IN1140 H2021 – Gruppeoppgaver
Språkmodeller og Ordklasser

1 NLTK og språkmodeller

Målet med denne oppgaven er å implementere to språkmodeller (en unigrammodell og trigrammodell) ved hjelp av Python's Natural Language Tool Kit (NLTK).
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/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 oppgaven i IN1140 (oblig 2a). Vi anbefaler at du også repeterer pensum fra forelesningen om språkmodeller før du jobber deg gjennom følgende oppgaver.

1.1 NLTK

Natural Language Tool Kit (NLTK) er en Python modul som brukes til å skrive programmer for språkprossessering. NLTK gir tilgang til både korpora og innebygde metoder. For mer informasjon om NLTK, besøk: `http://www.nltk.org/`. NLTK er installert på IFIs servere. Dersom du ønsker å installere det på egen maskin er det instruksjoner både på emnesiden og på NLTK-siden under "Installing NLTK".
For å kunne bruke NLTK i koden din, kan den importeres slik:

```
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 (fra 1789 til 2009). Hvert korpus inneholder et visst antall tekster. Du kan lese mer om dette her `https://www.nltk.org/book/ch02.html`.
Her skal vi jobbe med *Inaugural*-korpuset. For å kunne laste ned dette korpuset, skriver du følgende:

```
import nltk
from nltk.corpus import inaugural
```

For å liste alle dokumentene som finnes i korpuset kan du kjøre:

```
print(inaugural.fileids())
```

Vi skal nå fokusere på Obama sin innledende tale fra 2009. Vi kan enkelt få tilgang til både teksten (ved bruk av `raw`), ordene (ved bruk av `words`) og setningene (ved bruk av `sents`) av talen:

```
inaugural.raw("2009-Obama.txt")
inaugural.words("2009-Obama.txt")
inaugural.sents("2009-Obama.txt")
```

Spørsmål 1: Skriv ut ordene og setningene i Obamas tale. Hva slags datastruktur (f.eks. streng, liste, ordbok) brukes for å representere ordene i talen? Hva med setningene?

Vi lagrer ordene i talen i en variabel (`obama_words`). Vi kan deretter finne totalt antall ord (tokens) i talen slik:

```
obama_words = inaugural.words("2009-Obama.txt")
total_words = len(obama_words)
```

For å finne totalt antall distinkte ord (typer) gjør vi om ordene til små bokstaver og lagrer dem i en liste (`distinct_words`) som vi deretter sjekker lengden på:

```
distinct_words = []
for w in obama_words:
    distinct_words.append(w.lower())

total_distinct_words = len(set(distinct_words))
```

Spørsmål 2: Hvor mange tokens og typer inneholder Obamas tale? Hvorfor har vi gjort om listen til en mengde (`set`)? Hva skjer dersom vi ikke gjør det?

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 hvert ord skal vi bruke `Counter`. En `Counter` er en modul som holder styr på hvor mange ganger ulike verdier legges til. `Counter` må importeres før bruk slik:

```
from collections import Counter
```

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

```
fd_obama_words = Counter(obama_words)
```

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

```
print("Most common words: ",fd_obama_words.most_common(50))
```

Vi kan også enkelt finne ut hvor mange ganger visse ord forekommer i talen, for eksempel ordene “peace”, “America” og “america”:

```
print("Frequency of 'peace': ",fd_obama_words["peace"])
print("Frequency of 'America': ",fd_obama_words["America"])
print("Frequency of 'america': ",fd_obama_words["america"])
```

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 (tokens) som vi har regnet ut tidligere og lagre sannsynligheten for hvert ord i en ordbok (`probabilities`) der ordet er nøkkel og sannsynligheten er

verdi:

```
probabilities = {}
for word, count in fd_obama_words.items():
    probabilities[word] = count/total_words
```

Spørsmål 3: Ordboken `probabilities` inneholder sannsynligheten for ulike ord i teksten. Hva er nøkkel og hva er verdi i ordboken?

Sannsynlighetene for alle mulige utfall summerer til (tilnærmet) 1, det kan vi sjekke ved å gjøre:

```
print(sum(probabilities.values()))
```

Vi kan nå generere en tekst basert på en slik sannsynlighetsberegning. Vi skal her benytte oss av NumPy, et Python-bibliotek for ulike typer matematiske beregninger. Vi skal benytte oss av `random.choice` fra NumPy, som tar inn en liste med ord og deres sannsynligheter og velger et angitt antall vilkårlige ord som følger sannsynlighetsdistribusjonen. For eksempel:

