Assignment-1C: Linear Perceptron

1. Description and Implementation:

The perceptron is an algorithm for supervised learning of binary classifiers. It has many inputs (often called features) that are fed into a Linear unit that produces one binary output. Therefore, perceptron can be applied in solving Binary Classification problems where the sample is to be identified as belonging to one of the predefined two classes.

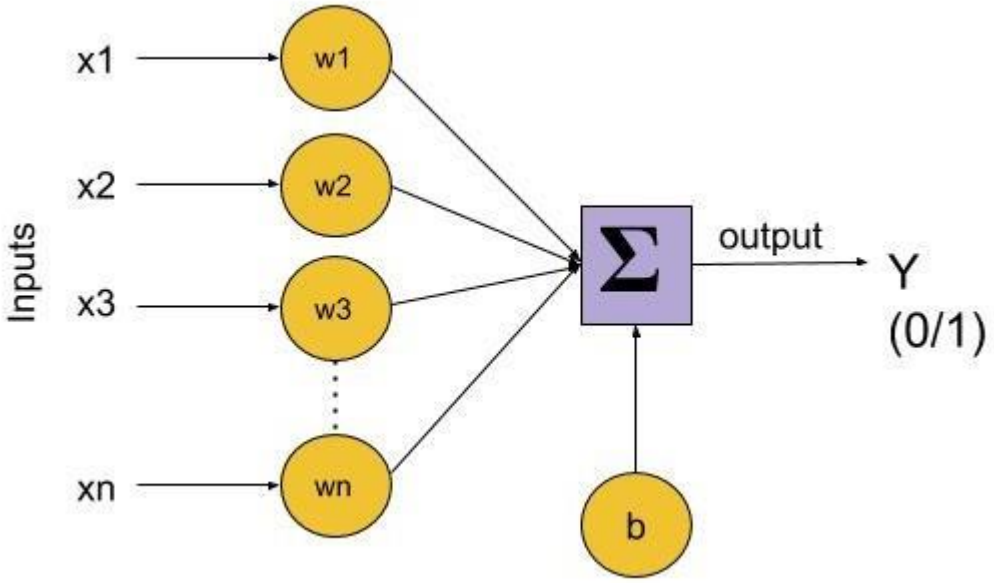


Fig.: Schematic of Perceptron

Since Perceptrons are Binary Classifiers (0/1), we can define their computation as follows:

$$f_\Phi(\mathbf{x}) = \begin{cases} 1 & \text{if } b + \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The dot product of two vectors of length n ($1 \le i \le n$) is $w.x = \sum_i w_i . x_i$

The function $f(x)=b+\boldsymbol{w}.\boldsymbol{x}$ is a linear combination of weight and feature vectors. Perceptron is, therefore, a linear classifier — an algorithm that predicts using a linear predictor function.

The weights signify the effectiveness of each feature x_i in **x** on the model's behaviour. Higher the weight w_i of a feature x_i, higher is its influence on the output. On the other hand, the bias 'b' is like the intercept in the linear equation. It's a constant that helps the model adjust in a way that best fits the data. The bias term assumes an imaginary input feature coefficient $x_o=1$.

Implementation:

- Reading the data and storing in data frames (df1 and df2). We have added bias here to ease the calculations.

```
In [2]: df1 = pd.read_csv('data/dataset_LP_1.txt', header = None)
        df2 = pd.read_csv('data/dataset_LP_2.csv', header = None)
        df1.insert(0, 'bias', 1)
        df2.insert(0, 'bias', 1)
        print(df1)
```

```
      bias        0         1        2        3  4
0        1  3.62160   8.66610  -2.8073 -0.44699  0
1        1  4.54590   8.16740  -2.4586 -1.46210  0
2        1  3.86600  -2.63830   1.9242  0.10645  0
3        1  3.45660   9.52280  -4.0112 -3.59440  0
4        1  0.32924  -4.45520   4.5718 -0.98880  0
...    ...      ...       ...      ...      ... ..
1367     1  0.40614   1.34920  -1.4501 -0.55949  1
1368     1 -1.38870  -4.87730   6.4774  0.34179  1
1369     1 -3.75030 -13.45860  17.5932 -2.77710  1
1370     1 -3.56370  -8.38270  12.3930 -1.28230  1
1371     1 -2.54190  -0.65804   2.6842  1.19520  1

[1372 rows x 6 columns]
```

- Splitting the data into train and test (ratio: 70 :30 respectively) and storing in X_train, Y_train, X_test and Y_test for both df1 and df2

```
X_train_1, Y_train_1, X_test_1, Y_test_1 = split(df1, ratio = 0.7)
X_train_2, Y_train_2, X_test_2, Y_test_2 = split(df2, ratio = 0.7)
```

```
def split(dataframe, ratio = 0.7):
    df_train = dataframe.sample(frac = ratio)
    df_test = dataframe.drop(df_train.index)

    train = df_train.to_numpy()
    test = df_test.to_numpy()

    X_train = train[:, :-1].reshape((-1, train.shape[1]-1))
    Y_train = train[:, -1].reshape((-1, 1))
    X_test = test[:, :-1].reshape((-1, test.shape[1]-1))
    Y_test = test[:, -1].reshape((-1, 1))

    return X_train, Y_train, X_test, Y_test
```

