeA42	-0.787375493	PropertyA124	0.759989067
PurposeA43	-0.875286827	Age	-0.017165993
PurposeA44	0.069254882	Other_PlansA142	-0.183536185
PurposeA45	-0.225440894	Other_PlansA143	-0.713785947
PurposeA46	0.419390122	HousingA152	-0.633933234
PurposeA48	-15.87760788	HousingA153	-0.925625727
PurposeA49	-1.007027702	Number_Credit	0.230421215
Credit_Amt	0.00013526	JobA172	0.257028976
SavingsA62	-0.326714166	JobA173	0.085735164
SavingsA63	-0.188339383	JobA174	-0.004784806
SavingsA64	-1.622852339	People	0.389251047
SavingsA65	-0.817431419	TelephoneA192	-0.170295864
Employment_StatusA72	-0.117363807	ForeignA202	-1.389343488
Employment_StatusA73	-0.109279881		

Supporting code for Question 10.3.1

Load the data and take a look

In [373]:
```
### Load the data and take a look
germcred = read.table("https://prod-edxapp.edx-cdn.org/assets/courseware/v1/a145a478beb6f64b59ec1de082b84235/asse
head(germcred,2)
dim(germcred)
germcred$V1=as.factor(germcred$V1)
germcred$V3=as.factor(germcred$V3)
germcred$V4=as.factor(germcred$V4)
germcred$V6=as.factor(germcred$V6)
germcred$V7=as.factor(germcred$V7)
germcred$V9=as.factor(germcred$V9)
germcred$V10=as.factor(germcred$V10)
germcred$V12=as.factor(germcred$V12)
germcred$V14=as.factor(germcred$V14)
germcred$V15=as.factor(germcred$V15)
germcred$V17=as.factor(germcred$V17)
germcred$V19=as.factor(germcred$V19)
germcred$V20=as.factor(germcred$V20)
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
A11	6	A34	A43	1169	A65	A75	4	A93	A101	...	A121	67	A143	A152	2	A173	1	A192	A201	1
A12	48	A32	A43	5951	A61	A73	2	A92	A101	...	A121	22	A143	A152	1	A173	1	A191	A201	2

1000 21

Add Column names for Context, tranform the dependent variable and split into training and test data

In [374]:
```
### Add Column names for Context
GCColumns = c("CA_Status","Duration","Credhist","Purpose","Credit_Amt","Savings","Employment_Status","Installment
colnames(germcred) = GCColumns
```

In [375]:
```
### transform response variable to 1-0
germcred$GB10 = germcred[,21]-1
```

In [376]:
```
set.seed(1)
testindx = sample(1:nrow(germcred),round(.2*nrow(germcred)))
testdata = germcred[testindx,]

## take every other row (remaining 80%) and call it traindata.

traindata = germcred[-testindx,]
```

Run and analyze the model

In [377]:
```
lr=glm(GB10~.-GB,data=traindata,family=binomial(link='logit'))
```

In [378]:
```
summary(lr)
```

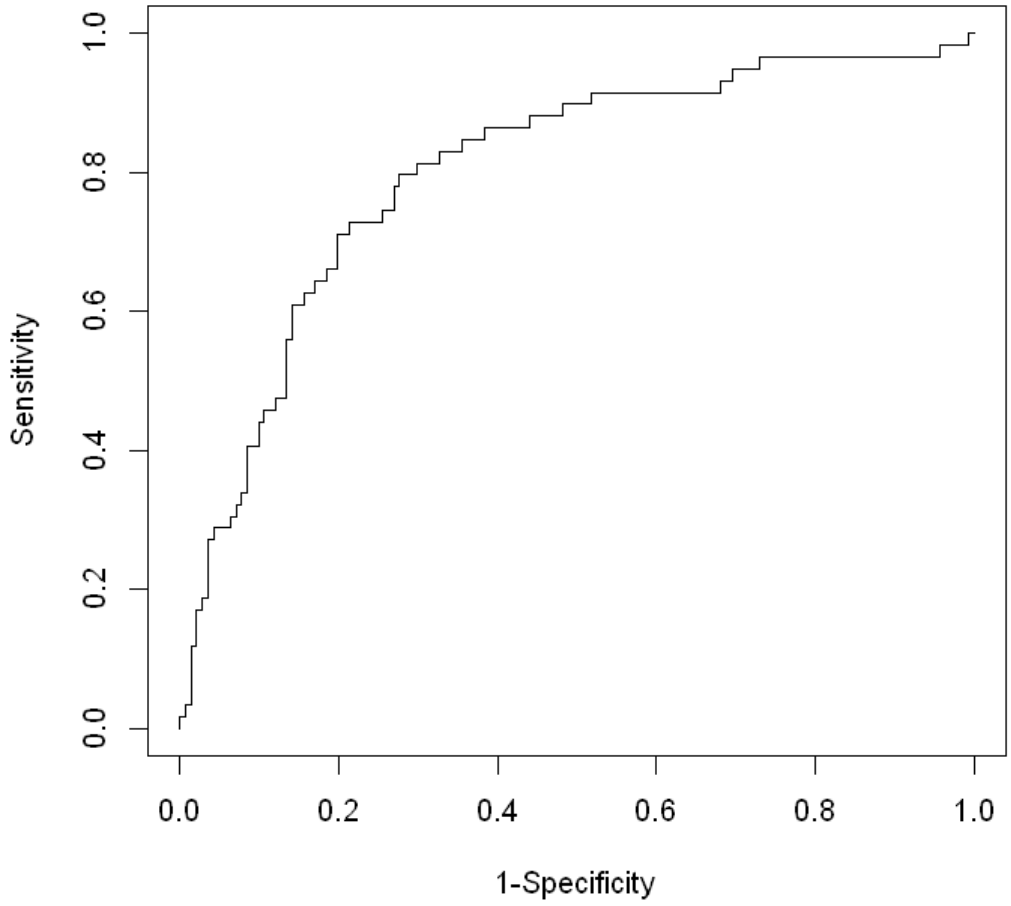
```
Call:
glm(formula = GB10 ~ . - GB, family = binomial(link = "logit"), 
    data = traindata)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.4842  -0.7090  -0.3503   0.7049   2.6465  

