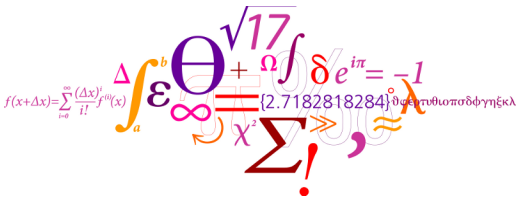
Assignment 1

02582 Computational Data Analysis

2020



Kgs. Lyngby, 2020

Authors:

1. Bjørn Hansen
s193035

2. Olena Dominika Holubowska
s190323

Contents

1 Introduction to the problem

The object of the case 1 is to form a model that predicts a response variable based off of a feature matrix. The data which is used in the assignment to create the model is a 100 by 101 matrix where one column is the response and the other 100 columns are respective features. No previous knowledge is known about the data. From viewing the data it is noted that the response column as well as 94 of the feature columns contain solely numerical values. The final five feature columns consist of five different categorical levels namely, "G", "H", "I", "J", "K", moreover all of these categorical columns contain multiple missing entries.

To achieve the goal of creating a model to predict the response variable the following steps were followed:
1) Identifying missing values.
2) Encoding categorical data.
3) Discover most reliable predictive model.
4) Use model to make predictions and estimate models root mean squared error.

2 Model and method

As mentioned earlier the data set which is being used in this assignment contains 100 observations and 100 features. This is relatively small but quite high in dimensions. The methods used to fit such a data set have to therefor be robust enough to deal with high dimensions but also be able to learn with a limited amount of observations. The methods which have been tested are the following: elastic net regression, regression via back propagating neural network, support vector regression, regression via random forest, gradient boosting with regression trees. In the section below each methods implementation is described in further detail.

2.1 Ridge and Lasso Regression

Regularized methods can be used to select only relevant features in the training dataset. Due to the lower dimensionality of the date the overfitting is prevented, the model is simple and computational efficient. The challenge is to balance between bias-variance trade off. Firstly Ridge regression was performed, which can be seen in appendix 6.2 which performs feature weight updates with extra term containing the squared L2 norm of the weights vector (Figure 2). Secondly lasso regression was implemented as in appendix 6.1 which performs feature weight updates with the L1 norm of the weights vector. During optimisation tends to cause the weights for some features to go all the way down to zero at some point in time (Figure 4). This effectively acts as feature elimination and supposedly eliminates features such that those causing most variance/overfitting are down-weighted and eliminated faster. By adjusting alpha paramiter elastic Net regression was obtained which is the combination of the other two, and the two penalty hyper-parameters also balance between L2 and L1 regularisation (Figure 6).

2.2 Neural Network

The neural network implemented is a so called back propagating network and consists of two hidden layers. The configuration is as follows: 13:5:3:1. The input layer has 13 inputs, the two hidden layers have five and three neurons and the output layer has (since regression is the goal) one single output. The neural network was implemented using the R package "neuralnet".

Neural networks are typically good at understanding anomalies and finding hidden parameters in data. Performance of the given neural network is many times dependent on larger data sets and although this assignment deals with a relatively small data set the neural network was tested anyways.

2.3 Support Vector Regression

Support Vector Machines are typically known for their classification capabilities, but they can also be used as a regression method. Support Vector Regression uses the same principles as the SVM for classification. The main idea is to fit a function *f(x)* to the data points and minimize the error that this function has. To implement support vector regression in this assignment the R library "kernlab".

2.4 Random Forest

Decision trees can be useful as they can be relatively easy to implement, they however have drawbacks such as being prone to over fitting and susceptible to small changes in input. To overcome some of these drawbacks a random forest can be utilized. Random forest is a bagging technique where many decision trees are trained in parallel. The output of the random forest is the mean predicted value (for regression) of the individual trees. In this assignment a random forest containing 100 trees was used with the help of the R library "randomForest".

2.5 Gradient Boosted Regression Trees

Boosting algorithms are in essence numerical optimization problems where the objective is to minimize the loss of the model by adding weight to lower performing "learners" using a gradient descent like procedure. This type of ensemble method is of a boosting nature as it is a stage-wise additive model where trees are added one at a time, existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss of the trees. For this assignment both an adaboost and gradient boosting procedure were tested. These methods are implemented via the R libraries "mboost" and "gdm"

3 Model and method

This section contains an explanation of the chosen methods used in finding the best predicting model. Model selection, missing values and factor handling are topics which are discussed below.

3.1 Model selection

As mentioned earlier various methods will be tested in order to find the best method to fit the given data set for the assignment. Each method will be trained on training data and then tested on test data. The data will be split in a 80:20 (train:test) fashion. Cross validation will also be used for some of the high performing models to insure the performance to accurate, the data fed to each model is also scaled. The root mean squared error (RMSE) will be estimated for each model and compared to the other methods. The method which produces the model with the lowest RMSE will be considered the best for predicting the given data set.

