Lab Vision Systems – Learning Vision Systems on Graphics Cards (MA-INF 4308)
Assignment 2 Issued: 06.11.2015 WS1516

Generating Toy Datasets

Quite often in this course you'll write an algorithm that makes specific assumptions about the data. To demonstrate that it works, you should be able to generate data which meets your assumptions. You could also work on real-world datasets right-away, but they tend to be high-dimensional and it is harder to see whether your implementation is working correctly. The exercises in this sheet should show you how to generate data for your algorithms and how to visually inspect the results.

For generating and visualizing data we'll use python. You'll see why, it's very convenient. Start by opening a terminal and entering `ipython notebook --pylab=inline`. A browser window should open which allows you to follow the tutorial in this assignment. If you haven't used ipython notebook before, I'd advise you to at least see the tutorial here:

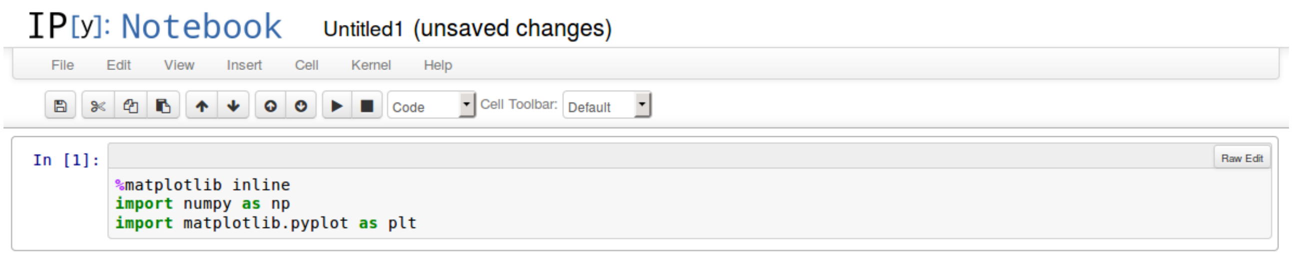
- IPython nootbook page: http://ipython.org/notebook.html
- Video tutorial: https://www.youtube.com/watch?v=H6dLGQw9yFQ#t=133

We will use two main libraries in python: NumPy and matplotlib.

- NumPy is a matrix library which supports slicing in a way which is much more advanced than other linear algebra libraries. (http://docs.scipy.org/doc/numpy/)
- matplotlib is a plotting library which allows you to do similar things to gnuplot, but with a real programming language at your finger tips. (http://matplotlib.org/)

You can use two libraries by inserting this code in the first cell on your ipython notebook.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```


IP[y]: Notebook
Untitled1 (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
Cell Toolbar: Default
Raw Edit

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
```

Toy Datasets for Supervised Learning

Datasets come in two basic variants.

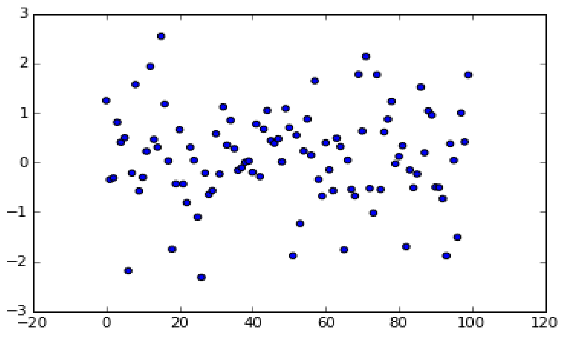
- For *unsupervised* learning we just have $\mathbf{D}=\mathbf{x}_i$ for $i=1\ldots N$ and $\mathbf{x}_i \in \mathbb{R}^M$.
- For *supervised* learning we typically have pairs with $\mathbf{D}=\mathbf{x}_i, y_i$, where in regression problems we have $y_i \in \mathbb{R}$ and in classification problems $y_i \in \mathbb{Z}$.

We'll start with a simple dataset containing only normally distributed points with mean zero and a standard deviation of 1, written as $y \propto N(0,1)$.

```
def S0(size=100):
    x = np.arange(size) # 0, 1, 2, ...size-1
    y = np.random.normal(0, 1, size=size)
    return x, y
```

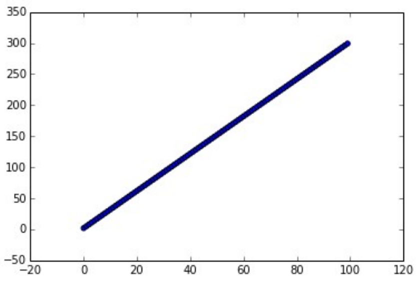
Let's have a look:

```
x, y = S0()
plt.scatter(x, y);
```



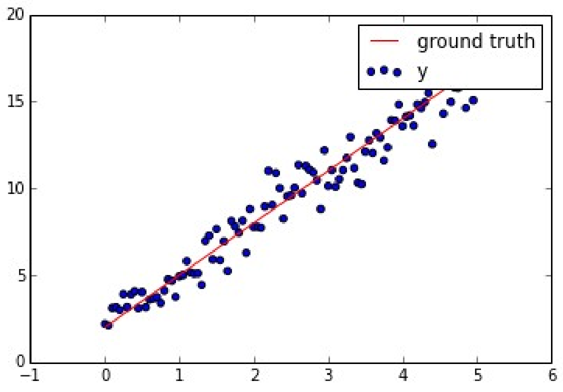
That by itself is not very interesting. We can make *y* dependent on *x*:

```
def S1(size=100):
    x = np.arange(size) # 0, 1, 2, ...size-1
    y = 3 * x + 2
    return x, y
x, y = S1()
plt.scatter(x, y);
```



Let's combine the two:

```
def S01(size=100, noise=1):
    x = np.arange(size)/20. # 0, 1, 2, ...size-1
    y = 3 * x + 2
    return x, y + np.random.normal(0, noise, size=size), y
x, y, y_gt = S01()
plt.scatter(x, y, label="y");
plt.plot(x, y_gt, "r", label="ground truth")
plt.legend();
```



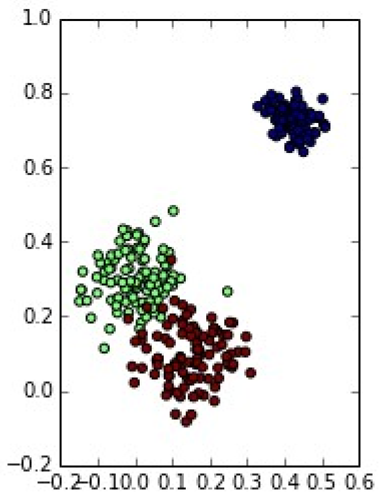
This is already an “interesting” regression dataset: We would like to infer the ground truth from the noisy values in *y*.

Generating Classification Datasets

We’ll assume for now that we have a few sources, and a measuring device that measures from which source our input is coming. However: every source can be identified by multiple values (variables), and the measuring is noisy.

```
def CD(n_classes=3, size=(100, 2)):
    means = np.random.uniform(size=(n_classes, size[1]))
    stds = np.random.uniform(0, 0.2, size=n_classes)
    X = [np.random.normal(m, s, size=size) for m, s in
zip(means,stds)]
    y = [[i] * size[0] for i in xrange(n_classes)]
    X = np.vstack(X)
    y = np.ravel(y)
    return X, y

np.random.seed(1)
X, y = CD()
plt.scatter(X[:,0], X[:,1], c=y)
plt.gca().set_aspect(1.)
```



That looks like one class which is very easy to distinguish, and two very tricky ones.

Saving Generated Datasets

You'll want to save the generated datasets to a file so your programs can process them. When your program prints data to a file, you can read it again. The functions `np.savetxt` and `np.loadtxt` should come in handy for this purpose.

Tasks

1. Generate a non-linear M=1-dimensional regression dataset with normal- distributed noise, e.g. where $y=\alpha\sin(\beta x)$! Visualize the dataset, and the ground truth.
2. Generate a linear M=2-dimensional regression dataset with normal- distributed noise! Note, that the dependencies between the variables in X are expressed using a matrix product (in the example above we just had a regular multiplication). Visualize the dataset and the ground truth. You'll need to do a 3D plot or multiple projections for this task.
3. Modify the classification example above so that the clusters become non- isotropic (i.e. have different standard deviations in x1 and x2 direction. Visualize the resulting dataset.
4. Modify the classification example above so that the x1 and x2 are correlated in every class. Hint: you need to do a matrix multiplication to express these interactions.
5. Write a C++ program that reads a matrix from `np.savetxt`. Multiply all values by two with CUDA C and saves the result to be read by `np.loadtxt`.