Iris_Perceptron_classifier

January 17, 2021

1 Training Perceptron with Iris dataset

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
# Importing Iris.csv
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/iris/
 ↪iris.data',
                 header=None)
```

[3]:
```
# Show data
df.head()
```

[3]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

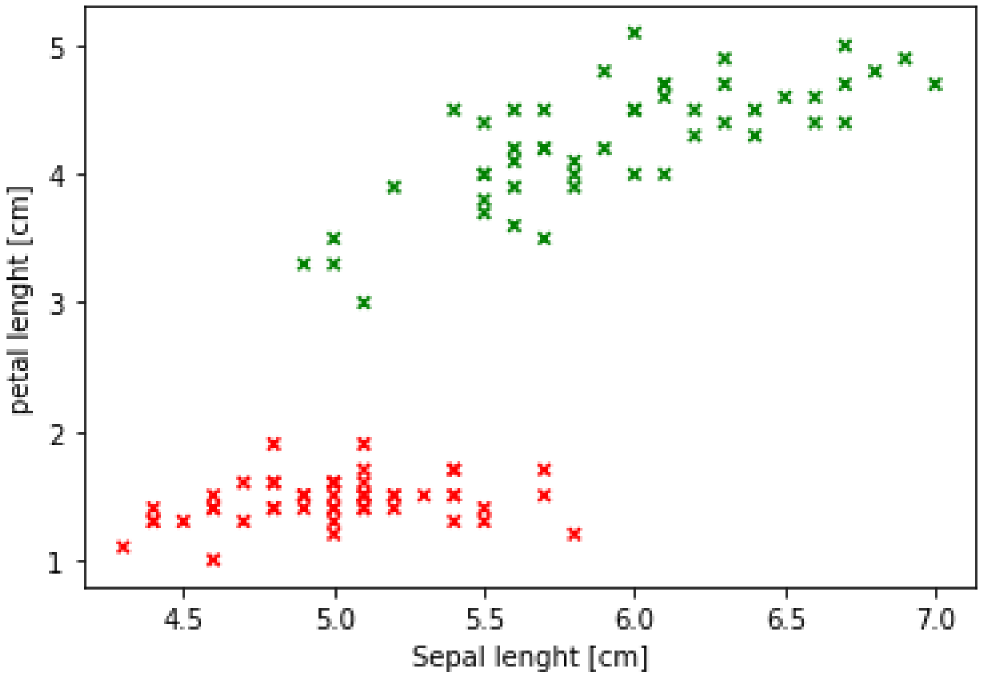
[4]:
```
# Slicing dataset, getting rid of Iris-virginica label
df = df[:99]
# Renaming columns
df.columns = ['Sepal lenght', 'Sepal width', 'Petal lenght', 'Petal width',␣
 ↪'Variety']
```

[5]:
```
df.tail()
```

[5]:

	Sepal lenght	Sepal width	Petal lenght	Petal width	Variety
94	5.6	2.7	4.2	1.3	Iris-versicolor
95	5.7	3.0	4.2	1.2	Iris-versicolor
96	5.7	2.9	4.2	1.3	Iris-versicolor
97	6.2	2.9	4.3	1.3	Iris-versicolor
98	5.1	2.5	3.0	1.1	Iris-versicolor

[104]:
```python
# Ploting the two groups of data classified by variety:
    # 'Iris-setosa': 'red'
    # 'Iris-versicolor': 'green'
df.plot(kind='scatter', x='Sepal lenght', y='Petal lenght',
        color=np.where(df['Variety']=='Iris-setosa','r', 'g'),
        xlabel='Sepal lenght [cm]', ylabel='petal lenght [cm]', marker='x');
```



1.1 Developing a perceptron model which can classify flowers

- We will be using 'Sepal lenght' and 'Petal lenght' as inputs
- We will expect the right 'Variety' as output
- The model activation function will be 'sigmoid(x)'
- Outputs will be between 0 and 1, so we have to convert 'Variety' to this codification

[7]:
```python
# Defining inputs
X = np.array([df['Sepal lenght'], df['Petal lenght']]).T
# Defining outputs. We will convert 'Iris-setosa' to 0 and 'Iris versicolor' to␣
 ↪1
y = np.array([df['Variety']]).T
y = np.where(y=='Iris-setosa',0,1)
```

[8]:
```python
class Perceptron:
    def __init__(self):
        pass
```

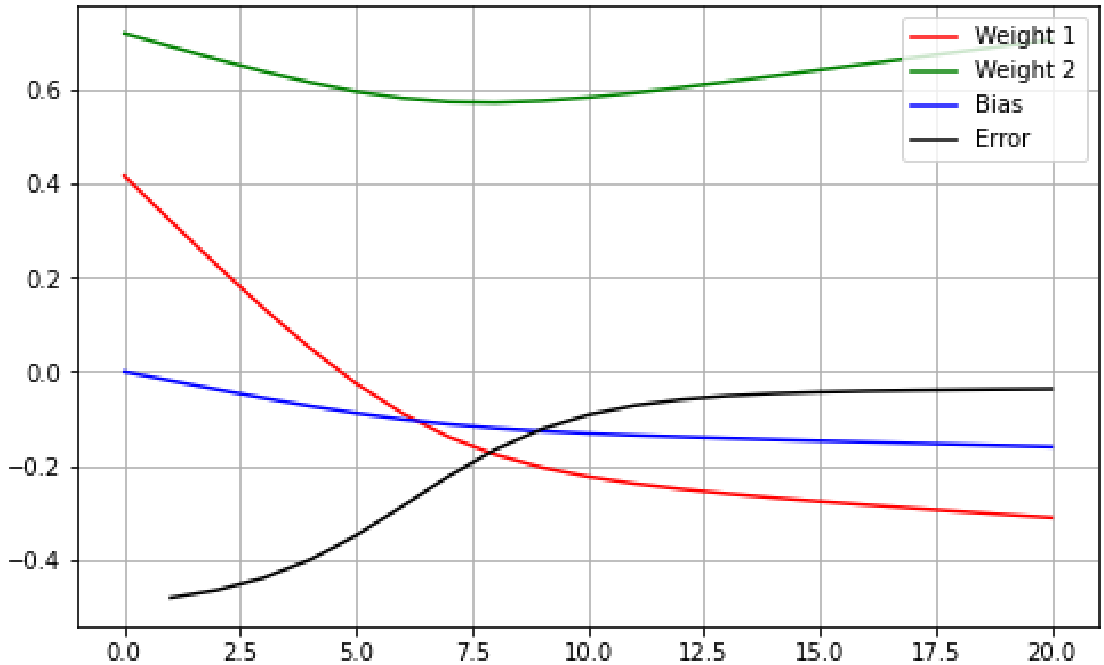
```python
    def d_sigmoid(self,X):
        return self.sigmoid(X) * (1 - self.sigmoid(X))