3.2 Evaluation of different approaches for handling missing data

Firstly, we investigate in the properties of the missing data and conclude that we are coping with Missing Completely at Random data. Since the data set is very limited, all methods involving deleting some of the data, was rejected at first. Since we don't have any knowledge about the variables, we decided to keep it simple and replace missing observation by the most common level.

3.3 Evaluation of different approaches to categorical data

The data set includes 4 variables divided in 5 categories: G, I, J, H and K which are not define as nominal or ordinal data. There is a lot of common methods to invert this type of data into numbers. In this case we are assuming the data is nominal and we can use one hot encoding.

3.3.1 Evaluation of One Hot Encoding

The one-hot encoder creates one column for each value to compare against all other values. For each new column, a row gets a 1 if the row contained that column's value and a 0 if it did not.In our data set this can be represented by this example:

95 x_94	96 x_95	97 C_1_G	98 C_1_H	99 C_1_I	100 C_1_J	101 C_1_K
2.3169	2.3352	0	0	0	0	0
-0.1162	-0.9051	0	0	0	0	1
1.4585	0.3953	1	0	0	0	0
0.4045	0.1499	0	0	1	0	0
0.2081	-0.3856	0	0	1	0	0
2.5409	-0.1073	1	0	0	0	0
-1.4192	-0.4713	0	1	0	0	0
-0.8024	0.0533	0	0	0	0	1

This method is used for nominal data. The biggest disadvantage of the method is producing additional columns which will slow down the algorithm but in our case the data set is small enough to apply the method.

4 Results

Presented below is are the results in the form of a bar plot and in table form comparing the RMSE results of each model using both training and test data.

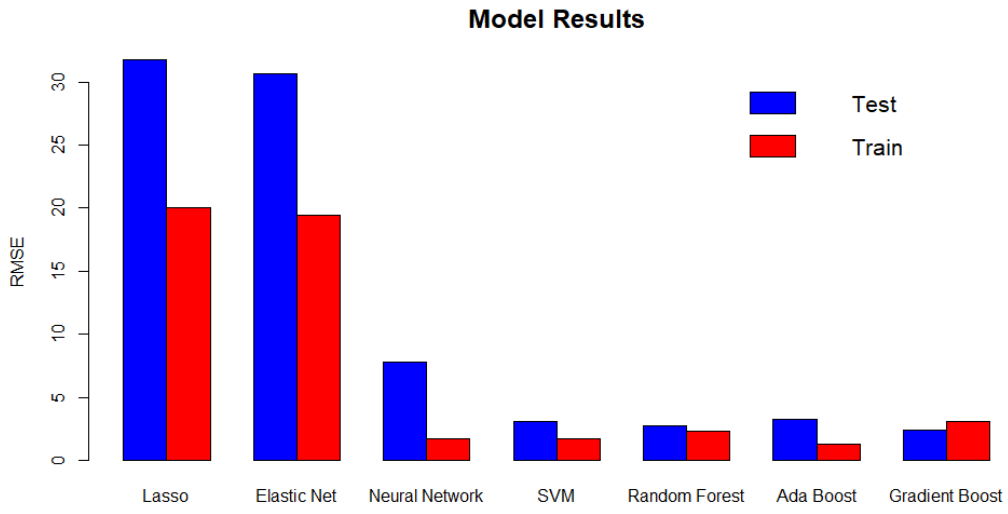


Figure 1: RMSE Results of all methods.

	Lasso	Elastic Net	Neural Network	SVM	Random Forest	Ada Boost	Gradient Boost
Test	31.75	30.69	7.79	3.05	2.73	3.24	2.53
Train	20.06	19.45	1.72	1.73	2.30	1.24	3.03

As is seen from the figures above the best performing methods where the support vector machine, random forest and boosting methods where all had relatively similar results. The worst performing was the lasso and elastic net.

4.1 Ridge/ Lasso Regression

Ridge regression we implemented as can be seen in appendix 6.2. From figure 6 we can find CV lambda = 2.7826 and then calculate mean square error MSE = 19.4597 for training and MSE=30.44 for test.