- Creating two objects of the Perceptron class for the two datasets.

```
perceptron1 = Perceptron()
perceptron2 = Perceptron()
```

In [5]:
```
class Perceptron:
    def __init__(self):
        self.w = None

    def model(self, X):
        return 1 if np.matmul(X, self.w) >= 0 else 0

    def predict(self, X):
        Y = [self.model(x) for x in X]
        return np.array(Y).reshape((-1, 1))

    def fit(self, X, Y, epochs = 1, alpha = 0.1):
        self.w = np.zeros(X.shape[1]).reshape((-1, 1))
        print(f'\nIntial w: {self.w.T}')
        accuracy = {}
        max_accuracy = 0
        max_weight = None
        wt_matrix = []

        for i in range(epochs):
            for x, y in zip(X, Y):
                x = x.reshape((1, -1))
                y = y.flatten()[0]
                y_pred = self.model(x)
                if y == 1 and y_pred == 0:
                    self.w = self.w + alpha * x.T
                elif y == 0 and y_pred == 1:
                    self.w = self.w - alpha * x.T

            wt_matrix.append(self.w)
            accuracy[i] = calc_accuracy(self.predict(X), Y)
            if accuracy[i] > max_accuracy:
                max_accuracy = accuracy[i]
                max_weight = self.w

        self.w = max_weight

        print(f'Max training Accuracy found: {max_accuracy}')
        plt.figure(figsize = (10,10))
        plt.plot(list(accuracy.keys()), list(accuracy.values()))
        plt.xlabel("Epoch Number")
        plt.ylabel("Accuracy")

        return self.w
```

- Finding the weight matrix using the train data and then predicting the class for the test data.

```
wt_matrix_1 = perceptron1.fit(X_train_1, Y_train_1, epochs = 1000, alpha = 0.1)
Y_pred_test_1 = perceptron1.predict(X_test_1)

wt_matrix_2 = perceptron2.fit(X_train_2, Y_train_2, epochs = 1000, alpha = 0.1)
Y_pred_test_2 = perceptron2.predict(X_test_2)
```

We have used two methods fit and predict. Fit method accepts training data and returns the final weight matrix corresponding to each feature.
Predict method accepts the testing data and predicts the class for that data.

- Finally calculating the accuracy.

```
print(f'\nTesting accuracy for dataset 1: {calc_accuracy(Y_pred_test_1, Y_test_1)}')
print(f'Best W found for dataset 1: {wt_matrix_1.T}')

print(f'\nTesting accuracy for dataset 2: {calc_accuracy(Y_pred_test_2, Y_test_2)}')
print(f'Best W found for dataset 2: {wt_matrix_2.T}')
```

Calc_accuracy accepts predicted class array and original class array and returns the accuracy of the model.
Accuracy = (true positive + true negative)/(true positive + true negative + false positive +false negative)

2. Accuracy of your model on both the datasets:

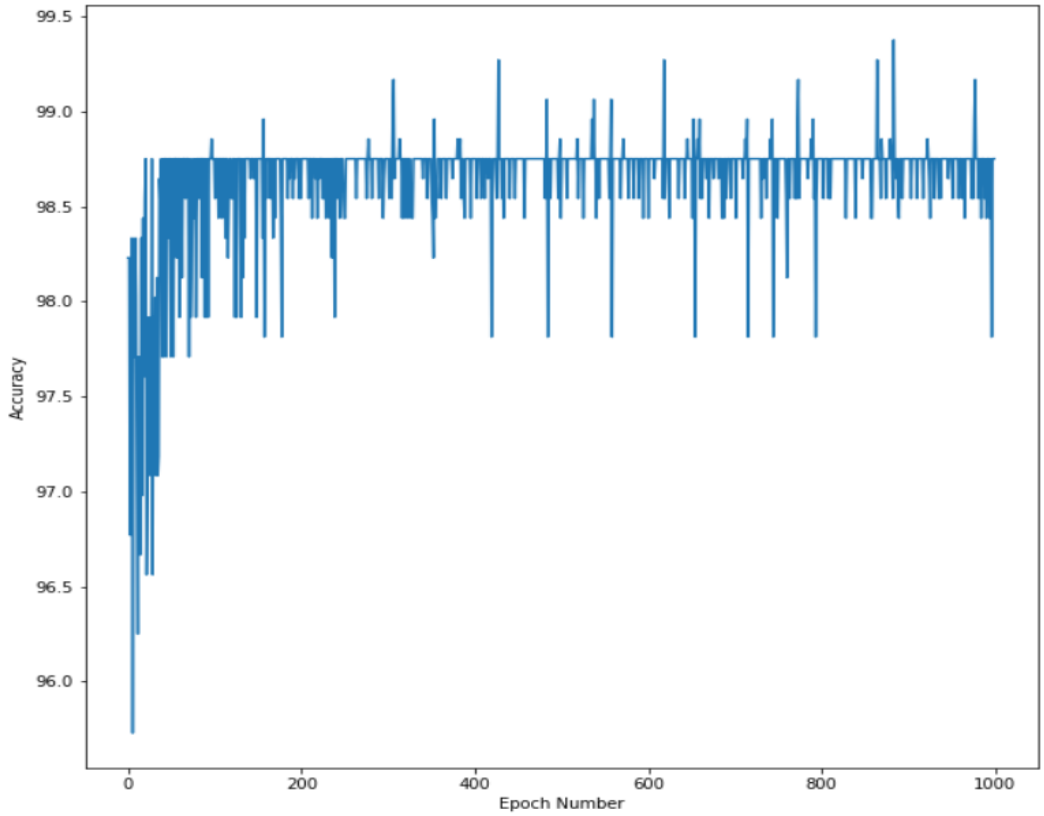
The max accuracy for my model for ‘dataset_LP_1.txt’ was 99.375% and for ‘dataset_LP_2.csv’ was 100%.

```
Intial w: [[0. 0. 0. 0. 0.]]
Max training Accuracy found: 99.375

Intial w: [[0. 0. 0. 0.]]
Max training Accuracy found: 100.0

Testing accuracy for dataset 1: 99.51456310679612
Best W found for dataset 1: [[ 15.6        -15.41178525  -8.032627   -10.3509016   -0.9373197 ]]

Testing accuracy for dataset 2: 100.0
Best W found for dataset 2: [[ 0.6        -0.07682335  0.52981757  3.08569756]]
```

