Oppgave 4, Språkmodeller og ordklasser

January 24, 2022

1 NLTK og språkmodeller

Målet med denne oppgaven er å implementere to språkmodeller (en unigrammodell og en trigram-modell) ved hjelp av Natural Language Toolkit (NLTK) i Python.

Vi skal fokusere på følgende punkter:

- Få tilgang til korpora i NLTK.
- Beregne frekvenser og såkalte frekvensdistribusjoner.
- Trene en språkmodell ved å beregne sannsynligheter fra frekvensdistribusjonene.
- Bruke språkmodellen til å beregne unigram- og trigram-sannsynligheter.
- Bruke en språkmodell til å generere tekst.

Det er viktig at du jobber deg gjennom følgende steg, og at du forstår hva som skjer ved hvert steg. Denne oppgaven er svært sentral for den neste obligatoriske innleveringen i IN1140 (oblig 2a). Vi anbefaler at du også repeterer pensum fra forelesningen om språkmodeller før du går gjennom koden her og gjør oppgavene som kommer til slutt.

Vi anbefaler at du skriver inn koden og kjører den på din egen maskin ettersom du går gjennom den.

1.1 NLTK

Natural Language Toolkit (NLTK) er en Python-modul som brukes til å skrive programmer for språkprossessering. NLTK gir tilgang til både korpora og innebygde metoder. Mer informasjon om NLTK finner du her. NLTK er installert på IFIs maskiner. Dersom du ønsker å installere det på egen maskin er det instruksjoner både på emnesiden og på NLTK-siden her.

For a kunne bruke NLTK i koden din, kan den importeres slik:

[1]:
```
import nltk
```

Som nevnt tidligere, gir NLTK tilgang til en del korpora. For eksempel *Gutenberg*-korpuset som inneholder en rekke eldre bøker, *Brown*-korpuset som inneholder romaner, noveller og nyhetsartikler, *Reuters*-korpuset som inneholder nyheter, og *Inaugural Address*-korpuset som inneholder amerikanske presidenters innledende taler (oppdateres fortløpende per dags dato). Hvert korpus inneholder et visst antall tekster. Du kan lese mer om dette i NLTK-boka her.

Her skal vi jobbe med Inaugural-korpuset. Selvom vi bruker Inaugural-korpuset i de følgende eksemplene, kan korpusmetodene vi går gjennom her er brukes på andre NLTK-korpora også. For å importere dette korpuset, skriver du følgende:

[2]:
```
from nltk.corpus import inaugural
```

Merk at dette krever at du har lastet ned NLTK-data. Se emnesiden eller veiledning her.

1.1.1 Metoden `fileids()`

Brukes på et korpus og returnerer en liste med alle filnavn i korpuset.

[3]:
```
print(inaugural.fileids())
```

```
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt',
'1801-Jefferson.txt', '1805-Jefferson.txt', '1809-Madison.txt',
'1813-Madison.txt', '1817-Monroe.txt', '1821-Monroe.txt', '1825-Adams.txt',
'1829-Jackson.txt', '1833-Jackson.txt', '1837-VanBuren.txt',
'1841-Harrison.txt', '1845-Polk.txt', '1849-Taylor.txt', '1853-Pierce.txt',
'1857-Buchanan.txt', '1861-Lincoln.txt', '1865-Lincoln.txt', '1869-Grant.txt',
'1873-Grant.txt', '1877-Hayes.txt', '1881-Garfield.txt', '1885-Cleveland.txt',
'1889-Harrison.txt', '1893-Cleveland.txt', '1897-McKinley.txt',
'1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt', '1913-Wilson.txt',
'1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt', '1929-Hoover.txt',
'1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt',
'1945-Roosevelt.txt', '1949-Truman.txt', '1953-Eisenhower.txt',
'1957-Eisenhower.txt', '1961-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt',
'1973-Nixon.txt', '1977-Carter.txt', '1981-Reagan.txt', '1985-Reagan.txt',
'1989-Bush.txt', '1993-Clinton.txt', '1997-Clinton.txt', '2001-Bush.txt',
'2005-Bush.txt', '2009-Obama.txt', '2013-Obama.txt', '2017-Trump.txt',
'2021-Biden.txt']
```

I skrivende stund inneholder korpuset presidenttaler fra USA fra 2017 og helt tilbake til 1789.

Listen kan naturligvis også tas vare på i en variabel:

[4]:
```
alle_filer = inaugural.fileids()
```

1.1.2 Hente ut tekst, ord og setninger

Det kan virke intuitivt å bruke metoden `open()` for å lese inn korpusinnholdet, men NLTK-korpora inneholder egne metoder til dette formålet. Disse skal vi se på nå.

