STSCI4740 Final Group Project

Daisy Cho (bsc223), Diana DeFilippis (dnd34), Andrew Stephens (ahs295)

Introduction

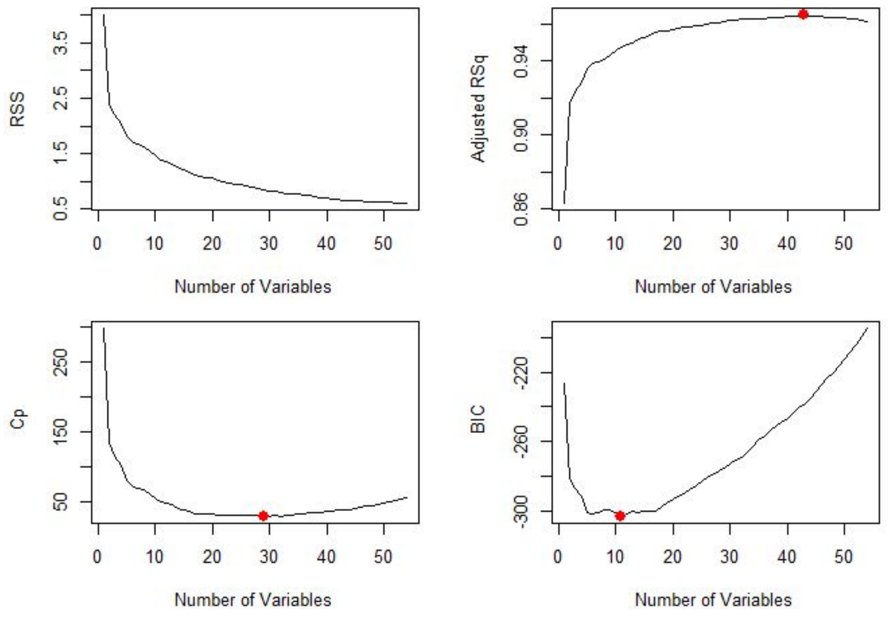
Understanding the factors that lead to divorce can provide useful information to those who seek to maintain a successful marriage. Survey responses of 170 individuals were analyzed by constructing several predictive machine learning models and examining the importance of their predictors. Participants filled out a survey of 54 questions on a scale of 0 (least agree) to 4 (most agree). The survey included questions that could indicate relationship health such as "My spouse and I have similar values in trust". Using these responses we sought to predict a response of married (class=0) or divorced (class=1), as well as understand the most important variables used in our predictions.

Identifying Important Features

Forward Subset Selection:

First we checked how many variables would create the smallest test error using cross validation. Cross-validating for all 54 variables, we found that only 6 variables have the smallest error. Based on cross-validation and forward stepwise selection, Attributes 6, 13, 15, 19, 20 and 40 are the most important variables.

Additionally, we can also check with RSS, adjusted R-squared, Cp and BIC.



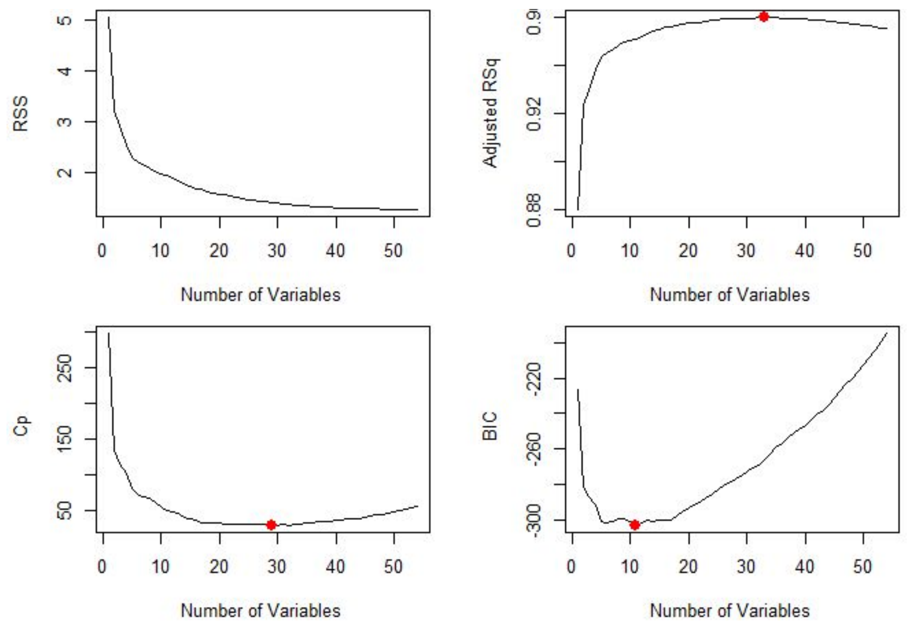
We can see RSS continuously decreases as variables are added to the model. However, adjusted R-squared is highest with 43 important variables, Cp states is second highest with 29 variables, and BIC is lowest with 11 important variables. In the corresponding plots above, the red point indicates the number of variables with the best measure of error. We did not indicate the lowest RSS error because it continues to decrease until all the variables are included.

