

# CS342 Machine Learning: Lab #1

Labs on January 14 & 15, 2016

*Week 1 of Term 2*

## Office Hours:

CS 3.07, Monday & Friday 10:00-11:00

Instructor: **Dr Theo Damoulas** (T.Damoulas@warwick.ac.uk)

Tutors: **Helen McKay** (H.McKay@warwick.ac.uk), **Shan Lin** (Shan.Lin@warwick.ac.uk)

In this first Lab we will familiarise ourselves with Python and some of the main libraries, and then run a simple linear regression with ordinary least squares on the Olympic dataset from the R&G book. Refer to the module website <https://www2.warwick.ac.uk/fac/sci/dcs/teaching/modules/cs342/> for the dataset, related background material, and module slides. If you are already comfortable with Python please proceed to the Machine Learning component of the Lab. If you do not manage to complete the Lab in the pre-designated hour then please complete it in your own time as it will help you better understand some of the course material.

## Introduction

All the machines in the Lab have the main Python libraries and environment you will need today. If you are using your own machine, you can install the **Anaconda** distribution. <https://www.continuum.io/downloads> which includes most of the libraries we will need for the Labs.

You will need an editor/IDE to develop your codes and if you don't have a preference we recommend using the **iPython Notebook** which is available on all DCS machines. The Jupyter iPython Notebook can be launched by typing any of following 3 commands in a Terminal (Linux or Mac) or Command Prompt (Windows): (**Note: To launch on DCS machines use jupyter-notebook**).

```
ipython notebook or jupyter notebook or jupyter-notebook
```

To start a new notebook select the 'New' button in the top right corner and select Notebooks-**Python2**.



The following image shows a basic interface of iPython Notebook. The interface splits into cells. Python code can be programmed in each cell. You can choose to run your python code in a single cell by (**Shift-Enter** / **Ctrl-Enter**) or the whole notebook by clicking on the menu Cell → Run All.



You can load/edit/run/save python files(.py) into an iPython notebook cell by calling magic commands. Here are some useful magic commands:

- **Load:** %load filename.py
- **Write/Save:** %writefile myfile.py
- **Run:** %run myfile.py
- **Inline Plotting:** %matplotlib inline

## Python Quick Guide

A good reference for Python is this online book: <http://www.greenteapress.com/thinkpython/html/index.html>. Below are a couple of general Python coding exercises to get you going. If you are not comfortable with Python try implementing these first before proceeding to the machine learning lab material.

### Example 1

Example 1 demonstrates basic Python syntax. Unlike many other languages which use a semicolon(;) to indicate the end of the statement, Python has no mandatory statement termination characters, however it is whitespace sensitive (i.e. you must indent your code properly). In addition, Python is dynamically, implicitly typed (i.e. you don't have to declare variables). Comments in Python start with the hash character # , and extend to the end of the physical line.

Listing 1: Basic Python Syntax

```
#Declaring variable type is not required
a_string = "hello, world"
an_integer = 12
a_float = 3.14
5 a_boolean = True

#To print a constant or variable, use commas to print several items,
print a_string, an_integer, a_float, a_boolean, "\n"

10 #A long statement may be split into different lines with a backslash:
print 'a long statement may be split using backslash', \
    'this is still the same statement', "\n"

#Indentation Example
15 x = 10
if x == 10:
    print ('x has a value of 10')
else:
    print ('x does NOT have a value of 10')
```

## Example 2

Listing 2 introduces the use of lists and dictionaries as data structures in Python. These are the building blocks for various other Python data structures, including DataFrames which can be used to store instances of data (similar to records in a database).

Listing 2: Examples of lists and dictionaries

```
myList = [1,2,3,4]

#range() creates a list of numbers between 0 and 10 in increments of 1
myList2 = range(0,10,1) #list of numbers 0 to 10
5

#this appends the object myList to the end of myList2 - i.e. the last element of
#myList2 is a list
myList2.append(myList)
print myList2

10

#remove(x) removes the first element in the list that matches x
myList2.remove(myList)

#to add each element of myList to myList2, we can use a for loop
15
for x in myList:
    myList2.append(x)
print myList2

#len(x) returns the length of x
20
print len(myList2)

#the sort function can take additional parameters such as a key,
#ascending/decending etc
myList2.sort()

25

#slicing lists - allows you to select segments of the list
print myList2[2:4]
print myList2[:3] # first 3 elements

30
#dict contain key value pairs
myDict = {'a':'hello','b':'world'}
print myDict['a']
print myDict['b']
```

## Example 3

Listing 3 shows how functions can be defined within Python scripts.

Listing 3: Example Python function

```
#This Fuction output the result 1+1
def onePlusOne():
    return 1+1

5 #This function prints a passed string
def printme( str ):
    print str
    return

10 #Function Return sum of input values a and b
def add (a,b):
    return a+b

#Call Function
15 x = onePlusOne()
    y = add(1,2)

#native print fuction == printme()function
    print x
20 printme(y)
```

## Pandas Quick Guide

### Example 4

Pandas is a Python library that provides easy to use data structures and data analysis tools. As Pandas is built on top of the Numpy package, both Pandas and Numpy packages are usually required to be imported for complete functionality. Example 4 below demonstrates how to import and export dataset in pandas. All the data imported will be stored into a data structure called a DataFrame. For more details on Pandas functions, see the API Reference at <http://pandas.pydata.org/pandas-docs/stable/api.html>.

Listing 4: Introduction of Pandas

```
#imports the Pandas and Numpy libraries and gives aliases pd/np.
import pandas as pd
import numpy as np

5 #import csv file. If file contains header row, set header to the line 0
  #the import dataset will be stored into DataFrame (male100)
  male100 = pd.read_csv('male100.csv',header=0)
  print male100

10 #output male100 dataset to demo.csv file
    male100.to_csv('demo.csv')
```

The table below is the output of printing the Male100 DataFrame. This dataset contains Olympics men's 100 metres winning times for each Olympic year. In later exercises, you will need to evaluate this dataset, which is available on the module webpage.

Table 1: Data Structure of Male100.csv

	Year	Time
0	1896	12.00
1	1900	11.00
2	1904	11.00
3	1906	11.20
4	1908	10.80
5	1912	10.80
6	1920	10.80
...	...	...

### Example 5

Pre-processing a dataset is often crucial in order to obtain a more accurate ML model. The following example provides some basic data processing functions available in Pandas. In general it is good practice to perform pre-processing stages on a copy of the original DataFrame so that the original data remains unaltered. To do this use `copydf = df.copy()` as `copydf = df` only passes by reference, therefore changes made to `copydf` will effect the original DataFrame.

Listing 5: Data Processing

```
import pandas as pd
import numpy as np

male100 = pd.read_csv('male100.csv',header=0)

5 #Each Column can be extracted by "daraframe name[column name]"
print male100['Time'], "\n"

#Calculate mean and standard deviation
10 #other common functions also available: max(), median()...
mean = male100['Time'].mean()
std = male100['Time'].std()

#To get some basic statistics, we can use the describe() method:
15 print male100['Time'].describe(), "\n"

print mean, std
```

## Example 6

Example 6 introduces basic graph plotting. If you are using iPython Notebook, a Magic Line `%matplotlib inline` can be used to activate the inline graph display.

Listing 6: Basic Plotting

```
#This magic command is used to activated the inline graph display
%matplotlib inline

import pandas as pd
5 import numpy as np

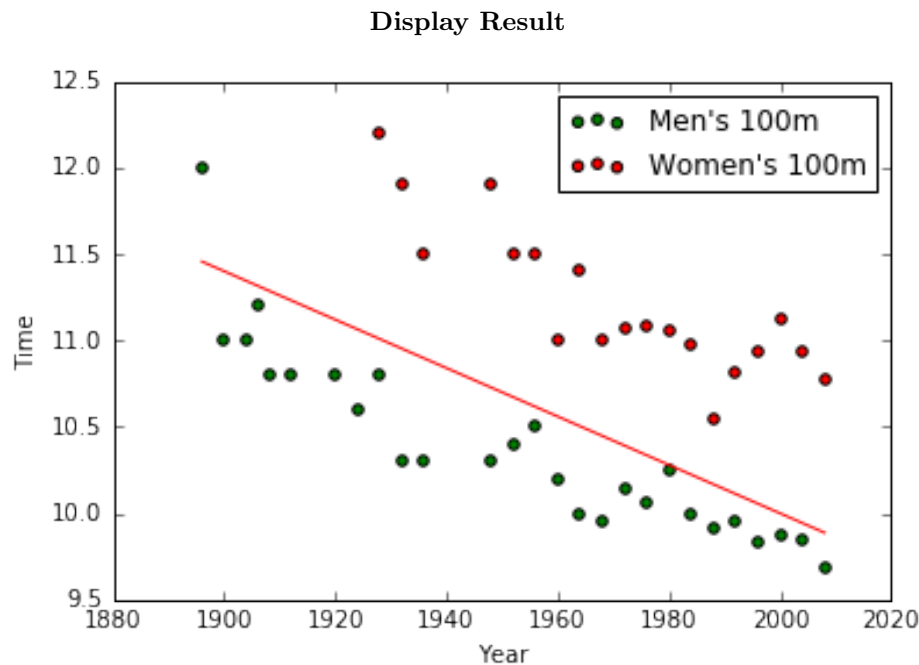
#import 'matplotlib.pyplot' to plot a simple stright line
import matplotlib.pyplot as plt

10 male100 = pd.read_csv('male100.csv',header=0)
female100 = pd.read_csv('female100.csv',header=0)

#Basic pandas plotting
male100.plot(x=0,y=1, kind = 'scatter', color='g', marker='v', label="Men's 100m")
15 #Simplified Version
male100.plot.scatter(0,1, color='g', label="Men's 100m")

#Two different dataset in one graph
20 ax = male100.plot(x=0,y=1, kind = 'scatter', color='g', label="Men's 100m")
female100.plot(x=0,y=1, kind = 'scatter', color='r', label="Women's 100m", ax = ax)

#we can use plt(imported from matplotlib) to plot a simple graph
#define a graph with repect to the all Olympic Years in male100.csv
25 y = -0.014*male100['Year']+38
plt.plot(male100['Year'],y,'r-',color = 'r')
```



The above code uses the function `DataFrame.plot(x, y, kind, color, label, ax)`. The first two parameters indicate which columns of the DataFrame will be used for the x and y axes. `color` is used to assign a colour to data points, `marker` defines their shape, and `label` gives the graph a name. `kind` allows the plotting method to be set, the default is line, however, many other plotting methods are available, including:

- `bar` or `barh` for bar plots
- `area` for area plots
- `hist` for histogram
- `scatter` for scatter plots
- `box` for boxplot
- `hexbin` for hexagonal bin plots
- `kde` or `density` for density plots
- `pie` for pie plots

A full list of parameters that can be used within `DataFrame.plot()` can be found at <http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html>. Additional colour options are also available [http://matplotlib.org/api/colors\\_api.html](http://matplotlib.org/api/colors_api.html), as are marker shapes [http://matplotlib.org/api/markers\\_api.html](http://matplotlib.org/api/markers_api.html). In order to plot different DataFrames in the same figure, we can use the `ax` variable, for example:

```
ax = df1.plot()
df2.plot(ax=ax)
```



# 1 Machine Learning Component

The material here builds on the first two lectures. As with the lectures, the first lab follows Chapter 1 from the Rogers & Girolami book. If you complete the lab material try the optional components that are based on lecture 3. In general, you can find the data and Matlab codes for these exercises in the R&G book: <https://www.dropbox.com/sh/7p6tult29idgliq/AAC3ipz88K?m> but at this stage you should start to implement them on your own and in Python. We will be providing the Python (unoptimised) "solutions" a week after each Lab. We are here to assist with your learning experience so if you need help ask us.

## 1.1 Feature scaling

→ Import the men's 100m Olympic dataset into a Pandas data structure. Calculate the mean and the standard deviation of each column. Subtract the mean value of each column from every element of the column. Then divide every element of the column with the standard deviation of each column.

