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

**Regularization:**

The objective of any Machine learning model is to learn the patterns from the training data and generalize it to effectively predict the unseen data. So, Generalization is a term used to describe a model’s ability to react to new data. There are many causes for the model’s lack of Generalization Overfitting of the model is one among them. Overfitting happens when a model gets too complicated that it learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. So, the variance of the estimator is increased.

Regularization is used as a strategy to increase the model’s generalization capacity by reducing the complexity of the model. Regularization attempts to improve a model’s performance by simplifying it. To simplify the model and reduce the complexity, Regularization penalizes the model. Regularization is used to calibrate machine learning models to minimize the adjusted loss function and prevent overfitting or underfitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

Examples of Regularization techniques:

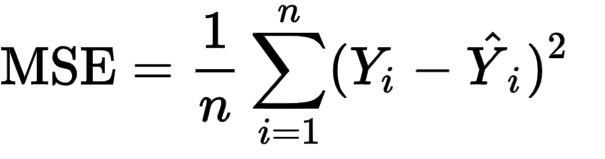
1. Lasso & Ridge Regression are used for regularization of Regression models.
2. Dropout is one of the regularization techniques used for regularization of a Deep Learning models.

**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.

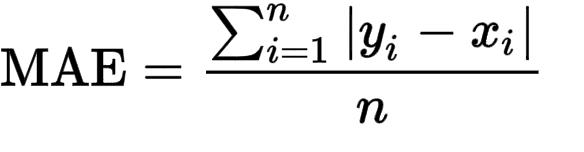
[Loss functions](https://en.wikipedia.org/wiki/Loss_function) measure how far an estimated value given by the predictive model is from its true value. They define an objective which the performance of the model is evaluated against, and the parameters learned by the model are determined by minimizing a chosen loss function. So, the loss function is the function that computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how your algorithm models the data. The process of reducing the loss continues by changing the parameters of the model until the lowest possible loss is achieved. Below are some of the examples of Loss functions and how they work.

**Common loss functions for regression models:**

**Mean Squared Error (MSE):** Mean Squared Error is the average of the squared differences between the actual and the predicted values. For a data point Yi and its predicted value Ŷi, where n is the total number of data points in the dataset, the mean squared error is defined



**Mean Absolute Error (MAE):** [Mean Absolute Error (also called L1 loss)](https://en.wikipedia.org/wiki/Mean_absolute_error) is one of the most simple yet robust loss functions used for regression models. Regression problems may have variables that are not strictly [Gaussian in nature](https://www.statisticssolutions.com/assumptions-of-linear-regression/) due to the presence of outliers (values that are very different from the rest of the data). Mean Absolute Error would be an ideal option in such cases because it does not take into account the direction of the outliers (unrealistically high positive or negative values). As the name suggests, MAE takes the average sum of the absolute differences between the actual and the predicted values. For a data point xi and its predicted value yi, n being the total number of data points in the dataset, the mean absolute error is defined as:

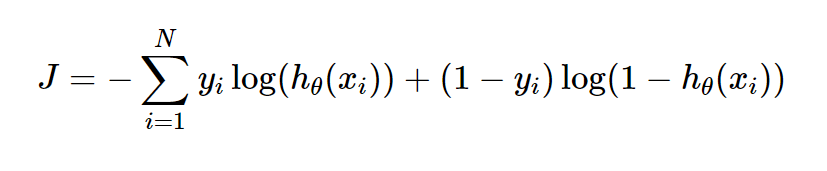


**Common loss functions for classification models:**

#### Binary Cross Entropy:

This is the most common loss function used for classification problems that have two classes. The word “entropy”, seemingly out-of-place, has a statistical interpretation. [Entropy](https://en.wikipedia.org/wiki/Entropy_(information_theory)) is the measure of randomness in the information being processed, and cross entropy is a measure of the difference of the randomness between two random variables.

If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases. Going by this, predicting a probability of .011 when the actual observation label is 1 would result in a high loss value. In an ideal situation, a “perfect” model would have a [log loss](https://www.kaggle.com/dansbecker/what-is-log-loss) of 0.



Where yi is the true label and hθ(xi) is the predicted value post hypothesis.

Since binary classification means the classes take either 0 or 1, if yi = 0, that term ceases to exist and if yi = 1, the (1-yi) term becomes 0.

#### Categorical Cross Entropy Loss:

Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when categorical cross entropy loss function is used is that the labels should be [one-hot encoded](https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179). This way, only one element will be non-zero as other elements in the vector would be multiplied by zero. This property is extended to an activation function called softmax.