Since BIC has the fewest important variables, we identified these 11 variables as Attributes 6, 13, 15, 17, 19, 20, 25, 26, 27, 40, and 49. Six of these we have already listed previously as important variables found from cross-validation.

__Backward Subset Selection:__

In addition to forward stepwise selection, we tested backward stepwise selection. From the cross-validation on backward stepwise selection, 32 variables were found to be important which is not listed since it is most of the predictor variables.

As we did previously, we plotted the RSS, Adjusted R-squared, Cp and BIC and indicated the best corresponding test error.



Adjusted R-squared is highest with 33 important variables, Cp is lowest with 21 variables, and BIC is lowest with 8 variables. Similarly to the forward stepwise selection, BIC had the smallest number of variables which were found to be Attributes 6, 17, 25, 26, 40, 46, 49, and 52.

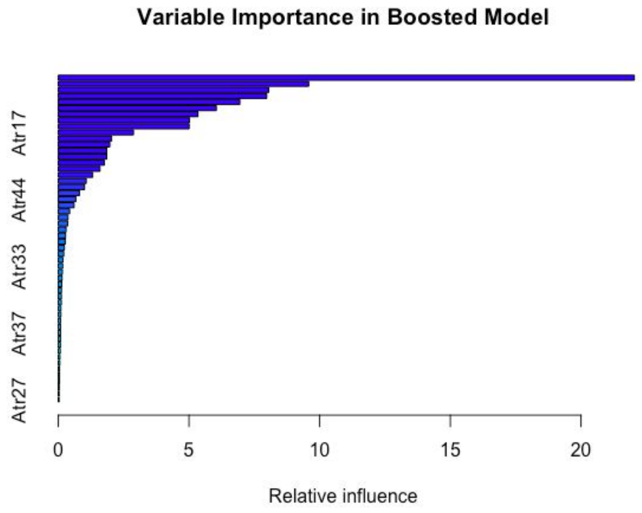
Predictive Modeling

Decision Tree:

Decision trees are a greedy, algorithmic method that uses information theory for decision making. Greedy algorithms do not guarantee a globally optimum solution, however, the local approximations generally produce good approximations. A basic decision tree was used without pruning to estimate divorce. Despite possibly overfitting the training data without pruning, we still achieved favorable results. The training MSE obtained was 0.00 and the test MSE was 0.0352. Additionally, the summary output tells us which variables were used in decision making. Attributes 18 and 20 were indicated as the most important variables here.

