**Text Analysis** in Python 3

**Word Frequency in Text**

So we are going to build a function which will count the word frequency in a text. We will consider a sample test text, & later will replace the sample text with the text file of books that we have just downloaded.

**Since we are going to count word frequency, therefore UPPERCASE and lowercase letters are the same.** We will convert the whole text into lowercase and save it.

**text = "This is my test text. We're keeping this text short to keep things manageable."**

**text = text.lower()**

Word frequency can be counted in various ways. **We are going to code, two such ways ( just for knowledge ).One using for loop and the other using Counter from collections, which proves to be faster than the previous one.**

The function will return a **dictionary** of **unique** **words** & its frequency as a key-value pair. So, we code:

**from collections import Counter**

**def count\_words(text):                   #counts word frequency**

**skips = [".", ", ", ":", ";", "'", '"']**

**for ch in skips:**

**text = text.replace(ch, "")**

**word\_counts = {}**

**for word in text.split(" "):**

**if word in word\_counts:**

**word\_counts[word]+= 1**

**else:**

**word\_counts[word]= 1**

**return word\_counts**

**# >>>count\_words(text)  You can check the function**

  #Using Counter method collections module

def count\_words\_fast(text):      #counts word frequency using Counter from collections

    text = text.lower()

    skips = [".", ", ", ":", ";", "'", '"']

    for ch in skips:

        text = text.replace(ch, "")

    word\_counts = Counter(text.split(" "))

    return word\_counts

    # >>>count\_words\_fast(text) #You can check the function

**Output :**The output is a dictionary holding the unique words of the sample text as key and the frequency of each word as value. Comparing the output of both the functions, we have:

*{‘were’: 1, ‘is’: 1, ‘manageable’: 1, ‘to’: 1, ‘things’: 1, ‘keeping’: 1, ‘my’: 1, ‘test’: 1, ‘text’: 2, ‘keep’: 1, ‘short’: 1, ‘this’: 2}*

*Counter({‘text’: 2, ‘this’: 2, ‘were’: 1, ‘is’: 1, ‘manageable’: 1, ‘to’: 1, ‘things’: 1, ‘keeping’: 1, ‘my’: 1, ‘test’: 1, ‘keep’: 1, ‘short’: 1})*

**Total Unique words:**

We are going to design another **function called word\_stats(),** which will take the word frequency dictionary( output of count\_words\_fast()/count\_words() ) as a parameter. The function will return the total no of unique words(sum/total keys in the word frequency dictionary) and a dict\_values holding total count of them together, as a tuple.

**def word\_stats(word\_counts):     # word\_counts = count\_words\_fast(text)**

**num\_unique = len(word\_counts)**

**counts = word\_counts.values()**

**return (num\_unique, counts)**

**Plotting Characteristic Features of Books**

We are going to plot, (i)Book length Vs Number of Unique words for all the books of different languages using [matplotlib](https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/" \t "_blank). We will import pandas to create a pandas dataframe, which will hold information on books as columns. We will categorize these columns by different categories such as – “language”, “author”, “title”, “length” & “unique” .To plot book length along x axis and Number of unique words along y axis, we code:

import os

import pandas as pd

book\_dir = "./Books"

os.listdir(book\_dir)

stats = pd.DataFrame(columns =("language", "author", "title", "length", "unique"))

# check >>>stats

title\_num = 1

for language in os.listdir(book\_dir):

for author in os.listdir(book\_dir+"/"+language):

for title in os.listdir(book\_dir+"/"+language+"/"+author):

inputfile = book\_dir+"/"+language+"/"+author+"/"+title

print(inputfile)

text = read\_book(inputfile)

(num\_unique, counts) = word\_stats(count\_words\_fast(text))

stats.loc[title\_num]= language, author.capitalize(), title.replace(".txt", ""),

sum(counts), num\_unique

title\_num+= 1

import matplotlib.pyplot as plt

plt.plot(stats.length, stats.unique, "bo-")

plt.loglog(stats.length, stats.unique, "ro")

stats[stats.language =="English"] #to check information on english books

plt.figure(figsize =(10, 10))

subset = stats[stats.language =="English"]

plt.loglog(subset.length, subset.unique, "o", label ="English", color ="crimson")

subset = stats[stats.language =="French"]

plt.loglog(subset.length, subset.unique, "o", label ="French", color ="forestgreen")

subset = stats[stats.language =="German"]

plt.loglog(subset.length, subset.unique, "o", label ="German", color ="orange")

subset = stats[stats.language =="Portuguese"]

plt.loglog(subset.length, subset.unique, "o", label ="Portuguese", color ="blueviolet")

plt.legend()

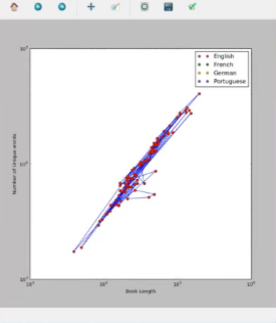
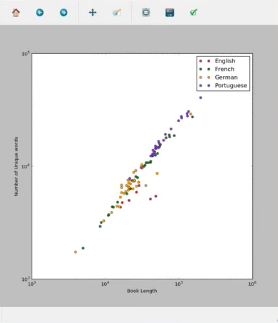
plt.xlabel("Book Length")

plt.ylabel("Number of Unique words")

plt.savefig("fig.pdf")

plt.show()

**Output:** We plotted two graphs, the first one representing every book of different language & author as simply a book.The red dots in the [first graph](https://drive.google.com/open?id=0ByqooCIGbyKiRzNKV1NxV1AteGs) represent a single book and they are connected by blue lines.The loglog plot creates discrete points [red here] and the linear plot creates linear curves [blue here], joining the points.The [second graph](https://drive.google.com/open?id=0ByqooCIGbyKiVHQwVEpENkdOb0E) is a logarithmic plot which displays books of different languages with different colours [red for English, Green for French etc] as discrete points.  
These graphs help in analysis facts visually about different books of vivid origin.From the graph, we came to know that Portugese books are longer in length and have greater number of unique words than German or English books.Plotting such data proves to be of great help for linguists.

# Topic Modeling with Gensim (Python)

Topic Modeling is a technique to extract the hidden topics from large volumes of text. Latent Dirichlet Allocation(LDA) is a popular algorithm for topic modeling with excellent implementations in the Python’s Gensim package. The challenge, however, is ***how to extract good quality of topics that are clear, segregated and meaningful***. This depends heavily on the quality of text preprocessing and the strategy of finding the optimal number of topics.

## Prerequisites – Download nltk stopwords and spacy model

We will need the stopwords from NLTK and spacy’s en model for text pre-processing. Later, we will be using the spacy model for lemmatization.

Lemmatization is nothing but converting a word to its root word. For example: the lemma of the word ‘machines’ is ‘machine’. Likewise, ‘walking’ –> ‘walk’, ‘mice’ –> ‘mouse’ and so on.

# Run in python console

import nltk; nltk.download('stopwords')

# Run in terminal or command prompt

python3 -m spacy download en

## Import Packages

The core packages used in this tutorial are re, gensim, spacy and pyLDAvis. Besides this we will also using matplotlib, numpy and pandas for data handling and visualization. Let’s import them.

import re

import numpy as np

import pandas as pd

from pprint import pprint

# Gensim

import gensim

import gensim.corpora as corpora

from gensim.utils import simple\_preprocess

from gensim.models import CoherenceModel

# spacy for lemmatization

import spacy

# Plotting tools

import pyLDAvis

import pyLDAvis.gensim # don't skip this

import matplotlib.pyplot as plt

%matplotlib inline

# Enable logging for gensim - optional

import logging

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings

warnings.filterwarnings("ignore",category=DeprecationWarning)

## Prepare Stopwords

We have already downloaded the stopwords. Let’s import them and make it available in stop\_words.

# NLTK Stop words

from nltk.corpus import stopwords

stop\_words = stopwords.words('english')

stop\_words.extend(['from', 'subject', 're', 'edu', 'use'])

## Import Newsgroups Data

We will be using the 20-Newsgroups dataset for this exercise. This version of the dataset contains about 11k newsgroups posts from 20 different topics. This is available as [newsgroups.json](https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json).

This is imported using pandas.read\_json and the resulting dataset has 3 columns as shown.

# Import Dataset

df = pd.read\_json('<https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json>')

print(df.target\_names.unique())

df.head()

* Introduction to Network Analysis
* Construct a simple Network

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Pygraphviz is a Python interface to the Graphviz graph layout and visualization package.

* Python language data structures for graphs, digraphs, and multigraphs.
* Nodes can be "anything" (e.g. text, images, XML records)
* Edges can hold arbitrary data (e.g. weights, time-series)
* Generators for classic graphs, random graphs, and synthetic networks
* Standard graph algorithms
* Network structure and analysis measures
* Basic graph drawing
* Open source BSD license
* Well tested: more than 1500 unit tests
* Additional benefits from Python: fast prototyping, easy to teach, multi-platform

### Creating a Graph

#### Create an empty Graph

Our first example of a graph will be an empty graph. To see the proper mathematical definition of a graph, you can have a look at our previous chapter [Graphs in Python](https://www.python-course.eu/graphs_python.php). The following little Python script uses NetworkX to create an empty graph:

import networkx as nx

G=nx.Graph()

print(G.nodes())

print(G.edges())

print(type(G.nodes()))

print(type(G.edges()))

If we save this script as "empty.py" and start it, we get the following output:

$ python3 empyty.py

[]

[]

<class 'list'>

<class 'list'>

We can see that the result from the graph methods nodes() and edges() are lists.

#### Adding Nodes to our Graph

Now we will add some nodes to our graph. We can add one node with the method add\_node() and a list of nodes with the method add\_nodes\_from():

import networkx as nx

G=nx.Graph()

# adding just one node:

G.add\_node("a")

# a list of nodes:

G.add\_nodes\_from(["b","c"])

print("Nodes of graph: ")

print(G.nodes())

print("Edges of graph: ")

print(G.edges())

#### Adding Edges to our Graph

G can also be created or increased by adding one edge at a time by the method add\_edge(), which has the two nodes of the edge as the two parameters. If we have a tuple or a list as the edge, we can use the asterisk operator to unpack the tuple or the list:

import networkx as nx

G=nx.Graph()

G.add\_node("a")

G.add\_nodes\_from(["b","c"])

G.add\_edge(1,2)

edge = ("d", "e")

G.add\_edge(\*edge)

edge = ("a", "b")

G.add\_edge(\*edge)

print("Nodes of graph: ")

print(G.nodes())

print("Edges of graph: ")

print(G.edges())

In our previous example, the first edge consists of the nodes 1 and 2, which had not been included in our graph so far. The same is true for the second edge with the tuple ("d", "e"). We can see that the nodes will be automatically included as well into the graph, as we can see from the output:

Nodes of graph:

['a', 1, 'c', 'b', 'e', 'd', 2]

Edges of graph:

[('a', 'b'), (1, 2), ('e', 'd')]

We can add a bunch of edges as a list of edges in the form of 2 tuples.

# adding a list of edges:

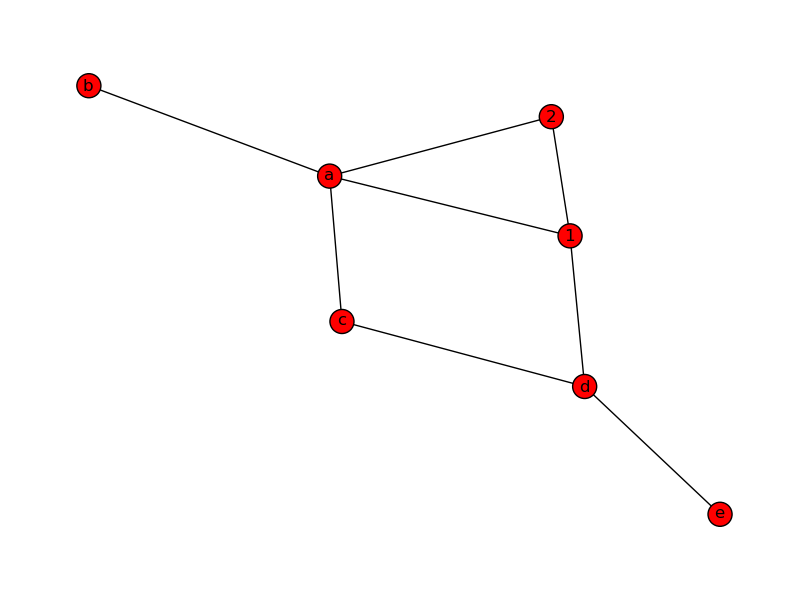
G.add\_edges\_from([("a","c"),("c","d"), ("a",1), (1,"d"), ("a",2)])

We can also print the resulting graph by using matplotlib:

nx.draw(G)

plt.savefig("simple\_path.png") # save as png

plt.show() # display



#### Generate Path Graph

We can create a Path Graph with linearly connected nodes with the method path\_graph(). The Python code code uses matplotlib. pyplot to plot the graph. We will give detailed information on matplotlib at a later stage of the tutorial:

import networkx as nx

import matplotlib.pyplot as plt

G=nx.path\_graph(4)

print("Nodes of graph: ")

print(G.nodes())

print("Edges of graph: ")

print(G.edges())

nx.draw(G)

plt.savefig("path\_graph1.png")

plt.show()