STATS 315B - Homework 2

Rachael Caelie (Rocky) Aikens, Christine Tataru, and Daniel Sosa

May 14, 2018

Problem 1:

(10) Random forests predict with an ensemble of bagged trees each trained on a bootstrap sample randomly drawn from the original training data. Additional random variation among the trees is induced by choosing the variable for each split from a small randomly chosen subset of all of the predictor variables when building each tree. What are the advantages and disadvantages of this random variable selection strategy? How can one introduce additional tree variation in the forest without randomly selecting subsets of variables?

Advantages: Without the random variable selection strategy, the trees from the bootstrapped samples are likely to be highly correllated: Since the input samples are very similar, the trees may tend to split on the same subset of highly correllated variables. However, averaging a forest of near-identical trees adds little value when it comes to decreasing the variance of the overall classifier. The random variable selection strategy introduces variation among the trees by ensuring that they are not all splitting on the same sequence of variables. This breaks down the correllations between trees in the forest.

Disadvantages: Suppose (as is very often the case), that many of the model variables are uninformative for prediction. In this case, the random variable selection strategy is likely to generate many trees which have splits on relatively useless variables.

Additional variation can be induced by: - limiting the sample sizes from which each base learner is trained. - during training, using random split points for each feature under consideration when constructing a split, rather than the optimal split point for each feature (as in the ExtraTrees approach).

Problem 2:

(5) Why is it necessary to use regularization in linear regression when the number of predictor variables is greater than the number of observations in the training sample? Explain how regularization helps in this case. Are there other situations where regularization might help? What is the potential disadvantage of introducing regularization? Why is sparsity a reasonable assumption in the boosting context. Is it always? If not, why not?

We showed in 315A that, when $n < p$, there is an infinite number of equally optimal solutions to the linear least squares problem. In these cases, all of those optimal solutions are likely to be highly overfit to the training data. Regularization helps us simplify the linear model we produce (decrease model variance), and allows us to select for models which are in line with our expectations (i.e. sparse ones) by adding a "prior" for our coefficients. This same logic applies when $n \geq p$, but p is very large, and/or many of our variables are uninformative.

However, introducing regularization adds bias to our model: too much regularization can be just as bad for performance as too little. At the extreme, putting infinite weight on the regularization in our model training results in a model with all coefficients equal to zero, which is useless.

In the boosting context, the "predictor variables" for our linear regression are the results from all possible base learners. However, almost all base learners are probably very poor predictors of the outcome; only a small subset of them are effective and should have nonzero weight. This means we generally want a sparse model. This assumption might not hold true if we happened to be dealing with a small base learner function class in which nearly all models were effective predictors. However, in practice, this nearly never happens.

Problem 3:

(15) Assume squared error loss. Show that the convex members of the power family of penalties, except for the lasso, have the property that solutions for $\lambda > 0$ have nonzero values for all coefficients at each path point.

3a. Power family produces dense solutions:

We first consider the general minimum found by setting the derivative with respect to a_j to 0.

$$\frac{d}{da_j}\hat{R}(a) + \lambda P(a) = \frac{d}{da_j}((y - xa)^2 + \sum_k^K a_k^\gamma) = -2x_j(y - xa) + \gamma a_j^{\gamma-1} = 0$$

2) assume toward contradiction that a_j = 0

$$-2x_j(y - xa) = 0$$

3) Contradiction. The probability of $y - xa = 0$ is 0 for some continuous y. Therefore, the likelihood of a_j being exactly equal to 0 is very low.

3b: Elastic net produces sparse solutions: The elastic net takes the general form

$$(y - xa)^2 + \sum_{j=1}^{n}(\gamma - 1)a_j^2/2 + (2 - \gamma)|a_j|$$

We would like to take the derivative w.r.t a_j as we did above, however, we cannot differentiate the absolute value. We therefore take directional derivatives. As before, we assume $a_j = 0$ and that a_j is a minimum. For this to be true, the derivatives with respect to a_j in every direction must be greater than 0

Assume without loss of generality that we move in the cardinal direction $e_1 = [1, 0, 0, ...]$

$$D_{e_1} = \lim_{\epsilon \to 0} \frac{1}{\epsilon}[R(a + \epsilon e_1) - R(a) + \lambda P(a + \epsilon e_1) - \lambda P(a)]$$

As we can expressly take the derivative of the risk component, we can easily see:

$$R(a + \epsilon e_j) - R(a) = 2\epsilon e_1 x_j(y - x(a))$$

We then consider the penalty component

$$\lambda P(a_j + \epsilon e_1) - \lambda P(a_j) = \frac{\lambda}{2}(\gamma - 1)(a_j + \epsilon e_1)^2 + \lambda(2 - \gamma)|a_j + \epsilon e_1| - \frac{\lambda}{2}(\gamma - 1)a_j^2 - \lambda(2 - \gamma)|a_j| =$$

We have assumed that $a_j = 0$

$$\lambda P(a_j + \epsilon e_1) - \lambda P(a_j) = \frac{\lambda}{2}(\gamma - 1)(\epsilon e_1)^2 + \lambda(2 - \gamma)|\epsilon e_1|$$

Putting these components together, we mandate that the derivative must be greater than 0:

$$2\epsilon e_1 x_j(y - x(a)) + \frac{\lambda}{2}(\gamma - 1)(\epsilon e_1)^2 + \lambda(2 - \gamma)|\epsilon e_1| > 0$$

Simplifying, we find:

$$x_j(y - x(a)) > -\frac{\lambda}{4}(\gamma - 1)\epsilon e_1 - \frac{\lambda}{2}(2 - \gamma)$$

We then perform the same process moving in the cardinal direction $e_{-1} = [-1, 0, 0, ...]$ and find:

$$x_j(y - x(a)) < -\frac{\lambda}{4}(\gamma - 1)\epsilon e_{-1} - \frac{\lambda}{2}(2 - \gamma)$$

In contrast to the power family penalty, the elastic net penalty allows a range of values for the values of $(y - x(a))$. This range is parameterized by λ adn γ. This means that the probability of any coefficient a_j having a minimum of 0 for any λ is much (infinitely?) higher than it was for the power family.

Problem 4:

Let

$$\tilde{j} = argmin_{1 \leq j \leq J} \min_{\rho} E[(Y - \rho X_j)^2].$$

We would like to show that $\tilde{j} = j^*$, as defined in the problem set-up. We'll start by finding a closed form expression for ρ^*:

$$\rho^* = argmin_{\rho} E[(Y - \rho X_j)^2].$$

For any j, we can apply the linearity of the expectation operator to expand $E[(Y - \rho X_j)^2]$ to:

$$E[Y^2] - 2\rho E[YX_j] + \rho^2 E[X_j^2]$$

Regardless of the distributions of Y and X_j, the above expression is convex in ρ. We can optimize this easily by differentiating with respect to ρ:

$$\frac{\partial}{\partial \rho} E[(Y - \rho X_j)^2] = -2E[YX_j] + 2\rho E[X_j^2]$$

Setting the derivative to zero and solving, we have:

$$\rho^* = \frac{E[YX_j]}{E[X_j^2]}$$

In fact, since $E[X_j^2] = 1$ for all j we can ignore the denominator. Thus ρ^* is simply $E[YX_j]$.

We can now write $\tilde{j}$ as:

$$\tilde{j} = argmin_{1 \leq j \leq J} E[(Y - E[YX_j]X_j)^2].$$

Applying the same properties of expectation as before, we can expand $E[(Y - \frac{E[YX_j]}{E[X_j^2]} X_j)^2]$ to:

$$E[Y^2] - 2E[YX_j]^2 + E[YX_j]^2 E[X_j^2]$$

Since $E[Y^2]$ does not depend on our choice of j this term can be dropped. Noting that $E[X_j^2] = 1$, we can write $\tilde{j}$ most simply as:

$$\tilde{j} = argmin_{1 \leq j \leq J} - E[YX_j]^2.$$

Now, we can see that this is the same value of j that maximizes $|E[YX_j]|$. Thus $\tilde{j} = j^*$.

Problem 5:

The partial dependence of F(x) on z_l can be expressed as the expectation of F(x) after integrating over all values $z_{/l}$

$$E_{z/l}[F(x)] = \int F(x) f_{z/l}(t) dt$$

Expanding F(x):

$$\int F_l(z_l) f_{z/l}(t) dt + \int F_{z/l}(t) f_{z/l}(t) dt$$

Simplifying, we find

$$F_l(z_l) * 1 + \int F_{z/l}(t) f_{z/l}(t) dt$$

It can be seen that the second term is a function integrated over it's entire domain, which will result in some constant c.

$$E_{z/l}[F(x)] = F_l(z_l) + c$$

We now consider the dependence of F(x) on z_l ignoring the other variables

$$E[F(x)|z_l] = \int F(x) f_{z/l|z_l}$$

$$= \int F_l(z_l) f_{z/l|z_l}(t) dt + \int F_{/l}(t) f_{z/l|z_l}(t) dt$$

$$F_l(z_l) * 1 + \int F_{/l}(t) f_{z/l|z_l}(t) dt$$

Unlike in the previous case, the second term does NOT integrate to a constant, because the probability density $f_{z/l|z_l}(t)$ depends on the given value of z_l

$$F_l(z_l) + \int F_{/l}(t) f_{z/l|z_l}(t) dt$$

If the values of $z_{/l}$ are completely independent of z_l, the two methods will be identical.

Conceptually, this expresses the idea that some subset of variables can be used to extract information about an objective based on their correlation with predictive variables.

Problem 6:

Binary classification: Spam Email. The dataset is a collection of 4601 emails of which 1813 were considered spam, i.e. unsolicited commercial email. The data set consists of 58 attributes of which 57 are continuous predictors and one is a class label that indicates whether the email was consideredspam (1) or not (0). Among the 57 predictor attributes are: percentage of the word "free" in the email, percentage of exclamationmarks in the email, etc. See file spam_stats315B_names.txt for the full list of attributes. The goal is, of course, to predict whether or not an email is "spam". This data set is used for illus-tration in the tutorial Boosting with R Programming. The data set spam_stats315B_train.csv represents a subsample of these emails randomly selected from spam_stats315B.csvto be usedfor training. The file spam_stats315B_test.csv contains the remaining emails to be used for evaluating results.

(a) Based on the training data, fit a gbm model for predicting whether or not an email is spam , following the example in the tutorial. What is your estimate of the misclassification rate? Of all the spam emails of the test set what percentage was misclassified, and of all the non-spam emails in the test set what percentage was misclassified?

