THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

GROUP PROJECT - ST443

Online News Popularity Prediction and Coordinate Descent Algorithm in Lasso

Group 12

Zhou Lu 14974 - 25%

Zijun Wu 15329 - 25%

Michel Dilmanian - 16302 - 25%

Jun Jie Ng - 17650 - 25%

Keywords: Machine Learning, Random Forest, Boosting, Lasso

December 11, 2018

1 Real World Data

With the development of the Internet and Web 2.0, the public's consumption of news has increasingly shifted to online sources. The internet has not only made it easier to consume news, but also to disseminate it throughout the world through "sharing". Since the popularity of news is relevant to the value of advertisements etc, **we want to build models to predict the popularity of articles and explore which factors drive the number of times an article is shared, which is important for both journalists and advertisers [1].** In this paper, we use two methods – regression and classification – to predict the popularity of news based on a large number of features. **Our results show that boosting and random forest are the best models for regression and classification, respectively**.

1.1 Data

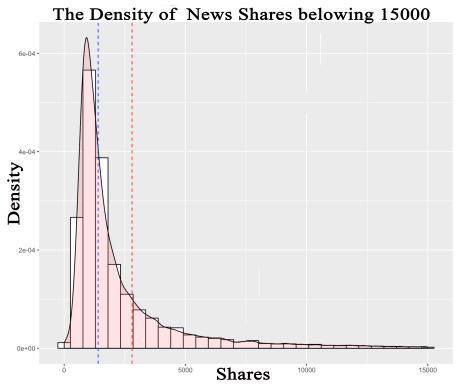
The source of our dataset is the UCI Machine Learning Repository [4]. The dataset includes various features about news articles published by the digital media website Mashable [3] in a period of two years. **There are 39,797 observations with 58 predictors (see full description in Figure 1), and the response variable is the number of shares in social networks.** The correlation matrix heat-map attached in the appendix shows that there are no strong collinearities among the variables.

Our aim is to predict the number of shares, a proxy for popularity. To do this, we carry out both regression and classification. To reduce the **strong left-skew** in the distribution of the response variable, **we take the logarithm of the variable in our regressions.** As for classification, we need to select a threshold to transform the response into a binary variable. Intuitively, the median (i.e. 1400) may seem like a good choice of threshold because it creates balanced groups. However, as Figure 6 shows, the data is clustered around 1100 shares, so choosing the median as a threshold value (first dotted line) will make it easy to misclassify the data. **Therefore, we aim to pick a threshold further to the right of this cluster.** We decide to use the third quartile (i.e. 2800 shares, second dotted line). Since this results in unbalanced groups, **we randomly sample from the "unpopular" group to get the same number of observations as in the "popular" group.** Although we may lose some information, the number of observations (i.e. 10,005) is large enough to do classification.

Feature	Type (#)
Words	
Number of words in the title	number (1)
Number of words in the article	number (1)
Average word length	number (1)
Rate of non-stop words	ratio (1)
Rate of unique words	ratio (1)
Rate of unique non-stop words	ratio (1)
Links	
Number of links	number (1)
Number of Mashable article links	number (1)
Minimum, average and maximum number of shares of Mashable links	number (3)
Digital Media	
Number of images	number (1)
Number of videos	number (1)
Time	
Day of the week	nominal (1)
Published on a weekend?	bool (1)

Feature	Type (#)
Keywords	
Number of keywords	number (1)
Worst keyword (min./avg./max. shares)	number (3)
Average keyword (min./avg./max. shares)	number (3)
Best keyword (min./avg./max. shares)	number (3)
Article category (Mashable data channel)	nominal (1)
Natural Language Processing	
Closeness to top 5 LDA topics	ratio (5)
Title subjectivity	ratio (1)
Article text subjectivity score and its absolute difference to 0.5	ratio (2)
Title sentiment polarity	ratio (1)
Rate of positive and negative words	ratio (2)
Pos. words rate among non-neutral words	ratio (1)
Neg. words rate among non-neutral words	ratio (1)
Polarity of positive words (min./avg./max.)	ratio (3)
Polarity of negative words (min./avg./max.)	ratio (3)
Article text polarity score and its absolute difference to 0.5	ratio (2)

Figure 1: The description of each features, from [1], all features are classified in 6 main feature group.



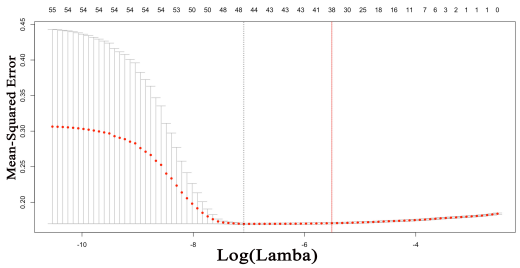


Figure 2: The density plot of shares below 18000, Figure 3: The plot of MSE versus $\log(\lambda)$, the dashed showing strong left-skewed, and the blue dashed line line is with the lowest MSE, and the red line is which is the median (1400) and the red dashed line is the has one standard error distance to the dashed line. third quarter (2800).

1.2 Method

To increase model interpretability and sparsity, we first perform variable selection using a Lasso. Based on the "one-standard-error" rule, the model with $\lambda = 0.0041$ is the simplest model whose accuracy is comparable with the best model [2]. This method reduces the number of variables to 38. We then we split the whole dataset into training (70%) and test sets (30%) by using a random holdout split.

1.3 Regression

In the regression part of our analysis, we begin with Linear Regression (the simplest regression model). We then perform Elastic Net, Random Forest (RF) and Boosting with number of trees $\in \{50, 100, 200, 500, 1000\}$. In order to compare models, we calculate the Mean Square Error (MSE) of prediction, shown in table 1.

Table 1: The MSE of each regression method - the bolded is the best model (we use a logarithm transformation of the response variable).

	Linear Regression	Elastic Net	RF	Boosting
MSE	55.12	1.08	0.74	**0.73**

1.4 Classification

As for classification, we begin with Logistic Regression, followed by Quadratic Discriminant Analysis (QDA) and K-Nearest Neighbours (K-NN) with $K \in \{1, 3, 5, 10, 20, 50\}$. We also use Random Forest (RF) and Boosting with number of trees$\in \{50, 100, 200, 500, 1000\}$. Finally, we also test Support Vector Machine (SVM) with *cost* $\in \{0.001, 0.01, 0.1, 1, 5, 10, 100\}$ for linear Kernel function. We use Accuracy, Precision, Recall, F1 and AUC (Area Under Curve) to test predictive performance of our models, shown in table 2.

2 Results

For our regression models, we pick Elastic Net tuning parameters $\lambda = 0.0014, \alpha = 0.6$ using cross validation. **Our final model choice is Boosting, as it has the lowest MSE**, although the Random Forest performs similarly. In the classification portion, the accuracy is highest for $K - NN$ when $K = 20$ and for SVM when cost is 100 (results in table 2). **We choose Random Forest as the best classification model with the highest criteria.** Figure 4 and figure 5 show the top 5 most important features in the best regression and

classification model, respectively. Clearly, "kw_avg_avg" (average keyword) is the most important feature in both models, and it even contributes significantly more than other features in the regression model (in the classification model, the five features contribute similarly). Moreover, the regression model indicates that shares of referenced articles in Mashable, closeness to LDA (latent Dirichlet allocation) topic and whether the article was published on the weekend are very important; the result of classification model also shows that shares of referenced articles in Mashable and closeness to LDA topic are important.

Table 2: The performance of models in classification.

Method	Logistic Regression	QDA	K-NN	RF	Boosting	SVM
Accuracy	0.635	0.543	0.586	**0.655**	**0.655**	0.630
Precision	0.635	**0.717**	0.513	0.645	0.643	0.625
Recall	0.596	0.107	0.588	**0.645**	0.629	0.631
F1	0.615	0.186	0.548	**0.645**	0.636	0.628
AUC	0.634	0.533	0.584	**0.652**	0.647	0.630

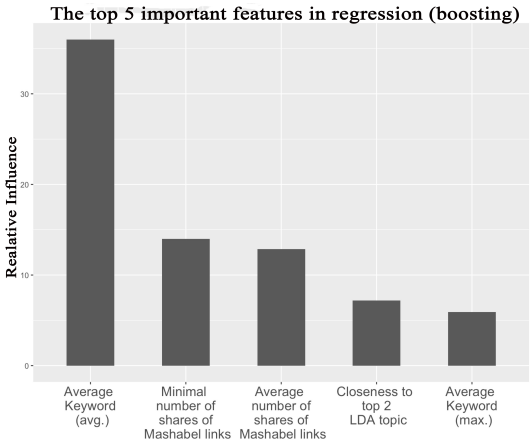


Figure 4: The top five Relative Influence, indicating top five most importance features in the classification.

