Perceptron

September 1, 2024

1 The Perceptron algorithm

In this assignment on Perceptrons, you will:

- Solve a few NumPy warm-up exercises.
- Implement the Perceptron training procedure, the Perceptron prediction function, and a function to compute prediction accuracy.
- Train the Perceptron on the simple logical AND and Shapes datasets and show that it converges.
  - Use the `sklearn` implementation of Perceptron on the same datasets above, compare trained parameters between your implementation and sklearn.
- Show empirically that the Perceptron does not converge on the XOR dataset.
- Add an engineered feature to the XOR dataset that makes it linearly separable, and train the Perceptron on the 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 showing the code and the output of all cells, and save it in the same folder that contains the notebook file.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and the notebook file .ipynb on Canvas.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

[ ]:
```
# Import all required packages.
import numpy as np
import utils
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

2.1 NumPy Warm-up

2.1.1 Obtain input dimensionality (10p)

[ ]:
```
# Return the dimensionality (number of dimensions) of input M
def dims(M):
    # YOUR CODE HERE



    # END CODE HERE

# Let's create 2 matrices and a vector
A = (np.arange(12).reshape(3, -1) - 4.5) / 5
v = np.array([2, -1, 1, 1])

print(dims(A)) # this should output 2
print(dims(v)) # this should output 1
print(dims(3)) # this should output 0
```

2.1.2 Combine a vector with a matrix (10p)

[ ]:
```
# Given 1D array v and 2D array M, create and return a matrix where:
#  - the first row is equal to v.
#  - the remaining rows are equal with the rows of M.
def topMatrix(M, v):
    # YOUR CODE HERE

    # END CODE HERE


M = topMatrix(A, v)

# This should output (4, 4).
print(M.shape)

M[0, 0] = 0

# This should output 2.
print(v[0])
```

2.1.3 Create matrix from alternating vectors (10p)

[ ]:
```
# Given two 1D arrays u and v and an integer n, create a 2D array with 2n rows
#    where the even index rows are equal to u
#    and the odd index rows are equal to v.
def alternateVectors(u, v, n):
    # YOUR CODE HERE




    # END CODE HERE

M = alternateVectors(np.ones(3), np.zeros(3), 2)

# This should output
# [[1. 1. 1.]
#  [0. 0. 0.]
#  [1. 1. 1.]
#  [0. 0. 0.]]
print(M)
```

2.1.4 Manipulate matrix diagonal, take one (10p)

[ ]:
```
# Create a new matrix computed by adding the same scalar c to all
# diagonal entries in an input square matrix M.
# Do this without using any iteration statements, i.e. no loops.
def add_scalar_to_diag(M, c):
    # YOUR CODE HERE

    # END CODE HERE

M = np.ones((2, 2))
N = add_scalar_to_diag(M, 3)
print(np.linalg.det(N)) # this should output 15.0.
```

2.1.5 Manipulate matrix diagonal, take two (10p)

[ ]:
```
# Create a new matrix computed by adding j^2 to each
# diagonal entry M[j,j] in an input square matrix M.
# Do this without using any iteration statements, i.e. no loops.
def add_squares_to_diag(M):
    # YOUR CODE HERE

    # END CODE HERE

M = np.ones((3, 3))
```

```
N = add_squares_to_diag(M)
print(np.diag(N)) # this should output [1. 2. 5.].
```

2.1.6 Compute row-wise sums with matrix products (10p)

[ ]:
```
def row_sums(M):
    # Create an array B such that A @ B computes
    #   the sum of elements in each row of A.

    # YOUR CODE HERE

    # END CODE HERE

    return A @ B

A = (np.arange(12).reshape(3, -1))
print(row_sums(A)) # this should output [ 6. 22. 38.]
```

2.2 The Perceptron training algorithm (50p)

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!'.

[ ]:
```
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
```

2.3 The prediction procedure (30p)

Given a dataset of examples X (one example $\mathbf{x}$ per row of X), use the trained perceptron parameters in $\mathbf{w}$ to predict a label: $+1$ if $\mathbf{w}^T\mathbf{x} > 0$, -1 otherwise.

Return the vector of predicted labels, one label for each example $\mathbf{x} \in X$.

[ ]:
```
def predict(X, w):
    # Add the bias feature.
    Xones = # YOUR CODE HERE

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

    # Compute prediction vector, +1 for positive scores, -1 for negative␣
↪scores, as integers.
    pred = # YOUR CODE HERE

    return pred
```

2.4 Computing the prediction accuracy (10p)

Given a vector `pred` of predicted labels and a vector `true` of correct labels, compute and return the percentage of predicted labels that are correct. Do it in one line of code.

[ ]:
```
def accuracy(pred, true):
    return #YOUR CODE HERE
```

2.5 Linear Classification using Perceptrons

2.5.1 A simple binary dataset: Logical AND

First, let's train the perceptron algorithm on a simple binary dataset that is linearly separable. Each example $x = [x_1, x_2]$ has two binary features, and the label $y(x) = x_1 \wedge x_2$ is the logical AND between the two features.

Create and store the 4 examples in the data matrix X and the label vector y.

[ ]:
```
# First example.
x1 = [0, 0]
y1 = -1

