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?