# perceptron

March 15, 2021

## 1 The Perceptron algorithm, Part 1

In this part 1 of the assignment, you will implement the following:

1. The Perceptron training procedure.
2. Show empirically that the perceptron does not converge on the XOR dataset.
3. Train the perceptron on the Shapes dataset and show that it converges.
4. Add an engineered feature to the XOR dataset that makes it linearly separable, and train the Perceptron on teh new dataset until convergence.

### 1.1 Write Your Name Here:

## 2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *perceptron.pdf* showing the code and the output of all cells, and save it in the same folder that contains the notebook file *perceptron.ipynb*.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing we will see when grading!
7. Submit **both** your PDF and notebook on Canvas.

[1]:
```
import numpy as np
import utils
```

### 2.1 1. The Perceptron training algorithm

Implement the training procedure for the Perceptron algorithm. Run for the specified number of epochs or until convergence, whichever happens first. The algorithm converged when it makes no mistake on the training examples. If the algorithm converged, display a message "Converged in <e> epochs!'.

[ ]:
```python
def train(X, y, E):
    """Perceptron training function.
    Args:
        X (np.ndarray): A 2D array training instances, one per row.
        y (np.ndarray): A vector of labels.
        E (int): the maximum number of epochs.

    Returns:
        np.ndarray: The learned vector of weights / parameters.
    """
    # Add bias feature to X.
    X = # YOUR CODE HERE

    # Initialize w wtih zero's.
    w = # YOUR CODE HERE

    for e in range(E):
        # YOUR CODE HERE




    return w
```

## 3 2. Run the Perceptron algorithm on the simple XOR dataset.

- Create the dataset and store it in X (the 4 examples with 2 features each) and y (the +1 or -1 labels).
- Run the perceptron algorithm for 20 epochs. Store the weight vector after each epoch (you amy have to modify the training procedure above to do this).
- After training, display all the weight vectors (one for each epoch).
- Do you see any pattern in how the weights change vs. epochs? Can you use this pattern to prove that the perceptron will not converged, no matter how many epochs it is run?

[ ]:
```python
# YOUR CODE HERE
```

### 3.1 Linear Classification using Perceptrons

Load the training examples from the '../data/shapes_colors.csv' file. Put the examples in the array X on rows, and the labels in the array y. Ignore the first row in the file (which contains the headings). Keep in mind that values are separated by commas and that the label is in the first column in the file. Hint: you may want to use the keyworded parameters *skiprows* and *delimiter* for np.loadtxt. See the documentation: https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html

[ ]:
```
shapes = np.loadtxt('YOUR CODE HERE')
print(shapes)
```

[ ]:
```
# All columns but the first contain the features.
X =
print(X)
```

[ ]:
```
# The first column contains the label.
y =
print(y)
```

[ ]:
```
# Train the Perceptron usign the raw features for 10 epochs.
w = train(X, y, 10)
print(w)
```