Coefficients:
                       Estimate Std. Error z value Pr(>|z|)    
(Intercept)           9.499e-01  1.234e+00   0.770 0.441432    
CA_StatusA12         -3.205e-01  2.463e-01  -1.301 0.193246    
CA_StatusA13         -6.980e-01  3.926e-01  -1.778 0.075389 .  
CA_StatusA14         -1.827e+00  2.655e-01  -6.882 5.89e-12 ***
Duration              3.004e-02  1.050e-02   2.860 0.004237 ** 
CredhistA31           2.579e-01  6.277e-01   0.411 0.681213    
CredhistA32          -7.246e-01  5.009e-01  -1.447 0.148025    
CredhistA33          -1.077e+00  5.535e-01  -1.946 0.051676 .  
CredhistA34          -1.462e+00  5.069e-01  -2.884 0.003928 ** 
PurposeA41           -1.651e+00  4.108e-01  -4.020 5.83e-05 ***
PurposeA410          -1.622e+00  7.969e-01  -2.036 0.041780 *  
PurposeA42           -7.874e-01  2.961e-01  -2.659 0.007833 ** 
PurposeA43           -8.753e-01  2.808e-01  -3.117 0.001826 ** 
PurposeA44            6.925e-02  8.276e-01   0.084 0.933311    
PurposeA45           -2.254e-01  6.244e-01  -0.361 0.718075    
PurposeA46            4.194e-01  4.571e-01   0.918 0.358852    
PurposeA48           -1.588e+01  4.509e+02  -0.035 0.971908    
PurposeA49           -1.007e+00  3.850e-01  -2.616 0.008900 ** 
Credit_Amt            1.353e-04  4.949e-05   2.733 0.006274 ** 
SavingsA62           -3.267e-01  3.195e-01  -1.023 0.306476    
SavingsA63           -1.883e-01  4.313e-01  -0.437 0.662357    
SavingsA64           -1.623e+00  6.185e-01  -2.624 0.008699 ** 
SavingsA65           -8.174e-01  2.944e-01  -2.776 0.005496 ** 
Employment_StatusA72 -1.174e-01  4.804e-01  -0.244 0.806993    
Employment_StatusA73 -1.093e-01  4.616e-01  -0.237 0.812874    
Employment_StatusA74 -7.447e-01  5.068e-01  -1.469 0.141719    
Employment_StatusA75 -1.578e-01  4.606e-01  -0.343 0.731861    
Installment_Rate      3.607e-01  9.925e-02   3.634 0.000279 ***
Personal_StatA92      5.603e-02  4.542e-01   0.123 0.901832    
Personal_StatA93     -5.986e-01  4.491e-01  -1.333 0.182542    
Personal_StatA94     -5.992e-02  5.226e-01  -0.115 0.908720    
Other_DebtorsA102     5.171e-01  4.695e-01   1.101 0.270688    
Other_DebtorsA103    -9.556e-01  4.702e-01  -2.032 0.042122 *  
Resident_Since       -1.069e-01  9.839e-02  -1.086 0.277373    
PropertyA122          1.338e-01  2.890e-01   0.463 0.643351    
PropertyA123          1.725e-01  2.648e-01   0.652 0.514617    
PropertyA124          7.600e-01  4.970e-01   1.529 0.126195    
Age                  -1.717e-02  1.047e-02  -1.640 0.101089    
Other_PlansA142      -1.835e-01  4.613e-01  -0.398 0.690714    
Other_PlansA143      -7.138e-01  2.654e-01  -2.690 0.007146 ** 
HousingA152          -6.339e-01  2.676e-01  -2.369 0.017821 *  
HousingA153          -9.256e-01  5.509e-01  -1.680 0.092897 .  
Number_Credit         2.304e-01  2.095e-01   1.100 0.271448    
JobA172               2.570e-01  7.665e-01   0.335 0.737364    
JobA173               8.574e-02  7.414e-01   0.116 0.907934    
JobA174              -4.785e-03  7.349e-01  -0.007 0.994805    
People                3.893e-01  2.814e-01   1.384 0.166510    
TelephoneA192        -1.703e-01  2.260e-01  -0.753 0.451158    
ForeignA202          -1.389e+00  6.496e-01  -2.139 0.032445 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 979.07  on 799  degrees of freedom
Residual deviance: 710.34  on 751  degrees of freedom
AIC: 808.34

