Group Project ST443 - Machine Learning and Data Mining

Group 5

Contents

1 Real World Data

A well-known business problem of companies in every industry is related to the “customer churn”. This phenomenon happens if customer of a certain firm decides to stop doing business with it. Usually, for companies is less costly to retain a customer with respect to obtain a new one.

The dataset that we decided to analyse contains information on 3150 customers of an Iranian telecommunication company aggregated over a time period of 9 months. The variable we are interested in predicting is Churn, which is binary and it assumes the value 0 if the company successfully retained the respective customer after 12 months from when this study began, and the value 1 if the customer left the company. We are interested in studying the possibility of whether a customer will “churn” or not. Being able to correctly predict this fact could lead to smaller customers loss for firms, and, hence, to major profits and customer loyalty.

Below, we explore the variables present in the data set. First, we notice that there are not missing values and that there is a class imbalance in favour of the 0 class (not churn). Second, from the names of the columns (*Call Failure, Complains, Subscription, Length, Charge Amount, Seconds of Use, Frequency of use, Frequency of SMS, Distinct Called Numbers, Age Group, Tariff Plan, Status, Age, Customer Value, Churn*) we notice that two of them are related to the age of customers: Age and Age.Group, with the latter being a transformation of Age that we decided to not use. From the set of 12 predictors that we are left with, we are interested in the ones which have the largest impact on the probability of a customer to churn. We expect that these variables might be Complains and Status. A high number of complaints for a customer usually translate to a bad experience with the services, therefore, we expect that a positive correlation with the probability to churn. Furthermore, the status of the customer, being active or non-active, might reflect the familiarity that a customer has with the service: we expect that a non-active customer is less likely to switch to a different company with respect to someone who frequently uses the service and is more aware of the quality of the service that they are receiving.

```r
lapply(c('ggplot2','plotly','class','lattice','caret','pROC','MASS','QuantPsyc'),
       require,character.only=T)
df=read.csv("customer_churn.csv")
```

```r
sum(is.na(df)) # number of missing values is zero
```

```
## [1] 0
```

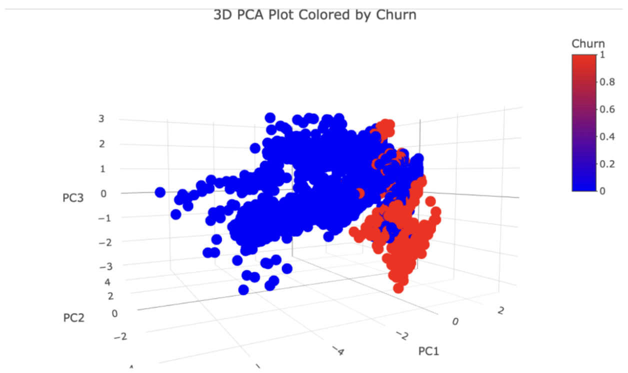
```r
sum(df$Churn==0)/nrow(df) # percentage of observation that did not churn: 84%
```

```
## [1] 0.8428571
```

```r
df<-subset(df, select = -Age.Group) # deleting age group
set.seed(301)
train_index = sample(1:nrow(df),nrow(df)*80/100)
# This is to split the data 80/20 for training and testing respectively.
train = df[train_index,]
test = df[-train_index,]
```

1.1 Data Visualisation

The plot below is obtained after applying PCA on our data. In particular, the first 3 components, which explain over 60% of the variance, were extracted. This dimensionality reduction allow us to visualise and understand the structure of the data. As we can see from the 3D plot, the decision boundary between the 2 classes of churn looks to be moderately non-linear. Hence, we expect that QDA might be the best classification approach for this data.



1.2 Methods

Given that our goal is to evaluate the performance of different classification methods on our dataset, the approaches we will study are the following: kNN, LDA/QDA and Logistic Regression.

1.2.1 kNN

The tuning of the hyperparameter k is done by performing a 5-fold cross-validation and the aim is to find the value of k that minimizes the validation error. Comparing the true value of the response on the row with the predicted ones on the columns, the following confusion matrix is obtained:

	0	1
0	508	37
1	28	57

From the table we can compute an accuracy of $\frac{508+57}{630} = 89.7\%$, a True Positive Rate (TPR), or sensitivity, of $\frac{57}{57+28} = 67.1\%$ and a False Positive Rate (FPR) of $\frac{37}{37+508} = 6.8\%$. The metrics considered to evaluate and compare the performance are the values of AUC and MER.