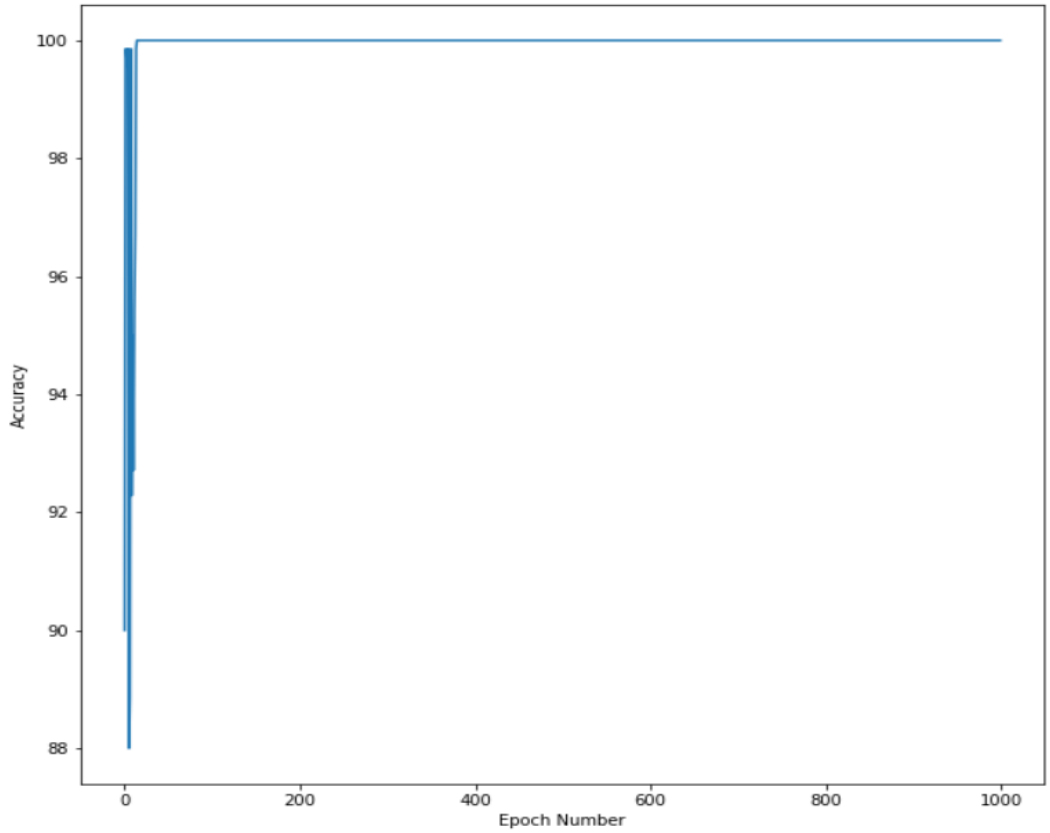
3. Dataset which was more linearly separable:

The dataset 2(‘dataset_LP_2.csv’) was more linearly separable as we got maximum accuracy of 100% for this. Moreover, the accuracy saturates at 100% after few epochs for dataset 2.
Whereas in dataset 1 (‘dataset_LP_1.txt’) the maximum accuracy was 99.375% and here the accuracy saturates around 98.75% after few epochs.

For dataset 1:



For dataset 2:



4. <u>Major limitations of the Perceptron classifier:</u>

1. A single-layer perceptron works only if the dataset is linearly separable.
2. The algorithm is used only for Binary Classification problems. However, we can extend the algorithm to solve a multiclass classification problem by introducing one perceptron per class. i.e., each perceptron results in a 0 or 1 signifying whether or not the sample belongs to that class.