Number of Fisher Scoring iterations: 14
```

In [379]:
```
anova(lr)
```

	Df	Deviance	Resid. Df	Resid. Dev
NULL	NA	NA	799	979.0715
CA_Status	3	104.9458649	796	874.1257
Duration	1	26.9902213	795	847.1354
Credhist	4	21.0410568	791	826.0944
Purpose	9	35.3749033	782	790.7195
Credit_Amt	1	1.8102328	781	788.9093
Savings	4	15.0224437	777	773.8868
Employment_Status	4	9.2471596	773	764.6396
Installment_Rate	1	9.1494924	772	755.4902
Personal_Stat	3	8.8054304	769	746.6847
Other_Debtors	2	6.8667835	767	739.8179
Resident_Since	1	0.3490832	766	739.4689
Property	3	2.0304983	763	737.4384
Age	1	3.1096614	762	734.3287
Other_Plans	2	8.0309229	760	726.2978
Housing	2	5.5000725	758	720.7977
Number_Credit	1	1.5093016	757	719.2884
Job	3	1.1336201	754	718.1548
People	1	1.9222617	753	716.2325
Telephone	1	0.4042711	752	715.8282
Foreign	1	5.4884301	751	710.3398

Plot and measure quality of fit

In [380]:
```
###Plot and measure quality of fit
library(ROCR)
p <- predict(lr, newdata=testdata, type="response")
pr <- prediction(p, testdata$GB10)
prf <- performance(pr, measure = "sens", x.measure = "fpr")
plot(prf, xlab="1-Specificity") ## 1-Specificity = FPR.  Changing label for consistency
```



In [381]:
```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
round(auc*100,2)
```

80.14

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.

Answer:

The best threshold probability for my model, in other words, the threshold probability that give me the best fit to the predictions in my test data set, is **47%, which give me a fit to the test data at 78.5% accuracy**. This was determined by testing the accuracy of the predictions against the test data at various thresholds and graphing the results below. I would assume that this number would hover around 50% because the dataset is already costed at 5:1 and the model takes that into account.

Supporting code for Question 10.3.2

Determine the best threshold for accuracy against the "costed" data set.

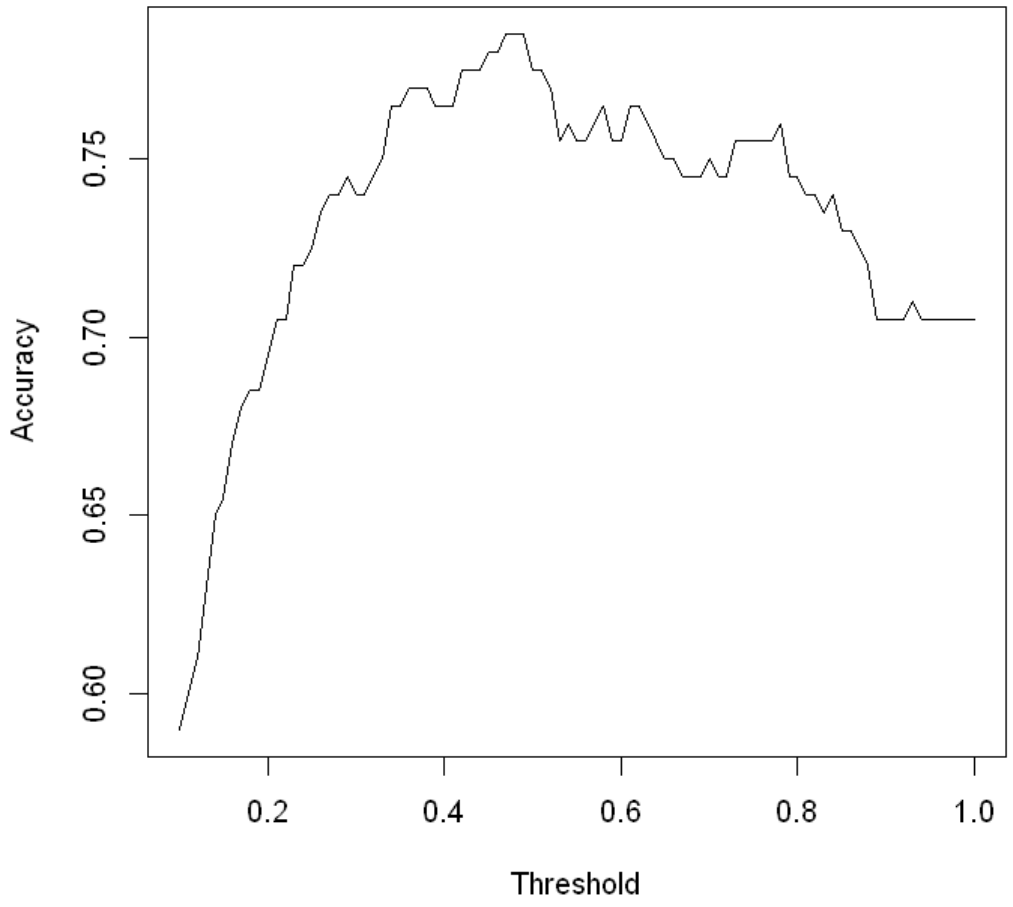
The predictions in original data set have already been "costed", and the model had to have taken that into account using the training data. So the task now is to determine the best fit of probability of this model against the origianl (costed) data.

In [382]:
```
Accuracy = c()
Threshold=seq(.1, 1, by=.01 )
for (i in Threshold){
    fitted.results <- predict(lr,newdata=testdata,type='response')
    fitted.results <- ifelse(fitted.results > i,1,0)
    misClasificError <- mean(fitted.results != testdata$GB10)
    Accuracy = append(Accuracy,1-misClasificError)
    print(paste('Accuracyfor Threshold',i,"is",1-misClasificError))}
```