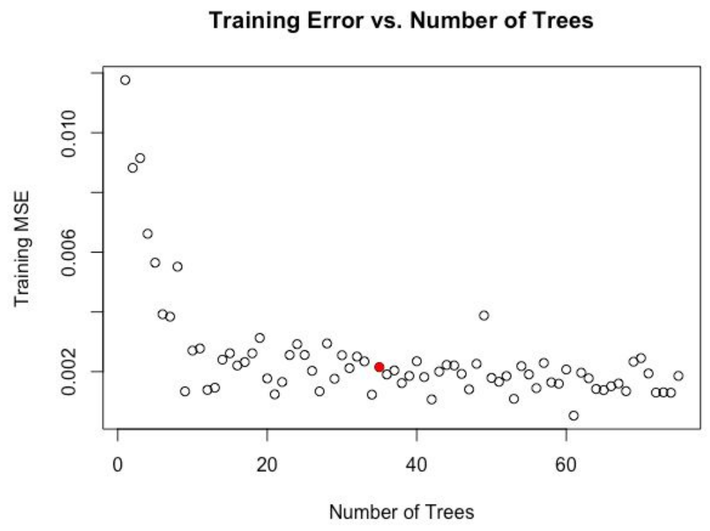
Boosting:

Boosting is an ensemble method commonly used to improve model performance that can be applied to decision trees. The training data is repeatedly sampled with replacement and is used to create multiple models. Using boosting on the divorce dataset produced a training MSE of 0.0019 and a test MSE of 0.0316. These results are similar to the original decision tree results but have the added benefit of showing variable importance. Boosting identified more important variables than the original decision tree; the top 5 most important variables in this model are Attributes 40, 19, 20, 26, and 18.

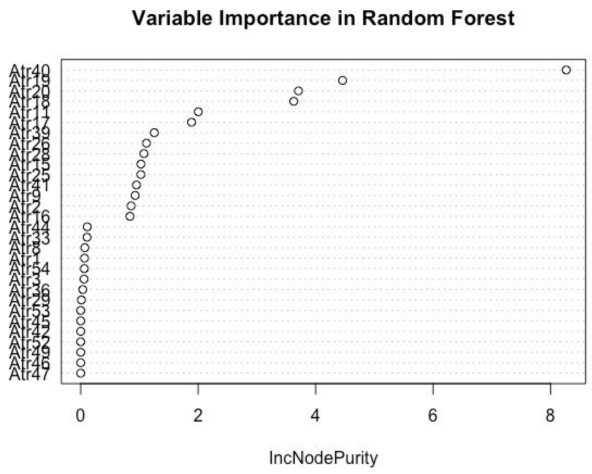


Random Forest:

Random forests are an additional ensemble method that can be used to improve prediction. Random forests take random subsets of the training data by sampling the predictor space as opposed to the samples. To select our model parameters, a function of the number of trees was created to examine the training MSE. Using the plot below we selected that roughly 35 trees would be sufficient for our model. At 35 trees the training error has decreased and we are still able to maintain computational efficiency.



The training MSE using a random forest with 35 trees is 0.0019 and the test MSE is 0.0316. The test MSE has improved again over both decision trees and bosting. We are additionally able to investigate variable importance in a random forest model. This is a measure of the mean decrease in the Gini Index and is illustrated below.



The top 5 predictors in this model are Attributes 40, 19, 20, 18, and 11. Notice that 4 of these 5 predictors match the predictors identified in the boosted model.

<u>Linear Discriminant Analysis:</u>

Linear Discriminant Analysis (LDA) uses discriminant functions to classify different observations based on Bayes Theorem.After splitting the data into training and testing datasets, the training data is used to build the model by creating the discriminant functions used to assign observations to a specific class. Discriminant functions are used to determine which posterior probability is highest, so the random observation is assigned to the class whose discriminant function is the largest. For this approach, the divorce dataset was split randomly and evenly into the test and training datasets so that there were 85 observations in each. Performing LDA on the training dataset and resulted in the following table:

Prediction/Training Class	Not Divorced	Divorced
Not Divorced	47	0
Divorced	0	38

This table shows that the model classified the observations in the training dataset perfectly, with zero false negative or false positive classifications, resulting in a training error rate of precisely 0. However, the training error rate is an unreliable measurement of how well a model actually performs. The high number of variables in the model likely overfits the data, so the training error rate cannot be trusted. Obtaining the test error rate is much more important, as it is a measure of how well a model performs. The table below shows the classification of the 85 observations in the test data based on the LDA model:

Prediction/Training Class	Not Divorced	Divorced
Not Divorced	39	6
Divorced	0	40

Here we see that there are misclassifications of some of the observations in the test data. The test error rate is (0+6)/85 = 0.07058824, a relatively low rate. This shows that the LDA model is fairly good at determining whether a person is divorced based on the 54 variables used to create the discriminant functions.

