IN4080, 2020, Exercise set 1:

Installation

At the IFI-cluster, put the following in your .bashrc-file:

- export PATH=/opt/ifi/anaconda3/bin/:$PATH
- export NLTK_DATA=/projects/nlp/nltk_data

Your PC (at least Windows): use Anaconda or Miniconda. See the guide.

Part 0 (for those with no background in Python, or those who need a fresh up)

The NLTK book teaches NLTK and Python simultaneously. We will use parts of the book. While reading the book, you should sit on the terminal and type the examples from the book.

- Start with Section 1 Computing with language and Section 2 A closer look at Python from http://www.nltk.org/book/ch01.html
- Then do exercises: 1, 3, 8, 16, 19 from Section 8 in same chapter.boo
- Work through Chapter 3, Section 2 Strings
- Do exercises: 9, 10, 13 from Chapter 1, Section 8.
- And exercises 2, 4, 5, 10 from Chapter 3, Section 12.
- Learn about dictionaries from Chapter 5, Section 3 Mapping words ...

We assume you know how to program in some language or other, but at some time you have to learn the quirks and quiddities of Python. Where to look?

- The NLTK book, e.g. sec.1.4, 2.3, and (eventually parts of) ch. 4
- Sooner or later you will have to consult the excellent official Python documentation (https://docs.python.org/3.4/), in particular the tutorial and library reference
- Think Python: How to Think Like a Computer Scientist is an easy introduction to Python.
- The Scipy Lecture Notes contains an introduction to Python for they who know how to program, in addition to descriptions of many of the other tools we will use, including NumPy, Matplotlib:plotting, Scipy, statistics in Python, and, even, scikit-learn.

Part 1 – For everybody

Exercise 1: nltk.FreqDist()

NLTK has a class for calculating frequencies, the nltk.FreqDist() class. To understand it better, it may be useful to see that the core is a Python dictionary. Make a Python function, *my_frequency*, which takes a list, *j*, as an argument and returns a dictionary, *d*. The dictionary *d* should take the members of *j* as keys and to each key, *k*, return the number of occurrences of *k* in *j*. Apply it to a list, e.g.*a*= ['this', 'is', 'a', 'stupid', 'sentence', 'this', 'is', 'also', 'stupid', 'this', 'not', 'sentence'] and get a dictionary, *my*. Consider the most important methods of the dictionary,

- my.items()
- my.keys()
- my.values()

Then use the nltk.FreqDist() and see that you get the same results

- fd = nltk.FreqDist(a)
- fd.items()
- fd.keys()
- fd.values()

The nltk.FreqDist() extends the dictionary class with several methods, see the end of Section 3 in Chapter 1 in the NLTK book. Try them out, in particular fd.tabulate() and fd.plot().

Observe that a NLTK frequency distribution behaves like a default dictionary. The following is OK

- In [583]: 'smart' in fd
- Out[583]: False
- In [584]: fd['smart'] += 1
- In [585]: fd['smart']
- Out[585]: 1

But my['smart'] += 1 would fail.

If you haven't done it before, you should on your own work your way through Section 3 in Chapter 1 in the NLTK book to get more familiar with the FreqDist() class. You will also make a first encounter with two key concepts we will meet again: Bigram and Collocation.

If you are familiar with the python class Counter, you may recognize that it has many of the properties as nltk.FreqDist().

Exercise 2

NLTK Ch1, Sec8, ex 26, 27, 28

Exercise 3

We move to Ch.2, Sec. 1 in the NLTK book to get some real language data. If you have not done it before, you should work your way through Section 1 in Chapter 2 in the NLTK book up to "Annotated text corpora".

Consider the Brown corpus. Make a NLTK frequency distribution for the pronouns: *I, he, she, we, they*, for the complete Brown corpus. Tabulate and plot it and keep the frequency distribution for later use.

Exercise 4 Conditional frequency distributions

We move on to Ch.2, Sec.2 in the NLTK book. Work your way through the section up to 2.4 Generating random text.

Make a NLTK conditional frequency distribution, which shows how the five pronouns from exercise 3 are distributed for the two genres *news* and *fiction*. Print a table of the results and make a plot as in the NLTK book. Keep these for later use.

matplotlib

plt.plot()

matplotlib is a package for making 2D plots and figures in Python. For they who have not used it before, we have given a few hints regarding how to get started in an appendix to this exercise set. For more on the plt.plot() command, see https://matplotlib.org/users/pyplot_tutorial.html and the official documentation https://matplotlib.org/contents.html. Try the following.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(np.arange(10))
>>> plt.show()
```

Exercise 5

Consider the frequency distribution you made in exercises 3. Use plt.plot() and plot the values from the largest to the smallest (cf. exercise 5). Your figure should have a similar form as what you got by using NLTKs fd.plot(), except that the NLTK plot includes categories on the x-axis.

The graphs that NLTK uses is a natural way to make figures when you have temporal data as in in figure 1.1 in the NLTK book Ch. 2, or you show cumulative data. It seems less natural for data like this. We can do better by using plt.plot(x, y, '.'), plotting balls instead of a curve. We could go on and add data to the x-axis, like NLTK.

```
>>> plt.xticks(np.arange(len(z)), z) # or
>>> plt.xticks(np.arange(len(z)), z, rotation=90)
```

provided z is an array containg the keys of the distribtuion in the same order.

plt.bar()

For data like this, it is more natural to use a bar chart. If y is an np.array containing the values of a frequency distribution, we can display this as a bar chart using

```
>>> plt.bar(x, y) # instead of plt.plot(x,y), x as in exercise 6
```

Try this. To add information on the x-axis about the categories, we can extend this to

```
>>> plt.bar(x, y, tick_label=z)
```

provided z is an array containing the keys of the distribtuion in the same order.

Exercise 6

Make a bar chart for your frequency distribution from exercise 3 with categories on the x-axis First, make a chart sorted by the category names. Then make a chart sorted by values from the largest to the smallest (cf. exercise 5)

To learn more on the args and kwargs of plt.bar() look at
https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar .

plt.hist()

Let us also see how to draw a histogram,
https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist
This is quite simple. We do not have to calculate frequencies (heights of the columns). If you have a list of numbers, *mynumbers*, the simple command

```
>>> plt.hist(mynumbers, bins=20)
```

will make a histogram with 20 bins.

Exercise 7

Make a histogram of the lengths of all the sentences in the Brown corpus.

plt.boxplot()

Exercise 8

Try plt.boxplot(lengths) where lengths is as in exercise 8, the lengths of the sentences in Brown.

END of exercises

Appendix: Numpy and matplotlib.pyplot

Many of you are familiar with these tools, but mabe not all of you. Here are few simple moves to get started.

NumPy

is a tool for numeric computing with Python. It adds both functionality and speed. The basic additional brick is the N-dimensional array data type. We will for now mainly consider one-dimensional arrays. A one-dimensional array is similar to a list but:

- All elements must be of the same type
- The array has a fixed length in the sense that we do not append or remove elements from the array
- It has additional methods and functionality

We may make a new array in many ways, e.g.

```
>>> import numpy as np
>>> a = range(10)
>>> b = np.array(a)
>>> c = np.arange(10)
>>> b
>>> c
>>> type(a)
>>> type(b)
>>> type(c)
>>> z=np.zeros(200)
>>> z
>>> np.ones(20)
```

Let us inspect some of the new functionality

```
>>> d = b+c
>>> d
>>> b*3
>>> g=b/3
>>> g
```

So far, this is regular linear algebra, adding vectors and multiplying with scalars. But NumPy also has more operations that are not standard linear algebra

```
>>> b+3
>>> b**3
>>> e = b*c
>>> e
```

Observe how NumPy does type cohesion in *a+b* and transforms *a* to an array. See how this differs from the list operation in *a+k*.

```
>>> a+b
>>> k = range(800,900)
>>> a+k
```

A particular useful NumPy function is linspace, see
https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html
Try it out and see that you understand what it is doing.

```
>>> np.linspace(0,1, 10)
>>> np.linspace(-2, 3, 51)
```

Name spaces

In the example, we imported NumPy as np. It is tempting instead to import everything from NumPy, as in

```
>>> import numpy
>>> from numpy import *
>>> a = range(10)
>>> b = array(a)
>>> c = arange(10)
```

This is convenient because it saves us from typing. However, there is a danger. If another module uses the same names for classes or functions, we get a name conflict, and we cannot access both functions using the same name. For example, NumPy has its own random module. This is different from the Python module *random*, and uses some of the same function names with a different interpretation. Thus if we import both *random* and everything from NumPy, we may experience a conflict. It is important to

- Know which name spaces you are using
- Consult the documentation for the functions before you use them.

If we import numpy as np, we should be safe.

Some tools for statistics in NumPy

The NumPy array has some built-in methods useful for statistics, e.g. consider the following. If *b* is the np.array from above, try

```
>>> b.mean()
>>> b.var()
>>> b.std()
```

To learn more on NumPy consider the NumPy user guide: https:/docs.scipy.org/doc/numpy-dev/user/index.html

matplotlib

plt.plot()

plt.plot()

matplotlib is a package for making 2D plots and figures in Python. For they who have not used it before, will give a few hints regarding how to get started. For more on the plt.plot() command, see https://matplotlib.org/users/pyplot_tutorial.html and the official documentation https://matplotlib.org/contents.html. Try the following.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(np.arange(10))
>>> plt.show()
```

If things are correctly installed, there should pop up a window with a graph. Close it and then try

```
>>> plt.plot(np.arange(10)**2)
>>> plt.show()
```

We have here let pyplot decide the interval we display, and the number of points for the plot. It is just an accident if this is the graph of x**2. To see this, try

```
>>> plt.plot(np.arange(3,10)**2)
>>> plt.show()
```

To take control over these parameters ourselves, the normal input to pyplot should be an np-array with values on the x-axis and another array with corresponding y values. So to get x**2 for integers from 3 to 9 we could use

```
>>> x = np.arange(3,10)
>>> y=x**2
>>> plt.plot(x,y)
>>> plt.show()
```

To draw several plots in the same graph, we can e.g. do

```
>>> x = np.linspace(-2, 3, 20)
>>> y=x**2
>>> plt.plot(x,y)
>>> y3 =x**3
>>> plt.plot(x, y3)
>>> plt.show()
```

The plt.plot() includes a lot of options in form of args and kwargs. In particular, we can control the color of the graph. Try in turn

```
>>> plt.plot(np.arange(3,10)**2, 'r')
>>> plt.show()


>>> plt.plot(np.arange(3,10)**2, 'b')
>>> plt.show()
```

We may also exchange or extend the graph with points. Try

```
>>> plt.plot(np.arange(3,10)**2, '.')
>>> plt.show()
```

and then

```
>>> plt.plot(np.arange(3,10)**2, 'b')
>>> plt.plot(np.arange(3,10)**2, 'rx')
>>> plt.show()
```

We can also add various text to the axis etc.

When building complex figures, it can be convenient to work interactively. We can achieve this by toggling the plt.interactive – feature

```
>>> plt.interactive(True)
```