Homework 7

**Chinki**

**May 8, 2017**

**The strenght of concrete with Artificial Neural Network**

ANN is the model that could reliably predict concrete strength given a listing of the composition of the input materials.

**Step 1: Collecting the data**

I am utilising data from UCI machine learning Data Repository.according to website the concrete dataset contains 1,030 examples of concrete with eight features describing the components used in the mixture.

**Step 2: Expolring & preparing the data**

# read in data and examine structure  
concrete <- read.csv("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml11/concrete.csv")  
str(concrete)

## 'data.frame': 1030 obs. of 9 variables:  
## $ cement : num 141 169 250 266 155 ...  
## $ slag : num 212 42.2 0 114 183.4 ...  
## $ ash : num 0 124.3 95.7 0 0 ...  
## $ water : num 204 158 187 228 193 ...  
## $ superplastic: num 0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...  
## $ coarseagg : num 972 1081 957 932 1047 ...  
## $ fineagg : num 748 796 861 670 697 ...  
## $ age : int 28 14 28 28 28 90 7 56 28 28 ...  
## $ strength : num 29.9 23.5 29.2 45.9 18.3 ...

We have 9 variables in which 8 are inputs & 1 strenght is output variable. In this example, we have 8 different features. I am going to normalize my data sets because I want all variables in the same scale.

# custom normalization function  
normalize <- function(x) {   
 return((x - min(x)) / (max(x) - min(x)))  
}

# apply normalization to entire data frame  
concrete\_norm <- as.data.frame(lapply(concrete, normalize))

# confirm that the range is now between zero and one  
summary(concrete\_norm$strength)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.2664 0.4001 0.4172 0.5457 1.0000

Getting summary of the original data.

# compared to the original minimum and maximum  
summary(concrete$strength)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2.33 23.71 34.44 35.82 46.14 82.60

Creating training & test data, training to build model & test to check model performance.

# create training and test data  
concrete\_train <- concrete\_norm[1:773, ]  
concrete\_test <- concrete\_norm[774:1030, ]

**Step 3: Training a model on the data**

I am using package neuralnet for neural network algorithm.

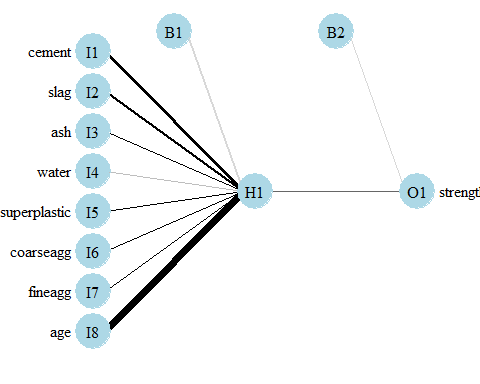
# train the neuralnet model  
library(neuralnet)  
# simple ANN with only a single hidden neuron  
set.seed(12345) # to guarantee repeatable results  
concrete\_model <- neuralnet(formula = strength ~ cement + slag +  
 ash + water + superplastic +   
 coarseagg + fineagg + age,  
 data = concrete\_train)

Visualization of neural topology using plot function.

# visualize the network topology  
plot(concrete\_model)

In this model there is one singal input node with the eight features followed by singal hidden node & singal output. R reports the number of training steps and an error measure called the Sum of Squared Errors (SSE), which as you might expect, is the sum of the squared predicted minus actual values. A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data. Let's try some better plot.

# alternative plot  
library(NeuralNetTools)  
# plotnet  
par(mar = numeric(4), family = 'serif')  
plotnet(concrete\_model, alpha = 0.6)



Thick black line represent higher weighted feature.Gray line is for the negative weighted feature. I represent features, H represent hidden node & O stands for output.B is for the baised parameters.

**Step 4: Evaluating model performance**

The compute() function returns a list with two components: $neurons, which stores the neurons for each layer in the network, and $net.result, which stores the predicted values.It is numerical prediction problem rather than classification. I will measure correlation between predicted value & true value.

