

TDDE01 – Machine Learning

Individual Laboration Report 2

Martin Estgren <mares480>

15 november 2016

1 Assignment 1

In this assignment we were tasked with implementing *feature selection* using the *k-fold cross validation* and *linear regression* algorithms. The result of this can be found in appendix: A - Code assignment 1.

The feature selection algorithm iterates through all possible combinations of *features* from the predictor variables and apply the *k-fold cross validation* on all of them. The feature combination with the lowest *sum of squared error* gets picked as the best combination.

$$\mathbf{A} = \{c_1, c_2, \dots, c_n\}, 1 \leq n \leq ncol(X) \quad (1)$$

\mathbf{A} in the equation above represents all the permutations of column index for the response variable. Each of theses combinations are sent to the *k-fold cross validation* where the data is split into K parts of equal size. The K are then iterated through and each data subset is used as testing data with the other $K - 1$ sets as training data. The *k-fold cross validation* uses the *linear regression* with *ordinary least squares* estimator. The produced predictions are then error checked with the *sum of squared error*

$$\sum_{i=1}^n (\hat{y}_i - y_i), n = nrow(y) \quad (2)$$

for fold K . After all the K folds for a given combination of features are calculated, the mean of all fold errors are calculated and returned as the final error value for the feature combination.

$$\frac{1}{K} \sum_{k=1}^K \left(\sum_{i=1}^n (\hat{y}_i - y_i) \right) \quad (3)$$

where $K = |\text{folds}|$ and $n = |y|$.

In the figure below the mean error for a given number of selected features can be shown.

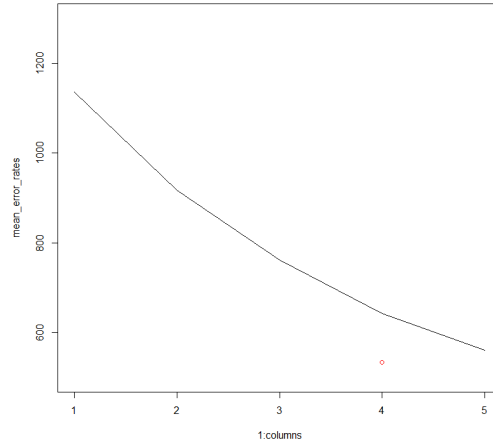
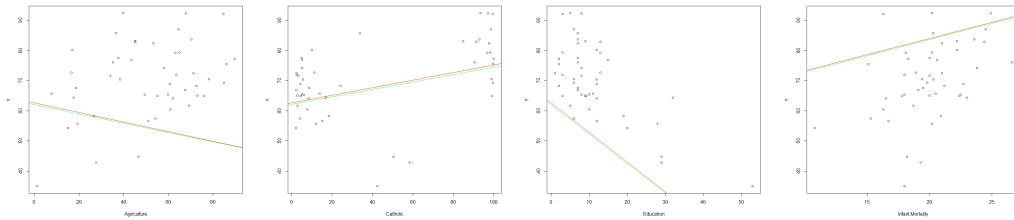


Figure 1: The mean error rate for different number of feature combinations.

The red dot indicates the best feature combination, i.e. the one with the lowest error score. For this data set it's $\{\text{Agriculture}, \text{Education}, \text{Catholic}, \text{Infant.Mortality}\}$. In general terms we can observe that the more features that are selected, the lower the error rate becomes.

The following graphs show the linear regression result for each of the different features that was returned by the best feature subset function described above.



The models aren't perfect but manages to hit fairly close to what humans would recognize as the best linear function for each data set.

2 Assignment 2

In this assignment we are tasked with analyzing the data set found in the file **tecator.xlsx**. The data is grouped into the columns: $\{Sample, Channel1, \dots, Channel100, Moisture, Protein\}$. The first task is to plot the *Moisture* and *Protein* and determine if a linear model would be a good approximation for the data. The plot can be seen in the figure below.

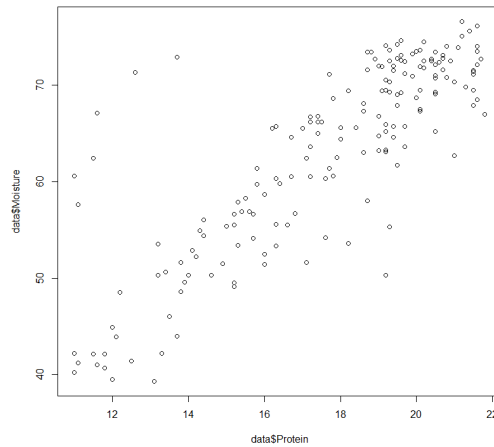


Figure 2: The *Moisture* plotted by *Protein*.

A linear approximation seems to be a good fit for the data.

Next we examine if a polynomial approximation of the data up to the n th power and using *mean squared error* (MSE) to examine the fitness of the polynomial model. The data set is split into two sets of equal size where one of the sets are the training set and the other the validation set.

