

Machine Learning Commando Course 2018

Session 0 - introduction

Russell Moore

ALTA / Computer Laboratory

July 2018

Inspirational Quote of the Day

**Laughing* I can't believe someone has decided to let you shape young minds!*

- Cash, owner of Caffiend Coffee

Introduction

- ▶ Machine Learning == techniques to configure algorithm behaviour (to be useful in solving a problem) without explicit programming.
- ▶ Usually involves learning from known examples (supervised learning) or trying to infer patterns from unmarked data (unsupervised learning).
- ▶ Also, semi-supervised learning that combines the two approaches.
- ▶ Generally speaking, we take a particular approach and use some metric to refine the weights or parameters for that approach - e.g. using squared error to choose the best coefficients for a line to separate two classes of points.

Book

Many of the exercises in this course are adapted from the following book:

- ▶ **Learning scikit-learn: Machine Learning in Python**
- ▶ By Raúl Garreta, Guillermo Moncecchi
- ▶ <https://www.amazon.co.uk/d/Books/Learning-scikit-learn-Machine-Python-Ra%C3%BA1-Garreta/1783281936>

Other material is variously sourced from Prof Ann Copestake, Dr Paula Buttery, Dr Sean Holden, Dr Jason Brownlee, Dr Andrew Ng, and also just plain made up.

Tools we'll be using

- ▶ **Python 3**
- ▶ **git**
- ▶ **virtualenv**
- ▶ **Numpy, Scipy** and ...
- ▶ **Scikit-learn**
- ▶ **Jupyter**

These are all useful tools and, except perhaps for Jupyter, you'll use them in pretty much every data-science oriented Python project.

Getting started

Python often feels like a game of installing stuff in order to install other stuff, in order to install other stuff...

You might as well install **git** first:

- ▶ You'll use it again! Like eating and breathing, git is a life skill.
- ▶ It is independent of the fact we're using Python
- ▶ `https://www.atlassian.com/git/tutorials/install-git`

Use git to get the workshop files

Now you have git, you can download the files for these workshops. You can't use them yet, but soon..

- ▶ You can put the files anywhere. I normally use `/git/my_project_name` or `c:\git\my_project_name` to keep all my git repos together, but it's your choice.
- ▶ Get the files: `git clone https://github.com/rjm49/ml_commando_2018`
- ▶ You now have a local repo for these files. You can always get the latest version of the files, using `git fetch`

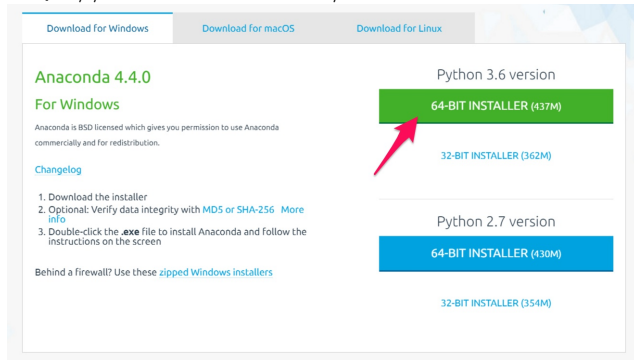
Next install Python 3

To get any further, we'll need Python. We'll be using Python 3.

- ▶ <https://www.python.org/downloads/>
- ▶ Make sure you get Python 3.x.x (64-bit version).

Install SciPy

The easiest way is to use the Anaconda bundle:
<https://www.continuum.io/downloads>



The screenshot shows the Anaconda download page for Windows. At the top, there are three tabs: "Download for Windows", "Download for macOS", and "Download for Linux". The "Download for Windows" tab is selected. Below the tabs, the page is titled "Anaconda 4.4.0 For Windows". It includes a brief description of the license, a link to the "Changelog", and a list of three steps for installation. To the right of the text, there are two sections for installers. The first section is for "Python 3.6 version" and contains two buttons: a green "64-BIT INSTALLER (437M)" and a blue "32-BIT INSTALLER (362M)". A red arrow points to the green 64-bit button. The second section is for "Python 2.7 version" and contains two buttons: a blue "64-BIT INSTALLER (430M)" and a blue "32-BIT INSTALLER (354M)".

Download for Windows Download for macOS Download for Linux

Anaconda 4.4.0

For Windows

Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.

[Changelog](#)

1. Download the installer
2. Optional: Verify data integrity with [MD5](#) or [SHA-256](#) [More info](#)
3. Double-click the **.exe** file to install Anaconda and follow the instructions on the screen

Behind a firewall? Use these [zipped Windows installers](#)

Python 3.6 version

64-BIT INSTALLER (437M)

32-BIT INSTALLER (362M)

Python 2.7 version

64-BIT INSTALLER (430M)

32-BIT INSTALLER (354M)

Install virtualenv

The point of virtualenv is that it allows you to create project-specific Python installations that do not interfere with your system-wide Python. A good habit to acquire!

- ▶ Run: `python3 -m venv /pathto/virtualenvironment`
- ▶ This will create a folder called **/pathto/virtualenvironment** which contains a nice new Python 3 installation
- ▶ CD into that folder, then into its `Scripts` folder, and run the `activate` command (use `source bin/activate` on unix).
- ▶ This will start your virtualenv (use `deactivate` to stop it...)

Install Scipy etc

If you'd prefer to install Scipy etc via pip on Windows, the easiest way to do it is to use Christoph Gohlke's pre-packaged wheel files.

Get the key bits from here:

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy>

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy>

The rest of it you can just install using pip. Good luck :)

Install Jupyter

Like you haven't already installed enough stuff, you'll need Jupyter to run the workshop files. Jupyter gives you an interactive environment to work through IPYNB 'notebook' files as presented here. Once set up, it's a bit nicer than working with a regular IDE.

- ▶ Activate your virtual environment, then run `pip install jupyter`
- ▶ Once this is installed, CD to the git repo.
- ▶ Fire up the notebook with `jupyter notebook`

Your browser should open and display a menu of the notebook files currently available.

Mission 0: Iris Classification

M/L Commandos are tough, but we can still appreciate nature. To get us set up, we'll explore a famous classic dataset of 150 samples of iris flowers, which is built into scikit-learn.



Figure 1: Iris versicolor. Pretty.

Irises - what we'll do...

This task is a case of *supervised learning*: we have a set of data (described by `X_iris`), which has been expertly labelled (labels are in `y_iris`). We can use these expert labels to train an algorithm to identify new, unknown flowers. Lots to do:

- ▶ Use scikit-learn built-in datasets
- ▶ Python introspection - have a look at data types
- ▶ NumPy indexing
- ▶ Drawing graphs with PyPlot
- ▶ Training and Test sets
- ▶ Scaling data
- ▶ Get a taste of the scikit-learn API
- ▶ Linear classification
- ▶ Intro to performance metrics

Preparing the data

Data preparation is important and makes up much of a M/L practitioner's working hours!

Split data into two sets:

- ▶ Training set: optimise your algorithm on a large set of known data;
- ▶ Test set: execute the final test of its performance on an unknown data set. This is to ensure the algorithm *generalises* to unseen data.

Often, a third dataset called the *validation set*, takes the role of test set during development. (Strictly, you should only test with your test set once, and then discard it, but no-one does this!)

Tip: Use sklearn's tools to split the datasets.

Preparing the data - Features, centre and scale

The numbers that describe your samples are called *features*. Our data has four features - the lengths and widths of sepals and petals. Sometimes you will have thousands of features.

Many ML algorithms are sensitive to skewed data and to differences between the magnitudes of features, so usually you will want to centre and scale your data.

For each dimension, sklearn's `StandardScaler` centres the data around its mean and scales it so that the standard deviation is unitary:

$$\mathbf{x}_{scaled} = \frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$$

If all your features are similarly distributed, you needn't do this, but if in doubt, do it!