# obtain model results  
model\_results <- compute(concrete\_model, concrete\_test[1:8])  
# obtain predicted strength values  
predicted\_strength <- model\_results$net.result  
# examine the correlation between predicted and actual values  
cor(predicted\_strength, concrete\_test$strength)

## [,1]  
## [1,] 0.8064655576

The predicted and the true values are 80.65% correlated.

# produce actual predictions by   
head(predicted\_strength)

## [,1]  
## 774 0.3258991537  
## 775 0.4677425372  
## 776 0.2370268181  
## 777 0.6718811029  
## 778 0.4663428766  
## 779 0.4685272270

I am getting normalised strenght of concerete.

concrete\_train\_original\_strength <- concrete[1:773,"strength"]  
  
strength\_min <- min(concrete\_train\_original\_strength)  
strength\_max <- max(concrete\_train\_original\_strength)

head(concrete\_train\_original\_strength)

## [1] 29.89 23.51 29.22 45.85 18.29 21.86

I am unnormalizing concrete strenght.

# custom normalization function  
unnormalize <- function(x, min, max) {   
 return( (max - min)\*x + min )  
}

strength\_pred <- unnormalize(predicted\_strength, strength\_min, strength\_max)  
strength\_pred

## [,1]  
## 774 28.212910787  
## 775 39.478112301  
## 776 21.154669896  
## 777 55.690797192  
## 778 39.366951260  
## 779 39.540432369  
## 780 39.928033889  
## 781 49.040354523  
## 782 27.907103923  
## 783 20.321053693  
## 784 7.266385231  
## 785 46.876247191  
## 786 48.856855398  
## 787 46.703583933  
## 788 45.448376324  
## 789 51.065068074  
## 790 43.680671652  
## 791 28.193980938  
## 792 28.827103782  
## 793 18.345241127  
## 794 18.252462770  
## 795 15.643701142  
## 796 17.801483462  
## 797 23.682825829  
## 798 13.976073367  
## 799 12.194355970  
## 800 33.449926982  
## 801 25.088099888  
## 802 52.607784390  
## 803 24.433849292  
## 804 34.735202333  
## 805 44.689978758  
## 806 43.020408989  
## 807 33.080435982  
## 808 34.849702382  
## 809 47.563249060  
## 810 6.938091659  
## 811 29.327314842  
## 812 52.193794638  
## 813 18.724170879  
## 814 23.403183011  
## 815 31.805906354  
## 816 26.771592010  
## 817 54.848833698  
## 818 26.325671398  
## 819 18.271643621  
## 820 21.589603938  
## 821 19.723934696  
## 822 24.236493867  
## 823 26.151069034  
## 824 44.930065559  
## 825 45.984424330  
## 826 36.569851275  
## 827 18.383419764  
## 828 37.547970377  
## 829 44.080234009  
## 830 17.365663875  
## 831 49.764429240  
## 832 23.644245310  
## 833 18.886834859  
## 834 46.263011829  
## 835 54.475179301  
## 836 20.727657800  
## 837 50.093261598  
## 838 35.938877950  
## 839 55.151231281  
## 840 43.539564013  
## 841 20.696074238  
## 842 44.549415622  
## 843 24.482322674  
## 844 16.201399197  
## 845 42.375778976  
## 846 45.270976016  
## 847 44.638527605  
## 848 10.311700822  
## 849 46.304752045  
## 850 19.307530759  
## 851 55.737413630  
## 852 24.793017815  
## 853 23.814544656  
## 854 33.546539494  
## 855 42.695335181  
## 856 49.811424381  
## 857 48.652348988  
## 858 52.638858641  
## 859 53.166049496  
## 860 24.183085073  
## 861 41.825425595  
## 862 54.927075804  
## 863 51.251536938  
## 864 53.452210772  
## 865 49.787431937  
## 866 29.885878058  
## 867 34.712601224  
## 868 44.396238610  
## 869 15.761370235  
## 870 51.046835680  
## 871 9.046006076  
## 872 18.467300274  
## 873 47.763396508  
## 874 16.520661292  
## 875 46.236355997  
## 876 