INF 2820 V2016: Obligatorisk innleverinsoppgave 2

- **Besvarelsene skal leveres i devilry innen torsdag 17.3 kl 18.00**
- **Filene det vises til finner du på**
  - **/projects/nlp/inf2820/scarrie**
  - **/projects/nlp/inf2820/cfg**

Innledning

Det er to deler – en del om ord og en del om setninger og kontekstfrie grammatikker.

Del 1 Ord

Vi har snakket en del på forelesningene om ord og leksemer og sammenhengene mellom dem. De følgende oppgavene vil se mer på dette.

Oppgave 1: Stemming (10 poeng)

Jeg lurte på hvordan jeg deler en fil i mindre deler. Jeg gikk til Google. Skulle jeg søke på "split" og "file" eller "splitting" og "file"? Måtte jeg søke på begge to for å finne alle interessante resultat? Heldigvis ikke. Jeg får omtrent de samme resultatene uansett hvilket av termparene jeg bruker. Antagelig bruker Google en teknikk som kalles "stemming" for å få til dette. Dette er en velkjent teknikk i informasjonsgjenfinning ("information retrieval"). En prøver å fjerne endelser fra ord og dermed få dem ned til en felles stamme ("stem"), for dette eksempelet "split", som så blir brukt for søk.

a) Se på Seksjon 3.6 Normalizing Text i NLTK-boka. Last inn LancasterStemmer og PorterStemmer. Hvordan vil de to takle eksempelet med "split/splitting"?

Innlevering: Svar på spørsmålet og utskrift som viser resultater fra kjøring.

b) Porter-stemmer er langt på vei blitt en standard stemmer som mange tyr til. For referanseformål er den frosset og blir ikke utviklet videre. Men det er mulig med forbedringer og dette prøver bl.a. Snowball-stemmeren på. Den kan lastes i NLTK ved

```
>>> snowball = nltk.SnowballStemmer('english')
```

Bruk for eksempel teksten 'grail.txt' som kan lastes som en liste av ord med kommandoen

```
>>> grail = nltk.corpus.webtext.words('grail.txt')
```

Finn deretter 6 forskjellige eksempler hvor Snowball-stemmeren og Porter-stemmeren gir forskjellig resultat. Vi regner ikke med forskjeller i stor og liten bokstav eller den lille u-en først, som vi skal komme tilbake til. (Har du problemer med u-en, kan det være en ide å gjøre neste oppgave først.)

Innlevering: 6 forskjellige ord som blir stemmet forskjellig av de to og resultatet de to stemmerne gir for disse ordene.

Hensikten med denne oppgaven var

- Å forstå begrepet stemming og hva det kan brukes til
- Å se at det finnes programmer som kan hjelpe oss til å løse denne typen oppgaver, og at det ofte er bedre å bruke et slikt program enn å programmere alt selv fra grunnen av.
- At ulike programpakker kan gi ulike resultat, og at vi bør være klar over dette når vi velger redskaper.

Oppgave 2: Norsk stemming, norske tegn, unicode og UTF-8 (10 poeng)

OBS: Denne oppgaven er beskrevet ut i fra slik IFIs Linux-cluster oppfører seg. Den kan fortone seg annerledes på din egen maskin.

Så langt har vi sett på engelsk. Det er ikke så mye for norsk språk i NLTK, men det er en Snowball-stemmer. Den kan lastes med

```
>>> norsk_stemmer = nltk.SnowballStemmer('norwegian')
```

a) Test den på en del norske ord som "kaster", "kastet", "splittet", "splitten", "splitting", "splittelse". Gir den de resultatene du forventer?

For engelsk stemming er det et mål å stemme "split" og "splitting" til samme ord, og ikke "split" for den ene og "splitt" for den andre. Kan du komme på liknende eksempler for norsk hvor et leksem har noen former med enkelkonsonant foran en endelse eller på slutten av ordet, og andre former hvor den tilsvarende konsonanten er enkel foran en endelse eller på slutten av ordet? Hvordan takler den norske Snowball-stemmeren disse?

Innlevering: Svar på spørsmålene.

b) Den norske stemmeren har imidlertid en annen type problem. Hva skjer når du prøver

```
>>> norsk_stemmer.stem('Påskeøya')
```

Norske tegn er et problem. Dette er et problem som språkteknologer må forholde seg til. Vi må også kunne arbeide med andre språk enn norsk og engelsk – språk som bruker andre tegnsett. Du bør lese seksjon 3.3 Text Processing with Unicode i NLTK-boka, frem til "Extracting Encoded Text from Files" og notatet om Unicode før du går videre.

Du kan nå bruke dette til å "stemme" norske ord ved først å gjøre dem om til unicode-strenger.

Les inn og stem den lille teksten nor2. Tenk over hva du gjør med punktumene på slutten av setningene.

Innlevering: Kode + resultatet av å stemme nor2.

