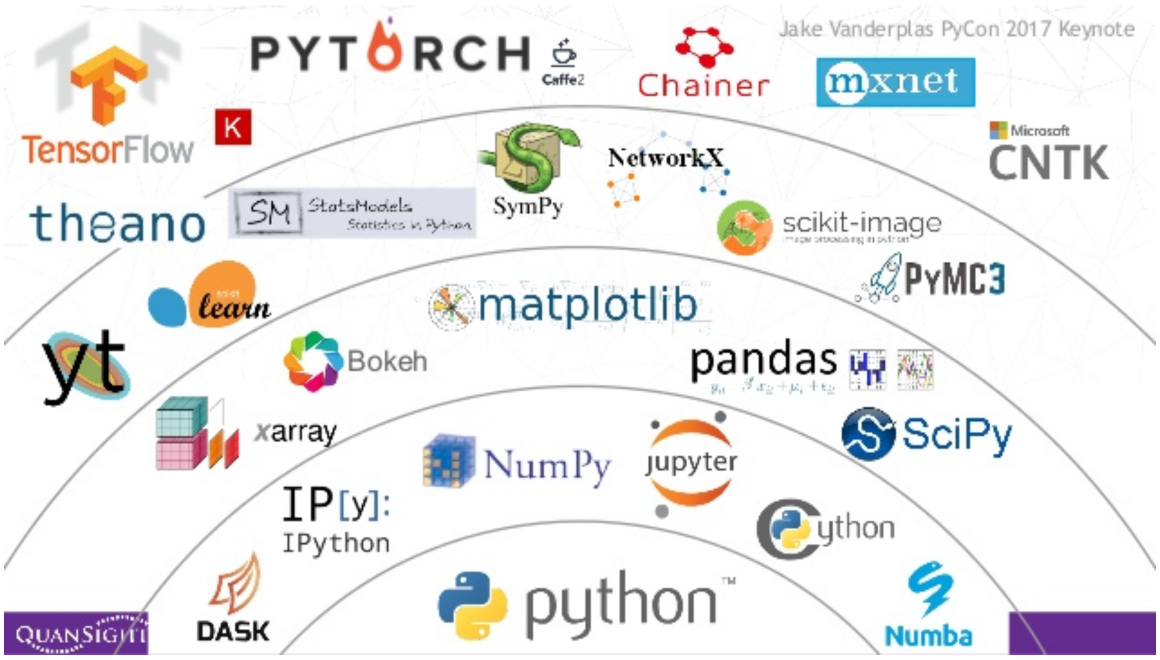
Machine Learning – Libraries, Data and exploring the mean square error (MSE)

Python has many of tools and libraries for machine learning and data science, as well as mathematics and the exact sciences:



In this practice, we will

1. Explore important python data science libraries in the **SciPy** Ecosystem.
2. Learn the **MSE** loss function.

The SciPy Ecosystem

Link: https://www.scipy.org/

SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Its core packages are:

NumPy

Link: https://numpy.org/

NumPy is a fundamental package for scientific computing with Python. The library makes the use of N-dimensional arrays easy and user-friendly. This library forms a basis for the SciPy ecosystem.

Pandas

Link: https://pandas.pydata.org/

Pandas is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool.

Matplotlib

Link: https://matplotlib.org/

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

IPython

Link: http://ipython.org/

The IPython project provides an enhanced interactive environment that includes, among other features, support for data visualization and facilities for distributed and parallel computation.
Most used with Jupyter Notebook.

Jupyter Notebook

Link: https://jupyter.org/

You already familiarized yourself with the Jupyter Notebook : an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It is used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

SciPy Library

Link: https://www.scipy.org/scipylib/index.html

The SciPy library a core package of the SciPy stack. It provides user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics.

SymPy

Link: https://www.sympy.org/en/index.html

SymPy is a symbolic mathematics Python libarary. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible to be comprehensible and easily extensible.

Note: We'll usually want to put all of our imports and initializations in one place at the start of the notebook.
This is particularly true when we want to download and install things on the kernel (the Linux computer that runs the notebook).

NumPy

The best way to start exploring data is to plot it on graph and observe the graph features.

We use NumPy to create and manipulate series (/arrays/vectors) of numbers.

1. Create a series of (x,y) points.
Start with creating an array of “x” values = 100 numbers from 0 to 99: Import numpy, and use the “arange” function to define the range of numbers.
Now create y values, using some function y(x).
Start with implementing the identity function `y(x) = x`. (hint – this function just needs to “copy” the x values)

Matplotlib

Matplotlib.pyplot is used to show data in graphs.

Graphs serve to observe and immediately see the function. Let’s observe the function you created (identity) in a graph.

2.Plot the points (x,y) you created in a scatter plot. Use the functions “rcParams” and “scatter”.

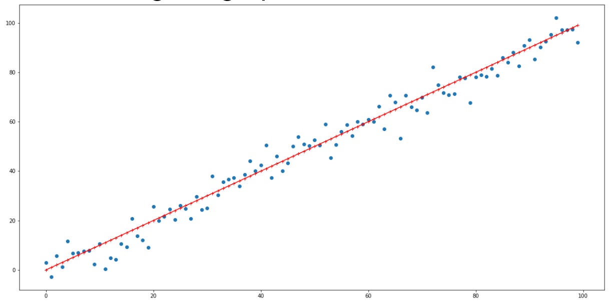
3.In an ideal world, our data will be a straight line, However, real-world data always contains Noise or Errors.

Add some noise (error) to your identity function: Create a random noise which has a normal distribution, with mean of 0 and a standard deviation of 5. Use the “random.normal” function. And add the noise to your “y” values.

4. To observe the effect of noise, it’s useful to see it together with the original “ideal” function of identity.

Plot both the new, “noisy y” series and the old one in the same graph.
To distinguish between the two, use different symbols for the new “noisy y” series, with the function “plot” and then “show”.

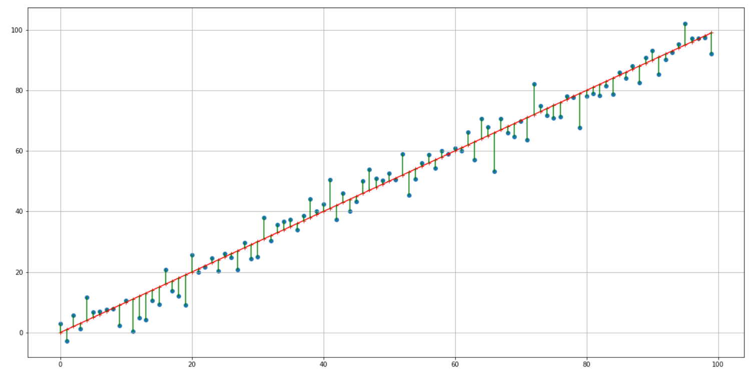
You should get a graph like this one:



5. To evaluate the effect of this noise, we need to find out how far are the new points from the original ideal function. For this we need to define and calculate a distance:

Again, the best start in data exploration is our visual observation: Before calculating the distance observe the distances of each new point to its original ideal point on the graph. In order to easily observe this, add vertical lines to the graph you plotted in 4. that represent the distance of each "noisy" point from its "ideal" point in the graph. You can explore and use the function "zip".

You should get a graph like this one:



6.As can be observed, some distance values in the graph are small and some large. We need one number to represents them all and provide a noise, or error measure.
This number could be a sum on all the distances, but then, with more data points we'll get a bigger error.
A good measure should be invariant to the number of data point used. So, a better measure will be the mean value – just divide the sum by the number of points.

Also note, that some distance values in the graph are positive (above the identity function line) and some negative. However, as a distance is defined as a positive measure, it should be changed, by either of the following methods:.

1. **MSE - Mean Squared Error** = Take the distance and raise it to the power of 2 to get `dist^2`.
2. **MAE - Mean Absolute Error** = Take the absolute value of the distance to get `|dist|`.

You can read more on the differences between these two approaches in MAE and RMSE — Which Metric is Better? . RMSE is the squared root of the MSE, people use both.

Calculate the MSE value for your data point:

7.The power of computing is creating an automated method that can apply this process: the graphic visualization and the MSE calculation for any function.

Create the functions "show_graph" and "calculate_mse" using "def"
Notes:

- In a good graph the axes are labelled and there is a title. A grid is also useful to help observe distances.
- A nice feature is to write the calculated MSE as the title of your graph or in a legend inside it.

8.Now use the functions you created on new data.
Write a method that will produce a noisy version from any original series of data points, then plot the graphs and calculate the MSE:

Write a function “create_new_y”.
Then use NumPy `sin` method to try your functions on `y(x) = sin(x)`.

ScyPy

9.If your graph of the original `sin` function is not smooth enough, ScyPy can be used to apply **interpolation** to make these connected lines smoother.

Write a function “show_graph_with_interpolation “ that interpolates the values of a series and plots them on a nice-looking graph (with axes, title, and grid). Use `scipy.interpolate` and `interp1d`.

10.Let’s define a method that’s even more general. The method should create any function we want and add any random noise (at this point – just changing the random noise parameters – the mean and SD).
Use all the previous functions to write the function “plot_ideal_and_noisy_version_of_data” which returns the graph and MSE for any fuction.

11. Let’s investigate the effect of the standard deviation on the MSE.

Insert different `sd` for the identity and sin functions and a log function, and observe how does the MSE changes. Produce graphs for each functions with SD values of 5,3,1 and 0.
You can explore the use of “`lambda`” function.

What is your observation on the SD “0” value? What is the reason for this ?

12.Now play with the MSE - check what happens if you try to calculate the MSE between an original series of points of one function and noisy points of other function.

Run it with the pairs of functions: “sin” and “cos”, and “linear” and “log”. (note: be careful with the log function: log(0) is infinity and log(1) is 0…).

You can now observe the differences in MSE in these two examples:
In the sin-cos combination, the functions are different, but the MSE is small -why?
In the linear-log combination, the MSE is very big – why?

Let’s try to reduce the linear-log MSE, by changing the slope of the linear function. It will now not be an identity, but rather a linear function y(x)=ax. Observe the effects of slopes 0.5, 0.05 and 0.005.

Let’s leave computing for a minute and interpret our results:
In order to compare across graphic results, if we have several graphs, a better view of the results is a table that summarizes the main results of all the graphs.
Here it is for the examples you plotted:

slope	1	0.5	0.05	0.005
MSE	3075	673	1	12

This table can assist us in determining the best slope, where best is the one that yields the lowest MSE:

The MSE is first reducing when the slope reduces, but this trend flips when we get to `slope=0.05` and then the slop increases for `slope=0.005`.

It means that the best MSE value for these functions is lying somewhere between `slope=0.5` and `slope=0.005`. The best slope in our 4 attempts produces MSE=1. Is it really the best one? or maybe there are better slopes out there?

To find “best” or “maximum/minimum” we need to calculate the *derivative* of a function in mathematics. Let’s try that:

SymPy

13.Explore the MSE definition using mathematics:

$$\text{MSE} = \frac{\sum_{i=1}^{n} \left(-y(x_i) + t_i\right)^2}{n}$$

To build this expression, use the SymPy library.

Building the expression is like a Lego® assembly, using building blocks.
You first need to define the smallest building blocks (symbols, indexes, functions, etc.).
Then, build the expression from these blocks.
The expressions are rendered with Unicode.

Note: In order to be compatible with the ML literature, change the variable names.

Call the original variable (the former `y)` as `t` (standing for “*true values”)*.

Call the noisy version of y (the former `new_y`) as just `y`.

The variable `x` can stay as `x`.

This will be the ML job: to fit the values “y” to their true values!

Let’s start!:

Import all the modules of the `sympy` library. Create the basic building blocks for the expressions of MSE: n (number of points), i – the index for x and t series(use the function “symbols”), xi, ti (use the function “Indexed”, and the y series (use “Function”)

Use “init_printing” to define the Unicode.

Now create the MSE expression from these basic building blocks

14.Now calculate the derivative of the MSE expression with SymPy, using “diff”.

15.In order to show the string representation of the derivative expression, you can use “srepr”

16.Plot the MSE expression, in a tree form graph, with nodes and edges.
Import **`pprint`** and `dotprint` (from **`sympy.printing.dot`**)

17.The graph can be represented in a better way, to allow interpretable visualization.
You can do it with the `graphviz` library, rendering the graph in **SVG** format.
This will also save the graph in a good resolution image at any size.
Write the function “draw_tree” that provides this graph.
Import “`Source`” from **`graphviz,`** `SVG from` **`IPython.display,`** and **`os`**

18.And lastly, let’s summarize all previous steps in a single method, that supports many functions, adds noise and calculates the MSE value for any series of data points and functions.
The input is series of points `x` and `t` (original points) and returns the noisy points.
We will transfer this `y` function with the `x` and `t` variables to the `mse` method, to calculate the MSE from the SymPy expression.

Demonstrate the method’s operation by calculating the MSE of 100 points of a `sin` function with noise.

More Information

MSE on Wikipedia:
Wikipedia MSE

An explanation of NumPy axes:
NumPy Axes Explained

Documentation of NumPy library:
NumPy Documentation

Documentation of matplotlib.pyplot.scatter method:
Matplotlib pyplot.scatter Documentation

Documentation of matplotlib.pyplot.plot method:
Matplotlib pyplot.plot Documentation

Documentation of plt.rcParams dictionary parameters:
Customizing Matplotlib with style sheets and rcParams

Tutorial for matplotlib.pyplot (`plt`) module:
Matplotlib pyplot Tutorial.

Tutorial for SymPy library:
SymPy Tutorial

Why there are so many ways to import libraries:
Official abbreviation for: import scipy as sp/sc.

There is another way of displaying the tree graph in IPython:
render_sympy_tree.py.

SymPy subscript:
How do I define a sympy symbol with a subscript string?

Markers in Matplotlib plot line:
Set markers for individual points on a line in Matplotlib

Site for resizing images:
resizeimage.net