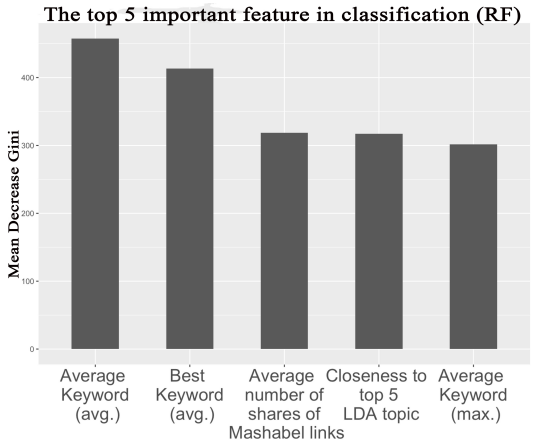


Figure 5: The top five Mean Decrease Gini variable, indicating top five most important features in the regression.

3 Further Discussion

Our analysis finds that the **number of keywords is the most important predictor of the popularity of an article.** This makes intuitive sense; the more keywords an article uses, the more likely it is for an article to be found through search engines like Google. Similarly, articles with larger number of links are more likely to be read.

To **test whether variable selection significantly affects the accuracy of prediction**, we also fit a Random Forest on the full dataset in the classification case. We find that the accuracy is improved marginally to 0.658 – a negligible improvement over 0.655 in the model with variable selection. In addition, since we drop half the dataset to obtain balanced groups in our classification approach, we also **investigate the effect of this information loss** here. When using the full dataset to run the Random Forest, the accuracy is improved significantly to 0.781, but other criteria perform very poorly (e.g. precision is 0.481). We think the high accuracy is biased because of the unbalanced groups and conclude that it is necessary to drop observations and obtain balanced groups.

References

[1] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. "A proactive intelligent decision support system for predicting the popularity of online news". In: *Portuguese Conference on Artificial Intelligence*. Springer. 2015, pp. 535–546.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, NY, USA: 2001.

[3] *Mashable*. URL: `https://mashable.com/`.

[4] UCI Machine Learning Repository. *Online News Popularity Data Set*. May 2015. URL: `http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity`.

