Project 1 on Machine Learning

Data Analysis and Machine Learning course

Jan 23, 2019

Regression analysis and resampling methods

The main aim of this project is to study in more detail various regression methods, including the Ordinary Least Squares (OLS) method, Ridge regression and finally Lasso regression. The methods are in turn combined with resampling techniques.

We will first study how to fit polynomials to a specific two-dimensional function called Franke's function. This is a function which has been widely used when testing various interpolation and fitting algorithms. Furthermore, after having etsablished the model and the method, we will employ resamling techniques such as the cross-validation and/or the bootstrap methods, in order to perform a proper assessment of our models.

The Franke function, which is a weighted sum of four exponentials reads as follows

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ + \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right).$$

The function will be defined for $x, y \in [0,1]$. Our first step will be to perform an OLS regression analysis of this function, trying out a polynomial fit with an x and y dependence of the form $[x, y, x^2, y^2, xy, \dots]$. We will also include cross-validation and bootstrap as resampling techniques. As in homeworks 1 and 2, we can use a uniform distribution to set up the arrays of values for x and y, or as in the example below just a fix values for x and y with a given step size. In this case we will have two predictors and need to fit a function (for example a polynomial) of x and y. Thereafter we will repeat much of the same procedure using the the Ridge and Lasso regression methods, introducing thus a dependence on the bias (penalty) λ.

Thereafter we are going to use (real) digital terrain data and try to reproduce these data using the same methods. We will also try to go beyond the second-order polynomials metioned above and explore which polynomial fits the data best. This part of the project is however optional. It gives you additonal points (30).

The Python fucntion for the Franke function is included here (it performs also a three-dimensional plot of it)

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
from random import random, seed

fig = plt.figure()
ax = fig.gca(projection='3d')

# Make data.
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
x, y = np.meshgrid(x,y)


def FrankeFunction(x,y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4


z = FrankeFunction(x, y)

# Plot the surface.
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-0.10, 1.40)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

Part a): Ordinary Least Square on the Franke function with resampling. We will thus again generate our own dataset for a function FrankeFunction(x, y)

where $x, y \in [0,1]$ could be defined by random numbers computed with the uniform distribution. The function $f(x,y)$ is the Franke function. You should explore also the addition an added stochastic noise to this function using the normal distribution $\mathcal{N}(\prime, \infty)$.

Use **scikit-learn** or write your own code (using either a matrix inversion or a singular value decomposition from e.g., **numpy**) or use your code from homeworks 1 and 2 and perform a standard least square regression analysis using polynomials in x and y up to fifth order. Find the confidence intervals of the parameters β by computing their variances, evaluate the Mean Squared error (MSE)

$$MSE(\hat{y}, \hat{\tilde{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2,$$

and the R^2 score function. If $\tilde{\hat{y}}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value, then the score R^2 is defined as

$$R^2(\hat{y}, \tilde{\hat{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2},$$

where we have defined the mean value of $\hat{y}$ as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i.$$

Perform a resampling of the data where you split the data in training data and test data. Implement the k-fold cross-validation algorithm or the bootstrap algorithm and evaluate again the MSE and the R^2 functions resulting from the test data.

Part b): Ridge Regression on the Franke function with resampling. Use **scikit-learn** or write your own code for the Ridge method, either using matrix inversion or the singular value decomposition as done in the previous exercise or howework 2 (see also chapter 3.4 of Hastie *et al.*, equations (3.43) and (3.44)). Perform the same analysis as in the previous exercise (for the same polynomials and include resampling techniques) but now for different values of λ. Compare and analyze your results with those obtained in part a). Study the dependence on λ while also varying eventually the strength of the noise in your expression for FrankeFunction(x, y).

Part c): Lasso Regression on the Franke function with resampling. This part is essentially a repeat of the previous two ones, but now with Lasso regression. In this case as well, you can also use the functionalities of **scikit-learn**. Give a critical discussion of the three methods and a judgement of which model fits the data best.

Part d): Optional: Introducing real data. This part is optional and gives (d+e) an additional score of 30 points on top of 100. With our codes functioning and having been tested properly on a simpler function we are now ready to look at real data. We will essentially repeat in part e) what was done in parts a-c). However, we need first to download the data and prepare properly the inputs to our codes. We are going to download digital terrain data from the website https://earthexplorer.usgs.gov/,

In order to obtain data for a specific region, you need to register as a user (free) at this website and then decide upon which area you want to fetch the digital terrain data from. In order to be able to read the data properly, you need to specify that the format should be **SRTM Arc-Second Global** and download the data as a **GeoTIF** file. The files are then stored in *tif* format which can be imported into a Python program using

```
scipy.misc.imread
```