Below, we preprocess the data:

```
spam_train <- read.csv("spam_stats315B_train.csv", header=F)
spam_test <- read.csv("spam_stats315B_test.csv", header=F)
rflabs<-c("make", "address", "all", "3d", "our", "over", "remove",
  "internet","order", "mail", "receive", "will",
  "people", "report", "addresses","free", "business",
  "email", "you", "credit", "your", "font","000","money",
  "hp", "hpl", "george", "650", "lab", "labs",
  "telnet", "857", "data", "415", "85", "technology", "1999",
```

```
  "parts","pm", "direct", "cs", "meeting", "original", "project",
  "re","edu", "table", "conference", "semicolon", "left_bracket_round", "left_bracket_square", "exclaim
  "CAPAVE", "CAPMAX", "CAPTOT","type")
  # Names for predictors and response
colnames(spam_train) <- colnames(spam_test) <- rflabs

#normalize all columns by mean and variance
train_type = spam_train$type
spam_train <- data.frame(apply(select(spam_train, -type), 2,
                               function(vec) return((vec - mean(vec)) / sd(vec))))
spam_train$type = train_type

test_type = spam_test$type
spam_test <- data.frame(apply(select(spam_test, -type), 2,
                              function(vec) return((vec - mean(vec)) / sd(vec))))
spam_test$type = test_type
```

Next, we write a function which performs 5 fold cross validation for various levels of shrinkage and tree depth, and reports the results:

```
gbm_cv <- function(weights = NULL, depth = 6) {
  # train gbm with 5 fold cv and return cv performance
  gbm0 <- gbm(type ~ .,
              data=spam_train,
              interaction.depth = depth,
              shrinkage = 0.01,
              n.trees= 4000,
              bag.fraction=0.7,
              cv.folds=5,
              distribution="bernoulli", verbose=F)

  best_iter <- gbm.perf(gbm0, method="cv")
  cv_error <- gbm0$cv.error[best_iter]

  return(c(cv_error = cv_error, best_iter = best_iter))
}
```

Using this function, we perform cross-validation to select the interaction depth for our model.

```
# This code is computationally intensive; we do not run it on compile
depths <- 1:6

cv_results <- lapply(depths, function(depth) gbm_cv(depth = depth))
```

From this procedure, we find that the optimal interaction depth in cross validation is 1.

```
# train best unweighted model
best_depth <- 1
best_iter <- 4000
gbm_unweighted <- gbm(type ~ .,
                      data=spam_train,
                      interaction.depth = best_depth,
                      shrinkage = 0.01,
                      n.trees= best_iter,
                      bag.fraction=0.7,
                      distribution="bernoulli", verbose=F)
```

Now, we can analyze the performance of our model on the test set.

```
gbm_unweighted.test <- predict(gbm_unweighted, spam_test, type="response", n.trees= best_iter)
gbm_unweighted.labels <- as.numeric(gbm_unweighted.test >= 0.5)
misclassification.error <- sum(spam_test$type != gbm_unweighted.labels) / nrow(spam_test)*100

test.pos.inds <- which(spam_test$type == 1)
test.neg.inds <- which(spam_test$type == 0)
false.negs <- sum(spam_test[test.pos.inds,]$type != gbm_unweighted.labels[test.pos.inds]) / length(test
false.pos <- sum(spam_test[test.neg.inds,]$type != gbm_unweighted.labels[test.neg.inds]) / length(test.
```

The performance statistics below were calculated for our best unweighted gbm model on the hold-out test set.

Misclassification error: 4.6936115 Spam misclassified as non-spam (false neg): 6.1488673 Non-spam misclassified as spam (false pos): 3.7117904

(b) Your classifier in part (a) can be used as a spam filter. One of the possible disadvantages of such a spam filter is that it might filter out too many good (non-spam) emails. Therefore, a better spam filter might be the one that penalizes misclassifying non-spam emails more heavily than the spam ones. Suppose that you want to build a spam filter that throws out no more that 0.3% of the good (non-spam) emails. You have to find and use a cost matrix that penalizes misclassifying good emails as spam more than misclassifying spam emails as good by the method of trial and error. Once you have constructed your final spam filter with the property described above, answer the following questions:

```
weights = ifelse(spam_train$type == 0, 100, 3)

gbm_weighted <- gbm(type ~ .,
                    data=spam_train,
                    weights = weights,
                    interaction.depth = best_depth,
                    shrinkage = 0.01,
                    n.trees= best_iter,
                    bag.fraction=0.7,
                    distribution="bernoulli", verbose=F)

gbm_weighted.test <- predict(gbm_weighted, spam_test, type="response", n.trees= best_iter)
gbm_weighted.labels <- as.numeric(gbm_weighted.test >= 0.5)
w.misclassification.error <- sum(spam_test$type != gbm_weighted.labels) / nrow(spam_test)*100

w.false.negs <- sum(spam_test[test.pos.inds,]$type != gbm_weighted.labels[test.pos.inds]) / length(test
w.false.pos <- sum(spam_test[test.neg.inds,]$type != gbm_weighted.labels[test.neg.inds]) / length(test.
```

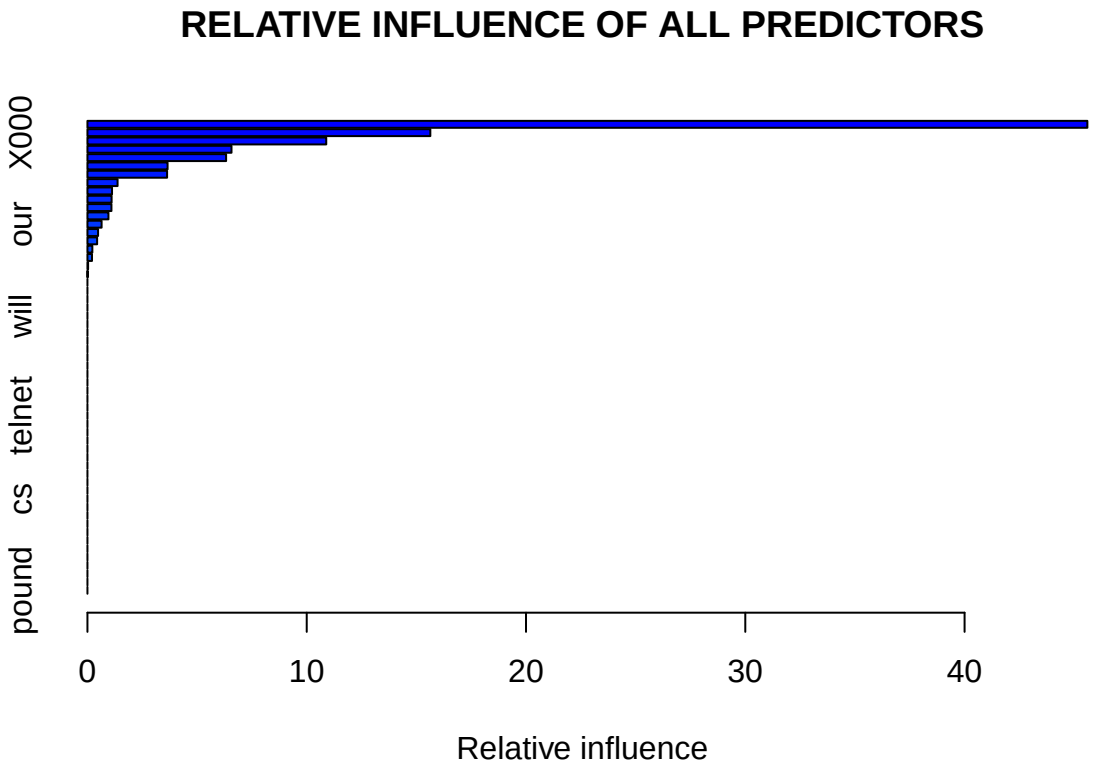
(i) What is the overall misclassification error of your final filter and what is the percentage of good emails and spam emails that were misclassified respectively?

The performance statistics below were calculated for our best unweighted gbm model on the hold-out test set. Clearly, the false negative rate is much higher as a tradeoff with our low false positive rate.

Misclassification: 17.1447197 Spam emails misclassified: 42.2330097 Good emails misclassified: 0.2183406

(ii) What are the important variables in discriminating good emails from spam for your spamfilter?

```
summary(gbm_weighted, main="RELATIVE INFLUENCE OF ALL PREDICTORS")
```



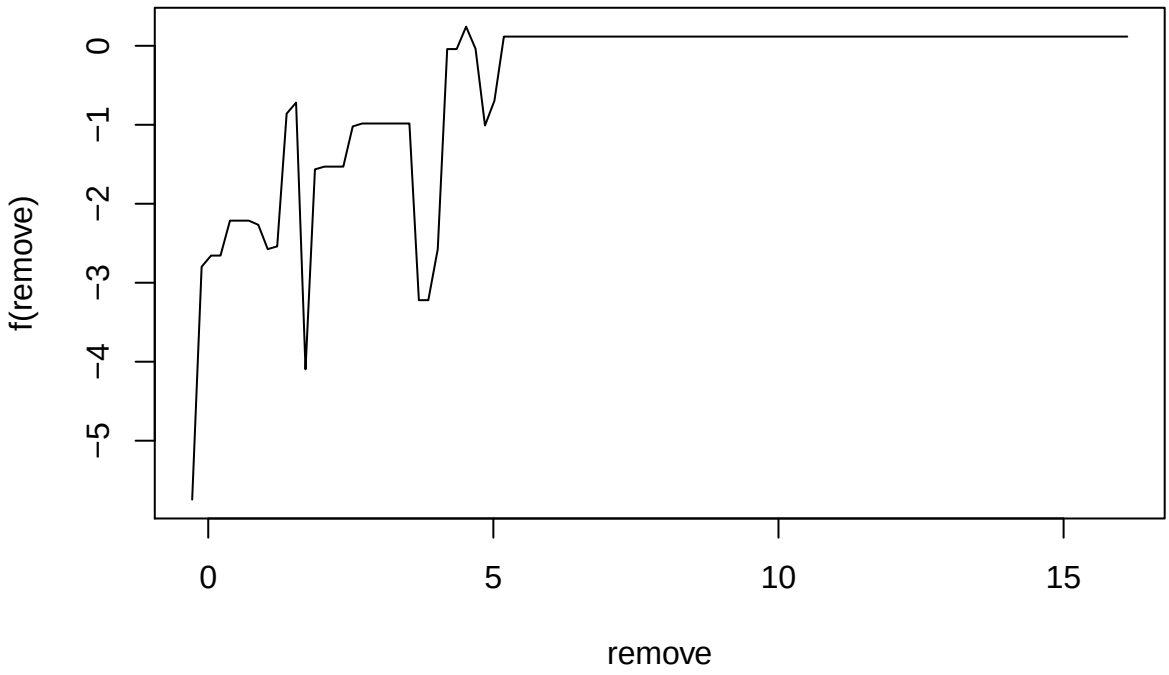

```
##                                 var     rel.inf
## remove                       remove 45.59952011
## X000                           X000 15.63598966
## money                         money 10.88958099
## exclaim                     exclaim  6.56883302
## dollar                       dollar  6.32304323
## free                           free  3.65428597
## credit                       credit  3.63298440
## CAPMAX                       CAPMAX  1.37397150
## font                           font  1.12149459
## CAPAVE                       CAPAVE  1.10195265
## X3d                             X3d  1.09444159
## CAPTOT                       CAPTOT  0.95842556
## our                             our  0.64605670
## business                   business  0.48472535
## your                           your  0.44112717
## receive                     receive  0.22320639
## internet                   internet  0.20802717
## over                           over  0.03083407
## all                             all  0.01149987
## make                           make  0.00000000
## address                     address  0.00000000
## order                       order  0.00000000
## mail                           mail  0.00000000
## will                           will  0.00000000
## people                       people  0.00000000
## report                       report  0.00000000
```

```
## addresses                     addresses  0.00000000
## email                             email  0.00000000
## you                                 you  0.00000000
## hp                                   hp  0.00000000
## hpl                                 hpl  0.00000000
## george                           george  0.00000000
## X650                               X650  0.00000000
## lab                                 lab  0.00000000
## labs                               labs  0.00000000
## telnet                           telnet  0.00000000
## X857                               X857  0.00000000
## data                               data  0.00000000
## X415                               X415  0.00000000
## X85                                 X85  0.00000000
## technology                   technology  0.00000000
## X1999                             X1999  0.00000000
## parts                             parts  0.00000000
## pm                                   pm  0.00000000
## direct                           direct  0.00000000
## cs                                   cs  0.00000000
## meeting                       meeting  0.00000000
## original                     original  0.00000000
## project                       project  0.00000000
## re                                   re  0.00000000
## edu                                 edu  0.00000000
## table                             table  0.00000000
## conference                   conference  0.00000000
## semicolon                     semicolon  0.00000000
## left_bracket_round   left_bracket_round  0.00000000
## left_bracket_square left_bracket_square  0.00000000
## pound                             pound  0.00000000
```