Logistic Regression:

Logistic regression is a statistical method that is often favored over linear regression when modeling probabilities because the function will always result in a value between 0 and 1. Once again, the divorce dataset is split randomly and evenly into a training dataset and a test dataset. Maximum likelihood is used to estimate the parameters based on the training data. Creating the model based on the training data results in the following confusion matrix:

Prediction/Training Class	Not Divorced	Divorced
Not Divorced	47	0

Divorced	0	38

Similar to LDA, the logistic regression model results in a training error rate of 0, with no false positives or false negatives. The extremely low training error rate is likely due to the high number of variables causing the model to overfit the training data. Therefore, turning to the test dataset, the resulting confusion matrix based on the logistic regression is as follows:

Prediction/Training Class	Not Divorced	Divorced
Not Divorced	38	7
Divorced	1	39

The test error rate calculated from the above table is (1+7)/85=0.09411765, which indicates a fairly good model with under 10% of observations being misclassified. Interestly and similar to LDA, the false positive rate contributes the most to the overall test error rate. This means that it's more likely that a divorced person will be classified as not divorced than a non-divorced person being classified as divorced. Ultimately, logistic regression does a decent job modeling the data, but there still are several methods that do a better job.

Linear Regression:

As a preliminary method, we performed a linear regression using all the variables from the dataset. The training data consisted of 70% of the data and the test data consisted of 30% of the data since the *n*, or the number of observations was relatively small in comparison to the *p*, or the predictor variables. The training MSE rate is 0.006924878 or 0.692%, and the test MSE rate is 0.05544854 or 5.545%. Surprisingly, since we were not sure if there was a linear interaction between Class and the predictor variables, the test error rate is small.

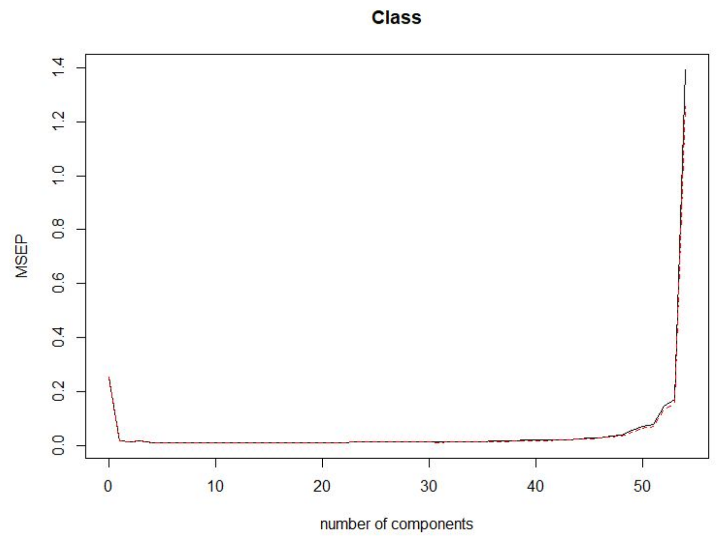
Ridge and Lasso Regression:

For both ridge and lasso regression, we tried two different methods. The first made 50% of the dataset the training data and the other made 70% of the dataset as the training data. For both methods, we ran a for loop of 50 iterations setting different seeds each time and then averaged out the MSE rates from each iteration. In order to find the best lambda for each regression, we used cross-validation. For ridge regression with the 50% training data, the average training MSE was 0.009943694 and the average test MSE was 0.02088043. With the 70% training data, the average training MSE was 0.007565424 and the average test MSE was 0.03358272.

