ADM - ASSIGNMENT -01

2025-03-09

Name – Anurodh Singh

Email- [asingh68@kent.edu](mailto:asingh68@kent.edu)

KSUID – 811288773

**PART – A**

**Q1. What is the main purpose of regularization when training predictive models?**

Ans. When training predictive models, regularization is primarily used to increase the model's capacity to generalize to previously unseen data. We frequently encounter the problem of the model performing well on training data but poorly on testing, novel, or unknown data. Overfitting, in which the model grows excessively complex and learns not just the actual patterns but also the noise and specificities of the training set, is frequently the cause of this disparity. It so finds it difficult to predict fresh data with any degree of accuracy.

We apply a penalty to the feature coefficients in order to regularize the model and solve this problem. By keeping the model from fitting the noise in the training data or depending too much on any one feature, these penalties aid in the reduction of model complexity. Through this procedure, the model is encouraged to concentrate on the most significant, pertinent patterns and eliminate extraneous features that would not hold true in unseen, real-world situations.

By applying these penalties to the feature weights, regularization techniques like Lasso (L1 regularization) and Ridge (L2 regularization) produce simpler models. Lasso, for instance, has the ability to exclude some feature coefficients from the model by shrinking them to zero. Ridge, however, retains all of the model's properties while penalizing big coefficients and bringing the features coefficients close to zero but not zero.

Regularization aims to simplify the model and increase its robustness by ensuring that it learns the underlying patterns rather than memorizing the training data. By preventing the model from being too complicated or overfitted, this increased robustness leads to higher performance on fresh data.

To sum up, regularization aids in finding a compromise between maintaining the model's simplicity so that it can generalize to new data and fitting the training set well. Predictions on fresh, actual data are more accurate as a result of this equilibrium.

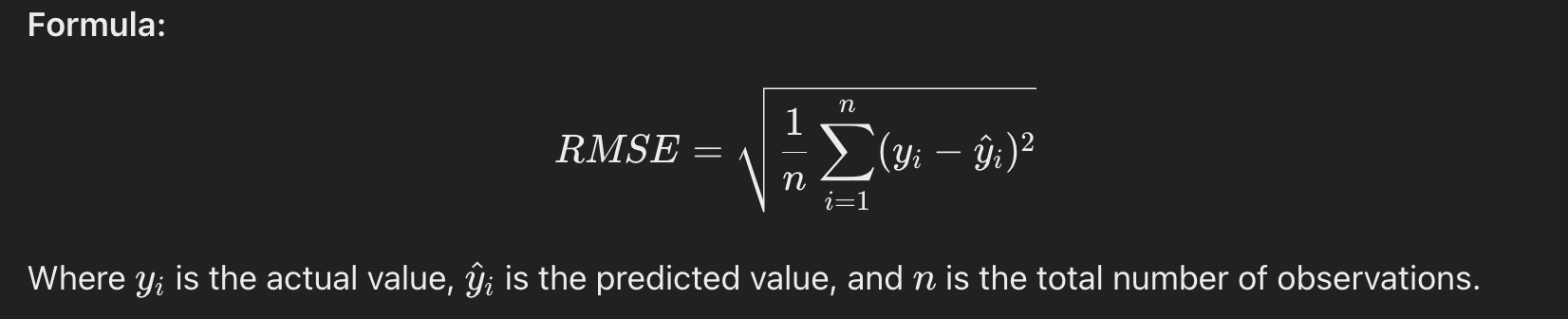
**Q2. What is the role of a loss function in a predictive model? And name two common loss functions for regression models and two common loss functions for classification models**.

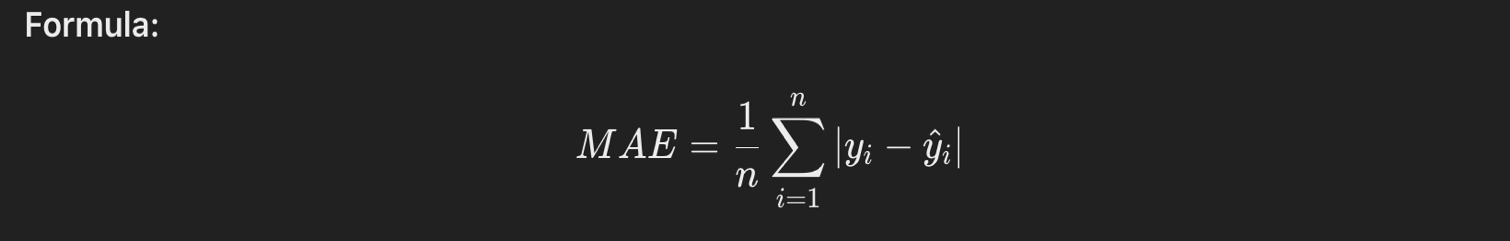
Ans. A loss function's goal is to quantify the discrepancy between the ground truth, or actual values, and the values that a model predicts. To put it simply, it measures how well or poorly the model is working. In a house price prediction model, for instance, if we attempt to forecast a home's price based on specific characteristics (such as size, location, or number of rooms), the loss function would compute the discrepancy between the actual and anticipated prices of the property.

