# LIFELONG MACHINE LEARNING

Submitted as a project report for mini project

For the partial fulfilment of the degree of

Bachelor of Technology

in

Information Technology

by

SOUMYADITYA MONDAL <2021ITB052>

AYUSH RAJ <2021TIB016>

AAYUSH KUMAR <2021ITB026>

Under the supervision of

Indrajit Banerjee

Associate Professor

Department of Information Technology

IIEST, Shibpur

May, 2023

Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
P.O. Botanic Garden, Howrah 711103, WB, India
+91 33 26684561-3 (3 lines)/0521-5 (5 lines) X260
https://www.iiests.ac.in/IIEST/AcaUnitDetails/IT

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Date:

TO WHOM IT MAY CONCERN

This is to certify that Soumyaditya Mondal <2021ITB052>, Ayush Raj <2021ITB016> and Aayush Kumar <2021ITB026> have done their mini project on Lifelong Machine Learning for the partial fulfilment of the degree of B.Tech. in Information Technology.

During this period, they have completed the project. The report has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

Prof. Indrajit Banerjee
Associate Professor
Department of Information Technology

Prof. Prasun Ghosal
Head of Department
Department of Information Technology

# Acknowledgement

This work would not have been possible without the help and support of the Prof. Indrajit Banerjee, Associate Professor in Department of Information Technology at IIEST Shibpur. I am especially indebted to Mr. Gaidenlung Kamei, a PHD student at IIEST Shibpur, who have been supportive of my career goals and who worked actively to provide me with the protected academic time to pursue those goals. As my teacher and mentor, they have taught me more than I could ever give them credit for here. They are the ultimate role models.

Soumyaditya Mondal <2021ITB052>

Ayush Raj <2021ITB016>

Aayush Kumar <2021ITb026>

# Index

# Chapter 1
# Introduction

Machine Learning has great and widespread applications in our practical world. Recent successes in deep learning brought ML to a new emerging height. Without effective ML algorithms, many industries would not have existed or flourished. Among different applications and models, Lifelong Machine Learning (or Lifelong Learning) is leading to more promising directions towards an ultimate goal of building machines that learn like humans.

A classic ML paradigm is to run an algorithm on a given dataset to generate a model, and then model is applied in real-life performance tasks. We call this paradigm ISOLATED LEARNING as it does not consider any related or previously learned knowledge. The fundamental problem with this isolated learning paradigm is that it does not retain and accumulate knowledge learned in past and use it in future learning. Without the ability to accumulate and use the past knowledge, a ML algorithm typically needs a large number of training examples in order to learn effectively. And this labeling of training data is often done manually which would be very labor-intensive and time-consuming.
Hence, we need a system that learns more and more over time and become more and more knowledgeable and more and more effective at learning. We need a system that can adapt the same but more efficient learning process as we human possess. LIFELONG MACHINE LEARNING or simply LIFELONG LEARNING (LL) aims at the same, i.e., to imitate the human learning process and capability.
Let us consider an example of Natural Language Processing (NLP) as it is easy to see the importance of LL to NLP for several reasons : -

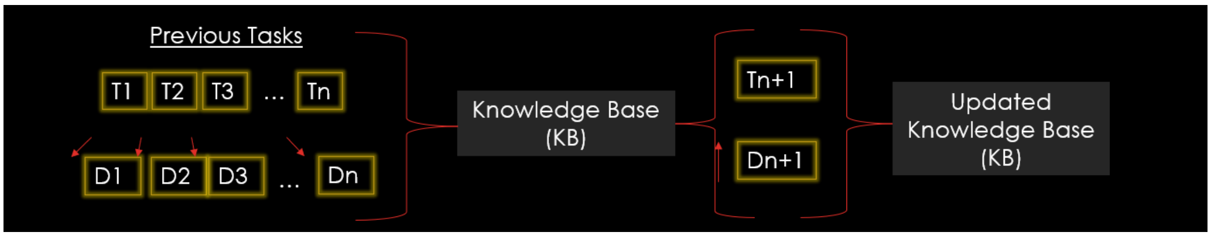
- First, words and phrases have almost the same meaning in all domains and all tasks.
- Second, sentences in every domain follow the same syntax or grammar.
- Third, almost all natural language processing problems are closely related to each other, which means that they are inter-connected and affect each other in some way.

The first to reasons ensure that the knowledge learned can be used across domains and tasks due to the sharing of the same expressions and meanings and the same syntax. The third reason ensures that LL can be used across different types of tasks.
Hence, we will be observing different implementation of this LL concept in our engineering life.