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

When a prediction model is built using a training set, it should be evaluated both on training set and a validation set. The model can be said as a good prediction model only if both the train error and validation error (Error on the data unseen by the model) are low.

Generally, there are two reasons why a prediction model is said to be a dumb model as below.

When the model is underfitted, which means that the model is not able to capture the relationship between the input and output variables accurately. So, the performance of such model is worse both on training set and validation set. So, we need to increase the complexity of the model so that it captures the relationship between the inputs and the output variables accurately.

When the model is overfitted, which means that the model performs very well on the training set with very high train accuracy whereas when the same model is applied on the validation set, the error is very high and high variance is found between the train error and validation error. This means that the model has become too complicated such that it learns the train data including all the noise and outliers and become sensitive even to a smallest change in the input variables. So, the objective of any machine learning model, Generalization is not achieved which makes the model perform very poor on the unseen data. This overfitting happens particularly when the train data is relatively small, and the complexity of the model is very high.

In the given example, the model with many hyperparameters is built using a relatively small dataset.

So, there is a high chance for the model to be overfitted. As explained above, the model is said to be a good prediction model only when it performs well both on the train data and the unseen data. Since, the given model has a very high chance of overfitting and high variance, it performs very well on the training set, but is likely to perform poorly on the unseen data due to overfitting.

So, we cannot trust the model only if it makes good prediction on the train data. Instead, it should be applied to the unseen data and check if variance is found between the train error and validation error. If there is high variance, the model is to be retrained by reducing the complexity of the model or use some regularization techniques for the model to generalize rather than just learning the patterns in the train dataset.

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

Lambda is a hyperparameter used in linear models such as Lasso or Ridge Regression for Regularization i.e., to reduce the complexity of the model.

How Lambda works to regularize the model in Lasso & Ridge Regression:

Linear regression models try to reduce the loss function (Ordinary Least squares) to arrive at the optimum weights given for each attribute. But, when the model is overfitted and cannot generalize well, we use regularization to penalize the model. This is done by adding an extra term for the loss function, the value of which is directly proportional to the Lambda value. So, when we tune this Lambda value to increase it, the total loss value is increased, and the model tries to shrink the coefficients of the parameters to arrive at the optimal solution. The idea is that by shrinking or regularizing the coefficients, prediction accuracy can be improved, variance can be decreased, and model interpretability can also be improved.

In ridge regression, we add a penalty by way of a tuning parameter lambda which is chosen using cross validation. The idea is to make the fit small by making the residual sum or squares small plus adding a shrinkage penalty. The shrinkage penalty is lambda times the sum of squares of the coefficients so coefficients that get too large are penalized. As lambda gets larger, the bias is unchanged, but the variance drops. The drawback of ridge is that it doesn’t select variables. It includes all the variables in the final model.

The cost function for ridge regression is:

Text, letter

Description automatically generated

In lasso, the penalty is the sum of the absolute values of the coefficients. Lasso shrinks the coefficient estimates towards zero and it has the effect of setting variables exactly equal to zero when lambda is large enough while ridge does not. Hence, much like the best subset selection method, lasso performs variable selection. The tuning parameter lambda is chosen by cross validation. When lambda is small, the result is essentially the least squares estimates. As lambda increases, shrinkage occurs so that variables that are at zero can be thrown away. So, a major advantage of lasso is that it is a combination of both shrinkage and selection of variables.

The cost function lasso regression is given below:

A picture containing text

Description automatically generated

Advanced\_Data\_Mining

Ram

3/11/2022

Loading the required libraries and the data

library(ISLR)

## Warning: package 'ISLR' was built under R version 4.1.2

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.1.2

## Loading required package: Matrix

## Loaded glmnet 4.1-3

attach(Carseats)  
summary(Carseats)