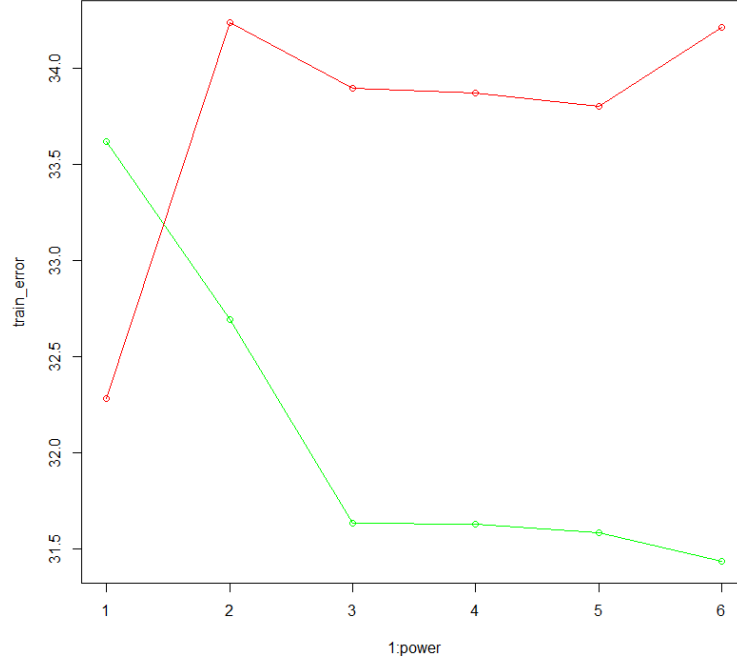


Figure 3: The training and validation error rate for the polynomial estimation given the MSE.

The **green line** is the error rate for the training data and the **red line** is the error rate for the validation set over the different power-levels. We can observe in the graph that the fitting of the training data decreases when the power-level increases but the error rate for the validation data increases with about the same magnitude. This can be explained with the fact that, as observed in the plot of the full data set above, the data would follow a linear model and the polynomial approximation leads to over fitting of the training set.

T.B.C The next part involves performing variable selection on the full data set with the *step* variable selection function. We get 64 coefficients from the step function.

The next part involves fitting a ridge regression model to the data set. The result can be observed in the following plot. We can see as the lambda increases, the coefficients converge to zero.

The lasso regression model results in fewer coefficients and less non continuous coefficient functions over the lambdas.

The final part involves plotting to cross validation score using the lasso model, the result can be observed in the graph below. The variance increase with the lambda and the validation score

In comparison with the step aic model the cross validation using lasso and k-fold resulted in much less coefficients.

3 Appendix: A - Code assignment 1

Listing 1: Code for assignment 1

```
library(readxl)

# folding indexes, returns start indexes for each
fold
indexes <- function(n,k){
  s = floor(n/k)
  indexes = matrix(1,k,2)
  for(i in 1:k){
    indexes[i,1] = (i-1) * s
    indexes[i,2] = i * s -1
  }
  indexes = indexes +1
  indexes[k,2] = n
  return(indexes)
}

# Deprecated because of reasons
binary_permutations <- function(n){
  indexes = matrix(0,2^n-1,n)
  for(i in 1:2^n-1){
    indexes[i,] = as.numeric(intToBits(i))[1:n]
  }
  return(indexes)
}

k_fold <- function(X,Y,K){
  n = nrow(X)
  fold_errors = matrix(0,K,1)
  fold_weights = matrix(0,K,(ncol(X)+1))
  indexes = indexes(n,K)
```

```

for( i in 1:K){
  test_fold_indexes = indexes[i,1]:indexes[i,2]
  train_fold_indexes = (1:n)[-test_fold_indexes]

  test_y = Y[test_fold_indexes]
  test_x = X[test_fold_indexes,]
  train_x = X[train_fold_indexes,]
  train_y = Y[train_fold_indexes]
  result = linear_regression(as.matrix(train_x),
    train_y,as.matrix(test_x),test_y)
  fold_errors[i,] = result$err
  fold_weights[i,] = result$param
}

return (list(weights=colMeans(fold_weights), err=
  fold_errors))
}
best_subset <- function(X,Y,K){
  n = nrow(X)
  columns = ncol(X)
  print(columns)

  errors = matrix(0,columns,2^columns)
  mean_error_rates = matrix(0,columns,1)
  best_error = Inf
  best_indexes = c()
  best_weights = c()
  for(n_features in 1:columns){
    binary_permutations = binary_permutations(ncol(X)
      ))
    binary_permutations = t(combn(1:columns,n_
      features))
    n_combinations = nrow(binary_permutations)

    combination_errors = matrix(Inf,n_combinations
      ,1)

    for(combination in 1:n_combinations){
      current_features = binary_permutations[
        combination,]
      filtered_x = as.matrix(X[current_features])

```

```

k_fold = k_fold(filtered_x,Y,K)
combination_errors[combination,] = mean(k_fold
    $err)
best_combination_index = which.min(combination
    _errors)

if(combination_errors[best_combination_index,]
    < best_error){
    best_weights = k_fold$weights
    best_indexes = binary_permutations[
        combination,]
    best_error = combination_errors[best_
        combination_index,]
}

}
mean_error_rates[n_features,] = mean(combination
    _errors)

}
plot(1:columns,mean_error_rates, type = "l", ylim
    = c(500,1300))
points(length(best_indexes),best_error, col="Red")
print(best_error)
return(list(indexes=best_indexes, weights=best_
    weights, err=best_error))
}

# Linear regression between two samples, one as x-
values and one as y-values
linear_regression <- function(x_train,y_train,x_
    test,y_test){
    x_train = cbind(matrix(1,nrow(x_train),1),x_
        train)
    x_test = cbind(matrix(1,nrow(x_test),1),x_test)
    w = solve(t(x_train) %*% x_train) %*% t(x_train)
        %*% y_train
    w = as.vector(w)
    pY = x_test %*% w
    errors = sum((y_test - pY)^2)
    return(list(param=w, pred=pY, err=errors))
}

```

```

}

data = swiss
set.seed(12345)
s = sample(1:nrow(data),nrow(data))
data = data[s,]
# =====
#fold_indexes(nrow(data),5)

X = data[,2:ncol(data)]
Y = as.matrix(data$Fertility)
best_features = best_subset(X,Y,5)
print(best_features)
X = X[,best_features$indexes]
print(X)
model = linear_regression(x_train = as.matrix(X),y_
  train = Y,x_test = as.matrix(X),y_test = Y)
for(i in 1:ncol(X)){
  plot(X[,i],Y, xlab = colnames(X)[i])
  # From best_features
  abline(best_features$weights[1],best_features$
    weights[i+1],col="Red")
  # From new regression
  abline(model$param[1],model$param[i+1],col="Green"
    )
}

```

4 Appendix: B - Code assignment 2

Listing 2: Code for assignment2

```

library(MASS)
library(readxl)
library(Matrix)
library(glmnet)

data = read_excel("tecator.xlsx")
plot(data$Protein,data$Moisture)
# answer: yes

```



```

# task 2
# Find a probailist model explaining  $M_i$ . where  $M$  is
  a polynomial model of the protein up to power  $i$ 
# Why is it important to use mean squared error whne
  fitting model?

# We approximate the model as  $w_0 + w_1x^1 + w_2x^2 \dots$ 
# Use MSE instead of SSE because MSE is normalized

set.seed(12345)
n = nrow(data)
train_indexes = sample(1:n,floor(n*0.5))
train_data = data[train_indexes,]
test_data = data[-train_indexes,]
poly = 6
train_error = matrix(0,poly,1)
test_error = matrix(0,poly,1)

for(i in 1:poly) {
  model = lm(Moisture ~ poly(Protein,i), data=train_
    data)
  train_predictions = predict(model,train_data)
  test_predictions = predict(model,test_data)

  train_error[i,] = mean((train_data$Moisture -
    train_predictions)^2)
  test_error[i,] = mean((test_data$Moisture - test_
    predictions)^2)
}

ylim = c(min(rbind(train_error,test_error)),max(
  rbind(train_error,test_error)))
plot(1:poly,train_error, col="Green", ylim=ylim)
lines(1:poly,train_error, col="Green")
points(1:poly,test_error,col="Red")
lines(1:poly,test_error, col="Red")

# Observation of the plot indicates that as we
  increase the polynomial level the functiuon

```

becomes more fitted for the training data and results in increasingly worse fit for the test data

```
model = lm(Fat ~ . - Protein - Moisture - Sample,
  data=data)
steps = stepAIC(model,direction="both", trace=FALSE)
coeff_aics = steps$coefficients
n_coeff_aics = length(coeff_aics)
print(n_coeff_aics)
# 64 columns were selected

data_y = data$Fat

data = data[,-which(colnames(data) == "Sample")]
data = data[,-which(colnames(data) == "Fat")]
data = data[,-which(colnames(data) == "Protein")]
data = as.matrix(data[,-which(colnames(data) == "
  Moisture")])

ridge = glmnet(x=data, y=data_y, alpha=0)
plot(ridge, xvar="lambda")

# task 6
# Do the same stuff but with LASSO, compare with
  ridge
lasso = glmnet(x=data, y=data_y, alpha=1)
plot(lasso, xvar="lambda")

# task 7
# CV to find the optimal LASSO, report the optimal
  lambda and how many variables were chosen by the
  model '
# make conculsions
# show a plot of CV scores in comparasion to
  lambda

lasso_cv = cv.glmnet(data,data_y, alpha=1)
lasso_cv_lambda = min(lasso_cv$lambda)
n_lass_cv = sum(coef(lasso_cv) != 0)
print(lasso_cv_lambda)
```

```
print(n_lass_cv)
plot(lasso_cv)
# task 8
# compare result from 4 and 7
```