Heart Disease Prediction using Support Vector Machines

❖ Introduction: -

Created a model to predict whether he/she is suffering from a heart disease. A customized function trains the model. This is a binary classification problem, the labels are either +1 or -1. The customized function is

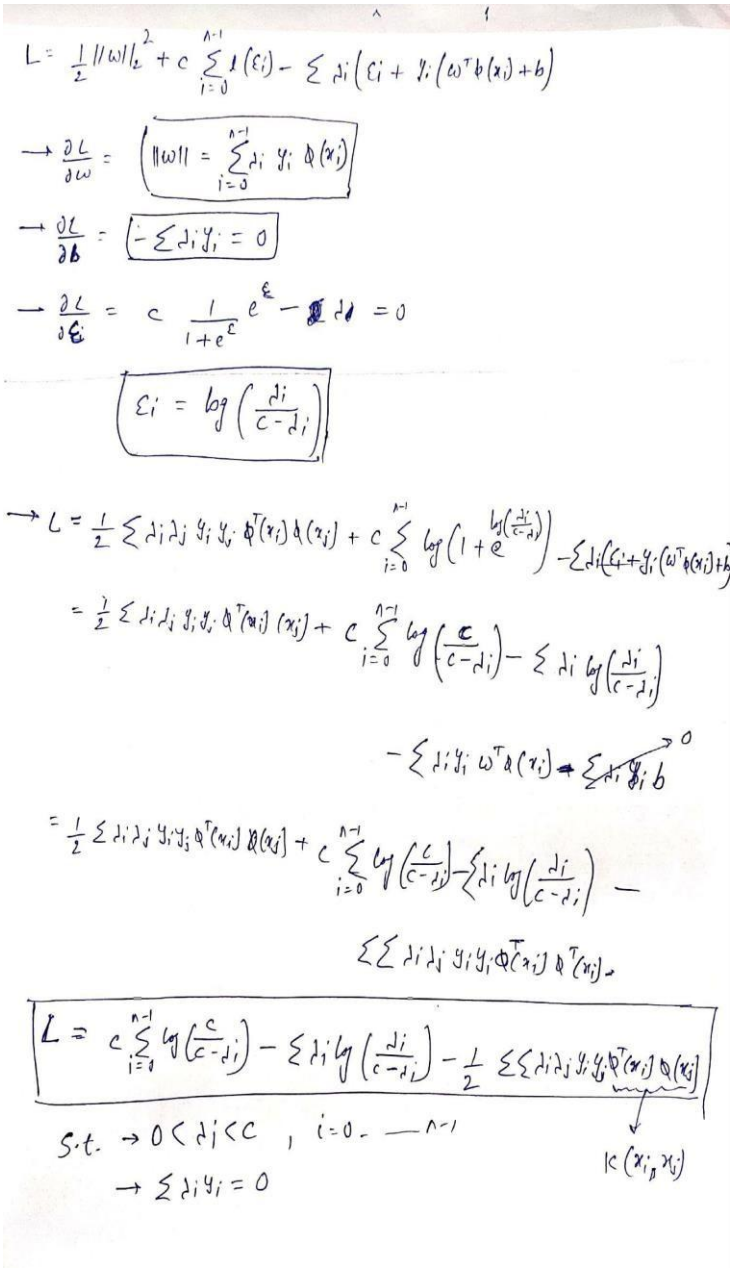
$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=0}^{n-1} l(\xi_i)$$
$$\text{subject to} \quad \xi_i = -y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)$$

where $l(x) = \log(1 + e^x)$ and

$$\{(\mathbf{x}_0, y_0), (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n-1}, y_{n-1})\}$$

is the training set.

First, we need to write down the Lagrangian of the optimization problem and KKT condition from above. Then we have to formulate the dual optimization problem. The derivation is mentioned below:



This equation is used to train the model.

- **Explanation:**
  - First, we finalise the optimisation problem as shown below.

```
def objective(self, alpha):
    return 0.5 * np.dot(alpha, np.dot(self.H, alpha) + np.sum(alpha*np.log(alpha/(self.C-alpha))) - self.C*np.sum(np.log(self.C/(self.C-alpha))) )
```

  - The constraints are defined as follows:

```
# Define the constraints
constraints = [{'type': 'ineq', 'fun': constraint1},
               {'type': 'eq', 'fun': constraint2, 'args': (Y,)}]
```

  - Three functions are declared each for linear, polynomial, and rbf.

```
def linear(x1, x2, p=1):
    return np.dot(x1, x2)

def polynomial(x1, x2, p=2):
    return (1 + np.dot(x1, x2)) ** p

def rbf(x1, x2, sigma=100, p=None):
    return np.exp(-linalg.norm(x1-x2)**2 / (2 * (sigma ** 2)))
```

  - **Data pre-processing:-** Before training the dataset is normalised using the formula below with zero mean and unit variance. The test data is normalised using the mean and variance computed over train data.

$$x' = \frac{x - \bar{x}}{\sigma}$$

```
# normalize test data
test_data[numerical_features] = (test_data[numerical_features] - means) / stds
```

  - **Train-test split:-** The target variable is coded with +1 and -1 as SVM classify points as either +1 or -1. Then data is shuffled before splitting into training and testing (80:20).

```
dataset['target'] = dataset['target'].replace(0, -1)
shuffled_data= dataset.sample(frac=1)
train_data_num = int(shuffled_data.shape[0]*0.8)
train_data = shuffled_data.iloc[:train_data_num]
test_data = shuffled_data.iloc[train_data_num:]
```

- Hessian matrix is created to solve the above equation.

```
# Evaluate Hessian Matrix depending on the kernel
self.X = X
self.Y = Y
H = np.zeros((X.shape[0],X.shape[0]))
for i in range(X.shape[0]):
    for j in range(X.shape[0]):
        H[i,j] = self.kernel(X[i], X[j], p=self.p)*Y[i]*Y[j]
self.H = H
```

- Using Scipy.optimize.minimise (python package), solved the above equation.

```
# Solve the SVM using scipy.optimize.minimize
init_alpha = np.ones(X.shape[0])*self.C*0.5
solution = minimize(self.objective, init_alpha, method='SLSQP', bounds=bounds, constraints=constraints)

# Extract the Lagrange multipliers from the solution
alpha = solution.x
```

- We have taken 5 cross-validation method for Hyperparameter tuning.

```
folds = cross_validation_split(train_data, 5)
```

- Through Hyperparameter tuning, we get the value of C value for which accuracy is highest.

```
print("Hyperparameter tuning for C ....")
for c in C_List:
    accuracy = 0
    print("Testing for C = " + str(c))
    for i in range(5):
        print("Fold "+str(i+1)+"/5")
        train_set = pd.concat(folds[j] for j in range(5) if j!=i)
        X_train_set = train_set.drop('target', axis=1)
        Y_train_set = train_set['target']
        X_validation_set = folds[i].drop('target', axis=1)
        Y_validation_set = folds[i]['target']
        myLinearSVM = SVM(kernel=linear, C=c)
        myLinearSVM.Train(X_train_set.to_numpy(),Y_train_set.to_numpy())
        accuracy += myLinearSVM.Predict(X_validation_set.to_numpy(), Y_validation_set.to_numpy(),False)
```

- And likewise, we get the values of P and sigma for maximum accuracy.

- By using the best hyperparameter values(as calculated in the last step), we trained the model by using training data and predicted using predict data (given below is for the linear kernel, likewise, we will get for polynomial and rbf kernel.)

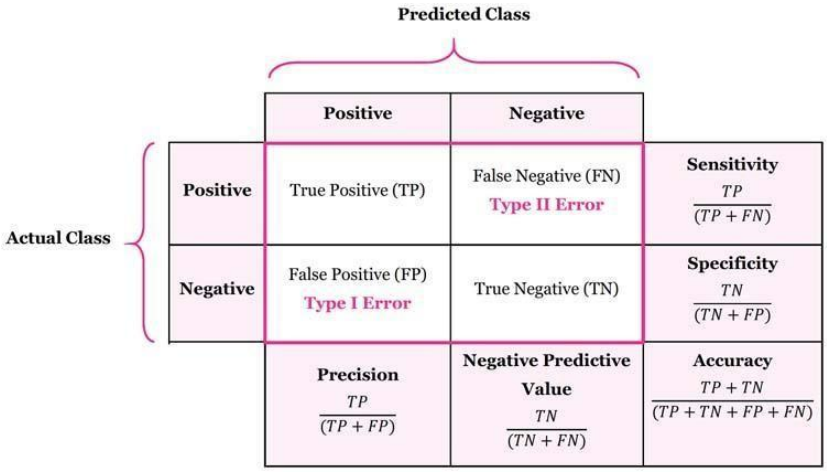
```
# Linear Kernel
print("Training Linear Kernel SVM...")
myLinearSVM = SVM(kernel=linear, C=best_C)
myLinearSVM.Train(X_train.to_numpy(),Y_train.to_numpy())
myLinearSVM.Predict(X_test.to_numpy(), Y_test.to_numpy(),doPrint=True)
```

- Also we trained the model using the Sklearn library and compare the accuracy with the model built from scratch.

```
# SVM Using Library function
print("SVM using Sklearn...")
svclassifier = LinearSVC()
svclassifier.fit(X_train, Y_train)
Y_pred = svclassifier.predict(X_test)

print(classification_report(Y_test,Y_pred))
```

- For the confusion matrix, True Positive(t_p), True Negative(t_n), False Positive(f_p), False Negative(f_n). (Print_Result). These values are calculated using simple if-else Conditions.



- The training dataset is used for different kernel functions to build a different model. The testing dataset is used in this different model and calculates precision, recall, f1-score and accuracy for all the experiments.

❖ Outcomes:-

1. Mathematical derivation of the equation is stated above.
2. Results of the Hyper-parameter tuning is as given below:-

```
best_C = 100
best_P = 2
best_S = 2.5
```

3. Results on the test split of the dataset:-

```
Training Linear Kernel SVM...
              precision    recall  f1-score   support

    neg (-1)       0.83      0.77      0.80       110
    pos (1)        0.76      0.82      0.79        95

    accuracy                           0.80       205

Training Polynomial Kernel SVM...
              precision    recall  f1-score   support

    neg (-1)       0.92      0.75      0.83       110
    pos (1)        0.77      0.93      0.84        95

    accuracy                           0.83       205

Training RBF Kernel SVM...
              precision    recall  f1-score   support

    neg (-1)       0.87      0.97      0.92       110
    pos (1)        0.96      0.83      0.89        95

    accuracy                           0.91       205
```

- Results obtained after using the Sklearn library.

```
SVM using Sklearn...
              precision    recall  f1-score   support

          -1       0.88      0.79      0.83       110
           1       0.78      0.87      0.83        95

    accuracy                           0.83       205
   macro avg       0.83      0.83      0.83       205
weighted avg       0.83      0.83      0.83       205
```