Appendices

```
#####################################Regression#################################
data <- read.csv("/Users/apple/Desktop/OnlineNewsPopularity.csv")
data <- data[,-c(1,2)]
train <- sample(nrow(data),0.7*round(nrow(data)),replace = F)
########################################## Lasso: variable selction###############
set.seed(666)
library(glmnet)
x <- model.matrix(data$shares~.-1, data=data)
y <- data$shares
fit.lasso <-glmnet(x,y)
plot(fit.lasso, xvar="lambda", label= TRUE)
plot(fit.lasso, xvar="dev", label= TRUE)
cv.lasso <-cv.glmnet(x, y)
plot(cv.lasso)
abline(v=log(3.522e-02), col="red")
## coefficent vector corresponding to the mse which is within one standard error
#of the lowest mse using the best lambda.
coef(cv.lasso)
## coefficient vector corresponding to the lowest mse using the best lambda
coef(glmnet(x,y, lambda=cv.lasso$lambda.min))
coef(glmnet(x,y, lambda=3.522e-02))
# the varaible left after doing the lasso
name<- c("num_hrefs", "num_imgs", "num_keywords", "data_channel_is_entertainment",
         "data_channel_is_bus","data_channel_is_socmed", "data_channel_is_tech",
         "data_channel_is_world", "kw_min_min", "kw_avg_avg",
         "self_reference_min_shares","self_reference_avg_sharess",
         "is_weekend","LDA_02","shares")
data_Lasso <- data[,name]
#############################linear regression ###################################
lr<- lm(shares~., data = data_Lasso[train,])
predict.test <- predict(lr, data_Lasso[-train,-15])
#MSE 55.1201
mean((predict.test-data[-train,15])^2)
############################ Elastic Net ########################################
x <- model.ma\begin{lstlisting}[language=R]
#############)^2)
res[train]
a <- seq(0.1, 0.9, 0.05)
```

```
search <- foreach(i = a, .combine = rbind) %dopar% {
  cv <- cv.glmnet(x, y, family = "gaussian", nfold = 10,
                  type.measure = "deviance", paralle = TRUE, alpha = i)
  data.frame(cvm = cv$cvm[cv$lambda == cv$lambda.1se],
             lambda.1se = cv$lambda.1se, alpha = i)
}
cv3 <- search[search$cvm == min(search$cvm), ]
cv.elastic <- glmnet(x, y, family = "gaussian",
                     lambda = cv3$lambda.1se, alpha = cv3$alpha)
coef(cv.elastic)
x_test<-model.matrix(shares~.-1, data=data[-train,])
predict.test <- predict(cv.elastic, x_test)
#MSE 1.081562
mean((predict.test-data[-train,59])^2)
## Bagging and Random Forest ###################################
library(randomForest)
set.seed(666)
## Recall that bagging is simply a special case of a random forest with m=p
# the dataset was selected by lasso
bag_new <- randomForest(shares ~. , data=data_Lasso,
                        subset=train,  importance=TRUE)
plot(bag_new)
## Predicted values on the testing data using bagging
yhat_bag <-predict(bag_new, newdata=data_Lasso[-train,-15])
plot(yhat_bag, new_test)
abline(0,1)
# MSE
mean((yhat_bag - data_Lasso[-train,15])^2)
## We can view the importance of each variable
importance(bag_new)
varImpPlot(bag_new)
set.seed(666)
rf_new <-randomForest(shares ~. , data=data_Lasso,
                      subset=train, mtry=4, importance=TRUE, ntree = 500)
rf_new_all <-randomForest(shares ~. , data=data,
                          subset=train, mtry=7, importance=TRUE, ntree = 500)
plot(rf_new)
plot(rf_new_all)
yhat.rf <-predict(rf_new, newdata=data_Lasso[-train,-15])
yhat.rf_all <-predict(rf_new_all, newdata=data[-train,-59])
##MSE RF 0.7175809
mean((yhat.rf_all - data[-train,59]) ^ 2)
#MSE RF 0.7398845
mean((yhat.rf - data_Lasso[-train,15]) ^ 2)

importance(rf_new)
varImpPlot(rf_new)
plot.randomForest(rf_new)
sum(yhat.rf==data_Lasso[-train,17])

library(ROCR)
predictions=as.vector(rf_new$votes[,2])
pred=prediction(predictions,data_Lasso[train,15])
perf_AUC=performance(pred,"auc") #Calculate the AUC value
```

```
AUC=perf_AUC@y.values[[1]]

perf_ROC=performance(pred,"tpr","fpr") #plot the actual ROC curve
plot(perf_ROC, main="ROC plot")
text(0.5,0.5,paste("AUC = ",format(AUC, digits=5, scientific=FALSE)))
## Perform Boosting
## Please learn the topic of Boosting from Session 8.3.4 of the textbook ISLR
library(gbm)
set.seed (666)
boost_new = gbm(shares~. , data=data_Lasso[train,], distribution
 = "gaussian",
                n.trees = 500
)
summary(boost_new)
yhat.boost = predict(boost_new, newdata = data_Lasso[-train,-15],
                     n.trees = 500, type = "response")
# MSE 0.7393814
mean((yhat.boost - data_Lasso[-train,15]) ^ 2)

#############################################################################
#########################Classificaiton#####################################
#############################################################################
data <- read.csv("/Users/apple/Desktop/OnlineNewsPopularity.csv")
data <- data[,-c(1,2)]
data$shares[data$shares <= 2800] <- 0
data$shares[data$shares > 2800] <- 1
#70 percent of dataset in the trainning data
set.seed(666)
library(dplyr)
# The heatmap for correlation
library(mlbench)
library(caret)
# calculate correlation matrix
correlationMatrix <- cor(data[,1:58])
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# print indexes of highly correlated attributes
print(highlyCorrelated)
colnames(data[,highlyCorrelated])
plot(correlationMatrix)
heatmap(correlationMatrix)
library(reshape2)
melt_cor <- melt(correlationMatrix)
library(ggplot2)
ggplot(data = melt_cor, aes(x=factor(Var1), y= factor(Var2), fill=value)) +
  geom_tile()
## Lasso: variable selction
library(glmnet)
x <- model.matrix(data$shares~.-1, data=data)
y <- data$shares
fit.lasso <-glmnet(x,y)
plot(fit.lasso, xvar="lambda", label= TRUE)
plot(fit.lasso, xvar="dev", label= TRUE)
cv.lasso <-cv.glmnet(x, y)
```

```
plot(cv.lasso)
abline(v=log(4.054e-03), col="red")
## coefficent vector corresponding to the mse which is within one standard
#error of the lowest mse using the best lambda.
coef(cv.lasso)
## coefficient vector corresponding to the lowest mse using the best lambda
coef(glmnet(x,y, lambda=cv.lasso$lambda.min))
coef(glmnet(x,y, lambda=4.054e-03))
# the varaible left after doing the lasso
name <- colnames(data)
del<-c(
  grep("n_tokens_title",name),
  grep("n_unique_tokens", name),
  grep("n_non_stop_unique_tokens", name),
  grep("num_videos", name),
  grep("data_channel_is_lifestyle", name),
  grep("kw_min_avg", name),
  grep("self_reference_max_shares", name),
  grep("weekday_is_wednesday", name),
  grep("weekday_is_friday", name)     ,
  grep("weekday_is_saturday", name),
  grep("weekday_is_sunday", name),
  grep("LDA_04", name),
  grep("global_sentiment_polarity", name)     ,
  grep("global_rate_positive_words", name)     ,
  grep("global_rate_negative_words" , name)     ,
  grep("rate_positive_words", name),
  grep("rate_negative_words", name) ,
  grep("avg_positive_polarity", name),
  grep("max_positive_polarity", name),
  grep("max_negative_polarity", name)
)
data_Lasso <- data[,-del]
data_Lasso <- rbind(sample_frac(data_Lasso[data_Lasso$shares==0,],1/3),
                    data_Lasso[data_Lasso$shares==1,])

#checked the group is balanced
sum(data_Lasso$shares==0)
sum(data_Lasso$shares==1)
train <- sample(nrow(data_Lasso),0.7*round(nrow(data_Lasso)),replace = F)


###################### Perform Logistic regression#########################
set.seed(666)
m_log <- glm(shares~. ,data = data_Lasso[train,] , family = binomial())
m_log_prob <- predict(m_log, newdata = data_Lasso[-train, 1:39], type = "response")
# Select 0.5 as threshold
m_log_prob[m_log_prob<=0.5] <- 0
m_log_prob[m_log_prob>0.5]<- 1
library(pROC)
table<- table(data_Lasso[-train,39], m_log_prob)
#  accuracy 0.6348667
mean(m_log_prob == data_Lasso[-train,39])
#recall 0.5956265
```

```
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.6350851
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.6147233
2*precison*recall/(precison+recall)
#auc 0.634
roc(data_Lasso[-train,39], m_log_prob)
######################## Quadratic Discriminant Analysis###################
# Perform QDA on the traning data set using only two predictors,
#1st lag and 2nd lag
qda_fit = qda(shares~. , data = data_Lasso, subset = train)
qda_pred = predict(qda_fit, data_Lasso[-train,-39])$class
# Confusion matrix
table(qda_pred, data_Lasso[-train,39])
# 0.5425225
mean(qda_pred == data_Lasso[-train,39])
table<- table(data_Lasso[-train,39], qda_pred)
#recall 0.1065602
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.7172897
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.1855545
2*precison*recall/(precison+recall)
#auc 0.5332
roc(data_Lasso[-train,39], as.numeric(qda_pred))
######################## Perform K-nearest neighbours ######################
library(class)
# Create training data for X
train_X = data_Lasso[train, 1:38]
# Create testing data for X
test_X = data_Lasso[-train,1:38]
# Create training data for Y
train_Direction = data_Lasso$shares[train]
Direction_test = data_Lasso$shares[-train]
set.seed(666)
plo <- c()
for (k in c(1,3,5,10,20,50)){
  knn_pred = knn(train_X, test_X, train_Direction, k = k)
  table(knn_pred, Direction_test)
  plo[k]<-mean(knn_pred == Direction_test)
}
plot(plo)
# when k= 20, the accuracy is highest, which is 0.5888644
knn_pred = knn(train_X, test_X, train_Direction, k = 20)
table<- table(knn_pred, Direction_test)

# accuary 0.5859786
mean(knn_pred == Direction_test)
# Bagging accuracy 0.6550671
mean(yhat_bag == data_Lasso[-train,39])
#recall 0.5878378
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.5133634
precison<-table[2,2]/(table[2,2]+table[1,2])
```

```
#F1 0.5480823
2*precison*recall/(precison+recall)
#auc 0.5844
roc(Direction_test, as.numeric(knn_pred))
######################## Bagging and Random Forest ######################
library(randomForest)
set.seed(666)
## Recall that bagging is simply a special case of a random forest with m=p

# the dataset was selected by lasso
bag_new <- randomForest(factor(shares) ~. , data=data_Lasso,
                        subset=train,  importance=TRUE)
plot(bag_new)
## Predicted values on the testing data using bagging
yhat_bag <-predict(bag_new, newdata=data_Lasso[-train,-39])
plot(yhat_bag, new_test)
abline(0,1)
table<- table(data_Lasso[-train,39], yhat_bag)
# Bagging accuracy 0.6550671
mean(yhat_bag == data_Lasso[-train,39])
#recall 0.6532454
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.6456261
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.6494134
2*precison*recall/(precison+recall)
#auc 0.655
roc(data_Lasso[-train,39], as.numeric(yhat_bag))

## We can view the importance of each variable
importance(bag_new)
varImpPlot(bag_new)


############################## Random Forest ################################
set.seed(666)
rf_new <-randomForest(factor(shares) ~. , data=data_Lasso,
                      subset=train, mtry=6, importance=TRUE, ntree = 500)
rf_new_all <-randomForest(factor(shares) ~. , data=data,
                          subset=train, mtry=7, importance=TRUE, ntree = 500)
plot(rf_new)
plot(rf_new_all)
yhat.rf <-predict(rf_new, newdata=data_Lasso[-train,-39])
yhat.rf_all <-predict(rf_new_all, newdata=data[-train,-59])
##Compute the test accuracy for the whole dataset
mean(yhat.rf_all == data[-train,59])
#Compute the test accuracy for the Lasso dataset
mean(yhat.rf == data_Lasso[-train,39])

table<- table(data_Lasso[-train,39], yhat.rf)
# accuracy 0.6550671
mean(yhat_bag == data_Lasso[-train,39])
#recall 0.644915
recall<-table[2,2]/(table[2,2]+table[2,1])
```

```
#precison 0.6446912
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.6448031
2*precison*recall/(precison+recall)
#auc 0.6524
roc(data_Lasso[-train,39], as.numeric(yhat.rf))
importance(rf_new)
varImpPlot(rf_new)
df<- data.frame(x = c("kw_avg_avg", "kw_max_avg","self_reference_avg_sharess",
                      "LDA_02", "kw_avg_max"),
                MeanDecreaseGini = c(457.28062, 413.12311,
                318.57822, 317.03859, 301.64292))
library(ggplot2)
positions <- df[,1]
ggplot(data= df, aes(x=x,y=MeanDecreaseGini)) +
  geom_bar(stat="identity",  width=0.5) +
  scale_x_discrete(limits = positions) +
  xlab("") +theme(axis.text.x = element_text(size = 12))
sum(yhat.rf==data_Lasso[-train,17])
###################### Perform Boosting ##############################
## Please learn the topic of Boosting from Session
#8.3.4 of the textbook ISLR
library(gbm)
set.seed (666)
boost_new = gbm(shares~. , data=data_Lasso[train,],
distribution = "bernoulli",
                n.trees = 5000
)
summary(boost_new)
yhat.boost = predict(boost_new,
newdata = data_Lasso[-train,-39], n.trees = 5000,
                     type = "response")
yhat.boost[yhat.boost>0.5] <- 1
yhat.boost[yhat.boost<=0.5] <- 0

table<- table(data_Lasso[-train,39], yhat.boost)
# accuracy 0.6550671
mean(yhat_bag == data_Lasso[-train,39])
#recall 0.6286012
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.6431108
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.6357732
2*precison*recall/(precison+recall)
#auc 0.6474
roc(data_Lasso[-train,39], as.numeric(yhat.boost))

##################################SVM#######################
# install.packages("e1071")
library(e1071)
data_Lasso$shares[data_Lasso$shares==0] <- -1
test <- sample(nrow(data),0.7*nrow(data),replace = F)
dat = data.frame(x = data_Lasso[test,-39],
                 y = as.factor(data_Lasso[test,39]))
```

```
## We now use tune() function to perform 10-fold corss
#validation to determine an optimal cost parameter
set.seed(666)
tune.out = tune(svm, y~., data = dat, kernel = "linear",
                ranges = list(cost = c(0.001, 0.01, 0.1,
                1, 5, 10, 100)),
                scale=T)
summary(tune.out)
## We see that cost = 0.1 results in the lowest cross
# validation error rate
## tune() function stores the best model obtained, which
# can be assessed as follows
bestmod = tune.out$best.model
summary(bestmod)

## The predict() function is used to predict the class label
#on a set of test observations
## First generate the test observations
xtest = matrix(rnorm(20*2), ncol = 2)
ytest = sample(c(-1,1), 20, rep = TRUE)
xtest[ytest == 1,] = xtest[ytest == 1,] + 1
ttest <- sample(nrow(data),0.05*nrow(data),replace = F)

testdat = data.frame(x = data_Lasso[-test,-39],
y = as.factor(data_Lasso[-test,39]))
## Now we predict the class labels of these test obervations.
#Here we use the best model obtained through cross
#validation in order to make predictions
ypred = predict(bestmod, testdat)
table(predict = ypred, truth = testdat$y)
mean(ypred==testdat$y)
#
table<- table(testdat$y, ypred)
# accuracy 0.6304016
mean(testdat$y == ypred)
#recall 0.6309116
recall<-table[2,2]/(table[2,2]+table[2,1])
#precison 0.6251273
precison<-table[2,2]/(table[2,2]+table[1,2])
#F1 0.6280061
2*precison*recall/(precison+recall)
#auc 0.6304
roc(testdat$y, as.numeric(ypred))
```