```
>>> import numpy as np
>>> aa_milne_arr = ['pooh', 'rabbit', 'piglet', 'Christopher']
>>> np.random.choice(aa_milne_arr, 5, p=[0.5, 0.1, 0.1, 0.3])
array(['pooh', 'pooh', 'pooh', 'Christopher', 'piglet'])
```

Vi sender ordene og sannsynlighetene vi beregnet for Obama-talen til `random.choice` som genererer (og skriver ut) en tekst på 20 ord slik:

```
text = np.random.choice(list(probabilities.keys()), 20,
                        p=list(probabilities.values()))
print(text)
```

Som du kan se, er den genererte teksten ikke veldig koherent. Den produserte teksten følger bare frekvensdistribusjonen for enkeltord i korpuset og ser ikke på kontektsten 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 `prod`-funksjon som multipliserer sammen alle tall i en liste (i dette tilfellet listen som inneholder sannsynligheten for hvert av ordene i teksten):

```
word_probabilities = []
for w in text:
    word_probabilities.append(probabilities[w])
print(np.prod(word_probabilities))
```

1.3 Språkmodeller

En språkmodell tilordner en sannsynlighet for en ordsekvens ved bruk av en sannsynlighetsdistribusjon. Språkmodeller har mange applikasjoner 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, så kan vi enkelt lese alle setningene i talen til Obama ved å bruke `sents`:

```
obama_sents = inaugural.sents("2009-Obama.txt")
```

For å lese kun første setning kan vi skrive:

```
first_sentence = obama_sents[0]
print(first_sentence)
```

Det er enkelt å hente ut bigrammer og trigrammer i en tekst ved bruk av NLTK. Det eneste du trenger å gjøre er følgende:

```
from nltk import bigrams, trigrams
print(list(bigrams(first_sentence)))
print(list(trigrams(first_sentence)))
```

Som du kan se, er ikke setningsbegynnelse og setningsslutt markert. For å inkludere denne informasjonen i bigrammmene og trigrammene dine kan du angi følgende:

```
print(list(bigrams(first_sentence, pad_left=True, pad_right=True)))
print(list(trigrams(first_sentence, pad_left=True, pad_right=True)))
```

Spørsmål 4: Inspisér resultatet av forrige kall. Hvordan skiller disse bi- og trigrammene seg fra de forrige?

Vi skal bruke disse trigrammene for å trene en språkmodell. For å gjøre det, skal vi bruke en ordbok som skal initialiseres med en *defaultdict*. *defaultdict* fungerer akkurat som en vanlig dictionary ("ordbok"), men den initialiseres med en funksjon ("*default factory*") som ikke tar noen argumenter og returnerer standardverdien (default) for en ikke-eksisterende nøkkel. En *defaultdict* vil aldri heve en *KeyError*. En hvilken som helst nøkkel som ikke eksisterer får verdien returnert av "*default factory*". For å kunne bruke *defaultdict* må vi importere den. Ta for deg følgende eksempel for å forstå hva en *defaultdict* er:

```
>>> from collections import defaultdict
>>> muffins = defaultdict(lambda: "Vanilje")
>>> muffins["Ida"] = "Blåbær"
>>> muffins["Stian"] = "Pecan nøtter"
>>> muffins["Ida"]
Blåbær
>>> muffins["Pedro"]
Vanilje
```

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

```
from collections import defaultdict
trigram_counts = defaultdict(lambda: defaultdict(lambda: 0))
trigram_model = defaultdict(lambda: defaultdict(lambda: 0.0))
```

Vi skal nå fylle ordboken `trigram_counts` med trigrammene i Obama-talen og antall forekomster av trigrammene:

```
for sentence in obama_sents:
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        trigram_counts[(w1, w2)][w3] += 1
```

Spørsmål 5: Ordboken `trigram_counts` inneholder tellinger for ulike trigrammer i teksten. Hva slags datastruktur er dette? Hva er nøkkel og hva er verdi i ordboken?

Nå er trigrammene våre i en ordbok, og vi får enkel tilgang til elementene:

