Ricostruire la divina commedia con le catene di Markov

Il nostro scopo è ricostruire la struttura del DNA.

Il problema è che non sappiamo cosa stiamo cercando!

Come possiamo capire se e quanto i nostri metodi funzionano in modo appropriato, e quali features catturano?

Per fare questo, possiamo usare un dataset di prova, che conosciamo bene, per testare se e quanto il nostro metodo sia affidabile e preciso.

Nel nostro caso, useremo la divina commedia come sostituto del DNA.

- sappiamo riconoscere al volo suoni simili all'italiano
- sappiamo distinguere italiano moderno da quello antico
- sappiamo che ha strutture quasi locali (endecasillabo, rima alternata) e così via.

Possiamo testare uno o più modelli per capire come e quanto riproduce queste caratteristiche.

Useremo una versione della divina commedia fornita gratuitamente dal progetto Gutenberg (https://www.gutenberg.org/).

Ho rimosso tutto il testo che non è la divina commedia, inclusi i titoli dei vari canti.

Questo sarà il nostro testo di riferimento

Potete trovare il file sul nostro GitHub:

https://github.com/UniboDIFABiophysics/PLSBigDataNetworks (https://github.com/UniboDIFABiophysics/PLSBigDataNetworks)

Vogliamo unire tutto il testo in un unico flusso di caratteri stampabili.

Per far questo ci dobbiamo appoggiare alla libreria di python chiamata *itertools* che ci fornisce gli strumenti per manipolare gli iteratori

In [1]:
```
import itertools as it
```

useremo due funzioni principali:

- **itertools.chain.from_iterable** per combinare le linee in un flusso unico
- **itertools.islice** per selezionare soltanto una parte del nostro testo invece che tutto

itertools.chain.from_iterable

In [2]:
```
import itertools as it

lista_di_liste = [[1, 2], [3, 4]]

for elemento in lista_di_liste:

    print(elemento)
```

```
[1, 2]
[3, 4]
```

In [3]:
```
for lista in lista_di_liste:

    for elemento in lista:

        print(elemento)
```

```
1
2
3
4
```

In [4]:
```
L = it.chain.from_iterable(lista_di_liste)

for elemento in L:

    print(elemento)
```

```
1
2
3
4
```

itertools.islice

In [5]:
```
lista_lunga = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for elemento in it.islice(lista_lunga, 5):

    print(elemento)
```

```
1
2
3
4
5
```

Per le liste potrei farlo anche in modo più semplice, ma in generale gli iteratori non supportano la sottoselezione in modo semplice.

In [6]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r') as testo:

    pass
```

In [7]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r') as testo:

    head = it.islice(testo, 10)

    for line in head:

        print(line)
```

```
Nel mezzo del cammin di nostra vita

mi ritrovai per una selva oscura,

ché la diritta via era smarrita.


Ahi quanto a dir qual era è cosa dura

esta selva selvaggia e aspra e forte

che nel pensier rinova la paura!


Tant’ è amara che poco è più morte;

ma per trattar del ben ch’i’ vi trovai,
```

In [8]:

```
file = './divinacommedia_cleaned.txt'

# devo leggere il file nel suo encoding, in questo caso non standard

with open(file, 'r', encoding='utf-8-sig') as testo:

    testo = it.chain.from_iterable(testo)

    head = it.islice(testo, 50)

    result = list(head)

    print(result)
```

```
['N', 'e', 'l', ' ', 'm', 'e', 'z', 'z', 'o', ' ', 'd', 'e', 'l', ' ', 'c', 'a
', 'm', 'm', 'i', 'n', ' ', 'd', 'i', ' ', 'n', 'o', 's', 't', 'r', 'a', ' ',
'v', 'i', 't', 'a', '\n', 'm', 'i', ' ', 'r', 'i', 't', 'r', 'o', 'v', 'a', 'i
', ' ', 'p', 'e']
```

Per rendere più semplice la nostra analisi, convertiamo tutto in minuscolo, in modo che le maiuscole non vengano viste come lettere differenti.

In [9]:
```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    head = it.islice(testo, 50)

    result = list(head)

    print(result)
```

```
['n', 'e', 'l', ' ', 'm', 'e', 'z', 'z', 'o', ' ', 'd', 'e', 'l', ' ', 'c', 'a
', 'm', 'm', 'i', 'n', ' ', 'd', 'i', ' ', 'n', 'o', 's', 't', 'r', 'a', ' ',
'v', 'i', 't', 'a', '\n', 'm', 'i', ' ', 'r', 'i', 't', 'r', 'o', 'v', 'a', 'i
', ' ', 'p', 'e']
```

Ora iniziamo con il nostro modello più semplice:

$\rightarrow$ **generiamo il testo semplicemente ripetendo le lettere in base a quanto sono frequenti**

Questo ci richiede per prima cosa di valutare la frequenza di queste lettere!

Per far questo, abbiamo uno strumento già predisposo, la classe **Counter**.

In [10]:
```
from collections import Counter

lettere = "aaaabb"

conteggi = Counter(lettere)

conteggi.most_common()
```

Out[10]:
```
[('a', 4), ('b', 2)]
```

In [11]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    head = it.islice(testo, None)

    conteggi = Counter(head)

    print(conteggi)
```

```
Counter({' ': 83019, 'e': 46680, 'a': 42412, 'i': 39559, 'o': 37610, 'n': 2648
7, 'r': 25922, 'l': 23498, 't': 22761, 's': 22631, 'c': 20821, '\n': 19053, 'd
': 15006, 'u': 13451, 'm': 12086, 'p': 11263, ',': 8513, 'v': 8168, '’': 7623,
'g': 7320, 'h': 7111, 'f': 5127, 'q': 3353, '.': 3275, 'b': 2981, 'z': 1855,
';': 1628, 'ì': 1383, 'ù': 1080, '«': 1062, '»': 1062, ':': 988, 'ò': 938, 'è
': 927, 'é': 903, 'à': 855, 'ï': 428, '?': 278, '!': 232, '‘': 109, 'ë': 85, '
ü': 55, '“': 51, '”': 51, 'ó': 30, '—': 18, 'ä': 8, 'x': 3, '(': 3, ')': 3, 'j
': 2, 'ö': 1, 'y': 1, '-': 1})
```

In [12]:

```
from pprint import pprint

pprint( conteggi.most_common(20) )
```

```
[(' ', 83019),
 ('e', 46680),
 ('a', 42412),
 ('i', 39559),
 ('o', 37610),
 ('n', 26487),
 ('r', 25922),
 ('l', 23498),
 ('t', 22761),
 ('s', 22631),
 ('c', 20821),
 ('\n', 19053),
 ('d', 15006),
 ('u', 13451),
 ('m', 12086),
 ('p', 11263),
 (',', 8513),
 ('v', 8168),
 ('’', 7623),
 ('g', 7320)]
```

Ora vogliamo estrarre le lettere a caso in modo proporzionale a quanto le abbiamo visto di frequente.

Per far questo, possiamo usare la funzione **choices** della libreria **random**.

In [13]:
```
from random import choices

lettere = ['a', 'b']

frequenze = [9, 1]

choices(lettere, weights=frequenze, k=10)
```

Out[13]:
```
['a', 'a', 'a', 'a', 'a', 'a', 'b', 'a', 'b', 'a']
```

In [14]:
```
# per chi avesse python 3.5 o più vecchio

from numpy.random import choice
from numpy import array

def choices(lista, weights, k=1):
    weights = array(weights)/sum(weights)
    res = choice(lista, k, p=weights)
    return list(res)

lettere = ['a', 'b']
frequenze = [9, 1]

choices(lettere, weights=frequenze, k=10)
```

Out[14]:
```
['a', 'a', 'b', 'a', 'a', 'b', 'b', 'a', 'a', 'a']
```

se volessimo usarlo per generare il testo, possiamo unire tutte queste lettere con un join

In [15]:
```
random = choices(lettere, weights=frequenze, k=10)

print(random)

testo = str.join('', random)

print(testo)
```

```
['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
aaaaaaaaaa
```

Possiamo usare **choices** in combinazione con **Counter** in modo semplice

In [16]:

```
lettere = "aaaabb"

conteggi = Counter(lettere)

print(conteggi.keys(), conteggi.values())
```

```
dict_keys(['a', 'b']) dict_values([4, 2])
```

In [17]:

```
lettere = "ciao mondo"

conteggi = Counter(lettere)

lettere = list(conteggi.keys())

frequenze = list(conteggi.values())

random = choices(lettere, weights=frequenze, k=200)

testo = str.join('', random)

print(testo)
```

```
ooocooo dmamoao  omoccd ninooo ooc d oinoa ano ddo docmiod d odooodooamnaodond
odcacaonimioondadmo coo ocdccondci ooioodoommoii odnononinoomooooaomi ooimnoco
oonadamnninmainiacimomocoacc o omoacoooonnda
```

In [18]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    head = it.islice(testo, 100)

    conteggi = Counter(head)


lettere = list(conteggi.keys())

frequenze = list(conteggi.values())

random = choices(lettere, weights=frequenze, k=200)

testo = str.join('', random)

print(testo)
```

```
 ao,zpa zvv rr  rtv ihvdtri
cavivr ti dsto d v eaavai,riazsc vi ranitmau
eovsiatacai rr lci ,i unmrr a nhercasiad v
tptrevveatrzaa   uc umas  e,cizmraumcasmae
ii p  rtvnim
i vaod iil  ah véuireses v
```

Il risultato è chiaramente deludente.

Non solo non assomiglia ad una frase italiana...non sembrano neppure parole!

Cosa sta succedendo?

Sappiamo che nell'italiano le parole non sono composte semplicemente dai suoni, ma da come i suoni i susseguono l'un l'altro.

Per rappresentare questa conoscenza, possiamo utilizzare un metodo della fisica matematica chiamato **Catene di Markov**.

Le catene di Markov

L'idea è semplice: invece di generare ogni lettera sulla base della sua frequenza nel testo, facciamo un passaggio leggermente più intelligente.

- Guardo l'ultima lettera del testo che ho generato finora.
- Considero nel testo originale quanto spesso ciascuna lettera segue la lettera che ho scelto.
- Genero la mia nuova lettera sulla base di questa **probabilità condizionata**.
- Ricomincio usando l'ultima lettera generata come punto di partenza.

Questo metodo è lo stesso (di base) usato dai cellulari per il suggerimento delle parole.

Se scrivete spesso "buona notte", "buona mattina" e "buona cena", una volta che inserite la parola "buona" da tastiera, il software sa che probabilmente dopo andrete ad inserire "notte", "mattina" o "cena".

Se non ha a disposizione un buon campione personalizzato, utilizza quelli delle persone della stessa lingua.

Questo ha creato non poco imbarazzo nel caso di frasi volgari o razziste, inserite dall'algoritmo come proposte per via del loro uso frequente (vedi il caso recente dell'IPhone che suggeriva la frase "Vesuvio lavali col fuoco")

Per ciascuna lettera dell'alfabeto dovremo quindi avere un **Counter** che mi tenga conto di quali lettere potranno seguire.

Per poter fare questo devo unire le lettere del mio testo in coppie consecutive di lettere!

Per fare questo uso una libreria esterna, **toolz**, che mi fornisce la funzione **sliding_window** che fa proprio questo

In [20]:

```
from toolz import sliding_window

lettere = "abcdefg"

coppie = sliding_window(2, lettere)

print(list(coppie))
```

```
[('a', 'b'), ('b', 'c'), ('c', 'd'), ('d', 'e'), ('e', 'f'), ('f', 'g')]
```

In [21]:

```
# per chi non potesse importare toolz

from collections import deque

def sliding_window(n, seq):

    mazzo = deque(maxlen=n)

    for value in seq:

        mazzo.append(value)

        if len(mazzo)==n:

            yield tuple(mazzo)

coppie = sliding_window(2, "abcde")

print(list(coppie))
```

```
[('a', 'b'), ('b', 'c'), ('c', 'd'), ('d', 'e')]
```

WARNING

Useremo degli algoritmi assolutamente non ottimali!

In questo caso ci servono solo per capire il procedimento, nella vita reale sono troppo lenti per funzionare davvero!