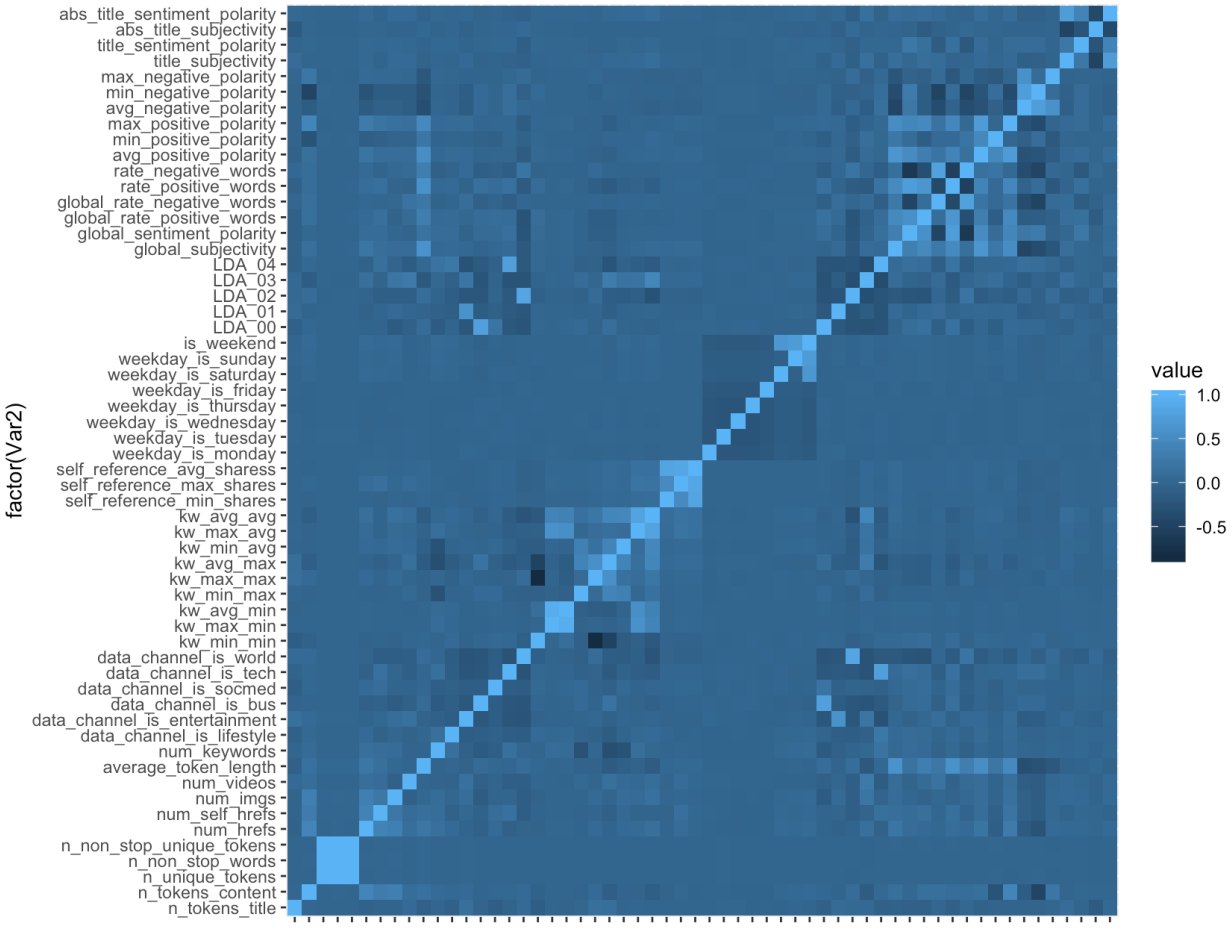


Figure 6: The heatmap for correlation.

1 Coordinate Descent Algorithm

In this section we analyse the performance of two regression shrinkage methods – the lasso and the elastic net – by implementing the "one-at-a-time" coordinate descent algorithm specified in Friedman et al. (2007). This section of the report is organised as follows. In the first part, we describe the development of our coordinate descent algorithm for the lasso and elastic net regularization. Next, we explain the data simulation process, followed by a brief example that illustrates how we obtain the tuning parameters $\hat{\lambda}$ for our models using Mean-Squared-Errors (MSE). Our numerical simulation results are analysed sections 4 and 5. We summarise our findings and conclude in section 6.

2 Coordinate Descent Algorithm for Lasso and Elastic Net

We developed the coordinate descent algorithm to solve the Lasso problem as follow. Our function $lasso\_alg(X, Y, \lambda)$ takes three parameters as arguments and returns the vector of coefficients β.

1. We begin by **scaling X** and initialising β_j and $\beta_j^{old} = 0$, $j = 1, ..., p$
2. We repeat the following steps until the algorithm converges for $j = 1, ..., p$:
   - We let $\beta_j^{old} = \beta_j$. The partial residual is computed by taking $r_{ij} = y_i - \sum_{k \neq j} x_{ik}\beta_k$.
   - We compute the least square coefficients of residuals r_{ij} on the j-th predictor by taking $\beta_j^* = \frac{1}{n} \sum_{i=1}^{n} x_{ij} r_{ij}$
   - We can compute β_j using the *soft thresholding* method as follows:

$$\beta_j = \text{sign}(\beta_j^*) \max((|\beta_j^*| - \lambda), 0)$$

3. In repeating (2), we define convergence as the Root-Mean-Square Error (RMSE) between the current and prior iteration being smaller than 10^{-10}, where

$$RMSE = \sqrt{\frac{(\beta_j - \beta_j^{old})^2}{n}}$$

4. We **re-scale** β and return it as the function output

The coordinate descent algorithm to solve the elastic net is very much similar to the algorithm used to solve the lasso. We create a function, $elastic\_net\_alg(X, Y, \lambda_1, \lambda_2)$, that takes five parameters and returns the vector of coefficients β. Since the objective function differs, we make a modest modification to the *soft thresholding* from (2.c) in the above algorithm above.

$$\beta_j = \text{sign}(\beta_j^*) \max((|\beta_j^*| - \lambda_1), 0) \ (1 + 2\lambda_2)^{-1}$$

We scale the data at the beginning so that each predictor has mean 0 and variance 1. In our derivation, $\beta_j^* = \frac{1}{\sum_{i=1}^{n}(x_{ij})^2} \sum_{i=1}^{n} x_{ij} r_{ij}$. Since we scale our data, $\sum_{i=1}^{n}(x_{ij})^2$ is close to n, hence we have $\beta_j^* = \frac{1}{n} \sum_{i=1}^{n} x_{ij} r_{ij}$ in the algorithm.

The aim of this project is to report the mean and standard error of the **mean-squared-error** (MSE) and the **number of estimated non-zero coefficients** for the lasso and the elastic net

across a number of cases. To ease implementation of this process, we create multiple functions to run the coordinate descent algorithms, select optimal tuning parameters, and evaluate the performance of optimised models. The final programme returns the three main reporting variables mentioned above, where $MSE = [\sum_{i=1}^{n} \sum_{j=1}^{p} (y_i - x_{ij}\beta_j)^2]/n$.

3 Data Simulation

The next step of our project is to generate simulations from the model $Y = X\beta + \sigma\epsilon$ where $\epsilon \sim N(0, I_n)$. In order to test the functionality of our program, we first set the parameters to the default case as follows.

1. Size of training dataset (n_train) = size of validation dataset ($n\_validation$) = 20
2. Size of testing dataset (n_test) = 200
3. $\beta = (3, 1.5, 0, 0, 2, 0, 0, 0)^T$
4. $\sigma = 3$
5. $corr(X_i, X_j) = 0.5^{|i-j|}$
6. Number of simulations of the entire process (r) = 50

We first have to simulate two sets of random variables for X and ϵ. For the noise component ϵ, we use the *rnorm* function within R to generate random normal samples with mean = 0 and sd = 1. The tricky part here is to simulate the X variables as *rnorm* only allows for random variables with zero correlations. In our simulations, we need the pairwise correlation between X_i and X_j to be equal to $0.5^{|i-j|}$. We build a *Sigma_create* function to set up a covariance matrix stated above. Next, we utilise the built-in *mvrnorm* function to simulate our X values, calling the *sigma* that we have just created. Once we have X and ϵ, we use matrix multiplication to obtain Y data set.

The data sets are then split into a training set, an independent validation set and an independent test set. The training set is used to run the coordinate descent algorithms for a range of parameter values, the validation set is used to select the optimal tuning parameters, and the test set is used to estimate the accuracy of the models with optimal tuning parameters.

4 Obtaining Tuning Parameters $\hat{\lambda}$

After simulating the data sets for X and Y, the next step is to find a way to select the optimal tuning parameters: $\hat{\lambda}$ in lasso and $\hat{\lambda}_1$, $\hat{\lambda}_2$ in elastic net. **The general logic here is to perform trial and error with different values of λ on the training data set.** A range of λ is fitted into the *lasso* and *elastic_net* functions described earlier. The returned β coefficients for each respective λ are tested on the validation data using the *evaluation* function that we coded. We select the optimal parameter $\hat{\lambda}$ that produces the smallest error (MSE) on the validation data. **After we have obtained optimal tuning parameters, we re-estimate the model on the training data and obtain our final results by running it on the test data.**

To illustrate how we select the optimal lambda, we plot lambda against the MSE when solved using the coordinate descent algorithm. As shown in Figure 1, the range of lambda is $(0, 5)$ with a length of 25. The analysis is performed with the default case and the optimal point is highlighted in red. We select $\hat{\lambda} = 2.08$ as it gives us the lowest MSE of 13.4, compared to other λ, when evaluated against the testing data.

A similar thought process is applied to generate a pair of most efficient lambdas for the elastic net algorithm. The results is plotted in Figure 2 below, and the lowest MSE of 10.9 is achieved when $\hat{\lambda}_1 = 1.04$ and $\hat{\lambda}_2 = 0.208$.

In the default case, we can say that all tuning parameters are **significantly greater** than zero. If we take a closer look at the error terms (MSE), we can say that the performance of the elastic net dominates the lasso problem. However, the number of estimated non-zero coefficients selected by lasso is the true value 3 whereas elastic net selects a number of 4. In this default case, there exist a trade-off between the prediction accuracy and variable selection. However, we can't draw any conclusion from just one sample run and hence we proceed to repeat the simulation by r (50 by default) amount of times.

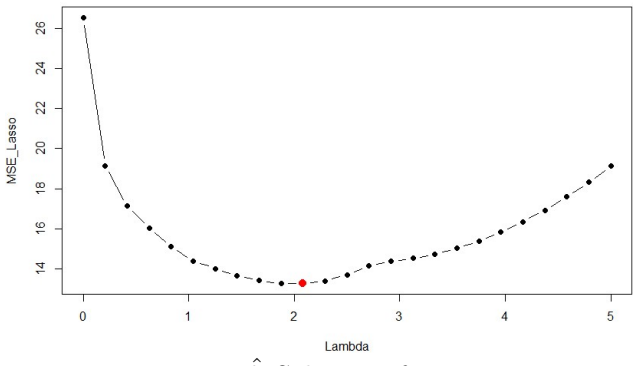


Figure 1: $\hat{\lambda}$ Selection for Lasso

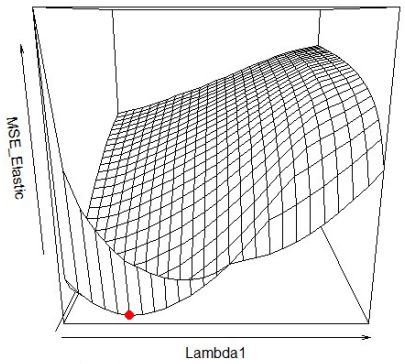


Figure 2: $\hat{\lambda}_1$ $\hat{\lambda}_2$ Selection for Elastic Net

We proceed to observe the different results produced by our algorithms. We can summarise the results into several scenarios when it comes to selecting the best Lambdas:

1. $\hat{\lambda}^{lasso} = \hat{\lambda}_1^{elastic} = \hat{\lambda}_2^{elastic} = 0$: The lasso and elastic net penalties are all reduced to Ordinary Least Square regression (OLS).

2. $\hat{\lambda}^{lasso} \neq 0$, $\hat{\lambda}_1^{elastic} \neq 0$, $\hat{\lambda}_2^{elastic} = 0$: The elastic net regularization will take the form of the lasso penalties instead.

3. $\hat{\lambda}^{lasso} \neq 0$, $\hat{\lambda}_1^{elastic} = 0$, $\hat{\lambda}_2^{elastic} \neq 0$: The elastic net regularization will shrink to a ridge regression problem in this scenario.

4. $\hat{\lambda}^{lasso} \neq 0$, $\hat{\lambda}_1^{elastic} \neq 0$, $\hat{\lambda}_2^{elastic} \neq 0$: The default case is an example of this scenario and it allows us to compare the performance between lasso and elastic net when using the simulated data sets.

5 Main Simulation Results

5.1 Base Case: $n > p$

Our default case sets the parameters exactly as described in Section 3. The bolded row in Table 1 below shows the main results of 50 simulations of our programme, focusing on reporting the three key variables of this project: the mean and standard error of MSE and the number of estimated non-zero coefficients.

Table 1: Results for Base Case with varying n

	Lasso			Elastic net		
No. of obs. (n)	MSE	MSE st. err.	# of vars	MSE	MSE st. err.	# of vars
120	16.53	0.89	4.5	16.10	0.76	5.4
240	**12.11**	**0.39**	**5.9**	**12.02**	**0.37**	**6.5**
480	10.56	0.16	6.2	10.58	0.17	6.4
960	9.60	0.08	5.7	9.62	0.09	5.7
1920	9.40	0.06	5.6	9.40	0.06	5.7
3840	9.21	0.03	6.0	9.21	0.03	6.0

As expected, **the elastic net regularization has better predictive performance than the lasso**, as shown in the smaller mean MSE for the elastic net. It also computes the MSE in a more stable manner, reflected in the lower standard error of the MSE. Importantly, the elastic net not only produces better prediction accuracy; **it also does well at variable selection**. The elastic net selected 6.5 variables on average across 50 simulations, only slightly higher than the 5.9 variables selected on average by the lasso.

We would also like to know how or whether these results vary with a change in the number of observations. As such, we repeat the Base Case simulation for datasets with various numbers of observations n, keeping the proportion of training, validation and testing sets equal.

Table 1 shows that our main findings hold when n is not too large. Given sufficiently large n, the elastic net problem takes the form of the lasso, with $\hat{\lambda}_2^{elastic} = 0$. And the MSE of using elastic net or lasso does not differ very much. The differences between the lasso and the elastic net shrink towards zero as $n \to \infty$, where all $\hat{\lambda} = 0$, and we have an Ordinary Least Square regression (OLS). We find that this is partly because of the small size of p relative to n; indeed, running the scenario with larger values of p yields greater differences between the lasso and elastic net. (We will discuss large values of p in more detail in Section 5.2.)

More generally, we explain this pattern as follows: as $n \to \infty$, we are able to make more reliable predictions (as shown in the standard error of the MSE), hence the MSE is negatively correlated with the number of observations. We therefore conclude that **the superiority of the elastic net over the lasso grows as our training dataset becomes more limited**. Working with a huge amount of data will result in the elastic net converging towards the lasso.

5.2 Case 2: $n < p$

The number of variables p has an opposite effect as the number of data n in relation with the estimation result. With a **large p relative to n, the elastic net penalty proves to be more advantageous than the lasso** as data become more scarce to estimate a large number of coefficients. Alternatively, the elastic net regularization will converge to the lasso regularization with a small number of variables. In this subsection, we are more interested in testing out a special case where the number of predictors is greater than the data size used to estimate the model.

Our second case is different from the default in one major way: we set the number of predictors, p, to be greater than the number of observations in the training data (n_train). Specifically, we maintain the size of training, validation and testing sets from our base case (20 training, 20 validation, 200 testing) and increase p to 25. Given the larger size of p, we generate non-overlapping coefficients β_j randomly by sampling without replacement and we leave the other parameters unchanged.

Theory predicts that the **lasso in this scenario selects at most n_train variables** before saturating. Our simulations results align with this finding. Across our 50 simulations, the average number of predictors selected by the lasso is 13.5, considerably lower than the 20.8 predictors chosen on average by the elastic net specification. It is also worth pointing out that error term for the elastic net specification ($MSE = 15.88$) is significantly lower than the lasso ($MSE = 18.55$) in this scenario. This makes intuitive sense, as there may be cases in which the optimal choice of non-zero coefficients is greater than n_train and elastic net can capture this better than the lasso.

5.3 Case 3: High pairwise correlations

In our third scenario, we vary the correlation between the first two predictors, X_1 and X_2, and record the results. Firstly, we discuss the situation when $X_1 = X_2$, then we proceed to run 50 simulations each for $corr(X_1, X_2)$ ranging from 0.1 to 0.9 (increasing by increments of 0.1). The main results in terms of prediction performance continue to hold: the elastic net performs slightly better in terms of the mean MSE, even though the difference are not that significant. The elastic net also has a slightly lower standard error than the lasso, and picks on average more variables than the lasso. For full results, please see Table 5 in the appendix.

The more interesting results in this scenario are the estimated values of β_1 and β_2, summarised in the Table 2 below. **When $X_1 = X_2$, we find that $\hat{\beta}_1 = \hat{\beta}_2$, which could be explained by theory.** Because when $X_1 = X_2$, $\hat{\beta}_1 + \hat{\beta}_2$ is close to $\beta_1 + \beta_2$. In elastic net model, the MSE keeps same no matter how $\hat{\beta}_1$ changes, but the penalty is minimal when $\hat{\beta}_1 = \hat{\beta}_2$. In lasso, both the MSE and penalty keep same because $\hat{\beta}_1 + \hat{\beta}_2$ is close to a constant, which means there are many possible solutions and our algorithm provides only one solution. **Besides, as the correlation increases, we find that the difference between the lasso's estimates of β_1 and β_2 narrows. The same is true for the elastic net, though to a lesser extent.** (See the Appendix for numerical results.)

