Homework 4

Contents

References

- Lectures 13-16 (inclusive).

Instructions

- Type your name and email in the “Student details” section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

▶ Show code cell source

Student details

Skip to main content

- **Last Name:**
- **Email:**

First, make sure that this dataset is visible from this Jupyter notebook. You may achieve this by either:

- Downloading the data file and then manually upload it on Google Colab. The easiest way is to click on the folder icon on the left of the browser window and click on the upload button (or drag and drop the file). Some other options are here.
- Downloading the file to the working directory of this notebook with this code:

```
url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/lectureboo
download(url)
```

It's up to you what you choose to do. If the file is in the right place, the following code should work:

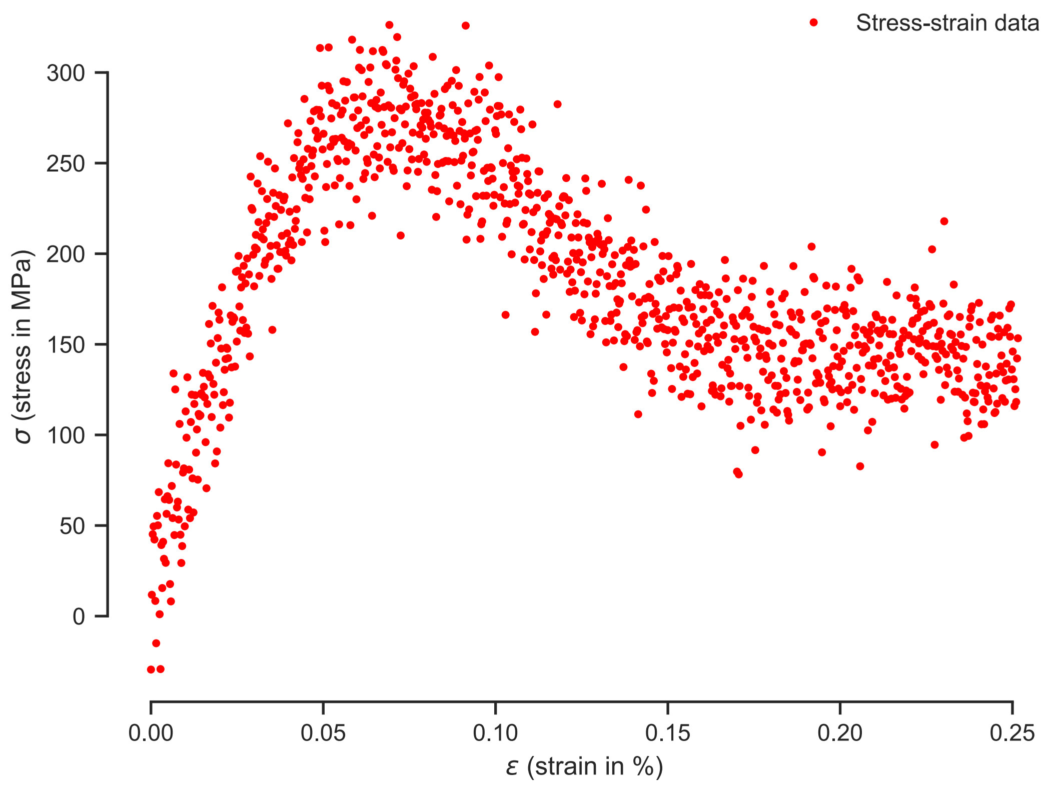
```
data =  np.loadtxt('stress_strain.txt')
```

The dataset was generated using a molecular dynamics simulation of a plastic material (thanks to Professor Alejandro Strachan for sharing the data!). Specifically, Strachan's group did the following:

- They took a rectangular chunk of the material and marked the position of each one of its atoms;
- They started applying a tensile force along one dimension. The atoms are coupled together through electromagnetic forces, and they must all satisfy Newton's law of motion.
- For each value of the applied tensile force, they marked the stress (force be unit area) in the middle of the material and the corresponding strain of the material (percent elongation in the pulling direction).
- Eventually, the material entered the plastic regime and broke. Here is a visualization of the data:

▶ Show code cell source

Skip to main content



Note that you don't necessarily get a unique stress for each particular value of the strain. This is because the atoms are jiggling around due to thermal effects. So, there is always this "jiggling" noise when measuring the stress and the strain. We want to process this noise to extract what is known as the stress-strain curve of the material. The stress-strain curve is a macroscopic property of the material, affected by the fine structure, e.g., the chemical bonds, the crystalline structure, any defects, etc. It is a required input to the mechanics of materials.

