First and second days: Exercise set 1

Data Analysis and Machine Learning for Nuclear Physics

Jul 21, 2021

Day one and two exercises

*

Exercise 1: Getting started

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.
- 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) 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 Python2.7)

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: Our first Python encounter

This exercise has as its aim to write a small program which reads in data from a **csv** file on the equation of state for dense nuclear matter. The file is localized at `https://github.com/mhjensen/MachineLearningMSU-FRIB2020/blob/master/doc/pub/Regression/ipynb/datafiles/EoS.csv`. Thereafter you will have to set up the design matrix $\boldsymbol{X}$ for the n datapoints and a polynomial of degree 3. The steps are:

- Write a Python code which reads the in the above mentioned file.
- Use for example **pandas** to order your data and find out how many data points there are.
- Set thereafter up the design matrix with dimensionality $n \times p$ where $p = 4$ and where you have defined a polynomial of degree $p - 1 = 3$. Print the matrix and check that the numbers are correct.

We recommend looking at the examples in the regression slides.

Solution.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# Where to save the figures and data files
PROJECT_ROOT_DIR = "Results"
FIGURE_ID = "Results/FigureFiles"
DATA_ID = "DataFiles/"

if not os.path.exists(PROJECT_ROOT_DIR):
    os.mkdir(PROJECT_ROOT_DIR)

if not os.path.exists(FIGURE_ID):
    os.makedirs(FIGURE_ID)

if not os.path.exists(DATA_ID):
    os.makedirs(DATA_ID)

def image_path(fig_id):
    return os.path.join(FIGURE_ID, fig_id)

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

def save_fig(fig_id):
    plt.savefig(image_path(fig_id) + ".png", format='png')

def R2(y_data, y_model):
    return 1 - np.sum((y_data - y_model) ** 2) / np.sum((y_data - np.mean(y_data)) ** 2)
def MSE(y_data,y_model):
    n = np.size(y_model)
    return np.sum((y_data-y_model)**2)/n

infile = open(data_path("EoS.csv"),'r')

# Read the EoS data as  csv file and organized into two arrays with density and energies
EoS = pd.read_csv(infile, names=('Density', 'Energy'))
EoS['Energy'] = pd.to_numeric(EoS['Energy'], errors='coerce')
EoS = EoS.dropna()
Energies = EoS['Energy']
Density = EoS['Density']
#  The design matrix now as function of various polytrops
X = np.zeros((len(Density),5))
X[:,0] = 1
X[:,1] = Density**(2.0/3.0)
X[:,2] = Density
X[:,3] = Density**(4.0/3.0)
X[:,4] = Density**(5.0/3.0)
# We split the data in test and training data
X_train, X_test, y_train, y_test = train_test_split(X, Energies, test_size=0.2)
# matrix inversion to find beta
beta = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
# and then make the prediction
ytilde = X_train @ beta
print("Training R2")
print(R2(y_train,ytilde))
print("Training MSE")
print(MSE(y_train,ytilde))
```

```
ypredict = X_test @ beta
print("Test R2")
print(R2(y_test,ypredict))
print("Test MSE")
print(MSE(y_test,ypredict))
```

*

Exercise 3: making your own data and exploring scikit-learn

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 $\mathcal{N}(\prime, \infty)$. The following simple Python instructions define our x and y values (with 100 data points).

```
x = np.random.rand(100,1)
y = 2.0+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

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

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.

Solution. The code here is an example of where we define our own design matrix and fit parameters β.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def save_fig(fig_id):
    plt.savefig(image_path(fig_id) + ".png", format='png')

def R2(y_data, y_model):
    return 1 - np.sum((y_data - y_model) ** 2) / np.sum((y_data - np.mean(y_data)) ** 2)
def MSE(y_data,y_model):
    n = np.size(y_model)
    return np.sum((y_data-y_model)**2)/n

x = np.random.rand(100)
y = 2.0+5*x*x+0.1*np.random.randn(100)