Metoden `raw()` brukes på et korpus. Som argument sender vi med navnet på filen vi er interesserte i. Denne filen må være i korpuset metoden kalles på. Dobbeltsjekk stavemåte og at filnavnet eksisterer i listen gitt av `fileids()` hvis du får en feilmelding. Metoden returnerer én streng med hele filinnholdet.

[5]:
```
kennedy_tekst = inaugural.raw('1961-Kennedy.txt')

# Skriver kun ut de 500 første tegnene
print(kennedy_tekst[:500])
```

```
Vice President Johnson, Mr. Speaker, Mr. Chief Justice, President Eisenhower,
Vice President Nixon, President Truman, reverend clergy, fellow citizens, we
```

```
observe today not a victory of party, but a celebration of freedom --
symbolizing an end, as well as a beginning -- signifying renewal, as well as
change. For I have sworn I before you and Almighty God the same solemn oath our
forebears l prescribed nearly a century and three quarters ago.

The world is very different now. For man holds in his
```

Metoden `words()` brukes på samme måte. Metoden returnerer en tokenisert utgave av teksten i form av en liste.

[6]:
```
kennedy_ord = inaugural.words('1961-Kennedy.txt')

# Skriver kun ut de 100 første ordene
print(kennedy_ord[:100])
```

```
['Vice', 'President', 'Johnson', ',', 'Mr', '.', 'Speaker', ',', 'Mr', '.',
'Chief', 'Justice', ',', 'President', 'Eisenhower', ',', 'Vice', 'President',
'Nixon', ',', 'President', 'Truman', ',', 'reverend', 'clergy', ',', 'fellow',
'citizens', ',', 'we', 'observe', 'today', 'not', 'a', 'victory', 'of', 'party',
',', 'but', 'a', 'celebration', 'of', 'freedom', '--', 'symbolizing', 'an',
'end', ',', 'as', 'well', 'as', 'a', 'beginning', '--', 'signifying', 'renewal',
',', 'as', 'well', 'as', 'change', '.', 'For', 'I', 'have', 'sworn', 'I',
'before', 'you', 'and', 'Almighty', 'God', 'the', 'same', 'solemn', 'oath',
'our', 'forebears', 'l', 'prescribed', 'nearly', 'a', 'century', 'and', 'three',
'quarters', 'ago', '.', 'The', 'world', 'is', 'very', 'different', 'now', '.',
'For', 'man', 'holds', 'in', 'his']
```

Når vi har tilgang til alle ordene, kan vi enkelt finne antall token i teksten slik:

[7]:
```
antall_token = len(kennedy_ord)
print('Antall token:', antall_token)
```

```
Antall token: 1546
```

Vi kan også lage en liste med ordtypene i teksten. Vi bør da gjøre alle bokstavene små, slik at f.eks. ikke *The* og *the* regnes som ulike ordtyper.

[8]:
```
# Lager først en tom liste
ordtyper = []

# Gjør alle bokstaver små og legger til
for word in kennedy_ord:
    ordtyper.append(word.lower())

# Fjerner duplikater
ordtyper = set(ordtyper)
```

Listen med ordtypene kan også lages slik:

[9]:
```
ordtyper = {word.lower() for word in kennedy_ord}
```

Dette kalles gjerne *"list comprehension"*. Denne måten å lage lister og sett på brukes hyppig på høyere nivå og kan være nyttig å forså. Se ressurser på emnesiden / fra gruppetimer og Trix-oppgaver her.

Antall ordtyper finner vi nå enkelt slik:

[10]:
```
antall_ordtyper = len(ordtyper)
print('Antall ordtyper:', antall_ordtyper)
```

```
Antall ordtyper: 546
```

Metoden `sents()` brukes på samme måte. Metoden returnerer en nøstet liste som representerer hhv. setningene og ordene i hver setning.