## Sales CompPrice Income Advertising   
## Min. : 0.000 Min. : 77 Min. : 21.00 Min. : 0.000   
## 1st Qu.: 5.390 1st Qu.:115 1st Qu.: 42.75 1st Qu.: 0.000   
## Median : 7.490 Median :125 Median : 69.00 Median : 5.000   
## Mean : 7.496 Mean :125 Mean : 68.66 Mean : 6.635   
## 3rd Qu.: 9.320 3rd Qu.:135 3rd Qu.: 91.00 3rd Qu.:12.000   
## Max. :16.270 Max. :175 Max. :120.00 Max. :29.000   
## Population Price ShelveLoc Age Education   
## Min. : 10.0 Min. : 24.0 Bad : 96 Min. :25.00 Min. :10.0   
## 1st Qu.:139.0 1st Qu.:100.0 Good : 85 1st Qu.:39.75 1st Qu.:12.0   
## Median :272.0 Median :117.0 Medium:219 Median :54.50 Median :14.0   
## Mean :264.8 Mean :115.8 Mean :53.32 Mean :13.9   
## 3rd Qu.:398.5 3rd Qu.:131.0 3rd Qu.:66.00 3rd Qu.:16.0   
## Max. :509.0 Max. :191.0 Max. :80.00 Max. :18.0   
## Urban US   
## No :118 No :142   
## Yes:282 Yes:258   
##   
##   
##   
##

# Filtering the required attributes, Scaling and conversion of data into matrix  
  
Carseats\_Filtered <- Carseats %>% select("Price",   
"Advertising","Population","Age","Income","Education") %>% scale(center = TRUE, scale = TRUE)  
x <- as.matrix(Carseats\_Filtered)  
y <- Carseats[,1]

# Let's apply simple linear regression on the data to evaluate the coefficients & R-squared value(Represents the percentage of the variance in the dependent variable that the independent variables explain collectively)  
  
z =lm(y~x)  
summary(z)

##   
## Call:  
## lm(formula = y ~ x)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -6.1113 -1.5385 -0.1214 1.4339 6.5244   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 7.49632 0.11258 66.584 < 2e-16 \*\*\*  
## xPrice -1.35877 0.11369 -11.951 < 2e-16 \*\*\*  
## xAdvertising 0.83400 0.11734 7.108 5.60e-12 \*\*\*  
## xPopulation -0.13723 0.11774 -1.166 0.2445   
## xAge -0.79358 0.11345 -6.995 1.15e-11 \*\*\*  
## xIncome 0.29267 0.11335 2.582 0.0102 \*   
## xEducation -0.09556 0.11356 -0.841 0.4006   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 2.252 on 393 degrees of freedom  
## Multiple R-squared: 0.3739, Adjusted R-squared: 0.3643   
## F-statistic: 39.11 on 6 and 393 DF, p-value: < 2.2e-16

mean(z$residuals^2)

## [1] 4.981366

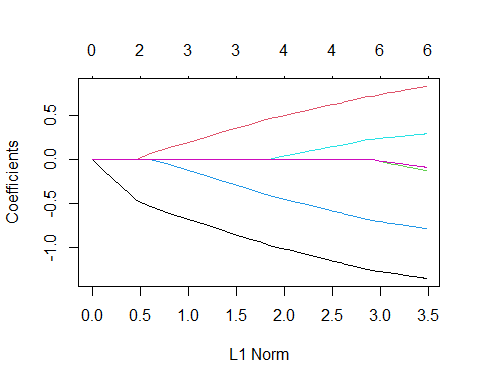
So, from the above results, we can see that only 36.43% of variance in the dependent variable (Sales) can be explained by the given attributes.

Now, let’s apply Lasso Regression on the same data and check whether we will be able to improve the R-Squared value.

fit <- glmnet(x, y )  
summary(fit)

## Length Class Mode   
## a0 62 -none- numeric  
## beta 372 dgCMatrix S4   
## df 62 -none- numeric  
## dim 2 -none- numeric  
## lambda 62 -none- numeric  
## dev.ratio 62 -none- numeric  
## nulldev 1 -none- numeric  
## npasses 1 -none- numeric  
## jerr 1 -none- numeric  
## offset 1 -none- logical  
## call 3 -none- call   
## nobs 1 -none- numeric

plot(fit)

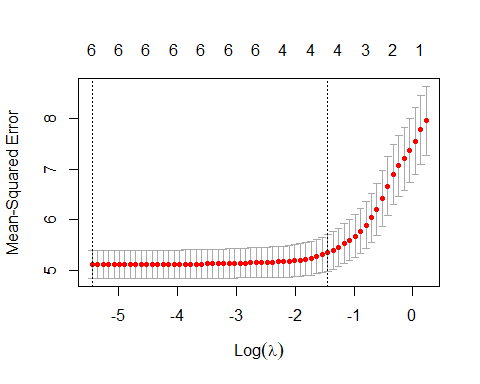


print(fit)

