Classification and Representation Learning

Autonomous work: Learning Theory

Hoel Le Capitaine

Academic year 2020-2021

Autonomous work

This is your second autonomous work.
This time, you need to write a report on it, that will be evaluated, and it will count for your final mark of this course.
You must write your report using LaTeX, and the IEEE double column conference format, available here:
`https://www.ieee.org/documents/ieee-latex-conference-template.zip`

Your work

- must be done alone (any plagiarism both into the code or the report will be severly punished),
- must contain your pdf document, and your source code (the file `crossval.py`) in python that is needed to run the experiments,
- will be deposited on the extradoc platform, by November, 6th.

Polynomial regression

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

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

- that can be reduced to linear regression (see slides of the lecture) $y = \mathbf{w} \cdot \mathbf{x}$, where $\mathbf{x} = [1 \; x \; x^2 \cdots x^n]$
- so we can apply the Widrow-Hoff learning rule to find $\mathbf{w}$

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

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 inversing the Gram matrix).

```
w = polyfit(x, y, deg)
```

This function takes the input values , the output values 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 = polyval(w, x)
```

Regression

Create a file `crossval.py` with the following content

```
from numpy import *
import matplotlib.pyplot as plt
# Load the data set
data = loadtxt('polynome.data')
# Separate the input from the output
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 = linspace(0., 1., 100)
  y = 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 = polyfit(X, Y, 2)
# Visualize the fit
visualize(w)
```

Questions

1. Run the script `crossval.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 `crossval.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 cross-validation to find the optimal degree for the polynomial regression. Separate the data set into $\mathcal{S}_{train}$ (70%) and $\mathcal{S}_{test}$ (30%). Train each polynome with and compute the generalization error on $\mathcal{S}_{test}$. Which degree of the polynome gives the minimal empirical error? Why?

   Hint: the data (X,Y) can be easily splitted 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 splitted 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 = array(['a', 'b', 'c', 'd', 'e', 'f'])
# Split it into 3 subsets.
x_split = hsplit(x, 3)
# Split it into 2 subsets at the index 4.
x_split = hsplit(x, [4])
# Get all subsets except the second one.
x_train = hstack((x_split[i] for i in range(3) if not i == 1))
```

That's all folks!