```
[1] "Accuracyfor Threshold 0.1 is 0.59"
[1] "Accuracyfor Threshold 0.11 is 0.6"
[1] "Accuracyfor Threshold 0.12 is 0.61"
[1] "Accuracyfor Threshold 0.13 is 0.63"
[1] "Accuracyfor Threshold 0.14 is 0.65"
[1] "Accuracyfor Threshold 0.15 is 0.655"
[1] "Accuracyfor Threshold 0.16 is 0.67"
[1] "Accuracyfor Threshold 0.17 is 0.68"
[1] "Accuracyfor Threshold 0.18 is 0.685"
[1] "Accuracyfor Threshold 0.19 is 0.685"
[1] "Accuracyfor Threshold 0.2 is 0.695"
[1] "Accuracyfor Threshold 0.21 is 0.705"
[1] "Accuracyfor Threshold 0.22 is 0.705"
[1] "Accuracyfor Threshold 0.23 is 0.72"
[1] "Accuracyfor Threshold 0.24 is 0.72"
[1] "Accuracyfor Threshold 0.25 is 0.725"
[1] "Accuracyfor Threshold 0.26 is 0.735"
[1] "Accuracyfor Threshold 0.27 is 0.74"
[1] "Accuracyfor Threshold 0.28 is 0.74"
[1] "Accuracyfor Threshold 0.29 is 0.745"
[1] "Accuracyfor Threshold 0.3 is 0.74"
[1] "Accuracyfor Threshold 0.31 is 0.74"
[1] "Accuracyfor Threshold 0.32 is 0.745"
[1] "Accuracyfor Threshold 0.33 is 0.75"
[1] "Accuracyfor Threshold 0.34 is 0.765"
[1] "Accuracyfor Threshold 0.35 is 0.765"
[1] "Accuracyfor Threshold 0.36 is 0.77"
[1] "Accuracyfor Threshold 0.37 is 0.77"
[1] "Accuracyfor Threshold 0.38 is 0.77"
[1] "Accuracyfor Threshold 0.39 is 0.765"
[1] "Accuracyfor Threshold 0.4 is 0.765"
[1] "Accuracyfor Threshold 0.41 is 0.765"
[1] "Accuracyfor Threshold 0.42 is 0.775"
[1] "Accuracyfor Threshold 0.43 is 0.775"
[1] "Accuracyfor Threshold 0.44 is 0.775"
[1] "Accuracyfor Threshold 0.45 is 0.78"
[1] "Accuracyfor Threshold 0.46 is 0.78"
[1] "Accuracyfor Threshold 0.47 is 0.785"
[1] "Accuracyfor Threshold 0.48 is 0.785"
[1] "Accuracyfor Threshold 0.49 is 0.785"
[1] "Accuracyfor Threshold 0.5 is 0.775"
[1] "Accuracyfor Threshold 0.51 is 0.775"
[1] "Accuracyfor Threshold 0.52 is 0.77"
[1] "Accuracyfor Threshold 0.53 is 0.755"
[1] "Accuracyfor Threshold 0.54 is 0.76"
[1] "Accuracyfor Threshold 0.55 is 0.755"
[1] "Accuracyfor Threshold 0.56 is 0.755"
[1] "Accuracyfor Threshold 0.57 is 0.76"
[1] "Accuracyfor Threshold 0.58 is 0.765"
[1] "Accuracyfor Threshold 0.59 is 0.755"
[1] "Accuracyfor Threshold 0.6 is 0.755"
[1] "Accuracyfor Threshold 0.61 is 0.765"
[1] "Accuracyfor Threshold 0.62 is 0.765"
[1] "Accuracyfor Threshold 0.63 is 0.76"
[1] "Accuracyfor Threshold 0.64 is 0.755"
[1] "Accuracyfor Threshold 0.65 is 0.75"
[1] "Accuracyfor Threshold 0.66 is 0.75"
[1] "Accuracyfor Threshold 0.67 is 0.745"
[1] "Accuracyfor Threshold 0.68 is 0.745"
[1] "Accuracyfor Threshold 0.69 is 0.745"
[1] "Accuracyfor Threshold 0.7 is 0.75"
[1] "Accuracyfor Threshold 0.71 is 0.745"
[1] "Accuracyfor Threshold 0.72 is 0.745"
[1] "Accuracyfor Threshold 0.73 is 0.755"
[1] "Accuracyfor Threshold 0.74 is 0.755"
[1] "Accuracyfor Threshold 0.75 is 0.755"
```