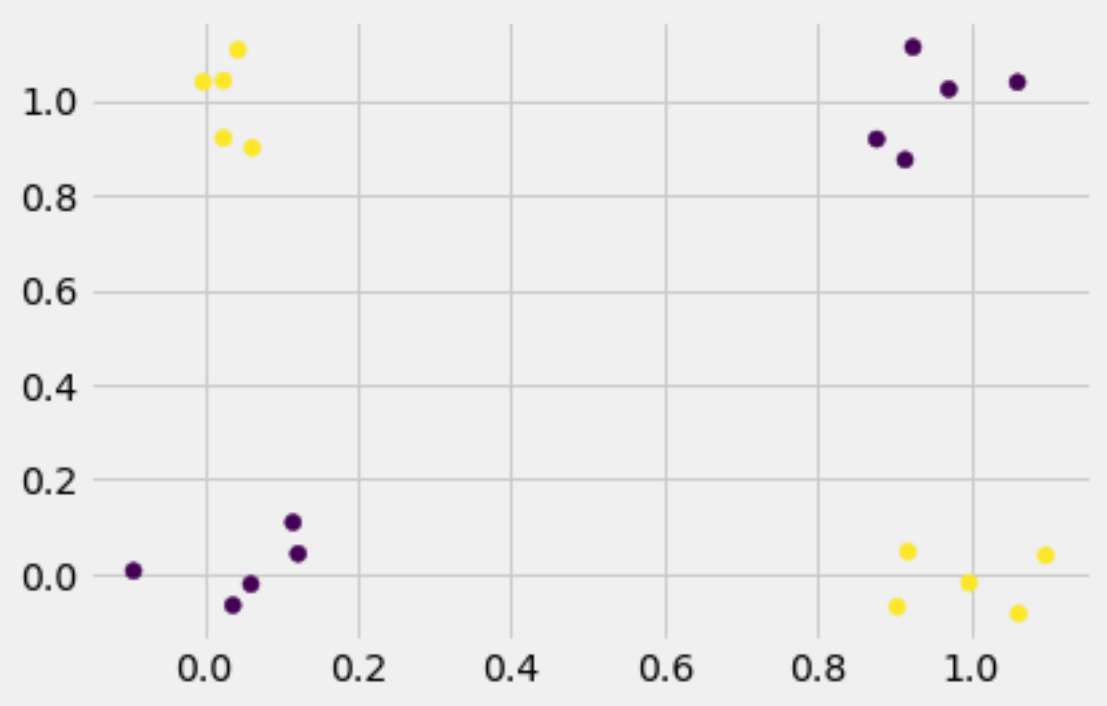
### 3.2 Nonlinear Classification with Perceptron and Feature Engineering

[2]:
```
# Create a more complex XOR dataset, with N points clustered around each of the␣
 ↪4 'corners'
def xor_dataset(N):
    """Generate XOR dataset.
    Args:
        N: number of points per example cluster.

    Returns:
        X: A 2D array with examples, one per line.
        y: A vector of labels.
    """
    X00 = (0, 0) + (np.random.sample((N, 2)) - 0.5) / 4
    y00 = np.full((N,), -1)
    X01 = (0, 1) + (np.random.sample((N, 2)) - 0.5) / 4
    y01 = np.full((N,), +1)
    X10 = (1, 0) + (np.random.sample((N, 2)) - 0.5) / 4
    y10 = np.full((N,), +1)
    X11 = (1, 1) + (np.random.sample((N, 2)) - 0.5) / 4
    y11 = np.full((N,), -1)

    X = np.row_stack((X00, X01, X10, X11))
    y = np.concatenate((y00, y01, y10, y11))

    return X, y
```

[3]:
```
X, y = xor_dataset(5)

import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

```
plt.scatter(X[:,0], X[:,1], c = y)
```

[3]: <matplotlib.collections.PathCollection at 0x7fb0200f58e0>




[ ]:
```
# Add a new feature x1 * x2 after the two features in each example from X.
Xnew = # YOUR CODE HERE

# This import registers the 3D projection, but is otherwise unused.
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xnew[:,0], Xnew[:,1], Xnew[:,2], c = y)
```

[ ]:
```
# Train the perceptron on the new XOR dataset using the 2 + 1 features.
w = train(Xnew, y, 10)
print(w)
```

[ ]:
```
# Display the decision boundary in the original 2D feature space.
xx, yy = np.meshgrid((np.arange(15) - 2) / 8, (np.arange(15) - 2) / 8)
zz = (-w[0] -w[1] * xx - w[2] * yy) / w[3]
print(zz.shape)
fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xnew[:,0], Xnew[:,1], Xnew[:,2], c = y)
ax.plot_surface(xx, yy, zz, alpha=0.2)
```

[ ]:
```
def predict(Xraw):
    # Add the engineered feature, to the right.
    Xnew = # YOUR CODE HERE

    # Add the bias feature. to the left.
    Xones = # YOUR CODE HERE

    # Compute prediction vector, where for each example x in Xones the output␣
 ↪should be wTx.
    pred = # YOUR CODE HERE

    return pred
```

[ ]:
```
import utils
utils.plot_decision_boundary(predict, X.T, y)
```

### 3.3 Analysis

Include an analysis of the results that you obtained in the experiments above.

### 3.4 Bonus

Anything extra goes here.

[ ]:
```

```