Table 2: Varying Pairwise Correlations

	Lasso		Elastic net	
Pairwise Correlations	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_1$	$\hat{\beta}_2$
0.1	2.4	1.1	2.3	1.1
0.2	2.5	1.1	2.4	1.1
0.3	2.5	1.2	2.4	1.2
0.4	2.5	1.2	2.4	1.2
0.5	**2.5**	**1.2**	**2.4**	**1.3**
0.6	2.5	1.3	2.4	1.3
0.7	2.5	1.3	2.4	1.4
0.8	2.5	1.4	2.3	1.4
0.9	2.3	1.6	2.3	1.5
$X_1 = X_2$	2.0	2.0	1.9	1.9

6 Additional Simulation Settings

6.1 Variance Parameter σ

We want to investigate the relationship between the parameter sigma σ and both the lasso penalty and the elastic net penalty. This parameter introduce randomness and volatility into our simulated data Y. The higher the coefficient of σ, the more unpredictable and random our Y is. We have tabulated the result in Table 3 of running the algorithm with different levels of sigma while keeping the other parameters constant.

We observ that when we decrease the variance of our simulation, the elastic penalties converge into the lasso penalties with $\hat{\lambda}_2^{elastic} = 0$. And the elastic net doesn't yield results greater than the lasso problem. However, if we were to reverse the process and **increase the randomness parameter σ, the mean MSE of elastic net shows superior result compared to the MSE of lasso**. This is proven to be true when $\hat{\lambda}_2^{elastic}$ is statistically greater than zero. The parameter σ also negatively correlates with the number of estimated non-zero predictors. **High σ will result in a better variable selection** as we can see that the number decreases to the true number of variables. We can conclude that the **higher the parameter σ, the more likely we are to choose elastic net over lasso penalties**.

Table 3: Varying the Variance Parameter σ

	Lasso			Elastic net		
σ	MSE	MSE st. err.	# of vars	MSE	MSE st. err.	# of vars
0.5	0.52	0.03	7.2	0.52	0.03	7.2
1	1.55	0.07	6.2	1.55	0.07	6.2
2	5.56	0.22	6.2	5.57	0.21	6.3
3	**12.10**	**0.39**	**5.9**	**12.00**	**0.37**	**6.5**
4	20.95	0.62	5.3	20.42	0.55	6.4
5	31.80	0.81	4.6	30.84	0.67	6.0

6.2 Sparsity Level of β

We also investigate the impact of sparsity on the performance of the lasso and elastic net. Here we measure sparsity as the number of coefficients that are equal to zero. The process is simple: we begin with a coefficient vector that contains only non-zero coefficients (sparsity = 0). We run 50 simulations on this model and then increase sparsity to 1. We repeat this process until sparsity = $p - 1$ (all coefficients except one is equal to zero).

Results of these simulations are visualised in Figure 3 and Figure 4 below. The MSE of predictions is consistently lower under the elastic net specification, although this result may not be statistically significant. The similar performance of the lasso is interesting considering that the lasso consistently comes closer to choosing the "true" number of non-zero coefficients. **Both models systematically select too many non-zero coefficients (not enough sparsity), but the lasso consistently picks a more sparse model than the elastic net.**

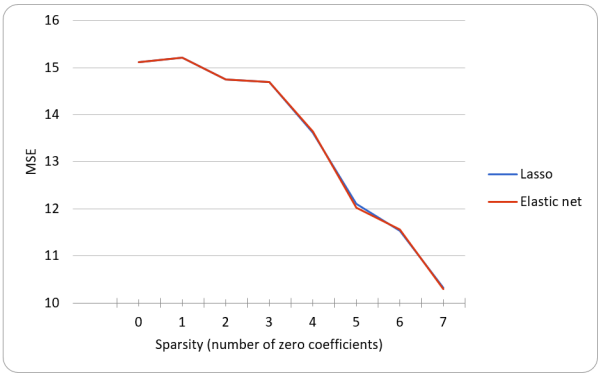


Figure 3: Mean MSE by Sparsity

Figure 4: Non-Zero Coefficients by Sparsity

The reason that the lasso picks a sparser model is illustrated in Figure 5. The elastic net typically selects a value for $\hat{\lambda}_1^{elastic}$ that is smaller than $\hat{\lambda}^{lasso}$. Accordingly, the elastic net assigns some small, non-zero value to $\hat{\lambda}_2^{elastic}$ (the tuning parameter representing the Ridge regression). Since Ridge regression does not do variable selection (i.e. it does not force coefficients to be exactly zero), this re-weighting of tuning parameters biases the elastic net towards less sparsity.

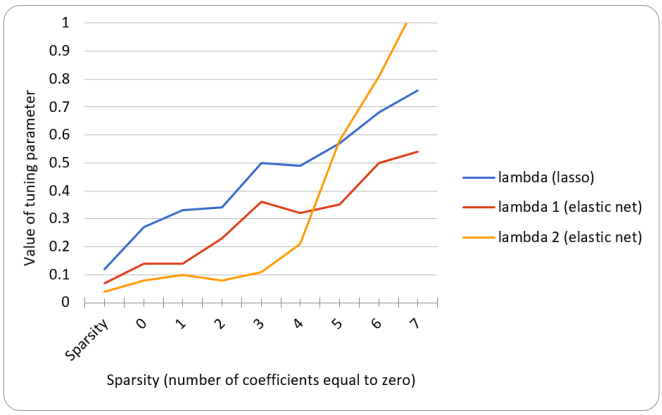


Figure 5: Tuning Parameters by Sparsity

6.3 Number of Simulations

In this subsection, we investigated how our model estimates shift if we change the number of times the process is looped. In the results tabulated in Table 4, we can see that the MSEs maintain roughly the same value disregarding the number of times it is looped. This is because the number of iterations doesn't heavily affect the estimated parameters. Instead, **the number of iterations is positively correlated to the reliability of the estimated parameters**. For example, one can take a look at the results simulated with 150 runs against the base case. The standard error of the MSE computed is strongly reduced for both lasso and elastic net. Not only that, the number of estimated non-zero coefficients also falls towards to the true value, which is 3.

Table 4: Varying the Number of Simulations r

	Lasso			Elastic net		
r	MSE	MSE st. err.	# of vars	MSE	MSE st. err.	# of vars
10	13.00	1.22	6.0	12.60	1.26	6.5
25	12.20	0.56	5.8	12.00	0.57	6.4
50	**12.10**	**0.39**	**5.9**	**12.00**	**0.37**	**6.5**
75	12.20	0.30	5.8	12.10	0.28	6.5
100	12.30	0.26	5.6	12.10	0.24	6.3
150	12.40	0.21	5.5	12.20	0.19	6.2

7 Conclusion

Summarising our findings, we first simulated data sets using the default case provided in the exercise sheet. The default case is useful as it can act as a proxy setting to test out other simulation settings. We set a constant seed of 16 throughout our entire project in order to generate reproducible results. The data sets are then split into three subsets with different purposes: training data to fit models, validation data to select tuning parameters and testing data to examine performance indicators.

We first computed an algorithm to solve the Lasso and the Elastic Net problem using the Coordinate Descent Algorithm. The Coordinate Descent Algorithm solved using the $soft thresholding$ is under the assumption of orthonormality. In order to apply our algorithm, we **scaled** X to generate the standardized X.

To select the optimal tuning parameters $\hat{\lambda}$, we tested in the range of $(0, 5)$. The best $\hat{\lambda}$ is associated with the lowest MSE when tested against the validation data set. In the interest of limiting computing time, we choose a sequence of lambdas that is not very granular. A more granular sequence of lambdas to test would give us more precise values for the tuning parameters, while requiring larger computing power. We could also increase our reliability of estimated parameters by performing Monte-Carlo Simulations with a sizeable number of runs.

By running though our default case 50 times, the result showed elastic net as the superior penalty compared to the lasso. This is reflected in a lower standard error in the MSE computed through the 50 iterations, pointing towards a **more stable computation**. The mean of the MSE is also reportedly to be lower in elastic net leading us to believe that elastic net **dominates lasso in terms of prediction accuracy**. Not only that, we can empirically observe that the elastic net **performs fairly well at variable selection**.

We proceed to run our algorithm with different settings and record the results. For $n<p$ scenario, we observed that the lasso selects variables with a maximum value equals to the size of the validation data before converging. In special case where our predictors are highly correlated, we find that the difference between both of the lasso and the elastic net's estimates of β_1 and β_2 narrows as the correlation increases. Especially when $X_1 = X_2$, we find that $\hat{\beta}_1 = \hat{\beta}_2$, which we also explain it by theory.

We also carried out additional simulation settings to investigate the relationship of various parameters against our results. Looking at the number of data size, the elastic net shows to be more advantageous than the lasso where data is scarce. Other than that, the elastic net is also the preferred regularization when it comes to an increased number of predictors. The prediction accuracy of elastic net compared to the lasso is improved where the simulated data is highly random.

