KernelPerceptron

October 23, 2024

1 The Kernel Perceptron algorithm

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

1. The Kernel Perceptron training and evaluation procedures.

2. Implement the Quadratic kernel.

3. Show empirically that the Kernel Perceptron converges on the XOR dataset.

4. Bonus points.

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 numpy as np
import utils

np.random.seed(1)
```

2.1 The Perceptron algorithm

You can copy here the implementation of the training and test procedures for the Perceptron algorithm in homework 1.

The training algorithm runs 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 perceptron_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 with zero's.
        w = # YOUR CODE HERE

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





        return w
```

Implement the Perceptron prediction function that takes as input the Perceptron parameters **w** and returns a vector with the labels (+1 or -1) predicted for the test examples in input data X.

Given a dataset of examples X (one example **x** per row of X), use the trained perceptron parameters in **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 perceptron_test(X, w):
        """Perceptron prediction function.
        Args:
            X (np.ndarray): A 2D array training instances, one per row.
            w (np.ndarray): A vector of parameters.


        Returns:
            np.ndarray: The vector of predicted labels.
        """
        # 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
```

2.2 The Kernel Perceptron

2.2.1 The Kernel Perceptron training procedure (30p)

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

Return the learned vector of dual parameters α in `alpha`.

[ ]:
```
def kperceptron_train(X, y, E, K):
    """Kernel 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.
        K (function): the kernel function.

    Returns:
        np.ndarray: The learned vector of dual parameters.
    """

    # Initialize dual parameters alpha with zero's.
    alpha = # YOUR CODE HERE

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






    return alpha
```

2.2.2 The Support Vectors (10p)

Write a function that takes as input the set of training examples, their labels, and the trained dual parameters, and return a tuple containing:

1. The support vectors;
2. Their labels;
3. Their dual parameters.

Remember, support vectors are training examples for which the corresponding dual parameter alpha is non-zero.

[ ]:
```
def support_vectors(X, y, alpha):
    """Select support vectors.
    Args:
        X (np.ndarray): A 2D array training instances, one per row.
        y (np.ndarray): A vector of labels.
        alpha (np.ndarray): The vector of dual parameters.

    Returns:
        X_s, y_s, alpha_s: The support vectors, their labels, and their dual␣
↪parameters.
    """

    # YOUR CODE HERE
    X_s =

    y_s =

    alpha_s =


    return X_s, y_s, alpha_s
```

2.2.3 The Kernel Perceptron prediction function (20p)

The function takes as input the Kernel Perceptron dual parameters alpha, support vectors, their labels, and the kernel, and returns a vector with the labels (+1 or -1) predicted for the test examples.

[ ]:
```
def kperceptron_test(X_s, y_s, alpha_s, K, X):
    """Kernel Perceptron prediction function.
    Args:
        X_s (np.ndarray): A 2D array of support vectors, one per row.
        y_s (np.ndarray): The vector of labels corresponding to the support␣
↪vectors.
        alpha_S (np.ndarray): The vector of dual parameters for the support␣
↪vectors.
        K (function): the kernel function.

        X (np.ndarray): A 2D array of test instances, one per row.

    Returns:
```

```
        np.ndarray: The vector of predicted labels for the test instances.
    """

    # Compute the kernel perceptron predictions on the examples in X.
    # YOUR CODE HERE
    pred =



    return pred
```

2.2.4 The quadratic kernel (10p)

Here is my implementation of the linear kernel $K(x, y) = 1 + x^T y$

```
[ ]: def linear_kernel(x, y):
         return 1 + x @ y
```

Implement the quadratic kernel $K(x, y) = (1 + x^T y)^2$.

```
[ ]: def quadratic_kernel(x, y):
         result = # YOUR CODE HERE

         return result
```

2.2.5 Experiment with the XOR dataset and the two perceptron algorithms.

```
[ ]: # 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
```

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

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

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

Let's train the Kernel Perceptron algorithm with the quadratic kernel for up to 50 epochs and evaluate its accuracy on a new set of test examples.

[ ]:
```
# Train the kernel perceptron
E = 50
alpha = kperceptron_train(X, y, E, quadratic_kernel)
print('alpha =', alpha)

# Find the support vectors.
X_s, y_s, alpha_s = support_vectors(X, y, alpha)
print('Found', alpha_s.size, 'support vectors out of', alpha.size, 'training␣
 ↪examples.')

# Test the kernel perceptron on a new set of test examples.
X_test, y_test = xor_dataset(2)
predictions = kperceptron_test(X_s, y_s, alpha_s, quadratic_kernel, X_test)
```

2.2.6 Accuracy computation (5p)

Write code to compute the accuracy of the Kernel Perceptron on the test examples.

[ ]:
```
# YOUR CODE HERE


accuracy =
print('Accuracy on test examples is:', accuracy)
```

2.3 [Bonus] From Dual to Primal parameters (10 bonus points)

A. Here, train the Perceptron algorithm on the XOR dataset, using the feature space corresponding to the quadratic kernel.

B. Empirically show that the trained weight vector **w** is equal with the weighted sum of the support vectors (slide 24) that were trained by the Kernel Perceptron on the same dataset.

[ ]:
```
def project_features(X):
    # Project the examples in X, which contain only 2 features x1 and x2, into␣
 ↪the feature space
```

```
    #     corresponding to the quadratic kernel shown on slide 29, i.e. from 2␣
↪to 5 features (do not include the bias feature)
    # Return the array of the new examples, each containing 5 features.

    # YOUR CODE HERE
    Xnew =



    return Xnew
```

[ ]:
```
# Train the perceptron on the projected version of the XOR dataset, using the 6␣
↪features.
Xnew = project_features(X)
E = 50
w = perceptron_train(Xnew, y, E)
print(w)
```

[ ]:
```
# Compute the weighted sum of the support vectors (slide 24) using X_s, y_s,␣
↪alpha_s.
kw = # YOUR CODE HERE


# Compare with w printed above. They should be the same.
print(kw)
```

2.4 Anything extra goes here.

- (10 p) For example, evaluate SVMs with a quadratic kernel on the XOR dataset (see SVM details in second part of the assignment).

[ ]:
```

```