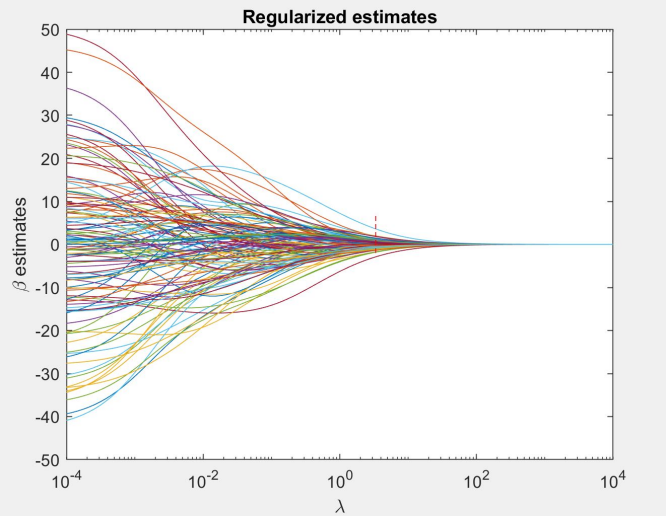


Figure 2: Estimates for ridge regression

The next model was lasso regression where MSE = 31.758161. By tuning alpha parameter we performed also elastic net regression where we found following errors:

$$\begin{array}{lll} \text{MSE} = 30.693078 & \text{DF} = 0 & \text{a} = 0.500000 \\ \text{MSE} = 30.177660 & \text{DF} = 0 & \text{a} = 0.100000 \\ \text{MSE} = 31.348111 & \text{DF} = 0 & \text{a} = 0.900000 \end{array}$$

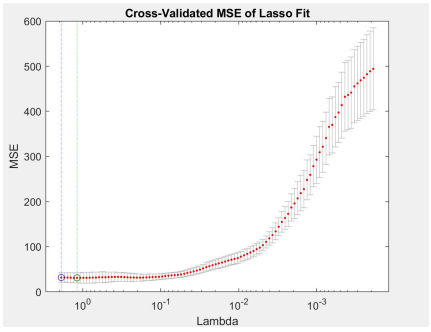


Figure 3: MSE of Lasso Fit

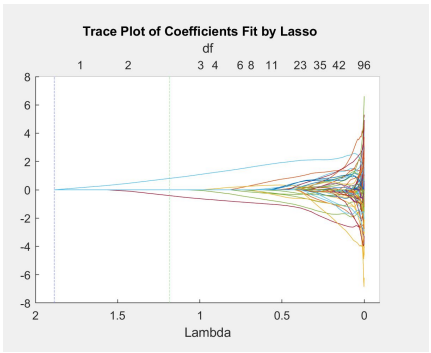


Figure 4: Coefficient fit by lasso

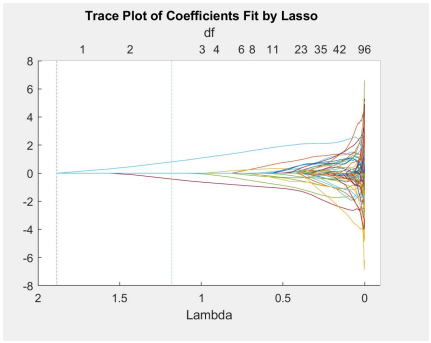


Figure 5: Trace plot for lasso

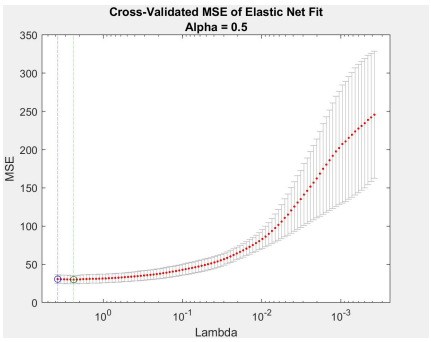


Figure 6: MSE for Elastic net

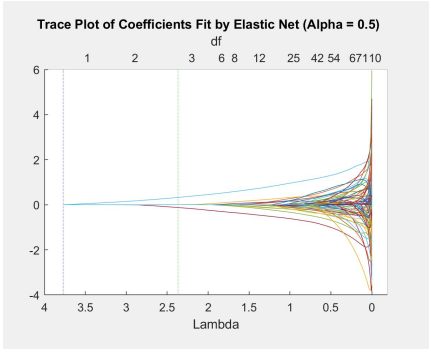


Figure 7: Coefficients for Elastic Net

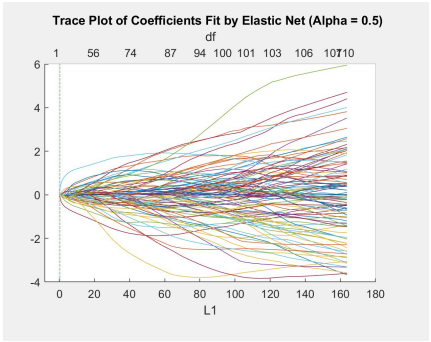


Figure 8: Trace for Elastic Net

4.2 Neural Network

The back propagating neural network had relatively high RMSE on the test data compared to the better performing models but had quite low RMSE on the training data. This can be a sign that the neural network is over fitting to the training data. This is some what expected due to the size of the data set which it was trained on.

4.3 Support Vector Machine

The support vector regression model say quite good results with low RMSE on both the training and test sets. This method was only slightly worse than some of the other tested methods.

4.4 Random Forest

The random forest regression performed well and had very consistent results with both the training and test RMSE being similar. In figure 9 below the error rate is shown diminishing over larger number of trees as anticipated.

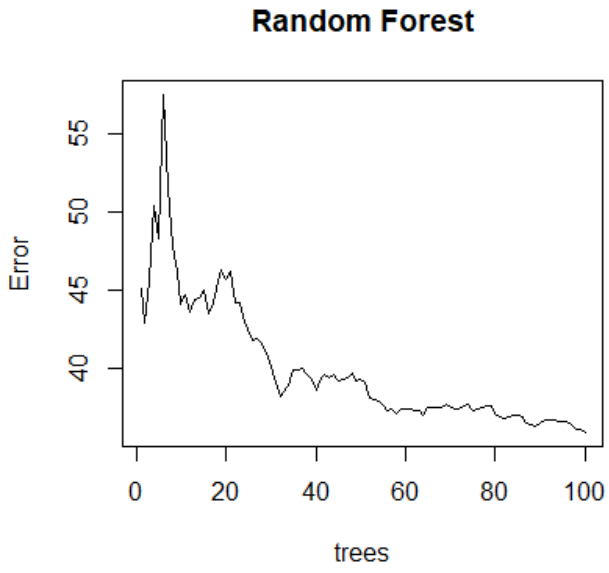


Figure 9: Error vs number of trees

4.5 Ada/ Gradient Boost

Both Ada and Gradient boost models produced low RMSE, the ada boost model had the lowest RMSE on the training data of any of the models and gradient boost had the lowest test RMSE. Surprisingly the RMSE was lower for the test data than for the training data. The RMSE varied only marginally between the train and test data set and cross validation was used for the gradient boosting method, because of this it would seem likely that the gradient boosting method is the best model for the assignment's given data set. To confirm this, the gradient boosting results are analyzed in the section below.

5 Model validation

To Validate the gradient boosting model as being the best predicting model the residuals are analyzed in the figures below.

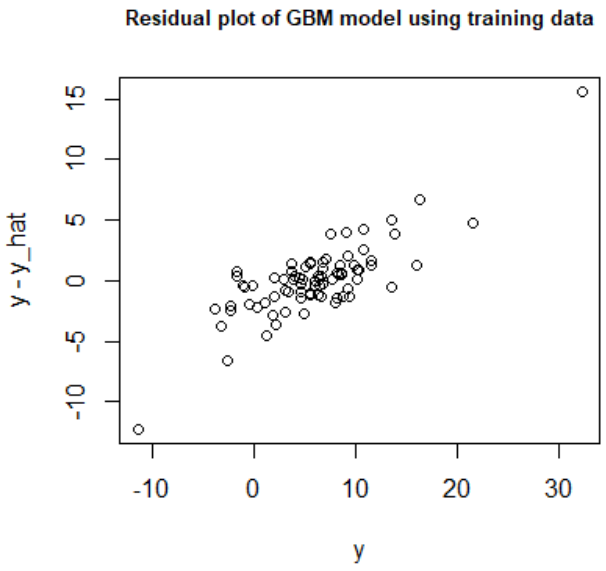


Figure 10: Plot of residuals for the Gradient Boosting Model.

Figure 10 shows that the model has the expected residual plot, where the difference in the real response value and the predicted response value has a linear trend. This proves that residuals are correlated with the response.
The gradient boosting model was based on the assumption of a Gaussian distribution, to check that this assumption can not be ruled out the residuals are again plotted below. The figure 11 shows that the residuals are not skewed but have a mean around zero. From viewing figure 10 and 11 it is not unreasonable to assume that the data follows a distribution approximate to Gaussian.

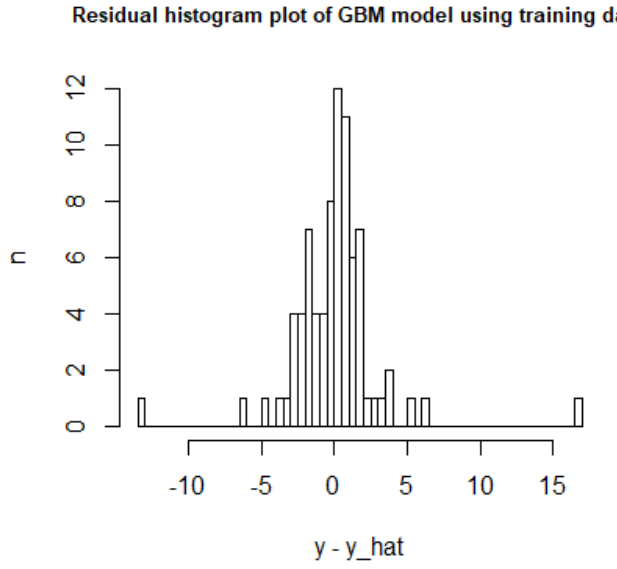


Figure 11: Histogram plot of residuals for the Gradient Boosting Model.

The information provided above does in fact suggest that the gradient boosting model is the best model to predict the assignments data set response variable y. Out of curiosity the influence of the various parameters on the model was investigated. Below is a plot showing in order the top 10 highest relative influencing variables on the gradient boosting model.

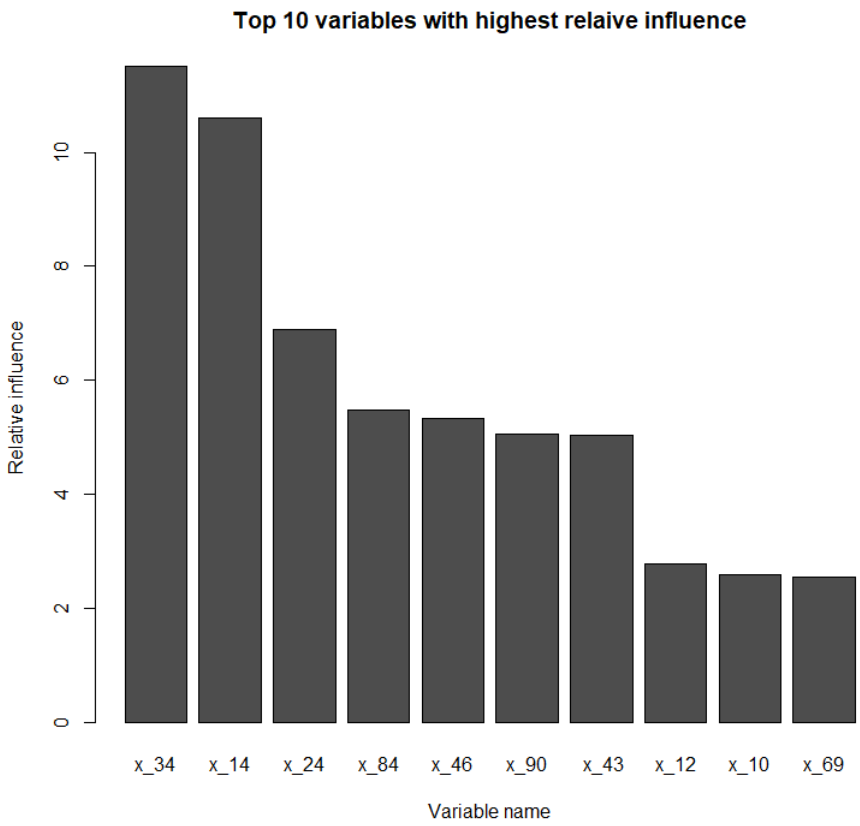


Figure 12: 10 most influential parameters.

6 Appendix

6.1 MATLAB code for Lasso/ Elastic Net

```
%%Lasso

clear all
clc
Data=readtable('case1Data.txt','PreserveVariableNames',true);

outputTable = createOneHotEncoding(Data,'C_1');
outputTable = createOneHotEncoding(outputTable,'C_2');
outputTable = createOneHotEncoding(outputTable,'C_3');
outputTable = createOneHotEncoding(outputTable,'C_4');
outputTable = createOneHotEncoding(outputTable,'C_5');

%change table to matrx
outputTable = outputTable{:,:} ;

% get the observation value and save them into 'X' and 'y'
X = outputTable(:,2:126); % parameter matrix
Y = outputTable(:,1); % result matrix
[n,p]=size(X);

a=0.9;
[B,FitInfo] = lasso(X,Y,'alpha',a,'CV',5,'Standardize',true,'MCrep',1);
lassoPlot(B,FitInfo,'PlotType','CV');
lassoPlot(B,FitInfo,'PlotType','Lambda');
lassoPlot(B,FitInfo,'PlotType','L1');
Index1SE = FitInfo.Index1SE;
fprintf('MSE = %f, DF = %i, a = %f\n',FitInfo.MSE(Index1SE),FitInfo.DF(Index1SE),a)
```

6.2 MATLAB code for Ridge Regression

```
% Ridge
clear all
clc
Data=readtable('case1Data.txt','PreserveVariableNames',true);

outputTable = createOneHotEncoding(Data,'C_1');
outputTable = createOneHotEncoding(outputTable,'C_2');
outputTable = createOneHotEncoding(outputTable,'C_3');
outputTable = createOneHotEncoding(outputTable,'C_4');
outputTable = createOneHotEncoding(outputTable,'C_5');

%change table to matrx
outputTable = outputTable{:,:} ;

%Ridge regression
% get the observation value and save them into 'X' and 'y'
X = outputTable(:,2:126); % parameter matrix
y = outputTable(:,1); % result matrix

% N: the number of observations
% p: the number of parameters
[N, p] = size(X);

[yn] = center(y); % center response
[Xn] = normalize(X); % normalize data

k = 100;
lambda = logspace(-4,4,k);
Beta = zeros(p,k);

for i = 1:k
    Beta(:,i) = (Xn'*Xn+lambda(i)*eye(p)) \ Xn'*yn;
end
%%%
fig1 = figure();
semilogx(lambda, Beta')
xlabel('\lambda')
ylabel('\beta estimates')
title('Regularized estimates')
axes1 = gca;

%Cross validation
K=99;
MSE = zeros(K,k);
Lambda_CV = 0;
I = crossvalind('kFold',N,K);
%%
for i=1:K
    Xtrain = X(I~=i,:);
    Ytrain = y(I~=i,:);
    Xtest  = X(I==i,:);
    Ytest  = y(I==i,:);

    [Ytrain,my] = center(Ytrain); % center training response
    Ytest = Ytest-my; % use the mean value of the training response to center the test response
    [Xtrain,mx,varx] = normalize(Xtrain); % normalize training data
    Xtest=normalizetest(Xtest,mx,varx); % normalize test data with mean and variance of training data

    for j=1:k
        Beta = (Xtrain'*Xtrain+lambda(j)*eye(p)) \ Xtrain'*Ytrain;
        MSE(i,j)= mean((Ytest-Xtest*Beta).^2);
    end
end

meanMSE = mean(MSE);
[MSE_obt, j_opt] = min(meanMSE);
Lambda_CV = lambda(j_opt);
%%%
```

```
fig2 = figure();
axes2 = copyobj(axes1,fig2);
hold on
semilogx([Lambda_CV Lambda_CV],[-2 7],'--r')
disp(['CV lambda = ',num2str(Lambda_CV)]);


%i will use now chosen lambda

lambda = Lambda_CV;
for i = 1:100
     Beta_New = (Xn'*Xn+lambda.*eye(125)) \ Xn'*yn;
 end

 MSE_fin= mean((yn-Xn*Beta_New).^2);
 meanMSE_fin = mean(MSE_fin);
 disp(['CV lambda = ',num2str(Lambda_CV)]);
 disp(['MSE = ',num2str(meanMSE_fin)]);
%
```

6.3 R code for data manipulation and methods: Neural Network, SVM, Random Forest, AdaBoost, GradientBoost

```
# Load needed packages
library(dplyr) # dplyr is package used to manipulate data
library(tidyr)
library(ggplot2)
library(glmnet)
library(caret)
library(neuralnet)
library(kernlab)
library(pracma) # for function logspace
library(pander)
library(tree)
library(randomForest)
library(mboost)



data = read.csv("case1Data.txt", sep=",", header=TRUE, na.strings="NA")
n = dim(data)[1] # number of observations
p = dim(data)[2] # number of parameters

# function to find the most common level (value) for a given column
calculate_mode = function(x) { # x is the data vector
  uniqx = unique(na.omit(x))
  uniqx[which.max(tabulate(match(x, uniqx)))]
}

mcl_97 = calculate_mode(data[,97]) # most common level for column 97 is k
mcl_98 = calculate_mode(data[,98]) # most common level for column 98 is J
mcl_99 = calculate_mode(data[,99]) # most common level for column 99 is H
mcl_100 = calculate_mode(data[,100]) # most common level for column 100 is H
mcl_101 = calculate_mode(data[,101]) # most common level for column 101 is H

# replacing all NaN values with the columns most common level
data$C_.1 = as.character(data$C_.1)
data$C_.1[data$C_.1 == " NaN"] = "K"
data$C_.1 = as.factor(data$C_.1)

data$C_.2 = as.character(data$C_.2)
data$C_.2[data$C_.2 == " NaN"] = "J"

data$C_.3 = as.character(data$C_.3)
data$C_.3[data$C_.3 == " NaN"] = "H"

data$C_.4 = as.character(data$C_.4)
data$C_.4[data$C_.4 == " NaN"] = "H"

data$C_.5 = as.character(data$C_.5)
data$C_.5[data$C_.5 == " NaN"] = "H"

# Encode factor variables by converting them to multiple "dummy variables" (aka "one hot" encoding)
dmy = dummyVars(" ~ . ", data=data)
data2 = data.frame(predict(dmy, newdata=data))

# Normalize data
normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Scale/ normalize data
maxs = apply(data2, 2, max)
mins = apply(data2, 2, min)
scaled = as.data.frame(scale(data2, center = mins, scale = maxs - mins))

normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

```
scaled2 = data.frame(y=data2[,1], scale(data2[,2:126]))

# Split scaled data into train and test (80/20)
n=dim(scaled2)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train_=scaled[id,]
test_=scaled[-id,]

# Split unscaled data into train and test (80/20)
n=dim(data2)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=data2[id,]
test=data2[-id,]

# Function to quickly calculate RMSE
rmse = function(error){
  sqrt(mean(error^2))
}

# Apply PCA
prc = prcomp(data2, scale=TRUE)
plot(prc)
summary(prc)
print(prc)

# Apply BH test to parameters
pvals = sapply(2:126, function(i)
  t.test(data2[,i]~data2$y, alternative="two.sided")$p.value)
names(pvals)<-colnames(data2[,-1])
# pvals[which(pvals<=0.05)]
sorted_pvals<-sort(pvals,decreasing =FALSE)
# L=sapply(1:4702, function(i) max(pvals[i],0.05*(i/4702)))
adjusted<-p.adjust(pvals,method="BH")
print(data.frame(features=names(pvals)[which(adjusted<0.05)]))

# fit a elastic net regression model
model_cvelastic1 = cv.glmnet(y=as.matrix(train[,1]), x=as.matrix(train[,-1]), scale=TRUE, alpha=0.2) # alpha=0 gives
plot(model_cvelastic1) # Check to make sure cross-validation looks ok
s = summary(model_cvelastic1)
model_elastic1 = glmnet(y=as.matrix(train[,1]), x=as.matrix(train[,2:126]), lambda=0.05)

y_hat = predict(model_cvelastic1, newx=as.matrix(test[,2:126]), s=model_cvelastic1$lambda.1se) # lambda.1se = leave
rmse.glm0 = rmse(y_hat - test$y)
rmse.glm0

plot(test$y, y_hat,col='red',main='Real vs predicted elastic', pch=18, cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='Elastic Net',pch=18,col='red', bty='n')

# Alternative ridge regression
n = dim(data2)[1] #nr. observations
#p = dim(data2)[2] #nr. parameters to estimate
p = 125
# y = data[,9]
# x = data[,1:8]
x = as.matrix(data2[,-1])
y = as.matrix(data2[,1])
xn = scale(x)
yn = y-mean(y)

m = 100; # try m values of lambda
lambdas = logspace(-3,4,m); # define k values of lambda on a log scale
betas_r = matrix(NaN,p,m); # prepare a matrix for all the ridge parameters
for (i in 1:m){ # repeat m times
  betas_r[,i] = solve(t(xn)%*%xn + lambdas[i]*diag(p), t(xn)%*%yn) # estimte ridge coefficients #ginv()?
}

# plot the results (log x-axis, linear y-axis), exclude the intercept
rainbowvec = rainbow(p-1)
plot(lambdas, betas_r[2,], log='x',type='l',ylim=c(-1/2,1),xlab=expression(lambda),
```

```
     ylab="Estimate",col=rainbowvec[1]) # ignore log warning, it plots in log scale just fine
for (i in 3:p){
  lines(lambdas, betas_r[i,],col=rainbowvec[i])
}


# fit lm regression model
model_lm = lm(y~., data=train)
y_hatlm = predict(model_lm, newdata=test)
rmse.lm = rmse(y_hatlm - test$y)

# fit nn
# 2 hidden layers with configuration: 13:5:3:1. The input layer has 13 inputs,
# the two hidden layers have 5 and 3 neurons and the output layer has, of course,
# a single output since we are doing regression.
nms = names(train_)
f = as.formula(paste("y ~", paste(nms[!nms %in% "y"], collapse = " + ")))
model_nn = neuralnet(f, data=train_, hidden=c(5,3), linear.output=TRUE)

y_hat.nn = compute(model_nn, test_[,2:126])
y_hat.nn_train = compute(model_nn, train_[,2:126])

y_hat.nn_ = y_hat.nn$net.result*(max(data2$y)-min(data2$y))+min(data2$y) # This will scale the predictions back to o
y_hat.nn_train_ = y_hat.nn_train$net.result*(max(data2$y)-min(data2$y))+min(data2$y)
test.r = (test_$y)*(max(data2$y)-min(data2$y))+min(data2$y) # Scale test data back to original
train.r = (train_$y)*(max(data2$y)-min(data2$y))+min(data2$y)
rmse.nn = rmse(y_hat.nn_ - test.r)
rmse.nn
rmse.nn_train = rmse(y_hat.nn_train_ - train.r)
rmse.nn_train

# fit SVM
model_svm = ksvm(x=as.matrix(train[,2:126]), y=as.matrix(train[,1]), scaled=T, C=5) # Cost constraint of 5
y_hat.svm = predict(model_svm, test[2:126])
#y_hat.svm_ = y_hat.svm*(max(data2$y)-min(data2$y))+min(data2$y)
rmse.svm = rmse(y_hat.svm - test[,1])
rmse.svm
# Calculate rmse for SVM using training data
y_hat.svm_train = predict(model_svm, train[2:126])
rmse.svm_train = rmse(y_hat.svm_train - train[,1])
rmse.svm_train