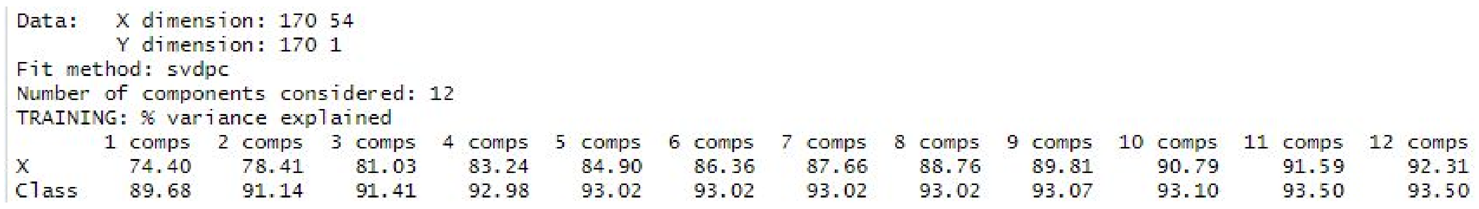
For lasso regression with the 50% training data, the average training MSE was 0.009478118 and the average test MSE was 0.01960212. With the 70% training data, the average training MSE was 0.007257026 and the average test MSE was 0.03360199. The training MSE decreases with the increase of observations in the training data, but the test MSE increases which may be true because of overfitting with all the variables to account for the variability with the larger training data.

<u>Principal Components Regression:</u>

For PCR we first standardized each predictor by setting "scale=TRUE" so that the scale on which each variable is measured will not affect the regression. We computed the ten-fold cross-validation error for each value of *M*, the number of principal components used. We then plotted the cross-validation scores using a validation plot which plots the MSE.



In the divorce dataset the smallest cross-validation error occurs when $M = 12$. This suggests that a small number of components might suffice for this model. We then fit the prediction with $M = 12$ and compute the train MSE, 0.007035013, and the test MSE, 0.4243354. Finally, we set the PCR on the full data set using $M = 12$, the number of components identified by cross-validation. Below is the summary of the PCR fit on the full data set.



```
Data: 	X dimension: 170 54
	Y dimension: 170 1
Fit method: svdpc
Number of components considered: 12
TRAINING: % variance explained
       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
X        74.40    78.41    81.03    83.24    84.90    86.36    87.66    88.76    89.81     90.79     91.59     92.31
Class    89.68    91.14    91.41    92.98    93.02    93.02    93.02    93.02    93.07     93.10     93.50     93.50
```

Conclusion

Nearly all the methods explored throughout this project were able to effectively predict divorce. The success of these models is not surprising because the divcorce dataset was already cleaned and had a high number of predictors to train on. All the models achieved low training MSE's and were additionally effective at predicting the test data. The method with the highest error rate is principal components regression, which may be lower with different numbers of components, however,the training error rate is quite low. Our prediction results are illustrated in the table of MSE's below.

Model	**Training MSE**	**Test MSE**
Decision Tree	0	0.03529412
Decision Trees (Boosting)	0.001917931	0.03162841
Random Forest	0.001930372	0.03162545
Ridge Regression	0.009943694	0.02088043
Lasso Regression	0.009478118	0.01960212
Principal Components Regression	0.007035013	0.4243354
Linear Regression	0.006924878	0.05544854
LDA	0	0.07058824
Logistic Regression	0	0.09411765

From these values we can conclude that random forest, ridge regression, and lasso regression were the best machine learning methods for the divorce dataset. This is a reasonable conclusion as ensemble methods are known to improve prediction accuracy, and ridge and lasso reduce the risk of overfitting the training data. There is a possibility of reducing these errors even further with the inclusion of more training data. However, we chose to randomly split the data in half due to the limited number of observations.

We were additionally able to identify several key factors used in predicting divorce. There were multiple variables that appeared consistently in both the subset selections and the tree-based methods which we believe to be the most important. We identified these variables to be Attributes 18, 19, 20, 26, and 40. Upon examining the divorce predictors scale, our results indicate that comparable core values regarding the foundations of marriage, as well as good levels of trust and communication are the most important factors in the success of a marriage.