    def sigmoid(self,X):
        return 1 / (1 + np.exp(-X))

    def predict(self,X):
        summation = np.dot(X, self.weights) + self.bias
        activation = self.sigmoid(summation)
        return activation

    def train(self,X,y,n_iter=5,learn_rate=0.1):
        # Defining weights and bias
        size = np.shape(X)[1]
        np.random.seed(1)
        self.weights = np.random.rand(1,size).T
        self.bias = np.random.rand(1)
        # Saving historic weights and bias && historic error
        self.hist = np.concatenate((self.weights[:,0],self.bias),axis=0)
        self.hist_error = []
        # Model fitting
        for _ in range(n_iter):
            z = self.predict(X)
            error = y - z
            self.hist_error.append(np.mean(error))
            # Multipliying np.arrays like this: [a,b,c]*[d,e,f] = [a*d, b*e, c*f]
            activated_error = error * self.d_sigmoid(z)
            # Updating weights and bias
            self.weights += np.array([np.mean(X * activated_error, axis=0)]).T * learn_rate
            self.bias += np.mean(activated_error) * learn_rate
            # Storing historic
            new_data= np.concatenate((self.weights[:,0],self.bias),axis=0)
            self.hist = np.vstack((self.hist,new_data))
```

[9]:
```python
# Training the model 100 times with a learning factor of 0.2
perceptron = Perceptron()
perceptron.train(X,y,20,0.2)
```

1.1.1 Plotting weight and error evolution

[110]:
```python
data = perceptron.hist
error = perceptron.hist_error
plt.figure(figsize=(8,5))
plt.plot(range(len(data)), data[:,0], color='r', label='Weight 1')
```