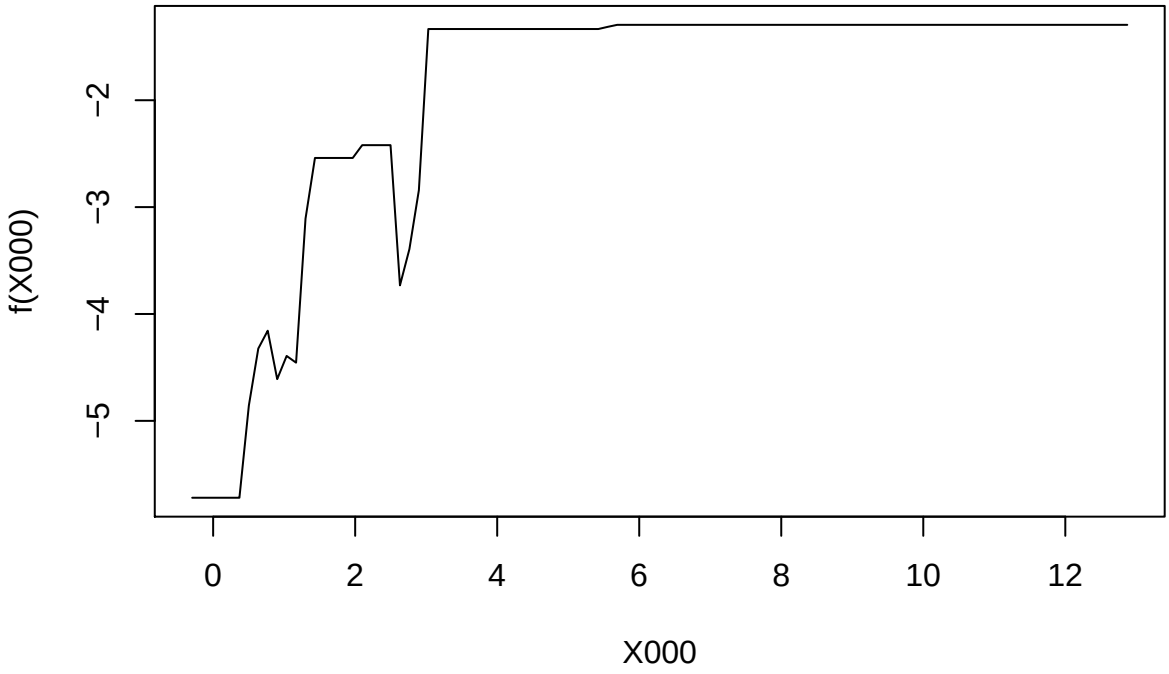
Using the built-in variable importance evaluator from the gbm package, we find that the most important variables are presence of the words "remove," "money", "000", "credit", and "free", and the number of exclaimation points and dollar signs. This makes logical sense based on the words and characters we tend to see in spam emails as opposed to others.

(iii) Using the interpreting tools provided by gbm, describe the dependence of the response on the most important attributes.

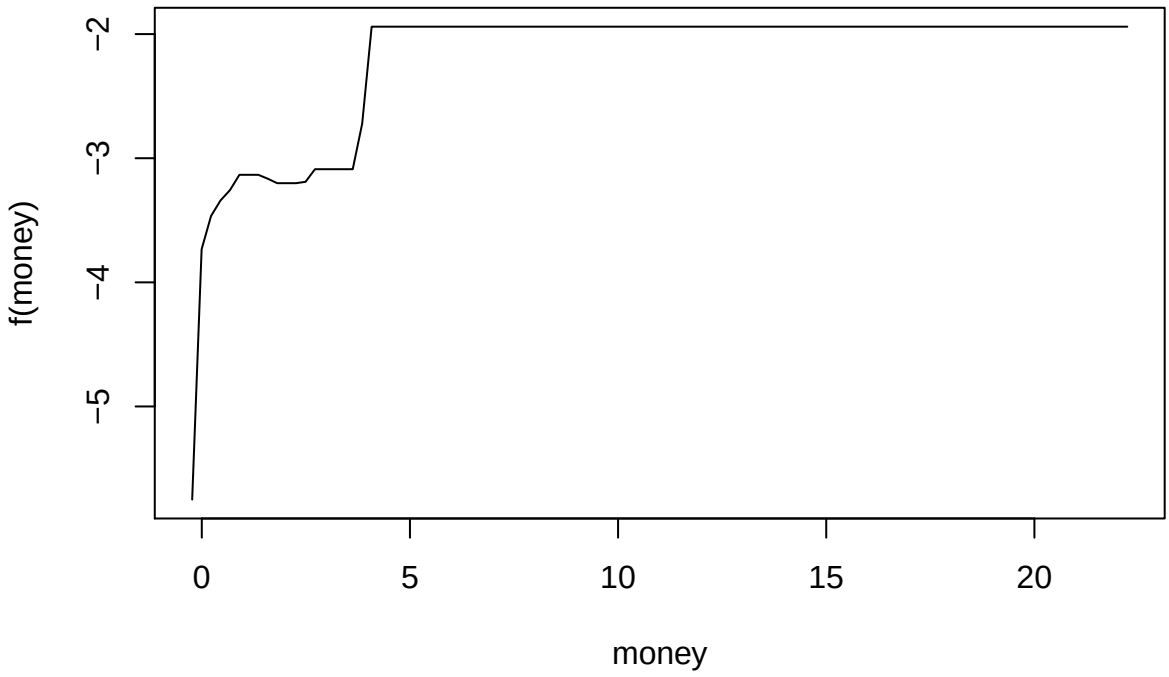
```
plot(gbm_weighted, i="remove")
```



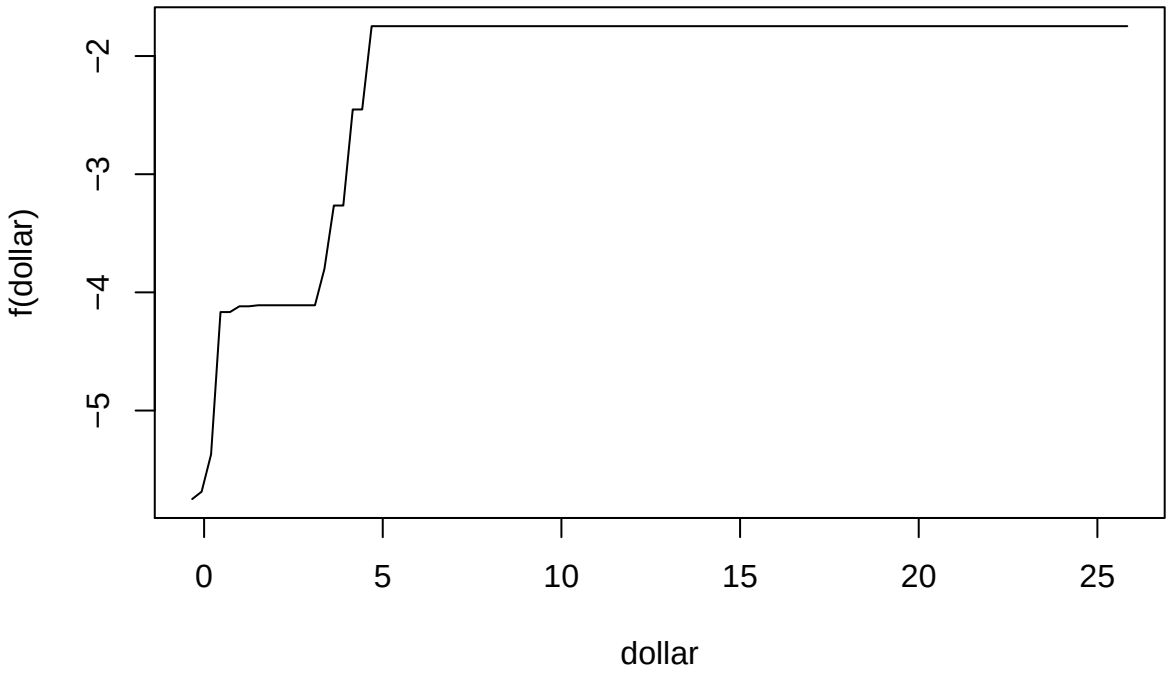

```
plot(gbm_weighted, i = "X000")
```




```
plot(gbm_weighted, i = "money")
```




```
plot(gbm_weighted, i = "dollar")
```



As might be expected, the number of exclamation points and dollar signs in a text are among the top features used in classifying it as spam/non-spam. Interestingly, the number of times the word "remove" appears in the test is also useful. (One can imagine, for example, that many spam emails include a tag line to "remove yourself from this mailing list.") All the variables show an upwards trend, suggesting that higher frequencies of these features results in a higher spam score.

However, the number of characters or words tends to be important. For example, we can see that the presence of a single dollar sign does not significantly increase the probability that an email will be classified as spam. However, the presence of several dollar signs corresponds to a much higher rate of "spam" labeling. Thus, a moderate number of "spammy" word or character occurances (normalized by mean and variance of occurences

throughout the dataset) will not necessarily "incriminate" an email as a spam email, but several instances of a "spammy" word is highly penalized.

Problem 7:

(15) Regression: California Housing.The data set calif_stats315B.csv consists of aggregated data from 20,640 California census blocks (from the 1990 census). The goal is to predict the median house value in each neighborhood from the others described in calif_stats315B.txt. Fit a gbm model to the data and write a short report that should include at least

```
calif <- read.csv("calif_stats315B.csv", header=F)
colnames(calif) <- c("Value", "Income", "Age", "Rooms", "Bedrooms", "Population", "Occupancy", "Latitud

gbm.calif <- gbm(Value ~ ., data=calif, train.fraction=0.8, interaction.depth=4, shrinkage = 0.05, n.tr
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2277          1.4390     0.0500    0.0589
##      2        1.1717          1.3715     0.0500    0.0554
##      3        1.1215          1.3095     0.0500    0.0503
##      4        1.0752          1.2536     0.0500    0.0468
##      5        1.0342          1.2020     0.0500    0.0407
##      6        0.9956          1.1558     0.0500    0.0384
##      7        0.9595          1.1117     0.0500    0.0350
##      8        0.9272          1.0742     0.0500    0.0315
##      9        0.8969          1.0362     0.0500    0.0289
##     10        0.8702          1.0016     0.0500    0.0270
##     20        0.6808          0.7736     0.0500    0.0131
##     40        0.4920          0.6107     0.0500    0.0074
##     60        0.3885          0.5651     0.0500    0.0027
##     80        0.3398          0.5611     0.0500    0.0014
##    100        0.3125          0.5480     0.0500    0.0007
##    120        0.2964          0.5392     0.0500    0.0005
##    140        0.2836          0.5268     0.0500    0.0007
##    160        0.2750          0.5258     0.0500    0.0001
##    180        0.2673          0.5149     0.0500    0.0005
##    200        0.2609          0.5140     0.0500    0.0001
##    220        0.2559          0.5021     0.0500   -0.0000
##    240        0.2510          0.5005     0.0500    0.0000
##    260        0.2469          0.5004     0.0500    0.0004
##    280        0.2428          0.4995     0.0500    0.0003
##    300        0.2391          0.4958     0.0500    0.0002
##    320        0.2359          0.4949     0.0500   -0.0000
##    340        0.2331          0.4936     0.0500   -0.0000
##    360        0.2304          0.4894     0.0500    0.0002
##    380        0.2277          0.4915     0.0500   -0.0001
##    400        0.2252          0.4921     0.0500   -0.0000
##    420        0.2225          0.4903     0.0500    0.0002
##    440        0.2200          0.4857     0.0500   -0.0000
##    460        0.2179          0.4851     0.0500   -0.0000
##    480        0.2160          0.4824     0.0500   -0.0000
##    500        0.2143          0.4834     0.0500   -0.0001
##    520        0.2130          0.4846     0.0500    0.0000
##    540        0.2108          0.4765     0.0500   -0.0001
```