Part A - Fitting the stress-strain curve in the elastic regime

The very first part of the stress-strain curve should be linear. It is called the *elastic regime*. In that region, say $\epsilon < \epsilon_l = 0.04$, the relationship between stress and strain is:

$$\sigma(\epsilon) = E\epsilon.$$

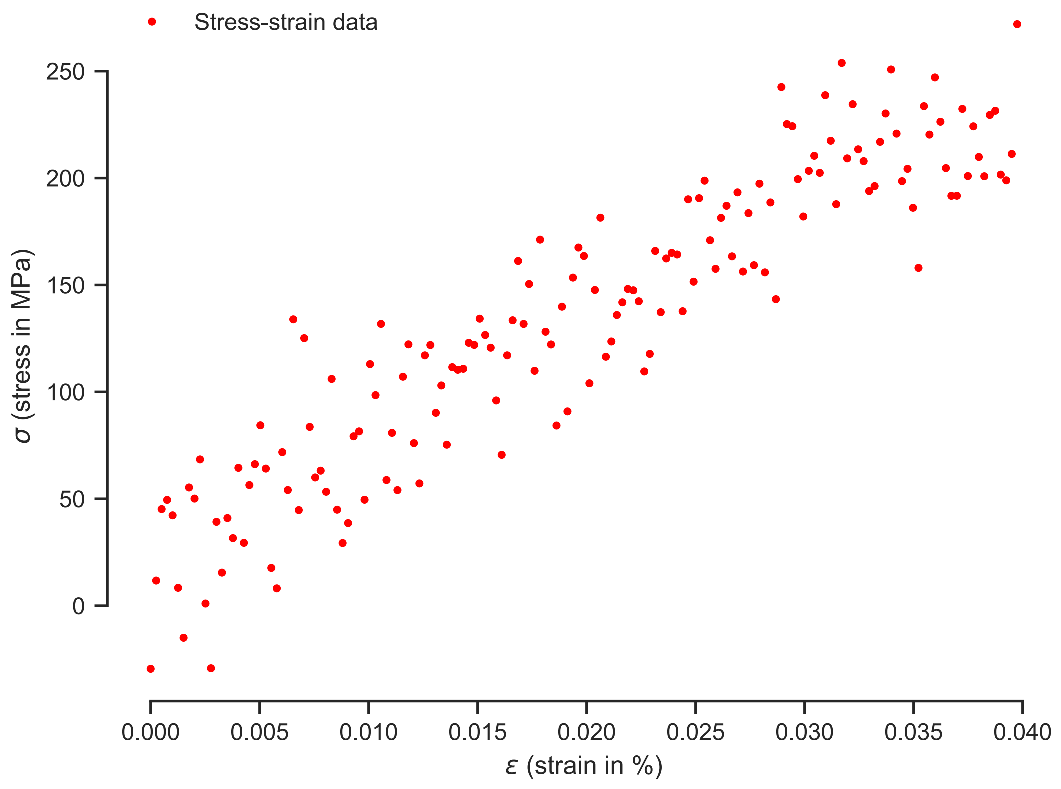
The constant E is known as the *Young modulus* of the material. Assume that you measure ϵ without noise, but your measured σ is noisy.

Subpart A.I

First, extract the relevant data for this problem, split it into training and validation datasets, and visualize the training and validation datasets using different colors.

```
# The point at which the stress-strain curve stops being linear
epsilon_l = 0.04
# Relevant data (this is nice way to get the linear part of the stresses and straints)
x_rel = x[x < 0.04]
y_rel = y[x < 0.04]

# Visualize to make sure you have the right data
plt.figure()
plt.plot(
    x_rel,
    y_rel,
    'ro',
    markersize=2,
    label='Stress-strain data'
)
plt.xlabel('$\epsilon$ (strain in %)')
plt.ylabel('$\sigma$ (stress in MPa)')
plt.legend(loc='best', frameon=False)
sns.despine(trim=True);
```



Skip to main content

Hint: You may use sklearn.model_selection.train_test_split if you wish.

```
# Split the data into training and validation datasets
# Hint: Consult the lecture notes
x_train, y_train, x_valid, y_valid = # Your code
```

