Assignment - 1

NANDINI NAIR

2023-10-23

Part - A

QA1. What is the main purpose of regularization when training predictive models?

Answer:

An efficient technique to prevent the model from overfitting is regularisation. In order to prevent underfitting, it attempts to maximise the model's performance on the training set. However, when the model becomes complicated, a penalty is added to prevent overfitting. When a model becomes overly tuned to training data and attempts to understand it by closely capturing both signals and noise, this is known as overfitting. In this case, the model becomes overly detailed and becomes less generalizable. There are various kinds of regularisation strategies, such as:

1. L1 Regularization also known as Lasso Regularization adds the absolute value of the magnitude of the coefficient as a penalty term to the loss function.
2. L2 Regularization also known as Ridge Regularization adds the squared magnitude of the coefficient as the penalty term to the loss function.
3. Dropout Regularization drops some of the units in the network or the model during the training. (Mostly used in image classification, speech recognition and natural language processing tasks).

Most often, Lasso or L1 Regularisation is utilised for Variable Selection; this approach is indifferent to the presence of superfluous variables in the model that have no bearing on the goal function. Thus, variable removal in L1 Regularisation is demonstrated. The most popular technique for preventing the model from overfitting is called ridge or L2 regularisation; it lowers the coefficients or weights of the attributes without changing any variables. The majority of regression models pick variables using L1, and then regularise the model using L2.

Regularisation can help to enhance the model's generalisation performance on unseen data by lowering the model's complexity and preventing overfitting. This is significant since the ultimate objective of predictive modelling is to develop models capable of accurately predicting the behaviour of novel, unseen data.

QA2. What is the role of a loss function in a predictive model? And name two common loss functions for regression models and two common loss functions for classification models.

Answer:

In a predictive model, the loss function serves as a critical tool to assess the model's performance. It quantifies the gap between predicted values and the actual values, with the primary objective of minimizing this difference, often referred to as "error" or "cost. A loss function can hold additional components beyond the error on the training set.

Loss Functions – Regression Models: Mean Squared Error (MSE) – This loss function calculates the mean squared difference between the predicted and actualvalues of the target variable.

MSE = (1/n) Σ(yi - ŷi)^2n is the number of samples,yi is the actual value of the target variable for the ith sample,ŷi is the predicted value of the target variable for the ith sample.

Mean Absolute Error (MAE) – This loss function measures the average absolute difference between the predicted and actual values of the target variable.

MAE = (1/n) Σ|yi - ŷi| n is the number of samples,yi is the actual value of the target variable for the ith sample,ŷi is the predicted value of the target variable for the ith sample.

Classification Models:

Binary Cross-Entropy Loss – This loss function is used for binary classification problems where the target variable has two possible values. It measures the difference between the predicted probabilities and the actual binary labels.

BCE = - (1/n) Σ(yi * log(ŷi) + (1-yi) * log(1-ŷi))

n is the number of samples,yi is the actual binary label (0 or 1) for the ith sample,ŷi is the predicted probability of the positive class for the ith sample.

Categorical Cross-Entropy Loss –

This loss function is used for multi-class classification problems where the target variable has more than two possible values. It measures the difference between the predicted class probabilities and the actual class labels.

CCE = - (1/n) Σ(yij * log(ŷij))

n is the number of samples, yij is the actual probability of the jth class for the ith sample (1 if the sample belongs to the jth class and 0 otherwise), ŷij is the predicted probability of the jth class for the ith sample

QA3. Consider the following scenario. You are building a classification model with many hyperparameters on a relatively small data set. You will see that the training error is extremely small. Can you fully trust this model? Discuss the reason.

Answer:

We can't fully trust a model with an extremely low training error because it might have overfit the training data. Overfitting happens when the model becomes too focused on the

training data's noise and random fluctuations, making it perform poorly on new, unseen data. The goal of a predictive model is to perform well on unseen data, so generalization is crucial.

As models become more complex, they often show a decrease in training error. However, if a model has many hyperparameters and a small dataset, the risk of overfitting increases. This is why a low training error doesn't guarantee good performance on new data.

To avoid overfitting and ensure the model generalizes well, techniques like "cross-validation" and "regularization" can be helpful.

QA4. What is the role of the lambda parameter in regularized linear models such as Lasso or Ridge regression models?

Answer:

The lambda parameter, often referred to as the regularization parameter, holds significant importance in regularized linear models like Lasso or Ridge Regression.

In these models, the objective is to minimize a loss function by adding a penalty term to it. Lambda governs the strength of this penalty.

A higher lambda value imposes a more potent penalty, resulting in a simpler model. However, this simplicity may lead to underfitting, where the model fails to capture the underlying data patterns. Conversely, a lower lambda value imposes a weaker penalty, creating a more complex model that can eventually overfit, focusing excessively on the training data and performing poorly on unseen data.

For L1 regularization (Lasso), increasing lambda tends to increase model sparsity, driving some coefficients to zero. This is particularly useful for variable selection and dimensionality reduction.

In L2 regularization (Ridge), increasing lambda reduces the magnitude of coefficients, helping to prevent overfitting. It doesn't eliminate input attributes but rather shrinks their coefficients.

Typically, in both Lasso and Ridge regression, lambda is chosen through cross-validation. Different lambda values are tested, and the one that yields the best performance on a validation set is selected. The optimal lambda value is data-specific and depends on the model's complexity.

In essence, lambda is a critical parameter for balancing model complexity and generalization, and its choice is determined by the specific dataset and modeling needs.

Part B :

```
knitr::opts_chunk$set(message = FALSE)
knitr::opts_chunk$set(warning = FALSE)

library("class")
library("caret")
library("ISLR")
library("ggplot2")
library("esquisse")
library("tinytex")
library("tidyverse")
library("dplyr")
library("glmnet")
library("glmnetUtils")
library("corrplot")
library("moments")
```

Loading the Data Set

```
data <- Carseats

new_data <- data %>%
select("Sales","Price","Advertising","Population","Age","Income","Education")

colMeans(is.na(new_data))

##       Sales       Price Advertising  Population         Age      Income
##           0           0           0           0           0           0
##   Education
##           0
```

1. **Build a Lasso regression model to predict Sales based on all other attributes ("Price", "Advertising", "Population", "Age", "Income" and "Education"). What is the best value of lambda for such a lasso model?**

```
set.seed(456)
Norm_Train <- preProcess(new_data[,-1], method = c("scale","center"))
Normalized_Train <- predict(Norm_Train, new_data)
#Normalising the data
train_x <- as.matrix(Normalized_Train[2:7])
train_y <- Normalized_Train[[1]]

set.seed(789)
cvfit = cv.glmnet(train_x,train_y,data=Normalized_Train,nfolds=5,alpha=1)
cvfit

##
## Call:  glmnet::cv.glmnet(x = train_x, y = train_y, data = Normalized_Train
```

```
,      nfolds = 5, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00431    62   5.181 0.3295       6
## 1se 0.28326    17   5.493 0.2863       4
```

#Best Lamda Value

```
cvfit$lambda.min
```

```
## [1] 0.004305309
```

As we can see 0.004305309 can be considered as the best lambda value for the lasso model which wehave built.

- Lamda1se*

```
cvfit$lambda.1se
```

```
## [1] 0.2832606
```

There's also another lambda value which we can use if it is accepted to have a bigger lambda value which will result in a more regularized model. This ensures that the cross validation error is still only up to one standard deviation bigger than the optimal value. For this model built the lambda value is 0.31087.

Looking at the coefficients that was eliminated

```
coef(cvfit, s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.35383399
## Advertising  0.82805813
## Population  -0.13061347
## Age         -0.78854992
## Income       0.28931898
## Education   -0.09102484
```

When the lambda is set to lambda.min i.e. 0.004305309 we can see that none of the attributes has been eliminated from the model.

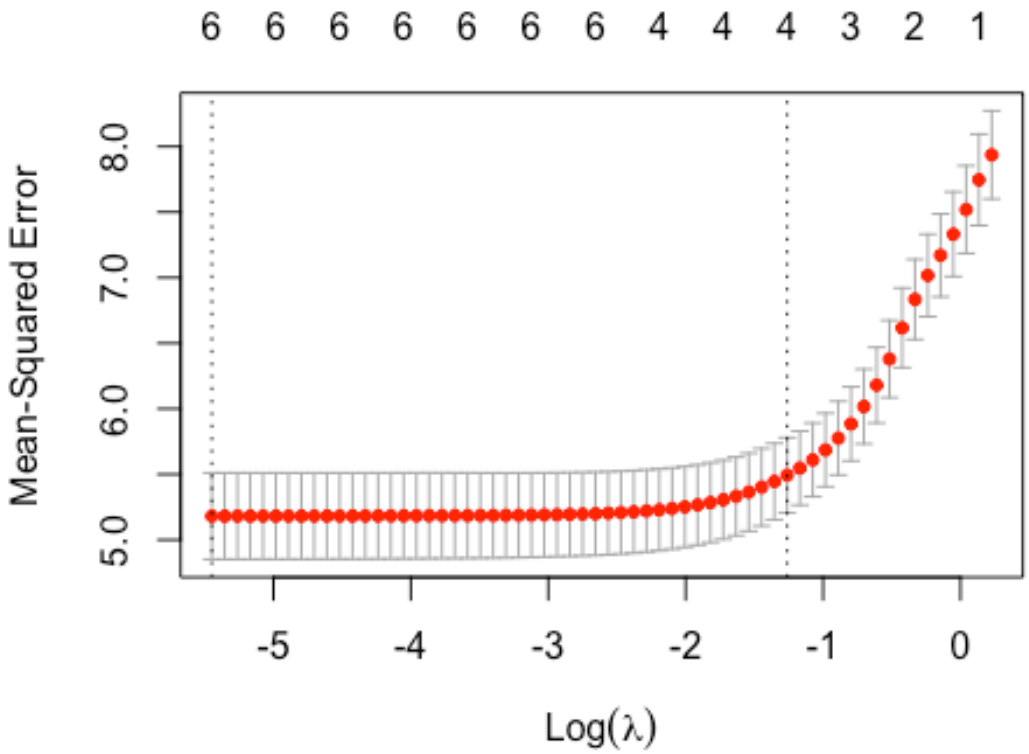
```
coef(cvfit, s="lambda.1se")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.04155049
## Advertising  0.51856113
## Population   .
```

```
## Age          -0.47486663
## Income        0.05364767
## Education     .
```

When the lambda is set to lambda.1se we can see that two of the attributes have been eliminated from the model i.e. “Population” and “Education” have been eliminated. **Plotting the Model**

```
plot(cvfit)
```



The Optimal (Min) and 1SE lambda values with respect to the above graph

```
lambda_min <- log(0.004305309)
lambda_min

## [1] -5.447906

lambda_1se <- log(0.3108781)
lambda_1se

## [1] -1.168354
```

2. What is the coefficient for the price (normalized) attribute in the best model (i.e. model with the optimal lambda)?

```
coef(cvfit, s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.35383399
## Advertising  0.82805813
## Population  -0.13061347
## Age         -0.78854992
## Income       0.28931898
## Education   -0.09102484
```

#The coefficient for the "Price Attribute" when the lambda is at minimum i.e. Optimal Lambdais -1.33844979.

3. **How many attributes remain in the model if lambda is set to 0.01? How that number changes if lambda is increased to 0.1? Do you expect more variables to stay in the model (i.e., to have non-zero coefficients) as we increase lambda?**

```
#lambda = 0.01
coef(cvfit,s=0.01)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.34732839
## Advertising  0.82025640
## Population  -0.12187511
## Age         -0.78190585
## Income       0.28488681
## Education   -0.08502705
```

When lambda (s) = 0.01 we have all the input attributes.

```
#lambda = 0.1
coef(cvfit,s=0.1)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  7.4963250
## Price       -1.2447750
## Advertising  0.7007231
## Population   .
## Age         -0.6775428
```

```
## Income       0.2139222
## Education    .
```

Whereas when we increased the lambda value from 0.01 to 0.1 we see that the "Education" and "Population" attribute has been removed by the model.

In general when we increase the value of the lambda we expect the number of attributes to reduce, the lambda value controls the strength of the L1 penalty term, which shrinks the coefficients towards zero. When lambda is very small, the L1 penalty has a negligible effectand the regression model behaves like ordinary least squares (OLS) regression. In this case,the model may include all the available attributes, resulting in overfitting (s = 0.01)

As lambda increases, the L1 penalty term becomes more dominant, causing some coefficients to shrink towards zero and some to become exactly zero. The coefficients of the attributes that become exactly zero are eliminated from the model, resulting in a reduction in the number ofattributes (s = 0.1).

Also, increasing the lambda will also result in reducing the coefficients of the attributes:

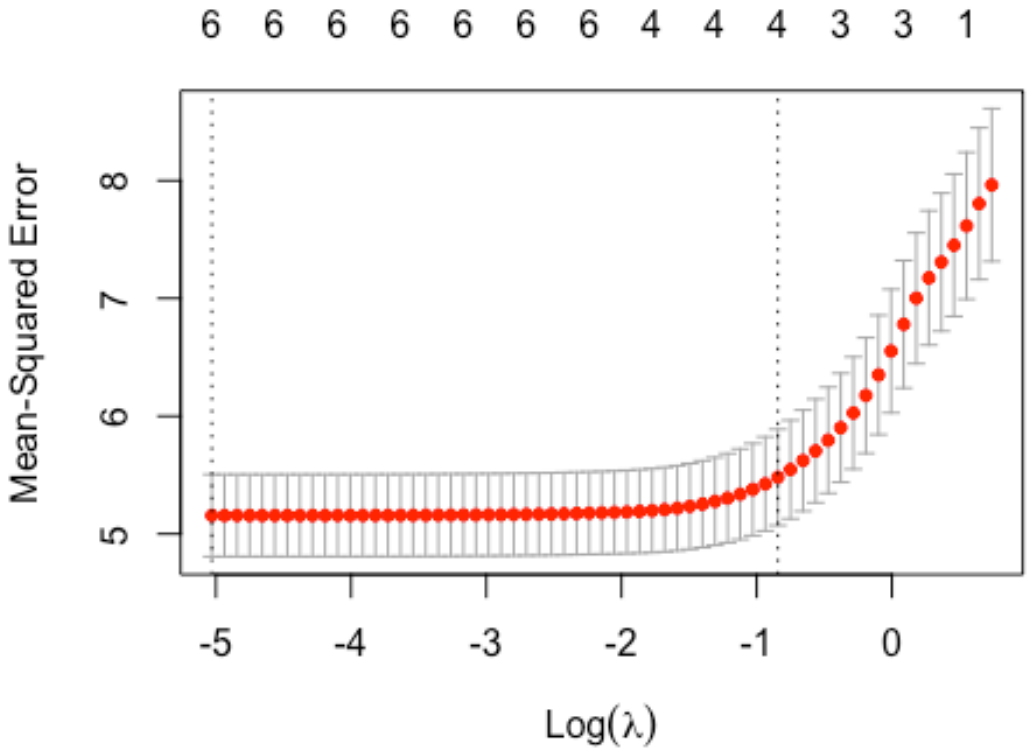
i.e. Price with 0.01 lambda = -1.34888753 whereas Price with 0.1 lambda = -1.2468705

4. **Build an elastic-net model with alpha set to 0.6. What is the best value of lambda for such a model?**

```
cvfit1 <-
cv.glmnet(train_x,train_y,data=Normalized_Train,nfolds=10,alpha=0.6)
cvfit1

##
## Call:  glmnet::cv.glmnet(x = train_x, y = train_y, data = Normalized_Train
,      nfolds = 10, alpha = 0.6)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure     SE Nonzero
## min 0.0065    63   5.156 0.3486       6
## 1se 0.4302    18   5.481 0.4083       4

plot(cvfit1)
```



Looking at the attributes that were eliminated when the model was at lambda.min and alpha = 0.6

```
coef(cvfit1,s="lambda.min")
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  7.49632500
## Price       -1.35288810
## Advertising  0.82765532
## Population  -0.13077204
## Age         -0.78811113
## Income       0.28949277
## Education   -0.09135756
```

The cvfit1 model at lambda.min resulted in eliminating none of the input attributes.

```
cvfit1$lambda.min
```

```
## [1] 0.006538062
```

The best value of lambda when the alpha = 0.6 is 0.0884631