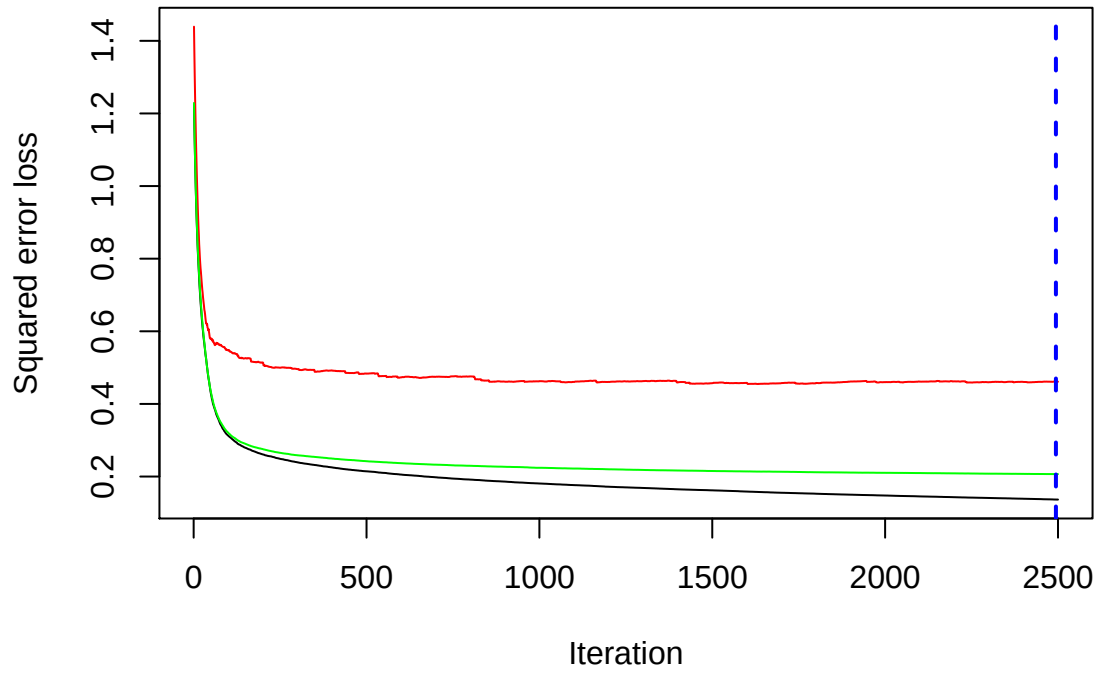
```
##    560        0.2089          0.4744     0.0500    0.0003
##    580        0.2070          0.4754     0.0500   -0.0001
##    600        0.2054          0.4730     0.0500   -0.0001
##    620        0.2039          0.4748     0.0500   -0.0000
##    640        0.2025          0.4735     0.0500   -0.0001
##    660        0.2009          0.4721     0.0500    0.0002
##    680        0.1994          0.4740     0.0500   -0.0001
##    700        0.1979          0.4746     0.0500    0.0000
##    720        0.1966          0.4746     0.0500   -0.0001
##    740        0.1951          0.4744     0.0500    0.0000
##    760        0.1939          0.4763     0.0500   -0.0000
##    780        0.1925          0.4753     0.0500    0.0000
##    800        0.1916          0.4757     0.0500   -0.0000
##    820        0.1904          0.4678     0.0500   -0.0000
##    840        0.1892          0.4641     0.0500   -0.0001
##    860        0.1879          0.4611     0.0500    0.0000
##    880        0.1870          0.4623     0.0500   -0.0000
##    900        0.1861          0.4614     0.0500   -0.0000
##    920        0.1849          0.4614     0.0500   -0.0001
##    940        0.1837          0.4622     0.0500   -0.0000
##    960        0.1827          0.4613     0.0500   -0.0000
##    980        0.1817          0.4617     0.0500   -0.0001
##   1000        0.1809          0.4625     0.0500   -0.0000
##   1020        0.1800          0.4616     0.0500   -0.0000
##   1040        0.1790          0.4631     0.0500   -0.0001
##   1060        0.1782          0.4623     0.0500   -0.0000
##   1080        0.1773          0.4598     0.0500    0.0000
##   1100        0.1764          0.4606     0.0500   -0.0000
##   1120        0.1756          0.4621     0.0500   -0.0000
##   1140        0.1749          0.4630     0.0500   -0.0000
##   1160        0.1739          0.4640     0.0500   -0.0000
##   1180        0.1729          0.4605     0.0500   -0.0000
##   1200        0.1719          0.4606     0.0500    0.0000
##   1220        0.1711          0.4612     0.0500    0.0001
##   1240        0.1705          0.4617     0.0500   -0.0000
##   1260        0.1698          0.4617     0.0500   -0.0000
##   1280        0.1691          0.4625     0.0500   -0.0000
##   1300        0.1685          0.4628     0.0500   -0.0000
##   1320        0.1679          0.4627     0.0500   -0.0000
##   1340        0.1671          0.4627     0.0500    0.0000
##   1360        0.1663          0.4635     0.0500   -0.0000
##   1380        0.1657          0.4633     0.0500   -0.0000
##   1400        0.1649          0.4591     0.0500   -0.0001
##   1420        0.1642          0.4601     0.0500   -0.0001
##   1440        0.1636          0.4559     0.0500   -0.0000
##   1460        0.1630          0.4560     0.0500   -0.0000
##   1480        0.1626          0.4564     0.0500   -0.0000
##   1500        0.1620          0.4571     0.0500   -0.0000
##   1520        0.1614          0.4588     0.0500   -0.0000
##   1540        0.1608          0.4574     0.0500   -0.0001
##   1560        0.1601          0.4573     0.0500   -0.0000
##   1580        0.1593          0.4574     0.0500   -0.0000
##   1600        0.1586          0.4579     0.0500   -0.0000
##   1620        0.1580          0.4552     0.0500   -0.0000
```

```
##   1640        0.1573          0.4554     0.0500   -0.0000
##   1660        0.1567          0.4556     0.0500   -0.0000
##   1680        0.1559          0.4564     0.0500   -0.0001
##   1700        0.1553          0.4573     0.0500   -0.0000
##   1720        0.1548          0.4586     0.0500   -0.0000
##   1740        0.1543          0.4586     0.0500   -0.0000
##   1760        0.1538          0.4563     0.0500   -0.0000
##   1780        0.1532          0.4562     0.0500   -0.0000
##   1800        0.1527          0.4573     0.0500   -0.0000
##   1820        0.1520          0.4586     0.0500   -0.0000
##   1840        0.1515          0.4587     0.0500   -0.0000
##   1860        0.1510          0.4591     0.0500   -0.0000
##   1880        0.1505          0.4604     0.0500   -0.0001
##   1900        0.1500          0.4617     0.0500   -0.0000
##   1920        0.1496          0.4620     0.0500   -0.0000
##   1940        0.1490          0.4629     0.0500   -0.0000
##   1960        0.1487          0.4610     0.0500   -0.0000
##   1980        0.1481          0.4590     0.0500   -0.0000
##   2000        0.1477          0.4600     0.0500   -0.0000
##   2020        0.1472          0.4605     0.0500   -0.0000
##   2040        0.1466          0.4609     0.0500   -0.0000
##   2060        0.1462          0.4605     0.0500   -0.0000
##   2080        0.1458          0.4606     0.0500   -0.0000
##   2100        0.1453          0.4616     0.0500    0.0000
##   2120        0.1449          0.4614     0.0500   -0.0000
##   2140        0.1445          0.4619     0.0500   -0.0000
##   2160        0.1440          0.4619     0.0500   -0.0000
##   2180        0.1435          0.4620     0.0500   -0.0001
##   2200        0.1431          0.4622     0.0500   -0.0000
##   2220        0.1425          0.4619     0.0500    0.0000
##   2240        0.1421          0.4590     0.0500   -0.0000
##   2260        0.1418          0.4595     0.0500   -0.0000
##   2280        0.1414          0.4603     0.0500    0.0000
##   2300        0.1409          0.4597     0.0500   -0.0000
##   2320        0.1405          0.4609     0.0500   -0.0001
##   2340        0.1401          0.4604     0.0500   -0.0000
##   2360        0.1397          0.4611     0.0500   -0.0000
##   2380        0.1392          0.4604     0.0500   -0.0000
##   2400        0.1388          0.4605     0.0500   -0.0000
##   2420        0.1384          0.4596     0.0500   -0.0000
##   2440        0.1380          0.4604     0.0500   -0.0000
##   2460        0.1374          0.4614     0.0500   -0.0000
##   2480        0.1371          0.4610     0.0500    0.0000
##   2500        0.1365          0.4610     0.0500   -0.0000
```

```
gbm.predict <- predict(gbm.calif, calif[-1])
```

```
## Using 1632 trees...
```

```
gbm.perf(gbm.calif, method="cv")
```

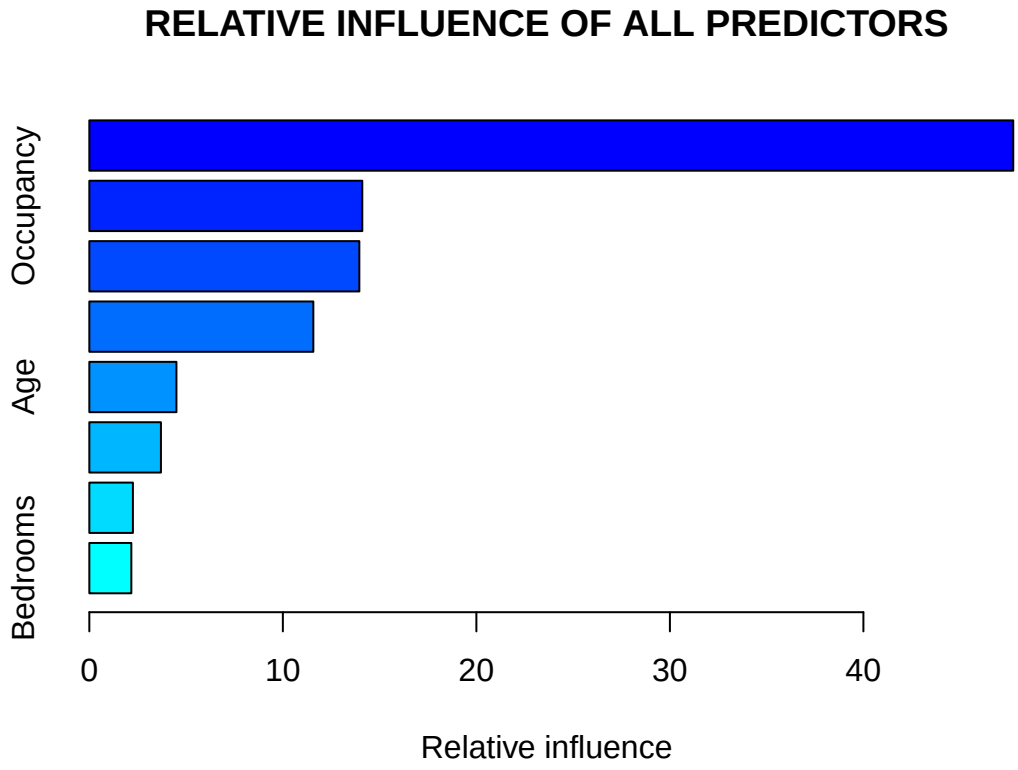



```
## [1] 2494
```

```
(MSE <- sum((gbm.predict - calif$Value)**2))
```

```
## [1] 4477.676
```

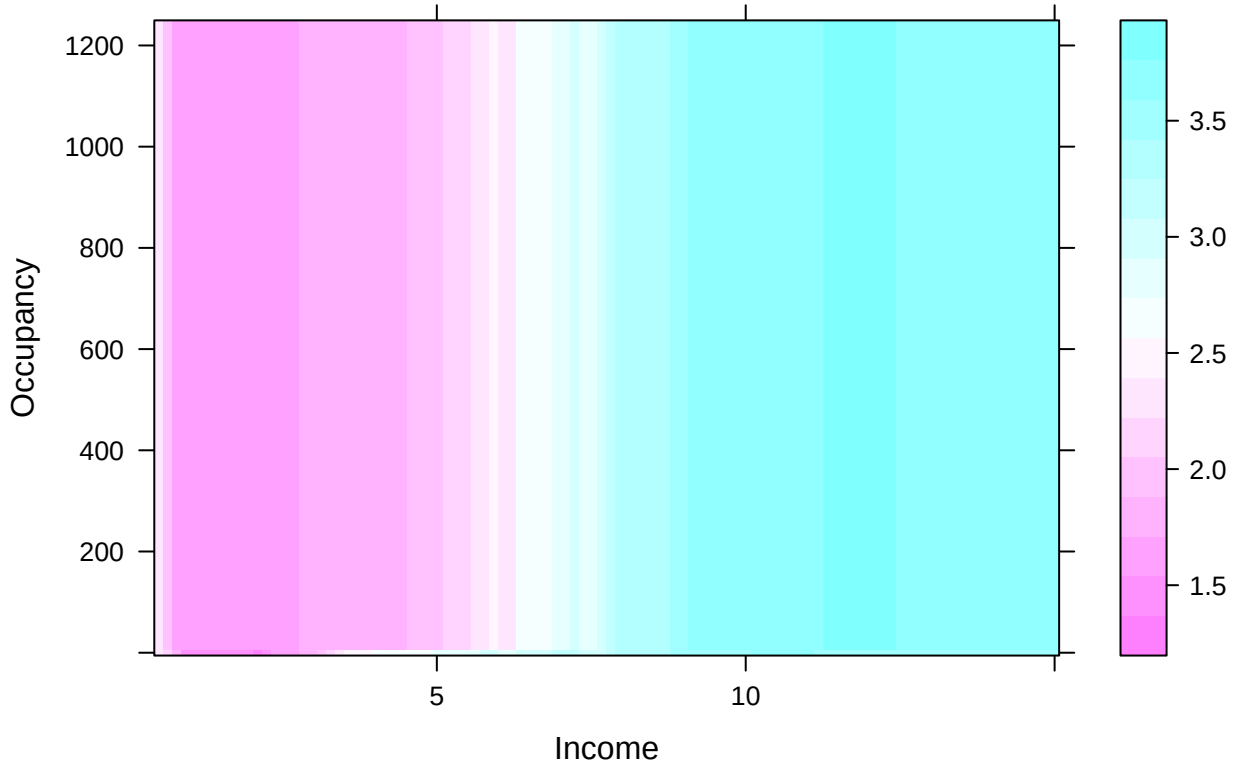
```
summary(gbm.calif, main="RELATIVE INFLUENCE OF ALL PREDICTORS")
```