Use the following to visualize your split:

```
plt.figure()
plt.plot(
    x_train,
    y_train,
    'ro',
    markersize=2,
    label='Training data'
)
plt.plot(
    x_valid,
    y_valid,
    'bx',
    markersize=2,
    label='Validation data'
)
plt.xlabel('$\epsilon$ (strain in %)')
plt.ylabel('$\sigma$ (stress in MPa)')
plt.legend(loc='best', frameon=False)
sns.despine(trim=True);
```

Subpart A.II

Perform Bayesian linear regression with the evidence approximation to estimate the noise variance and the hyperparameters of the prior.

```
# Your code here
```

Subpart A.III

Calculate the mean square error of the validation data.

```
# your code here
```

Skip to main content

Subpart A.IV

Make the observations vs predictions plot for the validation data.

```
# your code here
```

Subpart A.V

Compute and plot the standardized errors for the validation data.

```
# your code here
```

Subpart A.VI

Make the quantile-quantile plot of the standardized errors.

```
# your code here
```

Subpart A.VII

Visualize your epistemic and the aleatory uncertainty about the stress-strain curve in the elastic regime.

```
# your code here
```

Subpart A. VIII

Visualize the posterior of the Young modulus E conditioned on the data.

```
# your code here
```

Skip to main content

Subpart A.IX

Take five samples of stress-strain curve in the elastic regime and visualize them.

```
# your code here
```

Subpart A.X

Find the 95% centered credible interval for the Young modulus E.

```
# your code here
```

Subpart A.XI

If you had to pick a single value for the Young modulus E, what would it be and why?

```
# your code here
```

Your answer here

Part B - Estimate the ultimate strength

The pick of the stress-strain curve is known as the ultimate strength. We want to estimate it.

Subpart B.I - Extract training and validation data

Extract training and validation data from the entire dataset.

```
# your code here - Repeat as many text and code blocks as you like
x_train, y_train, x_valid, y_valid = # Your code
```

Use the following to visualize your split:

Skip to main content

```
plt.figure()
plt.plot(
    x_train,
    y_train,
    'ro',
    markersize=2,
    label='Training data'
)
plt.plot(
    x_valid,
    y_valid,
    'bx',
    markersize=2,
    label='Validation data'
)
plt.xlabel('$\epsilon$ (strain in %)')
plt.ylabel('$\sigma$ (stress in MPa)')
plt.legend(loc='best', frameon=False)
sns.despine(trim=True);
```

Subpart B.II - Model the entire stress-strain relationship.

To do this, we will set up a generalized linear model to capture the entire stress-strain relationship. Remember, you can use any model you want as soon as:

- It is linear in the parameters to be estimated,
- It has a well-defined elastic regime (see Part A).

I am going to help you set up the right model. We will use the Heavide step function to turn on or off models for various ranges of ϵ. The idea is quite simple: We will use a linear model for the elastic regime, and we are going to turn to a non-linear model for the non-linear regime. Here is a model that has the right form in the elastic regime and an arbitrary form in the non-linear regime:

$$f(\epsilon; E, \mathbf{w}_g) = E\epsilon\left[(1 - H(\epsilon - \epsilon_l)\right] + g(\epsilon; \mathbf{w}_g)H(\epsilon - \epsilon_l),$$

where

$$H(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{otherwise} \end{cases}$$

Skip to main content

and g is any function linear in the parameters $\mathbf{w}_g$.

You can use any model you like for the non-linear regime, but let's use a polynomial of degree d:

$$g(\epsilon) = \sum_{i=0}^d w_i\epsilon^i.$$

The full model can be expressed as:

$$\begin{aligned} f(\epsilon) &= \begin{cases} h(\epsilon) = E\epsilon,\ \epsilon < \epsilon_l, \\ g(\epsilon) = \sum_{i=0}^d w_i\epsilon^i, \epsilon \geq \epsilon_l \end{cases} \\ &= E\epsilon\left(1 - H(\epsilon-\epsilon_l)\right) + \sum_{i=0}^d w_i\epsilon^i H(\epsilon-\epsilon_l). \end{aligned}$$

We could proceed with this model, but there is a small problem: It is discontinuous at $\epsilon = \epsilon_l$. This is unphysical. We can do better than that!

To make the model nice, we force the h and g to match up to the first derivative, i.e., we demand that:

$$\begin{aligned} h(\epsilon_l) &= g(\epsilon_l) \\ h'(\epsilon_l) &= g'(\epsilon_l). \end{aligned}$$

We include the first derivative because we don't have a kink in the stress-strain. That would also be unphysical. The two equations above become:

$$\begin{aligned} E\epsilon_l &= \sum_{i=0}^d w_i\epsilon_l^i \\ E &= \sum_{i=1}^d i w_i\epsilon_l^{i-1}. \end{aligned}$$

We can use these two equations to eliminate two weights. Let's eliminate w_0 and w_1. All you have to do is express them in terms of E and $w_2,\dots,w_d$. So, there remain d parameters to estimate. Let's get back to the stress-strain model.

Skip to main content

$$f(\epsilon) = E\epsilon\left(1 - H(\epsilon - \epsilon_l)\right) + \sum_{i=0}^d w_i\epsilon^i H(\epsilon - \epsilon_l).$$

We can now use the expressions for w_0 and w_1 to rewrite this using only all the other parameters. I am going to spare you the details. The result is:

$$f(\epsilon) = E\epsilon + \sum_{i=2}^d w_i\left[(i-1)\epsilon_l^i - i\epsilon\epsilon_l^{i-1} + \epsilon^i\right]H(\epsilon - \epsilon_l).$$

Okay. This is still a generalized linear model. This is nice. Write code for the design matrix:

```
# Complete this code to make your model:
def compute_design_matrix(Epsilon, epsilon_l, d):
    """Compute the design matrix for the stress-strain curve problem.

    Arguments:
        Epsilon     -     A 1D array of dimension N.
        epsilon_l   -     The strain signifying the end of the elastic regime.
        d           -     The polynomial degree.

    Returns:
        A design matrix N x d
    """
    # Sanity check
    assert isinstance(Epsilon, np.ndarray)
    assert Epsilon.ndim == 1, 'Pass the array as epsilon.flatten(), if it is two dimer
    n = Epsilon.shape[0]
    # The design matrix:
    Phi = np.ndarray((n, d))
    # The step function evaluated at all the elements of Epsilon.
    # You can use it if you want.
    Step = np.ones(n)
    Step[Epsilon < epsilon_l] = 0
    # Build the design matrix
    Phi[:, 0] = # Your code here
    for i in range(2, d+1):
        Phi[:, i-1] = # Your code here
    return Phi
```

Visualize the basis functions here:

```
d = 4
eps = np.linspace(0, x.max(), 100)
```

Skip to main content

```
ax.plot(eps, Phis)
ax.set_xlabel('$\epsilon$ (strain in %))')
ax.set_ylabel('$\phi_i(\epsilon)$')
sns.despine(trim=True);
```

Subpart B.III

Fit the model using automatic relevance determination and demonstrate that it works well by doing everything we did above (MSE, observations vs. predictions plot, standardized errors, etc.).

```
# Your code here - Use as many blocks as you need!
model = # Just call the resulting model "model"
```

Subpart B.IV

Visualize the epistemic and aleatory uncertainty in the stess-strain relation.

```
# Your code here
```

Subpart B.V - Extract the ultimate strength

Now, you will quantify your epistemic uncertainty about the ultimate strength. The ultimate strength is the maximum of the stress-strain relationship. Since you have epistemic uncertainty about the stress-strain relationship, you also have epistemic uncertainty about the ultimate strength.

Do the following:

- Visualize the posterior of the ultimate strength.
- Find a 95% credible interval for the ultimate strength.
- Pick a value for the ultimate strength.

Hint: To characterize your epistemic uncertainty about the ultimate strength, you would have to

Skip to main content

- Define a dense set of strain points between 0 and 0.25.
- Repeatedly:

- Sample from the posterior of the weights of your model
- For each sample, evaluate the stresses at the dense set of strain points defined earlier
- For each sampled stress vector, find the maximum. This is a sample of the ultimate strength.

```
# Enter your code here
```

In this problem, we will need this dataset. The dataset was kindly provided to us by Professor Davide Ziviani. As before, you can either put it on your Google Drive or just download it with the code segment below:

```
url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/lectureboo
download(url)
```

Note that this is an Excel file, so we need pandas to read it. Here is how:

```
import pandas as pd
data = pd.read_excel('compressor_data.xlsx')
data
```

The data are part of an experimental study of a variable-speed reciprocating compressor. The experimentalists varied two temperatures, T_e and T_c (both in C), and they measured various other quantities. We aim to learn the map between T_e and T_c and measure Capacity and Power (both in W). First, let's see how you can extract only the relevant data.

```
# Here is how to extract the T_e and T_c columns and put them in a single numpy array
x = data[['T_e','T_c']].values
x
```

```
# Here is how to extract the Capacity
y = data['Capacity'].values
y
```

Skip to main content

Fit the following multivariate polynomial model to **both the Capacity and the Power**:

$$y = w_1 + w_2T_e + w_3T_c + w_4T_eT_c + w_5T_e^2 + w_6T_c^2 + w_7T_e^2T_c + w_8T_eT_c^2 + w_9T_e^3 + w_{10}T_c$$

where ϵ is a Gaussian noise term with unknown variance.

Hints:

- You may use sklearn.preprocessing.PolynomialFeatures to construct the design matrix of your polynomial features. Do not program the design matrix by hand.
- You should split your data into training and validation and use various validation metrics to ensure your models make sense.
- Use ARD Regression to fit any hyperparameters and the noise.

Part A - Fit the capacity

Subpart A.I

Please don't just fit. Split in training and test and use all the usual diagnostics.

```
# your code here - Repeat as many text and code blocks as you like
```

Subpart A.II

What is the noise variance you estimated for the Capacity?

```
# your code here
```

Subpart A.III

Which features of the temperatures (basis functions of your model) are the most important for

Skip to main content

```
# your code here
```

Part B - Fit the Power

Subpart B.I

Please don't just fit. Split in training and test and use all the usual diagnostics.

```
# your code here - Repeat as many text and code blocks as you like
```

Subpart B.II

What is the noise variance you estimated for the Power?

```
# your code here
```

Subpart B.III

Which features of the temperatures (basis functions of your model) are the most important for predicting the Power?

```
# your code here
```

On January 28, 1986, the Space Shuttle Challenger disintegrated after 73 seconds from launch. The failure can be traced to the rubber O-rings, which were used to seal the joints of the solid rocket boosters (required to force the hot, high-pressure gases generated by the burning solid propellant through the nozzles, thus producing thrust).

The performance of the O-ring material was sensitive to the external temperature during launch. This dataset contains records of different experiments with O-rings recorded at various times between 1981 and 1986. Download the data the usual way (either put them on Google

```
url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/lecturebo
download(url)
```

Even though this is a CSV file, you should load it with pandas because it contains some special characters.

```
raw_data = pd.read_csv('challenger_data.csv')
raw_data
```

The first column is the date of the record. The second column is the external temperature of that day in degrees F. The third column labeled `Damage Incident` has a binary coding (0=no damage, 1=damage). The very last row is the day of the Challenger accident.

We will use the first 23 rows to solve a binary classification problem that will give us the probability of an accident conditioned on the observed external temperature in degrees F. Before proceeding to the data analysis, let's clean the data up.

First, we drop all the bad records:

```
clean_data_0 = raw_data.dropna()
clean_data_0
```

We also don't need the last record. Remember that the temperature on the day of the Challenger accident was 31 degrees F.

```
clean_data = clean_data_0[:-1]
clean_data
```

Let's extract the features and the labels:

```
x = clean_data['Temperature'].values
x
```

```
y = clean_data['Damage Incident'].values.astype(np.float)
y
```

Skip to main content

Part A - Perform logistic regression

Perform logistic regression between the temperature (x) and the damage label (y). Refrain from validating because there is little data. Just use a simple model so that you don't overfit.

```
# your code here - Repeat as many text and code blocks as you like
```

Part B - Plot the probability of damage as a function of temperature

Plot the probability of damage as a function of temperature.

```
# your code here
```

Part C - Decide whether or not to launch

The temperature on the day of the Challenger accident was 31 degrees F. Start by calculating the probability of damage at 31 degrees F. Then, use formal decision-making (i.e., define a cost matrix and make decisions by minimizing the expected loss) to decide whether or not to launch on that day. Also, plot your optimal decision as a function of the external temperature.

```
# your code here - Repeat as many text and code blocks as you like
```

Previous
Homework 3

Next
Homework 5