# Second example.
x2 = [0, 1]
y2 = -1

# Third example.
x3 = [1, 0]
y3 = -1

# Fourth example.
x4 = [1, 1]
y4 = 1

# Let's put them in the data matrix X and label vector y.
X = np.array([x1, x2, x3, x4])
y = np.array([y1, y2, y3, y4])
X, y
```

[ ]:
```
# Let's plot the dataset.
plt.scatter(X[:,0], X[:,1], c = y, s = 75)
plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
```

[ ]:
```
# Define a function that creates a dataframe from the dataset X, y
# with columns 'label', 'x1', 'x2'.
def dataframe(X, y):
    data = [[l] + x.tolist() for (x, l) in zip(X, y)]
    columns = ['label'] + ['x' + str(i + 1) for i in range(X.shape[1])]
    df = pd.DataFrame(data, columns = columns)

    return df

# Create the logical AND dataframe.
df = dataframe(X, y)
print(df)

# Save AND dataframe into a '.csv' file.
df.to_csv('../data/and_2d.csv', index = False)
```

2.5.2 Train the Perceptron on the AND dataset (5p)

```
[ ]: w = # YOUR CODE HERE
     w
```

Let's plot the decision boundary, which is given by $w_0 + w_1 \times x_1 + w_2 \times x_2 = 0$. This means that $x_2 = -(w_0 + w_1 \times x_1)/w_2$.

```
[ ]: # Plot the dataset.
     plt.scatter(X[:,0], X[:,1], c = y, s = 75)

     # Plot the decision boundary.
     x1vals = np.array([0.6, 1.2])
     x2vals = -(w[0] + w[1] * x1vals) / w[2]
     plt.plot(x1vals, x2vals, '-g')

     plt.xlabel('x1')
     plt.ylabel('x2')
     plt.grid()
```

2.5.3 Compute and print the accuracy of Perceptron on the same AND dataset (5p)

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

2.5.4 Evaluate the sklearn Perceptron on the AND dataset (10p)

The Sklearn library provides classes for many ML algorithms, using a uniform interface API for training and evaluating them. Here use the Peceptron class `sklearn.linear_model.Perceptron` and to train the 3 parameters (bias w_0, and w_1, w_2), and compare with the values obtained using your implementation.

The full documentation is at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html. See in particular `fit()`, `score()`, `intercept_`, `coef_`.

```
[ ]: from sklearn.linear_model import Perceptron

     # Create a Perceptron instance using the sklearn library. Set it to not shuffle␣
      ↪the training data.
     clf = Perceptron(max_iter = 21, shuffle = False)

     # Train a Perceptron model using the 'fit' function. Instruct it to initialize␣
      ↪all parameters with zero.
     # YOUR CODE HERE

     # Compute and print accuracy on training examples.
     acc = # YOUR CODE HERE
     print('Accuracy on AND:', acc)

     # Store the trained parameters in an array u, first w0, then w1 and w2.
```

```
u = # YOUR CODE HERE

# Are the sklearn parameters u the same as the parameters
# w obtained with our implementation?
print('w =', w)
print('u =', u)
```

2.5.5 Another binary dataset: Shapes (10p)

Let's also try the perceptron algorithm on the Shapes dataset. Each example has 4 binary features, hence the feature space is 4D. Is this dataset linearly separable?

[ ]:
```
# Load the dataset from a `.csv` file using `pandas` .
shapes_df = pd.read_csv('../data/shapes_colors.csv')
shapes_df
```

[ ]:
```
type(shapes_df.values)
```

[ ]:
```
# Store the data matrix in X.
X = shapes_df.values[:,1:]
X
```

[ ]:
```
# Store the labels in y.
y = shapes_df.values[:,0]
y
```

[ ]:
```
# Train the Perceptron on the 4 examples for a maximum of 10 epochs.
# Will training converge? Is the dataset linearly separable?