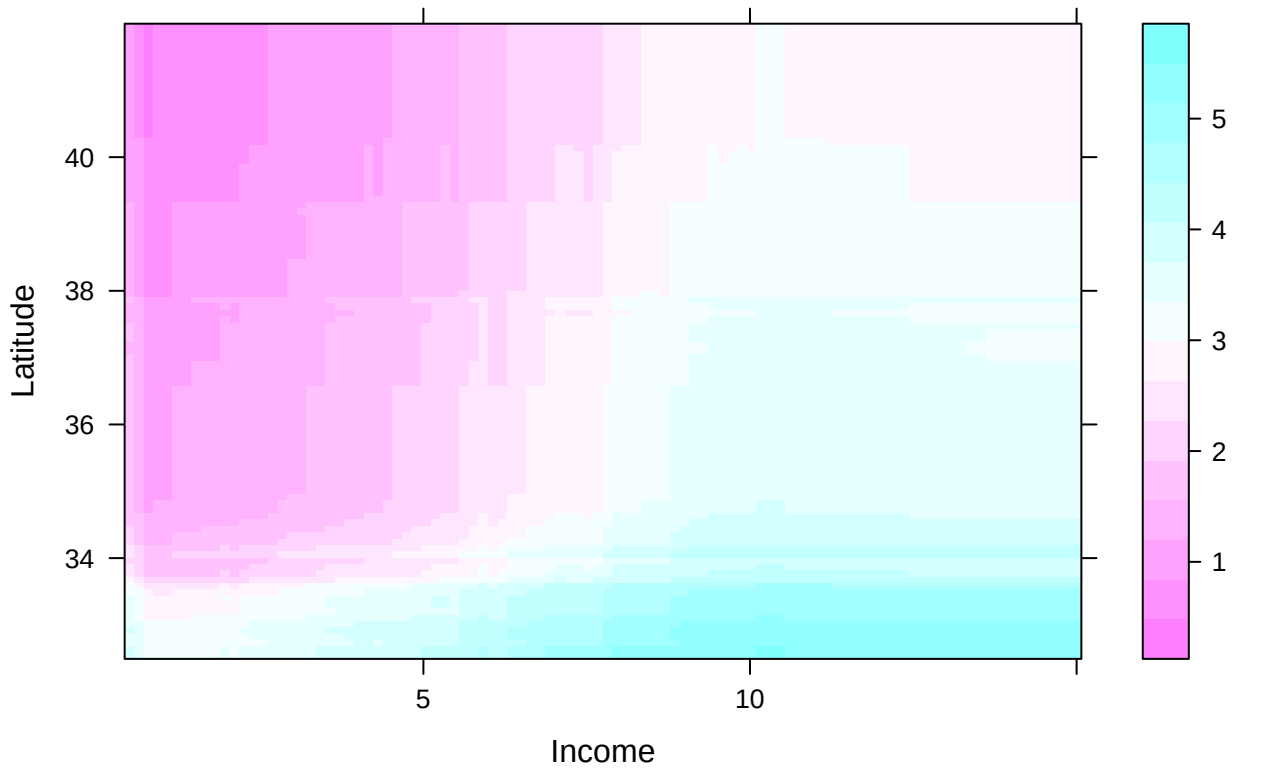

```
##                  var   rel.inf
## Income         Income 47.742753
```

```
## Occupancy   Occupancy 14.105923
## Longitude   Longitude 13.952941
## Latitude     Latitude 11.575511
## Age               Age  4.500359
## Rooms           Rooms  3.699686
## Population Population  2.252331
## Bedrooms     Bedrooms  2.170496
```