[11]:
```
kennedy_setninger = inaugural.sents('1961-Kennedy.txt')

# Skriver kun ut de 3 første setningene
print(kennedy_setninger[:3])
```

```
[['Vice', 'President', 'Johnson', ',', 'Mr', '.', 'Speaker', ',', 'Mr', '.',
'Chief', 'Justice', ',', 'President', 'Eisenhower', ',', 'Vice', 'President',
'Nixon', ',', 'President', 'Truman', ',', 'reverend', 'clergy', ',', 'fellow',
'citizens', ',', 'we', 'observe', 'today', 'not', 'a', 'victory', 'of', 'party',
',', 'but', 'a', 'celebration', 'of', 'freedom', '--', 'symbolizing', 'an',
'end', ',', 'as', 'well', 'as', 'a', 'beginning', '--', 'signifying', 'renewal',
',', 'as', 'well', 'as', 'change', '.'], ['For', 'I', 'have', 'sworn', 'I',
'before', 'you', 'and', 'Almighty', 'God', 'the', 'same', 'solemn', 'oath',
'our', 'forebears', 'l', 'prescribed', 'nearly', 'a', 'century', 'and', 'three',
'quarters', 'ago', '.'], ['The', 'world', 'is', 'very', 'different', 'now',
'.']]
```

1.2 Frekvensdistribusjon

En frekvensdistribusjon registrerer antall ganger hvert utfall av et eksperiment har skjedd. For eksempel kan en frekvensdistribusjon brukes til å beregne antall ganger hvert ord forekommer i et dokument. Vi har sett tidligere at vi enkelt kan få tilgang til alle ordene i korpuset ved bruk av `words()`. For å beregne frekvensdistribusjonen av ordene i en tekst skal vi bruke `Counter`. Et `Counter`-objekt holder styr på hvor mange ganger ulike verdier opptrer i f.eks. en liste.

`Counter` må importeres før bruk slik:

[12]:
```
from collections import Counter
```

Pass på at `Counter` må skrives med stor 'C'.

For å beregne frekvensdistribusjonen av ordene i talen til Kennedy kan du gjøre slik:

[13]:
```
fd_kennedy_ord = Counter(kennedy_ord)
```

I kodeblokken over oppretter vi et nytt `Counter`-objekt. `kennedy_ord` sendes inn til konstruktøren i `Counter`-objektet.

Nå kan vi skrive ut de 50 mest frekvente ordene i talen slik:

[14]:
```
mest_frekvente_ord = fd_kennedy_ord.most_common(50)

print(mest_frekvente_ord)
```

```
[(',', 85), ('the', 83), ('of', 65), ('.', 51), ('to', 38), ('and', 37), ('a',
29), ('we', 27), ('--', 25), ('in', 24), ('our', 21), ('that', 20), ('not', 19),
('for', 13), ('us', 12), ('this', 11), ('both', 10), ('are', 9), ('but', 8),
('world', 8), ('all', 8), ('Let', 8), ('it', 8), ('any', 8), ('do', 8), ('can',
8), ('let', 8), ('sides', 8), ('you', 7), ('is', 7), ('which', 7), ('from', 7),
('new', 7), ('by', 7), ('or', 7), ('those', 7), ('pledge', 7), ('be', 7), ('as',
6), ('who', 6), ('its', 6), ('will', 6), ('citizens', 5), ('I', 5), ('power',
5), ('shall', 5), ('To', 5), ('free', 5), ('them', 5), ('But', 5)]
```

Vi kan også enkelt finne ut hvor mange ganger visse ord forekommer i talen, for eksempel ordene *peace*, *America* og *america*. `Counter`-objektet er i bunn og grunn en ordbok med ekstra funksjonalitet. Vi kan derfor slå opp verdier slik som i en ordbok.

[15]:
```
print('Antall forekomster av "peace":', fd_kennedy_ord['peace'])
print('Antall forekomster av "America":', fd_kennedy_ord['America'])
print('Antall forekomster av "america":', fd_kennedy_ord['america'])
```

```
Antall forekomster av "peace": 4
Antall forekomster av "America": 2
Antall forekomster av "america": 0
```

Sannsynligheten for en setning kan beregnes ved bruk av relativ frekvens i et korpus (se forelesningsnotater). Her antar vi at forekomster av ord er uavhengige, noe som ikke stemmer i virkeligheten, men som vi likevel skal teste her. Da skal vi ta i bruk totalt antall ord (token) som vi har regnet ut tidligere, og lagre sannsynligheten for hvert ord i en ordbok der ordet er nøkkel og tilhørende sannsynlighet er verdi.

Metoden `items()` kan brukes på en ordbok, og dermed også på et `Counter`-objekt. Denne metoden returnerer en liste med tupler hvor første element i hvert tuppel er nøkkel og andre element er tilhørende verdi. Når vi skriver `for a, b in ordbok.items():` forstår Python automatisk at `a` skal referere til det første elementet i hvert tuppel (altså nøkkelen) og `b` til det andre elementet i hvert tuppel (altså den tilhørende verdien).

[16]:
```
# Oppretter først en tom ordbok
sannsynligheter = {}

# Itererer over ordtellingene
for word, antall in fd_kennedy_ord.items():
    sannsynligheter[word] = antall / antall_token
```

Vi kan nå skrive ut sannsynligheten til hvert ord slik:

'america' med liten 'a' forekommer ikke i teksten og er derfor ikke tilgjengelig i ordboken ***sannsynligheter****.*

[17]:
```
print('Sannsynlighet for "peace":', sannsynligheter['peace'])
print('Sannsynlighet for "America":', sannsynligheter['America'])
```

```
Sannsynlighet for "peace": 0.00258732212160414
Sannsynlighet for "America": 0.00129366106080207
```

Merk at vi ikke kan bruke metoden `most_common()` på `sannsynligheter`. Det er fordi `sannsynligheter` "bare" er en ordbok, ikke et `Counter`-objekt.

Sannsynlighetene for alle mulige utfall skal være (tilnærmet) lik 1. Dette kan vi kontrollere med metoden `sum()`, som tar imot en liste med tall som argument og returnerer summen av dem. Metoden `values()` brukes på en ordbok og returnerer en liste bestående av alle verdiene i ordboka.

[18]:
```
print(sum(sannsynligheter.values()))
```

```
1.0000000000000056
```

Nå har vi det vi trenger for å kunne generere en tekst basert på en slik sannsynlighetsberegning, men først må vi vite hvordan vi kan trekke ord tilfeldig basert på sannsynligheten til ordet.

1.3 Tilfeldige trekk

Vi skal her benytte oss av NumPy, et Python-bibliotek for ulike typer matematiske beregninger. Vi skal benytte oss av metoden `random.choice()` fra NumPy, som tar inn en liste med ord og deres sannsynligheter som argumenter og velger et angitt antall vilkårlige ord som følger sannsynlighetsdistribusjonen.

For å kunne bruke NumPy, må det først importeres:

[19]:
```
import numpy as np
```

Vi kunne også bare ha skrevet `import numpy`. Det er vanlig praksis å inkludere `as np` bak slik at vi kun trenger å referere til NumPy med `np` og ikke `numpy`. I teorien kan du også skrive `import numpy as n`, men det vil gjøre koden din mindre lesbar.

Dersom du får en feilmelding nå, kan det hende du ikke har installert NumPy på maskinen din. I så fall åpner du et terminalvindu og kjører `pip install numpy`, ev. `pip3 install numpy`.

For å illustrere hvordan vi utfører et tilfeldig trekk, lager vi først en enkel liste å trekke fra og en liste med tilhørende sannsynligheter:

[20]:
```
figurer = ['Ole Brumm', 'Kristoffer Robin', 'Nasse Nøff', 'Tigergutt', 'Tussi']

# Vi har én sannsynlighet for hver figur
figursannsynligheter = [0.3, 0.3, 0.2, 0.1, 0.1]  # sum = 1

tilfeldig_trekk = np.random.choice(figurer, p=figursannsynligheter)

print('Tilfeldig trekk:', tilfeldig_trekk)
```

```
Tilfeldig trekk: Nasse Nøff
```

Sannsynlighetene sendes inn i metoden som et nøkkelordsargument. Det vil si at vi setter nøkkelordet `p` (for *probabilities*) til å være lik listen med sannsynligheter. Nøkkelordsargumenter brukes hyppig på høyere nivå og kan være nyttige å forstå. Se ressurser på emnesiden / fra gruppetimer.

Vi kan også utføre flere uavhengige trekk:

[21]:
```
tre_tilfeldige_trekk = np.random.choice(figurer, 3, p=figursannsynligheter)
print('Tre trekk:', tre_tilfeldige_trekk)

fem_tilfeldige_trekk = np.random.choice(figurer, 5, p=figursannsynligheter)
print('Fem trekk:', fem_tilfeldige_trekk)
```

```
Tre trekk: ['Kristoffer Robin' 'Kristoffer Robin' 'Ole Brumm']
Fem trekk: ['Nasse Nøff' 'Ole Brumm' 'Ole Brumm' 'Tigergutt' 'Nasse Nøff']
```

La oss forsøke å generere en tilfeldig tekst basert på talen til Kennedy. Vi lar teksten bestå av 20 ord. Listene med ord og sannsynligheter må konverteres med `list()` for at NumPys `random.choice()` skal kunne tolke dem.

For oversikts skyld er argumentene skrevet på hver sin linje.

