

Machine Learning - Exercise 3

Cross validation

Polynomial regression

Polynomial regression consists of fitting some data (x, y) to a n -order polynome of the form:

$$y = f(x) = w_0 + w_1 \cdot x + w_2 \cdot x^2 + \dots + w_n \cdot x^n$$

By rewriting the unidimensional input x into the following vector:

$$\mathbf{x} = [1 \quad x \quad x^2 \quad \dots \quad x^n]$$

the problem can be reduced to linear regression:

$$y = \langle \mathbf{w} \cdot \mathbf{x} \rangle$$

and we can apply the Widrow-Hoff learning rule to find \mathbf{w} :

$$\Delta \mathbf{w} = \eta \cdot (y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle) \cdot \mathbf{x}_i$$

crossvalidation.py

You are provided with a script `crossvalidation.py` that should be modified in order to perform polynomial regression on the file `polynome.data`.

```
import numpy as np
import matplotlib.pyplot as plt

# Load the data set
data = np.loadtxt('polynome.data')
X = data[:, 0]
Y = data[:, 1]
N = len(X)

def visualize(w):
    # Plot the data
    plt.plot(X, Y, 'r.')
    # Plot the fitted curve
    x = np.linspace(0., 1., 100)
    y = np.polyval(w, x)
    plt.plot(x, y, 'g-')
    plt.title('Polynomial regression with order ' + str(len(w)-1))
    plt.show()

# Apply polynomial regression of order 2 on the data
w = np.polyfit(X, Y, 2)

# Visualize the fit
visualize(w)
```

polyfit and polyval

- A first solution to perform polynomial regression would be to adapt the code you wrote in the last exercise session for logistic regression. However, you saw that properly setting the correct learning rate and stop criteria can be quite tricky.
- The other solution for this exercise is to use the built-in functions of NumPy which can already perform polynomial regression in an optimized and proved-sure manner (Note: NumPy does not use gradient descent, but rather directly minimizes the error-function by inverting the Gram matrix).

```
p = np.polyfit(x, y, deg)
```

- This function takes the input values x , the output values y and the desired degree of the polynome deg , performs the polynomial regression and returns the adequate set of weights (beware: the higher-order coefficient comes first).
- Once the weights are obtained, one can use them to predict the value of an example with the function:

```
y = np.polyval(p, x)
```

Questions

1. Run the script `crossvalidation.py` in order to visualize the data (16 examples). According to you, which order should the polynome have at least, in order to perform a good fit?
2. Complete `crossvalidation.py` in order to apply the polynomial regression to the data contained in `polynome.data`, with degrees ranging from 1 (linear regression) to 20. Visualize all the fits. Starting from which degree do they look OK?
3. For each of these hypotheses, print the quadratic error on the training set. How does the training error evolve when the degree of the polynome is increased? What is the risk by taking the hypothesis with the smallest training error?

Questions

4. Apply **simple hold-out cross-validation** to find the optimal degree for the polynomial regression. Separate the data set into S_{train} (70%) and S_{test} (30%). Train each polynome with S_{train} and compute the generalization error on S_{test} . Which degree of the polynome gives the minimal empirical error? Why?

→ Hint: the data (X,Y) can be easily split using the NumPy function `hsplit` (check the documentation).

5. 70% of 16 training examples is really not a lot to hope for a good fit. Apply the **k-fold cross-validation** (with $k = 1$ or 2). Does it change something to the optimal degree of the polynome?

→ Hint: once the data has been split into $\frac{N}{k}$ subsets using `hsplit`, you can easily recombine all of them but one with the function `hstack`.

```
# A vector of 6 elements.
x = np.array(['a', 'b', 'c', 'd', 'e', 'f'])
# Split it into 3 subsets.
x_split = np.hsplit(x, 3)
# Split it into 2 subsets at the index 4.
x_split = np.hsplit(x, [4])
# Get all subsets except the second one.
x_train = np.hstack((x_split[i] for i in range(3) if not i == 1))
```