44.008915499  
## 877 15.788252948  
## 878 35.338644057  
## 879 55.721030895  
## 880 26.239781748  
## 881 54.219040311  
## 882 31.887794120  
## 883 28.775965442  
## 884 21.828939255  
## 885 49.764429240  
## 886 46.302614755  
## 887 44.990113139  
## 888 43.872794257  
## 889 33.613802430  
## 890 49.821748527  
## 891 22.489345534  
## 892 54.731284069  
## 893 53.523149545  
## 894 48.268894377  
## 895 39.425593582  
## 896 25.287008255  
## 897 34.585350422  
## 898 43.126060379  
## 899 46.312025009  
## 900 43.287860282  
## 901 54.227588359  
## 902 55.759561064  
## 903 55.126429340  
## 904 41.980980168  
## 905 23.770483604  
## 906 31.495206543  
## 907 30.872026632  
## 908 45.319018412  
## 909 48.325099603  
## 910 46.844044826  
## 911 16.395384725  
## 912 49.033749656  
## 913 35.609382272  
## 914 53.447559068  
## 915 28.490260269  
## 916 26.097392370  
## 917 34.553462242  
## 918 18.034082548  
## 919 46.017644897  
## 920 55.756594314  
## 921 19.113094271  
## 922 45.880359504  
## 923 33.667746328  
## 924 15.840638297  
## 925 15.250768816  
## 926 25.967077829  
## 927 49.040327527  
## 928 20.250009925  
## 929 53.463888607  
## 930 39.712945454  
## 931 18.635100239  
## 932 43.052489413  
## 933 36.781496963  
## 934 47.367089782  
## 935 36.459125460  
## 936 50.202821978  
## 937 49.768635382  
## 938 47.960542410  
## 939 23.039593502  
## 940 44.787449396  
## 941 22.994965622  
## 942 52.115231571  
## 943 30.231029721  
## 944 46.490832321  
## 945 13.518827249  
## 946 47.463311002  
## 947 33.652021124  
## 948 36.959189963  
## 949 42.675567254  
## 950 43.839085507  
## 951 55.441831705  
## 952 55.730302275  
## 953 42.495361138  
## 954 45.995593485  
## 955 14.559033517  
## 956 53.889321097  
## 957 47.638332031  
## 958 55.756976692  
## 959 14.065750036  
## 960 43.086479175  
## 961 27.286788837  
## 962 21.414631033  
## 963 14.686461194  
## 964 49.955191984  
## 965 41.489814802  
## 966 50.757732386  
## 967 36.406258094  
## 968 50.903995407  
## 969 18.617655592  
## 970 42.779147183  
## 971 27.548360275  
## 972 55.198346170  
## 973 21.614221641  
## 974 43.774007128  
## 975 33.582782000  
## 976 34.494050231  
## 977 31.427147694  
## 978 30.748169174  
## 979 41.713472991  
## 980 52.745480397  
## 981 48.698035754  
## 982 48.463177370  
## 983 39.479649564  
## 984 43.545565848  
## 985 51.163772080  
## 986 55.097430829  
## 987 16.960833436  
## 988 21.228459298  
## 989 16.372967444  
## 990 45.699914067  
## 991 40.015149409  
## 992 54.915041504  
## 993 52.363061591  
## 994 46.703583933  
## 995 22.971496748  
## 996 55.757710774  
## 997 23.665277416  
## 998 30.203783047  
## 999 38.388908212  
## 1000 31.819252685  
## 1001 44.736636088  
## 1002 20.515435999  
## 1003 43.646913655  
## 1004 53.414481990  
## 1005 24.202915410  
## 1006 49.321113911  
## 1007 15.705157900  
## 1008 28.509942443  
## 1009 51.916787243  
## 1010 27.794472365  
## 1011 15.938227741  
## 1012 31.463715541  
## 1013 44.909197894  
## 1014 48.094743404  
## 1015 29.573686435  
## 1016 32.339635939  
## 1017 21.638918442  
## 1018 54.759105089  
## 1019 20.858741569  
## 1020 30.157137857  
## 1021 10.357941496  
## 1022 25.792662773  
## 1023 17.857069663  
## 1024 12.193204624  
## 1025 27.194717549  
## 1026 17.787728300  
## 1027 52.970583241  
## 1028 41.241771843  
## 1029 55.704290534  
## 1030 46.250018028

