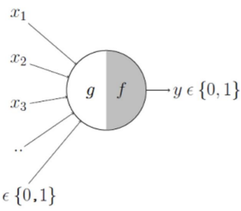
Build a single layer perceptron model from scratch

Author: Rajpal Virk | 10 January 2020

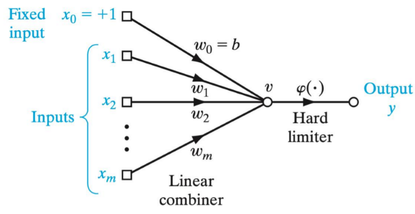
Introduction

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. source

McCulloch-Pitts model is one of the earliest computational models of a neuron. Model was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943.
Below is the simple representation of single layer perceptron. In this representation, perceptron has 2 parts.



First part 'g' takes all the input data along with bias and using a weight vector, it aggregates the input. This aggregated input is then passed through the second part 'f', which is an activation function. This activation function then decides the classification on this input aggregated data.
McCulloch-Pitts model uses hard limiter as the activation function and gives out the linear output in form of binary classification. It is tempting to think that will McCulloch-Pitts model perform better if we use a sigmoidal function rather hard limiter. It turns out that the steady-state, decision-making characteristics of a singlelayer perceptron are basically the same, regardless of whether we use hard-limiting or soft-limiting (differentiable nonlinear transfer function) as the source of nonlinearity in the neural model (Shynk and Bershad, 1991, 1992; Shynk, 1990). This results in a hyperplane separating linearly separable classes.



In the above figure, we have inputs x0, x1, x2.....xm, where x0 is fixed input equal to 1. w1, w2.....wm are the weights and w0 is the bias.
The weighted inputs are fed in single neuron. The resultant is the activation potential. We apply activation function on activation potential. In this case activation potential is hard limiter, which gives the output in form of either 0 or 1.

Import Required Libraries

In [1]:
```
# import required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Generate linearly separable data

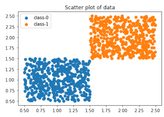
In [2]:
```
# Generate data
## Generate uniformly distributed data for class-0 with ranging from 0.5 to 1.5.
df_01_col_01 = np.random.uniform(low=0.5, high=1.5, size=500)
df_01_col_02 = np.random.uniform(low=0.5, high=1.5, size=500)
class_0 = np.full((500), 1)
df_01 = np.transpose(np.vstack((df_01_col_01, df_01_col_02, class_0)))

## Generate uniformly distributed data for class-1 with ranging from 1.5 to 2.5.
df_02_col_01 = np.random.uniform(low=1.5, high=2.5, size=500)
df_02_col_02 = np.random.uniform(low=1.5, high=2.5, size=500)
class_1 = np.full((500), 0)
df_02 = np.transpose(np.vstack((df_02_col_01, df_02_col_02, class_1)))

# Merging data of 2-classes to 1 dataframe
df = np.vstack((df_01, df_02))
print('First 5 and Last 5 rows of data:')
print(df[:5])
print(df[-5:])
print()
print()

# Visualize data
plt.scatter(df[:500, 0], df[:500, 1], label='class-0')
plt.scatter(df[-500:, 0], df[-500:, 1], label='class-1')
plt.title('Scatter plot of data')
plt.legend()
plt.show()
```

```
First 5 and Last 5 rows of data:
[[0.99582591 1.21417103 1.        ]
 [0.72238305 0.57544309 1.        ]
 [1.30812867 1.1544639  1.        ]
 [0.52232126 0.83738533 1.        ]
 [1.35753355 0.7115244  1.        ]]
[[2.28052756 1.53223281 0.        ]
 [1.69801212 2.37453285 0.        ]
 [2.23321674 2.38737553 0.        ]
 [2.36555908 2.04648175 0.        ]
 [1.89559348 1.75812982 0.        ]]
```



Data Pre-processing

In [3]:
```
# Splitting data in train and test data
X= df[:, :-1] # first 2 rows
y = df[:,-1] # last row
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Build a Single layer perceptron model

In [4]:
```
# Build a class object of SLP
class SLP(object):
    def __init__(self, lr, n_epochs): # lr = Learning Rate, n_epochs = number of epochs
        """ Input values of Learning Rate and No. of Epochs """
        self.lr = lr
        self.epochs = n_epochs

    def net_input_data(self, X):
        """ weighted summations of input values, weight and bias """
        return (np.dot(X, self.w) + self.b) # w = synaptic weight, b = bias

    def predict(self, X):
        """ Apply hard limiter to convert predictions in 0 and 1 """
        return (np.where(self.net_input_data(X) >= 0.5, 1, 0))

    def fit(self, X, y):
        """ Fit function on train data using weight = 0 and bias = 1 """
        self.w = np.zeros(X.shape[1]) # size = no. of input features (columns)
        self.b = float(1) # fixed float value = 1
        self.errors = []
        for _ in range (self.epochs):
            errors = 0
            for X_idx, desired in zip(X, y):
                diff = (desired - self.predict(X_idx))
                delta = self.lr * (desired - self.predict(X_idx))
                self.w += delta * X_idx
                errors += int(diff != 0.0)
            self.errors.append(errors)
        return self
```

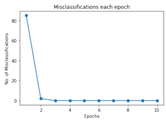
Train Model

In [5]:
```
# Setting hyper parameters
lr = 0.001
n_epochs = 10

# Build and train model
model = SLP(lr = lr, n_epochs = n_epochs)
model.fit(X_train, y_train)
print('Misclassifications each epoch: ', model.errors)
print()
print()

# Plot misclassifications each epoch
x_lim = np.arange(1, n_epochs+1)
y_lim = model.errors
marker = 'o'
plt.plot(x_lim, y_lim, marker = marker)
plt.title('Misclassifications each epoch')
plt.xlabel('Epochs')
plt.ylabel('No. of Misclassifications')
plt.show()
```

```
Misclassifications each epoch:  [85, 2, 0, 0, 0, 0, 0, 0, 0, 0]
```



Evaluate Model

In [6]:
```
y_pred = model.predict(X_test)
print('Model Accuracy is:', accuracy_score(y_true=y_test, y_pred=y_pred))
print()
print ('############ Confusion Matrix ############')
print('Confusion Matrix: \n', confusion_matrix(y_true=y_test, y_pred=y_pred))
print()
print ('############ Classification Report ############')
print('Classification_report: \n', classification_report(y_true=y_test, y_pred=y_pred))
```

```
Model Accuracy is: 1.0

############ Confusion Matrix ############
Confusion Matrix:
 [[102   0]
 [  0  98]]

############ Classification Report ############
Classification_report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       102
         1.0       1.00      1.00      1.00        98

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Above statistics indicate that our model is able to predict the binary classification with 100% accuracy.

Plot decision boundary

In [7]:
```
# Define plot function
def decision_boundary(X, y, classifier, resolution = 0.02):
    """
    In this function, we pass, X and y data along with model to eventually test whether our model
    is able to draw a precise decision boundry to separate 2 classes or not.
    """



#     cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:,0].min(), X[:, 0].max()
    x2_min, x2_max = X[:,1].min(), X[:, 1].max()

    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))


    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    plt.figure()
    plt.contourf(xx1, xx2, Z, alpha = 0.5)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x = X[y == cl, 0], y = X[y == cl, 1],alpha = 0.5)

    plt.xlabel("Input 1")
    plt.ylabel("Input 2")
    plt.title("Decision boundary separating 2 classes")
    plt.show()

# Plot a decision boundary for whole data
X= df[:, :-1] # first 2 rows
y = df[:,-1] # last row
decision_boundary(X, y, classifier=model, resolution=0.02)
```