This is a typical pre-processing step called "standardisation" that rescales our data in order to avoid problems of relative scale (e.g. when one attribute takes values in [10,1000] whereas another in [0.01, 0.2]). There are other such **feature scaling** steps that one can use, such as to scale the N D-dimensional inputs to unit length:  $\mathbf{x}_n = \frac{\mathbf{x}_n}{\|\mathbf{x}_n\|_p}$  where  $\|\mathbf{x}_n\|_p$  is the  $L_p$  norm of input  $\mathbf{x}_n$ . This is defined as  $L_p = \left(\sum_{d=1}^D |x_{nd}|^p\right)^{\frac{1}{p}}$  and for  $p=1$  it gives us the Manhattan norm:  $L_1 = \sum_{d=1}^D |x_{nd}|$ . Norms can be thought of, for our purposes, as different distances of the vectors  $\mathbf{x}_n$  from the centre of the Cartesian axis.

→ What does it mean if the std of an attribute/feature is zero? What should we do if that happens?

→ Create a function called *scaleData* that takes as input our input data  $\mathbf{X}$  and an option/flag second argument that defines which scaling the function implements. Implement both scaling techniques from above.

## 1.2 Linear regression example with Ordinary Least Squares (OLS)

→ Write a function called *linreg* that takes as inputs the training data  $\mathbf{X}, \mathbf{t}$  and returns  $\hat{\mathbf{w}}$  the OLS estimate of the parameters. The OLS solution, see lecture notes or the R&G book chapter, is:

$$\hat{w}_0 = \bar{t} - w_1 \bar{x} = \frac{1}{N} \sum_{n=1}^N t_n - w_1 \sum_{n=1}^N x_n \quad (1)$$

$$\hat{w}_1 = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - (\bar{x})^2} = \frac{\frac{1}{N} \left( \sum_{n=1}^N x_n t_n \right) - \bar{x}\bar{t}}{\left( \frac{1}{N} \sum_{n=1}^N x_n^2 \right) - (\bar{x})^2} \quad (2)$$

An equivalent and easier implementation of the above solution is to use the following vector/matrix representation (pages 15 - 22 in R&G book) that makes use of the transpose  $(\cdot)^T$  and inverse  $(\cdot)^{-1}$  operators:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t} \quad (3)$$

To implement this, Pandas data frames can easily be converted into matrices using *dataFrame.as\_matrix(columns)*. Once converted, matrix operations in the *numpy* library can be used such as *matrix.transpose()*.

→ Now apply this function on your *standardized* data  $\mathbf{X}$ , and targets  $\mathbf{t}$  from the Olympic dataset. Plot your dataset and the OLS hypothesis (the fitted line).

→ Use the built-in linear regression (OLS) package available in *scipy.stats* and compare the solutions. To call the built-in function import the library using: *from scipy import stats* and execute using: *stats.linregress(x,y)*.

→ Write a function called *predict* that takes as input your inferred OLS parameters and predicts the output  $t^*$  for any new input  $\mathbf{x}^*$ . You will have to apply the same standardisation to  $\mathbf{x}^*$  as you did for  $\mathbf{X}$ . Predict the winning time for 2025. What do you observe? What would you do if you were asked to give a realistic prediction?

### 1.3 Non-linear response from a linear model

→ Extend your *linreg* function to take as input a parameter that dictates the order of the polynomial feature expansion (see lecture notes and chapter). Repeat 1.2 but this time with a 3rd order polynomial expansion of the data. Your matrix  $\mathbf{X}$  should have  $N$  rows and  $D=4$  columns with the  $n$ th row as  $[1 \ x_n \ x_n^2 \ x_n^3]$  and your  $\mathbf{w}$  vector will be a 4-dimensional vector  $[w_0 \ w_1 \ w_2 \ w_3]$ .

### 1.4 [Based on Lecture 3] K-fold Cross-validation

Implement K-fold Cross Validation and use it to select the best model between the 1st order and 3rd order polynomials you have fitted to the data so far in previous sections. You may find the *cross\_validation* package from the *sklearn* library helpful for creating folds (*from sklearn import cross\_validation as cv*), particularly the *KFold* function.

### 1.5 [On your own time] Woman's vs Man's Olympic times

Perform linear regression with no feature expansion on the men's and women's 100 m. Using these models find the Olympic games when it is predicted for women to run a faster winning time than men. What are the predicted winning times? Do they seem realistic?