Codes

Forward Stepwise Selection

```
library(leaps)
regfit.fwd=regsubsets(Class~.,divorce_train,nvmax=54,method="forward")
reg.fwd.smry<-summary(regfit.fwd)
reg.fwd.smry
test.mat=model.matrix(Class~.,divorce_test)
val.errors=rep(NA,54)
for(i in 1:54){
  coefi=coef(regfit.fwd,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.errors[i]=mean((divorce_test$Class-pred)^2)
}
val.errors
which.min(val.errors)
coef(regfit.fwd,6)

par(mfrow =c(2 ,2) )
plot(reg.fwd.smry$rss , xlab =" Number of Variables ", ylab =" RSS ",
type ="l")
plot(reg.fwd.smry$adjr2 ,xlab =" Number of Variables ",
ylab =" Adjusted RSq ", type ="l")
which.max(reg.fwd.smry$adjr2)
points (43, reg.fwd.smry$adjr2 [43] , col =" red ", cex =2, pch =20)
plot(reg.fwd.smry$cp ,xlab =" Number of Variables ", ylab =" Cp",
type="l")
which.min(reg.fwd.smry$cp)
points (29, reg.fwd.smry$cp [29] , col =" red ", cex =2, pch =20)
plot(reg.fwd.smry$bic ,xlab =" Number of Variables ", ylab =" BIC",
type="l")
which.min(reg.fwd.smry$bic)
points (11, reg.fwd.smry$bic [11] , col =" red ", cex =2, pch =20)
coef(regfit.fwd,11)
```

Backward Stepwise Selection

```
set.seed(1)
regfit.bwd=regsubsets(Class~.,data,nvmax=54,method="backward")
reg.bwd.smry<-summary(regfit.bwd)
reg.bwd.smry
test.mat=model.matrix(Class~.,divorce_test) # create an X matrix of test data
val.errors=rep(NA,54)
for(i in 1:54){
  coefi=coef(regfit.bwd,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.errors[i]=mean((divorce_test$Class-pred)^2)
}
val.errors
which.min(val.errors)
coef(regfit.bwd,32)

par(mfrow =c(2 ,2) )
plot(reg.bwd.smry$rss , xlab =" Number of Variables ", ylab =" RSS ",
type ="l")
plot(reg.bwd.smry$adjr2 ,xlab =" Number of Variables ",
ylab =" Adjusted RSq ", type ="l")
which.max(reg.bwd.smry$adjr2)
points (33, reg.bwd.smry$adjr2 [33] , col =" red ", cex =2, pch =20)
plot(reg.bwd.smry$cp ,xlab =" Number of Variables ", ylab =" Cp",
type="l")
which.min(reg.bwd.smry$cp)
points (21, reg.bwd.smry$cp [21] , col =" red ", cex =2, pch =20)
plot(reg.bwd.smry$bic ,xlab =" Number of Variables ", ylab =" BIC",
type="l")
which.min(reg.bwd.smry$bic)
points (8, reg.bwd.smry$bic [8] , col =" red ", cex =2, pch =20)
coef(regfit.bwd,8)
```

Decision Tree

```
smp_size = 0.5*nrow(data) #Set sample size to 80% of data
train_ind = sample(seq_len(nrow(data)),size=smp_size)
datatrain=(data[train_ind,]) #Subset training data
datatest=(data[-train_ind,]) #Subset testing data

library(tree)
dt = tree(Class~., data = datatrain)
#Training MSE
pred = predict(dt, datatrain[,-55])
mean((datatrain$Class - pred)^2)
#Testing MSE
pred = predict(dt, datatest[,-55])
mean((datatest$Class - pred)^2)
summary(dt)
plot(dt)
```

Boosting

```
## Boosting
library(gbm)
boost=gbm(Class~., data=datatrain, distribution=
          "gaussian", n.trees = 5000, interaction.depth=4)
```

```
summary(boost, main="Variable Importance in Boosted Model", cex.lab = 0.5)
par(mfrow = c(1,2))
plot(boost ,i="Atr40", main = "Effects of Atr 40")
plot(boost ,i="Atr19", main = "Effects of Atr 19")

#Training MSE
pred = predict(boost, datatrain[,-55], n.trees=5000)
mean((datatrain$Class - pred)^2)
#Testing MSE
pred = predict(boost, datatest[,-55], n.trees=5000)
mean((datatest$Class - pred)^2)
```