In [22]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    head = it.islice(testo, 30)

    coppie = sliding_window(2, head)

    pprint(list(coppie))
```

```
[('n', 'e'),
 ('e', 'l'),
 ('l', ' '),
 (' ', 'm'),
 ('m', 'e'),
 ('e', 'z'),
 ('z', 'z'),
 ('z', 'o'),
 ('o', ' '),
 (' ', 'd'),
 ('d', 'e'),
 ('e', 'l'),
 ('l', ' '),
 (' ', 'c'),
 ('c', 'a'),
 ('a', 'm'),
 ('m', 'm'),
 ('m', 'i'),
 ('i', 'n'),
 ('n', ' '),
 (' ', 'd'),
 ('d', 'i'),
 ('i', ' '),
 (' ', 'n'),
 ('n', 'o'),
```

quali sono le coppie più comuni?

In [23]:
```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    # head = it.islice(testo, 300)

    coppie = sliding_window(2, testo)

    conteggi = Counter(coppie)

    pprint(conteggi.most_common(10))
```

```
[(('e', ' '), 17397),
 (('a', ' '), 13949),
 (('i', ' '), 11636),
 (('o', ' '), 10906),
 ((' ', 'c'), 9579),
 ((' ', 's'), 8676),
 ((' ', 'l'), 7659),
 ((' ', 'd'), 6975),
 (('e', 'r'), 6524),
 (('c', 'h'), 6523)]
```

Possiamo calcolare le frequenze delle coppie con "buona" precisione a partire dalla divina commedia?

Quante combinazioni posso avere?

Se consideriamo solo le lettere, gli spazi e le lettere accentate, abbiamo circa 30 caratteri.

Tutte le coppie sono date da $30^2 = 900$

Se tutte queste coppie fossero equiprobabili e ne volessimo avere almeno 10 campionamenti per coppia, dovremmo avere circa 10'000 lettere.

Quante lettere abbiamo in totale nella nostra divina commedia?

In [25]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    conteggio = sum(1 for c in testo)

print(conteggio)
```

```
529770
```

Quando useremo le triplette, avremo $30^3 = 27'000$ combinazioni, quindi avremo bisogno di circa 300'000 lettere per avere una buona copertura (cioè una buona stima delle frequenze).

Questo ci dice che non potremo davvero andare oltre le triplette, almeno senza renderci le cose difficili.

In [26]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    coppie = sliding_window(2, testo)

    conteggi = Counter(coppie)

pprint(conteggi.most_common(10))

# Quante coppie ci sono?

print(len(conteggi))
```

```
[(('e', ' '), 17397),
 (('a', ' '), 13949),
 (('i', ' '), 11636),
 (('o', ' '), 10906),
 ((' ', 'c'), 9579),
 ((' ', 's'), 8676),
 ((' ', 'l'), 7659),
 ((' ', 'd'), 6975),
 (('e', 'r'), 6524),
 (('c', 'h'), 6523)]
855
```

Anche considerando solo le coppie, non riusciamo a coprire tutti i casi possibili!

Però iniziamo già a vedere una prima feature della lingua italiana: le parole finiscono spesso con delle vocali.

Vediamo ora come possiamo sfruttare questo dizionario di conteggi per generare il nostro testo.

Dobbiamo partire da un **seme**, un elemento di testo iniziale.

Nel nostro caso, per cercare di rimanere il più generici possibile, useremo uno spazio vuoto.

Per poter generare la nuova lettera a partire dalla vecchia, dovremo selezionare soltanto quella parte del nostro dizionario che contiene la nostra lettera come parte iniziale.

Possiamo usare la funzione **item** dei dizionari per avere la sequenza delle coppie chiave valore, e filtrarle in base al loro punto di inizio

In [27]:
```
dizionario = {'a':1, 'b':2, 'c':3}
print(dizionario.items())
```

