Classification using Multilayer Perceptron to solve XOR problem

17BCE0136 R.S.Rahul Sai

In [1]:
```python
import numpy as np

def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
def sigmoid_prime(x):
    return sigmoid(x)*(1.0-sigmoid(x))
def tanh(x):
    return np.tanh(x)
def tanh_prime(x):
    return 1.0 - x**2
```

In [2]:
```python
class MLP:

    def __init__(self, layers, activation='tanh'):
        if activation == 'sigmoid':
            self.activation = sigmoid
            self.activation_prime = sigmoid_prime
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_prime = tanh_prime

        # Set weights
        self.weights = []

        for i in range(1, len(layers) - 1):
            r = 2*np.random.random((layers[i-1] + 1, layers[i] + 1)) -1
            self.weights.append(r)

        r = 2*np.random.random( (layers[i] + 1, layers[i+1])) - 1
        self.weights.append(r)

    def fit(self, X, y, learning_rate=0.2, epochs=100000):
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)

        for k in range(epochs):
            if k % 10000 == 0: print('epochs:', k)

            i = np.random.randint(X.shape[0])
            a = [X[i]]

            for l in range(len(self.weights)):
                dot_value = np.dot(a[l], self.weights[l])
                activation = self.activation(dot_value)
                a.append(activation)

            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]

            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_prime(a[l]))

            deltas.reverse()

            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

    def predict(self, x):
        a = np.hstack((np.ones(1).T,np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return np.asscalar(a)
```

In [3]:
```python
nn = MLP([2,2,1])

## defining the data
X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
Y = np.array([0, 1, 1, 0])

## Fitting the MLP
nn.fit(X, Y)
```

```
epochs: 0
epochs: 10000
epochs: 20000
epochs: 30000
epochs: 40000
epochs: 50000
epochs: 60000
epochs: 70000
epochs: 80000
epochs: 90000
```

In [4]:
```python
for e in X:
    print("X= %d Y=%d Prediction=%f" %(e[0],e[1],nn.predict(e)))
```

```
X= 0 Y=0 Prediction=0.000044
X= 0 Y=1 Prediction=0.996016
X= 1 Y=0 Prediction=0.997521
X= 1 Y=1 Prediction=0.000021
```

Classification of Autism Spectrum Disorder Cases

17BCE0136 R.S.Rahul Sai

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
from scipy.io import arff
import pandas as pd
import numpy as np
```

Description of dataset

Autistic Spectrum Disorder (ASD) is a neurodevelopment condition associated with significant healthcare costs, and early diagnosis can significantly reduce these. Unfortunately, waiting times for an ASD diagnosis are lengthy and procedures are not cost effective.

Hence, this dataset related to autism screening of children contains 20 features to be utilised for further analysis especially in determining influential autistic traits and improving the classification of ASD cases.

In [2]:
```
data = arff.loadarff('Autism-Child-Data.arff')
df = pd.DataFrame(data[0])
```

Preprocessing the data

In [3]:
```
def decodeStr(x):
    if type(x) is not float and str:
        return x.decode()
    else:
        return float(x)

df = df.applymap(decodeStr)
```

In [4]:
```
df=df.replace([np.inf, -np.inf,"?"], np.nan).dropna()
df=df.reset_index(drop=True)
```

In [5]:
```
df=df.drop(columns=['age_desc','relation','used_app_before','result'])
```

In [6]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
le=LabelEncoder()
```

In [7]:
```
# Encoding the data
dfX=df.iloc[:,:-1]
dfY=df.iloc[:,-1]
dfY=pd.DataFrame(le.fit_transform(dfY).reshape(-1,1))
```

In [8]:
```
dfX['gender']=le.fit_transform(dfX['gender'])
dfX['jundice']=le.fit_transform(dfX['jundice'])
dfX['austim']=le.fit_transform(dfX['austim'])
dfX=pd.get_dummies(dfX,columns=['ethnicity'],drop_first=True)
dfX=pd.get_dummies(dfX,columns=['contry_of_res'],drop_first=True)
```

In [9]:
```
X_train,X_test,Y_train,Y_test=train_test_split(dfX,dfY,test_size=0.3,random_state=0)
```

SVM Classifier (linear)

In [10]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
svc=SVC(kernel='linear')
```

In [11]:
```
svc.fit(X_train,Y_train)
Y_predict=svc.predict(X_test)
```

In [12]:
```
print(classification_report(Y_test,Y_predict))
print("Accuracy = {:.2f}".format(svc.score(X_test, Y_test.values)*100))
```

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.93        42
           1       0.89      0.94      0.91        33

    accuracy                           0.92        75
   macro avg       0.92      0.92      0.92        75
weighted avg       0.92      0.92      0.92        75

Accuracy = 92.00
```

SVM Classifier (RBF)

In [13]:
```
svc=SVC(kernel='rbf')
svc.fit(X_train,Y_train)
Y_predict=svc.predict(X_test)
```

In [14]:
```
print(classification_report(Y_test,Y_predict))
print("Accuracy = {:.2f}".format(svc.score(X_test, Y_test.values)*100))
```

```
              precision    recall  f1-score   support

           0       0.97      0.81      0.88        42
           1       0.80      0.97      0.88        33

    accuracy                           0.88        75
   macro avg       0.89      0.89      0.88        75
weighted avg       0.90      0.88      0.88        75

Accuracy = 88.00
```