Estimators in Scikit Learn

Scikit learn encapsulates different types of machine learning algorithm into objects called *estimators*. (Sometimes, *predictor* or *classifier*)

These objects support a consistent API: they are trained using the **fit(X,y)** (or **partial_fit(X,y)**) method, and used via the **predict(X)** method (which returns a predicted value of y).

Depending on the underlying mathematics of the estimator, we may be able to get addition information about the prediction (e.g. a probability score).

This consistency is very useful because it means we can try out different types of estimator with few code changes.

My First Classifier - Logistic Regression

We'll start with the Logistic Regression classifier, since it is one of the easiest to understand, is quick to train and quite versatile:

```
from sklearn import linear_model  
my_first = linear_model.LogisticRegression()
```

PS. The historical name "logistic regression" is a bit confusing, since it is a method of classification, not regression. Let's try to clear that up...

My First Classifier - Logistic Regression

The name comes from the "logistic curve". a classic s-shape (sigmoid) function which is used to map some input value to the probability of a (binary) outcome.

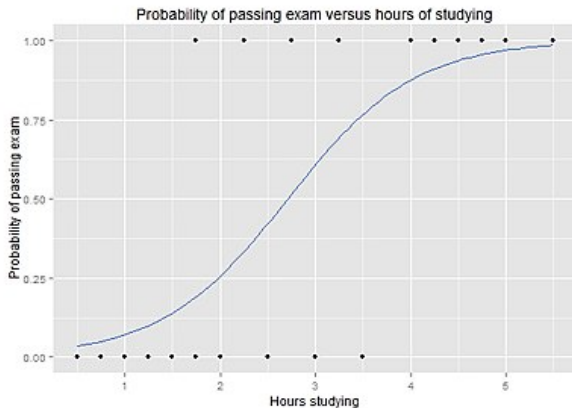


Figure 2: Logistic regression in action

My First Classifier - Logistic Regression

When we're dealing with more than one feature, we first combine the features into a single "score" via a weighted sum (this is the "regression" part):

$$h = w_0x_0 + w_1x_1 + \dots + w_nx_n + b$$

To get a probability, we squeeze this score into a range between 0,1 using a sigmoid:

$$\textit{sigmoid}(z) = \frac{1}{1 + e^{(-z)}}$$

So our overall predicted probability is:

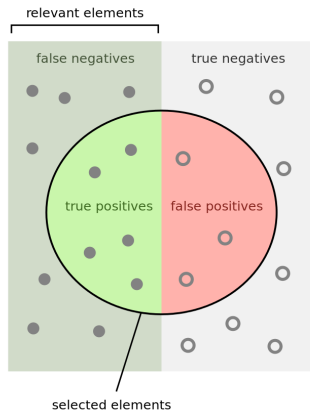
$$\hat{y} = \textit{sigmoid}(h)$$

Normally we treat w and x as vectors of numbers and treat the bias b as an extra dummy weight, so we can write:

$$h_w = \mathbf{w} \cdot \mathbf{x}$$

Precision and Recall

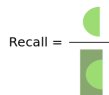
- ▶ Precision, p = % wanted of those predicted
- ▶ Recall, r = % predicted of those wanted
- ▶ Often combined into one handy metric called F_1 or "F-score" which is the harmonic mean of p and r .
- ▶ $F_1 = \frac{2pr}{p+r}$



How many selected items are relevant?



How many relevant items are selected?



Multiclass classification

Logistic regression is naturally only binary (“target” or “not target”), but we can isolate a given target class from all other classes. Then for N classes we can train N classifiers, each looking for a separate target. This is called a *One-vs-Rest* strategy. Many estimators in Scikit learn will do this automatically with appropriate configuration, and some are naturally multiclass.

Fill in the code blanks to implement multiclass classification for the full feature set.