# First day: Homework 3

**Data Analysis and Machine Learning**

Jan 21, 2020

## Day three and four exercises

The exercises here are somewhat longer and we expect to use at least two days on them.

**Exercise 1, Bias-Variance tradeoff and Bootstrap.** This exercise is a continuation of exercise 2 from the second homework set. In that exercise we computed the MSE and $R2$-score for the training data and the test data as functions of the complexity of a polynomial, that is the degree of a given polynomial.

One of the aims of that exercise was to reproduce Figure 2.11 of Hastie et al.

Our data is defined by $x \in [-3, 3]$ with a total of for example 100 data points.

```
np.random.seed()
n = 100
maxdegree = 14
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)
```

where $y$ is the function we want to fit with a given polynomial.

**Part (1a) Proving the bias-variance tradeoff.** Consider a dataset $\mathcal{L}$ consisting of the data $\mathbf{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$.

Let us assume that the true data is generated from a noisy model

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

Here $\epsilon$ is normally distributed with mean zero and standard deviation $\sigma^2$.

In our derivation of the ordinary least squares method we defined then an approximation to the function $f$ in terms of the parameters $\boldsymbol{\beta}$ and the design matrix $\boldsymbol{X}$ which embody our model, that is $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$.

The parameters $\boldsymbol{\beta}$ are in turn found by optimizing the means squared error via the so-called cost function

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right].$$

Show that you can rewrite this as

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_{i}(f_i-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_{i}(\tilde{y}_i-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2.$$

Explain what the terms mean, which one is the bias and which one is the variance and discuss their interpretations.

**Part (1b) Adding Bootstrap and Bias-Variance Tradeoff.** Add now bootstrapping as discussed in the Regression lectures (scroll down to the bias-variance code). Add also the expressions for the bias and the variance as discussed above.

Discuss the bias and variance tradeoff as function of your model complexity (the degree of the polynomial) and the number of data points, and possibly also your training and test data.

Try to make a figure similar to Fig. 2.11 of Hastie et al. You should include an analysis of the bias and variance for the test results. Figure 2.11 displays only the test and training MSEs while indicating regions of low/high bias and variance. You will most likely not get an equally smooth curve! Note also that when you calculate the bias, in all applications you don't know the function values $f_i$. You would hence replace them with the actual data points $y_i$.

**Exercise 2, Linear Regression for a two-dimensional function.** This is a longer exercise and the aim 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 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 established the model and the method, we will employ resamling like the bootstrap from the previous exercise in order to perform a proper assessment of our models. We will also study in detail the so-called Bias-Variance trade off.

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

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

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 (or bootstrap) as resampling technique. 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 set of fixed values for $x$ and $y$ with a given step size. We will fit a function (for example a polynomial) of $x$ and $y$. Thereafter we will repeat much of the same procedure using the Ridge and Lasso regression methods, introducing thus a dependence on the bias (penalty) $\lambda$.

Finally 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.

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()
```

**(2a) Ordinary Least Square on the Franke function with resampling.** We will generate our own dataset for a function FrankeFunction$(x, y)$ with $x, y \in [0, 1]$. 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 $\mathcal{N}(\prime, \infty)$.

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. You can use **scikit-learn** as well.

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

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

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

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

To set up the design matrix, the following code can be used

```
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


def create_X(x, y, n ):
if len(x.shape) > 1:
x = np.ravel(x)
```

```
y = np.ravel(y)

N = len(x)
l = int((n+1)*(n+2)/2) # Number of elements in beta
X = np.ones((N,l))

for i in range(1,n+1):
q = int((i)*(i+1)/2)
for k in range(i+1):
X[:,q+k] = (x**(i-k))*(y**k)

return X


# Making meshgrid of datapoints and compute Franke's function
n = 5
N = 1000
x = np.sort(np.random.uniform(0, 1, N))
y = np.sort(np.random.uniform(0, 1, N))
z = FrankeFunction(x, y)
X = create_X(x, y, n=n)
```

**Part (2b) Resampling techniques, adding more complexity.** Perform a resampling of the data where you split the data in training data and test data. Here you can write your own function or use the function for splitting training data provided by **Scikit-Learn**. This function is called *train_test_split*. You should also renormalize your data.

It is normal in essentially all Machine Learning studies to split the data in a training set and a test set (sometimes also an additional validation set). There is no explicit recipe for how much data should be included as training data and say test data. An accepted rule of thumb is to use approximately 2/3 to 4/5 of the data as training data.

Use then the bootstrap code you developed in the previous exercise to resample your data and evaluate again the MSE function resulting from the test data.

**Part (2c): Bias-variance tradeoff.** With a code which does OLS and includes bootstrap we will now discuss the bias-variance tradeoff in the context of continuous predictions such as regression. However, many of the intuitions and ideas discussed here also carry over to classification tasks and basically all Machine Learning algorithms.

Use the code from exercise 1 above and implement the bootstrap resampling and perform a bias-variance tradeoff analysis like you did in exercise 1.

**Part (2d): Ridge Regression on the Franke function with resampling.** Write your own code for the Ridge method, either using matrix inversion or the singular value decomposition ir use **scikit-learn** Perform the same analysis as in the previous three steps (for the same polynomials and include resampling techniques) but now for different values of $\lambda$. Compare and analyze your results with those obtained in parts 2a-2c). Study the dependence on $\lambda$.

Study also the bias-variance tradeoff as function of various values of the parameter $\lambda$. Comment your results.

**Part (2e): Lasso Regression on the Franke function with resampling.** This part is essentially a repeat of the previous ones, but now with Lasso regression. Write either your own code or use the functionalities of **Scikit-Learn** (recommended). Give a critical discussion of the three methods and a judgement of which model fits the data best.