Oppgave 3: Leksikon og morfologisk analyse (40 poeng)

Gir stemming alt vi ønsker oss? Jeg søkte i Google først på "regjering" og "foreslått" og dernest på "regjering" og "foreslo". Her fikk jeg helt forskjellige resultat. Det første søket fanget også opp forekomster av "foreslå", men ikke "foreslo". For å se at "foreslo" er en form av "foreslå", holder det ikke med metoder som fjerner endelser. Da trengs det tilgang til et leksikon.

Språkbanken, som ligger ved Nasjonalbiblioteket, har en del tilgjengelige ressurser for norsk språk, bl.a. leksika, se
http://www.nb.no/Tilbud/Forske/Spraakbanken/Tilgjengelege-ressursar/Leksikalske-ressursar
Vi skal her bruke SCARRIE-leksikonet, som ble utviklet av Victoria Rosén og Koenraad De Smedt ved Universitetet i Bergen, og Torbjørn Nordgård ved Norges Teknisk-Naturvitenskapelige Universitet. Vi har lastet ned leksikonet. Deretter har vi fjernet en del trekk knyttet til stil, som ikke er relevant for oss, slått sammen former som blir like etter at denne informasjonen er fjernet og lagret det i et kompakt format. For så å lese det inn i Python, har vi laget noen enkle objekter og metoder. Du bør laste ned hele mappen *scarrie* til ditt eget område.

Leksikonet består av mer enn 300 000 forskjellige ordformer. Hver ordform består av selve formen og to typer trekk, et morfologisk trekk og et syntaktisk trekk. Vi har laget en klasse for dette i Python. Når *wf* er et bestemt objekt i denne klassen kan det for eksempel se slik ut

```
>>> wf.form
u'kaster'
>>> wf.morf_feat
'V,pres'
>>> wf.syn_feat
'V_pres_indic_active_main_ditrans!intrans!trans'
```

Vi har brukt Unicode for formen, men vanlig ASCII for trekkene siden de ikke inneholder norske tegn.

a) Hva tror du de ulike delene av de to trekkene *wf.morf_feat* og *wf.syn_feat* i eksempelet er ment å representere? Analyser dem bit for bit. (Du kan ha nytte av å se på flere eksempler før du svarer.)

Innlevering: Svar på spørsmålene.

b) Du kan arbeide interaktivt med leksikonet i Python. Kjør programmet *read_lexicon.py* fra mappen *scarrie*. Innlesningen foregår ved at du lager et objekt i klassen *ScaryLexicon*, som i det følgende eksempelet. Denne har en attributt *words*. Dette er en Python dictionary som inneholder aller ordformene. Vi kan plukke ut et tilfeldig av dem og inspisere nærmere, som vi også gjorde over.

```
>>> sclex = ScaryLexicon()
>>> wf2 = sclex.words['w100000']
>>> wf2.form
u'glupskest'
>>> wf2.morf_feat
'Adj,indef,sg'
>>> wf2.syn_feat
'Adj_mfn_sup_indef_sg'
>>> wf2.ident
'w100000'
```

For å få mindre å skrive i det følgende, kan du for eksempel sette

```
>>> words = sclex.words
```

Hvert ord (ordform) har altså en unik identifikator, her 'w100000'. Grunnen til at vi må ha identifikatorer for ordformer, og ikke kan bruke selve formen som identifikator, er at flere ulike ordformer, skrives likt. Se på *words['w140843'], words['w140853']* og *words['w140902']*. Disse har samme form, men avviker fra hverandre på andre egenskaper. Identifikatorene tjener som ''keys'' for dictionary *words*. Vi har også lagt inn identifikatoren i objektet for ordet for lett å kunne gå den andre veien.

Når vi skal analysere tekst, er vi interessert i å gjenkjenne ordene vi ser og trekke ut deres morfologiske og syntaktiske egenskaper. Vi kunne skrevet en funksjon, som til en ordform som *'kaster'* går gjennom hele dictionary sclex.words for å plukke de ordene som har denne formen. Men det ville bli ineffektivt i praksis. Vi vil derfor gjøre dette en gang for alle for alle ordformer, lagre resultatene i en dictionary, som vi så kan bruke etter behov.

Lag en Python-dictionary *form_to_ids* som til en overflateform, som 'kaster', gir en liste av identifikatorer: alle identifikatorer av ordformer med denne formen. Altså, for *'kaster'* skal den gi ['w140843', 'w140853', 'w140902']. Den skal bruke Unicode-representasjon av formene, slik at den også kan takle norske tegn. Lag deretter en funksjon som til en form gir alle mulige analyser, for eksempel til 'kaster' noe slikt som

V,pres V_pres_indic_active_main_ditrans!intrans!trans
N,pl,indef N_m_pl_indef
N,sg,indef N_m_sg_indef

Innlevering: Kode+resultatet av å analysere 'kastet' og 'øyer'.