1.2.2 Logistic regression

The key hyperparameter in logistic regression is the number of variables selected. The tuning is done by performing a forward step-wise selection, starting from the null model and subsequently adding variables based on AIC minimization. The forward step-wise algorithm finds a local optimum model, whereas the best subset selection method finds the global optimum. However, the latter would be much more computationally intensive, since it would need to fit and evaluate $2^p = 2^{12}$= 4096 models.

```r
set.seed(300)
min.model <- glm(Churn ~ 1, family=binomial(link = "logit"), data=train)
max.model <- glm(Churn ~ ., family=binomial(link = "logit"), data=train)
glm.step <- step(min.model, scope = list(lower = min.model, upper = max.model),
              direction="forward", trace = 0)
summary(glm.step)$coefficients
```

```
##                           Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept)            -2.67412110 0.547001246 -4.888693 1.015077e-06
## Status                  1.47326530 0.222600734  6.618421 3.630566e-11
## Complains               4.12497120 0.314784638 13.104106 3.119168e-39
## Customer.Value          0.01066074 0.002335864  4.563941 5.020225e-06
## Distinct.Called.Numbers -0.01052468 0.010446738 -1.007461 3.137132e-01
## Subscription..Length   -0.03513314 0.010684660 -3.288185 1.008355e-03
## Call..Failure           0.12878561 0.019061635  6.756273 1.415868e-11
## Charge..Amount          -0.42302481 0.114170754 -3.705194 2.112289e-04
## Frequency.of.use        -0.05365938 0.008605294 -6.235625 4.499774e-10
## Frequency.of.SMS        -0.05925049 0.010410550 -5.691389 1.260099e-08
## Age                      0.02371104 0.011483023  2.064878 3.893459e-02
```

The analysis of the logistic regression output highlights that the predictor which has the most impact on the response variable is Complains, which meets our initial expectations.
Moreover, the results support our intuition that churning is most likely to take place among active users than inactive ones.

Again a confusion matrix is obtained:

	0	1
0	525	20
1	48	37

with values of accuracy, TPR and FPR equal to 89.2%, 43.5% and 3.7% respectively.

1.2.3 LDA and QDA

In this section we perform linear and quadratic discriminant analysis. Hence, we omit any discrete variables from our model, since discriminant analysis assumes that the data, given a class, follows the Multivariate Normal (MVN) distribution.

	0	1
0	536	9
1	74	11

LDA has an accuracy of 86.8%, a TPR of 12.9% and a FPR of 1.7%. Now the results for QDA are presented:

	0	1
0	372	173
1	12	73

QDA has an overall accuracy of 70.6%, a TPR of 85.9% and a FPR of 31.7%.

1.3 Conclusion

The table below shows that the best approach for the analysed dataset is Logistic Regression, with an AUC value 92%.

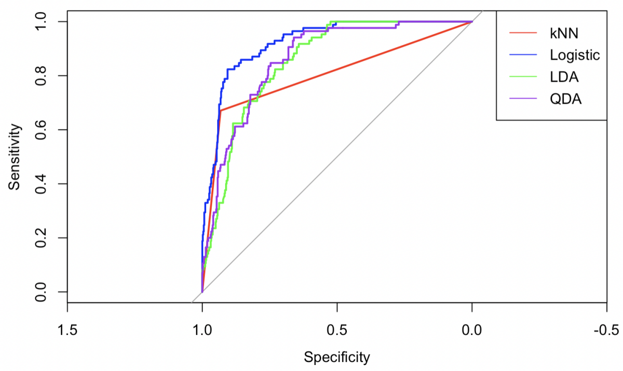
	KNN	Logistic Regression	LDA	QDA
MER	0.1	0.11	0.13	0.29
AUC	0.8	0.92	0.85	0.86
Profit	1295.6	639.1	249	-5062.2

In addition, our initial expectation about QDA outperforming did not occur. A reasonable explanation might be that, while moderately non-linear data is an indicator of the suitability of QDA, discriminant analysis requires also the assumption that the data, given each class, is drawn from an MVN distribution. We can check the normality of data by performing a Mardia test, which has as null hypothesis the normality of data. As we can see from the output, the p-value is 0, which means that the data is not normal.

```
data_without_discrete<-subset(df[df$Churn==0,], select = -c(Complains, Charge..Amount,
                                                          Tariff.Plan, Status, Churn))
mult.norm(data_without_discrete)$mult.test
```

```
##             Beta-hat      kappa p-val
## Skewness  42.16072 18656.1194     0
## Kurtosis 129.55206   100.9262     0
```

We perform this test only on the observations where churn is 0, as normality should hold for all the observations in a given class. The combination of a lack of normality and linearity explains why LDA and QDA yielded a relatively small AUC value. It follows a graphical representation of the ROCs of the analysed methods:



We can see how the blue line, corresponding to the logistic regression, covers the greatest area.