Random Forest

```
library(randomForest)
rf.er = function(x){
  rfmod = randomForest(Class~., data = datatrain, importance=TRUE, ntree=x)
  pred = predict(rfmod, datatrain[,-55])
  err = mean((datatrain$Class - pred)^2)
  return (err)
}

errs = numeric()
for (i in 1:75){
  errs[i] = rf.er(i)
}
plot(errs, xlab="Number of Trees", ylab = "Training MSE", main = "Training Error vs. Number
of Trees")
points(35,errs[35], pch=16, col = "red")

#Training MSE
rf.mod = randomForest(Class~., data = datatrain, importance=TRUE, ntree=35)
pred = predict(rf.mod, datatrain[,-55])
mean((datatrain$Class - pred)^2)
#Testing MSE
pred = predict(rf.mod, datatest[,-55])
mean((datatest$Class - pred)^2)

#summary(rf.mod)
importance(rf.mod)
par(mfrow = c(1,1))
varImpPlot(rf.mod,type=2, main = "Variable Importance in Random Forest")
```

Linear Regression

```
set.seed(1)
```

```
smp_size = 0.7*nrow(data)
train_ind = sample(seq_len(nrow(data)),size=smp_size)
divorce_train=(data[train_ind,])
divorce_test=(data[-train_ind,])
lm.fit<-lm(Class~., data=divorce_train)
lm.fit
lm.pred.train<- predict(lm.fit,divorce_train)
mean((lm.pred.train-divorce_train$Class)^2)
lm.pred.test<-predict(lm.fit,divorce_test)
mean((lm.pred.test -divorce_test$Class)^2)
```

Lasso and Ridge Regression

```
library(glmnet)
grid=10^seq(10,-2,length=100)

smp_size = 0.5*nrow(data)
train_ind = sample(seq_len(nrow(data)),size=smp_size)
divorce_train=(data[train_ind,])
divorce_test=(data[-train_ind,])
X_train=model.matrix(Class~.,divorce_train)[,-1]
Y_train=divorce_train$Class
X_te = model.matrix(Class~.,divorce_test)[,-1]
Y_te=divorce_test$Class
ridge.5.train<- rep(0,50)
lasso.5.train<- rep(0,50)
ridge.5.test<- rep(0,50)
lasso.5.test<- rep(0,50)

for(i in  1:50){
set.seed(i)
cv.ridge = cv.glmnet(X_train, Y_train, alpha =0, lambda = grid)
cv.lasso = cv.glmnet(X_train, Y_train, alpha =1, lambda = grid)
best.ridge = cv.ridge$lambda.min
best.lasso = cv.lasso$lambda.min
pred.ridge.train = predict(cv.ridge, s = best.ridge, newx = X_train)
pred.lasso.train = predict(cv.lasso, s = best.lasso, newx = X_train)
ridge_error.train = mean((pred.ridge.train - Y_train)^2)
lasso_error.train = mean((pred.lasso.train - Y_train)^2)
ridge_error.train
lasso_error.train
ridge.5.train[i]<- ridge_error.train
lasso.5.train[i]<- lasso_error.train
pred.ridge.test = predict(cv.ridge, s = best.ridge, newx = X_te)
pred.lasso.test = predict(cv.lasso, s = best.lasso, newx = X_te)
ridge_error.test = mean((pred.ridge.test - Y_te)^2)
```

```
lasso_error.test = mean((pred.lasso.test - Y_te)^2)
ridge_error.test
lasso_error.test
ridge.5.test[i]<- ridge_error.test
lasso.5.test[i]<- lasso_error.test
}
ridge.5.train
lasso.5.train
mean(ridge.5.train)
mean(lasso.5.train)

ridge.5.test
lasso.5.test
mean(ridge.5.test)
mean(lasso.5.test)

smp_size = 0.7*nrow(data)
train_ind = sample(seq_len(nrow(data)),size=smp_size)
divorce_train=(data[train_ind,])
divorce_test=(data[-train_ind,])
X_train=model.matrix(Class~.,divorce_train)[,-1]
Y_train=divorce_train$Class
X_te = model.matrix(Class~.,divorce_test)[,-1]
Y_te=divorce_test$Class
grid=10^seq(10,-2,length=100)
ridge.7.train<- rep(0,50)
lasso.7.train<- rep(0,50)
ridge.7.test<- rep(0,50)
lasso.7.test<- rep(0,50)

for(i in  1:50){
set.seed(i)
cv.ridge = cv.glmnet(X_train, Y_train, alpha =0, lambda = grid)
cv.lasso = cv.glmnet(X_train, Y_train, alpha =1, lambda = grid)
best.ridge = cv.ridge$lambda.min
best.lasso = cv.lasso$lambda.min
pred.ridge.train = predict(cv.ridge, s = best.ridge, newx = X_train)
pred.lasso.train = predict(cv.lasso, s = best.lasso, newx = X_train)
ridge_error.train = mean((pred.ridge.train - Y_train)^2)
lasso_error.train = mean((pred.lasso.train - Y_train)^2)
ridge.7.train[i]<- ridge_error.train
lasso.7.train[i]<- lasso_error.train
pred.ridge.test = predict(cv.ridge, s = best.ridge, newx = X_te)
pred.lasso.test = predict(cv.lasso, s = best.lasso, newx = X_te)
ridge_error.test = mean((pred.ridge.test - Y_te)^2)
```

```
lasso_error.test = mean((pred.lasso.test - Y_te)^2)
ridge.7.test[i]<- ridge_error.test
lasso.7.test[i]<- lasso_error.test
}
ridge.7.train
lasso.7.train
mean(ridge.7.train)
mean(lasso.7.train)
ridge.7.test
lasso.7.test
mean(ridge.7.test)
mean(lasso.7.test)
```