**Step 5: Improving model performance**

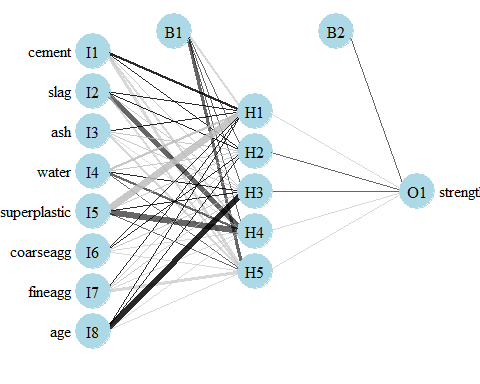
Now, I am increasing number of headen nodes for more accuracy.I am going to use 5 hidden nodes.

# a more complex neural network topology with 5 hidden neurons  
set.seed(12345) # to guarantee repeatable results  
concrete\_model2 <- neuralnet(strength ~ cement + slag +  
 ash + water + superplastic +   
 coarseagg + fineagg + age,  
 data = concrete\_train, hidden = 5, act.fct = "logistic")

Grafical representation of the model.

# plot the network  
plot(concrete\_model2)

# plotnet  
par(mar = numeric(4), family = 'serif')  
plotnet(concrete\_model2, alpha = 0.6)



# evaluate the results as we did before  
model\_results2 <- compute(concrete\_model2, concrete\_test[1:8])  
predicted\_strength2 <- model\_results2$net.result  
cor(predicted\_strength2, concrete\_test$strength)

## [,1]  
## [1,] 0.9244533426

The correlation between predicted & true value is 92.46% , which is better than the previous 1 hidden node.Notice that the reported error (measured again by SSE) has been reduced from 5.08 in the previous model to 1.63 here. Additionally, the number of training steps rose from 4,882 to 86,849, which should come as no surprise given how much more complex the model has become. Lets try different hidden nodes.

# try different activation function  
# a more complex neural network topology with 5 hidden neurons  
set.seed(12345) # to guarantee repeatable results  
concrete\_model2 <- neuralnet(strength ~ cement + slag +  
 ash + water + superplastic +   
 coarseagg + fineagg + age,  
 data = concrete\_train, hidden = 5, act.fct = "tanh")

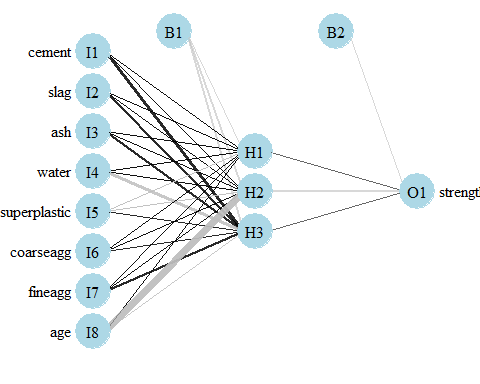
# evaluate the results as we did before  
model\_results2 <- compute(concrete\_model2, concrete\_test[1:8])  
predicted\_strength2 <- model\_results2$net.result  
cor(predicted\_strength2, concrete\_test$strength)

## [,1]  
## [1,] 0.5741729322

# a more complex neural network topology with 3 hidden neurons  
set.seed(12345) # to guarantee repeatable results  
concrete\_model2 <- neuralnet(strength ~ cement + slag +  
 ash + water + superplastic +   
 coarseagg + fineagg + age,  
 data = concrete\_train, hidden = 3, act.fct = "logistic")