Lastly, we would like to propose a metric to assess the performance of each method in terms of profit and cost. The telecom company plans to use the prediction to intervene offering a special discount to a potentially churning customer. This policy will lead to a cost, which we assume to be the 40% of the value of the customer. For customers which would have ultimately churned, a correct prediction will result in positive returns, as it is more costly to acquire a new customer than replacing them. However, for customers who would not have churned, a wrong prediction would lead to an economic loss for the company. Hence, we can model the profit generated by the company using a method, called A, that works as follows: $\pi(A) = \sum_{i=1}^{N}[v_i * I(y_i = \hat{y_i} = 0) - 0.4 * v_i * I(\hat{y_i} = 0)]$, where N is the number of data points in the test set. We notice that the formula suggests that the profits generated by the model are the sum of the values of all customers that we predict as staying and are actually doing that, minus a proportion of the values of who would have left, but did not, thanks to the company intervention that we assume to be always effective. Indeed, a lack of intervention by the company would result in all of them churning and, in turn, to the loss of the sum of their values. Nevertheless, this formula is not contemplating the economic loss of the customers churning, despite the prediction of them staying and in this case we are largely approximating the estimate based on the available data.

The results in the summary table above show that the method which the company should use is kNN, as it will maximise its profits, measured in terms of units of customer value. Despite the higher AUC value for logistic, the fact that kNN has a higher accuracy allows to avoid more losses. Also, it is interesting to see how the QDA yields a strongly negative profit value. This is explainable by noticing that QDA has a particularly high value of FPR compared to the other methods. Hence, the company would end up in spending a lot of money to retain customers that would have stayed in any case.

For future exploration, alternative methods such as Random Forests, Decision Trees, Bagging, Boosting and SVM could be considered to further enhance predictive accuracy, to explore the interplay of various factors in customer churn dynamics and to capture better the potential relationship between costumer churn and economic profit. In addition, we believe that the dataset we worked on does not include predictors which would be significant for customers of today. For instance, the speed at which clients access online contents could reflect their level of satisfaction towards the service provided.

Finally, in the profit model we make the assumption that the company intervention on predicted-to-churn customers is always effective. This is a strong assumption, and, for instance, the effectiveness of the policy of the firm might be represented by a probability distribution.

2 Coordinate descent algorithms

In this section we will apply coordinate descent algorithms on penalized regression problems, in particular on the LASSO and on the elastic net. This algorithm permits to obtain solutions for convex optimization problems in a faster way: this is due to the "one-at-a-time" method, which updates the beta components as we iteratively examine each predictor (Friedman, 2007). In the following section we show the code for the algorithm applied in the case of LASSO and elastic net.

2.1 Implementation of the algorithms

```
# First algorithm: lasso coordinate descent
lasso_coordinate_descent <- function(X, y, lambda, max_iter, tolerance) {
  n = length(y)
  p = ncol(X)
  beta0 = rep(0 ,p)
  beta_old = beta0

  for (iter in 1:max_iter) {
    for (j in 1:p) {
      # Compute partial residuals
      rij = y - X %*% beta0 + X[, j] * beta0[j]

      # Compute simple least squares coefficient of these residuals
      beta_star = sum(X[, j] * rij) / (n)

      # Update beta_j by soft thresholding
      beta0[j] = sign(beta_star) * max(0, abs(beta_star) - lambda)
    }
    # Check for convergence
    if ((sum((beta0 - beta_old)^2)) <= tolerance) {
      break
    }
    else{
      if(sum((beta0 - beta_old)^2)>1000){
         return(rep(0,p))}
      }
    beta_old = beta0

  }
  return(beta0)
}
```