# YOUR CODE HERE
```

[ ]:
```
# Predict and print the labels of all training examples.
# YOUR CODE HERE


# Compute and print accuracy on training data.
# YOUR CODE HERE
```

2.6 Nonlinear Classification with Perceptron and Feature Engineering

2.6.1 A slightly more complex binary dataset: Logical XOR

[ ]:
```
def xor_dataset(N, seed = 1):
    """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.
    """
    np.random.seed(seed)
    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
```

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

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

[ ]:
```
# Create dataframe for XOR dataset.
df = dataframe(X, y)

# Save XOR dataframe into a '.csv' file.
df.to_csv('../data/xor_2d.csv', index = False)
```

2.6.2 Train and evaluate the Perceptron on the XOR dataset (10p)

[ ]:
```
# First, train the Perceptron algorithm for 10 epochs using the original raw␣
 ↪features.
# Will training converge? Is the dataset linearly separable?
w = # YOUR CODE HERE
print('Weights:', w)

# Predict the labels of all training examples, compare with true labels in y.
predictions = #YOUR CODE HERE
print(predictions)
print(y)

# Compute prediction accuracy on training examples (fitting performance).
acc = # YOUR CODE HERE
print('Accuracy on XOR:', acc)
```

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

# Plot the decision boundary.
x1vals = np.array([-0.6, 1.2])
x2vals = -(w[0] + w[1] * x1vals) / w[2]
plt.plot(x1vals, x2vals, '-g')

plt.xlabel('x1')
plt.ylabel('x2')
plt.grid()
```

2.6.3 Questions

1. What is the best possible accuracy a linear classifier can get on this dataset using just the two features?
   2. [Bonus] If the Perceptron got less that the best possible accuracy, can you change the Perceptron algorithm to achieve this optimal accuracy?
2. Is there a feature $x_3 = f(x_1, x_2)$ such that the dataset becomes linearly separable in the feature space $\phi(\mathbf{x}) = [x_1, x_2, x_3]$?

2.6.4 Your Answers

1.
2.

2.6.5 Feature engineering for the XOR dataset (10p)

[ ]:
```
# Create a new data matrix Xnew from X by adding to each example in X a new␣
↪feature x3
# computed from x1 and x2 that makes the dataset linearly separable.
Xnew = # YOUR CODE HERE

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

# Plot the dataset in the new feature space [x1, x2, x3].
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xnew[:,0], Xnew[:,1], Xnew[:,2], c = y)
```

[ ]:
```
# Create dataframe for XOR 3D dataset.
df = dataframe(Xnew, y)

# Save XOR dataframe into a '.csv' file.
df.to_csv('../data/xor_3d.csv', index = False)
```

2.6.6 Train and evaluate the Perceptron on the new XOR dataset (10p)

```
[ ]: # Train Perceptron on the augmented dataset for 10 epochs.
w = # YOUR CODE HERE
print('Weights:', w)

# Predict the labels of all training examples, compare with true labels in y.
predictions = # YOUR CODE HERE

# Compute prediction accuracy on training examples (fitting performance).
acc = # YOUR CODE HERE
print('Accuracy on XOR:', acc)
```

Plot the dataset and the decision boundary in the 3D $[x_1, x_2, x_3]$ feature space

```
[ ]: # Plot the dataset in the 3D [x1, x2, x3] feature space.
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xnew[:,0], Xnew[:,1], Xnew[:,2], c = y)

# Plot the decision boundary (hyperplane).
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]
ax.plot_surface(xx, yy, zz, alpha = 0.2)
```

Plot the dataset and the decision boundary in the original 2D $[x_1, x_2]$ feature space

```
[ ]: # Use prediction labels sign(w*x)to plot discrete decision boundary.
def predict_on_raw(Xraw):
    # Add the engineered feature(s)
    Xnew = np.column_stack((Xraw, Xraw[:,0] * Xraw[:,1]))

    return predict(Xnew, w)

import utils
utils.plot_decision_boundary(predict_on_raw, X.T, y)
```

```
[ ]: # Use prediction scores (margin) w*x to plot level curves.
def scores(Xraw):
    # Add the engineered feature(s)
    Xnew = np.column_stack((Xraw, Xraw[:,0] * Xraw[:,1]))

    # Add the bias feature.
    Xones = np.column_stack((np.ones(Xnew.shape[0]), Xnew))

    # Compute score vector, where for each example x in Xones, the output␣
    ↪should be wTx.
    scores = Xones @ w
```

```
    return scores

import utils
utils.plot_decision_boundary(scores, X.T, y)
```

2.7 Bonus (25p)

- Show empirically that the perceptron does not converge on the 2D XOR dataset, no matter for how many epochs it would be run:
  - Run the perceptron algorithm for 20 epochs. Store the weight vector after each epoch (you will have to modify the training procedure 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 converge, no matter how many epochs it is run?

[ ]:

2.8 Open Bonus

Anything extra goes here. For example, create a visualization of Perceptron training that displays the dataset and the decision boundary as it changes after each epoch, as an animated gif (matplotlib should be able to do this).

[ ]:

2.9 Analysis of results (10p)

Include an analysis of the results that you obtained in the experiments above. Take advantage of the Jupyter Notebook markdown language, which can also process Latex and HTML, to format your report so that it looks professional.

[ ]: