Predicting blood donations

Maximilian Press

I decided to do some basic analysis of the blood donation dataset with predictions to see how good I could get using some basic tools.

Logistic regression

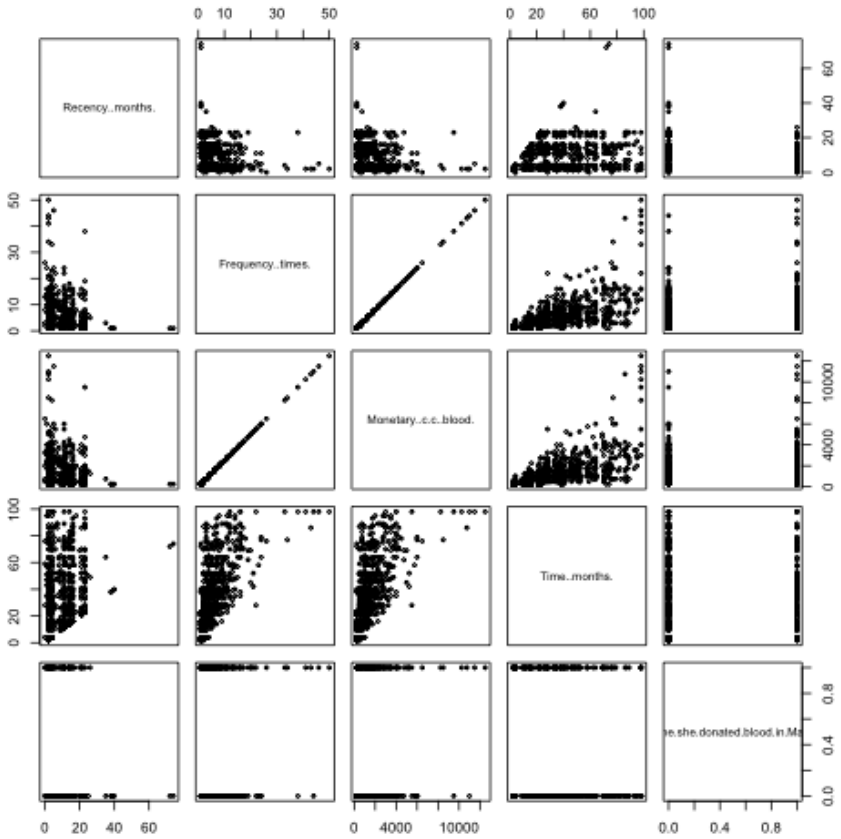
This was fairly simple. I chose to randomly sample 500 observations to train, and test on the remaining 248.

```
trans = read.csv('transfusion.data',header=T)
cor(trans)
```

```
##                                             Recency..months.
## Recency..months.                                   1.0000000
## Frequency..times.                                 -0.1827455
## Monetary..c.c..blood.                             -0.1827455
## Time..months.                                      0.1606181
## whether.he.she.donated.blood.in.March.2007        -0.2798689
##                                             Frequency..times.
## Recency..months.                                  -0.1827455
## Frequency..times.                                  1.0000000
## Monetary..c.c..blood.                              1.0000000
## Time..months.                                      0.6349403
## whether.he.she.donated.blood.in.March.2007         0.2186334
##                                             Monetary..c.c..blood.
## Recency..months.                                       -0.1827455
## Frequency..times.                                       1.0000000
## Monetary..c.c..blood.                                   1.0000000
## Time..months.                                           0.6349403
## whether.he.she.donated.blood.in.March.2007              0.2186334
##                                             Time..months.
## Recency..months.                               0.16061809
## Frequency..times.                              0.63494027
## Monetary..c.c..blood.                          0.63494027
## Time..months.                                  1.00000000
## whether.he.she.donated.blood.in.March.2007    -0.03585441
##                                             whether.he.she.donated.blood.in.March.2007
## Recency..months.                                                            -0.27986887
## Frequency..times.                                                            0.21863344
## Monetary..c.c..blood.                                                        0.21863344
## Time..months.                                                               -0.03585441
## whether.he.she.donated.blood.in.March.2007                                   1.00000000
```

Look at the data a little (Figure 1).

```
plot(trans,cex=.5)
```



Obviously some of these things are more meaningful than other things. I will sorta naively fit the model based on everything, ignoring the possibility of interactions.

First, fit a linear model, which is ok but not very interesting.

```
plot(train$Frequency..times.,jitter(train$whether.he.she.donated.blood.in.March.2007),xlab="
linmod = lm(whether.he.she.donated.blood.in.March.2007 ~ Frequency..times.,data = train)
summary(linmod)
```

```
##
## Call:
```

```
## lm(formula = whether.he.she.donated.blood.in.March.2007 ~ Frequency..times.,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7743 -0.2429 -0.2021 -0.1053  0.8116
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.174819   0.025734   6.793 3.14e-11 ***
## Frequency..times. 0.013625   0.003136   4.344 1.69e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4259 on 498 degrees of freedom
## Multiple R-squared:  0.03651,    Adjusted R-squared:  0.03458
## F-statistic: 18.87 on 1 and 498 DF,  p-value: 1.694e-05
```

```
abline(linmod)
```

So we had a low p-value, which is good right? Problem solved, everyone go home.