In the code, n indicates the number of observation, p is the number of predictors, λ is the tuning parameter, while *tolerance* is an arbitrary small value used to check the convergence of the $\boldsymbol{\beta}$.

```
# Second algorithm: elastic net coordinate descent
el_coordinate_descent <- function(X, y, lambda1, lambda2, max_iter, tolerance) {
  n = length(y)
  p = ncol(X)
  beta0 = rep(0 ,p)
  beta_old = beta0

  for (iter in 1:max_iter) {
    for (j in 1:p) {
      # Compute partial residuals
      rij = y - X %*% beta0 + X[, j] * beta0[j]

      # Compute simple least squares coefficient of these residuals
      beta_star = sum(X[, j] * rij) / (n)

      # Update beta_j by soft thresholding
      beta0[j] = sign(beta_star) * max(0, (abs(beta_star) - lambda2)) / (1 + 2 * lambda1)
    }
    # Check for convergence
    if ((sum((beta0 - beta_old)^2)) <= tolerance) {
      break
    }
    else{
      if(sum((beta0 - beta_old)^2)>1000){
         return(rep(0,p))}
      }
    beta_old = beta0
  }
  return(beta0)
}
```

The main and only difference between the two algorithms is located in the rule of soft thresholding.

While executing these algorithms, we notice that, in some cases, the beta coefficients had the tendency to inflate at each iteration. This resulted in the norm of the vector to reach considerably high values, to the point that `R` consider it to be `Inf`. By consequence, this led to the impossibility of evaluating the convergence and, to avoid this issue, we added a further check on the value of the norm: if this value becomes bigger than 1000, we set the beta components to be zeros.

Investigating the literature, we believe that our issue is related to Tseng (2001). The paper highlight that for non-(pseudo)convex function, *"the method may cycle without approaching any stationary point of f"*. For this reason, we are led to think that in cases involving specific combinations of data, predictors and hyperparameters, the function that we aim to minimise becomes nonconvex. Therefore, the coordinate descent method starts to generate bigger values and reaches infinity.

2.2 Evaluation of the performances

To evaluate the performance of our algorithms we run several simulations and consider three main scenarios with simulated data. In particular, data are generated from the model $\mathbf{y} = \mathbf{X}\beta + \sigma\epsilon$ where $\epsilon \sim N(0, \mathbf{I}_n)$. For 50 times, we generate from a multivariate normal distribution a set of 20 independent training data, a set of 20 validation data and a set of 200 testing data.

The three considered scenarios are:

- The pairwise correlation between $\mathbf{X}_i$ and $\mathbf{X}_j$ is set to be $corr(i, j) = 0.85^{|i-j|}$, which is the covariance matrix for the MVN distribution, where each predictor and the response are standardized. Moreover, the $\boldsymbol{\beta}$ is set to be $\boldsymbol{\beta} = (3, 1.5, 0, 0, 2, 0, 0, 0)^T$ and $\sigma = 3$.
- 3 predictors are considered: 2 of them have a high correlation (0.9) and the third one is nearly uncorrelated with the others.
- p progressively becomes higher than n. In this scenario a specific correlation structure has not been set and the beta values are randomly chosen, but they are never zero.

In general, when there is a considerable correlation structure between the predictors, we expect LASSO to select fewer variables than elastic net. And, particularly for the second scenario, where we have two strongly correlated predictors, we expect LASSO to select only one of the two. In addition, under the scenarios we considered, we expect elastic net to predominantly outperform LASSO, especially in the third scenario ($p > n$). This is due to the fact that LASSO should saturate and choose n predictors at most, even if more than n coefficients are known to be non-zero.

In the following sections we provide the results for each scenarios.