Now, let’s try to understand what LIFELONG LEARNING is!!

## What is LIFELONG LEARNING?

Lifelong learning is basically a continuous learning process where some knowledge is accumulated and maintained in knowledge base (KB) and using this previous knowledge for the purpose of future learning or discovering some new tasks.



Ideally, a LL learner should be able to:

- Learn and function in the open environment.
- Learn to improve the model performance in the application.

Hence, Key Characteristics of LL can be stated as:

➢ Continuous Learning Process
➢ Knowledge Accumulation
➢ Using the accumulated knowledge to learn more.
➢ Ability to discover new task.
➢ Ability to learn while working.

# Chapter 2
# About the Project

Diabetes is a chronic disease that affects millions of people worldwide, and early detection and management are essential to prevent long-term complications.
Machine Learning (ML) models have shown great promise in predicting the likelihood of an individual developing diabetes in the future, but these models require constant updates and improvements to ensure their accuracy and reliability. Additionally, there is a need for lifelong learning in diabetes prediction models to incorporate new data and adjust to changes in patients' lifestyles and health conditions.

## 1.1. Objective

To develop and ML-based diabetes prediction model that incorporates lifelong learning to adapt to changes in patients' health conditions and lifestyle factors.

## 1.2. Problem Statement

To show Lifelong Diabetes Prediction using SGD Regression.

## 1.3. Proposed Solution

We are hereby generating a Machine Learning Model and implementing the concept of lifelong learning to get the ultimate model. Serving to our purpose, we go over the following steps as mentioned:

- **Collect Dataset**
  First, we collect the data, showing the condition of patients suffering of diabetes in real-world, of fixed-width file format from any sources available.
  We choose to pick form "https://archive.ics.uci.edu/ml/datasets/Diabetes"

- **Converting Dataset to Data Frame**
  Now we convert the Dataset to tabular Data Frame in order to do efficient data analysis using the 'read_fwf' function of Pandas library.

- **Data Pre-processing**
  Data pre-processing refers to the steps and techniques applied to raw data before it is used for analysis or training a machine learning model. It is an essential step in the data pipeline and involves transforming, cleaning, and organizing the data to improve its quality and make it suitable for further analysis or modelling. Data pre-processing aims to address common challenges in data such as missing values, outliers, inconsistencies, and incompatible formats.
  We have done standardization, train-test split and data analysis under this domain.

- **Model Training**
  Finally, we will be training our model with above collected and modified (or pre-processed) using 'SGD Regression Model'.

- **Accuracy Calculation**
  We now check our model for some new data value and check for the accuracy in the result predicted by it.

- **Implementing LL Model**
  At last, we generate a portion that is implementing the concept of Lifelong Machine Learning Model to make it more efficient and better.

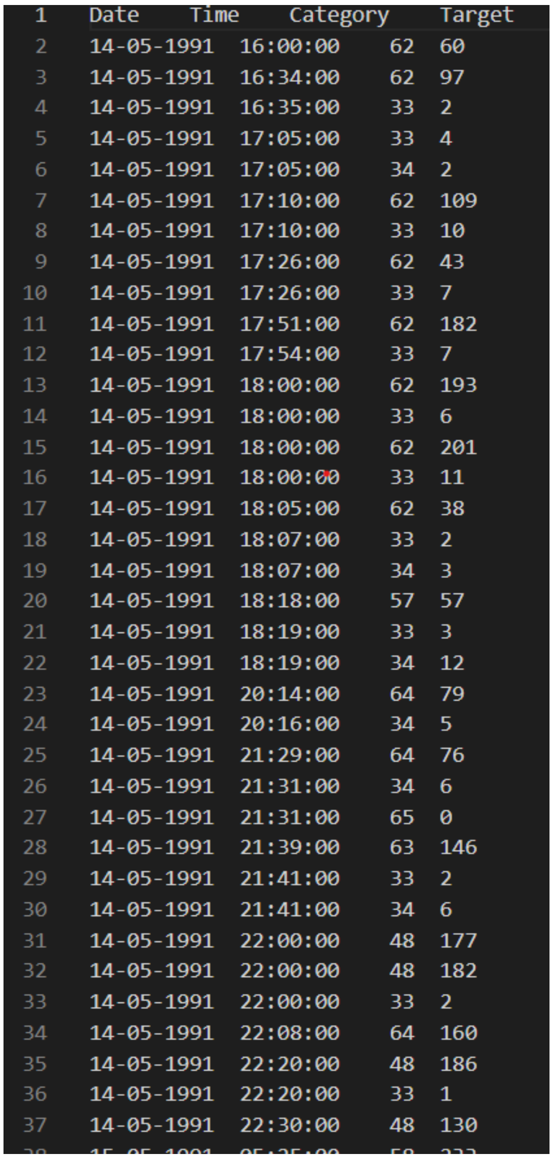
# Chapter 3

# Implementation and Experimental Procedure

## 3.1. Platforms Used:

1. Visual Studio Code
2. Jupyter Notebook
3. Google Collaboratory

## 3.2. Dataset Description



| Date | Time | Category | Target |
|---|---|---|---|
| 14-05-1991 | 16:00:00 | 62 | 60 |
| 14-05-1991 | 16:34:00 | 62 | 97 |
| 14-05-1991 | 16:35:00 | 33 | 2 |
| 14-05-1991 | 17:05:00 | 33 | 4 |
| 14-05-1991 | 17:05:00 | 34 | 2 |
| 14-05-1991 | 17:10:00 | 62 | 109 |
| 14-05-1991 | 17:10:00 | 33 | 10 |
| 14-05-1991 | 17:26:00 | 62 | 43 |
| 14-05-1991 | 17:26:00 | 33 | 7 |
| 14-05-1991 | 17:51:00 | 62 | 182 |
| 14-05-1991 | 17:54:00 | 33 | 7 |
| 14-05-1991 | 18:00:00 | 62 | 193 |
| 14-05-1991 | 18:00:00 | 33 | 6 |
| 14-05-1991 | 18:00:00 | 62 | 201 |
| 14-05-1991 | 18:00:00 | 33 | 11 |
| 14-05-1991 | 18:05:00 | 62 | 38 |
| 14-05-1991 | 18:07:00 | 33 | 2 |
| 14-05-1991 | 18:07:00 | 34 | 3 |
| 14-05-1991 | 18:18:00 | 57 | 57 |
| 14-05-1991 | 18:19:00 | 33 | 3 |
| 14-05-1991 | 18:19:00 | 34 | 12 |
| 14-05-1991 | 20:14:00 | 64 | 79 |
| 14-05-1991 | 20:16:00 | 34 | 5 |
| 14-05-1991 | 21:29:00 | 64 | 76 |
| 14-05-1991 | 21:31:00 | 34 | 6 |
| 14-05-1991 | 21:31:00 | 65 | 0 |
| 14-05-1991 | 21:39:00 | 63 | 146 |
| 14-05-1991 | 21:41:00 | 33 | 2 |
| 14-05-1991 | 21:41:00 | 34 | 6 |
| 14-05-1991 | 22:00:00 | 48 | 177 |
| 14-05-1991 | 22:00:00 | 48 | 182 |
| 14-05-1991 | 22:00:00 | 33 | 2 |
| 14-05-1991 | 22:08:00 | 64 | 160 |
| 14-05-1991 | 22:20:00 | 48 | 186 |
| 14-05-1991 | 22:20:00 | 33 | 1 |
| 14-05-1991 | 22:30:00 | 48 | 130 |

### Diabetes Dataset

1. Set of 70 data files having (29297*4) size.
2. Features of 4 columns are named as:
   - Date
   - Time
   - Category
   - Target
3. 5-step conversion process is used to generate final train & test data from the given dataset.
   - Date, month and year modification
   - Hour, minute, second modification
   - Handling missing values
   - Standardization
   - Train test split

## 3.3. Data Pre-processing

- **Data Analysis**
  Under the section of Data Analysis, we are arranging the collected dataset as per our model design using feature selection method.

- **Features Splitting**
  We, hereby go for splitting some targets and corelated features in the dataset. These targets include:
  Time – splitting it into hours-string, minutes-string and second-string.
  Date – splitting it into date-string, month-string and year-string.

- **Standardization**
  This is a way to limit the data range to avoid the high variance problems.

- **Train Test Split**
  Split the whole data by 4:1 ratio.

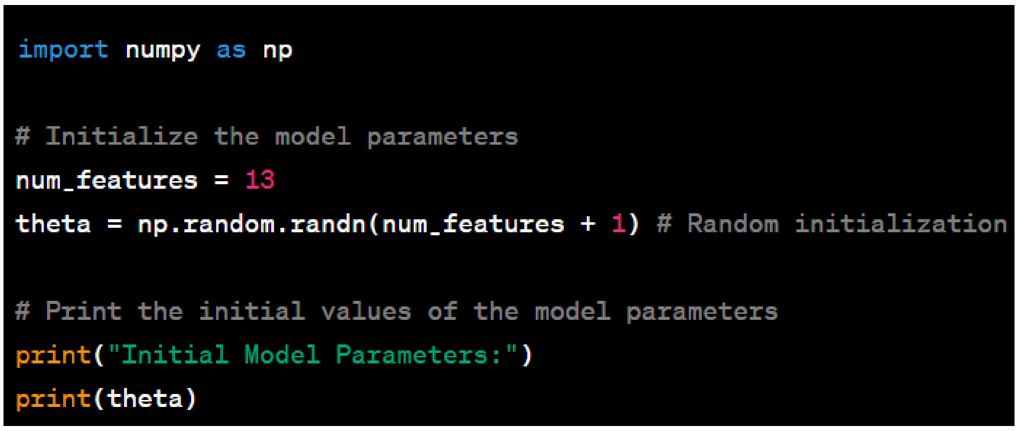
## 3.4. Models Used

- SGD Regression

```
from sklearn.linear_model import SGDRegressor
```

  - SGD is an optimization algorithm commonly used in machine learning to train regression models, among other applications.
  - Goals – to minimize the loss function of the regression model by adjusting the model parameters based on gradient of loss function with respect to all the parameters.

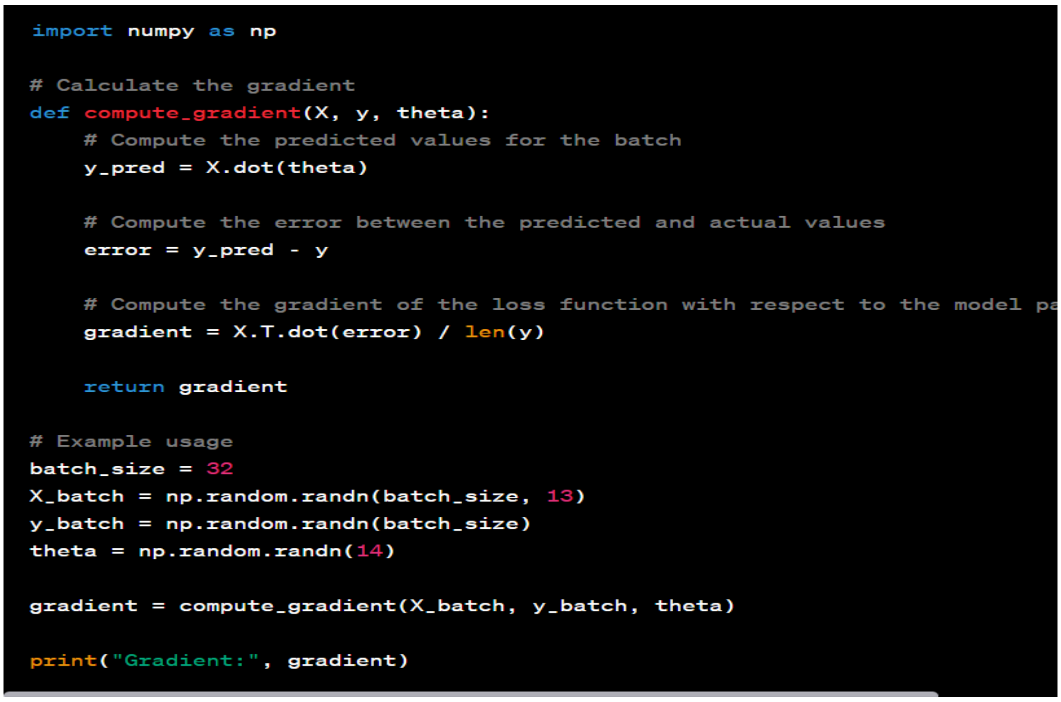
- Initialization



```
import numpy as np

# Initialize the model parameters
num_features = 13
theta = np.random.randn(num_features + 1) # Random initialization

# Print the initial values of the model parameters
print("Initial Model Parameters:")
print(theta)
```

In this example, we first import the necessary libraries and then define the number of features in our dataset. We then initialize the model parameter with random values using ‘np.random.randn’ function, which generate the random numbers from a normal distribution with mean 0 and variance 1.

- Gradient Calculation

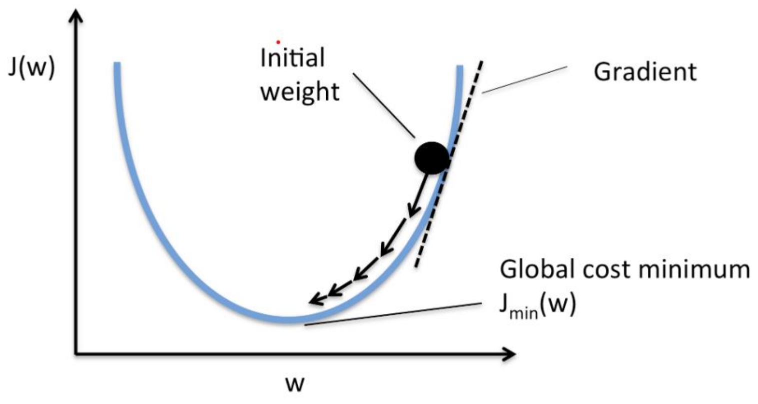


```python
import numpy as np

# Calculate the gradient
def compute_gradient(X, y, theta):
    # Compute the predicted values for the batch
    y_pred = X.dot(theta)

    # Compute the error between the predicted and actual values
    error = y_pred - y

    # Compute the gradient of the loss function with respect to the model pa
    gradient = X.T.dot(error) / len(y)

    return gradient

# Example usage
batch_size = 32
X_batch = np.random.randn(batch_size, 13)
y_batch = np.random.randn(batch_size)
theta = np.random.randn(14)

gradient = compute_gradient(X_batch, y_batch, theta)

print("Gradient:", gradient)
```

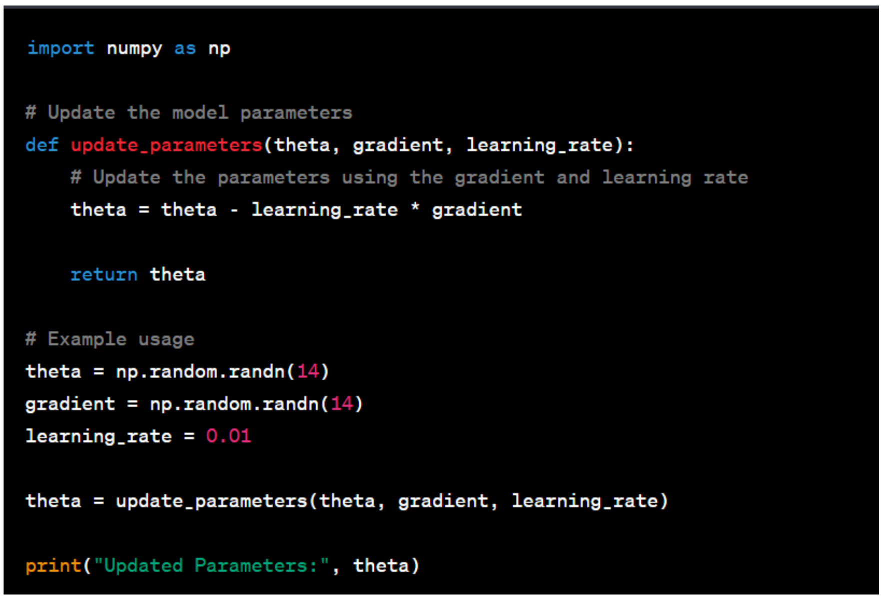
In this example, we define a function called ‘compute_gradient’ that takes as input a batch of training data points (‘**X**’ and ‘**Y**’) and the current values of the model parameters (‘**theta**’). The function then computes the predicted values for the batch using the dot product of ‘**X**’ and ‘**theta**’, computes the error between the predicted and actual values, and computes the gradient of the loss function with respect to the model parameters using the dot product of ‘**X.T**’ and ‘**error**’. Finally, the gradient is normalized by the length of y to account for the size of the batch.

We then generate a random batch of training data and model parameters to demonstrate how the function can be used. The ‘**batch_size**’ variable represents the number of training data points in the batch, and ‘**X_batch**’ and ‘**Y_batch**’ contain the feature and target values for the batch, respectively. The ‘**theta’** array contains the initial values of the model parameters.

Finally, we call the ‘compute_gradient’ function with the input data and model parameters, and print the resulting gradient. Note that the size of the gradient should be the same as the size of theta, since each element of theta corresponds to the weight of a feature in the regression equation.