[22]:
```
tekst = np.random.choice(list(sannsynligheter.keys()),
                         20,
                         p=list(sannsynligheter.values()))

print(tekst)
```

```
['arms' 'to' 'mine' 'and' 'ventures' '.' 'of' 'a' 'aggression' ',' 'but'
 'our' 'Let' 'us' 'present' 'land' 'failure' 'a' 'of' 'the']
```

Som vi ser, er den genererte teksten ikke veldig koherent. Den produserte teksten følger bare frekvensdistribusjonen for enkeltord i korpuset og ser ikke på kontektsten (ordene rundt) i det hele tatt.

Nå som vi allerede har sannsynligheten for alle ordene, kan vi også beregne sannsynligheten for en tekst. Fordi ordene er generert basert på at forekomster av ord er uavhengige, trenger vi kun å multiplisere alle sannsynlighetene sammen. Her benytter vi oss av NumPys metode `prod()` som multipliserer sammen alle tall i en liste (i dette tilfellet listen som inneholder sannsynligheten for hvert av ordene i teksten).

[23]:
```
ordsannsynligheter = []
for token in tekst:
    ordsannsynligheter.append(sannsynligheter[word])

print('Sannsynligheten for den genererte setningen:',
      np.prod(ordsannsynligheter))
```

```
Sannsynligheten for den genererte setningen: 1.6436532709985067e-64
```

1.4 Språkmodeller

En språkmodell tilordner en sannsynlighet for en ordsekvens ved bruk av en sannsynlighetsdistribusjon. Språkmodeller har mange bruksområder i språkteknologi. For eksempel, i talegjenkjenning kan de bli brukt til å forutsi det neste uttalte ordet. I følgende steg, skal vi prøve å trene en trigramsmodell. Som nevnt tidligere, kan vi lese inn alle setningene i talen til Obama ved å bruke `sents()`.

[24]:
```
setninger = inaugural.sents("1961-Kennedy.txt")
```

For å kun se tredje setning skriver vi:

[25]:
```
tredje_setning = setninger[2]
print(tredje_setning)
```

```
['The', 'world', 'is', 'very', 'different', 'now', '.']
```

1.4.1 Bigram og trigram

For å hente ut bigrammer og trigrammer fra en tekst, bruker vi hhv. metodene `bigrams()` og `trigrams()` fra NLTK:

[26]:
```
from nltk import bigrams, trigrams

print('Bigram:', list(bigrams(tredje_setning)))
print('\nTrigram:', list(trigrams(tredje_setning)))
```

```
Bigram: [('The', 'world'), ('world', 'is'), ('is', 'very'), ('very',
'different'), ('different', 'now'), ('now', '.')]

Trigram: [('The', 'world', 'is'), ('world', 'is', 'very'), ('is', 'very',
'different'), ('very', 'different', 'now'), ('different', 'now', '.')]
```

I bigrammene og trigrammene over har vi ingen markører for setningsstart og setningsslutt. Dette legger vi til ved å sette nøkkelordsargumentene `pad_left` og `pad_right` til `True`.

[27]:
```
print('Bigram:', list(bigrams(tredje_setning, pad_left=True, pad_right=True)))
print('\nTrigram:', list(trigrams(tredje_setning, pad_left=True,␣
 ↪pad_right=True)))
```

```
Bigram: [(None, 'The'), ('The', 'world'), ('world', 'is'), ('is', 'very'),
('very', 'different'), ('different', 'now'), ('now', '.'), ('.', None)]

Trigram: [(None, None, 'The'), (None, 'The', 'world'), ('The', 'world', 'is'),
('world', 'is', 'very'), ('is', 'very', 'different'), ('very', 'different',
'now'), ('different', 'now', '.'), ('now', '.', None), ('.', None, None)]
```

1.4.2 `None`-typen

Markørene over representeres av `None`, altså uspesifiserte verdier. I en `if`-avgjørelse vil `None` evalueres som `False`. Dette er greit å vite til senere når vi skal generere tekst.

[28]:
```
a = None
if a:
    print('True!')
if not a:
    print('False!')
```

```
False!
```

Det samme skjer dersom vi konverterer den til en sannhetsverdi med metoden `bool()`.

[29]:
```
print(bool(a))
```

```
False
```

1.4.3 `defaultdict`

Den siste komponenten vi trenger er Pythons `defaultdict`. Dette er en slags forbedret ordbok som vil spare oss for et par kodelinjer senere. Vi ser på et eksempel med en gang.

Dersom vi forsøker å slå opp en ordbok på en ikke-eksisterende nøkkel får vi en såkalt `KeyError` som vil stanse kjøringen av koden vår:

[ ]:
```
middagsplan = {}

middagsplan['mandag'] = 'panert kylling'
middagsplan['tirsdag'] = 'ovnsbakt laks'

print(middagsplan['onsdag'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/var/folders/16/bswkk8t15td75835k2hc39hw0000gn/T/ipykernel_7982/2852053685.py in <module>
      3 middagsplan['tirsdag'] = 'ovnsbakt laks'
      4
----> 5 print(middagsplan['onsdag'])

KeyError: 'onsdag'
```

Med en `defaultdict` kan vi angi en standardverdi med et såkalt lambda-uttrykk når vi oppretter den. Du trenger ikke å bekymre deg nøyaktig for hva et lambda-uttrykk er nå, så lenge du forstår at vi bruker det til å definere standardverdien.

Dersom vi forsøker å slå opp en `defaultdict` på en ikke-eksisterende nøkkel, vil nøkkelen legges til og standardverdien assosieres med nøkkelen som tilhørende verdi.

Før vi bruker `defaultdict`, må den importeres:

[2]:
```
from collections import defaultdict
```

```
                              # Her er standardverdien!
middagsplan = defaultdict(lambda: 'ikke planlagt')

middagsplan['mandag'] = 'panert kylling'
middagsplan['tirsdag'] = 'ovnsbakt laks'

print(middagsplan['onsdag'])
```

```
ikke planlagt
```

Vi ser at `'onsdag'` er lagt til som nøkkel med `'ikke planlagt'` som tilhørende verdi. Denne verdien kan naturligvis endres senere.

[31]:
```
print(list(middagsplan.items()))
```

```
[('mandag', 'panert kylling'), ('tirsdag', 'ovnsbakt laks'), ('onsdag', 'ikke
planlagt')]
```

På denne måten unngår vi å måtte sjekke om en nøkkel eksisterer før vi slår opp på den.

Slik som med vanlige ordbøker, kan vi bruke metodene `keys()` som henter ut alle nøklene, `values()` som henter ut alle verdiene, og `items()` som henter ut nøkler og verdier parvis. Disse kan konverteres til vanlige lister med metoden `list()`:

[4]:
```
antall_møbler = defaultdict(lambda: 0)
antall_møbler['sofa'] = 3
antall_møbler['stol'] = 6
antall_møbler['bord'] = 2
antall_møbler['kommode'] = 1

# Liste med alle nøkler
print('Nøkler:', list(antall_møbler.keys()))

# Liste med alle verdier
print('Verdier:', list(antall_møbler.values()))

# Liste med alle nøkler og verdier
print('Alt:', list(antall_møbler.items()))
```

```
Nøkler: ['sofa', 'stol', 'bord', 'kommode']
Verdier: [3, 6, 2, 1]
Alt: [('sofa', 3), ('stol', 6), ('bord', 2), ('kommode', 1)]
```

1.4.4 Trening av modellen

Nå skal vi begynne å trene modellen vår. La oss begynne med trigramsmodellen. Det første som må gjøres er å initialisere to `defaultdict`-er:

`trigramtellinger` og `trigrammodell`

Legg merke til at standardverdien i disse er en ny `defaultdict`. I `trigramtellinger` har denne standardverdi `0` fordi vi ønsker at observasjoner vi ikke har gjort før skal starte med verdien 0, mens i `trigrammodell` har denne standardverdi `0.0` fordi vi trenger desimaltall for å kunne representere sannsynlighet.

[32]:
```
# Først importerer vi det vi trenger
from collections import defaultdict
from nltk import trigrams
import numpy as np

# Leser inn setninger
kennedy_setninger = inaugural.sents('1961-Kennedy.txt')

# Oppretter defaultdict-er
trigramtellinger = defaultdict(lambda: defaultdict(lambda: 0))
trigrammodell = defaultdict(lambda: defaultdict(lambda: 0.0))
```

Vi teller antall ganger et ord kommer etter to foregående. Det er derfor vi bruker en nøstet ordbok. Vi slår først `trigramtellinger` opp på tuppelet av de to foregående ordene. Dette returnerer den "indre" ordboken som sier hvor mange ganger ulike ord kommer etter de to foregående.

[33]:
```
for setning in kennedy_setninger:
    for ord1, ord2, ord3 in trigrams(setning, pad_right=True, pad_left=True):
        trigramtellinger[ord1, ord2][ord3] += 1
```

Vi kan nå inspisere tellingene våre før vi går videre:

[34]:
```
# Antall ganger "citizens" står etter "My fellow"
print(trigramtellinger["My", "fellow"]["citizens"]) # 1

# Antall ganger "Americans" står etter "My fellow"
# Her brukes standardverdien 0. Med en vanlig
# ordbok ville vi ha fått en feil her.
print(trigramtellinger["My", "fellow"]["Americans"]) # 0

# Antall setninger som starter på "The"
print(trigramtellinger[None, None]["The"]) # 3

# Antall ganger en setning starter på et gitt ord
# Her ser vi på den "indre" ordboken
print(list(trigramtellinger[None, None].items()))
```

```
1
0
```

```
3
[('Vice', 1), ('For', 3), ('The', 3), ('And', 4), ('We', 3), ('Let', 8),
('This', 1), ('To', 5), ('United', 1), ('Divided', 1), ('But', 5), ('If', 1),
('Finally', 2), ('So', 1), ('Together', 1), ('All', 1), ('Nor', 1), ('In', 2),
('Since', 1), ('Now', 1), ('Can', 1), ('Will', 1), ('I', 2), ('My', 1), ('With',
1)]
```

Vi går videre og lager selve modellen. Modellen er ordboken `trigrammodell` som vi nå skal fylle med sannsynlighetene for ord gitt de to foregående.

[35]:
```
for foregående_ord in trigramtellinger:
    tellinger_totalt = sum(trigramtellinger[foregående_ord].values())
    for ord3 in trigramtellinger[foregående_ord]:
        trigrammodell[foregående_ord][ord3] =␣
↪trigramtellinger[foregående_ord][ord3] / tellinger_totalt
```

La oss nå sjekke sannsynlighetene for de samme ordene som tidligere:

[36]:
```
# Sannsynlighet for "citizens" etter "My fellow"
print(trigrammodell["My", "fellow"]["citizens"]) # 1.0

# Sannsynlighet for "Americans" etter "My fellow"
# Her brukes standardverdien 0.0. Med en vanlig
# ordbok ville vi ha fått en feil her.
print(trigrammodell["My", "fellow"]["Americans"]) # 0.0

# Sannsynlighet for setning på "The"
print(trigrammodell[None, None]["The"]) # 0.05769...

# Sannsynlighet for at en setning starter på et gitt ord
# Her ser vi på den "indre" ordboken
print(list(trigrammodell[None, None].items()))
```

```
1.0
0.0
0.057692307692307696
[('Vice', 0.019230769230769232), ('For', 0.057692307692307696), ('The',
0.057692307692307696), ('And', 0.07692307692307693), ('We',
0.057692307692307696), ('Let', 0.15384615384615385), ('This',
0.019230769230769232), ('To', 0.09615384615384616), ('United',
0.019230769230769232), ('Divided', 0.019230769230769232), ('But',
0.09615384615384616), ('If', 0.019230769230769232), ('Finally',
0.038461538461538464), ('So', 0.019230769230769232), ('Together',
0.019230769230769232), ('All', 0.019230769230769232), ('Nor',
0.019230769230769232), ('In', 0.038461538461538464), ('Since',
0.019230769230769232), ('Now', 0.019230769230769232), ('Can',
0.019230769230769232), ('Will', 0.019230769230769232), ('I',
0.038461538461538464), ('My', 0.019230769230769232), ('With',
0.019230769230769232)]
```

1.4.5 Generering av tekst

Nå skal vi endelig bruke modellen til å generere tekst. Vi tar i bruk en variabel `ferdig` med sannhetsverdi for å sjekke om den produserte teksten har nådd ønsket lengde. Vi begynner med å lage et tuppel (i dette tilfellet et par) av de to siste elementene i listen (altså variabelen `tekst`) med `tuple(tekst[-2:])`. (*Slice*-notasjon returnerer en liste som vi konverterer til et tuppel med `tuple()` fordi vi har lagret de foregående ordene i modellen som tupler, ikke lister.) Dette er bigrammet som danner konteksten for ordet som skal genereres. Deretter slår vi opp i trigrammodellen var for å finne alle mulige tredje ord i henhold til modellen og deres sannsynligheter.

NumPys `random.choice()` trekker deretter neste ord basert på denne sannsynlighetsdistribusjonen:

[38]:
```
# Vi starter teksten på to stk. setningsstart
tekst = [None, None]

ferdig = False
while not ferdig:
    # Henter ut de to foregående ord
    nøkkel = tuple(tekst[-2:])

    # Finner mulige neste ord og sannsynligheter
    mulige_ord = list(trigrammodell[nøkkel].keys())
    tilhørende_sannsynligheter = list(trigrammodell[nøkkel].values())

    # Trekker et av ordene
    tekst.append(np.random.choice(mulige_ord, p=tilhørende_sannsynligheter))

    # Sjekker om teksten slutter på to stk. setningsslutt
    if tekst[-2:] == [None, None]:
        ferdig = True

print(tekst)
```

```
[None, None, 'In', 'the', 'long', 'history', 'of', 'the', 'deadly', 'atom', ',',
'yet', 'both', 'racing', 'to', 'alter', 'that', 'uncertain', 'balance', 'of',
'terror', 'that', 'stays', 'the', 'hand', 'of', 'mankind', "'", 's', 'work',
'must', 'truly', 'be', 'our', 'own', '.', None, None]
```

Gratulerer, du har generert tekst!