2.2.1 Scenario 1

```
outperf1 = 0 # amount of times elastic net MSE is worse than lasso
equalperf1 = 0 # amount of times elastic net MSE is equal to lasso
nonzero_el_list1 = c() # list of non zero coefficients for every simulation
nonzero_lasso_list1 = c()
mse_el_list1 = c()
mse_lasso_list1 = c()

for (sim in 1:50){ # Setting 50 simulations
p = 8
r = matrix(NA,p,p)
for (i in 1:p){
  for (j in 1:p){
    r[i,j]= (0.85)^(abs(i-j))
  }
}
# The double for-loop structure generates r, the covariance matrix of the
# MVN distribution between the predictors

# Generating X
n.train = 20
n.validation = 20
n.test = 200
n = n.train + n.validation + n.test

library(MASS)
X.train = mvrnorm(n=n.train, Sigma=r, mu=rep(0,p))
X.train = as.matrix(X.train)

X.validation = mvrnorm(n=n.validation, Sigma=r, mu=rep(0,p))
X.validation = as.matrix(X.validation)

X.test = mvrnorm(n=n.test, Sigma=r, mu=rep(0,p))
X.test = as.matrix(X.test)

X = rbind(X.train,X.validation,X.test)

# Setting beta, sigma and epsilon
B = c(3, 1.5, 0, 0, 2, 0, 0, 0)
sigma = 3
epsilon = mvrnorm(n=1, Sigma=diag(n), mu=rep(0,n))

# Generating Y
Y = X %*% B + sigma * epsilon
Y = as.vector(Y)
Y = scale(Y)
```

```r
y.train = Y[1 : n.train]
y.validation = Y[(n.train+1) : (n.train+n.validation)]
y.test = Y[(n.train+n.validation+1) : n]

max_iter = 1000
tolerance = 1e-4

# LASSO hyperparameter tuning
lambda_list = seq(0,1,0.01)
mse_list = c()
for (lambda in lambda_list){
  lasso_beta <- lasso_coordinate_descent(X.train, y.train, lambda=lambda,
                                         max_iter, tolerance)
  y_hat = X.validation %*% lasso_beta
  mse_list= c(mse_list, mean((y_hat-y.validation)^2))
}
# Above we are computing the MSE of the LASSO regression on the validation set for
# each value of lambda between 0 and 3, with an increase of 0.01 for each iteration

# Selecting the value of lambda that gives the lowest MSE
lambda_min_lasso = lambda_list[which.min(mse_list)]

# Once we obtained the value of lambda, we proceed to run the coordinate descent
# algorithm and to evaluate its performance by measuring the MSE on the test data.

# Running the algorithm of the coordinate descent for LASSO
lasso_beta_min <- lasso_coordinate_descent(X.train, y.train, lambda=lambda_min_lasso,
                                           max_iter, tolerance)

# Computing the predicted values for y using the beta obtained from the algorithm
y_hat_min_lasso = X.test %*% lasso_beta_min

# Computing the MSE on the test
mse_test_lasso = mean((y_hat_min_lasso-y.test)^2)

nonzero_lasso = sum(lasso_beta_min != 0) # sum of the non-zero coefficients

# We proceed analogously with the elastic net coordinate descent algorithm
mse_list_el = c()
lambda1_list = seq(0,1,0.01)
n1 = length(lambda1_list)
lambda2_list = seq(0,1,0.01)
n2 = length(lambda2_list)
a = 1
matrix = matrix(nrow = n1*n2, ncol = 3)
for (i in 1:n1){
  for (j in 1:n2){
    el_beta <- el_coordinate_descent(X.train, y.train, lambda1=lambda1_list[i],
                                     lambda2=lambda2_list[j], max_iter, tolerance)
    y_hat = X.validation %*% el_beta
    matrix[a,] = c(lambda1_list[i], lambda2_list[j], mean((y_hat-y.validation)^2))
    a = a + 1
   }
}

index = which.min(matrix[,3])
lambda_min_el = matrix[index,]

el_beta_min <- el_coordinate_descent(X.train, y.train, lambda1=lambda_min_el[1],
                                     lambda2=lambda_min_el[2], max_iter, tolerance)

y_hat_min_el = X.test %*% el_beta_min
mse_test_el = mean((y_hat_min_el-y.test)^2)

nonzero_el = sum(el_beta_min != 0)
nonzero_el_list1 = c(nonzero_el_list1, nonzero_el)
nonzero_lasso_list1 = c(nonzero_lasso_list1, nonzero_lasso)
mse_el_list1 = c(mse_el_list1, mse_test_el)
mse_lasso_list1 = c(mse_lasso_list1, mse_test_lasso)

if (mse_test_lasso > mse_test_el){
  outperf1 = outperf1 + 1
} # Counting how many times elastic net outperforms LASSO according to the MSE
if(mse_test_lasso == mse_test_el){
  equalperf1 = equalperf1 + 1
} # Counting how many times the performance of the two methods are equal
}
```

