Homework 1

Data Analysis and Machine Learning

Jan 22, 2019

Exercise 1

The first exercise here is of a mere technical art. We want you to have

- git as a version control software and to establish a user account on a provider like GitHub. Other providers like GitLab etc are equally fine. You can also use the University of Oslo GitHub facilities.
- Install various Python packages

We will make extensive use of Python as programming language and its myriad of available libraries. You will find IPython/Jupyter notebooks invaluable in your work. You can run **R** codes in the Jupyter/IPython notebooks, with the immediate benefit of visualizing your data. You can also use compiled languages like C++, Rust, Fortran etc if you prefer. The focus in these lectures will be on Python.

If you have Python installed (we recommend Python3.6 or higher versions) 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 sympy pandas pillow

For **Tensorflow**, we recommend following the instructions in the text of Aurelien Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly

We will come back to **tensorflow** later.

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

For OSX users we recommend, 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 pyhton2.7)

If you don't want to perform these operations separately and venture into the hassle of exploring how to set up dependencies and paths, we recommend two widely used distrubutions which set up all relevant dependencies for Python, namely

- Anaconda,

which 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**.

- Enthought canopy

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

We recommend using **Anaconda**.

Exercise 2

You should install and explore

1. Numpy and Scipy
2. Matplotlib
3. Pandas
4. Jupyter notebook

Simple Random walk. Make then a simple program which simulates a random walk and then plots the first 100 values. The example here may help. import random import matplotlib.pyplot as plt position = 0 steps = 1000 walk = [] for i in range(steps): step = 1 if random.randint(0,1) else -1 position+= step walk.append(position)

plt.plot(walk[:100]) plt.show()

Simple Linear algebra example. Write a simple program which performs basic matrix-vector multiplications and finds also the inverse of a matrix. You could use the following example import numpy as np from numpy.linalg import inv x = np.array([[1., 2., 3.],[4.,5.,6.]]) y = np.array([[6.,23,],[-1.,7.],[8.,9.]]) print(x) print(y) z = x.dot(y) equivalent to np.dot(x,y) print(z) print(np.dot(x,y)) z = np.dot(x,np.ones(3)) or write it as z = x @ np.ones(3) print(z)

X = np.random.randn(5,5) mat = X.T.dot(X) print(inv(mat)) print(mat.dot(inv(mat)))

Exercise 3

We will generate our own dataset for a function $y(x)$ where $x \in [0,1]$ and defined by random numbers computed with the uniform distribution. The function y is a quadratic polynomial in x with added stochastic noise according to the normal distribution $\mathcal{N}(/,\infty)$. The following simple Python instructions define our x and y values (with 100 data points). x = np.random.rand(100,1) y = 5*x*x+0.1*np.random.randn(100,1)

1. Write your own code (following the examples under the regression slides) for computing the parametrization of the data set fitting a second-order polynomial.

2. Use thereafter **scikit-learn** (see again the examples in the regression slides) and compare with your own code.

3. Using scikit-learn, compute also the mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error defined as

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

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

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

You can use the functionality included in scikit-learn. If you feel for it, you can use your own program and define functions which compute the above two functions. Discuss the meaning of these results. Try also to vary the coefficient in front of the added stochastic noise term and discuss the quality of the fits.

Exercise 4, variance of the parameters β in linear regression

Show that the variance of the parameters β in the linear regression method (chapter 3, equation (3.8) of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer) is given as

$$\mathrm{Var}(\hat{\beta}) = \left(\hat{X}^T\hat{X}\right)^{-1}\sigma^2,$$

with

$$\sigma^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2,$$

where we have assumed that we fit a function of degree $p-1$ (for example a polynomial in x).

Solution to exercise 3

import numpy as np import matplotlib.pyplot as plt

n = 100 data points xmax = 1 error = 0.1

The following simple Python instructions define our x and y values (with 100 data points) x = np.random.rand(n,xmax) y = 5*x*x+error*np.random.randn(n,xmax) y = 5x$^2 + noise$

for xi, yi in zip(x, y):   print(xi, yi)

The above defines a set of equations

$\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon}$

where the elements of $\hat{y}$ are given by $y_i = 5x_i + \epsilon_i$. Afterwards, $\hat{y}$ is turned into a 100×1 matrix so that `np.linalg` can accept it as input.

We define the matrix $= \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_{99} & x_{99}^2 \end{bmatrix}$ with $x_0,\ x_1,\ \ldots x_{99}$ given by `x = np.random.rand(100,1)`. $\hat{X}$ is then given by:

X = np.c$_[np.ones((n,1)), x, x*x] columnwisearrayconcatenationprint(X)$

For an explanation of `np.c_` see `https://stackoverflow.com/questions/39136730/confused-about-numpy-c-document-and-sample-code`.