```
print(trigram_counts["My", "fellow"]["citizens"])
print(trigram_counts["My", "fellow"]["nonexistingword"])
print(trigram_counts[None, None]["The"])
```

Vi skal nå beregne sannsynligheter for trigrammene våre og samle disse i ordboken `trigram_model`. Som vi vet trenger vi frekvensen for et trigram (`total_trigramcount`) samt frekvensen for de to første ordene i trigrammet (`trigram_counts[w1_w2].values()`) for å beregne sannsynligheten for et trigram:

```
for w1_w2 in trigram_counts:
    total_trigramcount = sum(trigram_counts[w1_w2].values())
    for w3 in trigram_counts[w1_w2]:
        trigram_model[w1_w2][w3] = trigram_counts[w1_w2][w3]/total_trigramcount
```

La oss nå sjekke verdiene av de forrige testene:

```
print(trigram_model["My", "fellow"]["citizens"])
print(trigram_model["My", "fellow"]["nonexistingword"])
print(trigram_model[None, None]["The"])
```

Nå skal vi bruke modellen til å generere tekst. Vi tar i bruk en variabel (`sentence_is_finished`) med boolsk verdi for å sjekke om den produserte teksten har nådd ønsket lengde. Vi begynner med å lage en tuppel (i dette tilfellet et par) av de to siste elementene i listen (altså variabelen `text`) ved å si `tuple(text[-2:])`. Dette er bigrammet som danner konteksten for ordet som skal genereres. Deretter slår vi opp i tri-grammodellen vår for å finne alle mulige tredje ord i henhold til modellen og deres sannsynligheter.

`random.choice` genererer deretter neste ord basert på denne sannsynlighetsdistribusjonen:

```
text = [None, None]
sentence_is_finished = False
while not sentence_is_finished:
    key = tuple(text[-2:])
    words = list(trigram_model[key].keys())
    probs = list(trigram_model[key].values())
    text.append(np.random.choice(words, p=probs))

    if text[-2:] == [None, None]:
        sentence_is_finished = True

print(' '.join([t for t in text if t]))
```

Spørsmål 7: Hvorfor initialiseres `text`-listen med `[None, None]`? Og hvorfor brukes samme bigram som test på at setningen er ferdig?

Gratulerer! Da har du laget din første trigramsmodell som produserer tekst. Som du ser gir trigrammodellen en mye bedre og mer sammenhengende tekst enn unigrammodellen i den tidligere oppgaven.

Spørsmål 8: Hvordan kan vi beregne sannsynligheten til teksten vi nettopp genererte? Ta utgangspunkt i koden for å beregne sannsynligheten til teksten vi genererte i 1.2.

Spørsmål 9: Hva ville du ha endret i koden for å trene en bigramsmodell? Følg stegene over og tren din egen bigramsmodell.

2 Teoretisk: Språkmodeller

Formel:

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

Tekskorpus:

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

Table 1: Formel og tekstkorpus.

I Tabell 1 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 for å analysere Brown-korpuset, og vi skal fokusere på antall tildelte ordklasser til hvert ord som tegn på at et ord er flertydig.

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 en en liste av par (ord, tagg).

3. Hvor mange ord er flertydige i Brown-korpuset? Det vil si, hvor mange ordtyper forekommer med mer enn én ordklassetagg?
   Hint: Her skal du benytte deg av en ordbok (*dictionary*) `tags` som for hvert ord i Brown-korpuset inneholder alle mulige ordklasser den forekommer med (disse skal være representert som en liste).

4. Hvilket ord har flest tagger, og hvor mange distinkte tagger finner du? Her kan du bruke `tags` som du har laget i forrige spørsmål, sammen med `sorted()` metoden.
   Hint: Når du skal bruke `sorted()` metoden her, sørg for at du sjekker antall elementer i listen av verdier (med tanke på *values* i din *dictionary* `tags`), heller enn verdiene i seg selv.

5. Skriv en funksjon `freqs(w)` som tar et ord som argument og skriver ut hvor ofte ordet forekommer med hver av taggene. For eksempel forekommer *run* 20 ganger med NN, 11 ganger med VB, og 4 ganger med VBN.

6. Ved hjelp av funksjonen fra punkt 5, finn frekvenslisten for det mest flertydige ordet i Brown. Hva observerer du?

I deloppgavene 3 og 4 er det viktig at du passer på at for hvert ord telles hver distinkte tagg nøyaktig én gang; det vil si at for et ord med to forekomster med NP og tre med NN har vi listen `["NP", "NN"]` og ikke `["NP", "NN", "NN", "NN", "NP"]`.

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