40.220 The Analytics Edge Final Revision

Note: This set of notes serves as a quick refresher for material from Week 8 onwards and is not intended to substitute the class material. Please read your notes as well :)

1. CART (Classification And Regression Trees)

Importing necessary libraries

In [ ]:

```
library(rpart)
library(rpart.plot)
library(caTools)
library(ROCR)
```

Load example dataset

In [3]:

```
supreme <- read.csv("Dataset/supreme.csv")
```

In [ ]:

```
str(supreme)
```

In [ ]:

```
summary(supreme)
```

Preprocessing of dataset. We want to predict whether Judge Stevens reverses (labelled as 1) or affirms (labelled as 0) the lower court direction.

In [8]:

```
stevens <- subset(supreme[,c("docket","term","stevdir","petit","respon","circuit
stevens$rev <- as.integer((stevens$stevdir==0&stevens$lctdir=="conser")|(stevens
```

Splitting the dataset into train and test

In [9]:

```
set.seed(1)
spl<-sample.split(stevens$rev, SplitRatio = 0.7)
train<-subset(stevens, spl==TRUE)
test<-subset(stevens, spl==FALSE)
```

Training (classification tree)

Training a classification tree with *rev* as the output and the other variables (i.e. *petit, respon, circuit, unconst, lctdir, issue*) as predictors

In [60]:

```
# use as.factor to change into class labels for classification problem
cart<-rpart(as.factor(rev)~petit+respon+circuit+unconst+lctdir+issue, data=trai
# OR
cart<-rpart(rev~petit+respon+circuit+unconst+lctdir+issue, data=train, method="
cart
```

```
n= 434

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 434 195 1 (0.44930876 0.55069124)
    2) lctdir=liberal 205  82 0 (0.60000000 0.40000000)
      4) circuit=10th,11th,1st,3rd,4th,5th,6th,7th,8th,DC 109  31 0
(0.71559633 0.28440367)
        8) respon=DEF,ER,IP,POL,STATE 36   3 0 (0.91666667 0.08333333)
*
        9) respon=BUSINESS,EE,INDIAN,OF,OTHER,US 73  28 0 (0.61643836
0.38356164)
         18) petit=ER,IP,OF,POL 19   2 0 (0.89473684 0.10526316) *
         19) petit=BUSINESS,CITY,OTHER,STATE,US 54  26 0 (0.51851852
0.48148148)
           38) issue=CP,FA 16   4 0 (0.75000000 0.25000000) *
           39) issue=CR,DP,ECN,FED,JUD,PRIV,UN 38  16 1 (0.42105263 0.
57894737)
             78) circuit=10th,3rd,4th,5th,8th 25  12 0 (0.52000000 0.4
8000000)
              156) respon=BUSINESS,OTHER 18   7 0 (0.61111111 0.388888
89) *
              157) respon=EE,INDIAN,OF 7   2 1 (0.28571429 0.71428571)
*
             79) circuit=11th,6th,7th 13   3 1 (0.23076923 0.76923077)
*
      5) circuit=2nd,9th,FED 96  45 1 (0.46875000 0.53125000)
       10) respon=EE,IP,OF,STATE,US 14   3 0 (0.78571429 0.21428571) *
       11) respon=BUSINESS,DEF,INDIAN,OTHER 82  34 1 (0.41463415 0.585
36585)
         22) issue=CP,FA,JUD 36  16 0 (0.55555556 0.44444444)
           44) petit=BUSINESS,CITY,OF,OTHER,STATE 25   8 0 (0.68000000
0.32000000) *
           45) petit=US 11   3 1 (0.27272727 0.72727273) *
         23) issue=CR,DP,ECN,FED,PRIV,TAX 46  14 1 (0.30434783 0.69565
217)
           46) circuit=FED 8   3 0 (0.62500000 0.37500000) *
           47) circuit=2nd,9th 38   9 1 (0.23684211 0.76315789) *
    3) lctdir=conser 229  72 1 (0.31441048 0.68558952)
      6) respon=BUSINESS,CITY,DEF,EE,IP,OF,OTHER,US 169  67 1 (0.39644
970 0.60355030)
       12) circuit=10th,11th,1st,3rd,7th,8th,9th,DC,FED 110  54 0 (0.5
0909091 0.49090909)
         24) issue=DP,FA,TAX 17   3 0 (0.82352941 0.17647059) *
         25) issue=AT,CP,CR,ECN,FED,JUD,PRIV,UN 93  42 1 (0.45161290
0.54838710)
           50) petit=INDIAN,IP,OF,STATE,US 13   3 0 (0.76923077 0.2307
6923) *
           51) petit=BUSINESS,CITY,DEF,EE,ER,OTHER,POL 80  32 1 (0.400
00000 0.60000000)
            102) issue=CP,FED,PRIV,UN 30  14 0 (0.53333333 0.46666667)
```

```
              204) circuit=3rd,7th,9th 15   5 0 (0.66666667 0.3333333
3) *
              205) circuit=10th,11th,1st,8th,DC 15   6 1 (0.40000000
0.60000000) *
            103) issue=AT,CR,ECN,JUD 50  16 1 (0.32000000 0.68000000)
*
       13) circuit=2nd,4th,5th,6th 59  11 1 (0.18644068 0.81355932) *
      7) respon=ER,INDIAN,POL,STATE 60   5 1 (0.08333333 0.91666667) *
```

Note on Classification Tree:

- Intuition of how the training works:
  1. Start with all observations in a single region
  2. Select predictor variable X_j and cutpoint in the variable S that maximizes the impurity reduction in the two child regions/buckets. Impurity can be measured using Gini, Entropy and Misclassification. The smaller the values, the more pure the buckets
  3. Split the parent region into child regions using the selected predictor variable X_j^+ and S^+
  4. Repeat step 2 and 3 until a stopping criterion is reached, which is the number of terminal nodes in the tree
- Example of calculating Gini for a split:

$$Gini\ index\ for\ mth\ region = \sum_{k=1}^{K} P_{mk}(1 - P_{mk})$$

where p represents the proportion of training obs in the mth region for the kth class

Suppose there are two classes: {0,1} and bucket 1 splits into bucket 2 and 3:

Bucket 1	**Class 0**	**Class 1**
Raw count	10	5

Bucket 2	**Class 0**	**Class 1**
Raw count	1	4

Bucket 3	**Class 0**	**Class 1**
Raw count	9	1

$$Gini\ for\ bucket\ 1 = \frac{10}{15} * \frac{5}{15} + \frac{5}{15} * \frac{10}{15} = \frac{4}{9}$$
$$Gini\ for\ bucket\ 2 = \frac{1}{5} * \frac{4}{5} + \frac{4}{5} * \frac{1}{5} = \frac{8}{25}$$
$$Gini\ for\ bucket\ 3 = \frac{9}{10} * \frac{1}{10} + \frac{1}{10} * \frac{9}{10} = \frac{9}{50}$$

- Example of calculating impurity reduction for a split using Gini

$$Impurity\ reduction = Impurity(parent) - \left(\frac{N(left)}{N(parent)} * Impurity(left) + \frac{N(right)}{N(parent)} * Impurity(right)\right)$$
$$Impurity\ reduction = \frac{4}{9} - \left(\frac{5}{15} * \frac{8}{25} + \frac{10}{15} * \frac{9}{50}\right)$$

- Cost complexity parameter (cp) is a penalty for overfitting to the training data. It is assoicated to the number of terminal nodes in the trees. The higher the value of cp, the less complex the model. cp is

default to 0.01.
- By deafult, running rpart() will automatically execute pruning using k-fold cross validation.

Training (regression tree)

Training a regression tree

In [44]:

```
# code is exactly the same, except the output is a continuous variable, not a fa
```

Note on Regression Tree:

- The steps to spliting the trees are similar to classification tree, except that the objective function is to minimize the residual sum of squared:

$$\min_{j=1,..p} \min_{s} \sum_{i:x_{ij}<s} (y_i - \hat{y}_{R1})^2 + \sum_{i:x_{ij}>=s} (y_i - \hat{y}_{R2})^2$$

where j refers to the predictors and s refers to the cutpoint in variable j

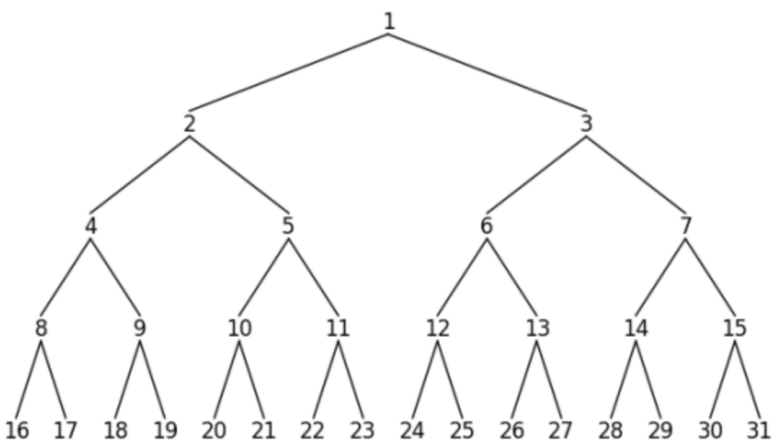
Results of model

Reading the output of CART model

- {node no} || {variable and values used for split} || {total no of obs in node} || {no of misclassified obs in node} || {predicted value for node} || {predicted probabilities/proportion}

```
1) root 434 195 1 (0.44930876 0.55069124)
  2) lctdir=liberal 205  82 0 (0.60000000 0.40000000)
  .
  .
  3) lctdir=conser 229  72 1 (0.31441048 0.68558952)
  .
  .
```

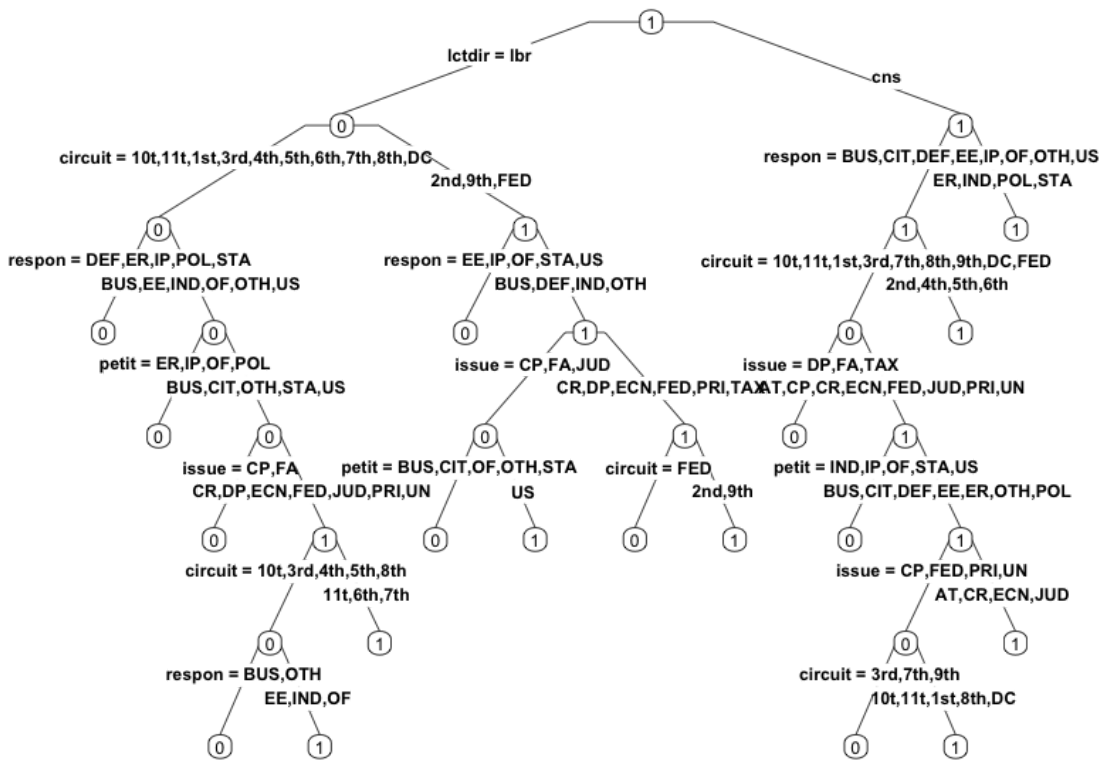
- Notice that 2 and 3 are branched from the same parent node (total number of obs adds up >> 205+229=434). You can find the child node by using this formula: $2n$ (left child) and $2n + 1$ (right child), where n is the index of the parent node