Here is a simple part of a Python code which reads and plots the data from such files

```
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# Load the terrain
terrain1 = imread('SRTM_data_Norway_1.tif')
# Show the terrain
plt.figure()
plt.title('Terrain over Norway 1')
plt.imshow(terrain1, cmap='gray')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

If you should have problems in downloading the digital terrain data, we provide two examples under the data folder of project 1. One is from a region close to Stavanger in Norway and the other Møsvatn Austfjell, again in Norway.

Part e) Optional OLS, Ridge and Lasso regression with resampling. Our final part deals with the parameterization of your digital terrain data. We will apply all three methods for linear regression as in parts a-c), the same type (or higher order) of polynomial approximation and the same resampling techniques to evaluate which model fits the data best.

At the end, you should pesent a critical evaluation of your results and discuss the applicability of these regression methods to the type of data presented here.

Background literature

1. For a discussion and derivation of the variances and mean squared errors using linear regression, see the Lecture notes on ridge regression by Wessel N. van Wieringen

2. The textbook of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer, chapters 3 and 7 are the most relevant ones for the analysis here.

A small python function that shows how to plot your results

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm # This is needed to define a colormap
import numpy as np
from random import random, seed

# Define own function to generate some height values
def FrankeFunction(x,y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4

# Make data.
n_row = 100
n_col = 1000

ax_row = np.random.randn(n_row)
ax_col = np.random.randn(n_col)

# If you wish to plot the data, remember to sort the axes such that the surface
# is rendered correctly

sort_inds_row = np.argsort(ax_row) # This functions returns the indices to
                                   # ax_row that makes ax_row sorted
                                   # (see the declaration of ax_row_sorted on how to use the
sort_inds_col = np.argsort(ax_col)

ax_row_sorted = ax_row[sort_inds_row]
ax_col_sorted = ax_col[sort_inds_col]

# Grid the data.
# This must be done such that surf (surface plot) understands which (x,y)- point corresponds
# Meshgrid returns matrices where the first matrix is ax_column repeated along the rows (ver
# where the second matrix has ax_row repeated along the columns (horizontal direction) .
# Each pair (rowmat[i,j], colmat[i,j]) for all i,j constitues a coordinate in
# the plane in the rectangular domain [min(ax_row), max(ax_row)]x[min(ax_column), max(ax_col
colmat, rowmat = np.meshgrid(ax_col_sorted, ax_row_sorted)

# This evaluates the height associated for each pair of coordinate made from np.meshgrid
z = FrankeFunction(rowmat, colmat)

"""
If you wish to make some predictons on z using some regression method and plot the result,
it is possible to do so by predicting every pair of (rowmat[i,j], colmat[i,j]).

One could think of the pair  (rowmat[i,j], colmat[i,j]) as x-and y-coordinates in the plane.
When you have your parameters, apply them to every  (rowmat[i,j], colmat[i,j]) - pair.
This could for instance be done by a double for loop, where you perform the predictions
within the innermost loop.

Every prediction can then be stored in a matrix, e.g z_pred
where z_pred[i,j] corresponds to the prediction of (rowmat[i,j], colmat[i,j]),
that can be visualized in the same manner as z in the code below.
"""

# Plot the generated surface.
fig = plt.figure() # Starts a new window

# specify that we are plotting in 3D
ax = fig.gca(projection='3d')

surf = ax.plot_surface(rowmat, colmat, z, linewidth = 0, antialiased = False, cmap=cm.viridi
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

# Add a color bar that shows which color each value is mapped to.
fig.colorbar(surf)

# Show the plot
plt.show()
```

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Send us **only** the report file! For the source code file(s) you have developed please provide us with your link to your github domain. The report file should include all of your discussions and a list of the codes you have developed. Do not include library files which are available at the course homepage, unless you have made specific changes to them.

- In your git repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.

Finally, we encourage you to collaborate. Optimal working groups consist of 2-3 students. You can then hand in a common report.

Software and needed installations

If you have Python installed (we recommend Python3) and you feel pretty familiar with installing different packages, we recommend that you install the following Python packages via **pip** as

1. pip install numpy scipy matplotlib ipython scikit-learn tensorflow sympy pandas pillow

For Python3, replace **pip** with **pip3**.

See below for a discussion of **tensorflow** and **scikit-learn**.

For OSX users we recommend also, after having installed Xcode, to install **brew**. Brew allows for a seamless installation of additional software via for example

1. brew install python3

For Linux users, with its variety of distributions like for example the widely popular Ubuntu distribution you can use **pip** as well and simply install Python as

1. sudo apt-get install python3 (or python for python2.7)

etc etc.

If you don't want to install various Python packages with their dependencies separately, we recommend two widely used distrubutions which set up all relevant dependencies for Python, namely

1. Anaconda Anaconda is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system **conda**

2. Enthought canopy is a Python distribution for scientific and analytic computing distribution and analysis environment, available for free and under a commercial license.

Popular software packages written in Python for ML are

- Scikit-learn,

- Tensorflow,

- PyTorch and
- Keras.

These are all freely available at their respective GitHub sites. They encompass communities of developers in the thousands or more. And the number of code developers and contributors keeps increasing.