# plot the network  
plot(concrete\_model2)

# plotnet  
par(mar = numeric(4), family = 'serif')  
plotnet(concrete\_model2, alpha = 0.6)



**Step 5: Improving model performance**

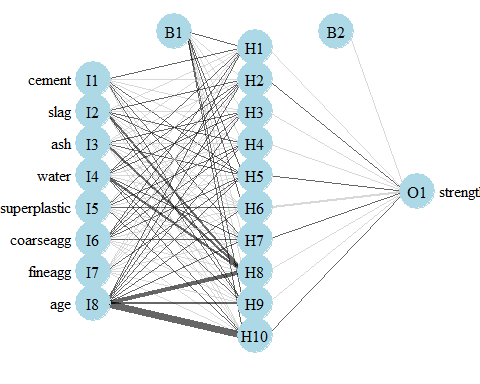
# evaluate the results as we did before  
model\_results2 <- compute(concrete\_model2, concrete\_test[1:8])  
predicted\_strength2 <- model\_results2$net.result  
cor(predicted\_strength2, concrete\_test$strength)

## [,1]  
## [1,] 0.9260549266

set.seed(12345) # to guarantee repeatable results  
concrete\_model2 <- neuralnet(strength ~ cement + slag +  
 ash + water + superplastic +   
 coarseagg + fineagg + age,  
 data = concrete\_train, hidden = 10, act.fct = "logistic")

# plot the network  
plot(concrete\_model2)

# plotnet  
par(mar = numeric(4), family = 'serif')  
plotnet(concrete\_model2, alpha = 0.6)



# evaluate the results as we did before  
model\_results2 <- compute(concrete\_model2, concrete\_test[1:8])  
predicted\_strength2 <- model\_results2$net.result  
cor(predicted\_strength2, concrete\_test$strength)

## [,1]  
## [1,] 0.9363509159

I am getting more correlation for hidden node 10, which is 93.64%.

Lets try H20 library code:

# using h2o deeplearning  
library(h2o)

##   
## ----------------------------------------------------------------------  
##   
## Your next step is to start H2O:  
## > h2o.init()  
##   
## For H2O package documentation, ask for help:  
## > ??h2o  
##   
## After starting H2O, you can use the Web UI at http://localhost:54321  
## For more information visit http://docs.h2o.ai  
##   
## ----------------------------------------------------------------------

##   
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':  
##   
## cor, sd, var

## The following objects are masked from 'package:base':  
##   
## %\*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,  
## colnames<-, ifelse, is.character, is.factor, is.numeric, log,  
## log10, log1p, log2, round, signif, trunc

h2o.init()

## Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 7 minutes 24 seconds   
## H2O cluster version: 3.10.4.6   
## H2O cluster version age: 23 days   
## H2O cluster name: H2O\_started\_from\_R\_chink\_dnr700   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 0.85 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 2   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## R Version: R version 3.3.3 (2017-03-06)

concrete.hex <- h2o.importFile("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml11/concrete.csv")

##   
 |   
 | | 0%  
 |   
 |=================================================================| 100%

summary(concrete.hex)

## Warning in summary.H2OFrame(concrete.hex): Approximated quantiles  
## computed! If you are interested in exact quantiles, please pass the  
## `exact\_quantiles=TRUE` parameter.

## cement slag ash water   
## Min. :102.0 Min. : 0.00 Min. : 0.00 Min. :121.8   
## 1st Qu.:192.1 1st Qu.: 0.00 1st Qu.: 0.00 1st Qu.:164.9   
## Median :272.6 Median : 21.92 Median : 0.00 Median :184.9   
## Mean :281.2 Mean : 73.90 Mean : 54.19 Mean :181.6   
## 3rd Qu.:349.9 3rd Qu.:142.68 3rd Qu.:118.26 3rd Qu.:191.9   
## Max. :540.0 Max. :359.40 Max. :200.10 Max. :247.0   
## superplastic coarseagg fineagg age   
## Min. : 0.000 Min. : 801.0 Min. :594.0 Min. : 1.00   
## 1st Qu.: 0.000 1st Qu.: 931.7 1st Qu.:730.8 1st Qu.: 7.00   
## Median : 6.376 Median : 967.8 Median :779.1 Median : 28.00   
## Mean : 6.205 Mean : 972.9 Mean :773.6 Mean : 45.66   
## 3rd Qu.:10.175 3rd Qu.:1029.1 3rd Qu.:824.0 3rd Qu.: 56.00   
## Max. :32.200 Max. :1145.0 Max. :992.6 Max. :365.00   
## strength   
## Min. : 2.33   
## 1st Qu.:23.68   
## Median :34.40   
## Mean :35.82   
## 3rd Qu.:46.10   
## Max. :82.60

splits <- h2o.splitFrame(concrete.hex, 0.75, seed=1234)  
dl <- h2o.deeplearning(x=1:8,y="strength",training\_frame=splits[[1]],activation = "Tanh",   
 hidden = c(200,200), distribution = "gaussian")

##   
 |   
 | | 0%  
 |   
 |====== | 10%  
 |   
 |==================== | 30%  
 |   
 |========================== | 40%  
 |   
 |======================================= | 60%  
 |   
 |==================================================== | 80%  
 |   
 |========================================================== | 90%  
 |   
 |=================================================================| 100%

dl.predict <- h2o.predict (dl, splits[[2]])

##   
 |   
 | | 0%  
 |   
 |=================================================================| 100%

cor(as.vector(dl.predict), as.vector(splits[[2]]$strength))

## [1] 0.8946682797

Correlation of predicted and true value is 88.31%.

dl@parameters

## $model\_id  
## [1] "DeepLearning\_model\_R\_1495315247302\_2"  
##   
## $training\_frame  
## [1] "RTMP\_sid\_927a\_2"  
##   
## $activation  
## [1] "Tanh"  
##   
## $seed  
## [1] -4641885755765058560  
##   
## $distribution  
## [1] "gaussian"  
##   
## $x  
## [1] "cement" "slag" "ash" "water"   
## [5] "superplastic" "coarseagg" "fineagg" "age"   
##   
## $y  
## [1] "strength"

h2o.performance(dl)

## H2ORegressionMetrics: deeplearning  
## \*\* Reported on training data. \*\*  
## \*\* Metrics reported on full training frame \*\*  
##   
## MSE: 48.15312245  
## RMSE: 6.939245092  
## MAE: 5.448877749  
## RMSLE: 0.2395121539  
## Mean Residual Deviance : 48.15312245

h2o.shutdown()

## Are you sure you want to shutdown the H2O instance running at http://localhost:54321/ (Y/N)?

**The stock Market Data**

The Smarket data,is part of the ISLR library.This data set consists of percentage returns for the S&P 500 stock index over 1,250 days, from the beginning of 2001 until the end of 2005.

library(ISLR)  
names(Smarket)

## [1] "Year" "Lag1" "Lag2" "Lag3" "Lag4" "Lag5"   
## [7] "Volume" "Today" "Direction"

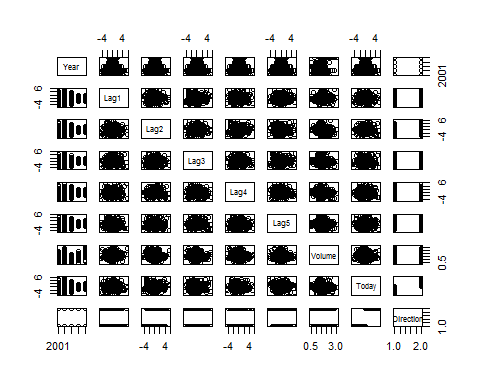
#Dimention of dataset  
dim(Smarket)

## [1] 1250 9

#Getting summary of dataset  
summary(Smarket)

## Year Lag1 Lag2   
## Min. :2001 Min. :-4.922000 Min. :-4.922000   
## 1st Qu.:2002 1st Qu.:-0.639500 1st Qu.:-0.639500   
## Median :2003 Median : 0.039000 Median : 0.039000   
## Mean :2003 Mean : 0.003834 Mean : 0.003919   
## 3rd Qu.:2004 3rd Qu.: 0.596750 3rd Qu.: 0.596750   
## Max. :2005 Max. : 5.733000 Max. : 5.733000   
## Lag3 Lag4 Lag5   
## Min. :-4.922000 Min. :-4.922000 Min. :-4.92200   
## 1st Qu.:-0.640000 1st Qu.:-0.640000 1st Qu.:-0.64000   
## Median : 0.038500 Median : 0.038500 Median : 0.03850   
## Mean : 0.001716 Mean : 0.001636 Mean : 0.00561   
## 3rd Qu.: 0.596750 3rd Qu.: 0.596750 3rd Qu.: 0.59700   
## Max. : 5.733000 Max. : 5.733000 Max. : 5.73300   
## Volume Today Direction   
## Min. :0.3561 Min. :-4.922000 Down:602   
## 1st Qu.:1.2574 1st Qu.:-0.639500 Up :648   
## Median :1.4229 Median : 0.038500   
## Mean :1.4783 Mean : 0.003138   
## 3rd Qu.:1.6417 3rd Qu.: 0.596750   
## Max. :3.1525 Max. : 5.733000

#Scatter plot of all variables of Dataset  
pairs(Smarket)



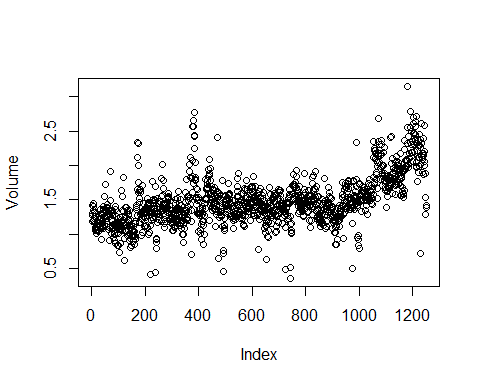
The cor() function produces a matrix that contains all of the pairwise correlations among the predictors in a data set.

#Correlation matrix  
cor(Smarket[,-9])

## Year Lag1 Lag2 Lag3 Lag4  
## Year 1.00000000 0.029699649 0.030596422 0.033194581 0.035688718  
## Lag1 0.02969965 1.000000000 -0.026294328 -0.010803402 -0.002985911  
## Lag2 0.03059642 -0.026294328 1.000000000 -0.025896670 -0.010853533  
## Lag3 0.03319458 -0.010803402 -0.025896670 1.000000000 -0.024051036  
## Lag4 0.03568872 -0.002985911 -0.010853533 -0.024051036 1.000000000  
## Lag5 0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641  
## Volume 0.53900647 0.040909908 -0.043383215 -0.041823686 -0.048414246  
## Today 0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527  
## Lag5 Volume Today  
## Year 0.029787995 0.53900647 0.030095229  
## Lag1 -0.005674606 0.04090991 -0.026155045  
## Lag2 -0.003557949 -0.04338321 -0.010250033  
## Lag3 -0.018808338 -0.04182369 -0.002447647  
## Lag4 -0.027083641 -0.04841425 -0.006899527  
## Lag5 1.000000000 -0.02200231 -0.034860083  
## Volume -0.022002315 1.00000000 0.014591823  
## Today -0.034860083 0.01459182 1.000000000

As one would expect, the correlations between the lag variables and today's returns are close to zero. In other words, there appears to be little correlation between today's returns and previous days' returns. The only substantial correlation is between Year and Volume.