2.2.2 Scenario 2

```r
outperf2 = 0 # amount of times elastic net MSE is worse than lasso
equalperf2 = 0 # amount of times elastic net MSE is equal to lasso
nonzero_el_list2 = c() # list of non zero coefficients for every simulation
nonzero_lasso_list2 = c() # list of non zero coefficients for every simulation
mse_el_list2 = c()
mse_lasso_list2 = c()

for (sim in 1:50){
p = 3
r = matrix(NA,p,p) # Setting the correlation structure for this scenario
for (i in 1:p){
  for (j in 1:p){
    if (i==j){
      r[i,j]=1
    }else{
      if ((i==1&j==2)|(i==2&j==1)){
      r[i,j]=0.9
    }else{
      r[i,j]=0.1
    }}}}

n.train = 20
n.validation = 20
n.test = 200
n = n.train + n.validation + n.test
```

```r
X.train = mvrnorm(n=n.train, Sigma=r, mu=rep(0,p))
X.train = as.matrix(X.train)
X.validation = mvrnorm(n=n.validation, Sigma=r, mu=rep(0,p))
X.validation = as.matrix(X.validation)
X.test = mvrnorm(n=n.test, Sigma=r, mu=rep(0,p))
X.test = as.matrix(X.test)
X = rbind(X.train,X.validation,X.test)

B = c(3, 1.5, 2)
sigma = 3
epsilon = mvrnorm(n=1, Sigma=diag(n), mu=rep(0,n))

Y = X %*% B + sigma * epsilon
Y = as.vector(Y)
Y = scale(Y)

y.train = Y[1 : n.train]
y.validation = Y[(n.train+1) : (n.train+n.validation)]
y.test = Y[(n.train+n.validation+1) : n]

max_iter = 1000
tolerance = 1e-4
lambda_list = seq(0,1,0.01)
mse_list = c()
for (lambda in lambda_list){
  lasso_beta <- lasso_coordinate_descent(X.train, y.train, lambda=lambda,
                                         max_iter, tolerance)
  y_hat = X.validation %*% lasso_beta
  mse_list= c(mse_list, mean((y_hat-y.validation)^2))
}

lambda_min_lasso = lambda_list[which.min(mse_list)]
lasso_beta_min <- lasso_coordinate_descent(X.train, y.train, lambda=lambda_min_lasso,
                                           max_iter, tolerance)
y_hat_min_lasso = X.test %*% lasso_beta_min
mse_test_lasso = mean((y_hat_min_lasso-y.test)^2)
nonzero_lasso = sum(lasso_beta_min != 0)

mse_list_el = c()
lambda1_list = seq(0,1,0.01)
n1 = length(lambda1_list)
lambda2_list = seq(0,1,0.01)
n2 = length(lambda2_list)
a = 1
matrix = matrix(nrow = n1*n2, ncol = 3)
for (i in 1:n1){
  for (j in 1:n2){
    el_beta <- el_coordinate_descent(X.train, y.train, lambda1=lambda1_list[i],
                                     lambda2=lambda2_list[j], max_iter, tolerance)
    y_hat = X.validation %*% el_beta
    matrix[a,] = c(lambda1_list[i], lambda2_list[j], mean((y_hat-y.validation)^2))
    a = a + 1
   }
}

index = which.min(matrix[,3])
lambda_min_el = matrix[index,]

el_beta_min <- el_coordinate_descent(X.train, y.train, lambda1=lambda_min_el[1],
                                     lambda2=lambda_min_el[2],max_iter,tolerance)
y_hat_min_el = X.test %*% el_beta_min
mse_test_el = mean((y_hat_min_el-y.test)^2)

nonzero_el = sum(el_beta_min != 0)
nonzero_el_list2 = c(nonzero_el_list2, nonzero_el)
nonzero_lasso_list2 = c(nonzero_lasso_list2, nonzero_lasso)
mse_el_list2 = c(mse_el_list2, mse_test_el)
mse_lasso_list2 = c(mse_lasso_list2, mse_test_lasso)

if (mse_test_lasso > mse_test_el){
  outperf2 = outperf2+ 1
}
if(mse_test_lasso == mse_test_el){
  equalperf2 = equalperf2 + 1
}
}
```

The table below shows the results for a first comparison for scenario 1 and 2.

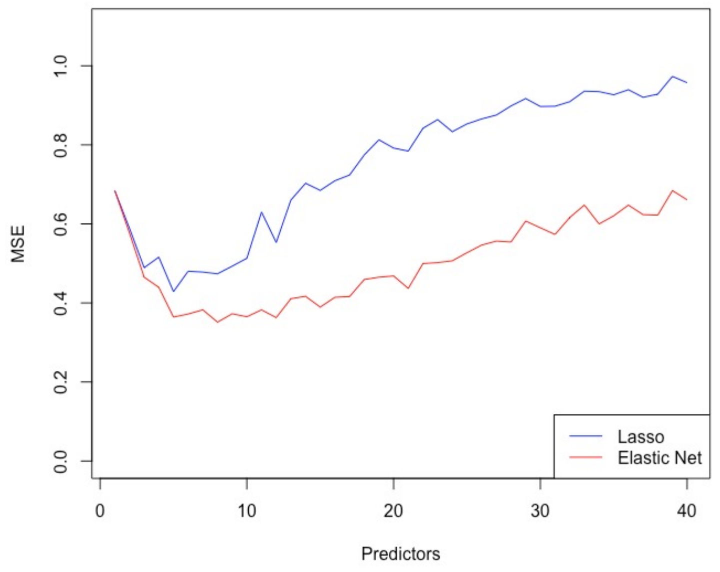
	Scenario 1	Scenario 2
Lasso MSE is higher (%)	64	42
Lasso MSE is lower (%)	18	8
MSEs are equal	18	50
Lasso non-zero coeff	3.36	2.5
EN non-zero coeff	6.7	2.92
Mean MSE LASSO	0.34	0.34
Mean MSE EN	0.29	0.32

As we can see from the table, Elastic Net is superior to LASSO in both scenarios terms of MSEs. However, in the first scenario, as far as variable selection is concerned, LASSO performs a better selection, closer to the number of non-zero coefficient of the true β, when there is a significant correlation structure between predictors. In the second scenario, LASSO selects only one of the two highly correlated parameters, whereas Elastic Net selects almost 3 predictors on average. These results meet our initial expectations.

2.2.3 Scenario 3

```r
mse_el_list_50 = c()
mse_lasso_list_50 = c()
for (p in 1:40){ # number of predictors varies from 1 to 40
nonzero_lasso_list_40 = c()
nonzero_el_list_40 = c()
mse_el_list = c()
mse_lasso_list = c()
for (sim in 1:50){ # making 50 simulations

n.train = 20
```

```r
n.validation = 20
n.test = 200
n = n.train + n.validation + n.test

X.train = matrix(rnorm(n.train*p),n.train,p)
X.train = as.matrix(X.train)
X.validation = matrix(rnorm(n.validation*p),n.validation,p)
X.validation = as.matrix(X.validation)
X.test = matrix(rnorm(n.test*p),n.test,p)
X.test = as.matrix(X.test)
X = rbind(X.train,X.validation,X.test)

B = runif(p,-4,4) #randomly set the components of beta
sigma = 3
epsilon = mvrnorm(n=1, Sigma=diag(n), mu=rep(0,n))

Y = X %*% B + sigma * epsilon
Y = as.vector(Y)
Y = scale(Y)

y.train = Y[1 : n.train]
y.validation = Y[(n.train+1) : (n.train+n.validation)]
y.test = Y[(n.train+n.validation+1) : n]

max_iter = 1000
tolerance = 1e-4
lambda_list = seq(0,2,0.01)
mse_list = c()
for (lambda in lambda_list){
  lasso_beta <- lasso_coordinate_descent(X.train, y.train, lambda=lambda,
                                         max_iter, tolerance)
  y_hat = X.validation %*% lasso_beta
  mse_list= c(mse_list, mean((y_hat-y.validation)^2))
}

lambda_min_lasso = lambda_list[which.min(mse_list)]

lasso_beta_min <- lasso_coordinate_descent(X.train, y.train, lambda=lambda_min_lasso,
                                           max_iter, tolerance)
y_hat_min_lasso = X.test %*% lasso_beta_min
mse_test_lasso = mean((y_hat_min_lasso-y.test)^2)

nonzero_lasso = sum(lasso_beta_min!=0)

mse_list_el = c()
lambda1_list = seq(0,2,0.01)
n1 = length(lambda1_list)
lambda2_list = seq(0,2,0.01)
n2 = length(lambda2_list)
a = 1
matrix = matrix(nrow = n1*n2, ncol = 3)
for (i in 1:n1){
  for (j in 1:n2){
    el_beta <- el_coordinate_descent(X.train, y.train, lambda1=lambda1_list[i],
                                      lambda2=lambda2_list[j], max_iter, tolerance)
    y_hat = X.validation %*% el_beta
    matrix[a,] = c(lambda1_list[i], lambda2_list[j], mean((y_hat-y.validation)^2))
    a = a + 1
   }
}

index = which.min(matrix[,3])
lambda_min_el = matrix[index,]

el_beta_min <- el_coordinate_descent(X.train, y.train, lambda1=lambda_min_el[1],
                                  lambda2=lambda_min_el[2], max_iter, tolerance)
y_hat_min_el = X.test %*% el_beta_min
mse_test_el = mean((y_hat_min_el-y.test)^2)

mse_el_list = c(mse_el_list, mse_test_el)
mse_lasso_list = c(mse_lasso_list, mse_test_lasso)
if (p == 40){
  nonzero_lasso = sum(lasso_beta_min!=0)
  nonzero_el = sum(el_beta_min!=0)
  nonzero_lasso_list_40 = c(nonzero_lasso_list_40, nonzero_lasso )
  nonzero_el_list_40 = c(nonzero_el_list_40, nonzero_el )
}
}
# Storing the value of the mean of the MSE for LASSO and EN for 50 simulations
# and for each value of p
mse_el_list_50 = c(mse_el_list_50, mean(mse_el_list))
mse_lasso_list_50 = c(mse_lasso_list_50, mean(mse_lasso_list))
}
less10_lasso = sum(nonzero_lasso_list_40 < 10)
less10_el = sum(nonzero_el_list_40 < 10)
more20_lasso = sum(nonzero_lasso_list_40 >= 20)
more20_el = sum(nonzero_el_list_40 >= 20)
# Plotting a lines chart to compare trends of MSE in LASSO and EN when p is increasing
plot(1:40,mse_lasso_list_50, type = 'l',ylim = c(0,1.3),col='blue',
     xlab = 'Predictors',ylab = 'MSE')

lines(1:40,mse_el_list_50, type = 'l',col = 'red')

legend("bottomright",legend = c('Lasso','Elastic Net'), lty = c(1,1),
       col = c('blue','red'))
```



From the graph we can notice that, as the number of predictors increases, on average, the elastic net always outperforms the LASSO. In addition, the differences between the MSEs computed by the two methods becomes larger.

	non-zero<10	$10 \leq$ non-zero<20	non-zero≥ 20
Lasso	16	24	0
EN	2	21	17

The number of non-zero coefficients estimated by LASSO and elastic net are reported in the table above. In particular they refer to the case $p = 40$ and they are classified in intervals. It can be observed how LASSO never estimates more than 20 (n) non-zero coefficients, whereas elastic net selects variable in a way that better reflects the true value of β.

2.3 Limitations and further research

Our project covers the use of simulated data in three distinct scenarios, offering a comprehensive evaluation of the performances of our algorithms under varying correlation structures and dimensionalities.

However, the project acknowledges a notable weakness related to the encountered issue during the execution of the algorithm, where beta coefficients tended to inflate, potentially leading to convergence problems. While the issue is recognised and addressed, the project does not delve into a comprehensive resolution, leaving room for further investigation into the root causes and potential remedies. Additionally, the reliance on simulated data, while valuable for controlled experimentation, limits the real-world applicability of our project. Integrating real datasets into the analysis could provide insights into the performance of the algorithms in more practical scenarios.

Further analysis could lead to explore additional scenarios or parameter variations to gain a more nuanced understanding of algorithm behaviour. Investigating the impact of varying sample sizes on algorithm performance would contribute insights into scalability. Finally, comparisons with alternative optimisation methods, such as proximal gradient descent or stochastic gradient descent, would enrich the evaluation.

2.4 Conclusion

The analysis conducted in this project supports the conclusion that elastic net generally outperforms LASSO, aligning with our initial expectations. However, additional insights emerged from the specific scenarios investigated. In Scenario 1, it was observed that elastic net exhibited a less favorable variable selection compared to LASSO. Despite this, the MSE consistently favored elastic net, showcasing its robust predictive accuracy in this context. Scenario 2 revealed a pattern where lasso consistently selected the first and third variables. This aligns with our expectations as there was a high correlation between the first and the second predictor. Instead, elastic net correctly performs as, on average, it is not doing any variable selection. The most compelling findings emerged in Scenario 3, where elastic net not only outperformed LASSO in all aspects but also demonstrated increasing dominance as the dimensionality (p) increased. This underscores the superiority of elastic net in high-dimensional settings.

In summary, the dynamic performance of elastic net, especially in high-dimensional scenarios, suggests its versatility and efficiency, making it a valuable choice for a broad range of applications.

3 Academic References

- Friedman, J., Hastie, T. 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*.
- Tseng, P. (2001). Convergence of block coordinate descent method for nondifferentiable minimization, *J. Opt. Theory Appl.* **109**: 474–494.

4 Dataset Source

- Iranian Churn Dataset. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5JW3Z.