In essence, the loss function gives a numerical value that expresses the inaccuracy or difference between the actual value and the value predicted by the model. This aids in modifying the model's parameters to enhance its forecasts.

**Common Loss Functions:**

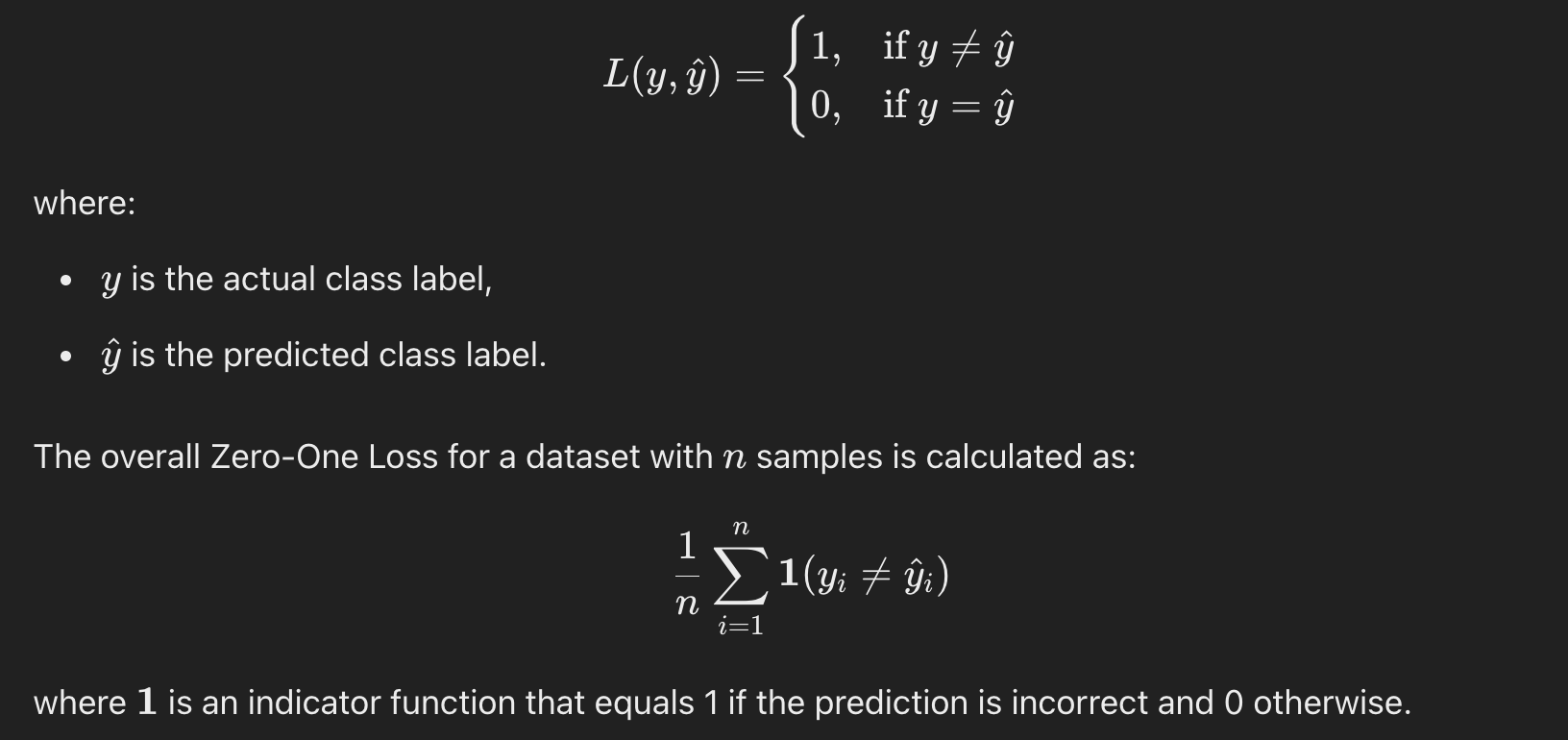
The two common loss functions for regression model are:

1. **Root Mean Square Error** : **RMSE** is a commonly used loss function in regression that measures the average magnitude of errors between predicted and actual values. It is calculated by taking the square root of the mean of squared differences between actual and predicted values. Because errors are squared before averaging, **RMSE penalizes larger errors more heavily than smaller ones**, making it particularly useful when large deviations need to be minimized. However, this also makes it more sensitive to outliers, which can significantly impact the overall error value.  
   