#Scatterplot of volume  
attach(Smarket)  
plot(Volume)



**Logistic Regression**

#Running logistic regression  
glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume , data=Smarket ,family=binomial )  
glm.fit

##   
## Call: glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +   
## Volume, family = binomial, data = Smarket)  
##   
## Coefficients:  
## (Intercept) Lag1 Lag2 Lag3 Lag4   
## -0.126000 -0.073074 -0.042301 0.011085 0.009359   
## Lag5 Volume   
## 0.010313 0.135441   
##   
## Degrees of Freedom: 1249 Total (i.e. Null); 1243 Residual  
## Null Deviance: 1731   
## Residual Deviance: 1728 AIC: 1742

#getting summary  
summary(glm.fit)

##   
## Call:  
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +   
## Volume, family = binomial, data = Smarket)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.446 -1.203 1.065 1.145 1.326   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -0.126000 0.240736 -0.523 0.601  
## Lag1 -0.073074 0.050167 -1.457 0.145  
## Lag2 -0.042301 0.050086 -0.845 0.398  
## Lag3 0.011085 0.049939 0.222 0.824  
## Lag4 0.009359 0.049974 0.187 0.851  
## Lag5 0.010313 0.049511 0.208 0.835  
## Volume 0.135441 0.158360 0.855 0.392  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 1731.2 on 1249 degrees of freedom  
## Residual deviance: 1727.6 on 1243 degrees of freedom  
## AIC: 1741.6  
##   
## Number of Fisher Scoring iterations: 3

The predict() function can be used to predict the probability that the market will go up, given values of the predictors.

glm.probs=predict(glm.fit,type="response")  
glm.probs[1:10]

## 1 2 3 4 5 6 7   
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509   
## 8 9 10   
## 0.5092292 0.5176135 0.4888378

#Getting contrast   
contrasts(Direction)

## Up  
## Down 0  
## Up 1

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, Up or Down.

glm.pred=rep("Down",1250)  
glm.pred[glm.probs>0.5]="up"

The first command creates a vector of 1,250 Down elements. The second line transforms to Up all of the elements for which the predicted probability of a market increase exceeds 0.5.

#Creating table   
table(glm.pred ,Direction )

## Direction  
## glm.pred Down Up  
## Down 145 141  
## up 457 507

(507+145)/1250

## [1] 0.5216

mean(glm.pred==Direction )

## [1] 0.116

In this case, logistic regression correctly predicted the movement of the market 52.2%ofthetime.

train=(Year<2005)  
Smarket.2005= Smarket [!train ,]  
dim(Smarket.2005)

## [1] 252 9

Direction.2005= Direction [!train]

The object train is a vector of 1,250 elements, corresponding to the observations in our data set.

glm.fit=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,data = Smarket,family = binomial,subset=train)  
glm.probs=predict(glm.fit,Smarket.2005,type = "response")

glm.pred=rep("Down",252)  
glm.pred[glm.probs>0.5]="up"  
table(glm.pred,Direction.2005)

## Direction.2005  
## glm.pred Down Up  
## Down 77 97  
## up 34 44

mean(glm.pred==Direction.2005)

## [1] 0.3055556

mean(glm.pred!=Direction.2005)

## [1] 0.6944444

glm.fit=glm(Direction~Lag1+Lag2 ,data=Smarket ,family=binomial, subset=train)   
glm.probs=predict(glm.fit ,Smarket.2005, type="response")   
glm.pred=rep("Down",252)  
glm.pred[glm.probs >.5]="up"  
table(glm.pred ,Direction.2005)

## Direction.2005  
## glm.pred Down Up  
## Down 35 35  
## up 76 106

mean(glm.pred==Direction.2005)

## [1] 0.1388889

106/(106+76)

## [1] 0.5824176

predict (glm.fit ,newdata=data.frame(Lag1=c(1.2,1.5), Lag2=c(1.1,-0.8)),type="response")

## 1 2   
## 0.4791462 0.4960939