Til slutt bruker vi `join()` (se tidligere gruppetimer / oblig 1a) til å konvertere teksten til en streng. Legg merke til at vi hvordan vi bruker *list comprehension* til å luke ut setningsmarkørene `None`. Disse evalueres, som du husker, som `False`.

[ ]:
```
tekst_as_str = ' '.join([word for word in text if word])

# Linjen over tilsvarer:
tekst_as_str = ''
for word in tekst:
    if word: # Tilsvarer 'if word != None'
        tekst_as_str += word

print(tekst_as_str)
```

1.4.6 Spørsmål

1. Hvorfor initialiseres `tekst` med `[None, None]`? Hvorfor to stk. `None`?

2. Hvordan kan vi beregne sannsynligheten for teksten vi nettopp genererte? Ta utgangspunkt i koden i delen "**Frekvensdistribusjon**".

2 Teori om språkmodeller

Formel:

$$P(w_1 ... w_n) = \prod_{i=1}^{k} P(w_i | w_{i-1})$$

Korpus:

`<s> Petra spiller piano <\s>`

`<s> Katherine spiller ikke piano <\s>`

`<s> Ludovico spiller piano <\s>`

Over finner du en formel for en språkmodell (en såkalt bigrammodell) og et lite tekstkorpus. Bruk bigrammodellen og tekstkorpuset til å beregne sannsynligheten for setningen `<s> Katherine spiller piano <\s>`. Vis hvilke sannsynligheter du trenger og hvordan disse beregnes fra korpuset.

3 Flertydighet

I denne oppgaven skal vi se litt nærmere på flertydigheten i språket. Vi skal bruke Python til å analysere Brown-korpuset, og vi skal fokusere på antall tildelte ordklasser til hvert ord for å illustrere flertydigheten.

En oversikt over hva de ulike tag-ene i Brown-korpuset står for finner du et stykke nede på siden her.

Merk at oppgavene her skal løses uten bruk av NLTKs funksjonalitet for frekvens (`nltk.FreqDist` og `nltk.ConditionalFreqDist`).

1. Hva mener vi når vi sier at det finnes flertydighet i språket? Gi et eksempel for norsk, og et for engelsk.
2. Her skal vi bruke ordklassetaggede data fra nyhetsdelen i Brown-korpuset. Opprett en variabel `brown_news` som er en liste av tupler `(ord, tag)`. Du kan importere Brown-korpuset med `from nltk.corpus import brown`. Deretter kan ordene leses inn med `brown.tagged_words(categories='news')`. (Du kan lese mer om denne metoden og andre korpusmetoder i NLTK-dokumentasjonen her. Vi har allerede gått gjennom noen av dem i denne oppgaven.)
3. Hvor mange typer er flertydige i nyhetsdelen i Brown-korpuset? Det vil si, hvor mange ordtyper forekommer med mer enn én ordklasse-tag? Her kan du benytte deg av en ordbok eller `defaultdict` (denne kan du f.eks. kalle `tags`) som for hvert ord inneholder alle mulige ordklasser den forekommer med, representert som en liste. Pass på at du ikke tar vare duplikater av tag-er.
4. Finn de fem mest flertydige ordene (de med flest antall tag-er). Her kan du først konvertere ordboken `tags` til en liste med tupler `(ord, [tag1, tag2, ...])`. Deretter kan du sortere denne listen med `sort()`. For å sortere basert på lengden til det andre elementet i hvert tuppel kan du sette nøkkelordsargumentet `key` i `sort()` til å være lik `lambda x: len(x[1])`, og `reverse` til `True` for å sortere synkende.
5. Hvor mange unike tag-er forekommer i nyhetsdelen i Brown-korpuset?
6. Skriv en funksjon `freqs()` som tar inn et ord som argument og skriver ut en frekvensliste som sier hvor ofte ordet forekommer med hver av taggene. For eksempel forekommer ordet *run* 20 ganger med `NN`, 11 ganger med `VB`, og 4 ganger med `VBN`.
7. Ved hjelp av funksjonen fra punkt 6, finn frekvenslisten for de aller mest flertydige ordene i nyhetsdelen i Brown-korpuset. Hva observerer du?