2. **Mean Absolute Error** : **MAE** measures the average absolute difference between actual and predicted values, treating all errors equally regardless of their magnitude. Unlike RMSE, **MAE does not square the errors, making it less sensitive to outliers** and providing a more straightforward interpretation of prediction accuracy. MAE is useful when all errors should be weighted equally and is easier to understand in practical applications.

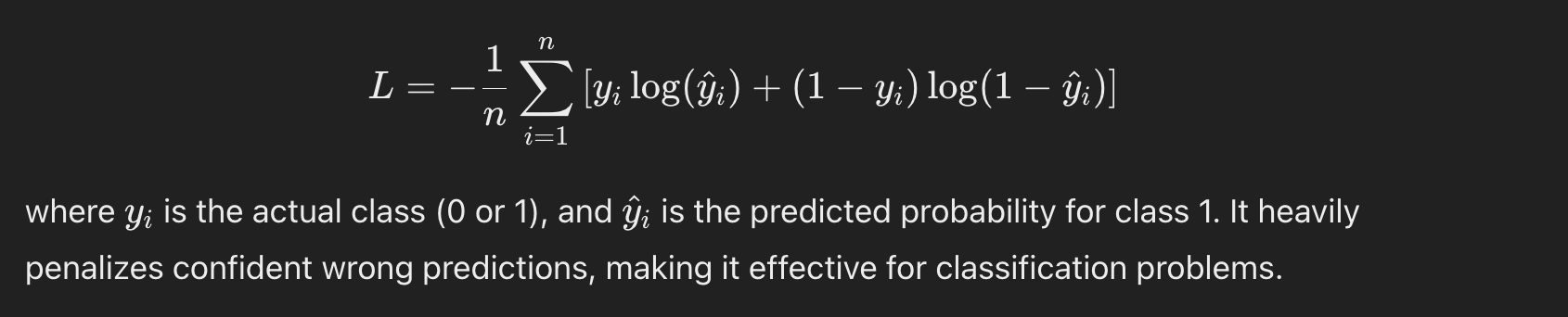


The two common loss functions for classification model are :

1. **Zero - One Loss** : **Zero-One Loss** is a simple loss function used in classification problems that counts the number of incorrect predictions. It assigns a loss of **1** for each incorrect prediction and **0** for correct ones. The overall loss is the proportion of misclassified instances, making it useful for evaluating model accuracy. However, it does not provide insights into how confident or close the predictions were.



1. **Cross-Entropy Loss (Log Loss)**: This loss function is widely used for classification tasks, particularly in logistic regression and neural networks. It measures the difference between the actual class labels and the predicted probability distribution.



**Q3. Consider the following scenario. You are building a classification model with many hyperparameters on a relatively small dataset. You will see that the training error is extremely small. Can you fully trust this model? Discuss the reason.**

Ans. When building a classification model with many hyper parameters on a relatively small dataset, if the training error is extremely small, **we cannot fully trust the model's performance**. The small training error could be an indication that the model has memorized the training data, a phenomenon known as overfitting. In this case, the model may perform exceptionally well on the training data (also called the in-sample data), but it may not generalize well to unseen data or test data (also known as out-of-sample data).

**Reasons for Not Fully Trusting the Model:**

1. **Overfitting**:
   1. Overfitting occurs when the model becomes **too complex** and learns not only the true underlying patterns in the data but also the **noise** and **random fluctuations** that do not generalize well to new data. As a result, the model fits the training data very closely but performs poorly on unseen data.
   2. This is especially problematic when using a small dataset because the model is more likely to memorize specific details in the data rather than learn broader, more generalizable patterns.
2. **Impact of Many Hyperparameters**:
   1. The presence of many **hyperparameters** increases the model's complexity and flexibility, allowing it to fit the training data in very specific ways. While this can help the model capture the training data well, it also **increases the risk of overfitting**. The model may learn patterns that are unique to the training set but do not apply to new data.
   2. More hyperparameters allow the model to fine-tune itself to fit the nuances of the training data, but this can come at the cost of **generalization ability**, especially when the dataset is small.

**How to Improve the Model:**

**Cross-Validation**:

* 1. To better understand the model's ability to generalize, you should use **cross-validation**. This involves splitting the dataset into multiple subsets, training the model on different subsets, and evaluating its performance on the remaining data. This helps provide a more reliable estimate of how the model will perform on unseen data and reduces the likelihood of overfitting.

**Regularization**:

* 1. Implementing **regularization techniques** (such as **L1** or **L2 regularization**) can help mitigate overfitting by adding penalties to large model coefficients. Regularization encourages the model to prioritize simpler, more generalizable patterns instead of overly complex ones that fit the noise in the training data.

**Simplify the Model**:

* 1. Reducing the **number of hyperparameters** or simplifying the model architecture can help prevent overfitting. A simpler model is less likely to memorize the training data and is more likely to generalize well to new, unseen data.

**More Data**:

* 1. If feasible, collecting more data is one of the best ways to reduce the risk of overfitting. More data allows the model to learn broader, more general patterns and helps it avoid memorizing the specific characteristics of the training set.

**Q4. What is the role of the lambda parameter in regularized linear models such as Lasso or Ridge regression models?**

In regularized linear models, such Lasso (L1 regularization) and Ridge (L2 regularization) regression, the lambda parameter is essential for controlling the trade-off between model performance and complexity.

Lambda's function is to regulate how strongly the model is regularized. In particular, it establishes the penalty that is applied to the feature coefficients during training. By discouraging excessively large coefficients, this penalty helps avoid overfitting and makes sure the model doesn't get too complicated.

**Lasso (L1 regularization):**

The lambda parameter in Lasso (L1 regularization) penalizes the absolute values of the coefficients. More coefficients are decreased to zero as lambda rises in magnitude. This procedure results in feature selection, whereby only the most important features are kept in the model and unimportant features are removed by giving them a coefficient of zero.

To put it another way, Lasso employs lambda to assist the model in concentrating on the most crucial aspects, hence decreasing model complexity and guaranteeing improved generalization to unknown data.

**Ridge (L2 regularization):**

The penalty is imposed on the squared values of the coefficients in Ridge (L2 regularization). Ridge decreases coefficients toward zero, but never quite to zero, in contrast to Lasso, which sets them to zero. As a result, less significant characteristics have smaller coefficients but nevertheless make a contribution to the model, although a lesser one.

In Ridge regression, the model becomes increasingly regularized as lambda rises, which means that while the less significant features are not entirely eliminated from the model, their impact is reduced. By doing this, the model is able to take into account irrelevant features without becoming overly dependent on them.

In conclusion, Lambda regulates the degree of regularization, striking a balance between feature relevance (by keeping the most crucial features) and model complexity (by punishing large coefficients).

Lambda promotes feature selection in Lasso (L1) by essentially eliminating unimportant features from the model by decreasing their coefficients to zero.

Lambda reduces, but does not completely remove, the coefficients of less significant features in Ridge (L2), resulting in smaller, more stable coefficients.

For the model to be both sophisticated enough to correctly forecast new, unknown data and simple enough to prevent overfitting, the lambda must be chosen carefully.

**PART – B**

QB1. 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? (Hint1: Do not forget to scale your input attributes – you can use the caret preprocess() function to scale and center the data. Hint 2: glment library expect the input attributes to be in the matrix format. You can use the as.matrix() function for converting)  
  
QB2. What is the coefficient for the price (normalized) attribute in the best model (i.e. model with the optimal lambda)?

QB3. 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? (

QB4. Build an elastic-net model with alpha set to 0.6. What is the best value of lambda for such  
a model?

library(ISLR)  
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)

## Loading required package: Matrix

## Loaded glmnet 4.1-8

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

head(Carseats)

## Sales CompPrice Income Advertising Population Price ShelveLoc Age Education  
## 1 9.50 138 73 11 276 120 Bad 42 17  
## 2 11.22 111 48 16 260 83 Good 65 10  
## 3 10.06 113 35 10 269 80 Medium 59 12  
## 4 7.40 117 100 4 466 97 Medium 55 14  
## 5 4.15 141 64 3 340 128 Bad 38 13  
## 6 10.81 124 113 13 501 72 Bad 78 16  
## Urban US  
## 1 Yes Yes  
## 2 Yes Yes  
## 3 Yes Yes  
## 4 Yes Yes  
## 5 Yes No  
## 6 No Yes

Carseats\_Filtered <- Carseats %>% select(Sales, Price, Advertising, Population, Age, Income, Education)  
head(Carseats\_Filtered)