```
dict_items([('a', 1), ('b', 2), ('c', 3)])
```

In [28]:
```
chiavi = [c for c, v in dizionario.items() if c in ['b', 'c']]

print(chiavi)
```

```
['b', 'c']
```

In [29]:
```
# Equivalente a:

chiavi = []

for c,v in dizionario.items():

    if c in ['b','c']:

        chiavi.append(c)

print(chiavi)
```

```
['b', 'c']
```

In [30]:
```
dizionario = {'a':1, 'b':2, 'c':3}

chiavi = [c for c, v in dizionario.items() if c in ['b', 'c']]

valori = [v for c, v in dizionario.items() if c in ['b', 'c']]

print(chiavi, valori)
```

```
['b', 'c'] [2, 3]
```

In [31]:
```
conteggi_fake = {('a', 'b'):1, ('b', 'c'): 2, ('a', 'p'): 3}

lettera = 'a'

chiavi = [c for c, v in conteggi_fake.items() if c[0]==lettera]

chiavi
```

Out[31]:
```
[('a', 'b'), ('a', 'p')]
```

In [32]:
```
def inizia_con(sequence, letter):

    return sequence[0]==letter
```

In [33]:
```
conteggi_fake = {('a', 'b'):1, ('b', 'c'): 2, ('a', 'p'): 3}

lettera = 'a'

chiavi = [c for c, v in conteggi_fake.items() if inizia_con(c, lettera)]

chiavi
```

Out[33]:
```
[('a', 'b'), ('a', 'p')]
```

In [34]:
```
conteggi_fake = {('a', 'b'):1, ('b', 'c'): 2, ('a', 'p'): 3}

lettera = 'a'

# c[-1] ritorna l'ultimo elemento di c

chiavi = [c[-1] for c, v in conteggi_fake.items() if inizia_con(c, lettera)]

valori = [v for c, v in conteggi_fake.items() if inizia_con(c, lettera)]

chiavi, valori
```

Out[34]:
```
(['b', 'p'], [1, 3])
```

Ora possiamo fare lo stesso con il nostro vettore dei conteggi ottenuto dalla divina commedia

In [36]:

```
nuovo_testo = ['e']

for i in range(10):

    lettera = nuovo_testo[-1]

    lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, lettera)]

    frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]

    prossima_lettera = choices(lettere, frequenze)[0]

    print(lettera, prossima_lettera)
```

```
e
e
e
e s
e
e r
e
e
e t
e m
```

Ci siamo dimenticati di attaccare l'ultima lettera generata al nostro testo!

In [37]:
```
nuovo_testo = [' ']

for i in range(10):

    lettera = nuovo_testo[-1]

    lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, lettera)]

    frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]

    prossima_lettera = choices(lettere, frequenze)[0]

    print(lettera, prossima_lettera)

    nuovo_testo.append(prossima_lettera)

print(nuovo_testo)
```

```
  c
c o
o n
n z
z i
i l
l e
e
  m
m e
[' ', 'c', 'o', 'n', 'z', 'i', 'l', 'e', ' ', 'm', 'e']
```

In [38]:
```
print(str.join('', nuovo_testo))
```

```
conzile me
```

Proviamo ora un testo più lungo!

In [39]:
```
nuovo_testo = [' ']

for i in range(200):

    lettera = nuovo_testo[-1]

    lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, lettera)]

    frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]

    prossima_lettera = choices(lettere, frequenze)[0]

    nuovo_testo.append(prossima_lettera)

print(str.join('', nuovo_testo))
```

```
 ’i nchel ma;
ndovëan’ated bea livevarili.
ppi posstoseno pua, di erimer simi ci lu connde m’a mar, ci.
cce cue?», trerve suer’ v’iedera ’l cie fir n ranenglena iorar lasum’ate lo, u
oipura cco,
iri ca
```

Il risultato lascia ancora a desiderare...

Proviamo ad estendere alle triplette: in questo caso consideriamo le coppie di lettere come punto di partenza e la nuova lettera sulla base di queste.

Dobbiamo però ricostruire il nostro intero dataset dei conteggi.

In [40]:

```
file = './divinacommedia_cleaned.txt'

with open(file, 'r', encoding='utf-8-sig') as testo:

    linee = (linea.lower() for linea in testo)

    testo = it.chain.from_iterable(linee)

    coppie = sliding_window(3, testo)

    conteggi = Counter(coppie)

print(len(conteggi))

pprint(conteggi.most_common(5))
```

```
6059
[((' ', 'c', 'h'), 4036),
 (('c', 'h', 'e'), 3871),
 (('h', 'e', ' '), 3630),
 ((' ', 'd', 'i'), 3338),
 (('.', '\n', '\n'), 3171)]
```

Il procedimento per generare il testo è assolutamente identico, ma il controllo sarà sulle ultime due lettere e non soltanto sull'ultima.

In [41]:

```
nuovo_testo = ['c', 'h']

for i in range(10):

    lettera = tuple(nuovo_testo[-2:])

    lettere = [c[-1] for c, v in conteggi.items() if c[:2]==lettera]

    frequenze = [v for c, v in conteggi.items() if c[:2]==lettera]

    prossima_lettera = choices(lettere, frequenze)[0]

    print(lettera, repr(prossima_lettera))

    nuovo_testo.append(prossima_lettera)

print(str.join('', nuovo_testo))
```

```
('c', 'h') 'i'
('h', 'i') ' '
('i', ' ') 'd'
(' ', 'd') 'i'
('d', 'i') 's'
('i', 's') 's'
('s', 's') 'e'
('s', 'e') ' '
('e', ' ') 'p'
(' ', 'p') 'i'
chi disse pi
```

In [42]:

```
def inizia_con(sequence, letter):

    return sequence[:len(letter)]==letter
```

In [43]:

```
nuovo_testo = ['c', 'h']

for i in range(10):

    lettera = tuple(nuovo_testo[-2:])

    lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, lettera)]

    frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]

    prossima_lettera = choices(lettere, frequenze)[0]

    print(lettera, repr(prossima_lettera))

    nuovo_testo.append(prossima_lettera)

print(str.join('', nuovo_testo))
```

```
('c', 'h') 'i'
('h', 'i') 'a'
('i', 'a') ' '
('a', ' ') 'c'
(' ', 'c') 'o'
('c', 'o') 'l'
('o', 'l') 't'
('l', 't') 'r'
('t', 'r') 'a'
('r', 'a') 'b'
chia coltrab
```

Possiamo facilmente estendere questo procedimento ad ancora più lettere.

Vedremo che con l'aumentare della lunghezza della sequenza avrò parole sempre più simili all'italiano, ma il numero di casi disponibili calerà sensibilmente!

In [45]:

```
L = 3
file = './divinacommedia_cleaned.txt'
with open(file, 'r', encoding='utf-8-sig') as testo:
    linee = (linea.lower() for linea in testo)
    testo = it.chain.from_iterable(linee)
    coppie = sliding_window(L+1, testo)
    conteggi = Counter(coppie)

nuovo_testo = ['n', 'e', 'l']
for i in range(200):
    lettera = tuple(nuovo_testo[-L:])
    lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, lettera)]
    frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]
    prossima_lettera = choices(lettere, frequenze)[0]
    nuovo_testo.append(prossima_lettera)
print(str.join('', nuovo_testo))
```

```
nel dissile mal piego spescrista;
percuote
lo forte,
là givi de’ mezzo scesso ciaschiar mando, in quel cota ghesensarà ti ancia tan
ta,

amma inver’ ell’ è contende, segno al quando, e foco morta d’un «or
```

Posso generalizzare questo procedimento con una funzione per generare dei testi lunghi a piacere, usando una funzione.

In [46]:

```
def genera_testo(nuovo_testo='nel', lunghezza=2):
    L = lunghezza
    nuovo_testo = list(nuovo_testo)
    file = './divinacommedia_cleaned.txt'
    with open(file, 'r', encoding='utf-8-sig') as testo:
        linee = (linea.lower() for linea in testo)
        testo = it.chain.from_iterable(linee)
        coppie = sliding_window(L+1, testo)
        conteggi = Counter(coppie)

    while True:
        lettera = tuple(nuovo_testo[-L:])
        lettere = [c[-1] for c, v in conteggi.items() if inizia_con(c, letter
a)]
        frequenze = [v for c, v in conteggi.items() if inizia_con(c, lettera)]
        prossima_lettera = choices(lettere, frequenze)[0]
        nuovo_testo.append(prossima_lettera)
        nuovo_testo = nuovo_testo[1:]
        yield prossima_lettera
```

In [47]:
```
testo = str.join("", it.islice(genera_testo('nel', 2), 200))
print(testo)
```

