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

Christos Giatsidis, Jesse Read, Nikolaos Tziortziotis and Michalis Vazirgiannis

September 18, 2017

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 several times a week, 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 (which are easily and automatically obtainable), without having to manually extract samples. Furthermore, the model could be analyzed for greater understanding of growth drivers. In this lab we will build such a 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 files `main.py` and `tools.py` associated with this lab. It operates on the accompanying data supplied in the files `data_train.csv` and `data_test.csv` located in the `data/` folder. To run the pipeline, simply type `python main.py` on the command line.

3.1 Loading and inspecting the data

The training data is loaded with the following Python code. Note that the final column contains the *target* variable (i.e., *label*, or output), and the other columns contain the *features* (i.e., input attributes). These are stored in variables `X` and `y`, respectively.

```
# Load the data

data = loadtxt('data/data_train.csv', delimiter=',')

# Prepare the data

X = data[:,0:-1]
y = data[:,-1]
```

The distribution of each variable can be inspected by using Matplotlib's `hist` function to create a histogram. The following code demonstrates how to create and show a histogram of 10 bins, from the data of the second feature (temperature).

```
# Inspect the data

figure()
hist(X[:,1], 10)
show()
```

Notice 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.

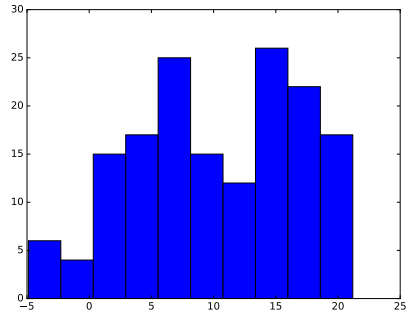
The data can also be plotted for visual inspection. This is demonstrated for two attributes.

```
figure()
plot(X[:,1],X[:,2], 'o')
xlabel('x1')
ylabel('x2')
show()
```

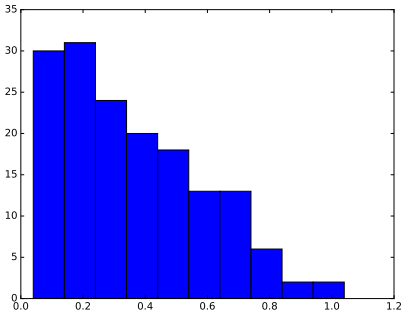
The result is shown in Figure 1c.

> **Task 1**
> Inspect the distribution of other variables. First, build the histogram for all other variables. Hint: The histogram of the second feature should look like that of Figure 1b. Secondly, plot X_1 (week number) versus the target Y. Hint: this plot should look like that of Figure 1d.

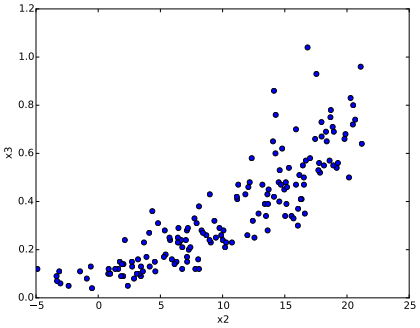
Figure 1d shows the correlation between X_1 (week number) and the target variable (number of cells). We can see how the growth season starts at about the 12th week of the year, peaks in the summer months, before ceasing in the final months of the year.



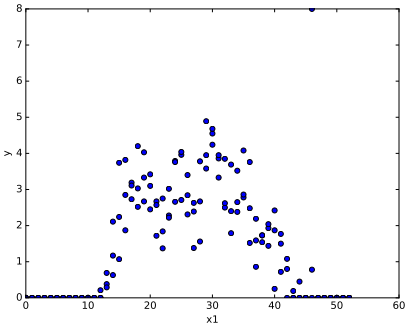
(a) Histogram of feature X_2



(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.

3.2 Preprocessing

Preprocessing is a fundamental step in the machine learning pipeline. 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.

In the next section a linear model will be used. However, we observed in Figure 1d that the relationship of at least the X_1 feature is non-linear with respect Y. This motivates us to use polynomial basis functions to expand the feature space in a way that it produces a new feature space that can be fitted well with a linear model. This has been included in the preprocessing via the function `poly_exp` (in `tools.py`), for which the second parameter indicates the degree to the polynomial. Using degree 2 would produce a new feature space

$$\mathbf{z}_i = [x_1, x_2, x_3, x_1^2, x_2^2, x_3^2]$$

for the i-th instance. Finally, a column of 1s is added to the new input feature space so that the intercept of a linear model does not need to be calculated separately.

3.3 Building a model

This is a regression problem, since the target label is continuous. We will use ordinary least squares regression. The ordinary least squares estimator of coefficients is

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

> **Task 3**
> Implement ordinary least squares in Python using the `dot` function in NumPy, so as to obtain a vector of coefficients. Print out the coefficients using the `print` function. Hint: Since we inserted an extra column of `1`s the intercept is calculated automatically into `w[0]` (where `w` is whatever name you chose for the vector of coefficients).

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 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

The estimator $\hat{\beta}$ can now be evaluated using the test data. Namely, we make predictions $\hat{\mathbf{y}} = \mathbf{Z}\hat{\beta}$, using the Python code `y_pred = dot(Z_test,w)` (where `w` is the name you chose for the coefficient vector). We then evaluate the predictions. For this task we use the mean squared error as the loss metric:

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

Task 5
Implement the MSE calculation as Python code (an empty function definition can be found in `tools.py`). 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?

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.

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.

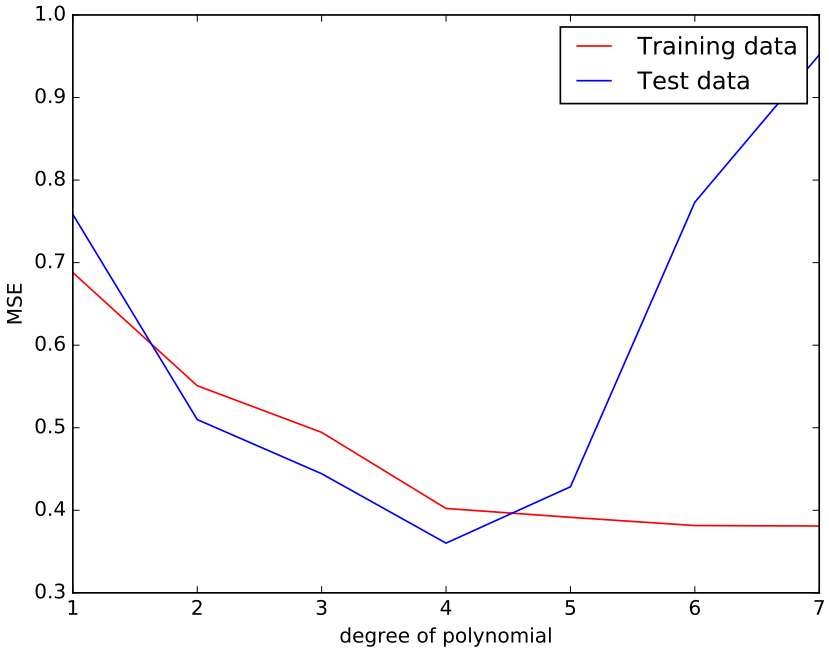


Figure 2: MSE with respect to different degrees of polynomials