One interesting point to note here is when we increase the sparsity of β, we will have a trade-off between prediction accuracy and variable selection. Elastic net will generally give us a better prediction accuracy compared to the lasso. However, when the predictors are sparse enough (coefficients of $\beta = 0$), lasso tends to choose the "true" number of non-zero coefficients due to it's coefficient reduction property. Elastic net is a combination of both the lasso and the ridge regression penalty. The ridge regression is not capable in performing variables selection hence it reduces the ability of the elastic net to select variables.

Appendices

A References

Friedman, J., Hastic. and Hofling, H. (2007). Pathwise coordinate optimization, *The Annals of Applied Statistics* **1**: 302 - 332.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso, *Journal of Royal Statistical Society, Series B* **58**: 267 - 288.

Tseng, P.(1988). Coordinate ascent for maximizing nondifferentiable concave functions, *Technical Report.*

Tibshirani, R. (2001) *Convergence of block coordinate descent method for nondifferentiable maximization, J.Opt. Theory Appl.* **109**: 474 - 494.

B Figures and Tables

Table 5: Varying the Pairwise Correlations

	Lasso			Elastic net		
Pairwise Correlations	MSE	MSE st. err.	# of vars	MSE	MSE st. err.	# of vars
0.1	12.13	0.38	5.8	12.10	0.37	6.1
0.2	12.10	0.38	5.8	12.00	0.37	6.2
0.3	12.00	0.38	5.8	11.89	0.37	6.2
0.4	11.94	0.38	5.7	11.86	0.38	6.2
0.5	**12.1**	**0.39**	**5.9**	**12.0**	**0.37**	**6.5**
0.6	12.03	0.45	5.8	11.95	0.42	6.2
0.7	11.76	0.38	5.7	11.75	0.38	6.2
0.8	11.65	0.38	5.8	11.67	0.37	6.2
0.9	11.59	0.40	5.7	11.58	0.39	6.2
$X_1 = X_2$	11.57	0.38	5.9	11.73	0.38	6.4

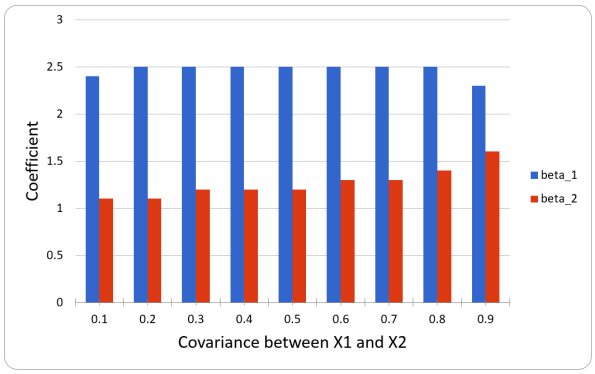


Figure 6: Estimated β_1 and β_2 for Lasso

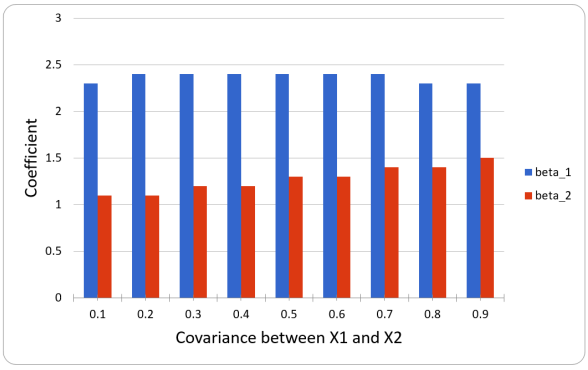


Figure 7: Estimated β_1 and β_2 for Elastic Net

Table 6: Varying Sparsity of β

	Lasso			Elastic net		
sparsity of β	MSE	MSE st. err.	# of vars	MSE	MSE st. err.	# of vars
0	15.10	0.67	7.9	15.10	0.67	7.9
1	15.20	0.67	7.8	15.20	0.67	7.8
2	14.80	0.65	7.5	14.80	0.65	7.5
3	14.70	0.71	7.3	14.70	0.71	7.3
4	13.60	0.52	6.8	13.60	0.51	6.9
5	**12.10**	**0.39**	**5.9**	**12.00**	**0.37**	**6.5**
6	11.30	0.28	4.8	11.10	0.26	6.0
7	10.40	0.22	3.1	10.10	0.18	4.5