```
vedo sitora inume,
percomio rizio ava ti falta te,

chia”,
te sì che luina ’l gento der temò che no pede dice;

e pio dercarvidalto;
e mai ai smo ghi guardatti sempiendesmutte ’l milo a che che per mo
```

In [48]:
```
testo = str.join("", it.islice(genera_testo('che', 2), 200))
print(testo)
```

```
 le riche di veatia, per chi fu, al storte
sì cam si pa, ma qui onsa;

postropossol che nesa ada in d’oglie;
coltolsentà riasci le to,
fu vosto in mi sandel ma
perché no miaredio a lio a subbuolvanici
```

In [53]:
```
testo = str.join("", it.islice(genera_testo('nel', 3), 200))
print(testo)
```

```
 chi maltro a la vità, comico.

qualude l’uom sotto,
«ben follatorna;
le fermare,

sor sul che de e quelle,
non fu dolce non si stes igura».

quando sé ismandi tu verse
non saravam là già che hanno ci
```

In [46]:
```
testo = str.join("", it.islice(genera_testo('nel ', 4), 200))
print(testo)
```

```
ciglia,
ci ad ogne la la giù, non stipa;
«e or veroìs, trifon mi volli è di piè cupido segue e di morta parrebber li pi
eta.

e stesso guado.

l’odïernol, ch’altro frutto pome.

così divizia donai.

lo
```

In [47]:
```
testo = str.join("", it.islice(genera_testo('onore', 5), 200))
print(testo)
```

```
di santa zita!
mette».

ed elli scese,
contenne.

ma del viso ch’io era conforma un che ti smarrito riso;
ma più tenem per cui dischiude,
non pur l’ombra, li staman, quale ha posso
com’ esser monferr
```

Possiamo vedere come il testo generato assomigli all'italiano in modo superficiale, ma le frasi non hanno senso.

Potremmo usare la nostra conoscenza dell'esistenza delle parole per generare delle frasi più sensate, ma saremmo sempre ciechi alle strutture di ordine superiore.

extra: Generazione Markoviana delle parole

uso lo stesso algoritmo di prima, ma divido il testo in parole invece che in singole lettere.

Per la generazione devo evitare di ricalcolare ogni volta tutti i conteggi, ma li salvo di volta in volta in modo che se trovo due parole uguali ricarico il valore vecchio. Questo non è ancora perfetto, ma almeno mi salvo molti calcoli inutili!

In [54]:

```
import string
L = 2
with open(file, 'r', encoding='utf-8-sig') as testo:
    linee = (linea.lower() for linea in testo)
    caratteri = it.chain.from_iterable(linee)
    gruppi = it.groupby(caratteri, lambda s: s in string.whitespace)
    parole = (gruppo for (is_whitespace, gruppo) in gruppi if not is_whitespac
e)
    testo = (str.join('', parola) for parola in parole)
    coppie = sliding_window(L, testo)
    head = it.islice(coppie, 20)
    conteggi = Counter(head)
print(conteggi)
```

```
Counter({('nel', 'mezzo'): 1, ('mezzo', 'del'): 1, ('del', 'cammin'): 1, ('cam
min', 'di'): 1, ('di', 'nostra'): 1, ('nostra', 'vita'): 1, ('vita', 'mi'): 1,
('mi', 'ritrovai'): 1, ('ritrovai', 'per'): 1, ('per', 'una'): 1, ('una', 'sel
va'): 1, ('selva', 'oscura,'): 1, ('oscura,', 'ché'): 1, ('ché', 'la'): 1, ('l
a', 'diritta'): 1, ('diritta', 'via'): 1, ('via', 'era'): 1, ('era', 'smarrit
a.'): 1, ('smarrita.', 'ahi'): 1, ('ahi', 'quanto'): 1})
```

In [55]:

```
def genera_testo_parole(nuovo_testo=['nel'], lunghezza=2):
    L = lunghezza
    nuovo_testo = list(nuovo_testo)
    file = './divinacommedia_cleaned.txt'
    with open(file, 'r', encoding='utf-8-sig') as testo:
        linee = (linea.lower() for linea in testo)
        caratteri = it.chain.from_iterable(linee)
        gruppi = it.groupby(caratteri, lambda s: s in string.whitespace)
        parole = (gruppo for (is_whitespace, gruppo) in gruppi if not is_whites
pace)
        testo = (str.join('', parola) for parola in parole)
        coppie = sliding_window(L+1, testo)
        conteggi = Counter(coppie)

    coppie_memoize = dict()
    while True:
        lettera = tuple(nuovo_testo[-L:])
        try:
            coppie = coppie_memoize[lettera]
        except KeyError:
            coppie = [(c[-1], v) for c, v in conteggi.items() if c[:L]==letter
a]
            coppie_memoize[lettera] = coppie

        lettere = [c for c, v in coppie]
        frequenze = [v for c, v in coppie]
        prossima_lettera = choices(lettere, frequenze)[0]
        nuovo_testo.append(prossima_lettera)
        nuovo_testo = nuovo_testo[1:]
        yield prossima_lettera
```

In [50]:
```
testo = str.join(" ", it.islice(genera_testo_parole(['nel', 'mezzo'], 2), 200))
print(testo)
```

```
s’avvivava, e d’ogne parte i pesi. e se’ or sotto l’emisperio giunto ch’è cont
raposto a quel ch’i’ dico, gloria di latin», disse, «per cui mostrò ciò che ce
la ’l vapor che ’l guasco l’alto arrigo inganni, parran faville de la pietra n
on diversi. e poi fui famiglia del buon gherardo e guido di carpigna? oh romag
nuoli tornati in bastardi! quando in faenza un bernardin di fosco, verga genti
l di picciola gente; sì che tu per certo verria meno per lo natural costume, l
e pole insieme, al cominciar ne lagrimai. diverse lingue, orribili favelle, pa
role di gran voce, sì ch’ogne musa ne sarebbe schiva. qual è quei che diceva p
ria, «tu parli d’arno». e l’altro foro aiutò sì che ’n sì distesa lingua lo di
cer mio, ch’al tuo sentir si sterna, ove dinanzi dissi: “u’ ben s’impingua, se
non fosse umilïato ad incarnarsi. or per empierti bene ogne disio, ritorno a c
ompiér lo cammin corto parte lo secondo giron dal terzo, e dove sile e cagnan
s’accompagna, tal signoreggia e va con la lancia e coi remi, quantunque può, c
iascun pinger sua barca»; dritto sì come il fiammeggiar ti manifesta. ma l’alt
a carità, che fa la ragna. piangerà feltro ancora la
```

In [51]:
```
testo = str.join(" ", it.islice(genera_testo_parole(['nel', 'mezzo', 'del'],
3), 200))
print(testo)
```

```
cammin di nostra vita dipartille. poscia ch’io ebbi rotta la persona di due pu
nte mortali, io mi rendei, piangendo, a quei che volontier perdona. orribil fu
ron li peccati miei; ma la bontà infinita ha sì gran braccia, che prende ciò c
he si vuole, e avea galigaio dorata in casa sua già l’elsa e ’l pome. grand’ e
ra già la colonna del vaio, sacchetti, giuochi, fifanti e barucci e galli e qu
ei ch’arrossan per lo staio. lo ceppo di che nacquero i calfucci era già grand
e, e già eran tratti a le curule sizii e arrigucci. oh quali io vidi quei che
son disfatti per lor superbia! e le palle de l’oro fiorian fiorenza in tutt’ i
suoi gran fatti. così facieno i padri di coloro che, sempre che la vostra mise
ria non mi tange, né fiamma d’esto ’ncendio non m’assale. donna è gentil nel c
iel che più si spieghi di nostra condizion com’ ell’ è ora; e se rimane, dite
come, poi che sarete visibili rifatti, esser porà ch’al veder non vi nòi». «do
nna del ciel, di queste cose accorta», rispuose ’l mio maestro sorrise di tant
o; e più d’onore ancora assai mi fenno, ch’e’ sì mi fecer de la
```

Se vado a guardare il risultato, risulta essere pezzi di frasi uniche mescolati insieme in base ai pezzi in comune (le congiunzioni). Questo capita perchè le possibili combinazioni di parole sono così tante che anche con tutta la divina commedia, sto sottocampionando le varie possibilità.

In particolare, con sequenze di tre parole, sto ripescando le frasi orginali per intero perchè data una tripletta di parole sono praticamente uniche.

In [ ]: