Department of Computer Engineering
University of Peradeniya

Machine Learning Lab Two **13-14/03/2024**

Objective

Implementing a linear classifier using the Perceptron algorithm.
The perceptron algorithm computes a linear classifier using a stochastic error correcting learning algorithm. It is simple and has much historic relevance to this subject and makes a good starting point to learn more sophisticated models and algorithms.

Implementation

Generate 100 samples each from two bi-variate Gaussian densities with distinct means $\boldsymbol{m}_1 = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ and $\boldsymbol{m}_2 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$, and identical covariance matrix $\boldsymbol{C} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. (Hint: Use material developed in Lab One).

```
NumDataPerClass = 200

# Two-class problem, distinct means, equal covariance matrices
#
m1 = [[0, 5]]
m2 = [[5, 0]]
C  = [[2, 1], [1, 2]]

# Set up the data by generating isotropic Guassians and
# rotating them accordingly
#
A = np.linalg.cholesky(C)

U1 = np.random.randn(NumDataPerClass,2)
X1 = U1 @ A.T + m1

U2 = np.random.randn(NumDataPerClass,2)
X2 = U2 @ A.T + m2
```

The distribution of your data should look like what is shown in Fig. 1. Note the data are linearly separable (*i.e.* a linear class boundary will classify the data correctly).

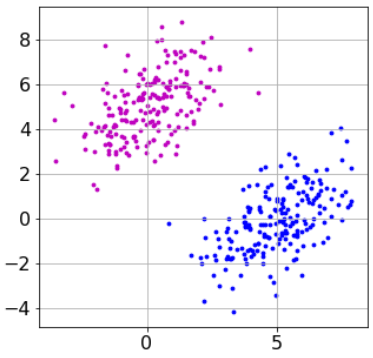

Figure 1: Training Data, sampled from two Bivariate Gaussian Densities

We will now train a perceptron algorithm to classify this data. A perceptron is a linear classifier whose training is done by error correction. If the weights of the perceptron are denoted $\boldsymbol{w}$ and the input features are in vector $\boldsymbol{x}$, a perceptron decision function assigns the data to one class or the other depending on whether $\boldsymbol{w}^T \boldsymbol{x} \lessgtr 0$.
The algorithm is as follows:

Inputs $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathcal{R}^d$, $y_n \in (-1, +1)$
Initialize weights $\boldsymbol{w}$
Generate index $0 \leq \tau \leq N$ at random
If $(\boldsymbol{w}^T \boldsymbol{x}^{(\tau)} \leq 0)$
. $\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \alpha\, y(\tau)\, \boldsymbol{x}^{(\tau)}$

Note the scalar product $\boldsymbol{w}^T \boldsymbol{x}^{(\tau)}$ times the target $y^{(\tau)}$ being positive over all the data is our goal. Upon seeing a random data, we only update if this data is misclassified. This is why we refer to this algorithm as a *stochastic error correcting* algorithm. Stochastic because we are lookign at random presentations of data and error correcting because we only update when the current example is misclassified.
We will derive this in a formal setting after we have studied regression, by setting up an error function and optimising it (minimising it) by gradient descent.
Here are snippets of code to help you do this.

1. For simplicity, I have assumed the following (some of which you are free to change and study the effect):
   - There is an equal number of data `NumDataPerClass` in each class
   - We use an equal partition of the data into training and test sets, taken at random.
2. Concatenate data from two classes into one array. (Fig. 1).

```
X = np.concatenate((X1, X2), axis=0)
```

3. Setting up targets (labels): we set $+1$ and -1 as labels to indicate the two classes.

```
labelPos = np.ones(NumDataPerClass)
labelNeg = -1.0 * np.ones(NumDataPerClass)
y = np.concatenate((labelPos, labelNeg))
```

4. Partitioning the data into training and test sets

```
rIndex = np.random.permutation(2*NumDataPerClass)
Xr = X[rIndex,]
yr = y[rIndex]

# Training and test sets (half half)
#
X_train = Xr[0:NumDataPerClass]
y_train = yr[0:NumDataPerClass]
X_test  = Xr[NumDataPerClass:2*NumDataPerClass]
y_test  = yr[NumDataPerClass:2*NumDataPerClass]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

Ntrain = NumDataPerClass;
Ntest  = NumDataPerClass;
```

5. Calculating the percentage of correctly classified examples

```
def PercentCorrect(Inputs, targets, weights):
    N = len(targets)
    nCorrect = 0
    for n in range(N):
        OneInput = Inputs[n,:]
        if (targets[n] * np.dot(OneInput, weights) > 0):
            nCorrect +=1
    return 100*nCorrect/N
```

6. Iterative error correcting learning

```
# Perceptron learning loop
#

# Random initialization of weights
#
w = np.random.randn(2)
print(w)

# What is the performance with the initial random weights?
#
print('Initial Percentage Correct: %6.2f' %(PercentCorrect(X_train, y_train, w)))

# Fixed number of iterations (think of better stopping criterion)
#
MaxIter=1000

# Learning rate (change this to see convergence changing)
#
alpha = 0.002

# Space to save answers for plotting
#
P_train = np.zeros(MaxIter)
P_test  = np.zeros(MaxIter)

# Main Loop
#
for iter in range(MaxIter):

    # Select a data item at random
    #
    r = np.floor(np.random.rand()*Ntrain).astype(int)
    x = X_train[r,:]

    # If it is misclassified, update weights
    #
    if (y_train[r] * np.dot(x, w) < 0):
        w += alpha * y_train[r] * x

    # Evaluate trainign and test performances for plotting
    #
    P_train[iter] = PercentCorrect(X_train, y_train, w);
    P_test[iter]  = PercentCorrect(X_test, y_test, w);

print('Percentage Correct After Training: %6.2f  %6.2f'
      %(PercentCorrect(X_train, y_train, w), PercentCorrect(X_test, y_test, w)))
```

7. Plot learning curves

```
fig, ax = plt.subplots(figsize=(6,4))
ax.plot(range(MaxIter), P_train, 'b', label = "Training")
ax.plot(range(MaxIter), P_test, 'r', label = "Test")
ax.grid(True)
ax.legend()
ax.set_title('Perceptron Learning')
ax.set_ylabel('Training and Test Accuracies', fontsize=14)
ax.set_xlabel('Iteration', fontsize=14)
plt.savefig('learningCurves.png')
```

The expected results of training a perceptron might look similar to Fig. 2

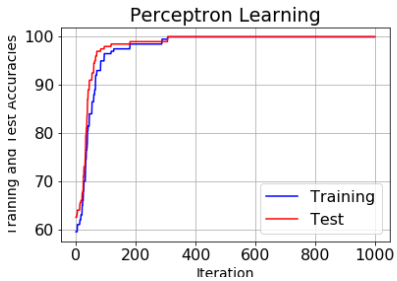

Figure 2: Learning Curves for Classifying two Gaussian Distributed Data

8. The `scikitlearn` package is an excellent source of machine learning algorithms in `Python`. Compare the performance of your perceptron algorithm on the two-class Gaussian dataset with that of the perceptron tool in the `scikitlearn` package. Here is a snippet of code to help you get started:

```
# Scikitlearn can do it for us
#
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
model = Perceptron()
model.fit(X_train, y_train)
yh_train = model.predict(X_train)
print("Accuracy on training set: %6.2f" %(accuracy_score(yh_train, y_train)))

yh_test = model.predict(X_test)
print("Accuracy on test set: %6.2f" %(accuracy_score(yh_test, y_test)))

if (accuracy_score(yh_test, y_test) > 0.99):
    print("Wow, Perfect Classification on Separable dataset!")
```

9. Consider the problem with means at $\boldsymbol{m}_1 = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix}$ and $\boldsymbol{m}_2 = \begin{bmatrix} 10.0 \\ 10.0 \end{bmatrix}$ with the covariance matrices equal and the same as before. Does the perceptron as implemented solve this problem? If not what modification is needed to help solve this problem? Hint:

```
O = np.ones((2*NumDataPerClass, 1))
X = np.append(X, O, axis=1)

w = np.random.randn(3)
```

10. Download a two class classification problem from the UCI machine Learning Repository of benchmark datasets `https://archive.ics.uci.edu/ml/index.php` and classify using your own perceptron algorithm. How does the performance compare to any quoted results on this dataset by other researchers? You may need more python tools to read and manipulate downloaded data (*e.g.* `Pandas`).

Report

Summarise your work in a report of **no more than two pages**.

Mahesan Niranjan February 2024
Material adapted from module Foundations of Machine Learning, taught at the School of Electronics and Computer Science, University of Southampton.