INF1820 V2017 – Oblig 2b
Språkmodeller og ordklassetagging

Innleveringsfrist, fredag 17 mars

Lever inn svarene dine i Devilry (`https://devilry.ifi.uio.no/`) i filer som angir brukernavnet ditt, slik: `oblig2b_brukernavn.py`. Pass på at fila di kan kjøres som et program; det skal ikke være en REPL-sesjon limt inn i en fil.

En perfekt besvarelse på denne oppgaven er verdt 100 poeng.

1 Evaluering av språkmodeller (50 poeng)

I denne oppgaven skal vi evaluere n-gram språkmodeller og se litt på viktigheten av riktig metode når vi evaluerer. Mer spesifikt skal du implementere et kvalitetsmål for språkmodeller som heter *perpleksitet*, og se på hvordan kvalitetsmålet oppfører seg når vi evaluerer en bigrammodell på data som modellen er trent på i motsetning til data som modellen ikke har sett før.

Hvis vi sier at en språkmodell m tar en setning s og beregner en sannsynlighet $m(s)$ for setningen, er perpleksiteten til modellen $PP_m(S)$ for et korpus S av setninger, $S = s_1, s_2, s_3, \ldots$ definert som:

$$PP_m(S) = 2^{-\frac{1}{N}\sum_{s \in S} \log_2 m(s)} \tag{1}$$

der N er det totale antallet ord i S.

For å implementere perpleksitetsberegningen trenger vi først kode til å beregne sannsynlighetene $P(w|w')$ som er selve modellen. Dette har jeg allerede implementert i fila `lm.py` som kan lastes ned fra obligsiden på emnesidene. Denne inneholder en klasse som har skjelettet til en språkmodell. For å lage en bigrammodell, trent på hele Brown-korpuset unntatt den første seksjonen (adventure, som vi skal bruke i evalueringen), sier du `m = LM()`. For å finne sannsynligheten til et ord `w` gitt et forrige ord `prev` kan du kalle metoden `p`: `prob = m.p(w, prev)` som vil beregne sannsynligheten. Implementasjonen er litt mer komplisert enn bare et oppslag i bigramtabellen for å håndtere ukjente ord riktig.

1.1 Del 1: `logprob`

Det første vi skal implementere er en metode `logprob(self, s)` som tar en setning og beregner logaritmen[1] til sannsynligheten $m(s)$ modellen gir setningen s. Vi minner om at en bigrammodell angir sannsynligheten $P(w_1 \ldots w_n)$ ved uttrykket:

$$P(w_1 \ldots w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1}) \tag{2}$$

Vi får da:

$$\log P(w_1 \ldots w_n) = \sum_{i=1}^{n} \log P(w_i|w_{i-1}) \tag{3}$$

[1] Se seksjon A for en kort introduksjon til logaritmer.

Metoden `logprob` må dermed hente ut bigrammer fra inputsetningen ved hjelp av funksjonen `bigrams` fra NLTK (se startkoden for hvordan denne funksjonen kan brukes) og summere opp logaritmen av de enkelte sannsynlighetene for hvert bigram, og returnere summen. Logaritmen beregnes med `log(x, 2)` som hentes ved hjelp av: `from math import log`

1.2 Del 2: `perplexity`

Med en metode som beregner log-sannsynligheten til en enkelt setning, kan vi nå implementere selve beregningen av perpleksiteten. Du skal her implementere metoden `perplexity(self, sents)` som tar en liste setninger, og kan kalles slik for å returnere perpleksiteten:
`m.perplexity(nltk.corpus.brown.sents(categories="news"))`

Dette gjøres lettest ved å iterere over setningene i `sents` og samle opp antall ord vi har observert (som vil være N i ligning 1) og sammenlagt log-sannsynlighet for setningene ($\sum_{s \in S} \log_2 m(s)$). Hvis vi sier at summen av log-sannsynlighetene er kalt l, er så perpleksiteten $2^{\frac{-l}{N}}$. Eksponensieringen kan utføres i Python ved hjelp av funksjonen `pow`: `pow(2, x)` beregner 2^x. Funksjonen importeres på samme måte som `log`.

1.3 Evalueringen

Vi er nå i stand til å evaluere modellen vår. Kjør perpleksitetsmetoden på både "adventure"-delen av Brown-korpuset og på "news"-delen. Hva er perpleksiteten for hver av de to delene? Videre, hva er årsaken til at de er såpass forskjellige, og hvilken betydning har dette for hvordan vi må gå fram når vi skal teste en modell?

2 Zipfianske distribusjoner (20 poeng)

Vi har i forelesningene diskutert hvordan mange fenomener er distribuert etter den såkalte Zipf-distribusjonen. Det vil si at frekvensen f til et element er lik en konstant k delt på rangen r, der k er antall ganger det mest frekvente elementet er observert:

$$f = \frac{k}{r} \tag{4}$$

Vi skal i denne oppgaven sammenligne de observerte forekomstene av ord og tagger i nyhetsdelen av Brown-korpuset med frekvensene som forutses av uttrykket over.

Skriv en funksjon `zipfity(lst)` som tar som argument en liste av elementer. Funksjonen skal så telle hvor mange ganger de distinkte elementene forekommer, enten ved hjelp av kode som ligner på den du skrev i forrige del av obligen eller ved hjelp av `nltk.probability.FreqDist`[2] hvis du foretrekker det, og så skrive ut for de ti mest frekvente elementene:

- Selve elementet
- Den observerte frekvensen i korpuset
- Den teoretiske frekvensen etter Zipf-formelen

[2] Se `http://www.nltk.org/api/nltk.html\#nltk.probability.FreqDist` for en beskrivelse av hvilke muligheter denne klassen gir deg.

3 Ordklassetagging med regulære uttrykk (30 poeng)

I denne oppgaven skal du lage en ordklassetagger med regulære uttrykk.

Taggeren med regulære uttrykk i NLTK-boka (del 5.4.2, under overskriften *The Regular Expression Tagger*) sjekker kun noen få uttrykk, så her er det masse rom for forbedring: for eksempel kan vi tagge alle ord på *-able* som adjektiv. Definer en tagger ved hjelp av `nltk.RegexpTagger` der du har minst 10 uttrykk i tillegg til de som er nevnt i boka. Dokumenter alle reglene dine, og gi minst ett eksempel på ord som dekkes.

Husk å håndtere liten og stor bokstav, og at enkeltord også kan brukes i de regulære uttrykkene, slik at f.eks. *the* alltid vil tagges som bestemmer.

Bruk `adventure`-kategorien i Brown mens du utvikler taggeren din ved å teste nøyaktigheten til taggeren (med `evaluate`-metoden). Når du er fornøyd med reglene dine, test taggeren på kategorien `fiction` i Brown, og rapporter resultatene. Det er viktig at du ikke endrer reglene dine etter dette.

Merk: Poengene du får på denne oppgaven er ikke basert på nøyaktigheten til taggeren i den endelige evalueringen på `fiction`-kategorien! Det er også helt vanlig at nøyaktigheten til en tagger synker når man tester den på et annet korpus enn den ble utviklet.

Til slutt skal programmet ditt lese inn filen `test_setninger.txt` som er lagt ut sammen med denne obligen, tagge den, og skrive ut resultatet. Kopier outputen inn i filen din og diskuter minst 3 av feilene taggeren gjør, og kom med forslag til hvordan den kan forbedres.

A Logaritmer

Kort fortalt er logaritmer en metode som lar oss omforme multiplikasjonsoperasjoner til addisjonsoperasjoner. Denne egenskapen får vi fra:

$$\log(a * b) = \log a + \log b \tag{5}$$

Altså: logaritmen til et produkt er lik summen av logaritmene til faktorene i produktet. Vi får dermed også:

$$\log \prod f(x) = \sum \log f(x) \tag{6}$$

Merk at logaritmen kun er definert for tall større enn null.[3]

Logaritmen har en annen nyttig egenskap når vi driver med sannsynligheter. Det viser seg at når vi ganger sammen forholdsvis mange tall nærme null (som sannsynlighetene i modellene våre tenderer til å være) får vi et problem med at resultatet fort blir så lite at det ikke lett kan representeres i datamaskinen. Vi bruker da logaritmer fordi de omdanner tall mellom 0 og 1 til tall mellom $-\infty$ og 0. Dette gir oss tall med større absolutt verdi, som sammen med at vi legger dem sammen i stedet for å gange dem sammen gjør at resultatene kan representeres i maskinen.

[3] Ja, jeg vet at dette ikke er helt sant, men hvis du vet det trenger du heller ikke lese dette tilleggsavsnittet.