Except this is obviously a really crappy model. This can be shown if we try to predict test values (new data that wasn't used to build the model, just plugging new values into the model function) and compare them to the actual values of the test outcome.

```
linpred = predict(linmod,newdata=test)
linpredplot = plot(jitter(test$whether.he.she.donated.blood.in.March.2007), linpred,
xlab='True value (jittered)', ylab='Predicted value', xlim = c(-.2,1.2), ylim = c(0,1), cex
#abline( a = 0, b = 1 )
points( c(0,1), c(0,1), cex = 2, pch = 19 )
```

```
prediction = cbind(linpred,test[,5])
a = ROC(prediction)
```

```
# training set
trainindex = sample(1:748,500)
train = trans[trainindex,]
# test set
test = trans[!(1:nrow(trans) %in% trainindex),]
```

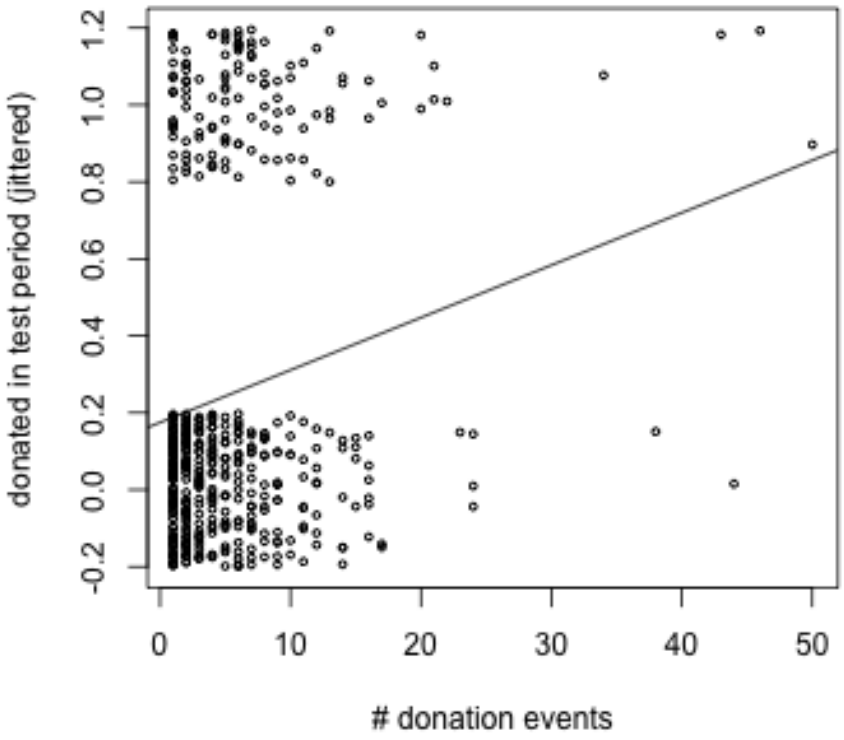


Figure 1: plot of chunk unnamed-chunk-3

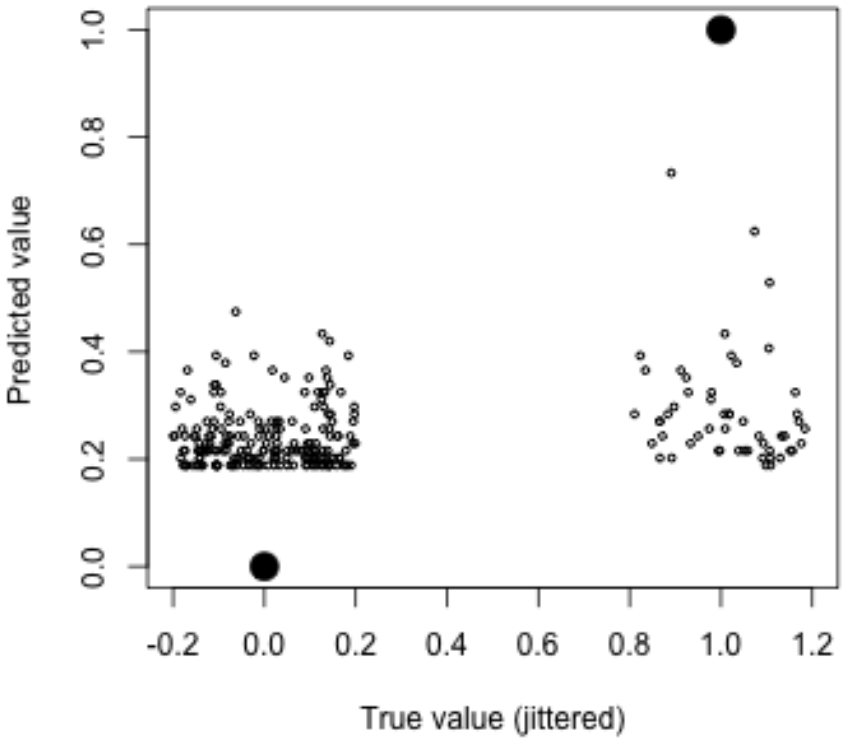


Figure 2: plot of chunk unnamed-chunk-4

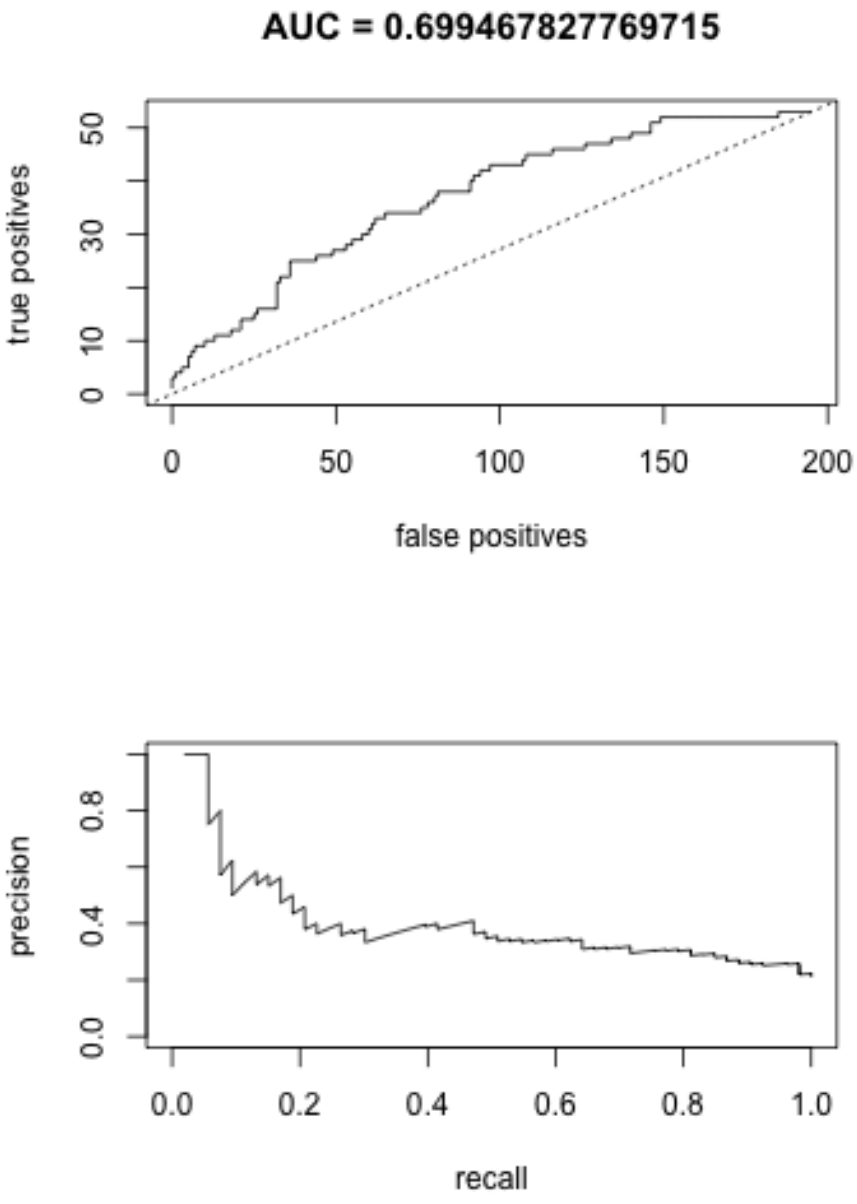
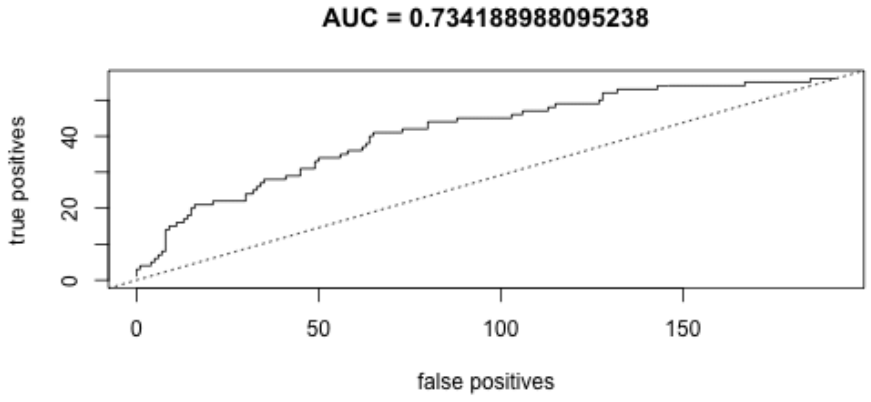


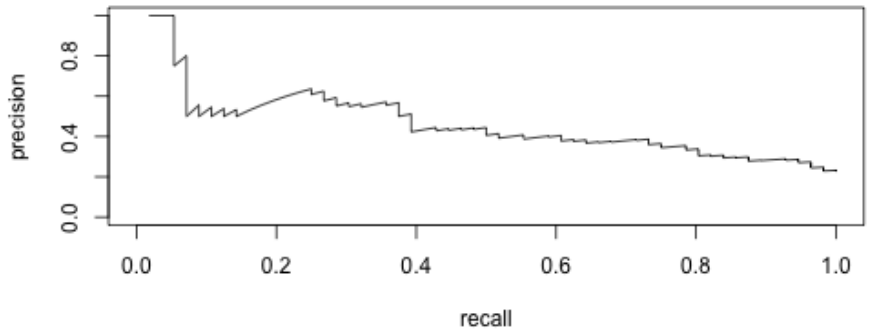
Figure 3: plot of chunk unnamed-chunk-5

```
trainfit = glm(whether.he.she.donated.blood.in.March.2007 ~ Recency..months. + Frequency..ti

# do some predictions
predictor = predict.glm(trainfit,newdata=test)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

prediction = cbind(predictor,test[,5])

# really crude look at prediction success
cor(prediction,method='spearman')

##           predictor
## predictor 1.0000000 0.3391207
##           0.3391207 1.0000000

# some utility functions
source('roc.R')
```