##   
## Call: glmnet(x = x, y = y)   
##   
## Df %Dev Lambda  
## 1 0 0.00 1.25500  
## 2 1 3.36 1.14400  
## 3 1 6.15 1.04200  
## 4 1 8.47 0.94940  
## 5 1 10.39 0.86500  
## 6 1 11.99 0.78820  
## 7 2 14.62 0.71820  
## 8 3 18.08 0.65440  
## 9 3 21.12 0.59620  
## 10 3 23.64 0.54330  
## 11 3 25.73 0.49500  
## 12 3 27.46 0.45100  
## 13 3 28.91 0.41100  
## 14 3 30.10 0.37450  
## 15 4 31.12 0.34120  
## 16 4 32.13 0.31090  
## 17 4 32.97 0.28330  
## 18 4 33.67 0.25810  
## 19 4 34.25 0.23520  
## 20 4 34.73 0.21430  
## 21 4 35.13 0.19520  
## 22 4 35.46 0.17790  
## 23 4 35.74 0.16210  
## 24 4 35.97 0.14770  
## 25 4 36.16 0.13460  
## 26 4 36.31 0.12260  
## 27 4 36.45 0.11170  
## 28 4 36.55 0.10180  
## 29 4 36.64 0.09276  
## 30 6 36.75 0.08451  
## 31 6 36.86 0.07701  
## 32 6 36.95 0.07017  
## 33 6 37.02 0.06393  
## 34 6 37.09 0.05825  
## 35 6 37.14 0.05308  
## 36 6 37.18 0.04836  
## 37 6 37.21 0.04407  
## 38 6 37.24 0.04015  
## 39 6 37.27 0.03658  
## 40 6 37.29 0.03333  
## 41 6 37.30 0.03037  
## 42 6 37.32 0.02767  
## 43 6 37.33 0.02522  
## 44 6 37.34 0.02298  
## 45 6 37.35 0.02094  
## 46 6 37.35 0.01908  
## 47 6 37.36 0.01738  
## 48 6 37.36 0.01584  
## 49 6 37.37 0.01443  
## 50 6 37.37 0.01315  
## 51 6 37.37 0.01198  
## 52 6 37.38 0.01092  
## 53 6 37.38 0.00995  
## 54 6 37.38 0.00906  
## 55 6 37.38 0.00826  
## 56 6 37.38 0.00752  
## 57 6 37.38 0.00686  
## 58 6 37.38 0.00625  
## 59 6 37.38 0.00569  
## 60 6 37.38 0.00519  
## 61 6 37.38 0.00472  
## 62 6 37.38 0.00430

So, in the above results we can see that 37.38% (An increase of 1%) of variance in the dependent variable (Sales) can be explained by the given attributes by applying regularization and the best Lambda is 0.00430

# Finding the minimum Lambda value  
cvfit = cv.glmnet(x, y)  
plot(cvfit)



cvfit$lambda.min

## [1] 0.004305309

cvfit$lambda.1se

## [1] 0.2351679

# Let's see the coefficients of the final equation for different values of Lambada   
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

coef(fit,s = 0)

## 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

coef(fit,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

coef(fit,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 .

coef(fit,s = 0.2)

## 7 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 7.4963250  
## Price -1.1338813  
## Advertising 0.6013226  
## Population .   
## Age -0.5669483  
## Income 0.1264650  
## Education .

coef(fit,s = 0.4)

## 7 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 7.4963250  
## Price -0.9091614  
## Advertising 0.3995350  
## Population .   
## Age -0.3452469  
## Income .   
## Education .

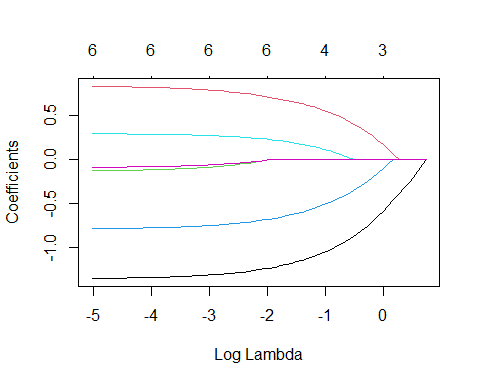
So,the coefficient of “Price” attribute is “-1.35” for the optimal lambada value.

we can see that the coefficients are slightly shrunk when regularization is applied and further shrinkage of coefficients can be observed when Lambda is increased and 2 variables are eliminated when lambada in increased to 0.1, 3 variables are eliminated when Lambda is increased to 0.2 and 5 variables are eliminated when Lambda is increased to 0.4.

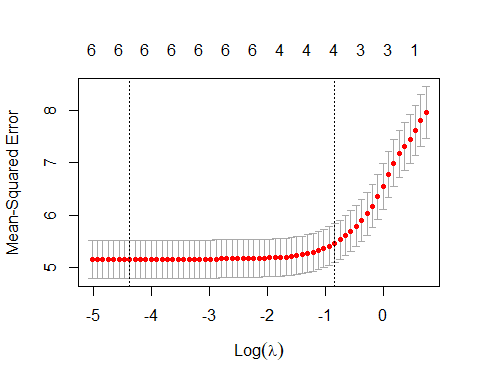
So, we can infer that the number value of coefficients of the attribute keeps decreasing and finally eliminated after certain value of Lambda is reached.

Building an elastic-net model with alpha set to 0.6.

fit.elnet = glmnet(x,y, alpha = 0.6)  
plot(fit.elnet, xvar = "lambda")



plot(cv.glmnet(x,y, alpha = 0.6))



summary(fit.elnet)

## Length Class Mode   
## a0 63 -none- numeric  
## beta 378 dgCMatrix S4   
## df 63 -none- numeric  
## dim 2 -none- numeric  
## lambda 63 -none- numeric  
## dev.ratio 63 -none- numeric  
## nulldev 1 -none- numeric  
## npasses 1 -none- numeric  
## jerr 1 -none- numeric  
## offset 1 -none- logical  
## call 4 -none- call   
## nobs 1 -none- numeric

print(fit.elnet)

##   
## Call: glmnet(x = x, y = y, alpha = 0.6)   
##   
## Df %Dev Lambda  
## 1 0 0.00 2.09200  
## 2 1 2.67 1.90600  
## 3 1 5.03 1.73700  
## 4 1 7.09 1.58200  
## 5 1 8.90 1.44200  
## 6 1 10.47 1.31400  
## 7 2 12.89 1.19700  
## 8 3 16.00 1.09100  
## 9 3 18.95 0.99370  
## 10 3 21.49 0.90540  
## 11 3 23.67 0.82500  
## 12 3 25.55 0.75170  
## 13 3 27.15 0.68490  
## 14 3 28.52 0.62410  
## 15 4 29.75 0.56860  
## 16 4 30.91 0.51810  
## 17 4 31.89 0.47210  
## 18 4 32.72 0.43020  
## 19 4 33.43 0.39190  
## 20 4 34.02 0.35710  
## 21 4 34.52 0.32540  
## 22 4 34.93 0.29650  
## 23 4 35.29 0.27020  
## 24 4 35.58 0.24620  
## 25 4 35.83 0.22430  
## 26 4 36.04 0.20440  
## 27 4 36.21 0.18620  
## 28 4 36.36 0.16970  
## 29 4 36.48 0.15460  
## 30 6 36.60 0.14090  
## 31 6 36.73 0.12830  
## 32 6 36.84 0.11690  
## 33 6 36.93 0.10660  
## 34 6 37.01 0.09709  
## 35 6 37.07 0.08846  
## 36 6 37.12 0.08060  
## 37 6 37.17 0.07344  
## 38 6 37.20 0.06692  
## 39 6 37.23 0.06097  
## 40 6 37.26 0.05556  
## 41 6 37.28 0.05062  
## 42 6 37.30 0.04612  
## 43 6 37.31 0.04203  
## 44 6 37.33 0.03829  
## 45 6 37.34 0.03489  
## 46 6 37.34 0.03179  
## 47 6 37.35 0.02897  
## 48 6 37.36 0.02639  
## 49 6 37.36 0.02405  
## 50 6 37.37 0.02191  
## 51 6 37.37 0.01997  
## 52 6 37.37 0.01819  
## 53 6 37.37 0.01658  
## 54 6 37.38 0.01510  
## 55 6 37.38 0.01376  
## 56 6 37.38 0.01254  
## 57 6 37.38 0.01143  
## 58 6 37.38 0.01041  
## 59 6 37.38 0.00949  
## 60 6 37.38 0.00864  
## 61 6 37.38 0.00788  
## 62 6 37.38 0.00718  
## 63 6 37.38 0.00654

So, in the above results we can see that 37.38% of variance in the dependent variable (Sales) can be explained by the given attributes by applying regularization with alpha set to 0.6 and the best Lambda is 0.00654