- Parameter Update



```
import numpy as np

# Update the model parameters
def update_parameters(theta, gradient, learning_rate):
    # Update the parameters using the gradient and learning rate
    theta = theta - learning_rate * gradient

    return theta

# Example usage
theta = np.random.randn(14)
gradient = np.random.randn(14)
learning_rate = 0.01

theta = update_parameters(theta, gradient, learning_rate)

print("Updated Parameters:", theta)
```

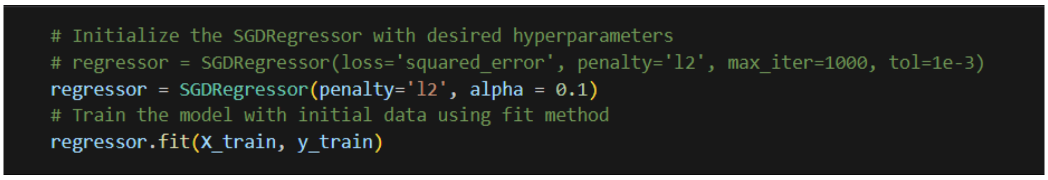
In this example, we define a function called ‘**update_parameters**’ that takes as input the current values of the model parameters (‘**theta**’), the calculated gradient of the loss function (‘**gradient**’), and the learning rate of the algorithm (‘**learning_rate**’). The function then updates the model parameters by subtracting the product of the gradient and learning rate from the current values of ‘**theta**’.

We then generate random values for ‘**theta**’, ‘**gradient**’, and ‘**learning_rate**’ to demonstrate how the function can be used.

Finally, we call the '**update_parameters**' function with the input data and print the resulting values of '**theta**'. The updated values of '**theta**' should be closer to the minimum of the loss function than the initial values.

- Repeat
  Repeat gradient calculation and parameter update for a fixed number of iterations for all the dataset on the point of view of lifelong learning model.

- Inbuilt model for SGD Regression



```
# Initialize the SGDRegressor with desired hyperparameters
# regressor = SGDRegressor(loss='squared_error', penalty='l2', max_iter=1000, tol=1e-3)
regressor = SGDRegressor(penalty='l2', alpha = 0.1)
# Train the model with initial data using fit method
regressor.fit(X_train, y_train)
```

The above steps inside SGD Regression are summarized in this inbuilt function during project code is performed.

We, in our model, also use the concept of '**REGULARIZATION**'.

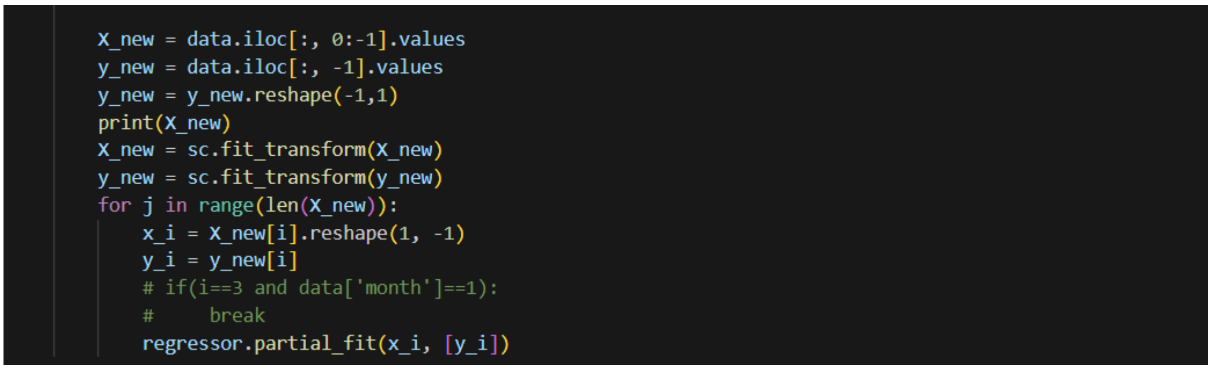
Regularization is a technique used in machine learning to prevent overfitting, which occurs when a model learns the training data too well and fails to generalize to new, unseen data.

L2 regularization, also known as Ridge regularization, adds a penalty term to the loss function that is proportional to the squared values of the weights. The L2 penalty has the effect of shrinking the weights towards zero, without eliminating them completely.

By using L2 regularization in the SGD regression model, we are able to prevent overfitting and improve the generalization performance of the model on new, unseen data.

## 3.5. LL Implementation

Implementing the concept of Lifelong machine learning over the new dataset. First, we collect the dataset and do the similar data pre-processing as for the training dataset. Then we used the 'partial_fit' concept for the purpose of updating the model incrementally.



```
X_new = data.iloc[:, 0:-1].values
y_new = data.iloc[:, -1].values
y_new = y_new.reshape(-1,1)
print(X_new)
X_new = sc.fit_transform(X_new)
y_new = sc.fit_transform(y_new)
for j in range(len(X_new)):
    x_i = X_new[i].reshape(1, -1)
    y_i = y_new[i]
    # if(i==3 and data['month']==1):
    #     break
    regressor.partial_fit(x_i, [y_i])
```

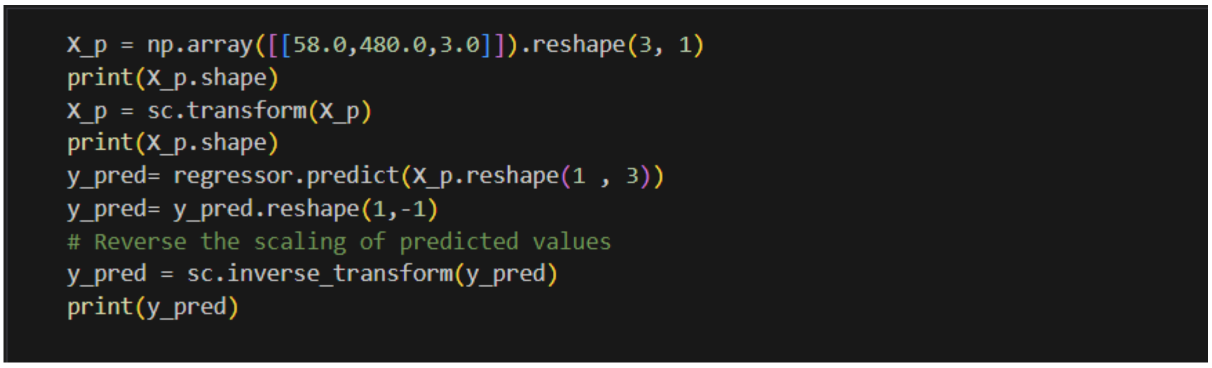
Partial fit is a concept in lifelong learning that refers to the process of updating or refining a model incrementally over time as new data becomes available. In other words, it is a way to adapt a model to changing circumstances without starting from scratch every time. Partial fit is a key aspect of lifelong learning because it allows a model to be updated as new data becomes available, rather than requiring the model to be retrained from scratch. This can save time and resources, and also enable the model to adapt more quickly to changing circumstances.

# Chapter 4

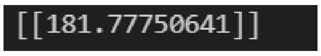
# Results and Accuracy Calculations

## 4.1. Value Prediction

### a) Value prediction for the data before implementation of lifelong learning

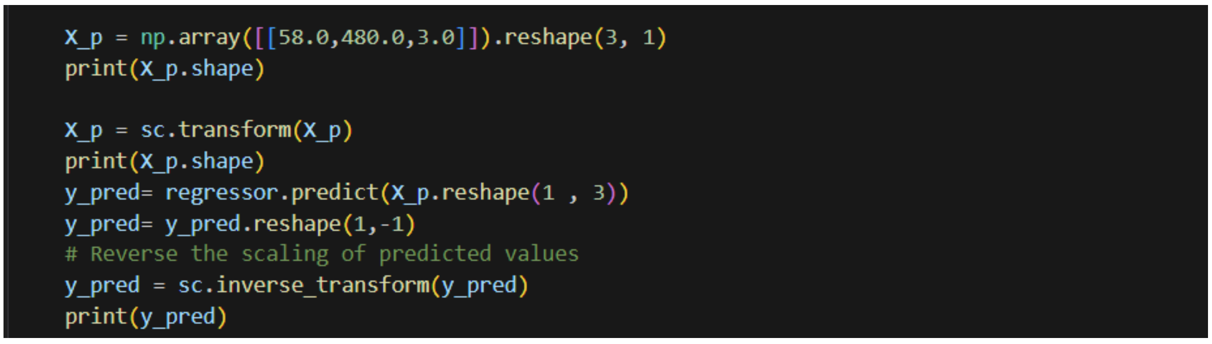


```
X_p = np.array([[58.0,480.0,3.0]]).reshape(3, 1)
print(X_p.shape)
X_p = sc.transform(X_p)
print(X_p.shape)
y_pred= regressor.predict(X_p.reshape(1 , 3))
y_pred= y_pred.reshape(1,-1)
# Reverse the scaling of predicted values
y_pred = sc.inverse_transform(y_pred)
print(y_pred)
```

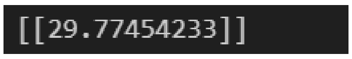


```
[[181.77750641]]
```

### b) Value prediction for the data after implementation of lifelong learning.



```
X_p = np.array([[58.0,480.0,3.0]]).reshape(3, 1)
print(X_p.shape)

X_p = sc.transform(X_p)
print(X_p.shape)
y_pred= regressor.predict(X_p.reshape(1 , 3))
y_pred= y_pred.reshape(1,-1)
# Reverse the scaling of predicted values
y_pred = sc.inverse_transform(y_pred)
print(y_pred)
```

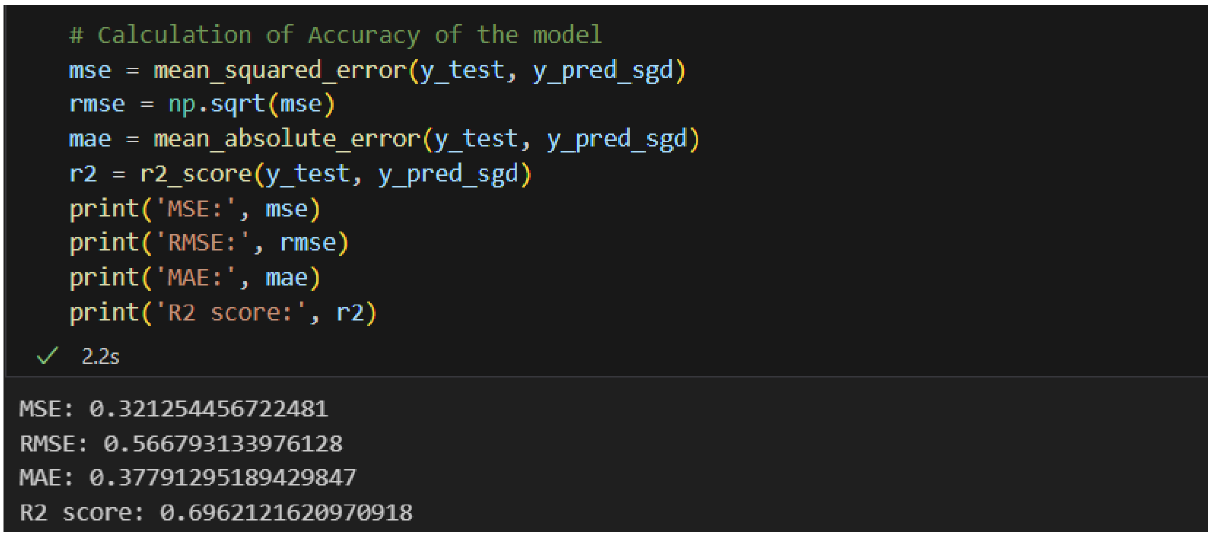


```
[[29.77454233]]
```

In our model, predicted values for the same data are:

- Before lifelong learning –> 181.78
- After lifelong learning –> 29.77

## 4.2. Accuracy Calculation



```
# Calculation of Accuracy of the model
mse = mean_squared_error(y_test, y_pred_sgd)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred_sgd)
r2 = r2_score(y_test, y_pred_sgd)
print('MSE:', mse)
print('RMSE:', rmse)
print('MAE:', mae)
print('R2 score:', r2)
```

✓ 2.2s

```
MSE: 0.321254456722481
RMSE: 0.566793133976128
MAE: 0.37791295189429847
R2 score: 0.6962121620970918
```

In SGD Regression (Stochastic Gradient Descent Regression), several evaluation metrics can be used to assess the accuracy of the model. Here are the commonly used metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) score.

1. Mean Squared Error (MSE):
   - The MSE measures the average squared difference between the predicted values and the true values. It gives higher weights to larger errors.
   - Lower MSE values indicate better accuracy.
2. Root Mean Squared Error (RMSE):
   - The RMSE is the square root of the MSE and provides an interpretable measure of the average prediction error.
   - Like MSE, lower RMSE values indicate better accuracy.
3. Mean Absolute Error (MAE):
   - The MAE measures the average absolute difference between the predicted values and the true values. It provides a more interpretable measure of average prediction error.
   - Lower MAE values indicate better accuracy.
4. R-squared (R2) Score:
   - The R2 score represents the proportion of the variance in the dependent variable that is predictable from the independent variables.
   - R2 score ranges from 0 to 1, where 0 indicates the model explains none of the variance and 1 indicates a perfect fit.
   - Higher R2 scores indicate better accuracy.

Hence, ACCURACY = 67.28 %

## <u>REFERENCES</u>

- Chen, Zhiyuan & Liu, Bing. (2016). Lifelong Machine Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. 10. 1-145. 10.2200/S00737ED1V01Y201610AIM033.
- Source for the Dataset used for training the model: 'https://archive.ics.uci.edu/ml/datasets/Diabetes'