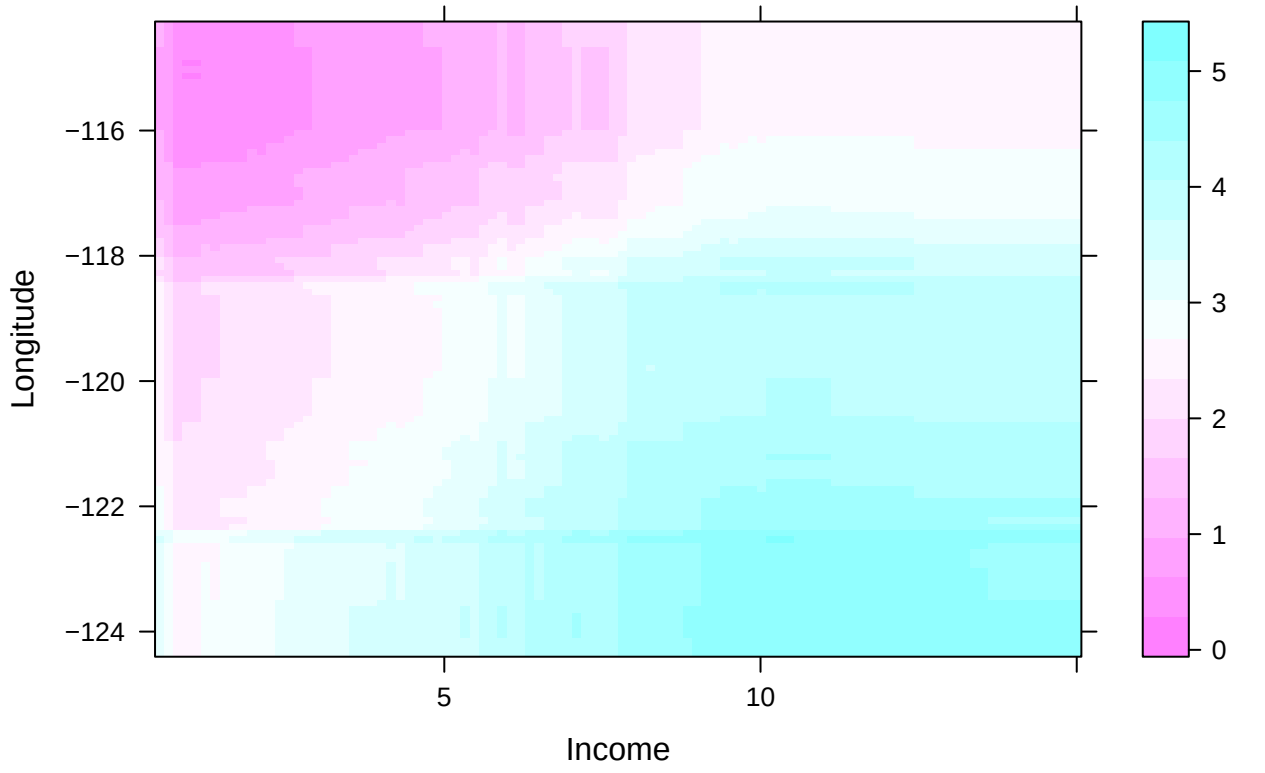
```
plot(gbm.calif, c(1,6))
```



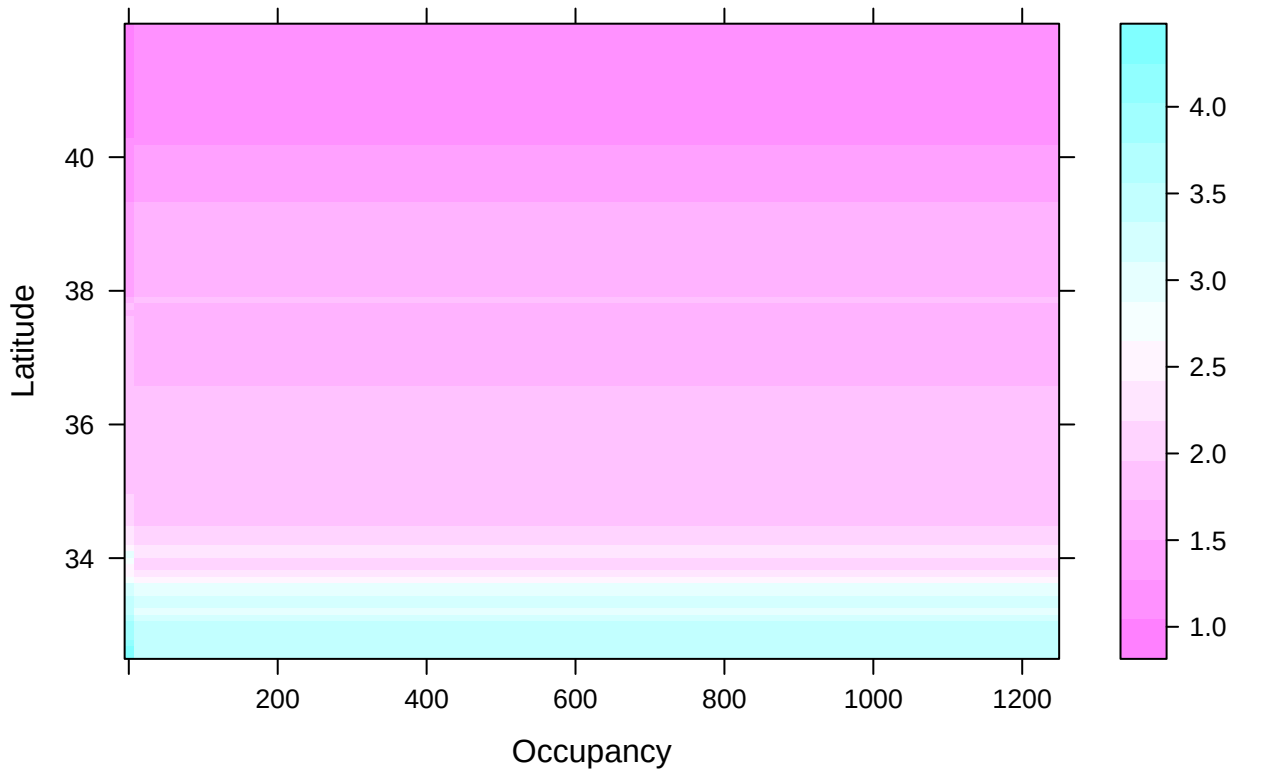

```
plot(gbm.calif, c(1,7))
```



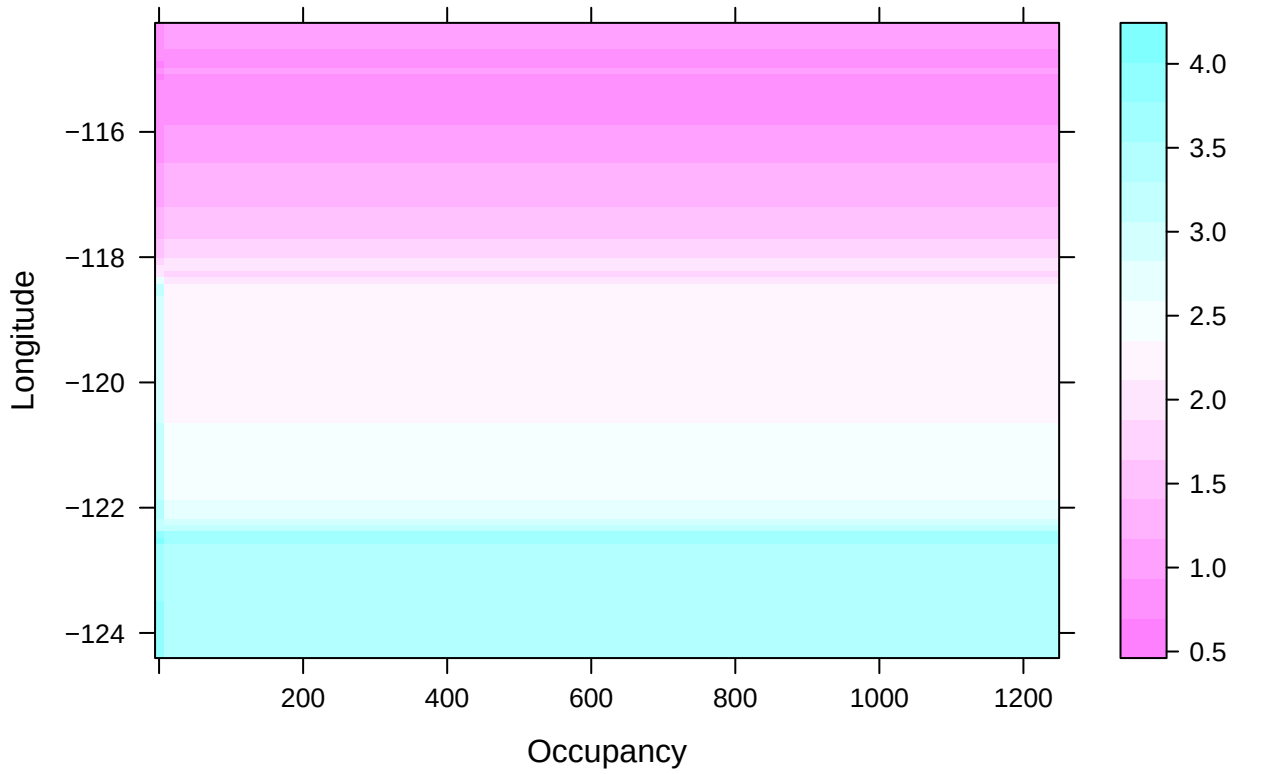

```
plot(gbm.calif, c(1,8))
```



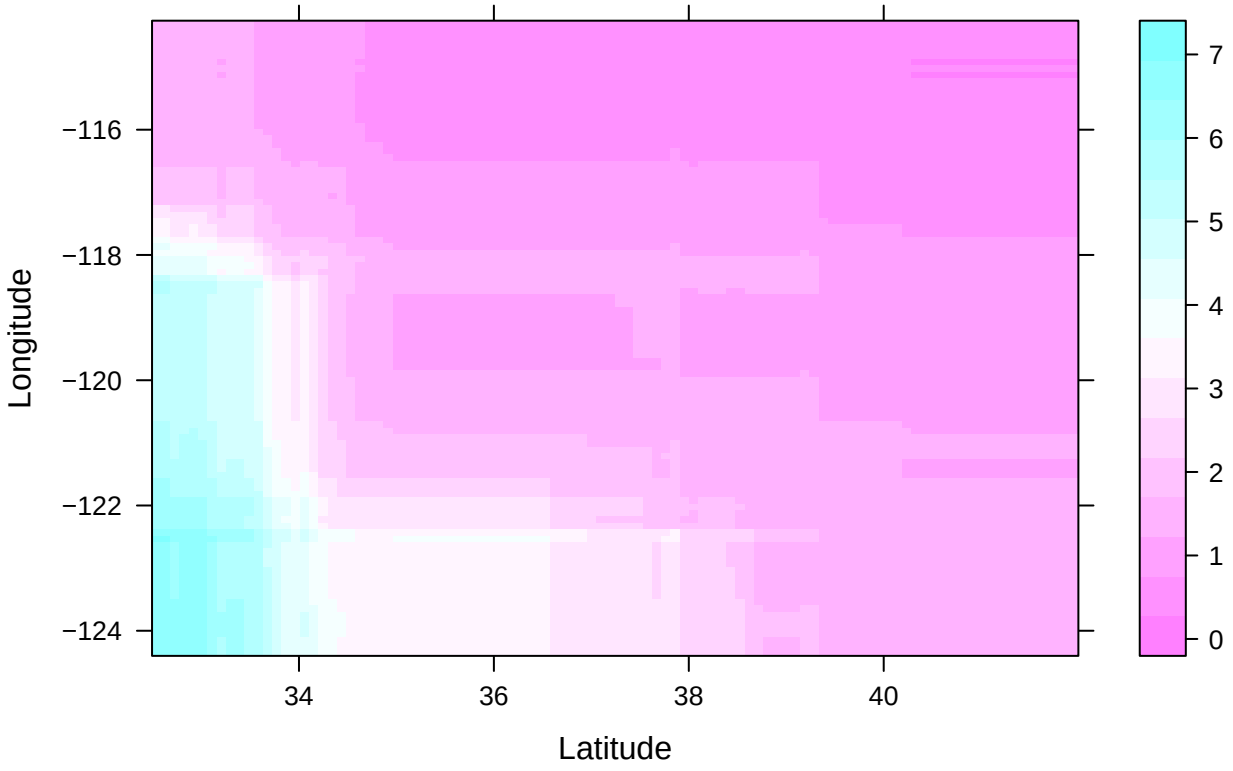

```
plot(gbm.calif, c(6,7))
```




```
plot(gbm.calif, c(6,8))
```




```
plot(gbm.calif, c(7,8))
```



(a) The prediction accuracy of gbm on the data set.

Mean squared error is 4424.596

(b) Identification of the most important variables.

In order: Income, Occupancy, Longitude, Latitude, Age, Rooms, Population, Bedrooms.

(c) Comments on the dependence of the response on the most important variables (you may want to consider partial dependence plots (plot) on single and pairs of variables, etc.).

For potential interactions, we only considered Income, Occupancy, Longitude, and Latitude. Occupancy has no noticable interactions with any of the other 3 big variables. However, all 3 other important variables exhibit a good deal of interaction with one another as shown by the partial dependence plots.

Problem 8:

(15) Regression: Marketing data.The data setage_stats315B.csv was already usedin Homework 1. Reviewage_stats315B.txtfor the information about order of attributes etc.

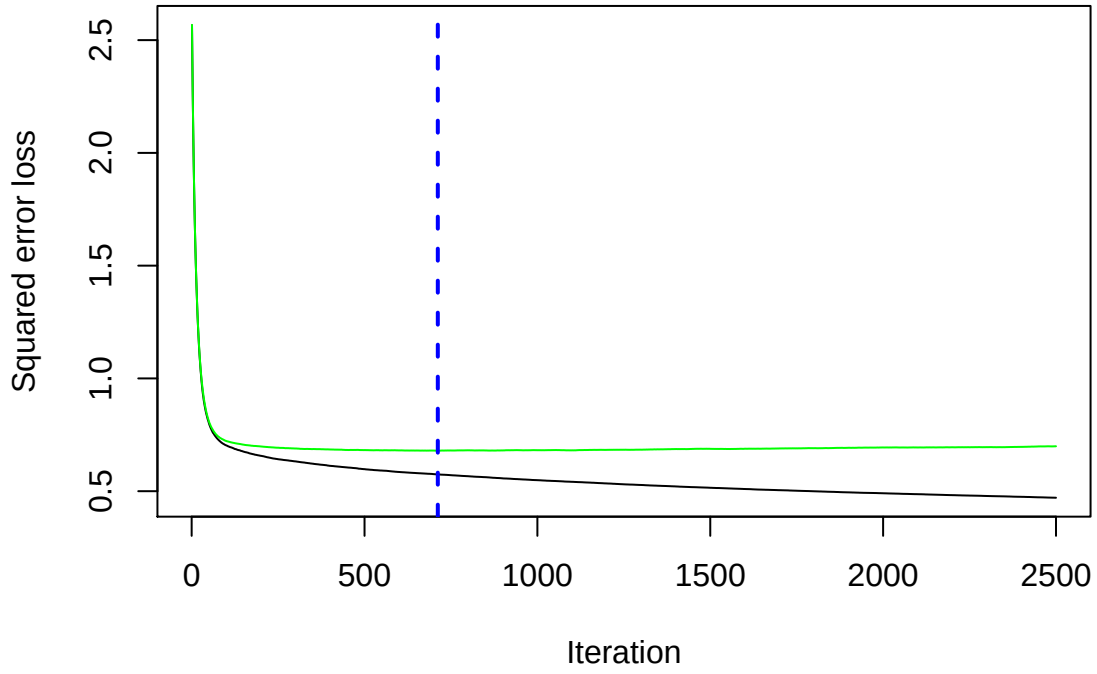
```
age <- read.csv("age_stats315B.csv", header=T)
factor_columns <- c(2,3,4,5,9,12,13,14)
ordered_columns <- c(1,6,7,8,10,11)
age[, factor_columns] <- lapply(age[factor_columns], as.factor)
age[, ordered_columns] <- lapply(age[ordered_columns], as.ordered)

train_ind <- sample(1:nrow(age), size = nrow(age) * .8)
test_ind <- seq(1,nrow(age))[!(seq(1,nrow(age)) %in% train_ind)]

gbm.age <- gbm(age ~ ., data=age[train_ind,], interaction.depth=4, shrinkage = 0.05, n.trees=2500, bag.
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        2.5657             nan     0.0500    0.1579
##      2        2.4147             nan     0.0500    0.1491
##      3        2.2788             nan     0.0500    0.1336
##      4        2.1555             nan     0.0500    0.1156
##      5        2.0448             nan     0.0500    0.1101
##      6        1.9445             nan     0.0500    0.0984
##      7        1.8539             nan     0.0500    0.0908
##      8        1.7702             nan     0.0500    0.0822
##      9        1.6953             nan     0.0500    0.0759
##     10        1.6274             nan     0.0500    0.0698
##     20        1.1805             nan     0.0500    0.0290
##     40        0.8595             nan     0.0500    0.0072
##     60        0.7636             nan     0.0500    0.0034
##     80        0.7240             nan     0.0500    0.0009
##    100        0.7033             nan     0.0500   -0.0001
##    120        0.6909             nan     0.0500    0.0005
##    140        0.6807             nan     0.0500    0.0001
##    160        0.6716             nan     0.0500    0.0003
##    180        0.6635             nan     0.0500   -0.0000
##    200        0.6569             nan     0.0500   -0.0001
##    220        0.6502             nan     0.0500   -0.0001
##    240        0.6443             nan     0.0500   -0.0002
##    260        0.6399             nan     0.0500    0.0000
##    280        0.6362             nan     0.0500   -0.0001
##    300        0.6322             nan     0.0500   -0.0002
##    320        0.6280             nan     0.0500   -0.0001
##    340        0.6240             nan     0.0500   -0.0000
##    360        0.6200             nan     0.0500   -0.0001
##    380        0.6167             nan     0.0500   -0.0002
##    400        0.6124             nan     0.0500   -0.0002
##    420        0.6096             nan     0.0500   -0.0001
##    440        0.6067             nan     0.0500   -0.0001
##    460        0.6040             nan     0.0500   -0.0001
##    480        0.6011             nan     0.0500   -0.0001
##    500        0.5977             nan     0.0500   -0.0003
##    520        0.5948             nan     0.0500   -0.0002
##    540        0.5924             nan     0.0500   -0.0001
##    560        0.5907             nan     0.0500   -0.0002
##    580        0.5876             nan     0.0500   -0.0002
##    600        0.5852             nan     0.0500   -0.0001
##    620        0.5832             nan     0.0500   -0.0002
##    640        0.5811             nan     0.0500   -0.0000
##    660        0.5794             nan     0.0500   -0.0002
##    680        0.5777             nan     0.0500   -0.0003
##    700        0.5757             nan     0.0500   -0.0001
##    720        0.5736             nan     0.0500   -0.0002
##    740        0.5720             nan     0.0500    0.0001
##    760        0.5701             nan     0.0500   -0.0002
##    780        0.5680             nan     0.0500   -0.0001
##    800        0.5664             nan     0.0500   -0.0002
##    820        0.5643             nan     0.0500   -0.0003
##    840        0.5628             nan     0.0500   -0.0001
##    860        0.5610             nan     0.0500   -0.0002
```

```
##    880        0.5589            nan     0.0500   -0.0001
##    900        0.5568            nan     0.0500    0.0001
##    920        0.5555            nan     0.0500   -0.0001
##    940        0.5536            nan     0.0500   -0.0002
##    960        0.5520            nan     0.0500   -0.0001
##    980        0.5504            nan     0.0500   -0.0002
##   1000        0.5489            nan     0.0500   -0.0001
##   1020        0.5473            nan     0.0500   -0.0002
##   1040        0.5458            nan     0.0500   -0.0001
##   1060        0.5446            nan     0.0500   -0.0002
##   1080        0.5428            nan     0.0500   -0.0002
##   1100        0.5416            nan     0.0500   -0.0001
##   1120        0.5403            nan     0.0500   -0.0001
##   1140        0.5388            nan     0.0500   -0.0001
##   1160        0.5375            nan     0.0500   -0.0001
##   1180        0.5359            nan     0.0500   -0.0002
##   1200        0.5345            nan     0.0500   -0.0002
##   1220        0.5329            nan     0.0500   -0.0001
##   1240        0.5313            nan     0.0500   -0.0002
##   1260        0.5299            nan     0.0500   -0.0003
##   1280        0.5288            nan     0.0500   -0.0002
##   1300        0.5276            nan     0.0500   -0.0002
##   1320        0.5261            nan     0.0500   -0.0002
##   1340        0.5248            nan     0.0500   -0.0002
##   1360        0.5235            nan     0.0500   -0.0003
##   1380        0.5225            nan     0.0500   -0.0002
##   1400        0.5212            nan     0.0500   -0.0001
##   1420        0.5197            nan     0.0500   -0.0002
##   1440        0.5185            nan     0.0500   -0.0001
##   1460        0.5175            nan     0.0500   -0.0002
##   1480        0.5164            nan     0.0500   -0.0003
##   1500        0.5152            nan     0.0500   -0.0000
##   1520        0.5141            nan     0.0500   -0.0005
##   1540        0.5131            nan     0.0500   -0.0001
##   1560        0.5120            nan     0.0500   -0.0002
##   1580        0.5109            nan     0.0500   -0.0002
##   1600        0.5097            nan     0.0500   -0.0002
##   1620        0.5087            nan     0.0500   -0.0001
##   1640        0.5073            nan     0.0500   -0.0002
##   1660        0.5063            nan     0.0500   -0.0001
##   1680        0.5055            nan     0.0500   -0.0002
##   1700        0.5045            nan     0.0500   -0.0002
##   1720        0.5038            nan     0.0500   -0.0003
##   1740        0.5026            nan     0.0500   -0.0001
##   1760        0.5016            nan     0.0500   -0.0001
##   1780        0.5008            nan     0.0500   -0.0003
##   1800        0.4997            nan     0.0500   -0.0001
##   1820        0.4988            nan     0.0500   -0.0002
##   1840        0.4977            nan     0.0500   -0.0002
##   1860        0.4968            nan     0.0500   -0.0002
##   1880        0.4956            nan     0.0500   -0.0002
##   1900        0.4949            nan     0.0500   -0.0001
##   1920        0.4939            nan     0.0500   -0.0002
##   1940        0.4929            nan     0.0500   -0.0001
```

```
##   1960        0.4923             nan     0.0500   -0.0002
##   1980        0.4918             nan     0.0500   -0.0003
##   2000        0.4909             nan     0.0500   -0.0002
##   2020        0.4899             nan     0.0500   -0.0002
##   2040        0.4892             nan     0.0500   -0.0001
##   2060        0.4882             nan     0.0500   -0.0003
##   2080        0.4875             nan     0.0500   -0.0001
##   2100        0.4868             nan     0.0500   -0.0001
##   2120        0.4861             nan     0.0500   -0.0002
##   2140        0.4850             nan     0.0500   -0.0003
##   2160        0.4842             nan     0.0500   -0.0001
##   2180        0.4833             nan     0.0500   -0.0001
##   2200        0.4825             nan     0.0500   -0.0002
##   2220        0.4817             nan     0.0500   -0.0003
##   2240        0.4809             nan     0.0500   -0.0001
##   2260        0.4803             nan     0.0500   -0.0003
##   2280        0.4797             nan     0.0500   -0.0001
##   2300        0.4787             nan     0.0500   -0.0001
##   2320        0.4780             nan     0.0500   -0.0002
##   2340        0.4774             nan     0.0500   -0.0001
##   2360        0.4767             nan     0.0500   -0.0001
##   2380        0.4757             nan     0.0500   -0.0002
##   2400        0.4751             nan     0.0500   -0.0002
##   2420        0.4744             nan     0.0500   -0.0002
##   2440        0.4734             nan     0.0500   -0.0001
##   2460        0.4727             nan     0.0500   -0.0002
##   2480        0.4717             nan     0.0500   -0.0001
##   2500        0.4710             nan     0.0500   -0.0001
```

```
gbm.perf(gbm.age, method="cv")
```




```
## [1] 712
```

```
gbm.predict <- predict(gbm.age, age[test_ind,-1])
```

```
## Using 712 trees...
```

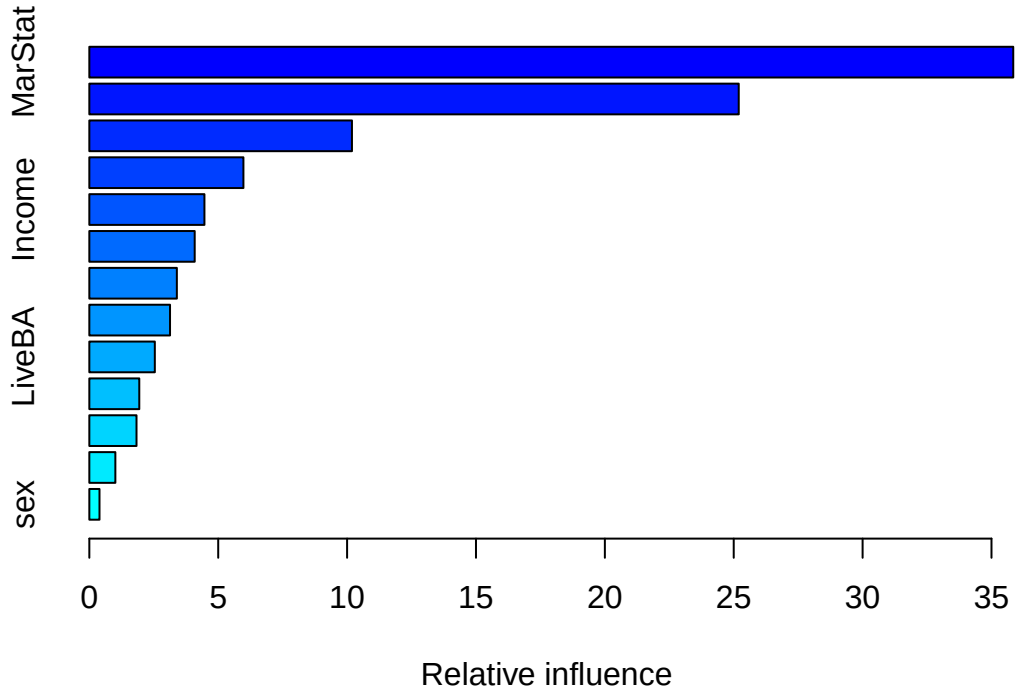
```
(acc <- sum(round(gbm.predict) == age[test_ind,]$age)/length(test_ind) )
```

```
## [1] 0.5287026
```

```
model <- rpart(age ~ ., data = age[train_ind,])
rpart.pred <- predict(model, age[test_ind,-1])
(acc_rpart <- sum(round(rpart.pred) == age[test_ind,]$age)/length(test_ind) )
```

```
## [1] 0.08495982
```

```
summary(gbm.age)
```




```
##                 var    rel.inf
## MarStat     MarStat 35.8441726
## Occup         Occup 25.1944675
## HouseStat HouseStat 10.1891681
## Edu             Edu  5.9776798
## Income       Income  4.4651435
## Persons     Persons  4.0877845
## Under18     Under18  3.3958716
## Ethnic       Ethnic  3.1311108
## LiveBA       LiveBA  2.5413008
## TypeHome   TypeHome  1.9383147
## DualInc     DualInc  1.8305235
## Lang           Lang  1.0100524
## sex             sex  0.3944103
```

(a) Fit a gbm model for predicting age form the other demographic attributes and compare the accuracy with the accuracy of your best single tree from Homework 1.

The accuracy is much higher using GBM (54% as compared to 9.7% of getting the exact age class right)

(b) Identify the most important variables.

The most important are in order: MarStat, Occup, HouseStat, Edu, Persons, Ethnic, Income, Under18, LiveBA, DualInc, TypeHome, Lang, Sex. Marital status and occupation are clearly the most important variables.

Problem 9.

(15) Multiclass classification: marketing data.The data set occup_stats315B.csvcomes from the same marketing database used in Homework 1. The description of the attributes can be found in occup_stats315B.txt. The goal in this problem is to fit a gbm model to predict the type of occupation from the 13 other demographic variables.

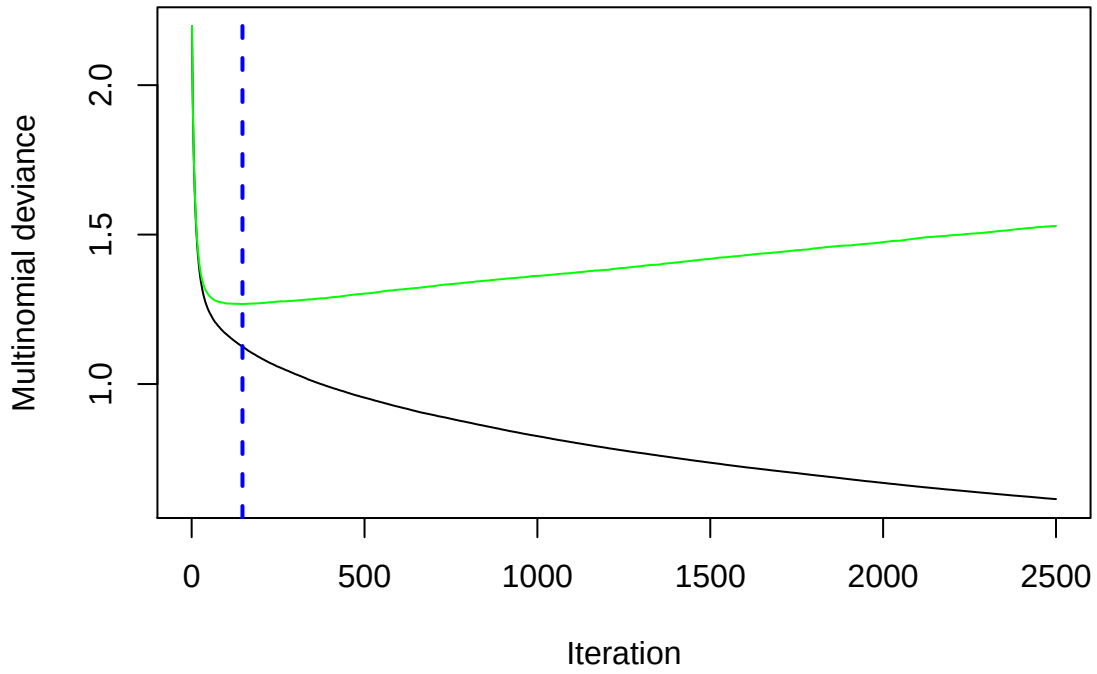
```
occup <- read.csv("occup_stats315B.csv", header=F)
colnames(occup) <- c("Occup", "TypeHome", "sex", "MarStat", "age", "Edu", "Income", "LiveBA", "DualInc"
factor_columns <- c(1,2,3,4,9,12,13,14)
ordered_columns <- c(5,6,7,8,10,11)
occup[, factor_columns] <- lapply(occup[factor_columns], as.factor)
occup[, ordered_columns] <- lapply(occup[ordered_columns], as.ordered)

train_ind <- sample(1:nrow(occup), size = nrow(occup) * .8)
test_ind <- seq(1,nrow(occup))[!(seq(1,nrow(occup)) %in% train_ind)]

gbm.occup <- gbm(Occup ~ ., data= occup[train_ind,], interaction.depth=4, shrinkage = 0.05, n.trees=250
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        2.1972             nan     0.0500    0.2466
##      2        2.0661             nan     0.0500    0.1838
##      3        1.9686             nan     0.0500    0.1447
##      4        1.8898             nan     0.0500    0.1192
##      5        1.8234             nan     0.0500    0.1035
##      6        1.7678             nan     0.0500    0.0902
##      7        1.7188             nan     0.0500    0.0757
##      8        1.6775             nan     0.0500    0.0694
##      9        1.6400             nan     0.0500    0.0613
##     10        1.6060             nan     0.0500    0.0544
##     20        1.4088             nan     0.0500    0.0207
##     40        1.2735             nan     0.0500    0.0033
##     60        1.2217             nan     0.0500    0.0008
##     80        1.1906             nan     0.0500   -0.0005
##    100        1.1672             nan     0.0500   -0.0012
##    120        1.1476             nan     0.0500   -0.0012
##    140        1.1306             nan     0.0500   -0.0012
##    160        1.1147             nan     0.0500   -0.0012
##    180        1.1003             nan     0.0500   -0.0016
##    200        1.0870             nan     0.0500   -0.0014
##    220        1.0746             nan     0.0500   -0.0012
##    240        1.0634             nan     0.0500   -0.0014
##    260        1.0530             nan     0.0500   -0.0018
##    280        1.0435             nan     0.0500   -0.0016
##    300        1.0332             nan     0.0500   -0.0018
##    320        1.0240             nan     0.0500   -0.0013
##    340        1.0141             nan     0.0500   -0.0017
##    360        1.0056             nan     0.0500   -0.0016
```

```
##    380        0.9977            nan     0.0500   -0.0016
##    400        0.9897            nan     0.0500   -0.0018
##    420        0.9825            nan     0.0500   -0.0018
##    440        0.9753            nan     0.0500   -0.0012
##    460        0.9678            nan     0.0500   -0.0015
##    480        0.9609            nan     0.0500   -0.0017
##    500        0.9545            nan     0.0500   -0.0017
##    520        0.9483            nan     0.0500   -0.0016
##    540        0.9418            nan     0.0500   -0.0014
##    560        0.9355            nan     0.0500   -0.0019
##    580        0.9292            nan     0.0500   -0.0016
##    600        0.9235            nan     0.0500   -0.0015
##    620        0.9179            nan     0.0500   -0.0016
##    640        0.9117            nan     0.0500   -0.0013
##    660        0.9059            nan     0.0500   -0.0012
##    680        0.9009            nan     0.0500   -0.0016
##    700        0.8962            nan     0.0500   -0.0014
##    720        0.8910            nan     0.0500   -0.0020
##    740        0.8864            nan     0.0500   -0.0013
##    760        0.8811            nan     0.0500   -0.0018
##    780        0.8761            nan     0.0500   -0.0018
##    800        0.8716            nan     0.0500   -0.0017
##    820        0.8666            nan     0.0500   -0.0016
##    840        0.8619            nan     0.0500   -0.0015
##    860        0.8571            nan     0.0500   -0.0014
##    880        0.8524            nan     0.0500   -0.0014
##    900        0.8475            nan     0.0500   -0.0017
##    920        0.8428            nan     0.0500   -0.0011
##    940        0.8385            nan     0.0500   -0.0014
##    960        0.8338            nan     0.0500   -0.0020
##    980        0.8296            nan     0.0500   -0.0017
##   1000        0.8258            nan     0.0500   -0.0015
##   1020        0.8217            nan     0.0500   -0.0015
##   1040        0.8174            nan     0.0500   -0.0013
##   1060        0.8131            nan     0.0500   -0.0016
##   1080        0.8093            nan     0.0500   -0.0017
##   1100        0.8053            nan     0.0500   -0.0016
##   1120        0.8014            nan     0.0500   -0.0015
##   1140        0.7976            nan     0.0500   -0.0016
##   1160        0.7937            nan     0.0500   -0.0017
##   1180        0.7901            nan     0.0500   -0.0016
##   1200        0.7865            nan     0.0500   -0.0019
##   1220        0.7827            nan     0.0500   -0.0016
##   1240        0.7791            nan     0.0500   -0.0016
##   1260        0.7756            nan     0.0500   -0.0016
##   1280        0.7724            nan     0.0500   -0.0016
##   1300        0.7690            nan     0.0500   -0.0015
##   1320        0.7658            nan     0.0500   -0.0015
##   1340        0.7625            nan     0.0500   -0.0012
##   1360        0.7591            nan     0.0500   -0.0014
##   1380        0.7559            nan     0.0500   -0.0013
##   1400        0.7526            nan     0.0500   -0.0018
##   1420        0.7494            nan     0.0500   -0.0015
##   1440        0.7462            nan     0.0500   -0.0013
```

```
##   1460        0.7430            nan     0.0500   -0.0013
##   1480        0.7399            nan     0.0500   -0.0013
##   1500        0.7369            nan     0.0500   -0.0018
##   1520        0.7339            nan     0.0500   -0.0013
##   1540        0.7308            nan     0.0500   -0.0013
##   1560        0.7277            nan     0.0500   -0.0012
##   1580        0.7248            nan     0.0500   -0.0014
##   1600        0.7219            nan     0.0500   -0.0014
##   1620        0.7192            nan     0.0500   -0.0015
##   1640        0.7165            nan     0.0500   -0.0014
##   1660        0.7138            nan     0.0500   -0.0014
##   1680        0.7110            nan     0.0500   -0.0014
##   1700        0.7085            nan     0.0500   -0.0013
##   1720        0.7058            nan     0.0500   -0.0015
##   1740        0.7033            nan     0.0500   -0.0015
##   1760        0.7006            nan     0.0500   -0.0014
##   1780        0.6979            nan     0.0500   -0.0019
##   1800        0.6951            nan     0.0500   -0.0013
##   1820        0.6926            nan     0.0500   -0.0013
##   1840        0.6899            nan     0.0500   -0.0012
##   1860        0.6875            nan     0.0500   -0.0018
##   1880        0.6846            nan     0.0500   -0.0014
##   1900        0.6819            nan     0.0500   -0.0019
##   1920        0.6793            nan     0.0500   -0.0014
##   1940        0.6766            nan     0.0500   -0.0016
##   1960        0.6742            nan     0.0500   -0.0010
##   1980        0.6716            nan     0.0500   -0.0014
##   2000        0.6692            nan     0.0500   -0.0014
##   2020        0.6666            nan     0.0500   -0.0015
##   2040        0.6643            nan     0.0500   -0.0017
##   2060        0.6617            nan     0.0500   -0.0012
##   2080        0.6593            nan     0.0500   -0.0014
##   2100        0.6569            nan     0.0500   -0.0017
##   2120        0.6546            nan     0.0500   -0.0016
##   2140        0.6524            nan     0.0500   -0.0013
##   2160        0.6502            nan     0.0500   -0.0013
##   2180        0.6478            nan     0.0500   -0.0015
##   2200        0.6459            nan     0.0500   -0.0013
##   2220        0.6437            nan     0.0500   -0.0015
##   2240        0.6416            nan     0.0500   -0.0014
##   2260        0.6394            nan     0.0500   -0.0014
##   2280        0.6372            nan     0.0500   -0.0014
##   2300        0.6351            nan     0.0500   -0.0014
##   2320        0.6329            nan     0.0500   -0.0015
##   2340        0.6309            nan     0.0500   -0.0015
##   2360        0.6288            nan     0.0500   -0.0014
##   2380        0.6267            nan     0.0500   -0.0011
##   2400        0.6248            nan     0.0500   -0.0014
##   2420        0.6230            nan     0.0500   -0.0016
##   2440        0.6209            nan     0.0500   -0.0013
##   2460        0.6188            nan     0.0500   -0.0014
##   2480        0.6168            nan     0.0500   -0.0018
##   2500        0.6150            nan     0.0500   -0.0015
```

```
best_iter = gbm.perf(gbm.occup, method="cv")
```




```
gbm.predict <- predict(gbm.occup, occup[test_ind,-1], type = "response", n.trees = best_iter)
predict_probs <- matrix(unlist(gbm.predict), ncol = 9, byrow = F) #n by classes

class_preds <- apply(predict_probs, 1, function(probs) return(which(probs == max(probs))))
true_class <- occup[test_ind, 1]


misclassification.error <- sum(class_preds != true_class) / length(true_class) * 100
misclass_perclass <- data.frame(1:9)
misclass_perclass$error <- sapply(1:9, function(i) return( sum(class_preds[true_class == i] != i) / sum
```

(a) Report the test set misclassification error for gbm on the data set, and also the misclassification error for each class.

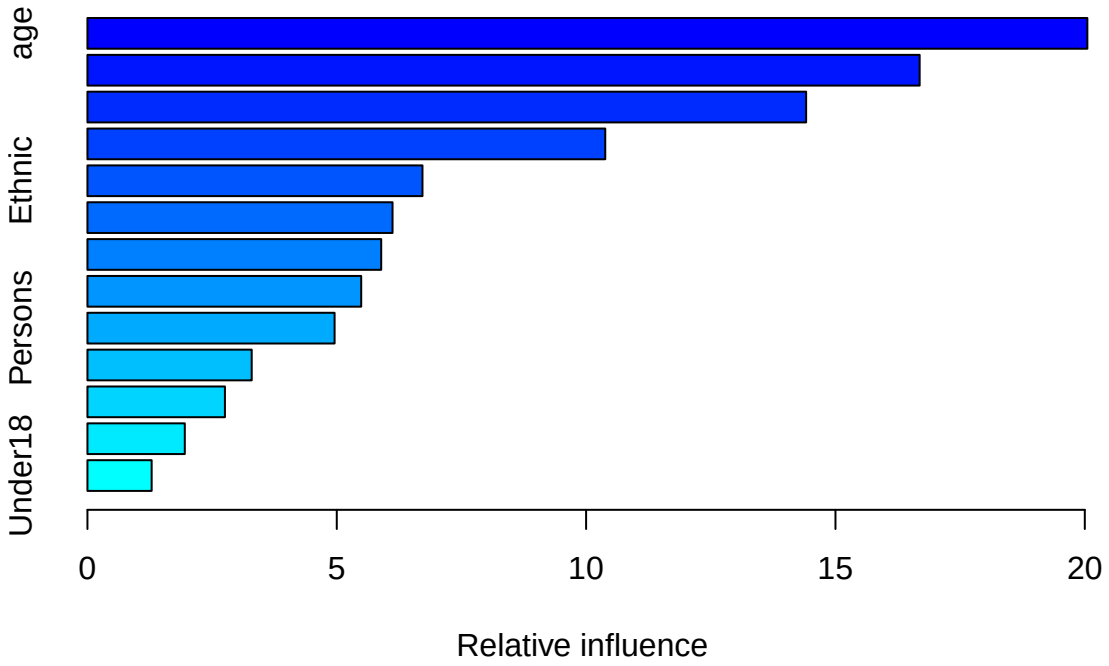
Misclassification error: 44.0180587

```
print(misclass_perclass)
```

```
##   X1.9    error
## 1    1 19.96234
## 2    2 96.08939
## 3    3 68.75000
## 4    4 75.57604
## 5    5 41.05960
## 6    6 21.61290
## 7    7 74.41860
## 8    8 18.18182
## 9    9 83.07692
```

(b) Identify the most important variables.

```
imp <- summary(gbm.occup)
```




```
print(imp[1:5, ])
```

```
##                 var   rel.inf
## age             age 20.051401
## Income       Income 16.688423
## Edu             Edu 14.411872
## HouseStat HouseStat 10.383548
## Ethnic       Ethnic  6.718745
```