**Regularization in Machine Learning**

Introduction

When training a machine learning model, the model can be easily overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit the model to our test set. Regularization techniques help reduce the possibility of overfitting and help us obtain an optimal model. In this article titled ‘The Ultimate Guide to Regularization in Machine Learning, you will learn everything you need to know about regularization.

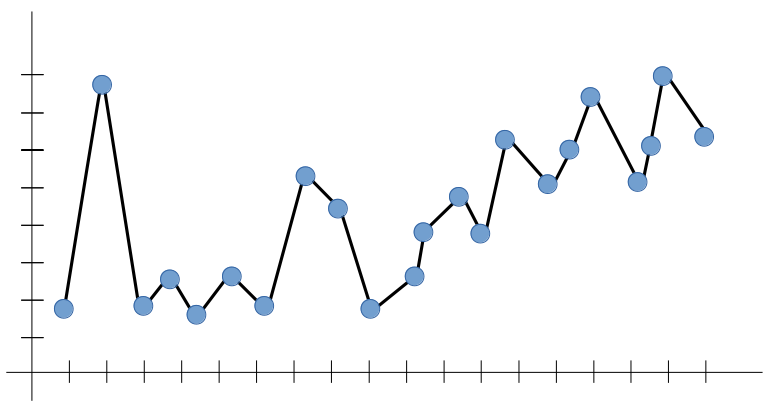
Understanding Overfitting and Underfitting

To train our machine learning model, we provide it with data to learn from. The process of plotting a series of data points and drawing a line of best fit to understand the relationship between variables is called Data Fitting. Our model is best suited when it can find all the necessary patterns in our data and avoid random data points, and unnecessary patterns called noise.

If we allow our machine learning model to look at the data too many times, it will find many patterns in our data, including some that are unnecessary. It will learn well on the test dataset and fits very well. It will learn important patterns, but it will also learn from the noise in our data and will not be able to make predictions on other data sets.

A scenario where a machine learning model tries to learn from the details along with the noise in the data and tries to fit each data point to a curve is called Overfitting.

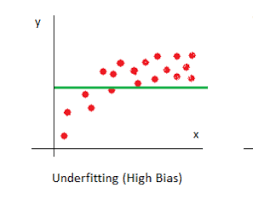
In the figure below, we can see that the model is fit for every point in our data. If new data is provided, the model curves may not match the patterns in the new data, and the model may not predict very well.



Conversely, in the scenario where the model has not been allowed to look at our data enough times, the model will not be able to find patterns in our test data set. It won’t fit our test data set properly and won’t work on new data either.

A scenario where a machine learning model can neither learn the relationship between variables in the test data nor predict or classify a new data point is called Underfitting.

The image below shows an underequipped model. We can see that it doesn’t fit the data given correctly. He did not find patterns in the data and ignored much of the data set. It cannot work with both known and unknown data.

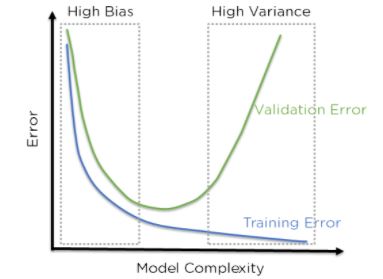


What are the Bias and Variance?

Bias comes out when the algorithm has limited flexibility to learn from the dataset. These models pay little attention to the training data and oversimplify the model, so the validation or prediction errors and training errors follow similar trends. Such models always lead to high errors in the training and test data. High bias causes under-adjustment in our model.

The variance defines the sensitivity of the algorithm to specific data sets. A high-variance model pays close attention to the training data and does not generalize, so the validation or prediction errors are far from each other. Such models usually perform very well on the training data but have a high error rate on the test data. High deviation causes an overshoot in our model.

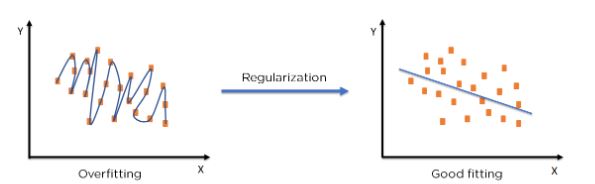
An optimal model is one in which the model is sensitive to the pattern in our model but can also generalize to new data. This occurs when both bias and variance are optimal. We call this the Bias-Variance Tradeoff, and we can achieve this in models over or under-fitted models using regression.



The above figure shows that when the bias is high, the error in both the test and training sets is also high. When the deviation is high, the model performs well on our training set and gives a low error, but the error on our test set is very high. In the middle of this, there is a region where bias and variance are in perfect balance with each other here too, but training and testing errors are low.

Regularization in Machine Learning?

Regularization refers to techniques used to calibrate machine learning models to minimize the adjusted loss function and avoid overfitting or underfitting.

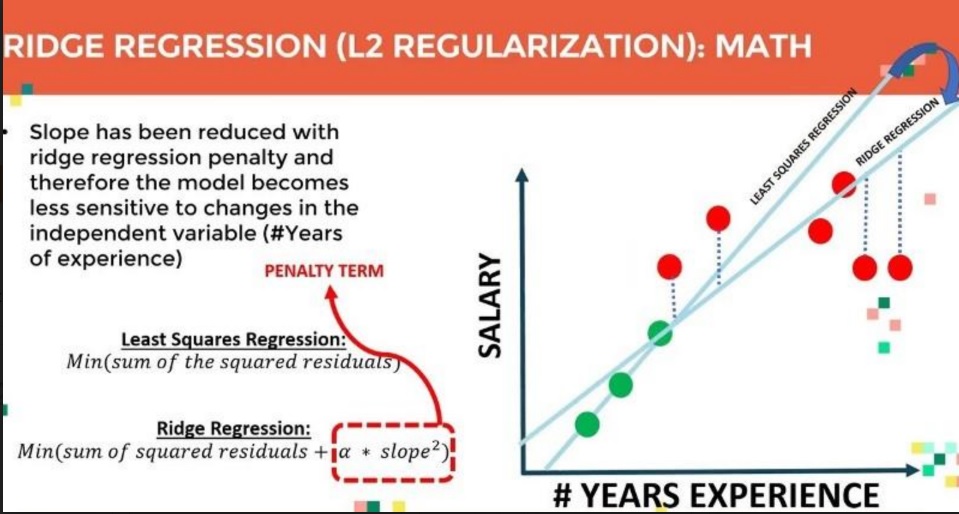


Regularization Techniques

**Ridge Regularization**

Also known as Ridge Regression, it adjusts models with overfitting or underfitting by adding a penalty equivalent to the sum of the squares of the magnitudes of the coefficients.

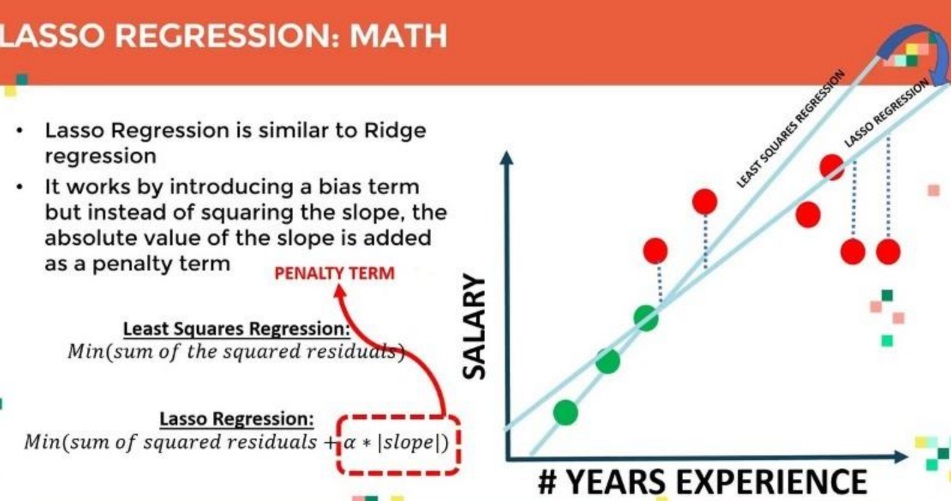
This means that the mathematical function representing our machine learning model is minimized and the coefficients are calculated. The size of the coefficients is multiplied and added. Ridge Regression performs regularization by reducing the coefficients present. The function shown below shows the cost function of the ridge regression.



**Lasso Regularization**

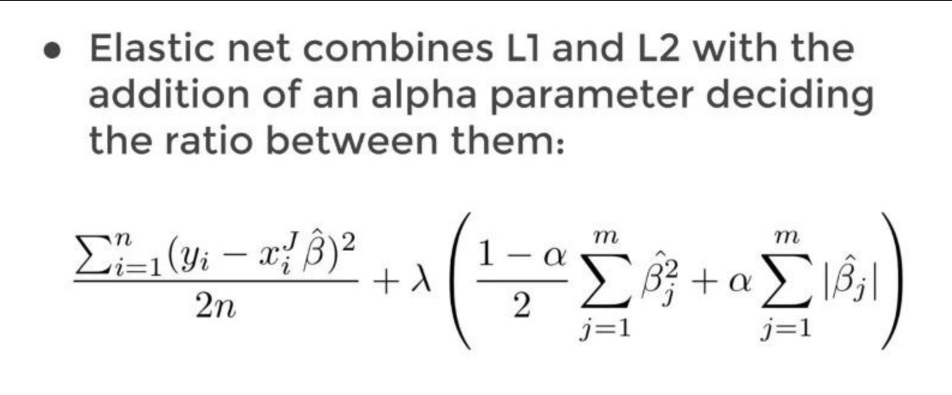
Modifies overfitted or under-fitted models by adding a penalty equivalent to the sum of the absolute values ​​of the coefficients.

Lasso regression also performs coefficient minimization, but instead of squaring the magnitudes of the coefficients, it takes the actual values ​​of the coefficients. This means that the sum of the coefficients can also be 0 because there are negative coefficients. Consider the cost function for the lasso regression.



**Elastic Net**

Elastic Net combines L1 and L2 With the addition of an alpha Parameter.



Regularization Using Python in Machine Learning

Let’s see how regularization can be implemented in Python. We have taken the Advertising Dataset on which we will use linear regression to predict Advertisement cost.

We start by importing all the necessary modules.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

We then load the Advertisement Dataset from sklearn’s datasets.

df = pd.read\_csv("Advertising.csv")

Splitting the Dataset into Training and Testing Dataset:

Applying the Train Train Split:

**Python Code:**

Now we can use them for training our linear regression model. We’ll start by creating our model and fitting the data into it. We then predict on the test set and find the error in our prediction using mean\_squared\_error. Finally, we print the coefficients of our linear regression model.

**Ridge Regression:**

 Modelling with default Parameters:

from sklearn.linear\_model import Ridge

ridge\_model = Ridge()

ridge\_model.fit(X\_train, y\_train)

Predictions and Evaluation Of Ridge Regression:

test\_predictions = ridge\_model.predict(X\_test)

train\_predictions = ridge\_model.predict(X\_train)

**Hyperparameter Tuning of Ridge :**Identifying the best alpha value for Ridge Regression:

from sklearn.model\_selection import GridSearchCV

estimator = Ridge()

estimator = Ridge()

param\_grid = {"alpha":list(range(1,11))}

model\_hp = GridSearchCV(estimator, param\_grid, cv = 5)

model\_hp.fit(X\_train, y\_train)

model\_hp.best\_params\_

**Lasso Regularization:**

Modeling of Lasso Regularization:

from sklearn.linear\_model import Lasso

lasso\_model = Lasso()

lasso\_model.fit(X\_train, y\_train)

Predictions and Evaluation Of Lasso Regression:

test\_predictions = lasso\_model.predict(X\_test)

train\_predictions = lasso\_model.predict(X\_train)

from sklearn.metrics import mean\_squared\_error

train\_rmse = np.sqrt(mean\_squared\_error(y\_test, test\_predictions))

test\_rmse = np.sqrt(mean\_squared\_error(y\_train, train\_predictions))

print("train RMSE:", train\_rmse)

print("test RMSE:", test\_rmse)

**Hyperparameter Tuning of Lasso:**

Identifying the best alpha value for Lasso Regression:

param\_grid = {"alpha": list(range(1,11))}

model\_hp = GridSearchCV(estimator, param\_grid, cv =5)

model\_hp.fit(X\_train, y\_train)

model\_hp.best\_estimator\_

**Elastic Net Regularization:**

modeling of Elastic Net Regularization:

from sklearn.linear\_model import ElasticNet

enr\_model = ElasticNet(alpha=2, l1\_ratio = 1)

enr\_model.fit(X\_train, y\_train)

Predictions and Evaluation Of Elastic Net:

test\_predictions = enr\_model.predict(X\_test)

train\_predictions = enr\_model.predict(X\_train)

from sklearn.metrics import mean\_squared\_error

train\_rmse = np.sqrt(mean\_squared\_error(y\_test, test\_predictions))

test\_rmse = np.sqrt(mean\_squared\_error(y\_train, train\_predictions))

print("train RMSE:", train\_rmse)

print("test RMSE:", test\_rmse)

**Hyperparameter Tuning of Elastic Net:**Identifying the best alpha value for Elastic Net:

from sklearn.model\_selection import GridSearchCV

enr\_hp = GridSearchCV(estimator, param\_grid)

enr\_hp.fit(X\_train, y\_train)

enr\_hp.best\_params\_

param\_grid = { "alpha" : [0, 0.1, 0.2, 1, 2, 3, 5, 10],

"l1\_ratio" : [0.1, 0.5, 0.75, 0.9, 0.95, 1]}

estimator = ElasticNet()

Conclusion

In this article – The Ultimate Guide to Regularization in Machine Learning, we learned about the different ways models can become unstable by being under- or overfitted. We observed the role of bias and Variance. We then moved to regularization techniques to overcome overfitting and underfitting. and Finally, we Saw a Python Code Implementation.

Also In this article, we studied Overfittings and Underfitting of a Linear model and how regularization Techniques can be used to overcome these issues.

* We learned about the [L1 and L2 Regularization](https://www.analyticsvidhya.com/blog/2021/09/lasso-and-ridge-regularization-a-rescuer-from-overfitting/), which are added to the cost function.
* In each of these algorithms, we can set to specify several [hyperparameters](https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/" \t "_blank).
* We can use GridSearchCV or the respective Hyperparameter Tuning algorithms of the given respective regression model to find the optimal hyperparameters.

You can try to compare the performance of these algorithms on a dataset and check which algorithm performed better using a performance metric like Root Mean Square Error or RMSE.