#  The design matrix now as function of a given polynomial
X = np.zeros((len(x),3))
X[:,0] = 1.0
X[:,1] = x
X[:,2] = x**2
# We split the data in test and training data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# matrix inversion to find beta
beta = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
print(beta)
# and then make the prediction
ytilde = X_train @ beta
print("Training R2")
print(R2(y_train,ytilde))
print("Training MSE")
print(MSE(y_train,ytilde))
ypredict = X_test @ beta
print("Test R2")
print(R2(y_test,ypredict))
print("Test MSE")
print(MSE(y_test,ypredict))
```

*

Exercise 4: mean values and variances in linear regression

This exercise deals with various mean values ad variances in linear regression method (here it may be useful to look up chapter 3, equation (3.8) of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer).

The assumption we have made is that there exists a function $f(\boldsymbol{x})$ and a normal distributed error $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$ which describes our data

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$$

We then approximate this function with our model from the solution of the linear regression equations (ordinary least squares OLS), that is our function f

is approximated by $\tilde{\boldsymbol{y}}$ where we minimized $(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$, with

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}.$$

The matrix $\boldsymbol{X}$ is the so-called design matrix.
aragraph!paragraph>paragraph>-0.5em

a) Show that the expectation value of $\boldsymbol{y}$ for a given element i

$$\mathbb{E}(y_i) = \mathbf{X}_{i,*}\,\beta,$$

and that its variance is

$$\mathrm{Var}(y_i) = \sigma^2.$$

Hence, $y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}, \sigma^2)$, that is $\boldsymbol{y}$ follows a normal distribution with mean value $\boldsymbol{X}\boldsymbol{\beta}$ and variance σ^2.

Solution. We can calculate the expectation value of $\boldsymbol{y}$ for a given element i

$$\mathbb{E}(y_i) = \mathbb{E}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}) + \mathbb{E}(\varepsilon_i) \;\; = \;\; \mathbf{X}_{i,*}\,\beta,$$

while its variance is

$$\begin{aligned}
\mathrm{Var}(y_i) = \mathbb{E}\{[y_i - \mathbb{E}(y_i)]^2\} \;\; &= \;\; \mathbb{E}(y_i^2) - [\mathbb{E}(y_i)]^2 \\
&= \mathbb{E}[(\mathbf{X}_{i,*}\,\beta + \varepsilon_i)^2] - (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 \\
&= \mathbb{E}[(\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 + 2\varepsilon_i \mathbf{X}_{i,*}\,\boldsymbol{\beta} + \varepsilon_i^2] - (\mathbf{X}_{i,*}\,\beta)^2 \\
&= (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 + 2\mathbb{E}(\varepsilon_i)\mathbf{X}_{i,*}\,\boldsymbol{\beta} + \mathbb{E}(\varepsilon_i^2) - (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 \\
&= \mathbb{E}(\varepsilon_i^2) \;\; = \;\; \mathrm{Var}(\varepsilon_i) \;\; = \;\; \sigma^2.
\end{aligned}$$

Hence, $y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}, \sigma^2)$, that is $\boldsymbol{y}$ follows a normal distribution with mean value $\boldsymbol{X}\boldsymbol{\beta}$ and variance σ^2 (not be confused with the singular values of the SVD).
aragraph!paragraph>paragraph>-0.5em

b) With the OLS expressions for the parameters $\boldsymbol{\beta}$ show that

$$\mathbb{E}(\boldsymbol{\beta}) = \boldsymbol{\beta}.$$

Solution.

$$\mathbb{E}(\boldsymbol{\beta}) = \mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}.$$

This means that the estimator of the regression parameters is unbiased.
aragraph!paragraph>paragraph>-0.5em

c) Show finally that the variance of $\boldsymbol{\beta}$ is

$$\mathrm{Var}(\boldsymbol{\beta}) \;\; = \;\; \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}.$$

Solution. The variance of $\boldsymbol{\beta}$ is

$$\begin{aligned}
\mathrm{Var}(\boldsymbol{\beta}) &= \mathbb{E}\{[\boldsymbol{\beta}-\mathbb{E}(\boldsymbol{\beta})][\boldsymbol{\beta}-\mathbb{E}(\boldsymbol{\beta})]^T\} \\
&= \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}-\boldsymbol{\beta}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}-\boldsymbol{\beta}]^T\} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\,\mathbb{E}\{\mathbf{Y}\,\mathbf{Y}^T\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1}-\boldsymbol{\beta}\,\boldsymbol{\beta}^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\,\{\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T+\sigma^2\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1}-\boldsymbol{\beta}\,\boldsymbol{\beta}^T \\
&= \boldsymbol{\beta}\,\boldsymbol{\beta}^T+\sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}-\boldsymbol{\beta}\,\boldsymbol{\beta}^T \;=\; \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1},
\end{aligned}$$

where we have used that $\mathbb{E}(\mathbf{Y}\mathbf{Y}^T)=\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T+\sigma^2\,\mathbf{I}_{nn}$. From $\mathrm{Var}(\boldsymbol{\beta})=\sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}$, one obtains an estimate of the variance of the estimate of the j-th regression coefficient: $\boldsymbol{\sigma}^2(\hat{\beta}_j)=\boldsymbol{\sigma}^2\sqrt{[(\mathbf{X}^T\mathbf{X})^{-1}]_{jj}}$. This may be used to construct a confidence interval for the estimates.

In a similar way, we can obtain analytical expressions for say the expectation values of the parameters $\boldsymbol{\beta}$ and their variance when we employ Ridge regression, allowing us again to define a confidence interval.

*

Exercise 5: Playing with nuclear masses

Finally, try now to write your own code (you can use the example the nuclear masses in the lecture slides on Regression and Getting started from Day1, that reads in the nuclear masses and compute the proton separation energies, the two-neutron and two-proton separation energies and finally the shell gaps for selected nuclei.

Finally, try to compute the Q-values for $\beta-$ decay for selected nuclei.

Solution. Let us study the Q values associated with the removal of one or two nucleons from a nucleus. These are conventionally defined in terms of the one-nucleon and two-nucleon separation energies. With the functionality in **pandas**, two to three lines of code will allow us to plot the separation energies. The neutron separation energy is defined as

$$S_n=-Q_n=BE(N,Z)-BE(N-1,Z),$$

and the proton separation energy reads

$$S_p=-Q_p=BE(N,Z)-BE(N,Z-1).$$

The two-neutron separation energy is defined as

$$S_{2n}=-Q_{2n}=BE(N,Z)-BE(N-2,Z),$$

and the two-proton separation energy is given by

$$S_{2p}=-Q_{2p}=BE(N,Z)-BE(N,Z-2).$$

Using say the neutron separation energies (alternatively the proton separation energies)

$$S_n = -Q_n = BE(N,Z) - BE(N-1,Z),$$

we can define the so-called energy gap for neutrons (or protons) as

$$\Delta S_n = BE(N,Z) - BE(N-1,Z) - (BE(N+1,Z) - BE(N,Z)),$$

or

$$\Delta S_n = 2BE(N,Z) - BE(N-1,Z) - BE(N+1,Z).$$

This quantity can in turn be used to determine which nuclei could be interpreted as magic or not. For protons we would have

$$\Delta S_p = 2BE(N,Z) - BE(N,Z-1) - BE(N,Z+1).$$

To calculate say the neutron separation we need to multiply our masses with the nucleon number A. The example here is for the neutron separation energies for the oxygen isotopes. Note the simple function we use to compute the neutron separation energies

```
# Her we pick the oyxgen isotopes
Nucleus = df.loc[lambda df: df.Z==8, :]
# drop cases with no number
Nucleus = Nucleus.dropna()
# Here we do the magic and obtain the neutron separation energies, one line of code!!
Nucleus['NeutronSeparationEnergies'] = Nucleus['Energies'].diff(+1)
```

If we want another isotope we need simply to change the Z value. For isotones, we fix simply the neutron number. Furthermore, if we wish to compute say the two-neutron separation energies of the oyxgen isotopes we need simply to write

```
# Her we pick the oyxgen isotopes
Nucleus = df.loc[lambda df: df.Z==8, :]
# drop cases with no number
Nucleus = Nucleus.dropna()
# Here we do the magic and obtain the neutron separation energies, one line of code!!
Nucleus['NeutronSeparationEnergies'] = Nucleus['Energies'].diff(+2)
```

Note the +2 in the function **diff(+2)**! Easy, isn't it? It is easy to change to two-proton separation energies. The full example here is for the neutron separation energies.

```
# Common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from pylab import plt, mpl
plt.style.use('seaborn')
mpl.rcParams['font.family'] = 'serif'


# Where to save the figures and data files
```

```
PROJECT_ROOT_DIR = "Results"
FIGURE_ID = "Results/FigureFiles"
DATA_ID = "DataFiles/"

if not os.path.exists(PROJECT_ROOT_DIR):
    os.mkdir(PROJECT_ROOT_DIR)

if not os.path.exists(FIGURE_ID):
    os.makedirs(FIGURE_ID)

if not os.path.exists(DATA_ID):
    os.makedirs(DATA_ID)

def image_path(fig_id):
    return os.path.join(FIGURE_ID, fig_id)

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

def save_fig(fig_id):
    plt.savefig(image_path(fig_id) + ".png", format='png')

infile = open(data_path("MassEval2016.dat"),'r')


# Read the experimental data with Pandas
Masses = pd.read_fwf(infile, usecols=(2,3,4,6,11),
              names=('N', 'Z', 'A', 'Element', 'Ebinding'),
              widths=(1,3,5,5,5,1,3,4,1,13,11,11,9,1,2,11,9,1,3,1,12,11,1),
              header=39,
              index_col=False)

# Extrapolated values are indicated by '#' in place of the decimal place, so
# the Ebinding column won't be numeric. Coerce to float and drop these entries.
Masses['Ebinding'] = pd.to_numeric(Masses['Ebinding'], errors='coerce')
Masses = Masses.dropna()
# Convert from keV to MeV.
Masses['Ebinding'] /= 1000
A = Masses['A']
Z = Masses['Z']
N = Masses['N']
Element = Masses['Element']
Energies = Masses['Ebinding']*A

df = pd.DataFrame({'A':A,'Z':Z, 'N':N,'Element':Element,'Energies':Energies})
# Her we pick the oyxgen isotopes
Nucleus = df.loc[lambda df: df.Z==8, :]
# drop cases with no number
Nucleus = Nucleus.dropna()
# Here we do the magic and obtain the neutron separation energies, one line of code!!
Nucleus['NeutronSeparationEnergies'] = Nucleus['Energies'].diff(+1)
print(Nucleus)
#MakePlot([Nucleus.A], [Nucleus.NeutronSeparationEnergies], ['b'], ['Neutron Separation Energy'],
save_fig('Nucleus')
plt.show()
```