REGULARIZATION

In Machine Learning, we train our data and let ot learn from it. When fitting our data points, our goal is to find the best fit when it can find all necessary patterns in our data and avoid random data points and unnecessary patterns called Noise. While training a machine learning model, the model can easily be overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit a model onto our test set.

Regularization:

This is the technique that helps in avoiding overfitting and also it increasese model interpretability.

This is a form of regression, that constrains / regularizes or shrinks the coefficient estimates towards zero.

In addition, this technique discourages learning a more complex or flexible model, so as avoid the risk of overfitting.

Further, Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

Overfitted Model:

The model is not able to predict the output or target column for the unseen data by introducing noise in the output.

When a Machine learning model tries to learn from the details along with the noise in the data and tries to fit each data point on the curve.

Underfitting model:

When a Machine learning model can neither learn the relationship between variables in the testing data nor predict or classify a new data point.

"Noise" - this means those data points in the dataset which don't really represent the true properties of your data, but only due to a random chance.

Types of Regularization Techniques

1. Ridge Regression
2. Lasso Regression

Application on Dataset

We will use dataset that describes First year students GPA. The data set contains information about 219 randomly selected first-year students at a midwestern college. This dataset contains variables such as GPA(first year GPA), HSGPA(high school GPA), SATV, SATM, Male(an indicator variable for Male vs Female), HU, SS, White(an indicator variable for white student vs non white), FirstGen(an indicator variable for first-generation students), and CollegeBound. We will be using GPA as our response veariable and other variables as predictors. Our main goals are;

1. Pull off some of the data to use as training data and some to use as test data. Set up a grid for λ (but here we will use alpha instead).
2. Fit a ridge regression to the training data, then use cross-validation to choose the optimal value of λ. For this value of λ, calculate the test MSE on our test (validation) set. Record this test MSE, the value of λ, and the coefficients from the fitted ridge regression. Are any of the estimated coefficients 0? Is this what you would expect?
3. Repeat the previous bullet point for lasso regression.

Note: Lower RMSE and higher R-Squared values are indicative of a good model.

In [19]:
```
# importing libraries
%matplotlib inline

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
```

In [2]:
```
# Loading dataset
GPA = pd.read_csv('FirstYearGPA.csv')
```

In [3]:
```
# Looking at the first 5 column of the FirstYearGPA dataset
GPA.head()
```

Out[3]:

	GPA	HSGPA	SATV	SATM	Male	HU	SS	FirstGen	White	CollegeBound
0	3.06	3.83	680	770	1	3.0	9.0	1	1	1
1	4.15	4.00	740	720	0	9.0	3.0	0	1	1
2	3.41	3.70	640	570	0	16.0	13.0	0	0	1
3	3.21	3.51	740	700	0	22.0	0.0	0	1	1
4	3.48	3.83	610	610	0	30.5	1.5	0	1	1

In [4]:
```
# looking at the summary of the dataframe which includes the index dytpe and c
olumn, non null values, and memory usage
# GPA.info()
```

In [5]:
```
# Looking at the descriptive statistics
# Descriptive statistics includes those that summarize the central tendency, d
ispersion, and shape of a dataset's distribution, excluding null values.
# GPA.describe()
```

In [6]:
```
# Preparing dataset
target_column = ['GPA']
predictors = list(set(list(GPA.columns))-set(target_column))

# Normalizing by scaling the predictors between 0 and 1. This is done when the
units of the varibalrs differ significantly and may influence the modeling pro
cess.
# GPA[predictors] = GPA[predictors]/GPA[predictors].max()
```

We will build our model on the training set and evaluate its performance on the test set. This is called the holdout-validation approach for evaluating model performance.

X : independent variables (Other variables except GPA)

y : dependent variables (GPA)

The output shows that the shape of the training set has 153 observation of 9 variables and test set has 66 observation of 9 variables.

In [7]:
```
# Creating the training and Test datasets
X = GPA[predictors].values
y = GPA[target_column].values

# or could use
# X = GPA.iloc[:, :-1].values
# y = GPA.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=40)

print(X_train.shape)
print(X_test.shape)
```

```
(153, 9)
(66, 9)
```

Linear Regression

The simplest form of regression is linear regression, which assumes that the predictors have a linear relationship with the target variable. The input variables are assumed to have a Gaussian distribution and are not correlated with each other (a problem called multi-collinearity).

The linear regression equation can be expressed in the following form: y = a_1x_1 + a_2x_2 + a_3x_3 + ... + a_x_n + b, where y is the target variable, x1, x2, x3, ... xn are the features, a1, a2, a3, ... an are the coefficients, and b is the parameter of the model.

The parameters a and b in the model are selected through the ordinary least squares (OLS) method. This method works by minimizing the sum of squares of residuals (actual value - predicted value).

Linear regression algorithm works by selecting coefficients for each independent variable that minimizes a loss function. However, if the coefficients are large, they can lead to over-fitting on the training dataset, and such a model will not generalize well on the unseen test data. To overcome this shortcoming, we will do regularization as shown below, which penalizes large coefficients.

In [8]:
```
# Multiple Linear regression model
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [9]:
```
# Genereating prediction on test set
lm_train_pred = lm.predict(X_train)
```

In [10]:
```
# prints the evaluation metrics (RMSE and R-squared) on the train set
print(np.sqrt(mean_squared_error(y_train,lm_train_pred)))
print(r2_score(y_train, lm_train_pred))

# prints the evaluation metrics (RMSE and R-squared) on the test set
lm_test_pred = lm.predict(X_test)
print(np.sqrt(mean_squared_error(y_test,lm_test_pred)))
print(r2_score(y_test, lm_test_pred))
```

```
0.3796918126336791
0.37486548794143515
0.37363947388891877
0.21748129673159344
```

Ridge Regression model training data results:

1. RMSE : 0.379692
2. R-squared : 0.37486548794143526 or approximately 37%

Ridge Regression model testdata results:

1. RMSE : 0.373639
2. R-squared : 0.21748129673159644 or approximatly 22%

In [11]:
```
# retrieving the intercept
print(lm.intercept_)
```

```
[0.52401237]
```

In [12]:
```
# retrieving the coefficients
print(lm.coef_)
```

```
[[ 7.82661088e-04  5.43897320e-05  1.53005147e-02 -4.95044729e-03
   1.61712780e-01 -7.74565719e-02 -8.42162608e-02  1.10147628e-02
   5.08814019e-01]]
```

RIDGE REGRESSION

Ridge regression is also knowns as L-2 norm

Ridge regression modifies the overfitted or underfitted models by adding the penalty equivalent to the sum of the squares of the magnitude of coefficients.

By changing the values of the penalty function, we are controlling the penalty term. The higher the penalty, it reduces the magnitude of coefficients. It shrinks the parameters. Therefore, it is used to prevent multicollinearity, and it reduces the model complexity by coefficient shrinkage.

Ridge Regression performs regularization by shrinking the coefficients present. Ridge regression seeks coefficient estimates that fit the data well, by making the RSS small.

Creating an object of the target variable and predictor variables.

In [13]:
```
# Ridge regression model with an alpha value of 0.01
rr = Ridge(alpha=0.01)

# fitting the model to the training data
rr.fit(X_train, y_train)

# predicting
pred_train_rr= rr.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print(np.sqrt(mean_squared_error(y_train,pred_train_rr)))
print(r2_score(y_train, pred_train_rr))

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_rr= rr.predict(X_test)
print(np.sqrt(mean_squared_error(y_test,pred_test_rr)))
print(r2_score(y_test, pred_test_rr))
```

```
0.37969182630313725
0.37486544292992907
0.37363514830697914
0.21749941488780444
```

Ridge Regression model training data results:

1. RMSE : 0.379692
2. R-squared : 0.37486544292992907 or 37%

Ridge Regression model testdata results:

1. RMSE : 0.373635
2. R-squared : 0.21749941488780455 or 22%

Results for Ridge regression looks similar to the Linear Regression. We will look into Lasso Regression and see if there is any improvement.

In [14]:
```
print(rr.intercept_)
```

```
[0.52455127]
```

In [15]:
```
print(rr.coef_)
```

```
[[ 7.83065411e-04  5.46101724e-05  1.53048554e-02 -4.98757225e-03
   1.61626624e-01 -7.73468585e-02 -8.42127159e-02  1.10174104e-02
   5.08545821e-01]]
```

LASSO REGRESSION

Lasso stands for Least Absolute and Selection Operator which is know as the L-1 norm

Lasso regression is used to reduce the complexity of the model

Lasso regression is similar to Ridge regression except that the penalty term incudes the absolute weights instead of a square weights.

In this technique, the L1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero which means there is a complete removal of some of the features for model evaluation when the tuning parameter λ is sufficiently large. Therefore, the lasso method also performs Feature selection and is said to yield sparse models.

In [16]:
```
# Lasso Regression Model with an alpha value of 0.01
model_lasso = Lasso(alpha=0.01)

# fitting the model to the training data
model_lasso.fit(X_train, y_train)

# predicting
pred_train_lasso= model_lasso.predict(X_train)

# prints the evaluation metrics (RMSE and R-squared) on the train set
print(np.sqrt(mean_squared_error(y_train,pred_train_lasso)))
print(r2_score(y_train, pred_train_lasso))

# prints the evaluation metrics (RMSE and R-squared) on the test set
pred_test_lasso= model_lasso.predict(X_test)
print(np.sqrt(mean_squared_error(y_test,pred_test_lasso)))
print(r2_score(y_test, pred_test_lasso))
```

```
0.382996095166865
0.36393762985663713
0.370054775739954
0.23242424310395493
```

Lasso Regression model training data results:

1. RMSE : 0.382996
2. R-squared : 0.36393762633768645 or 36%

Lasso Regression model testdata results:

1. RMSE : 0.370055
2. R-squared : 0.23242444674596452 or 23%

We can observe that there is an improve in the test result of Lasso Regression.

In [17]:
```
print(model_lasso.intercept_)
```

```
[0.57499553]
```

In [18]:
```
print(model_lasso.coef_)
```

```
[ 9.77780003e-04  8.11181069e-05  1.70581894e-02 -0.00000000e+00
  9.68053558e-02 -0.00000000e+00 -0.00000000e+00  1.18657493e-02
  4.36251056e-01]
```

The performance of the models is summarized below:

1. Linear Regression Model: Test set RMSE of 0.373639 and R-square of 22%.
2. Ridge Regression Model: Test set RMSE of 0.373635 and R-square of 22%.
3. Lasso Regression Model: Test set RMSE of 0.370055 and R-square of 23%.

Lasso Regression model performs better than the linear and Ridge regression model. Overall, all the models are performing well with decent R-squared and stable RMSE values. The most ideal result would be an RMSE value of zero and R-squared value of 1.