The solution for the parameters $\hat{\beta}$ is given by

$\hat{\beta} = \left(\hat{X}^T\hat{X}\right)^{-1}\hat{X}^T\hat{y}$

or

beta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y) print("y = " + str(beta[2,0]) + "*x$^2 + " + str(beta[1,0]) + " * x + " + str(beta[0,0]))$

Now we plot the resulting function over the data points to visualise how good the fit looks (two points would be enough for a line, but it won't cut it for a quadratic fit):

nfit = 100 xplot = np.arange(nfit)*xmax / (nfit - 1) nfit points, evenly distributed print(xplot) print(xplot**2)

Then define a new matrix $\hat{X}_{\mathrm{plot}}$ for the evenly distributed values we want to use for plotting:

Xplot = np.c$_[np.ones((nfit,1)), xplot, xplot**2] concatenatecolumns(asabove)$

Finally, calculate the fitted values $\hat{y}_{\mathrm{predict}}$ using that matrix and the fitted parameters $\hat{\beta}$:

ypredict = Xplot.dot(beta)

We can plot and compare with the true quadratic function $y_{\mathrm{true}} = 5x^2$:

ytrue = 5*xplot*xplot

plt.plot(x, y ,'ro') plt.plot(xplot, ytrue, label="y_{true}") plt.plot(xplot, ypredict, label="y_{predict}") plt.xlabel(r'x') plt.ylabel(r'y') plt.title(r'Quadratic Regression') plt.legend() plt.show()

We see that this produces a decent enough fit.

We use thereafter **scikit-learn**.

from sklearn.linear$_modelimportLinearRegression$

clf2 = LinearRegression() clf2.fit(X, y) ysklearn = clf2.predict(Xplot)

print("ypredict = " + str(clf2.coef$_[0, 2]) + " * x^2 + " + str(clf2.coef_[0, 1]) + " * x + " + str(clf2.coef_[0, 0]))print("ysklearn = " + str(beta[2, 0]) + " * x^2 + " + str(beta[1, 0]) + " * x + " + str(beta[0, 0]))notethattheindicesarereversedinthescikit-learnapproachcomparedtowhatwedidbefore : theshapeis(1, n)insteadof(n, 1)$

plt.plot(x, y ,'ro') plt.plot(xplot, ytrue, label="y_{true}") plt.plot(xplot, ypredict, label="y_{predict}") plt.plot(xplot, ysklearn, label="y_{sklearn}") plt.xlabel(r'x') plt.ylabel(r'y') plt.title(r'Quadratic Regression') plt.legend() plt.show()

Plotting the absolute relative error for the two predictions:

err$_predict = abs(ypredict[:, 0]-ytrue)/abs(ytrue)thepredictedy'shaveshape(n, 1)err_sklearn = abs(ysklearn[:, 0] - ytrue)/abs(ytrue)$

plt.plot(xplot, err$_predict, label = "\epsilon_{\text{predict}}$") plt.plot(xplot, err$_sklearn, label = "\epsilon_{\text{sklearn}}$") plt.xlabel(r'$x$') plt.ylabel(r'$\epsilon_{\text{rel}}$') plt.axis([0, xmax, 0, 2]) plt.title(r'Absolute relative error') plt.legend() plt.show()

It is indeed hard to see any difference between the two approaches:

plt.plot(xplot, abs(err$_predict), label = "\epsilon_{\text{predict}}$") plt.plot(xplot, abs(err$_sklearn), label = "\epsilon_{\text{sklearn}}$") plt.xlabel(r'$x$') plt.ylabel(r'$\epsilon_{\text{rel}}$') plt.axis([0, xmax, 0, 0.02]) plt.title(r'Absolute relative error') plt.legend() plt.show()

The mean squared error is the expected value of the quadratic error:

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

from sklearn.metrics import mean$_squared_error$

Note that the y values are not sorted, but ypredict and ysklearn are (we used Xplot to find these, not X) Thus, done with plotting, let us instead make new predictions based on X

ypredict2 = X.dot(beta) ysklearn2 = clf2.predict(X)

Then we find the MSE:

print("Mean squared error (ypredict):", mean$_squared_error(y, ypredict2))print("Meansquarederror(ysklear$
$", mean_squared_error(y, ysklearn2))$

The R^2 score function is defined as

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

where

- $\bar{y}$ is the mean value of $\hat{y}$
- $\bar{y}_i$ is the ith predicted value, and
- y_i is the corresponding true value

from sklearn.metrics import r2$_{s}core$

print("R$^2 score(ypredict) : ", r2_score(y, ypredict2))print("R^2 score(ysklearn) :$ $", r2_score(y, ysklearn2))$