How good is this model anyways? (Figure 2)

```
a=ROC(prediction)
```





So it's not great, but okay (i.e. AUC>0.5). Specifically, the precision goes to hell very quickly. Does stepwise regression help it by getting rid of spurious variables that are overfitting?

```
stepfit = step(trainfit)
```

```
## Start:  AIC=482.84
## whether.he.she.donated.blood.in.March.2007 ~ Recency..months. +
##     Frequency..times. + Monetary..c.c..blood. + Time..months.
##
##
## Step:  AIC=482.84
## whether.he.she.donated.blood.in.March.2007 ~ Recency..months. +
##     Frequency..times. + Time..months.
##
##                     Df Deviance    AIC
```

```
## <none>                    474.84 482.84
## - Time..months.      1    487.12 493.12
## - Frequency..times.  1    499.18 505.18
## - Recency..months.   1    505.56 511.56
```

So it looks like "Monetary" is pretty much colinear with "frequency", so no additional information (removing it seems to leave exactly the same model). Also, recency does not seem to add much, so I am going to remove that.

```
curated_fit = glm(whether.he.she.donated.blood.in.March.2007 ~ Frequency..times. + Time..mor
curated_prediction = predict.glm(curated_fit,newdata=test)

prediction = cbind(curated_prediction,test[,5])

a=ROC(prediction)
```

Didn't really change much (Figure 3). Lost a little AUC, but not much for removing 2 explanatory variables in slavish devotion to occam's razor. Precision seems to fall apart a bit, though. While logistic regression is nice and simple, it is not doing a super job, so I will move on to see if anything else does better.

Naive Bayes

Naive Bayes is an attractively simple classification technique. It is similar to the initial logistic regression implemented above, because of its assumption of independence of predictor variables. It uses a straightforward interpretation of Bayes' rule to compute probabilities of each variable belonging to each class. While we only have a binary outcome, it is possible that NB will perform better for some reason.

```
require(e1071)

## Loading required package: e1071

## Warning: package 'e1071' was built under R version 3.2.2

# this function wants response to be a factor
classifier = naiveBayes(train[,1:4],as.factor(train[,5]))
print(classifier)

##
## Naive Bayes Classifier for Discrete Predictors
```

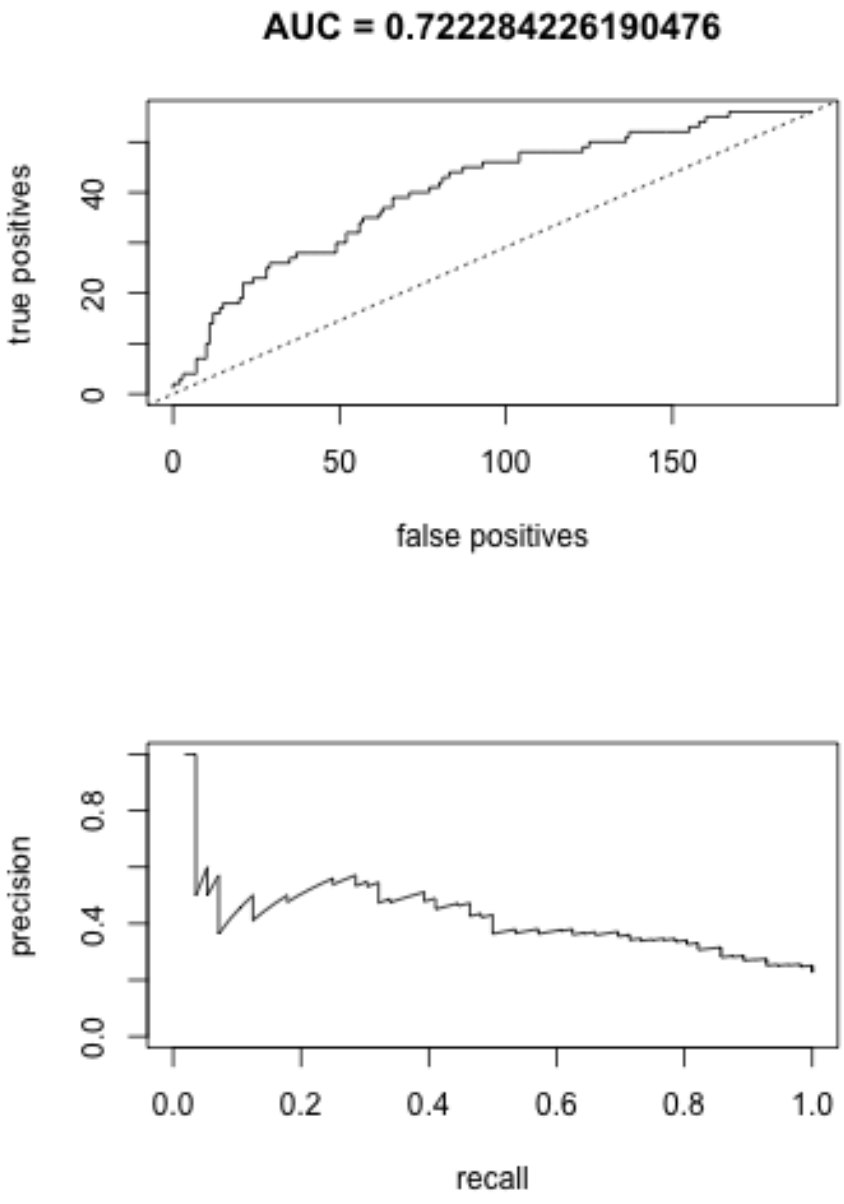
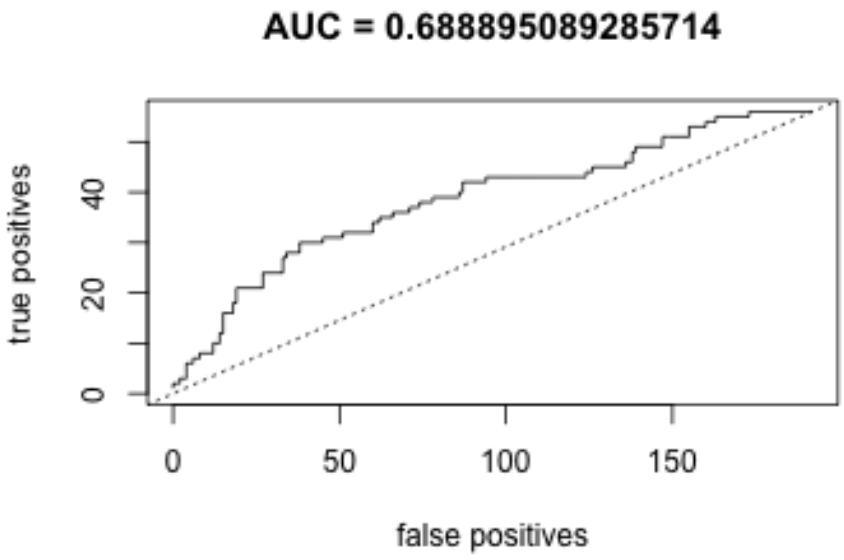


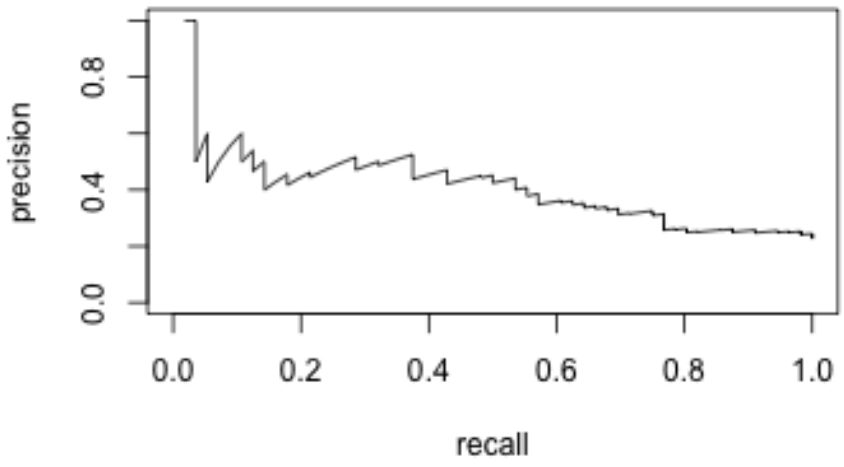
Figure 4: Performance of logistic regression with reduced model

```
##
## Call:
## naiveBayes.default(x = train[, 1:4], y = as.factor(train[, 5]))
##
## A-priori probabilities:
## as.factor(train[, 5])
##     0     1
## 0.756 0.244
##
## Conditional probabilities:
##                       Recency..months.
## as.factor(train[, 5])      [,1]     [,2]
##                     0 10.399471 7.731001
##                     1  5.122951 4.821745
##
##                       Frequency..times.
## as.factor(train[, 5])     [,1]     [,2]
##                     0 4.873016 4.889198
##                     1 7.836066 8.239555
##
##                       Monetary..c.c..blood.
## as.factor(train[, 5])     [,1]     [,2]
##                     0 1218.254 1222.299
##                     1 1959.016 2059.889
##
##                       Time..months.
## as.factor(train[, 5])     [,1]     [,2]
##                     0 35.68783 25.06422
##                     1 33.23770 24.51212
```

```
bayespredict = cbind(predict(classifier,test[,-5]),test[,5])
```

```
a=ROC(bayespredict)
```





Well, it turns out that Bayesian statistics is not the answer to everything (Figure 4). About the same as the reduced logistic regression model. The curve is weirdly step-like, wonder what's going on there. Perhaps because NB is specifying categorical cutoffs in the continuous data?

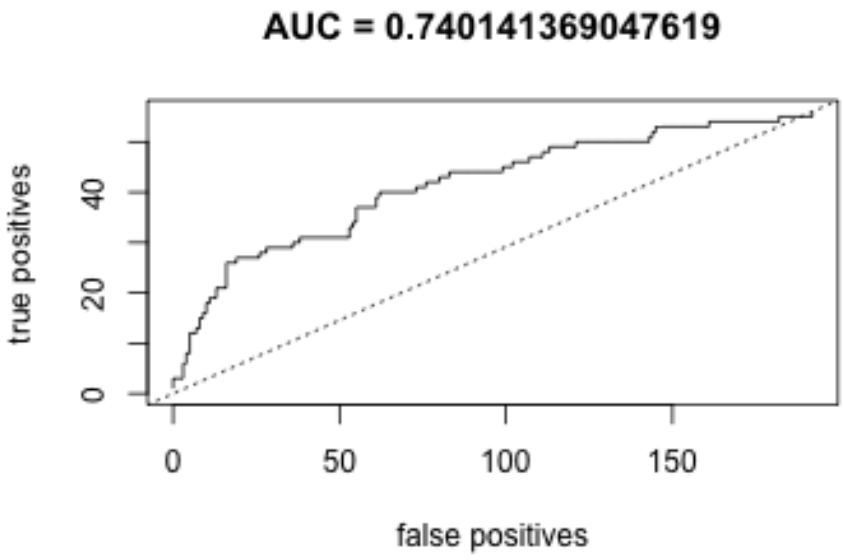
Interaction effects

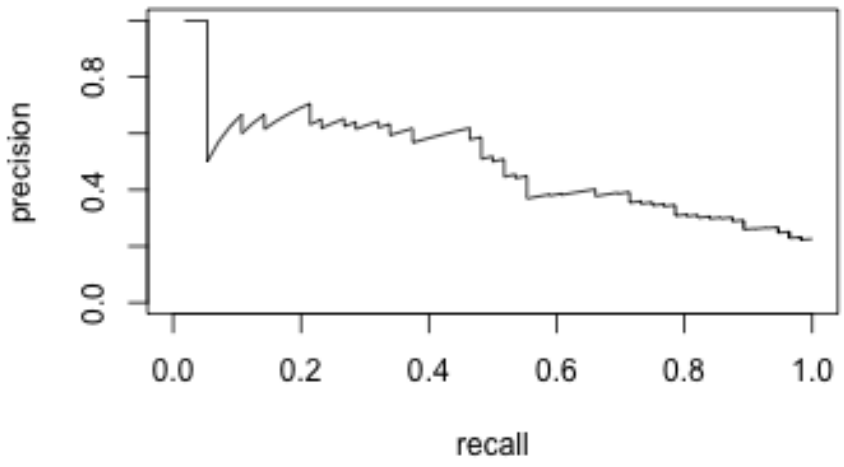
So far I have made the simplifying assumption that the variables are independent. This obviously isn't the case. Maybe what I am missing is interactions between variables, which contain something extra. I will go back to the logistic regression model, except this time add interactions.

```
interfit = glm(whether.he.she.donated.blood.in.March.2007 ~ Recency..months. * Frequency..ti

interstep = step(interfit)

## Start:  AIC=473.88
## whether.he.she.donated.blood.in.March.2007 ~ Recency..months. *
##     Frequency..times. * Time..months.
##
##                                                    Df Deviance    AIC
## <none>                                                  457.88 473.88
## - Recency..months.:Frequency..times.:Time..months.  1   460.40 474.40

summary(interstep)

##
## Call:
## glm(formula = whether.he.she.donated.blood.in.March.2007 ~ Recency..months. *
##     Frequency..times. * Time..months., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9828  -0.7314  -0.4612  -0.2570   2.3918
##
## Coefficients:
##                                                   Estimate Std. Error
## (Intercept)                                     -1.1768198  0.3683353
## Recency..months.                                -0.0585015  0.0496780
## Frequency..times.                                0.4552780  0.0944032
## Time..months.                                   -0.0232070  0.0103680
## Recency..months.:Frequency..times.              -0.0284082  0.0128723
## Recency..months.:Time..months.                   0.0009083  0.0012139
## Frequency..times.:Time..months.                 -0.0036012  0.0010170
## Recency..months.:Frequency..times.:Time..months.  0.0002541  0.0001437
##                                                 z value Pr(>|z|)
## (Intercept)                                      -3.195 0.001398 **
## Recency..months.                                 -1.178 0.238951
## Frequency..times.                                 4.823 1.42e-06 ***
## Time..months.                                    -2.238 0.025200 *
```

```
## Recency..months.:Frequency..times.                -2.207 0.027320 *
## Recency..months.:Time..months.                     0.748 0.454328
## Frequency..times.:Time..months.                   -3.541 0.000399 ***
## Recency..months.:Frequency..times.:Time..months.   1.768 0.077007 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 555.65  on 499  degrees of freedom
## Residual deviance: 457.88  on 492  degrees of freedom
## AIC: 473.88
##
## Number of Fisher Scoring iterations: 5
```

```
predictor = predict.glm(interstep,newdata=test)
interpredict = cbind(predictor,test[,5])

a=ROC(interpredict)
```





So... that's actually the best prediction (Figure 5), if you give it points for less parameters (kinda), but it's still nothing to write home about. Probably the interaction is meaningful, so it's helping a little, but we remain unamused by the performance of this predictor.

In various runs, the logistic-with-interactions precision seems generally more reliable than any other predictor, but I haven't strictly quantified it with cross-validation. Specifically, the first few predictions here seem to be more accurate than the others, and precision takes longer to decay.

Addendum

After doing this analysis, I looked at the paper from which the dataset came. They made things more complicated by treating the blood donation visits as a series of Bernoulli trials. Maybe they had more complete data than we, but it seemed to me that without knowing the starting point or the number of actual blood drives these people had gone through it was weird to model it that way. And if I interpret their performance properly, their model is no better or a little worse than the ones I present.

Given that there are really only 3 independent variables for prediction ("Monetary" is just a linear transformation of "Frequency"), this is pretty squarely a high-n low-p problem. My intuition is that scraping for slightly better performance with more complicated methods is more likely to cause trouble and mislead people through overfitting than to add any additional power. With this somewhat skeptical view of how data analysis is generally performed, I rest my case.

Addendum 2: Nearest neighbor.

Apparently nearest-neighbor is good. I am trying it out. Figure 6: k=2, Figure 7: k=3, Figure 8: k=4, Figure 9: k=5.

```
library(class)
nn2_pred = knn(train[,1:4],test=test[,1:4] ,cl=train[,5],k=2)
nn2_predict = cbind(nn2_pred,test[,5])

a=ROC(nn2_predict)


nn3_pred = knn(train[,1:4],test=test[,1:4] ,cl=train[,5],k=3)
nn3_predict = cbind(nn3_pred,test[,5])
a=ROC(nn3_predict)


nn4_pred = knn(train[,1:4],cl=train[,5],test=test[,1:4] ,k=4)
nn4_predict = cbind(nn4_pred,test[,5])
a=ROC(nn4_predict)


nn5_pred = knn(train[,1:4],test[,1:4],cl=train[,5] ,k=5)
nn5_predict = cbind(nn5_pred,test[,5])
a=ROC(nn5_predict)
```

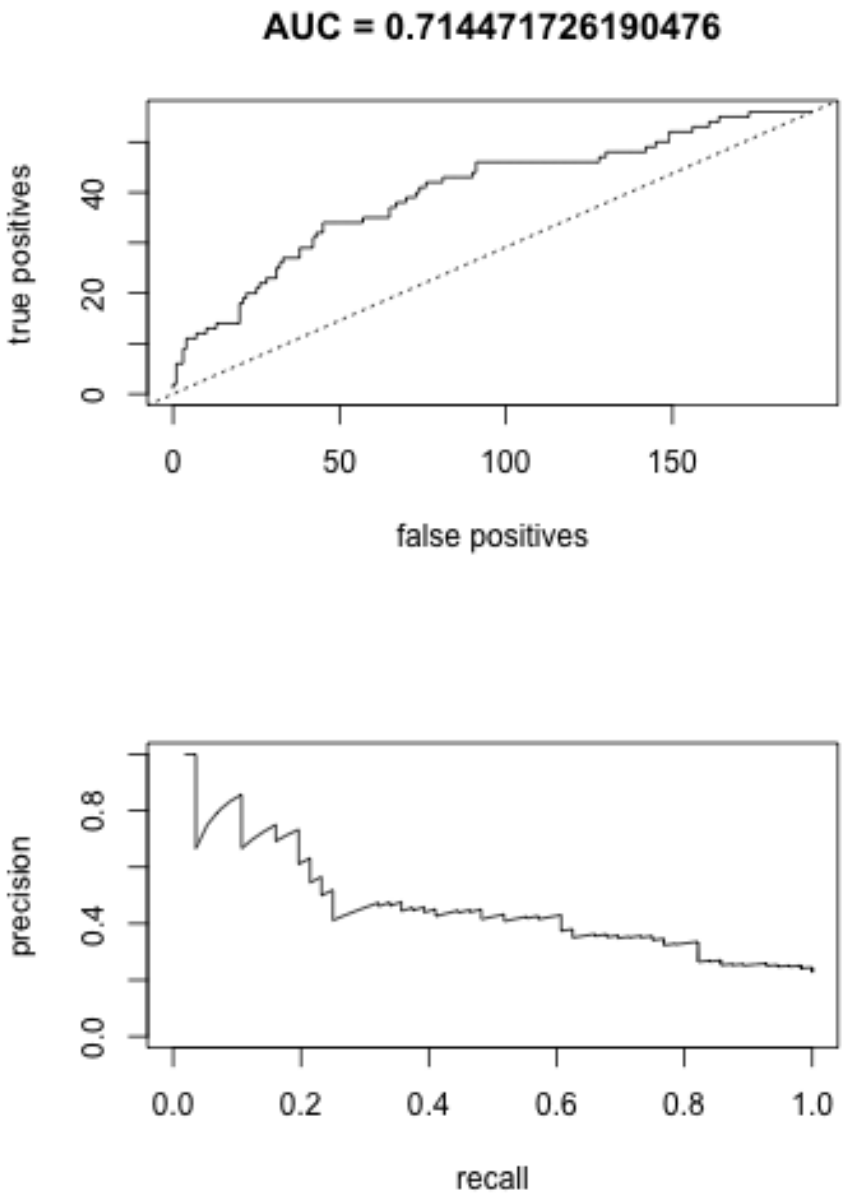


Figure 5: Performance of kNN with k=2.

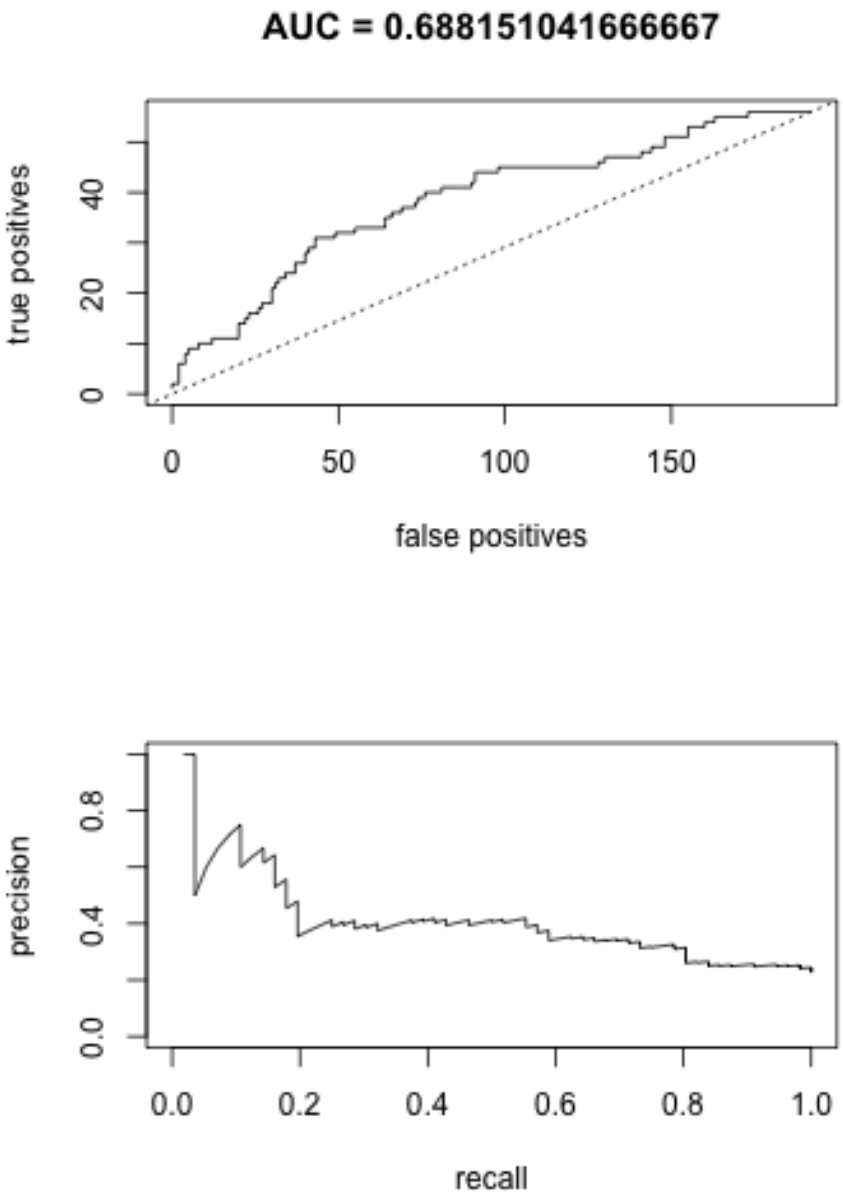


Figure 6: Performance of kNN with k=3.