Plotting the CART

In [22]:

```
# plots the tree and adds label to the tree
# prp(cart1)
# labels nodes with the predicted label, not just the leaves
# prp(cart1, type=1)
# draws separate labels for left and right directions for all nodes and label no
prp(cart, type=4)
# also plot probability per class of observations
# prp(cart1, extra=4, type=4)
# probability times fraction of observations at node (sum across level is 1)
# prp(cart1,extra=9,type=4)
```



Display cp table for model

- shows different values of alpha tried
- from smallest tree to largest one
- rel error is the training loss
- xerror is loss from the cross validation set

In [23]:

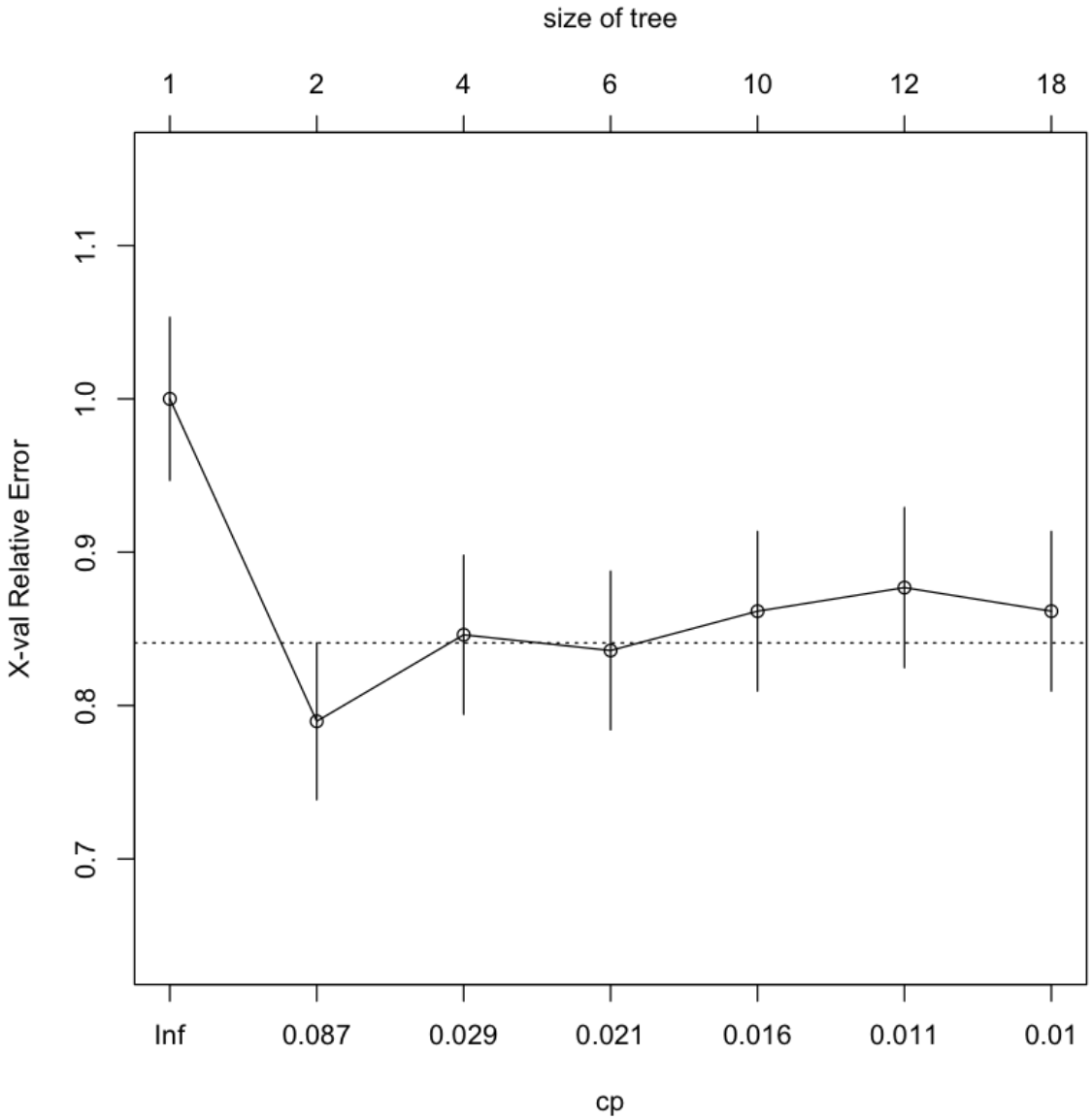
```
1 printcp(cart)
```

```
Classification tree:
rpart(formula = rev ~ petit + respon + circuit + unconst + lctdir +
    issue, data = train, method = "class")

Variables actually used in tree construction:
[1] circuit issue   lctdir  petit   respon

Root node error: 195/434 = 0.44931

n= 434

        CP nsplit rel error  xerror     xstd
1 0.210256      0   1.00000 1.00000 0.053142
2 0.035897      1   0.78974 0.78974 0.051116
3 0.023077      3   0.71795 0.84615 0.051861
4 0.018803      5   0.67179 0.83590 0.051737
5 0.012821      9   0.57949 0.86154 0.052037
6 0.010256     11   0.55385 0.87692 0.052203
7 0.010000     17   0.49231 0.86154 0.052037
```

Plot cross-validated error across different cp

In [24]:

```
1 plotcp(cart)
```



Pruning the tree

When finding the model with the lowest cross-validated error, use a cp value higher than the corresponding cp value to guaranteee the pruned tree. E.g. round up to the nearest second decimal place

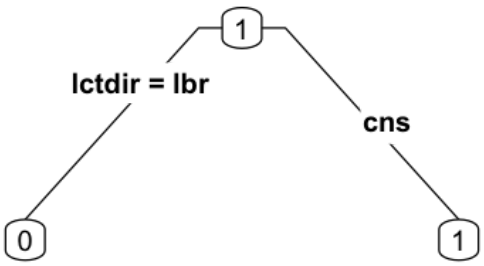
In [36]:

```
1 prune_cart<-prune(cart,cp=0.036)
2 prune_cart
```

```
n= 434

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 434 195 1 (0.4493088 0.5506912)
  2) lctdir=liberal 205  82 0 (0.6000000 0.4000000) *
  3) lctdir=conser 229  72 1 (0.3144105 0.6855895) *
```

In [45]:

```
1 prp(prune_cart, type=4)
```



Building a more complex tree

Setting the cp parameter lower in rpart allows the building of a more complex tree

In [43]:

```
modified_cart<-rpart(as.factor(rev)~petit+respon+circuit+unconst+lctdir+issue, data=train)
printcp(modified_cart)
```

```
Classification tree:
rpart(formula = as.factor(rev) ~ petit + respon + circuit + unconst +
    lctdir + issue, data = train, cp = 1e-08)

Variables actually used in tree construction:
[1] circuit issue   lctdir  petit   respon

Root node error: 195/434 = 0.44931

n= 434

          CP nsplit rel error  xerror     xstd
1 0.21025641      0   1.00000 1.00000 0.053142
2 0.03589744      1   0.78974 0.78974 0.051116
3 0.02307692      3   0.71795 0.88205 0.052256
4 0.01880342      5   0.67179 0.88718 0.052307
5 0.01282051      9   0.57949 0.88718 0.052307
6 0.01025641     11   0.55385 0.89231 0.052358
7 0.00769231     17   0.49231 0.86154 0.052037
8 0.00000001     19   0.47692 0.86154 0.052037
```

Prediction

Class prediction

In [ ]:

```
cart_predict_class<-predict(cart, newdata=test, type='class')
cart_predict_class
```

Probability prediction for ALL classes

In [ ]:

```
cart_predict_prob<-predict(cart, newdata=test)
cart_predict_prob
```

Note: to obtain probability prediction for class label 1 in binary classification, use this:

In [ ]:

```
cart_predict_prob[,2]
```

ROC curve and AUC

To obtain the ROC curve, you must feed in probabilities, NOT labels

In [56]:

```
ROCRpred<-prediction(cart_predict_prob[,2], test$rev)
```

Plotting the ROC curve

In [ ]:

```
ROCRperf <- performance(ROCRpred, measure='tpr', x.measure='fpr')
plot(ROCRperf)
```

Obtaining the AUC value

In [ ]:

```
performance(ROCRpred, measure='auc')
```

2. Random Forest

Importing necessary library

In [59]:

```
library(randomForest)
```

```
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
```

Key features of RF

- It is a combination of decision trees, known as an ensemble model, where the final prediction is obtained using majority vote or averaging across trees
- Each CART trains on a subset of data that is randomly chosen with replacement (bootstrapping), preventing overfitting to dataset
- At each split, a random sample of $m = \sqrt{p}$ predictors are considered, ensuring low covariance between CART

Training

In [61]:

```
forest<-randomForest(as.factor(rev)~petit+respon+circuit+unconst+lctdir+issue, d
```

Prediction

Class prediction

In [ ]:

```
forest_predict_class<-predict(forest, newdata=test)
forest_predict_class
```

Probability prediction

In [ ]:

```
forest_predict_prob<-predict(forest, newdata=test, type='prob')
forest_predict_prob
```

Determining the 'best' parameter

Metric 1: Reduction in impurity

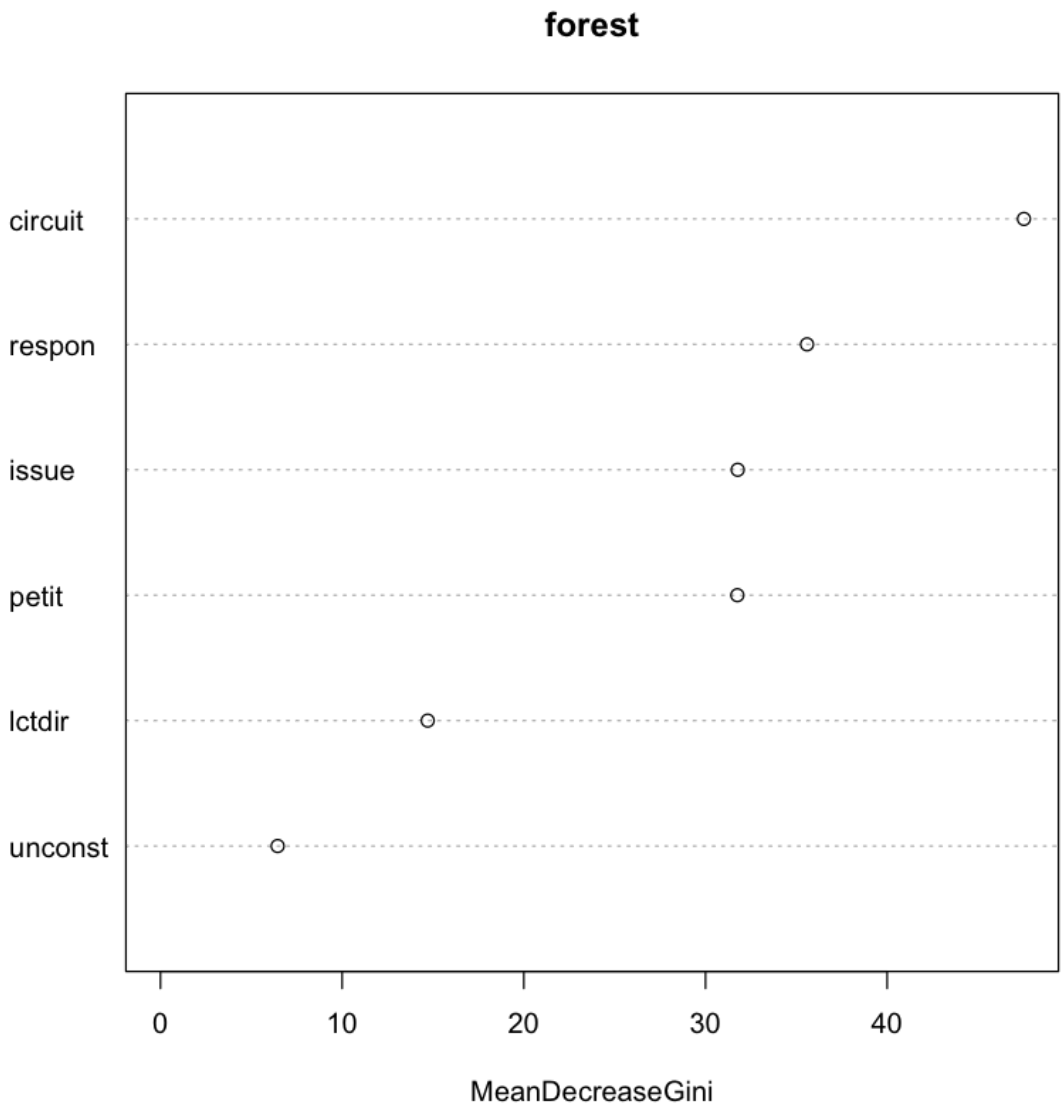
In [69]:

```
importance(forest) # shows the purity of each variable. the higher the better
```

	MeanDecreaseGini
petit	31.752584
respon	35.587801
circuit	47.526841
unconst	6.449626
lctdir	14.708817
issue	31.772368

In [70]:

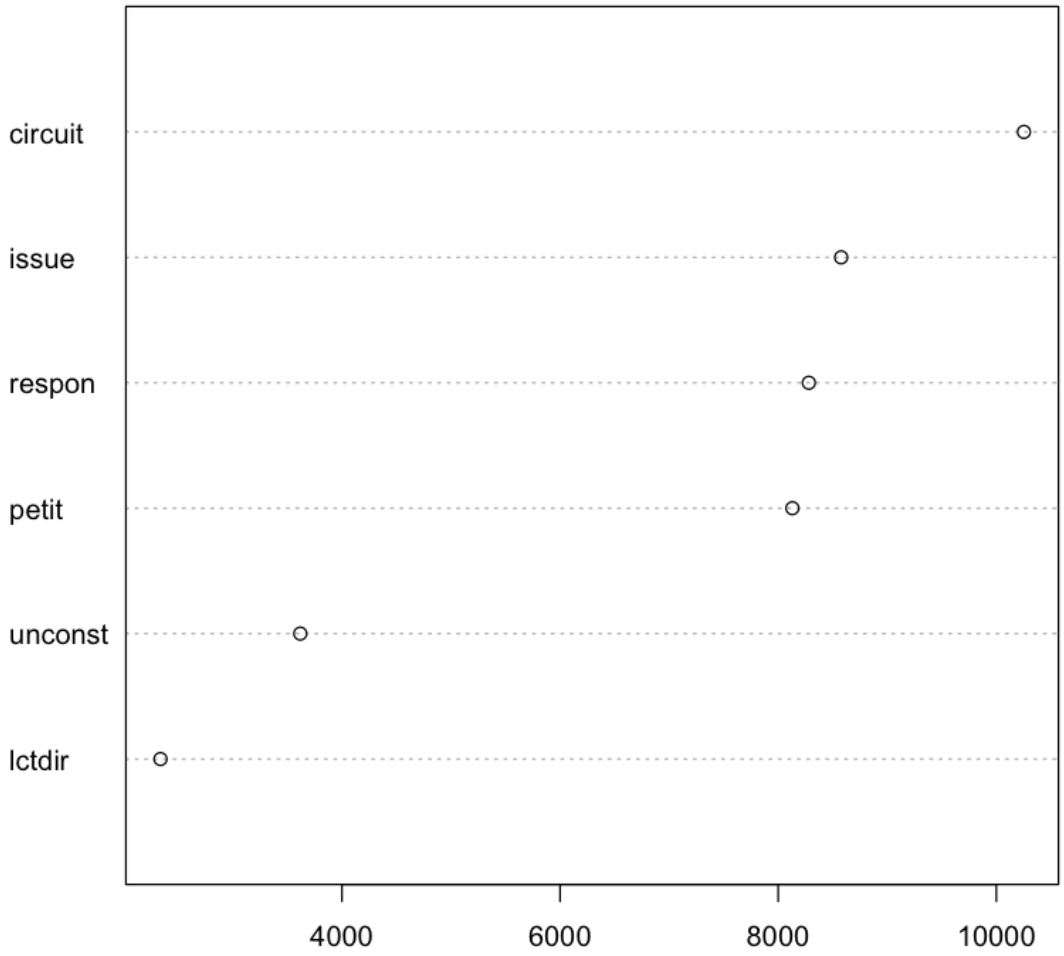
```
1 varImpPlot(forest) # plots the purity of each variable
```



Metric 2: Frequency of variable use

In [72]:

```
vu<-varUsed(forest,count=TRUE) # get all the variables
vusorted<-sort(vu,decreasing=FALSE,index.return=TRUE) # sort variables by freque
dotchart(vusorted$x,names(forest$forest$xlevels[vusorted$ix]))
```



3. Text Analytics

Import necessary library

In [117]:

```
library(tm)
```

Load example dataset

In [110]:

```
twitter<-read.csv("Dataset/twitter.csv", stringsAsFactors=FALSE)
twitter$Negative<-as.factor(twitter$sentiment<=2)
```

Generating a corpus from tweets

In [77]:

```
corpus<-Corpus(VectorSource(twitter$tweet))
```

Pre-processing

Step 1: Lowercase the characters

In [ ]:

```
# if there is error, run the subsequent steps until step 5 before coming back t
corpus<-tm_map(corpus,tolower)
```

Step 2: Remove stopwords in english

In [ ]:

```
corpus<-tm_map(corpus,removeWords,stopwords("english"))
```

Step 3: Remove other possible useless words

In [ ]:

```
corpus<-tm_map(corpus,removeWords,c("drive","driver","driving","self-driving","
```

Step 4: Remove punctuation

In [ ]:

```
corpus<-tm_map(corpus,removePunctuation)
```

Step 5: Stemming

In [ ]:

```
corpus<-tm_map(corpus,stemDocument)
```

Step 6: Creating a Document Term Matrix

In [99]:

```
freq<-DocumentTermMatrix(corpus)
freq
```

```
<<DocumentTermMatrix (documents: 2664, terms: 5843)>>
Non-/sparse entries: 25015/15540737
Sparsity           : 100%
Maximal term length: 41
Weighting          : term frequency (tf)
```

In [100]:

```
1 inspect(freq[1,])
```

```
<<DocumentTermMatrix (documents: 1, terms: 5843)>>
Non-/sparse entries: 6/5837
Sparsity           : 100%
Maximal term length: 41
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs awesom blind driverless googl invest money place print self two
   1      0     0          0     0      1     1     1     1    1   1
```

Step 7: Remove infrequent words to reduce number of features

In [96]:

```
1 findFreqTerms(freq,lowfreq=50)  # find words that occur at least 50 times acros
```

'self' 'awesom' 'driverless' 'googl' 'will' 'autonom' 'traffic' 'vehicl' 'good' 'realli'
'just' 'car' 'cant' 'now' 'wait' 'cool' 'technolog' 'the' 'drive' 'first' 'this' 'futur'
'road' 'say' 'year' 'human' 'make' 'one' 'accid' 'time' 'want' 'think' 'thing'
'new' 'saw' 'can' 'get' 'love' 'much' 'see' 'come' 'use' 'street' 'way' 'great'
'via' 'day' 'uber' 'today' 'take' 'wheel' 'ride' 'like' 'need' 'peopl' 'look' 'driver'
'amp' 'california' 'selfdriv' 'work' 'robot'

In [97]:

```
1 freq[,"day"] # word 'day' occured in 49/2664 entries
```

```
<<DocumentTermMatrix (documents: 2664, terms: 1)>>
Non-/sparse entries: 49/2615
Sparsity           : 98%
Maximal term length: 3
Weighting          : term frequency (tf)
```

In [101]:

```
1 freq<-removeSparseTerms(freq,0.995) # remove sparse terms from the document ter
2 freq
```

```
<<DocumentTermMatrix (documents: 2664, terms: 283)>>
Non-/sparse entries: 14316/739596
Sparsity           : 98%
Maximal term length: 10
Weighting          : term frequency (tf)
```

Step 8: Converting the Document Term Matrix to a matrix, then to a dataframe

In [113]:

```
twittersparse<-as.data.frame(as.matrix(freq))
colnames(twittersparse)<-make.names(colnames(twittersparse)) # ensure that colu
twittersparse$Neg<-twitter$Negative
head(twittersparse)
```

invest	money	place	self	two	awesom	driverless	googl	help	will	⋯	crash	fbi	kill	abl
1	1	1	1	1	0	0	0	0	0	⋯	0	0	0	0
0	0	0	0	0	1	1	1	1	1	⋯	0	0	0	0
0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0
0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0
0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0
0	0	0	0	0	0	0	0	0	0	⋯	0	0	0	0

4. Naive Bayes Model

Using Naives Bayes model on prediction of negative sentiments based on words in a sentence

Import the necessary library

In [118]:

```
library(caTools)
library(e1071)
```

Splitting the dataset into train and test

In [114]:

```
set.seed(123)
spl<-sample.split(twittersparse$Neg,SplitRatio = 0.7)
train<-subset(twittersparse,spl==TRUE)
test<-subset(twittersparse,spl==FALSE)
```

Training

A naive bayes classifier seeks to find the probability of being labelled as class k, given that it has seen predictor 1 to p. This can be modelled as such:

$$P(C_k|X_1,\ldots,X_p) = \frac{P(C_k)P(X_1|C_k)\ldots P(X_p|C_k)}{\sum_l P(C_l)P(X_1|C_l)\ldots P(X_p|C_l)}$$

From the training, we can find $P(C_k)$ and $P(X_i|C_k)$

In [122]:

```
naivebayes<-naiveBayes(Neg~., data=train)
summary(naivebayes)
```

```
        Length Class  Mode
apriori   2    table  numeric
tables  283    -none- list
levels    2    -none- character
call      4    -none- call
```

Apriori probabilities of the classes, $P(C_k)$

In [ ]:

```
naivebayes$apriori
```

Mean and variance for each class. E.g. if there are 2 classes, there will be 4 parameters for each word/feature. Mean and variance will be fed as parameters in a normal distribution for evaluating the conditional probability, $P(X_i|C_k)$:

$$P(X_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{ki}^2}} e^{-\frac{(X_i-\mu_{ki})^2}{2\sigma_{ki}^2}}$$

In [ ]:

```
naivebayes$tables
```

Prediction

In [ ]:

```
naivebayes_predict<-predict(naivebayes, newdata=test, type='class')
naivebayes_predict
```

5. Clustering

Goal: Identify movies that are similar based on genres

Load the example dataset

In [162]:

```
movies<-read.csv("Dataset/movies.csv", stringsAsFactors = FALSE)
genres<-read.csv("Dataset/genres.csv", header=FALSE,sep="|",col.names=c("X1","X
head(movies)
head(genres)
```

movieId	title
1	Toy Story (1995)
2	Jumanji (1995)
3	Grumpier Old Men (1995)
4	Waiting to Exhale (1995)
5	Father of the Bride Part II (1995)
6	Heat (1995)

X1	X2	X3	X4	X5	X6	X7
Adventure	Animation	Children	Comedy	Fantasy		
Adventure	Children	Fantasy				
Comedy	Romance					
Comedy	Drama	Romance				
Comedy						
Action	Crime	Thriller				

Pre-processing

In [135]:

```
# each variable has a different number of factor levels, hence need to creates a
fac<-union(union(union(union(union(union(levels(genres$X1),levels(genres$X2)),le
# standardize factor level with same 20 categories for each variable
genres$X1<-factor(genres$X1,fac)
genres$X2<-factor(genres$X2,fac)
genres$X3<-factor(genres$X3,fac)
genres$X4<-factor(genres$X4,fac)
genres$X5<-factor(genres$X5,fac)
genres$X6<-factor(genres$X6,fac)
genres$X7<-factor(genres$X7,fac)
# an empty matrix with row=#movies and col=#genres
M<-matrix(0,nrow=8569,ncol=20)
# reset column names for matrix
colnames(M)<-fac
# genres[i,"X1"] returns the exact string name of the genre, which can be used i
# for each movie, label 1 for genre X1 to X7 if exists
for (i in 1:8569){
  M[i,genres[i,"X1"]]=1
  M[i,genres[i,"X2"]]=1
  M[i,genres[i,"X3"]]=1
  M[i,genres[i,"X4"]]=1
  M[i,genres[i,"X5"]]=1
  M[i,genres[i,"X6"]]=1
  M[i,genres[i,"X7"]]=1
}
# convert matrix to a dataframe
Data <-as.data.frame(M)
 # add in the title
Data$title<-movies$title
# drop the 19th column
Data<-Data[,-19]
head(Data)
```

Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir
0	1	1	1	1	0	0	0	1	0
0	1	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0

K-means clustering

General algorithm:

1. Start with a random set of k observations. E.g. for k=5, we choose 5 random points as start
2. Assign each observation to the cluster where the centroid is closest (assignment step)
3. Calculate the centroids of the observations in the new cluster (update step)
4. Repeat Step 2 and 3 until assignment converges

In [ ]:

```
set.seed(1)
# nstart represents 20 different random initial configuration. kmeans will choos
cluster_kmeans<-kmeans(Data[,1:19], centers=10,nstart=20)
cluster_kmeans
```

Note:

- increasing *nstart* allows finding a better solution and decreases the error, but increases the time taken to run
- increasing *centers* decreases the error and increases the fit, but it loses the ability to find similarities

Within cluster sum of squared errors for each cluster

In [145]:

```
cluster_kmeans$withinss
```

363.567877629074 771.092145015151 1463.78130590349 441.071691176466
1155.74197530864 1077.61851156981 636.899505766041 740.077803203597
293.305851063817 381.623225806405

Total within cluster sum of squared errors

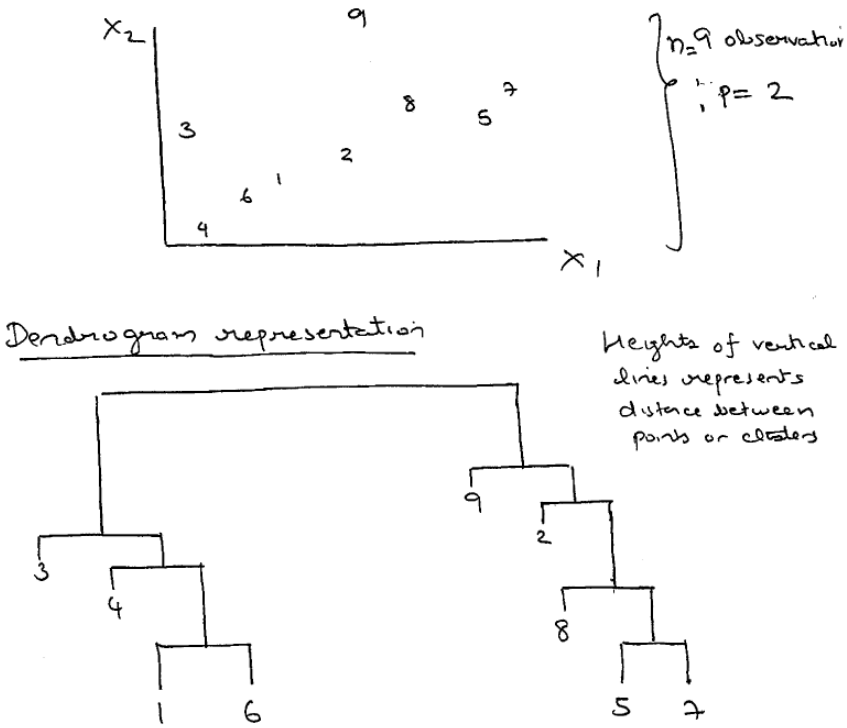
In [141]:

```
cluster_kmeans$tot.withinss
```

7324.7798924425

Hierarchical clustering

Hierarchical clustering provides a tree-based representation of observations from bottom-up. The fusion of two leafs correspond to similar observations. Observations that fuse at the bottom tend to be more similar than those fused at the top. The height of the tree indicates how different the observations are.



To identify clusters from a dendogram, make a horizontal cut across the dendogram

General algorithm:

1. Begin with n observations and a measure of pairwise dissimilarities. Each observation starts off as a cluster on its own
2. For i = n,n-1,...,2, evaluate the pairwise dissimilarities between each cluster and find the two clusters that are most similar. Fuse these two clusters into one.

Types of distance measures (dissimilarities):

1. Euclidean distance
2. Manhatten distance

Types of agglomeration method (dissimilarities between groups):

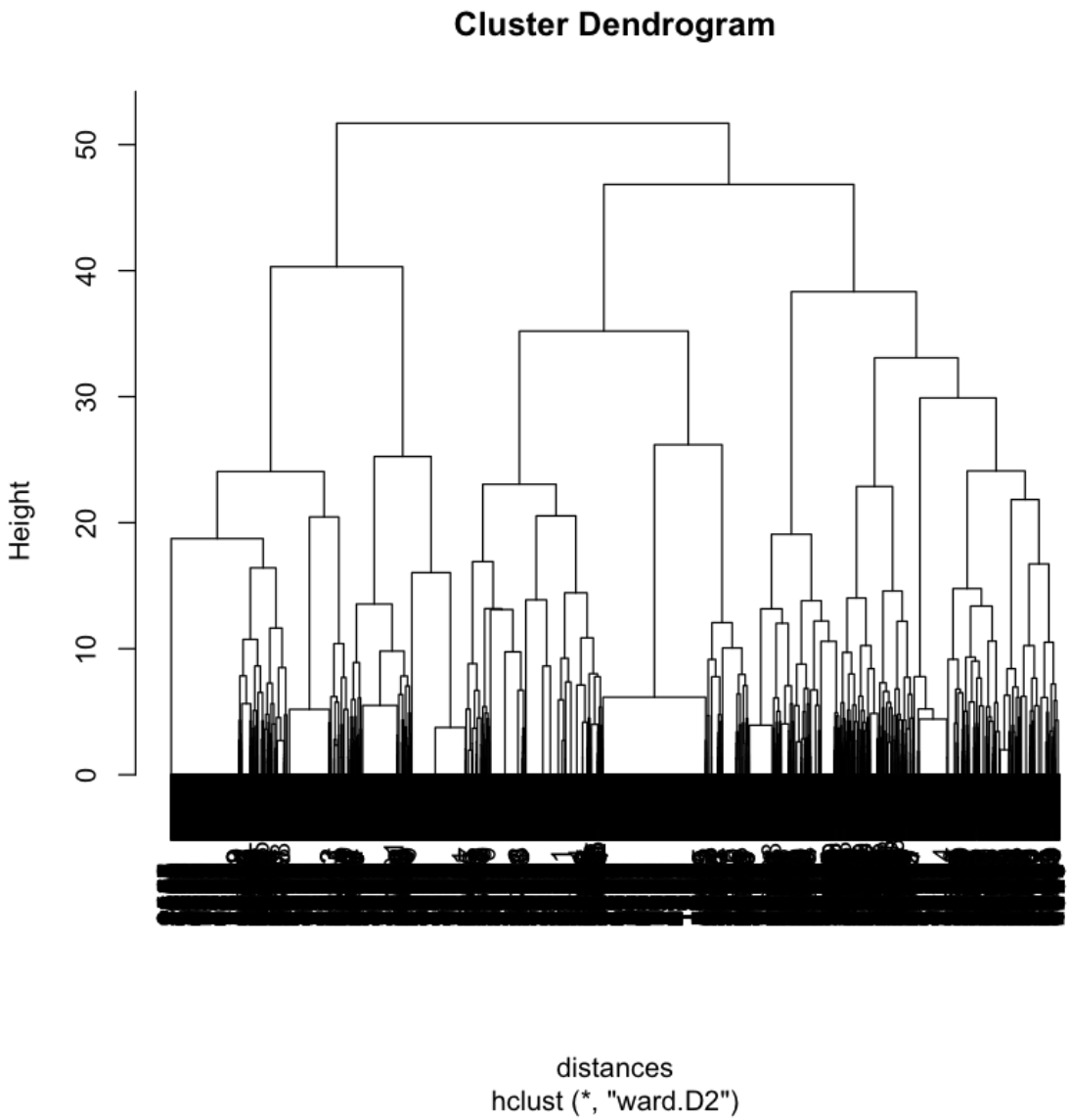
1. Complete linkage: max distance between two points in cluster A and B
2. Average linkage: average distance between two points in cluster A and B
3. Ward's criterion: minimizes the total within cluster variance

In [150]:

```
# computes distances between every movies using the first 19 colums with Euclide
distances<-dist(Data[,1:19], method="euclidean")
# hierarchical cluster using distances (ward method finds spherical clusters)
cluster_hi<-hclust(distances,method="ward.D2")
```

In [151]:

```
1 plot(cluster_hi)
```



Cuts the tree such that there are only 10 clusters

In [155]:

```
1 cluster_hi_smol<-cutree(clusterhi, k=10)
```

Identifying the main genres in each cluster

In [158]:

```
# an empty matrix with row=#genres and columns=#clusters
Categories<-matrix(0,nrow=19, ncol=10)
for(i in 1:19){
  # i refers to the genre
  # at each iteration, we determine the mean value of each cluster for a single
  # the higher the value, the more significant the genre is in the cluster
  Categories[i,]<-tapply(Data[,i],cluster_hi_smol, mean)
}
rownames(Categories)<-colnames(Data)[1:19]
Categories
```

Action	0.179775281	0.000000000	0.1263768116	0.3825757576	0.365241636	0.110604333
Adventure	0.457865169	0.000000000	0.0684057971	0.0340909091	0.396840149	0.025085519
Animation	0.363764045	0.005725191	0.0023188406	0.0015151515	0.045539033	0.007981756
Children	0.526685393	0.013358779	0.0075362319	0.0015151515	0.094795539	0.001140251
Comedy	0.532303371	1.000000000	0.9304347826	0.0128787879	0.121747212	0.168757127
Crime	0.012640449	0.000000000	0.1460869565	0.5318181818	0.018587361	0.031927024
Documentary	0.000000000	0.000000000	0.0133333333	0.0022727273	0.005576208	0.009122007
Drama	0.191011236	0.431297710	0.3408695652	0.6500000000	0.263940520	0.164196123
Fantasy	0.620786517	0.000000000	0.0081159420	0.0060606061	0.020446097	0.103762828
Film-Noir	0.000000000	0.000000000	0.0005797101	0.0621212121	0.001858736	0.000000000
Horror	0.016853933	0.000000000	0.0040579710	0.0348484848	0.116171004	0.916761688
Musical	0.132022472	0.000000000	0.1176811594	0.0007575758	0.006505576	0.004561003
Mystery	0.036516854	0.000000000	0.0255072464	0.1628787879	0.036245353	0.193842645
Romance	0.139044944	1.000000000	0.0626086957	0.0022727273	0.027881041	0.019384265
Sci-Fi	0.085674157	0.000000000	0.0034782609	0.0242424242	0.568773234	0.095781072
Thriller	0.005617978	0.000000000	0.0626086957	0.6446969697	0.263011152	0.489167617
War	0.007022472	0.000000000	0.0023188406	0.0037878788	0.005576208	0.002280502
Western	0.005617978	0.000000000	0.0000000000	0.0030303030	0.192379182	0.001140251
IMAX	0.071629213	0.000000000	0.0028985507	0.0083333333	0.057620818	0.012542759

6. Collaborative filtering

Collaborative filtering uses existing information of user ratings to make prediction on missing ratings or to obtain top N recommendations

Types of collaborative filtering techniques:

1. **Baseline model:** predict missing rating based on average ratings across all user for each item (baseline model 1) OR based on average ratings across all items for each user (baseline model 2)
2. **User-based collaborative filtering:** identify similar users and use known ratings of item A by user 1 to predict unknown rating of item A by user 2, where user 1 and 2 are found to be similar
3. **Item-based collaborative filtering:** identify similar items and use known ratings of item A by user 1 to predict unknown rating of item B by user 1, where item A and B are found to be similar

Load the example dataset

In [160]:

```
ratings<-read.csv("Dataset/ratings.csv")
head(ratings)
```

userId	movieId	rating
1	6	2
1	22	3
1	32	2
1	50	5
1	110	4
1	164	3

Pre-processing

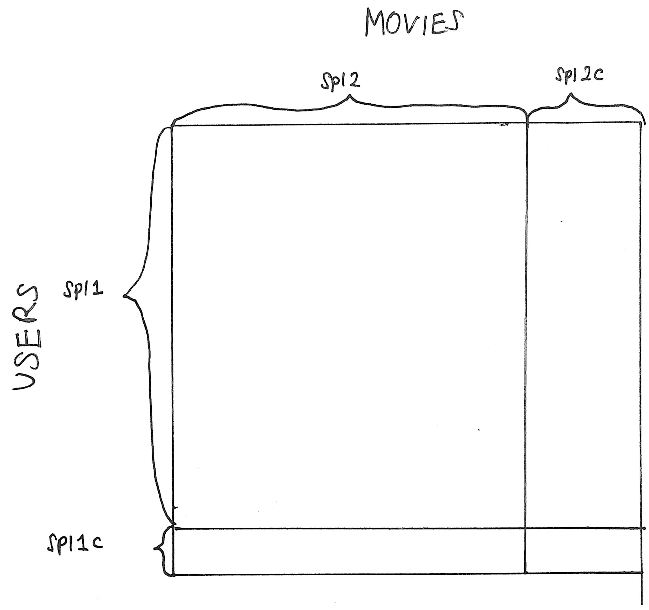
In [164]:

```
# an empty matrix with row=#users and column=#movies
Data<-matrix(nrow = length(unique(ratings$userId)), ncol=length(unique(ratings$r
# naming the row and columns with index of unique users and movies
rownames(Data)<-unique(ratings$userId)
colnames(Data)<-unique(ratings$movieId)
# fill the matrix with the ratings based on userid and movieid
for(i in 1:nrow(ratings)){
  Data[as.character(ratings$userId[i]),as.character(ratings$movieId[i])]<-ratinç
}
head(Data)
```

```
ERROR while rich displaying an object: Error in sprintf(wrap, header,
body): 'fmt' length exceeds maximal format length 8192

Traceback:
1. FUN(X[[i]], ...)
2. tryCatch(withCallingHandlers({
 .     rpr <- mime2repr[[mime]](obj)
 .     if (is.null(rpr))
 .         return(NULL)
 .     prepare_content(is.raw(rpr), rpr)
 . }, error = error_handler), error = outer_handler)
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. doTryCatch(return(expr), name, parentenv, handler)
6. withCallingHandlers({
 .     rpr <- mime2repr[[mime]](obj)
 .     if (is.null(rpr))
 .         return(NULL)
 .     prepare_content(is.raw(rpr), rpr)
 . }, error = error_handler)
7. mime2repr[[mime]](obj)
8. repr_latex.matrix(obj)
9. repr_matrix_generic(obj, sprintf("\\begin{tabular}{%s}\n%%s%%s\\end
{tabular}\n",
 .     cols), "%s\\\\\n\\hline\n", "  &", " %s &", "%s", "\t%s\\\\\n",

 .     "%s &", " %s &", escape_fun = latex_escape_vec, ...)
10. sprintf(wrap, header, body)
```

	6	22	32	50	110	164	198	260	296	303	⋯	7213	7252	7274	7275	4828	92481
1	2	3	2	5.0	4	3	3	5	4	3	⋯	NA	NA	NA	NA	NA	NA
2	NA	NA	3	4.0	NA	NA	NA	NA	NA	NA	⋯	NA	NA	NA	NA	NA	NA
3	NA	NA	NA	NA	5	NA	NA	5	NA	NA	⋯	NA	NA	NA	NA	NA	NA
4	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	⋯	NA	NA	NA	NA	NA	NA
5	NA	NA	NA	4.5	NA	NA	NA	NA	NA	NA	⋯	NA	NA	NA	NA	NA	NA
6	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	⋯	NA	NA	NA	NA	NA	NA

Split the dataset. We want to predict ratings in region: (spl1c,spl2c)



Note: the *sample()* function will sample randomly and will NOT sample based on a straight cut as illusrated above. It is drawn only to show the split ratio

In [165]:

```
set.seed(1)
spl1<-sample(1:nrow(Data),0.98*nrow(Data))
spl1c<-setdiff(1:nrow(Data),spl1) # choose rows not selected by spl1
set.seed(2)
spl2<-sample(1:ncol(Data),0.8*ncol(Data))
spl2c<-setdiff(1:ncol(Data),spl2) # choose columns not selected by spl2
```

Baseline model 1

Predict missing rating based on average ratings across all user for each item

In [166]:

```
Base1<-matrix(nrow=length(spl1c), ncol=length(spl2c))
# for each user id in spl1c
for(i in 1:length(spl1c)){
    # mean across the whole of column
    Base1[i,]<-colMeans(Data[spl1,spl2c],na.rm=TRUE)
    # users would have the same prediction for same item
}
head(Base1)
```

4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1
4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1
4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1
4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1
4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1
4.124214	1	4.055172	3.633721	3.415301	3.388889	3.5	3.694245	3.805556	3.763158	⋯	1

Baseline model 2

Predict missing rating based on average ratings across all items for each user

In [167]:

```
Base2<-matrix(nrow=length(spl1c), ncol=length(spl2c))
# for each movie id in spl2c
for(j in 1:length(spl2c)){
    # mean across the whole row
    Base2[,j]<-rowMeans(Data[spl1c,spl2],na.rm=TRUE)
    # items would have the same prediction for same user
}
head(Base2)
```

3.750000	3.750000	3.750000	3.750000	3.750000	3.750000	3.750000	3.750000	3.750000	3.750
3.817391	3.817391	3.817391	3.817391	3.817391	3.817391	3.817391	3.817391	3.817391	3.817
3.866667	3.866667	3.866667	3.866667	3.866667	3.866667	3.866667	3.866667	3.866667	3.866
4.136364	4.136364	4.136364	4.136364	4.136364	4.136364	4.136364	4.136364	4.136364	4.136
3.568966	3.568966	3.568966	3.568966	3.568966	3.568966	3.568966	3.568966	3.568966	3.568
3.152778	3.152778	3.152778	3.152778	3.152778	3.152778	3.152778	3.152778	3.152778	3.152

User-based collaborative filtering

Predict missing rating based on average ratings across nearest 250 users for each item

In [169]:

```
User <- matrix(nrow=length(spl1c),ncol=length(spl2c))
# for temporarily storing correlation between 1 user in spl1c and all users in s
Cor<-matrix(nrow=length(spl1),ncol=1)
# for storing correlation between users in spl1c and users in spl1 in decreasing
Order<-matrix(nrow=length(spl1c),ncol=length(spl1))

# for each user in spl1c
for(i in 1:length(spl1c)){
    # for each user in spl1
    for(j in 1:length(spl1)){
        # comparing the rating pattern between user in spl1c and user in spl1
        Cor[j]<-cor(Data[spl1c[i],spl2],Data[spl1[j],spl2], use="pairwise.comple
    }
    # sort the correlation in decreasing order and returns the INDEX, not the va
    v<-order(Cor,decreasing = TRUE, na.last = NA)
    # insert NA to account for users who have no common ratings of movies with t
    Order[i,]<-c(v,rep(NA,times=length(spl1)-length(v)))
}
head(Order)
```

```
"the standard deviation is zero"Warning message in cor(Data[spl1c[i],
spl2], Data[spl1[j], spl2], use = "pairwise.complete.obs"):
"the standard deviation is zero"Warning message in cor(Data[spl1c[i],
spl2], Data[spl1[j], spl2], use = "pairwise.complete.obs"):
"the standard deviation is zero"Warning message in cor(Data[spl1c[i],
spl2], Data[spl1[j], spl2], use = "pairwise.complete.obs"):
"the standard deviation is zero"Warning message in cor(Data[spl1c[i],
spl2], Data[spl1[j], spl2], use = "pairwise.complete.obs"):
"the standard deviation is zero"
```

11	69	108	155	164	348	374	379	381	444	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
33	37	45	53	123	181	260	266	281	298	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	14	31	42	50	55	104	106	108	110	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1	26	35	53	81	147	292	309	316	374	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
15	25	47	49	58	143	147	182	183	191	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
21	26	31	128	140	143	181	206	255	281	⋯	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

In [182]:

```
# for each user in spl1c
for (i in 1:length(spl1c)){
    # Order[i,1:250] returns the indexes of nearest 250 users in spl1 matrix
    # spl1[Order[i,1:250]] returns the ORIGINAL indexes of nearest 250 users in
    # Data[spl1[Order[i,1:250]],spl2c] returns the ratings for nearest 250 users
    # apply a mean across nearest 250 users for each item to generate predicted
    User[i,]<-colMeans(Data[spl1[Order[i,1:250]],spl2c],na.rm=TRUE)
}
head(User)
```

4.176471	1	4.054878	3.692982	3.467391	3.423077	3.300000	3.652174	3.761111	3.666667
4.268657	1	4.057377	3.700000	3.351190	3.208333	3.000000	3.403509	3.761538	3.575000
4.130719	1	4.126263	3.680851	3.336634	3.366667	3.700000	3.762821	3.830357	3.770833
4.161074	1	4.223077	3.734375	3.282051	3.500000	3.666667	3.566667	3.789855	4.107143
4.187023	NaN	4.070175	3.736111	3.467742	3.300000	3.000000	3.673469	3.838710	3.593750
4.166667	1	4.050633	3.750000	3.394366	3.192308	3.375000	3.769231	3.851852	3.675000

Model Comparison

Finding the nearest 250 neighbours to predict ratings achieve a lower RMSE score. Yey!

In [184]:

```
RMSEBase1<-sqrt(mean((Data[spl1c,spl2c]-Base1)^2, na.rm=TRUE))
RMSEBase1
RMSEBase2<-sqrt(mean((Data[spl1c,spl2c]-Base2)^2, na.rm=TRUE))
RMSEBase2
RMSEUserPred<-sqrt(mean((Data[spl1c,spl2c]-User)^2, na.rm=TRUE))
RMSEUserPred
```

0.931029570020264

0.99533769837973

0.89855670979697

7. Singular Value Decomposition

Given a rectangular matrix X of dimension m x n, a SVD of X is of the form (assume m>n):

$$X = USV^T$$

where the size of the matrix are

- X: m x m
- U: m x m
- S: m x n
- Vt: n x n

Note:

1. Both matrices U and V are orthorgonal matrices, namely $U^TU = UU^T = I$ and $V^TV = VV^T = I$ where I is an identity matrix
2. Matrix s is a diagonal matrix with diagonal entries $\sigma_1, \sigma_2, .., \sigma_n >= 0$ where $n = min(m, n)$ at the top and 0's filling the rest of the matrix
3. $\sigma_1, \sigma_2, .., \sigma_n$ in S is arranged in descending order (i.e. $\sigma_1 >= \sigma_2 ..$)

Low rank approximation

Idea: we drop smaller singular values to approximate. E.g. A rank-10 approximation takes the top 10 singular values. Hence, we take the corresponding first 10 columns in U and first 10 rows in V^T

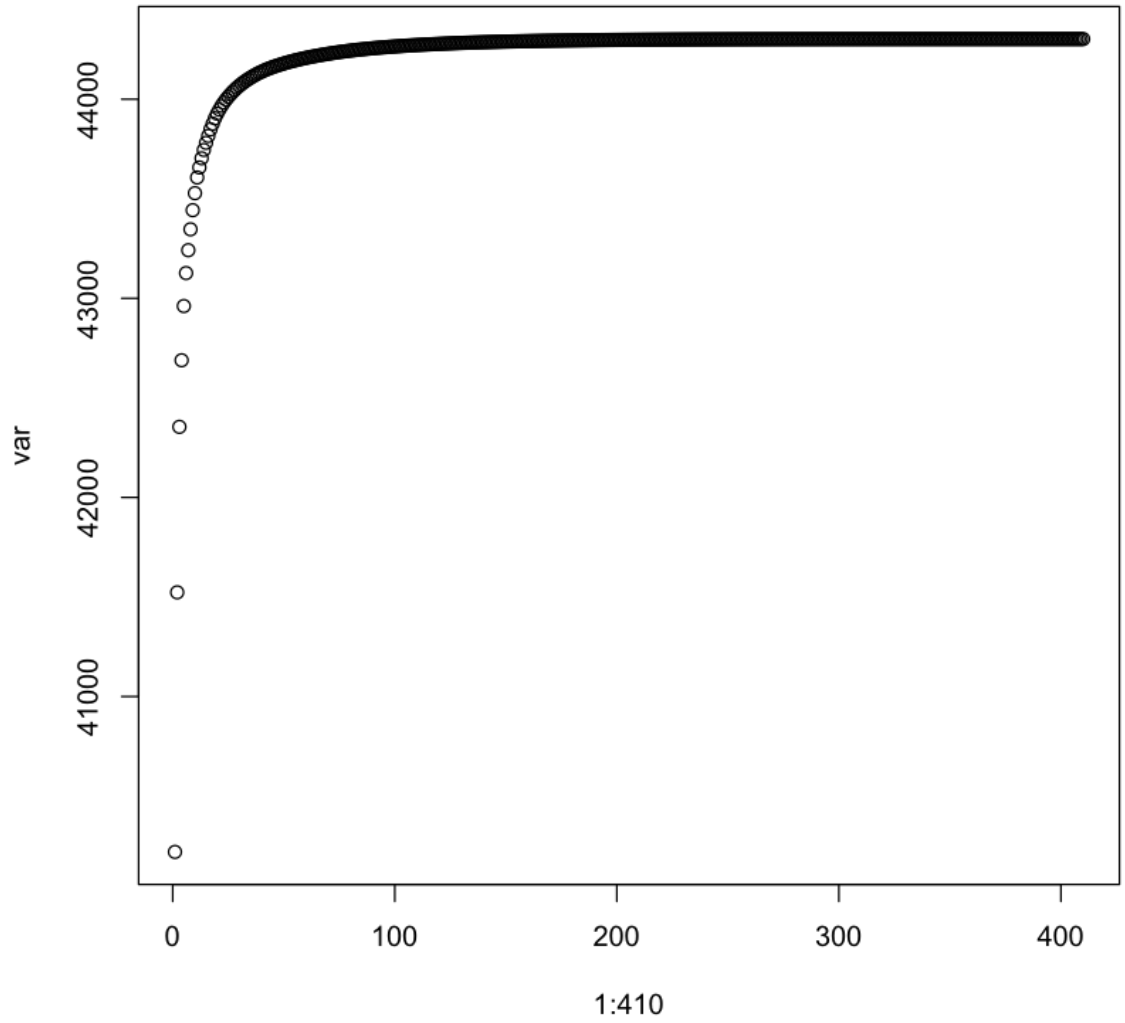
In [188]:

```
library(jpeg)
lky<-readJPEG("Dataset/lky.jpg")
s<-svd(lky[,,1])

# rank 10 approximation
lky10<- s$u[,1:10]%*%diag(s$d[1:10])%*%t(s$v[,1:10])

# rank 50 approximation
lky50<- s$u[,1:50]%*%diag(s$d[1:50])%*%t(s$v[,1:50])
```

Plot a graph to illustrate the significance of the singular values

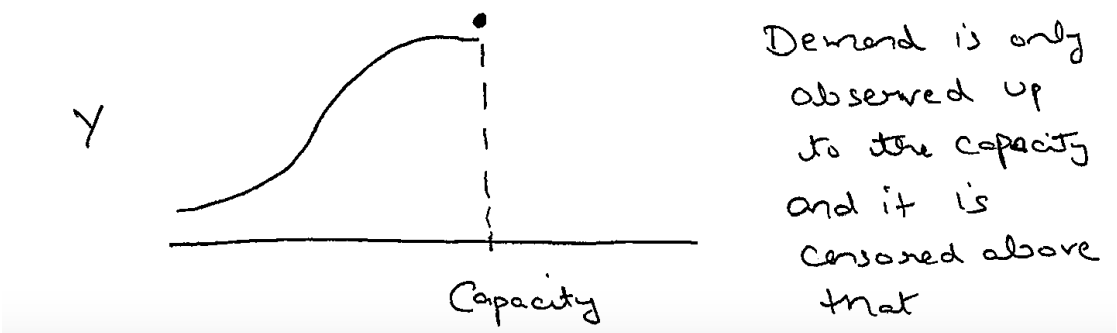
In [189]:

```
var<-cumsum(s$d^2) # cumulative function
plot(1:410, var)
```



8. Tobit and Censored Data

Censored variables are variables with a large fraction of observations at either the min or max. The following diagram shows a **right censored data**



Motivation for Tobit model:

- Modelling observations at the min/max using vanilla linear regression gives erronous results due to mixed signals from features. (i.e. observations at the min/max can have a wide range of features, which is not modelled well by linear regression)
- Removing observations at min/max results in a severe decrease in dataset size

Tobit model deals with censored regression problems. Let's assume a left-censored variable y at 0:

$$y_i = \begin{cases} y^*, & \text{if } y^* \geq 0 \\ 0, & \text{if } y^* < 0 \end{cases}$$

$$y^* = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} + \epsilon_i, \forall i = 1, .., n$$

where $\{\beta_0, .., \beta_p\}$ are the model coefficients, $x_1, .., x_p$ the predictors, n the number of observations, and ϵ as the model error. $\epsilon_i \sim N(0, \sigma^2)$ is assumed.

We want to find β that maximises the log-likelihood of the observations. Since the data is censored, we need to decompose this into two sections - where the observations were truncated to zero, and those which were not:

$$LL(\beta) = \sum_{y_i=0} l_i(\beta) + \sum_{y_i>0} l_i(\beta)$$

Let us consider the log-likelihood for an observation which was censored (i.e. $y_i = 0$). Since $y_i = \max\{y_i^*, 0\}$, we assume that $y_i^* \leq 0$, that is to say,

$$\begin{aligned} l_i(\beta) &= \log\Big(P(\beta^T x_i + \epsilon_i \leq 0)\Big) \\ &= \log\Big(P(\epsilon_i \leq -\beta^T x_i)\Big) \\ &= \log\left(\Phi\Big(-\frac{\beta^T x_i}{\sigma}\Big)\right) \\ &= \log\left(1 - \Phi\Big(\frac{\beta^T x_i}{\sigma}\Big)\right) \end{aligned}$$

For an uncensored observation $y_i > 0$, the log-likelihood of the observation is essentially the log-density that the error ϵ_i makes up for the difference between $y_i^* = y_i$ and $\beta^T x_i$. Representing said density as $f_\epsilon(\cdot)$, we have:

$$\begin{aligned} l_i(\beta) &= \log\Big(f_\epsilon(y_i - \beta^T x_i)\Big) \\ &= \log\left(\frac{1}{\sigma} \cdot \phi\Big(\frac{y_i - \beta^T x_i}{\sigma}\Big)\right) \end{aligned}$$

Finally, we write the log-likelihood of the Tobit model parameterised by linear regression coefficients β as such:

$$LL(\beta) = \sum_{y_i=0} \log\left(1 - \Phi\Big(\frac{\beta^T x_i}{\sigma}\Big)\right) + \sum_{y_i>0} \log\left(\frac{1}{\sigma} \cdot \phi\Big(\frac{y_i - \beta^T x_i}{\sigma}\Big)\right)$$

Import the necessary library

In [190]:

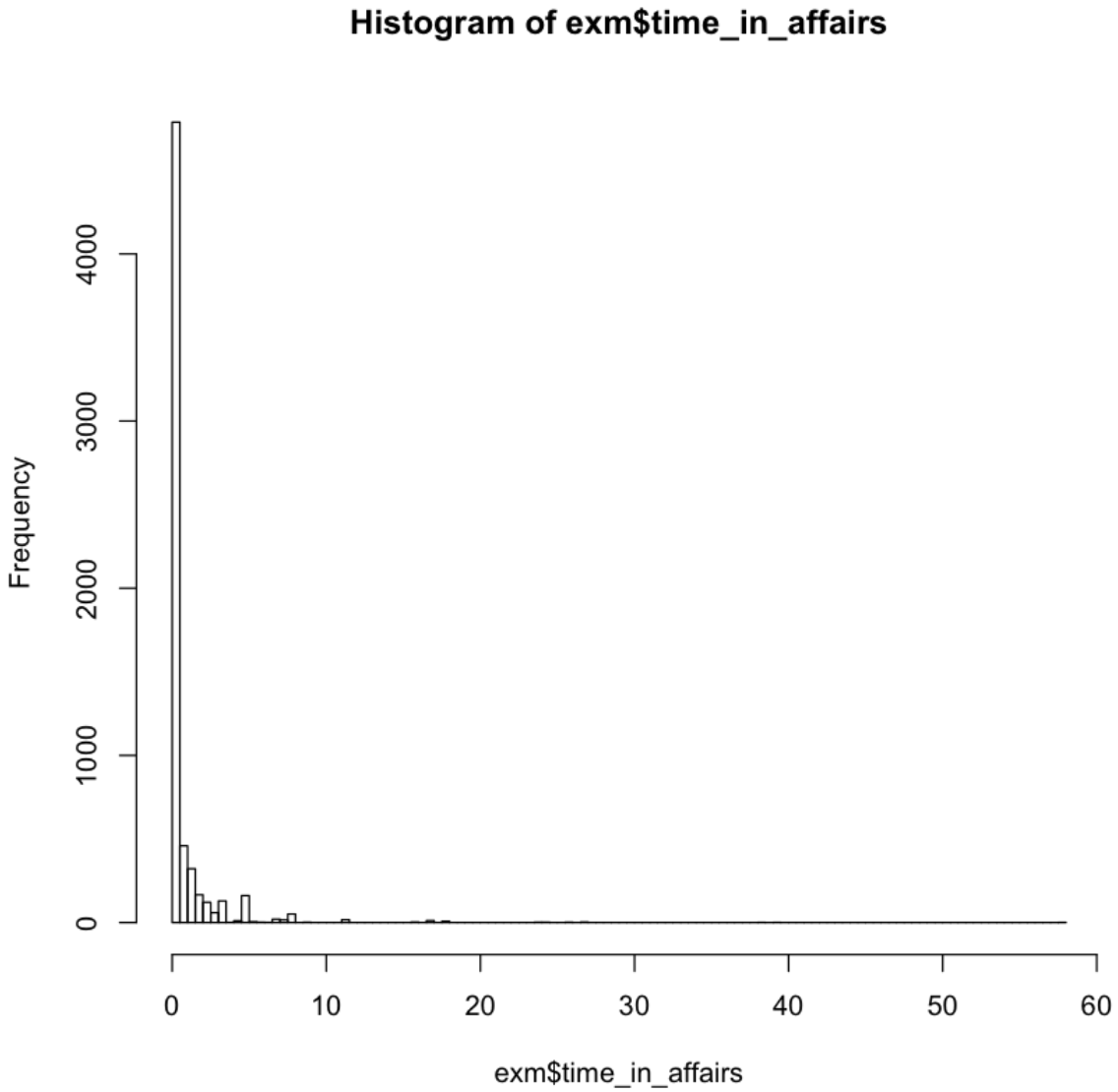
```
1 library(survival)
```

```
Attaching package: ‘survival’

The following object is masked from ‘package:rpart’:

    solder
```

Load the example data

In [194]:

```
1 exm<-read.csv("Dataset/extramarital.csv")
2 hist(exm$time_in_affairs,breaks = 100)
```



Split the dataset into train and test

In [197]:

```
set.seed(100)
spl<-sample(nrow(exm),0.7*nrow(exm))
train<-exm[spl,]
test<-exm[-spl,]
```

Training

Create a left-censored survival object using *Surv()* as response and fit a tobit model assuming error is Gaussian

In [199]:

```
tobit<-survreg(Surv(time_in_affairs, time_in_affairs>0,type="left")~.,data=trai
summary(tobit)
```

```
Call:
survreg(formula = Surv(time_in_affairs, time_in_affairs > 0,
    type = "left") ~ ., data = train, dist = "gaussian")
                 Value Std. Error      z       p
(Intercept)      7.0440     0.8144   8.65 < 2e-16
marriage_rating -1.4505     0.0847 -17.12 < 2e-16
age             -0.0828     0.0288  -2.88 0.00399
yrs_married      0.1043     0.0272   3.83 0.00013
religiosity     -0.8853     0.0985  -8.98 < 2e-16
education       -0.1074     0.0428  -2.51 0.01208
occupation       0.4027     0.0935   4.31 1.7e-05
Log(scale)       1.4759     0.0206  71.73 < 2e-16

Scale= 4.37

Gaussian distribution
Loglik(model)= -5436.9   Loglik(intercept only)= -5676.2
        Chisq= 478.62 on 6 degrees of freedom, p= 3.4e-100
Number of Newton-Raphson Iterations: 4
n= 4456
```

Prediction

In [211]:

```
tobit_predict<-predict(tobit, newdata=test)
# prediction from tobit model can be negative
tobit_predict
```

10
-0.313908029723007
12
-4.77065373054739
13
-0.0202050647015344
14
-3.88539170169856
24
-1.63593514671836
26
-4.94780848229577
27
0.278302921245045
30
-0.099095679686573
33

In [202]:

```
# tobit model would perform better at detecting individuals who have time_in_af
table(tobit_predict<=0, test$time_in_affairs==0)
```

```
       FALSE TRUE
FALSE     92   43
TRUE     523 1252
```

Survival function and Hazard rate

The survival function, $S(t)$ expresses the probability that a subject survives longer than time t. It is defined as follows:

$$S(t) = P(T \geq t) = 1 - F(t)$$

Given that a subject has survived till time t, we are interested in the probability of the event occuring (the subject not surviving): $P(t \leq T \leq t + \Delta t)$. We can then define the hazard rate, $\lambda(t)$, which is the instantaneous rate of occurence of the event:

$$\lambda(t) = \lim_{\Delta t -> 0} \frac{P(t \leq T \leq t + \Delta t | T \geq t)}{\Delta t} = \lim_{\Delta t -> 0} \frac{P(t \leq T \leq t + \Delta t)}{P(T \geq t)\Delta t} = \lim_{\Delta -> 0} \frac{F(t + \Delta t) - F(t)}{S(t)\Delta t} = \frac{f(t)}{S(t)}$$

Kaplan-Meier estimator

Objective: *estimate the survival function,* $S(t)$

The Kaplan-Meier estimate of the survival function is defined as such:

$$S(t) = \Pi_{i:t_i<t}(1 - \frac{d_i}{n-i})$$

This is really simply $\frac{\#people\ alive\ at\ time\ t}{\#people}$

In [213]:

```
heart<-read.csv("Dataset/heart.csv")
head(heart)
```

start	stop	event	age	surgery	transplant	id
0	50	1	31	0	0	1
0	6	1	52	0	0	2
0	1	0	54	0	0	3
1	16	1	54	0	1	3
0	36	0	40	0	0	4
36	39	1	40	0	1	4

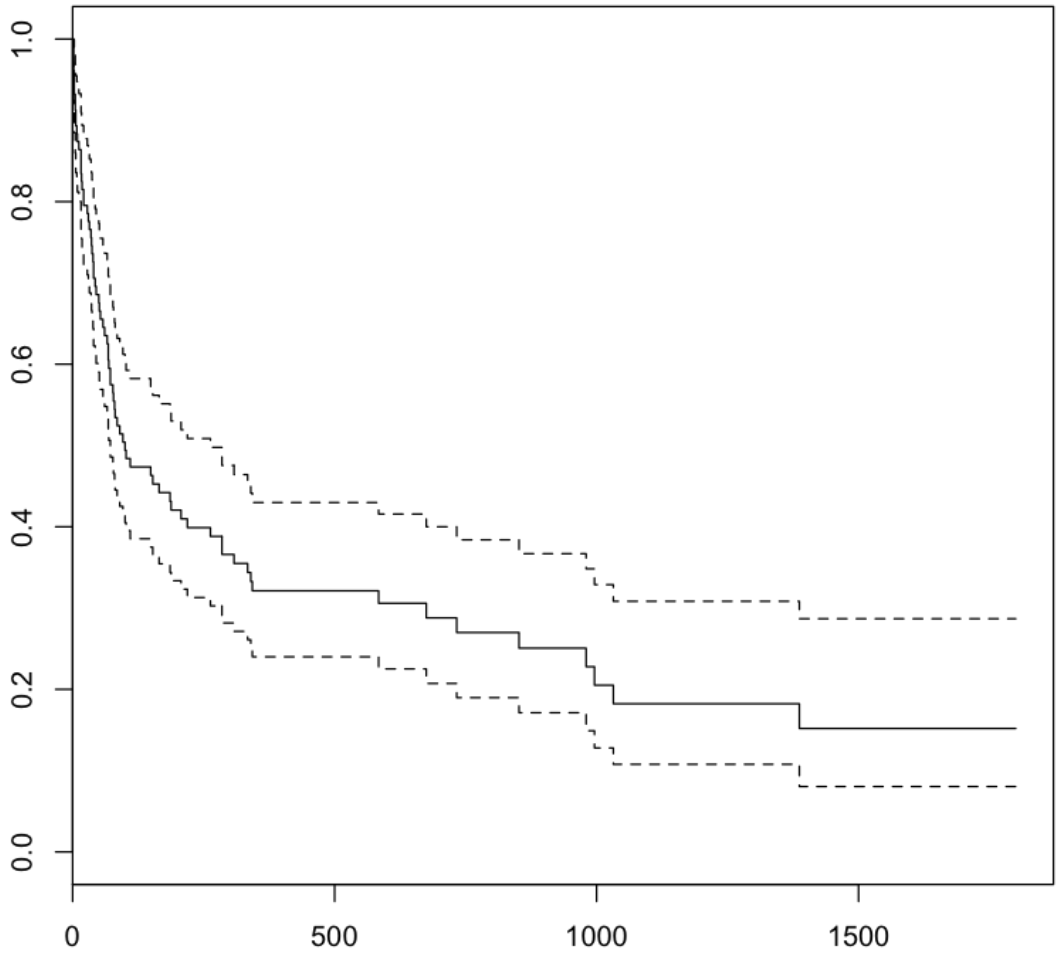
In [222]:

```
km<-survfit(Surv(start,stop,event)~1,data=heart)
# details on the fit for the kaplan-meier curve
# impt var are n.risk and n.event
summary(km, censored=TRUE)
```

```
Call: survfit(formula = Surv(start, stop, event) ~ 1, data = heart)

   time n.risk n.event censored survival std.err lower 95% CI upper 9
5% CI
    0.0      0       0        0    1.000 0.00000       1.0000
1.000
    1.0    103       1        2    0.990 0.00966       0.9715
1.000
    2.0    102       3        3    0.961 0.01904       0.9246
0.999
    3.0     99       3        3    0.932 0.02480       0.8847
0.982
    4.0     96       0        2    0.932 0.02480       0.8847
0.982
    4.5     96       0        1    0.932 0.02480       0.8847
0.982
    5.0     96       2        2    0.913 0.02782       0.8597
0.969
    6.0     94       2        1    0.893 0.03043       0.8355
0.955
```

In [216]:

```
1 plot(km)
```



Cox proporitional hazard

Objective: *describe the hazard rate $\lambda(t)$ as a function of predictors*

Cox proporitional hazard model estimate of the hazard rate is defined as follows:

$$\lambda(t) = \lambda_0(t)e^{\beta_1 x_1 + \beta_2 x_2 + .. + \beta_p x_p}$$

where λ_0 is the baseline hazard, which corresponds to the value of the hazard when all predictors are equal to zero

In [217]:

```
*its a cox proportional hazard model, where age, surgery, transplat variables are use
<-coxph(Surv(start,stop,event)~age+surgery+transplant,data=heart)
mary(cox)
```

```
Call:
coxph(formula = Surv(start, stop, event) ~ age + surgery + transplant,

    data = heart)

  n= 172, number of events= 75

               coef exp(coef) se(coef)      z Pr(>|z|)
age         0.03026   1.03072  0.01397  2.166   0.0303 *
surgery    -0.77139   0.46237  0.35966 -2.145   0.0320 *
transplant  0.01970   1.01990  0.30823  0.064   0.9490
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

           exp(coef) exp(-coef) lower .95 upper .95
age           1.0307     0.9702    1.0029    1.0593
surgery       0.4624     2.1628    0.2285    0.9357
transplant    1.0199     0.9805    0.5574    1.8661

Concordance= 0.599  (se = 0.037 )
Rsquare= 0.06   (max possible= 0.969 )
Likelihood ratio test= 10.55  on 3 df,   p=0.01
Wald test            = 9.53  on 3 df,   p=0.02
Score (logrank) test = 9.85  on 3 df,   p=0.02
```

In [220]:

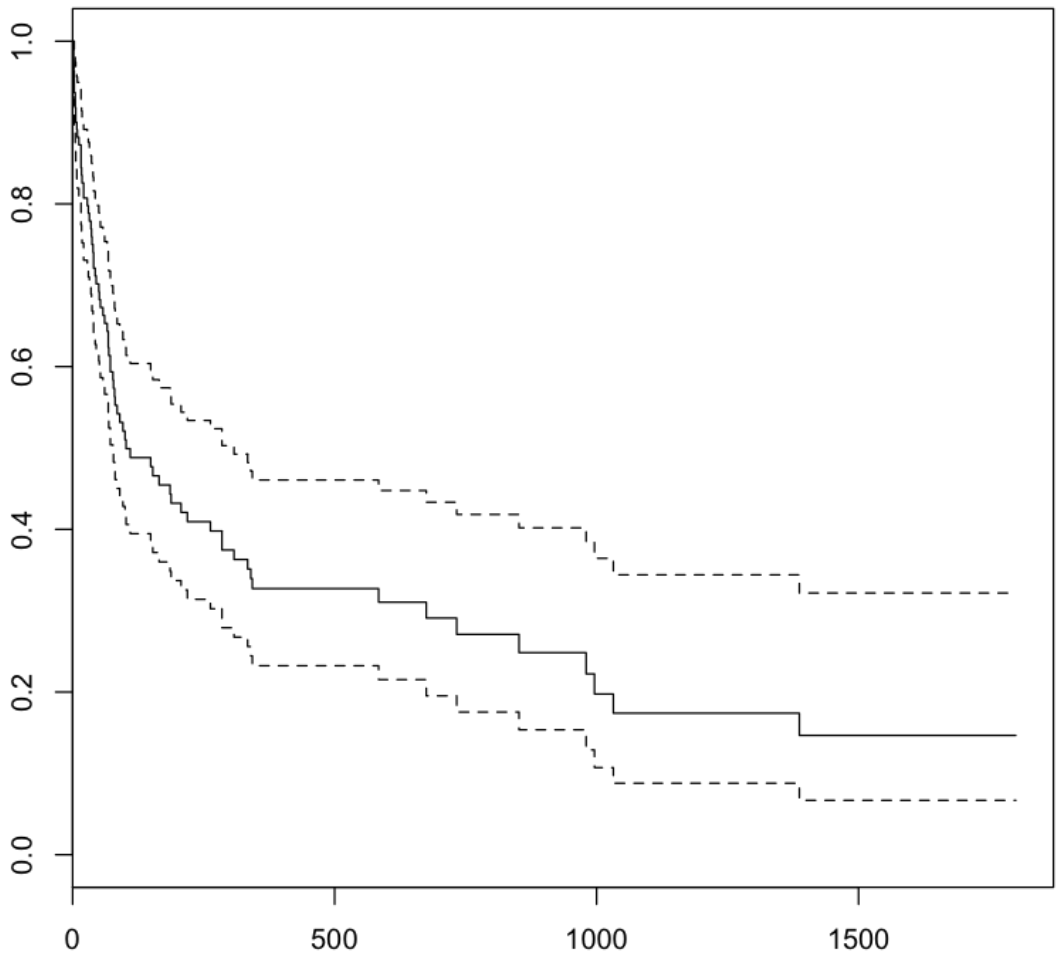
```
1 summary(survfit(cox))
```

```
Call: survfit(formula = cox)

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1    103       1    0.991 0.00911        0.9732        1.000
    2    102       3    0.964 0.01837        0.9285        1.000
    3     99       3    0.937 0.02446        0.8899        0.986
    5     96       2    0.918 0.02774        0.8656        0.974
    6     94       2    0.900 0.03064        0.8421        0.962
    8     92       1    0.891 0.03197        0.8305        0.956
    9     91       1    0.882 0.03322        0.8191        0.949
   12     89       1    0.873 0.03441        0.8077        0.943
   16     88       3    0.845 0.03757        0.7740        0.921
   17     85       1    0.835 0.03854        0.7629        0.914
   18     84       1    0.826 0.03943        0.7520        0.907
   21     83       2    0.807 0.04096        0.7307        0.892
   28     81       1    0.798 0.04158        0.7204        0.884
   30     80       1    0.788 0.04216        0.7100        0.876
   32     78       1    0.779 0.04272        0.6995        0.867
   35     77       1    0.769 0.04322        0.6891        0.859
   36     76       1    0.760 0.04370        0.6787        0.850
   37     75       1    0.750 0.04414        0.6684        0.842
   39     74       1    0.740 0.04454        0.6581        0.833
   40     72       2    0.721 0.04533        0.6374        0.816
   43     70       1    0.711 0.04569        0.6272        0.807
   45     69       1    0.702 0.04606        0.6170        0.798
   50     68       1    0.692 0.04640        0.6067        0.789
   51     67       1    0.682 0.04674        0.5966        0.780
   53     66       1    0.673 0.04706        0.5864        0.771
   58     65       1    0.663 0.04738        0.5762        0.763
   61     64       1    0.653 0.04770        0.5660        0.754
   66     63       1    0.643 0.04802        0.5558        0.745
   68     62       2    0.624 0.04867        0.5351        0.727
   69     60       1    0.614 0.04900        0.5246        0.717
   72     59       2    0.594 0.04965        0.5038        0.699
   77     57       1    0.583 0.04998        0.4933        0.690
   78     56       1    0.573 0.05031        0.4826        0.681
   80     55       1    0.563 0.05063        0.4719        0.671
   81     54       1    0.552 0.05095        0.4611        0.662
   85     53       1    0.542 0.05129        0.4502        0.652
   90     52       1    0.531 0.05162        0.4393        0.643
   96     51       1    0.521 0.05195        0.4283        0.633
  100     50       1    0.510 0.05229        0.4171        0.623
  102     49       1    0.499 0.05261        0.4060        0.614
  110     47       1    0.488 0.05296        0.3946        0.604
  149     45       1    0.477 0.05334        0.3831        0.594
  153     44       1    0.466 0.05372        0.3715        0.584
  165     43       1    0.455 0.05409        0.3599        0.574
  186     41       1    0.443 0.05446        0.3484        0.564
  188     40       1    0.432 0.05480        0.3369        0.554
  207     39       1    0.421 0.05511        0.3255        0.544
  219     38       1    0.409 0.05544        0.3140        0.534
  263     37       1    0.398 0.05574        0.3024        0.524
  285     35       2    0.375 0.05630        0.2790        0.503
  308     33       1    0.363 0.05650        0.2675        0.492
  334     32       1    0.351 0.05672        0.2559        0.482
  340     31       1    0.339 0.05688        0.2444        0.471
  343     29       1    0.327 0.05712        0.2323        0.461
```

```
  584     21       1    0.310 0.05799        0.2153        0.448
  675     17       1    0.291 0.05916        0.1953        0.433
  733     16       1    0.271 0.06005        0.1754        0.418
  852     14       1    0.248 0.06095        0.1536        0.402
  980     11       1    0.222 0.06191        0.1288        0.384
  996     10       1    0.198 0.06170        0.1071        0.364
 1032      9       1    0.174 0.06053        0.0880        0.344
 1387      6       1    0.147 0.05879        0.0669        0.322
```

In [218]:

```
1 # plot survival fit for cox model
2 plot(survfit(cox))
```



In [ ]:

```
1
```