# Group Number: 51
# Roll Number(s) : 21CS10071 Tirzah Grace.Yarranna
# Project Code: DPSVM1
# Project Title: Diabetes Prediction using Support Vector Machines

Diabetes Prediction using SVMs

Overview:

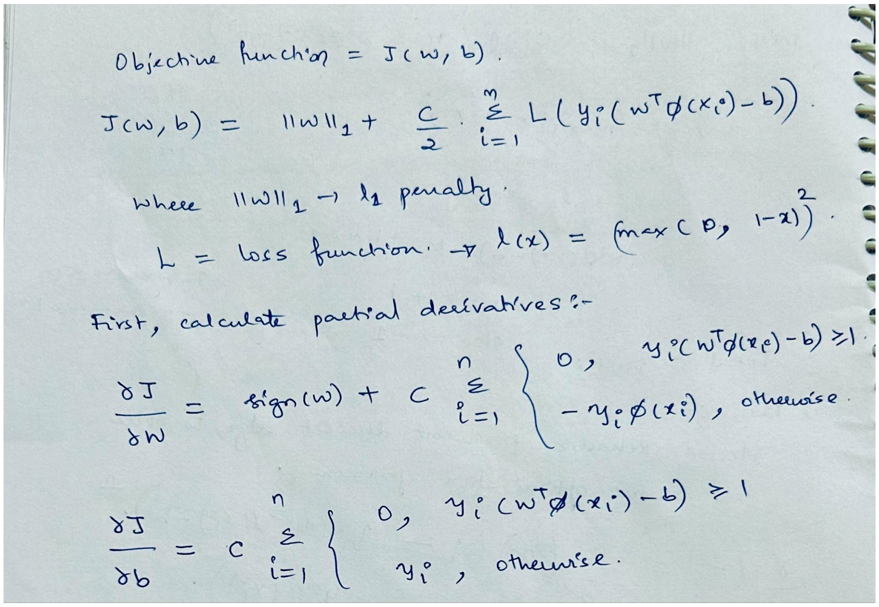
In this project, the aim is to predict the likelihood of a patient having diabetes using Support Vector Machines (SVMs).

Steps:

1. **Test-Train split:**
   - The dataset is randomly split into a training set (80%) and a test set (20%) to train and evaluate the SVM model.

2. **Data pre_processing: (pre_process.py)**
   - standardized the features by computing the mean and standard deviation on the training set and applying the same transformation to the test set.

3. **Implementation of the Model:**
   a. Built **Stochastic Gradient Descent algorithm** for solving the optimization problem.



Objective function = $J(w, b)$.

$$J(w,b) = \|w\|_1 + \frac{C}{2} \cdot \sum_{i=1}^{m} L(y_i(w^T\phi(x_i)-b)).$$

where $\|w\|_1 \to l_1$ penalty.

L = loss function. $\to l(x) = (\max(0, 1-x))^2$.

First, calculate partial derivatives:-

$$\frac{\partial J}{\partial w} = \text{sign}(w) + C \sum_{i=1}^{n} \begin{cases} 0, & y_i(w^T\phi(x_i)-b) \geq 1. \\ -y_i\phi(x_i), & \text{otherwise.} \end{cases}$$

$$\frac{\partial J}{\partial b} = C \sum_{i=1}^{n} \begin{cases} 0, & y_i(w^T\phi(x_i)-b) \geq 1 \\ y_i, & \text{otherwise.} \end{cases}$$

```
Gradient Descent algorithm:
Until max_iterations or improvement in objective
function is less than tolerance
      Compute gradients/partial derivatives
      Update w and b, based on learning rate, partial
      Derivative calculated
```

b. **SVM** class

Defined an SVM class encapsulating model fitting, gradient computation, and prediction functionalities.

i. __init__(C, learning_rate, tolerance, max_iterations): Constructor to initialize the SVM model with specified hyperparameters.
ii. fit(X, y): Method to train the SVM model using stochastic gradient descent.
iii. _objective_function(X, y): Method to compute the objective function value.
iv. predict(X): Method to make predictions using the trained model.
v. _compute_gradients(X, y): Method to compute gradients of the objective function.

c. **Hyper parameter** tuning using **Cross_Validation**

Hyper parameters are : C, learning_rate.

cross_validation(X_train_normalized, y_train, n_splits): Function to perform cross-validation for hyperparameter tuning.

i. Split the train set into train and validation folds.
ii. Iterate over different combinations of hyperparameters (C and learning_rate).
iii. Train SVM models with each combination and evaluate on the validation set.
iv. Select the best hyperparameters based on validation performance.

Result obtained:

Best C: 10
Best Learning rate: 0.01

d. **Trained** the model using Best parameters

Result obtained:

```
Final weights (w):  [2.7843161  7.28963386 0.65537159
4.74889977 4.8942154  7.7529929
 3.40641769 4.20862025]
Final bias (b):  0.791
```

e. **Evaluated** the model performance on **test_split.**

4. **Compare the results:**
    a. Evaluated the sklearn module **sklearn.svm.LinearSVC**
    b. Compared the **Classification reports** of both the models.

Results of the comparision.

```
Results of the SVM Model Implemented:
              precision    recall  f1-score   support

           0       0.83      0.66      0.73        99
           1       0.55      0.76      0.64        55

    accuracy                           0.69       154
   macro avg       0.69      0.71      0.69       154
weighted avg       0.73      0.69      0.70       154

Results of the Inbuilt Linear SVC:
              precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154
```

This classification report says:

- The accuracy of the Inbuilt Linear SVC model (**0.75**) is slightly higher than that of the implemented SVM model (**0.69**).
- The precision for class 0 is similar between both models, but the precision for class 1 is slightly higher for the Inbuilt Linear SVC model.
- The recall for class 0 is slightly higher for the Inbuilt Linear SVC model, while the recall for class 1 is slightly higher for the implemented SVM model.

- The F1-score for class 0 is similar between both models, but the F1-score for class 1 is slightly higher for the Inbuilt Linear SVC model.

Overall, both models perform reasonably well, but the Inbuilt Linear SVC model exhibits slightly better performance in terms of accuracy and some other metrics.