C R Codes

```
# Load packages
library(MASS)
library(stats)

#########################
### lasso algorithm ###
#########################

lasso_alg = function(X,Y,lambda){
  std_x = apply(X, 2, sd)
  X = scale(X) # scale
  n = nrow(X)
  p = ncol(X)
  b = rep(0, time = p)
  b_new = rep(0, time = p)
  b_old = rep(0, time = p)
  r = rep(0, time = n)
  dif = 1
  k = 0
  while(dif >= 10^(-10)){ # converge setting
    b_old = b
    for(j in 1:p){
      r = Y - X%*%b + X[,j]*b[j]
      b_j = X[,j]%*%r/n
      b_new[j] = sign(b_j)*max(0, abs(b_j) - lambda)
      b[j] = b_new[j]
      k = k + 1
    }
    dif = sqrt(sum((b_new-b_old)^2))
  }
```

```
  b = b/std_x # re-scale
  return(b)
}

##############################
### elastic net algorithm ###
##############################

elastic_net_alg = function(X,Y,lambda,lambda2){
  std_x = apply(X, 2, sd)
  X = scale(X)
  n = nrow(X)
  p = ncol(X)
  b = rep(0, time = p)
  b_new = rep(0, time = p)
  b_old = rep(0, time = p)
  r = rep(0, time = n)
  dif = 1
  k = 0
  while(dif >= 10^(-10)){
    b_old = b
    for(j in 1:p){
      r = Y - X%*%b + X[,j]*b[j]
      b_j = X[,j]%*%r/n
      b_new[j] = (sign(b_j)*max(0, abs(b_j) - lambda))/(1 + 2*lambda2)
      b[j] = b_new[j]
      k = k + 1
    }
    dif = sqrt(sum((b_new-b_old)^2))
  }
  b = b/std_x
  return(b)
}

############################
### evaluation function ###
############################

evaluation = function(b,X,Y){
  MSE = sum((Y-X%*%b)^2)/length(Y)
  num = 0
  for(i in 1:length(b)){
    if(round(b[i],2)!=0){
      num = num +1
    }
  }
  result = list(MSE=MSE, num=num)
  return(result)
```

```
}

################ select the regularization parameters

#############################################
### 1) Fcn to choosing lambda for lasso ###
#############################################

lambda_lasso = function(lambda_range,X_train,Y_train,X_validation,Y_validation
,X_test,Y_test){
  lasso_MSE = rep(0, time = length(lambda_range))
  for(i in 1:length(lambda_range)){
    fit_lasso = lasso_alg(X_train,Y_train,lambda_range[i])
    lasso_MSE[i] = evaluation(fit_lasso,X_validation,Y_validation)$MSE
  }
  lambda_best = lambda_range[which.min(lasso_MSE)]
  b_lasso = lasso_alg(X_train,Y_train,lambda_best)
  lasso_mse = evaluation(b_lasso,X_test,Y_test)$MSE
  lasso_num = evaluation(b_lasso,X_test,Y_test)$num
  result = list(lasso_MSE=lasso_MSE, lambda_best=lambda_best,
                b_lasso=b_lasso, lasso_mse=lasso_mse, lasso_num=lasso_num)
  return(result)
}

#########################################################
### 2) Fcn to choose lambda, lambda2 for elastic_net ###
#########################################################

lambda_elastic_net = function(lambda_range,lambda2_range,X_train,Y_train,
X_validation,Y_validation,X_test,Y_test){
  elastic_net_MSE =matrix(0, length(lambda_range), length(lambda2_range))
  for(i in 1:length(lambda_range)){
    for(j in 1:length(lambda2_range)){
      fit_elastic_net = elastic_net_alg(X_train,Y_train,lambda_range[i],
      lambda2_range[j])
      elastic_net_MSE[i,j] = evaluation(fit_elastic_net,X_validation,
      Y_validation)$MSE
    }
  }
  lambda_best = lambda_range[which(elastic_net_MSE == min(elastic_net_MSE),
  arr.ind = TRUE)[1]]
  lambda2_best = lambda2_range[which(elastic_net_MSE == min(elastic_net_MSE),
  arr.ind = TRUE)[2]]
  b_elastic_net = elastic_net_alg(X_train,Y_train,lambda_best,lambda2_best)
  elastic_net_mse = evaluation(b_elastic_net,X_test,Y_test)$MSE
  elastic_net_num = evaluation(b_elastic_net,X_test,Y_test)$num
  result = list(elastic_net_MSE=elastic_net_MSE, lambda_best=lambda_best,
                lambda2_best=lambda2_best, b_elastic_net=b_elastic_net,
```

```
                  elastic_net_mse=elastic_net_mse,
                  elastic_net_num=elastic_net_num)
  return(result)
}

##############################
### Sigma_create function ###
##############################

Sigma_create = function(p,cor,s){
  sigma <- matrix(NA,p,p)
  for(i in 1:p){
    for(j in 1:p){
      if(i==j) sigma[i,j] = s[i] # variance of the j-th component
      else sigma[i,j] = sqrt(s[i]*s[j])*cor^abs(i-j)
      # covar between  i-th and j-th components
    }
  }
  return(sigma)
}

# ------------------------------------------------------------------------------

####################################
### Fcn Running entire programme ###
####################################

simulation = function(ns,ps,mus,ss,cors,b_trues,s_epss,ks){

  set.seed(16)

  # Set parameters
  Sigma = Sigma_create(ps,cors,ss)

  #if(ks!=0) Sigma[1,2] = Sigma[2,1] = ks

  n_train = 20
  n_val = 20

  ### Create variables for loops
  r = 50 # Number of repetitions

  sim_lasso_l = rep(NA, r)
  sim_elastic_l1 = rep(NA, r)
  sim_elastic_l2 = rep(NA, r)

  sim_lasso_mse = rep(NA, r)
  sim_lasso_num = rep(NA, r)
```

```
sim_elastic_mse = rep(NA, r)
sim_elastic_num = rep(NA, r)

sim_lasso_b = matrix(NA, r, p)
sim_elastic_b = matrix(NA, r, p)
i = 0
# Start loop
while (i <= r){

  # Simulate data

  # Simulate predictors (X)
  X <- mvrnorm(ns, mus, Sigma)
  if(ks!=0) X[,1]=ks*X[,2]

  # Subset predictors into train, val, test
  X_train = X[1:n_train,]
  X_validation = X[(n_train+1):(n_train+n_val),]
  X_test = X[-(1:(n_train+n_val)),]

  # Generate Y based on X
  Y = X%*%b_trues + rnorm(ns, mean=0, sd=1)*s_eps
  Y_train = Y[1:n_train]
  Y_validation = Y[(n_train+1):(n_train+n_val)]
  Y_test = Y[-(1:(n_train+n_val))]

  # Lasso

  lambda_range = seq(0,2,length=21)

  # Find optimal lambda, MSE and beta
  lasso = lambda_lasso(lambda_range,X_train,Y_train,
                       X_validation,Y_validation,X_test,Y_test)

  # Store results
  sim_lasso_l[i] = lasso$lambda_best
  sim_lasso_mse[i] = lasso$lasso_mse
  sim_lasso_num[i] = lasso$lasso_num
  sim_lasso_b[i,] = lasso$b_lasso

  # Graphical analysis
  #plot(lambda_range, lasso$lasso_MSE, ylab="lasso_MSE", xlab="lambda",
  #     pch=19, type="b")
  #points(lasso$lambda_best, lasso$lasso_MSE[which.min(lasso$lasso_MSE)],
  #       col="red", cex=2, pch=20)

  # Elastic net
```

```
  lambda_range = seq(0,2,length=21)
  lambda2_range = seq(0,2,length=21)

  # Find optimal lambdas, MSE and beta
  elastic_net = lambda_elastic_net(lambda_range,lambda2_range,
                                   X_train,Y_train,
                                   X_validation,Y_validation,X_test,Y_test)

  # Store results
  # print(elastic_net$lambda2_best)
  sim_elastic_l1[i] = elastic_net$lambda_best
  sim_elastic_l2[i] = elastic_net$lambda2_best
  sim_elastic_mse[i] = elastic_net$elastic_net_mse
  sim_elastic_num[i] = elastic_net$elastic_net_num
  sim_elastic_b[i,] = elastic_net$b_elastic_net

  # Graphical analysis
  #pmat = persp(lambda_range,lambda2_range,elastic_net$elastic_net_MSE)
  #mypoints = trans3d(elastic_net$lambda_best, elastic_net$lambda2_best,
  #                   elastic_net$elastic_net_MSE
  [which(elastic_net$elastic_net_MSE ==
  min(elastic_net$elastic_net_MSE), arr.ind = TRUE)],
  #                   pmat=pmat)
  #points(mypoints, col="red", cex=2, pch=20)
  i = i + 1
}

# Report average simulation results
result_lasso_l = mean(sim_lasso_l)
result_elastic_l1 = mean(sim_elastic_l1)
print(sim_elastic_l2)
result_elastic_l2 = mean(sim_elastic_l2)

result_lasso_mse = mean(sim_lasso_mse)
result_elastic_mse = mean(sim_elastic_mse)

result_lasso_mse_se = sd(sim_lasso_mse)/sqrt(length(sim_lasso_mse))
result_elastic_mse_se = sd(sim_elastic_mse)/sqrt(length(sim_elastic_mse))

result_lasso_num = mean(sim_lasso_num)
result_elastic_num = mean(sim_elastic_num)

result_lasso_b = colMeans(sim_lasso_b)
result_elastic_b = colMeans(sim_elastic_b)

result = list(result_lasso_l=result_lasso_l,
result_elastic_l1=result_elastic_l1,
result_elastic_l2=result_elastic_l2,
```

```
                    result_lasso_mse=result_lasso_mse,
                    result_elastic_mse=result_elastic_mse,
                    result_lasso_mse_se=result_lasso_mse_se,
                    result_elastic_mse_se=result_elastic_mse_se,
                    result_lasso_num=result_lasso_num,
                    result_elastic_num=result_elastic_num,
                    result_lasso_b=result_lasso_b,
                    result_elastic_b=result_elastic_b)
  return(result)
}
#-------------------------------------------------------------------------------

# Set parameters -1
n = 240
p = 8
b_true = c(3, 1.5, 0, 0, 2, 0, 0, 0)

mu = rep(0, time = p)
s = rep(1, time = p)
cor = 0.5

s_eps = 3

k = 0

sim1 = simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor,
b_trues=b_true,s_epss=s_eps,ks=k)

# Set parameters -2
n = 240
p = 25
b_true = sample(seq(0,10,1)/10, p, replace = TRUE)

mu = rep(0, time = p)
s = rep(1, time = p)
cor = 0.5

s_eps = 3

k = 0
sim2_zero = simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor,
b_trues=b_true,s_epss=s_eps,ks=k)

# Set parameters -3.1
n = 240
p = 8
b_true = c(3, 1.5, 0, 0, 2, 0, 0, 0)
```

```
mu = rep(0, time = p)
s = rep(1, time = p)
cor = 0

s_eps = 3

k = seq(-1,1,0.2)
sim3_1 = list()
for(i in 1:length(k)){
  sim3_1[[length(sim3_1)+1]] <- simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor,
  b_trues=b_true,s_epss=s_eps,ks=k[i])
}


# Set parameters -3.2
n = 240
p = 8
b_true = c(3, 1.5, 0, 0, 2, 0, 0, 0)

mu = rep(0, time = p)
s = rep(1, time = p)
cor = 0

s_eps = 3

k = seq(0.1,0.9,0.1)
sim3_2zero = list()
for(i in 1:length(k)){
  sim3_2zero[[length(sim3_2zero)+1]] <- simulation(ns=n,ps=p,mus=mu,ss=s,
  cors=cor, b_trues=b_true,s_epss=s_eps,ks=k[i])
}

# Set parameters -4.1
# n = 240
# p = 8
# b_true = c(3, 1.5, 0, 0, 2, 0, 0, 0)
#
# mu = rep(0, time = p)
# s = rep(1, time = p)
# #cor = 0.5
#
# s_eps = 3
# k = 0
#
# cor = seq(0,0.9,0.1)
# sim4_1 = list()
# for(i in 1:length(cor)){
#   sim4_1[[length(sim4_1)+1]] = simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor[i],
```

```
b_trues=b_true, s_epss=s_eps, ks=k)
# }

# Set parameters -4.2
n = 240
p = 8
#b_true = c(0, 0, 0.5, 1, 1.5, 2, 2.5, 3)

b_true = matrix(rep(sample(seq(1,10,1)/4, p, replace = FALSE),p),ncol=p,byrow=T)
for(i in 1:p){
  for(j in 1:p){
    if(i<j) b_true[i,j] = 0
  }
}

mu = rep(0, time = p)
s = rep(1, time = p)
cor = 0.5

s_eps = 3
k = 0

sim4_2 = list()
for(i in 1:p){
  sim4_2[[length(sim4_2)+1]] = simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor,
  b_trues=b_true[i,],s_epss=s_eps,ks=k)
}

# Set parameters -4.3
n = 240
p = 8
b_true = c(3, 1.5, 0, 0, 2, 0, 0, 0)

mu = rep(0, time = p)
s = rep(10, time = p)
cor = 0.5

#s_eps = 3
k = 0

s_eps = seq(1,20,2)
sim4_3 = list()
for(i in 1:length(s_eps)){
  sim4_3[[length(sim4_3)+1]] = simulation(ns=n,ps=p,mus=mu,ss=s,cors=cor,
  b_trues=b_true,s_epss=s_eps[i],ks=k)
}

################ package
```

```
# library(glmnet)
# lasso.train <-glmnet(X_train, Y_train)
# pred.test <-predict(lasso.train,X_validation)
# rMSE <-sqrt(apply((Y_validation-pred.test)^2,2,mean))
# plot(log(lasso.train$lambda), rMSE, type="b", xlab="log(lambda)")
# lambda.best <-lasso.train$lambda[order(rMSE)[1]]
# lambda.best
```