INF554 - MACHINE LEARNING I
ÉCOLE POLYTECHNIQUE
Lab 1: Introduction to the Machine Learning Pipeline

1 Introduction

The goal of this lab is to demonstrate the *machine learning pipeline*. We give a step by step overview of a typical machine learning task, and at each step describe the subtasks that need to be performed. For further information regarding the concepts relevant to this lab, see the lecture slides.

2 The Task

Some data is available in the `data` folder. This data consists of rows of measurements associated with cell growth in Scots pine trees monitored in the commune of Walscheid in France. Each row corresponds to the data of one week. The features are the week number, and the average measured temperature and soil moisture over that week. The final column denotes the number of new cells (known as tracheids) measured during that week (the number is an average over measurements from several trees).

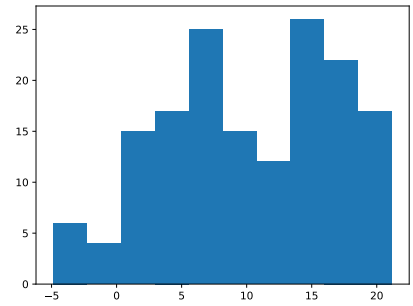
Counting cells involves manually extracting micro-core samples from the tree, and counting the cells under a microscope. Therefore it would be time saving and beneficial if a computational model was constructed, such that an estimate of growth could be made automatically given the environmental measurements such as temperature and daylight hours (which are easily and automatically obtainable). Furthermore, the model could be analyzed for greater understanding of growth drivers, and we could also build a forecasting model.

3 Implementation of a Machine Learning Pipeline

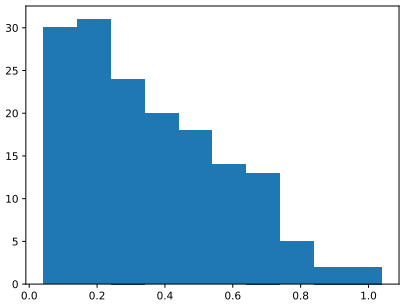
In this section, a machine learning pipeline will be implemented to load and preprocess the data, and from this data to build and evaluate a regression model. Each of the following subsections include tasks to be completed. In each case, you will need to complete code in the Python script `main.py`. The data is given in the `data/` folder. To run the pipeline, simply run the script.

3.1 Loading and inspecting the data

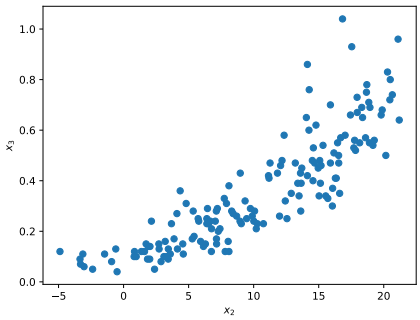
Have a look at the code in `main.py`. The training data is loaded and stored in `X` and `y`, respectively. The distribution of each variable can be inspected by, for example, using MATPLOTLIB's `hist` function. Notice/recall that in Python the first feature is indexed at `0`, whereas typically we refer to X_1 in mathematical notation. Therefore the second feature is indexed at `1`. The result should look like that in Figure 1a.



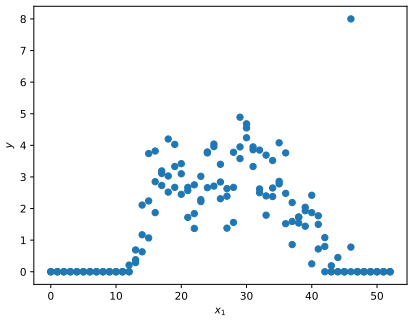
(a) Histogram of feature X_2



(b) Histogram of feature X_3



(c) Plot of X_2 vs X_3



(d) Plot of X_1 vs Y

Figure 1: Inspecting the data.

The result is shown in Figure 1c.

Task 1
Build the histogram for all other variables, then also plot X_1 (week number) versus the target. Hint: you can check your results against those in Figure 1.

Figure 1d shows the correlation between X_1 (week number) and the target variable (number of cells). We can see how the growth season starts at about the 12th week of the year, peaks in the summer months, before ceasing in the final months of the year.

3.2 Preprocessing

Preprocessing is a fundamental step in the machine learning pipeline. In practice, it takes up most of the time. It can involve cleaning the data, dealing with missing values, removing outliers, dimensionality reduction, feature selection and feature engineering. Usually preprocessing is guided by the data exploration process (inspecting the data). For example, there appears to be at outlier in the target column, visible in Figure 1d, possible caused by an '8' being recorded instead of a '0'. We will come back to this later at evaluation time.

In the data exploration phase it was also noticeable that the attributes were of different scales. A common technique is to standardize each attribute to mean 0 and standard deviation 1 so that each variable will be considered equally.

Task 2
Standardize the data of each of the input attributes. Hint: NumPy has a function `mean`. Calling `mean(X,axis=0)` will return a vector of means, one for each column. The function `std` can be used in a similar way for the standard deviation.

3.3 Building a model

A standard approach for regression tasks (such as this one, where the target label is continuous) is *ordinary least squares*; the estimator for which is given below in Eq. (1). However, this is a linear model, yet we observed in Figure 1d that the relationship of the X_1 feature is non-linear with respect Y. To turn a linear model into a non-linear one, we can use polynomial *basis functions* to create a new feature space, such that by fitting a linear model to this new feature space we obtain a non-linear model. This has been included in the preprocessing[1] via the function `phi`, where the second parameter indicates the degree to the polynomial. Using degree 2 would produce a new feature space

$$\mathbf{z} = \phi(\mathbf{x}) = [1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2]$$

for the i-th instance. For degree one, we have the original feature space,

$$\mathbf{z} = \phi(\mathbf{x}) = [1, x_1, x_2, x_3]$$

except that we have added a column of 1s to the new input feature space so that the intercept/bias does not need to be calculated separately.

The ordinary least squares estimator of coefficients is

$$\hat{\boldsymbol{\beta}} = (\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{Z}^\top \mathbf{y} \tag{1}$$

(the 'squares' refer to those in Eq. (2)).

Task 3
Implement ordinary least squares in Python, to obtain a vector of coefficients. Hint: Since Python 3.6 you can use the `@` symbol for matrix multiplication. You should also use the `inv` function, as imported in the code provided. Print out the coefficients using the `print` function.

Having $\hat{\boldsymbol{\beta}}$, we now have a model, for which we can make new predictions for any new test point;

$$\hat{y} = \hat{\boldsymbol{\beta}} \mathbf{z}$$

Ques 1
Derive $\hat{\boldsymbol{\beta}}$, by minimizing Eq. (2).

Ques 2
Think of something that could go wrong using `inv` as suggested above.

3.4 Loading and processing the *test* data

We now have a model that can supply predictions given new data instances, i.e., given a new set of measurements, we can estimate the number of cells of new growth for that week. But before deploying

[1] It is a matter of perspective whether this step is considered part of building the model, or data preprocessing

a model, we should first test it to know how useful it is. To test the model we first must obtain and process *test data*. It is important to conduct an identical preprocessing on the test data as the train data. In this case, standardization, feature expansion, and adding of a column of ones. The true labels of the test data should be held aside and not used in any process except for model evaluation.

> **Task 4**
> Load and preprocess the test data (`data_test.csv`) as done for the train data, but use different variable names (for example, `X_test` and `y_test`). Note that the preprocessing is not recalculated on the test data, we standardize using the same mean and standard deviation as estimated from the training data.

3.5 Evaluation and comparison

We now evaluate the quality of predictions. For this task we use the *mean squared error* as the loss metric, over N examples as

$$\mathsf{MSE}(\hat{\boldsymbol{\beta}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

> **Task 5**
> Implement the MSE calculation as Python code. This involves comparing `y_test` and `y_pred`, such that `mse = MSE(y_test,y_pred)`. Print out the value obtained. Hints: the number of examples in a vector can be obtained by the `len` function; the `sum` function in NumPy will sum the values of a vector.

> **Task 6**
> Additionally calculate the MSE where all predictions for test examples are simply the mean of Y in the training data.

So far, you obtain something like the following (`w` values have been rounded).

```
w =  [ 1.902  0.117  0.691  0.031 -0.676  0.24   0.031]
MSE on test data   0.546191331548
MSE baseline       1.7691746529
```

> **Task 7**
> What is the change to MSE if we replace the outlier (see Section 3.2) with a different value (e.g., 0)? By making other changes in the preprocessing step, can you improve on the best result so far?

> **Ques 3**
> Outliers can have a powerful influence on model accuracy. Explain this influence with regrard to Eq. (2).

Why is it important to use a test set for evaluation and not just simply measure the MSE on the training data that we had originally? The answer is to avoid *overfitting*. Overfitting can be demonstrated by comparing results on the training and test sets for different parameter configurations. Figure 2 shows what happens when we vary the degree of the polynomial expansion. On the test data we can achieve better and better results with more complexity, but in reality, anything more than a degree 5 polynomial is clearly overfitting, and the model is worse on new data that it hasn't seen before.

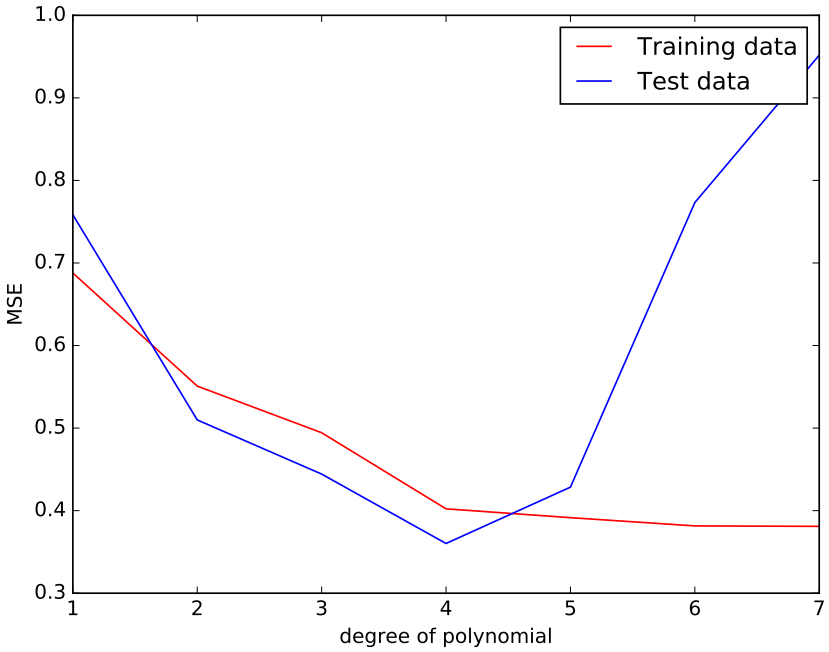


Figure 2: MSE with respect to different degrees of polynomials

Task 8

(Bonus task) Reproduce the results of Figure 2. Hints: put the test procedure in a loop, incrementing the degree of polynomial in the expansion each time, then use plotting functions as demonstrated in the earlier tasks; at each iteration of the loop you will need to obtain predictions on both the training and test data using your `MSE` function.

Task 9

(Bonus task) Apply the regression task using only the two principal components (i.e., apply PCA before the regression). You may implement PCA yourself or use `sklearn.decomposition.PCA` from the Scikit-Learn framework.