## Sales Price Advertising Population Age Income Education  
## 1 9.50 120 11 276 42 73 17  
## 2 11.22 83 16 260 65 48 10  
## 3 10.06 80 10 269 59 35 12  
## 4 7.40 97 4 466 55 100 14  
## 5 4.15 128 3 340 38 64 13  
## 6 10.81 72 13 501 78 113 16

# Preprocessing data: Scaling and centering predictors  
preproc <- preProcess(Carseats\_Filtered[, -1], method = c("center", "scale"))  
Carseats\_Scaled <- predict(preproc, Carseats\_Filtered[, -1])  
head(Carseats\_Scaled)

## Price Advertising Population Age Income Education  
## 1 0.1776010 0.6563550 0.07572445 -0.6989069 0.1551667 1.18296763  
## 2 -1.3851191 1.4081936 -0.03284107 0.7208201 -0.7381360 -1.48824960  
## 3 -1.5118261 0.5059873 0.02822704 0.3504565 -1.2026533 -0.72504468  
## 4 -0.7938196 -0.3962189 1.36494005 0.1035475 1.1199335 0.03816025  
## 5 0.5154865 -0.5465866 0.50998655 -0.9458160 -0.1664223 -0.34344221  
## 6 -1.8497116 0.9570904 1.60242713 1.5232746 1.5844509 0.80136517

# Converting predictors to matrix format (required for glmnet)  
X <- as.matrix(Carseats\_Scaled)  
y <- Carseats\_Filtered$Sales # Target variable  
  
# Performing Lasso regression with cross-validation to find best lambda  
set.seed(123) # For reproducibility  
cv\_lasso <- cv.glmnet(X, y, alpha = 1, nfolds = 10)  
  
# Finding the Best lambda value  
best\_lambda <- cv\_lasso$lambda.min  
cat("The value of optimal lambda is:", best\_lambda, "\n")

## The value of optimal lambda is: 0.004305309

# Finding Lambda 1se (more regularized choice)  
lambda\_1se <- cv\_lasso$lambda.1se  
cat("The value of 1se lambda is:", lambda\_1se, "\n")

## The value of 1se lambda is: 0.2832606

# Extracting coefficients for the best model (lambda.min)  
lasso\_coef <- coef(cv\_lasso, s = best\_lambda)  
  
# Finding the coefficient for "Price"  
price\_coef <- lasso\_coef["Price",]  
  
# Printing the coefficient for Price  
cat("The coefficient for the Price attribute (normalized) is:", price\_coef, "\n")

## The coefficient for the Price attribute (normalized) is: -1.353834

# Extracting coefficients for the best model (lambda.min)  
lasso\_coef1 <- coef(cv\_lasso, s = lambda\_1se)  
  
# Finding the coefficient for "Price"  
price\_coef1 <- lasso\_coef1["Price",]  
  
# Printing the coefficient for Price  
cat("The coefficient for the Price attribute (1se) is:", price\_coef1, "\n")

## The coefficient for the Price attribute (1se) is: -1.04155

# Extracting coefficients for lambda = 0.01  
coef\_lambda\_0.01 <- coef(cv\_lasso, s = 0.01)  
num\_nonzero\_0.01 <- sum(coef\_lambda\_0.01 != 0) - 1 # Exclude intercept  
  
# Extracting coefficients for lambda = 0.1  
coef\_lambda\_0.1 <- coef(cv\_lasso, s = 0.1)  
num\_nonzero\_0.1 <- sum(coef\_lambda\_0.1 != 0) - 1 # Exclude intercept  
  
# Printing results  
cat("Number of attributes remaining in the model at lambda = 0.01:", num\_nonzero\_0.01, "\n")

## Number of attributes remaining in the model at lambda = 0.01: 6

cat("Number of attributes remaining in the model at lambda = 0.1:", num\_nonzero\_0.1, "\n")

## Number of attributes remaining in the model at lambda = 0.1: 4

# Performing Elastic Net regression with alpha = 0.6 and cross-validation  
set.seed(123) # For reproducibility  
cv\_elastic\_net <- cv.glmnet(X, y, alpha = 0.6, nfolds = 10)  
  
# Finding Best lambda value  
best\_lambda\_en <- cv\_elastic\_net$lambda.min  
  
# Printing best lambda  
cat("The optimal lambda for Elastic Net (alpha = 0.6) is:", best\_lambda\_en, "\n")

## The optimal lambda for Elastic Net (alpha = 0.6) is: 0.006538062