```
plt.plot(range(len(data)), data[:,1], color='g', label='Weight 2')
plt.plot(range(len(data)), data[:,2], color='blue', label='Bias')
plt.plot(range(1,len(error)+1), error, color='black', label='Error')
plt.grid(True)
plt.legend(loc='upper right')
plt.show();
```



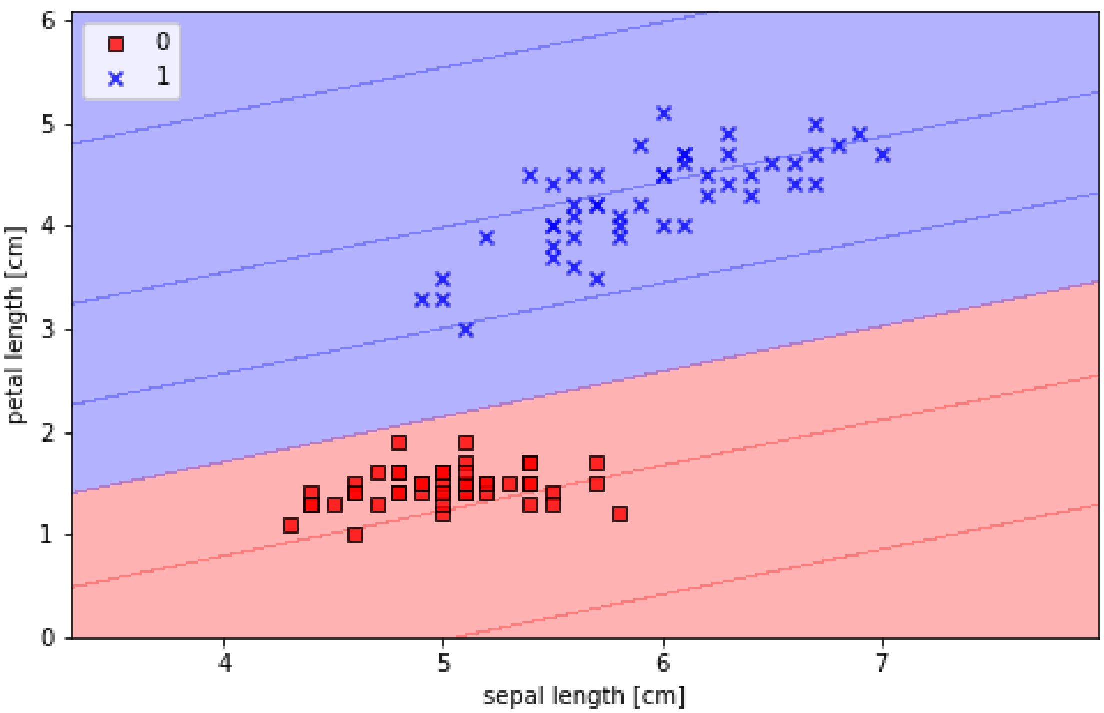
1.1.2 Visualizing desidion boundaries for two labels

[91]:

```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    # Meshgrid() + ravel() is the equivalent of 2 for loop to cover a 2D grid
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
```

```
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    # 2 times loop
    for idx, cl in enumerate(np.unique(y)):
        matches = np.where(y==cl)
        plt.scatter(x=X[matches[0], 0], y=X[matches[0], 1], alpha=0.8,␣
↪c=colors[idx],
                marker=markers[idx],label=cl, edgecolor='black')
```

[99]:
```
plt.figure(figsize=(8,5))
plot_decision_regions(X, y, classifier=perceptron, resolution=0.01)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left');
```



1.1.3 Evolution of desition boundaries

In a for loop that will run 'n' times we will do the following:
- We will train the model 'Iterator_n' times - We will save the output in a folder

[103]:
```
perceptron2 = Perceptron()
for i in range(10):
    # Training perceptron 'i' times
```

```
    perceptron2.train(X,y,i,0.2)
    # Setting figure size
    fig = plt.figure(figsize=(12,8))
    # Creating decision plot for the model
    plot_decision_regions(X, y, classifier=perceptron2, resolution=0.01)
    plt.xlabel('sepal length [cm]')
    plt.ylabel('petal length [cm]')
    plt.legend(loc='upper left')
    fname =  'plot/' + "{:03d}".format(i)
    plt.savefig(fname, dpi=75, transparent=False, format='jpg')
    # Closing figure in order not to display it
    plt.close(fig)
```