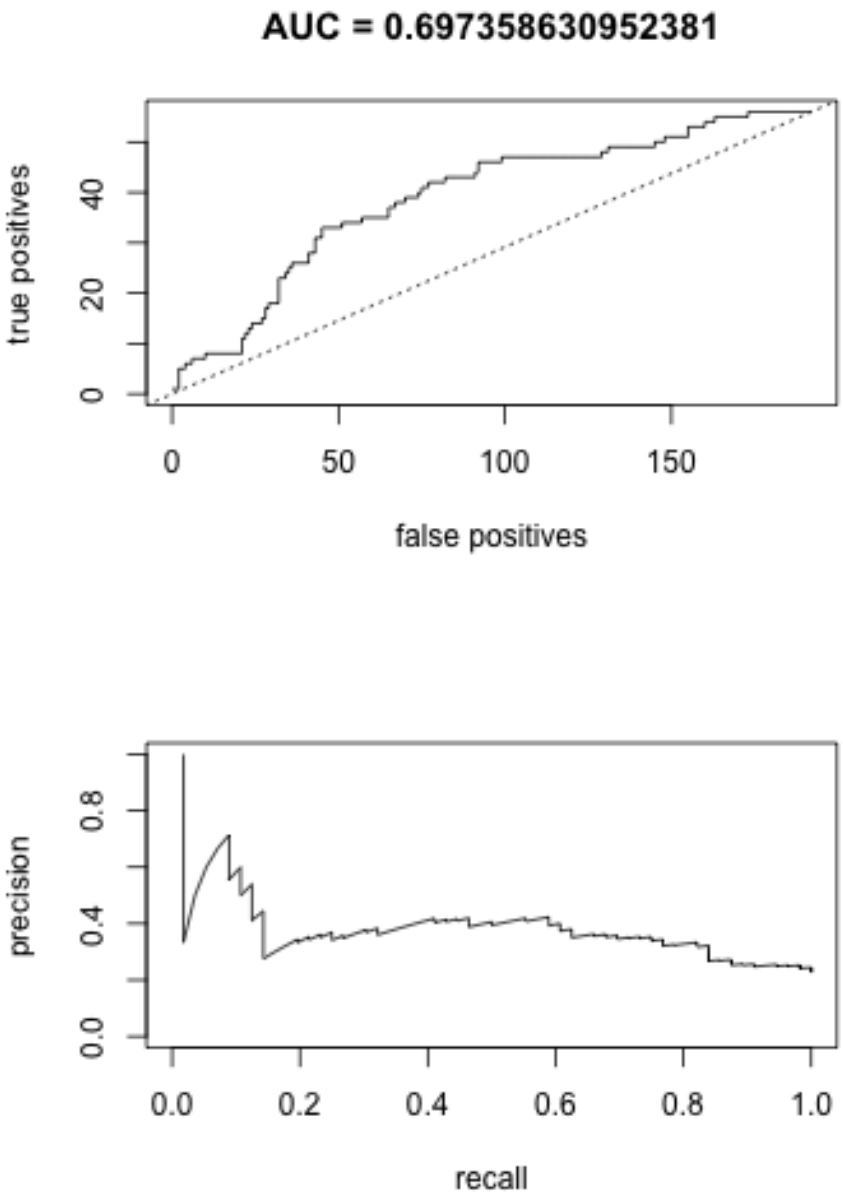


Figure 7: Performance of kNN with k=4.

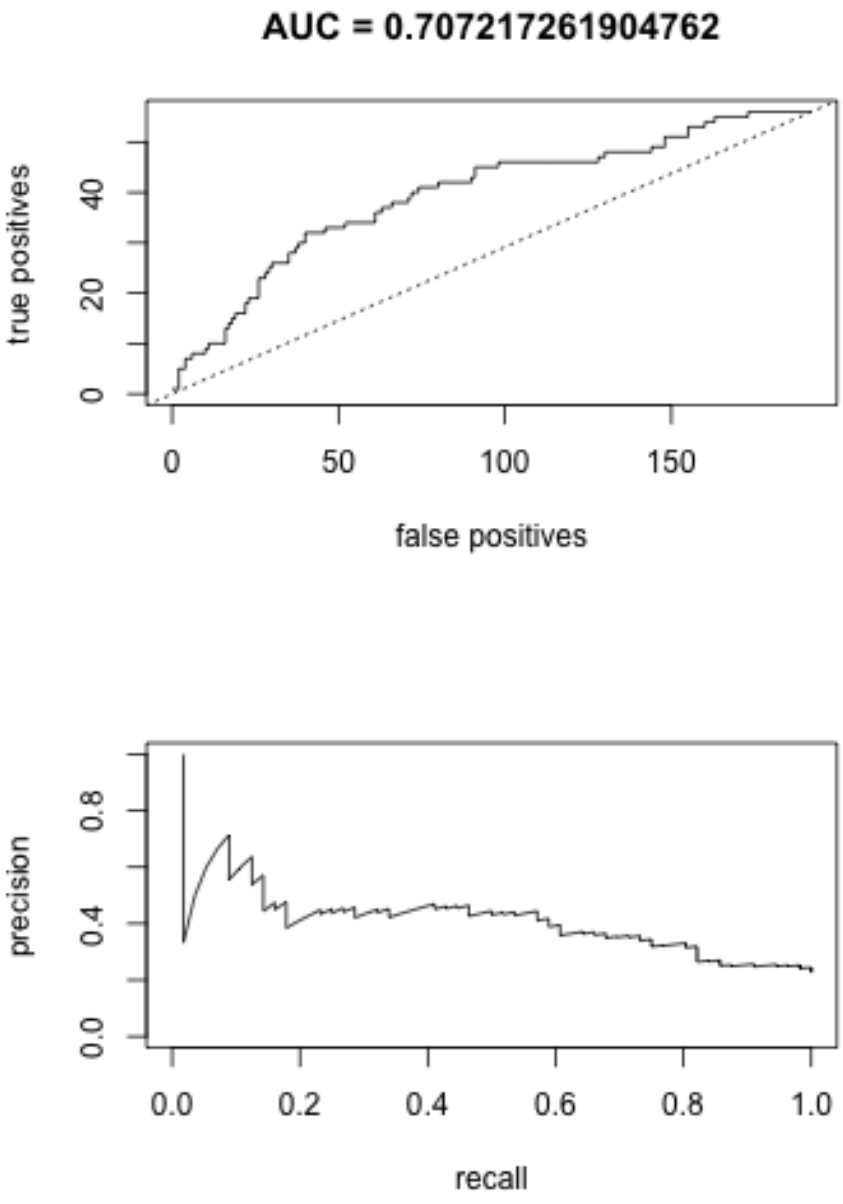


Figure 8: Performance of kNN with k=5.