Principal Components Regression

```
library(pls)
set.seed(1)
smp_size = 0.5*nrow(data)
train_ind = sample(seq_len(nrow(data)),size=smp_size)
divorce_train=(data[train_ind,])
divorce_test=(data[-train_ind,])
pcr.fit=pcr(Class~.,data=divorce_train,scale=TRUE,validation="CV")
par(mfrow =c(1 ,1) )
validationplot(pcr.fit,val.type="MSEP")
pcr.pred.train= predict(pcr.fit,data=divorce_train, ncomp=12)
mean((pcr.pred.train-divorce_train$Class)^2)
pcr.pred.test= predict(pcr.fit,data=divorce_test, ncomp=12)
mean((pcr.pred.test-divorce_test$Class)^2)
pcr.fit.full=pcr(Class~., data=data,scale=TRUE,ncomp=12)
summary(pcr.fit.full)
```

LDA

```
set.seed(1)
index <- sample(1:nrow(divorce), nrow(divorce) / 2)
train <- divorce[index, ]
test <- divorce[-index, ]
library(MASS)
(LDA <- lda(Class ~ ., data = train))
(lda_pred <- predict(LDA, test))
(lda_pred2 <- predict(LDA, train))
table(lda_pred2$class, train$Class)
(1-mean(lda_pred2$class == train$Class))
table(lda_pred$class, test$Class)
(1-mean(lda_pred$class == test$Class))
```

QDA (did not work-too computationally expensive)

```
set.seed(1)
(QDA <- qda(Class ~ ., data=divorce, subset=train))
qda_pred <- predict(QDA, test)
table(qda_pred$class, test$Class)
(1-mean(qda_pred$class == train$Class))
table(qda_pred$class, test$Class)
(1-mean(qda_pred$class == test$Class))
```

Logistic Regression

```
(log_reg <- glm(Class ~ ., data = train, family = binomial))
log_pred <- predict(log_reg, train, type = 'response')
lr_pred <- round(log_pred)
table(lr_pred, train$Class)
(1-mean(lr_pred == train$Class))
log_pred <- predict(log_reg, test, type = 'response')
lr_pred <- round(log_pred)
table(lr_pred, test$Class)
(1-mean(lr_pred == test$Class))
```