```
[1] "Accuracyfor Threshold 0.76 is 0.755"
[1] "Accuracyfor Threshold 0.77 is 0.755"
[1] "Accuracyfor Threshold 0.78 is 0.76"
[1] "Accuracyfor Threshold 0.79 is 0.745"
[1] "Accuracyfor Threshold 0.8 is 0.745"
[1] "Accuracyfor Threshold 0.81 is 0.74"
[1] "Accuracyfor Threshold 0.82 is 0.74"
[1] "Accuracyfor Threshold 0.83 is 0.735"
[1] "Accuracyfor Threshold 0.84 is 0.74"
[1] "Accuracyfor Threshold 0.85 is 0.73"
[1] "Accuracyfor Threshold 0.86 is 0.73"
[1] "Accuracyfor Threshold 0.87 is 0.725"
[1] "Accuracyfor Threshold 0.88 is 0.72"
[1] "Accuracyfor Threshold 0.89 is 0.705"
[1] "Accuracyfor Threshold 0.9 is 0.705"
[1] "Accuracyfor Threshold 0.91 is 0.705"
[1] "Accuracyfor Threshold 0.92 is 0.705"
[1] "Accuracyfor Threshold 0.93 is 0.71"
[1] "Accuracyfor Threshold 0.94 is 0.705"
[1] "Accuracyfor Threshold 0.95 is 0.705"
[1] "Accuracyfor Threshold 0.96 is 0.705"
[1] "Accuracyfor Threshold 0.97 is 0.705"
[1] "Accuracyfor Threshold 0.98 is 0.705"
[1] "Accuracyfor Threshold 0.99 is 0.705"
[1] "Accuracyfor Threshold 1 is 0.705"
```

In [383]:
```
plot(Threshold,Accuracy,type="l")
```



In [ ]:
```

```