# Fit random forrest
model_rf = randomForest(y~., data=train, ntree=100)
  #randomForest(x=train[,-1], y=train[,1], scaled=T)
plot(model_rf, main="Random Forest")
y_hat.rf = predict(model_rf, newdata=test[,-1])
rmse.rf = rmse(y_hat.rf - test[,1])
rmse.rf
test.err = double(125)
for(i in 2:126){
  model_rf = randomForest(y~., data=train, mtry=i)
  #randomForest(x=train[,-1], y=train[,1], scaled=T)
  y_hat.rf = predict(model_rf, newdata=test[,-1])
  test.err[i] = rmse(y_hat.rf - test[,1])
}
test.err
plot(test.err)
# Calculate rmse for random forest using training data
y_hat.rf_train = predict(model_rf, train[,2:126])
rmse.rf_train = rmse(y_hat.rf_train - train[,1])
rmse.rf_train

# Find importance of parameters with random forrest model, higher -> more important
round(importance(model_rf), 2)

# Fit regression tree

# Fit Adaboost model
model_ada = blackboost(formula=y~., data=train, control=boost_control(mstop=100), family=GaussReg())
y_hat.ada = predict(model_ada, newdata=test)
```

```
y_hat.ada_train = predict(model_ada, newdata=train)
rmse.ada = rmse(y_hat.ada - test[,1])
rmse.ada_train = rmse(y_hat.ada_train - train[,1])
rmse.ada
rmse.ada_train

# Fit gbm (gradiaent boosting model)
library(gbm)
model_gbm = gbm(y~., data=train, cv.fold=10, distribution="gaussian")

y_hat.gbm = predict(model_gbm, newdata=test, n.trees=100)
y_hat.gbm_train = predict(model_gbm, newdata=train, n.trees=100)
rmse.gbm = rmse(y_hat.gbm - test[,1])
rmse.gbm_train = rmse(y_hat.gbm_train - train[,1])
rmse.gbm # Seems to be best model so far
rmse.gbm_train

model_gbm2 = gbm(y~ x_34 +x_14+ x_24+ x_84+ x_46+ x_90+ x_43+ x_12+ x_10+ x_69,
                 data=train, cv.fold=10)
yhat.gbm2 = predict(model_gbm2, newdata=test, n.trees=100)
rmse.gbm2 = rmse(yhat.gbm2 - test[,1])
rmse.gbm2


# plot residuals
plot(train$y, train$y - model_gbm$fit, xlab="y", ylab="y - y_hat",
     main="Residual plot of GBM model using training data", cex.main=0.8)
hist(train$y - model_gbm$fit, breaks=50, xlab="y - y_hat", ylab="n",
     main="Residual histogram plot of GBM model using training data", cex.main=0.8)
head(summary(model_gbm, 10))
barplot(rel_infl, names.arg = c("x_34", "x_14", "x_24", "x_84", "x_46", "x_90"," x_43", "x_12","x_10", "x_69"),
        xlab="Variable name", ylab="Relative influence", main="Top 10 variables with highest relaive influence")

# Regression tree with library rpart
library(e1071)
library(caret)
library(rpart)
numFolds <- trainControl(method = "cv", number = 10) # Produce CV train sets (k = 10)
cpGrid <- expand.grid(.cp = seq(0.01, 0.5, 0.01)) # To try many different cp values
model_rpart1 <- train(y~., data = train, method = "rpart", trControl = numFolds, tuneGrid = cpGrid) # lowest rmse fo
model_rpart2 = rpart(y~., data = train, method = "anova", cp = 0.1)
y_hat.rpart <- predict(model_rpart2, newdata = test)
rmse.rpart = rmse(y_hat.rpart - test[,1])
rmse.rpart # Still higher than randomForest package used above ...

# Plot results of each model using test and train data.
model_names = c("Lasso", "Elastic Net", "Neural Network", "SVM", "Random Forest", "Ada Boost", "Gradient Boost")
results = structure(list(Lasso=c(31.75816,20), Elastic_Net=c(30.6930,19.45), Neural_Network=c(rmse.nn,rmse.nn_train)
                         SVM=c(rmse.svm,rmse.svm_train),  Random_Forst=c(rmse.rf,rmse.rf_train),
                         Ada_Boost=c(rmse.ada,rmse.ada_train), Gradient_Boost=c(rmse.gbm,rmse.gbm_train)),
                    .Names=model_names, class="data.frame", row.names=c("Test", "Train"))
attach(results)
print(results)
barplot(as.matrix(results), main="Model Results", ylab="RMSE", col=c("blue", "red"), beside=TRUE, cex.main=1.5)
legend("topright", c("Test", "Train"), cex=1.3, bty="n", fill=c("blue", "red"))
```