c) Vi vil også at analysen vår skal gi grunnformen, som også kalles lemmaet, til de ulike ordfomene. For eksempel ønsker vi til 'foreslo' å finne 'foreslå'. Scarrie-leksikonet gir ikke direkte svar på dette. Det inneholder ikke lemmaer. Men det Scarrie-leksikonet gjør, er å samle sammen ordformer til et leksem (som det kaller "Lexical Entry"). Leksemet inneholder ikke annen informasjon utover at det grupperer sammen ordformer. Under innlesningen av ScaryLexicon har vi innført en identifikator for hvert leksem. I dictionary-en *sclex.lexemes* blir hver leksemidentifikator tilordnet en liste av ordifikatorer som sier hvilke ordformer som er i dette leksemet. For eksempel vil

```
>>> sclex.lexemes['x30027']
['w140902', 'w140903', 'w140904', 'w140905', 'w140906']>>>
```

Og ser vi nærmere etter står dette for:

w140902 kaster	N,sg,indef	N_m_sg_indef
w140903 kastere	N,pl,indef	N_m_pl_indef
w140904 kasteren	N,sg	N_m_sg_def
w140905 kasterer	N,pl,indef	N_m_pl_indef
w140906 kasterne	N,pl	N_m_pl_def

I dette tildfellet er den naturlige siteringsformen, den du vil finne i en ordbok, 'kaster'. Den kommer først, men det gjør den ikke for andre leksem. Sjekk f.eks. leksem x30016.

Du skal nå skrive en kodebit som til en leksemidentifikator returnerer den ordformen som er den mest naturlige siteringsformen, for eksempel til x30027 gir den 'kaster', og til x30016 gir den 'kaste'. Du må her tenke igjennom flere ting:

- Hvilken form er den mest naturlige å finne i en ordbok for ulike ordklasser, som verb (V), substantiv (N) og adjektiv (A)?
- Hvordan finner du frem til denne ved hjelp av trekkene morf_feat og syn_feat?
- Hvordan skriver du koden for å finne frem til denne?

Innlevering: Kode + resultatet av å bruke koden på leksemene med identifikatorer x30027, x30049, x30061

d) Nå får vi beregnet et lemma fra et helt leksem. Men det vi ønsker er en "lemmatizer", en rutine som tar en ordform og gir lemmaet, for eksempel til 'foreslo' gir 'foreslå'.

Utvid nå funksjonen *analyze()* fra pkt (b) slik at den i tillegg returnerer et lemma. Da kan resultatet bli noe slikt for 'murer' som argument:

mur	morf N,pl,indef	N_m_pl_indef
mure	morf V,pres	V_pres_indic_active_main_intrans!ref!trans
mure	morf N,pl,indef	N_f_pl_indef
murer	morf N,sg,indef	N_m_sg_indef

Innlevering: Kode+resultat av analyse av 'kaster', 'kastet', 'fisker', 'øyer', 'foreslo' og 'gås'.

e) Hvor mange orformer og hvor mange leksem er det i leksikonet? Gi eksempler på gode norske ord som ikke er i leksikonet selv med så mange ord.

Innlevering: Svar på spørsmålene

Etterord: en moral fra denne oppgaven er at det ikke bare er å lemmatisere et ord som utgangspunkt for søk. Formen alene, for eksempel 'murer', gir ikke et entydig lemma, men flere mulige lemma. For å finne ut hvilket som er riktig, må vi finne ut hvilken av de morfologiske og syntaktiske analysene som er riktig i setningen. Vi må tagge eller pare setningen. Og det forventer vi ikke av an søkemotor.

Del 2 Kontekstfrie grammatikker

Vi vil bruke NLTK sitt format for å skrive kontekstfrie grammatikker. Du kan skrive grammatikken til fil og bruke *nltk.data.load()* for å lese den inn i Python. Innlest slik blir grammatikken et Python-objekt. En forutsetning for at du skal forstå dette og komme i gang, er at du har jobbet deg gjennom delen om kontekstfrie grammatikker i ukeoppgavesett 4, og oppgave 3 på ukeoppgavesett 5.

Oppgave 4: Basisgrammatikk (20 poeng)

Vi tar utgangspunkt i oppgave5 fra oblig 1. Vi skal nå lage en kontekstfri grammatikk (CFG) for samme fragment som vi der laget en NFA for. Vi vil bruke NLTK-klassen *grammar* for å representere grammatikken. Du kan skrive grammatikken til fil og bruke *nltk.data.load()* for å lese den inn i Python.

På grunn av mulige problemer for noen typer parsere vil vi dele oppgaven i to. Se først på fragmentet uten preposisjonalfraser (PP-ledd) og lag en grammatikk for denne. Kall den *basic.cfg*. I grammatikken skal du bruke naturlige kategorinavn (ikke-terminaler) som NP og VP slik det er hintet til i oppgaveteksten for oppgave 5 (oblig 1) og vist i eksempler i NLTK-boka, seksjon 8.1-8.3. Leksikonet skal inneholde minst 10 av hver av:

- Substantiv, egennavn
- Substantiv fellesnavn (disse skal også forekomme både i bestemt og ubestemt form)
- Adjektiv
- Intransitive verb
- Transitive verb
- Minst 5 hver i de øvrige verb-klassene.

(Leksikonet skal ha litt størrelse for at vi senere skal se effekten av å effektivisere parserne.)

Lag så en chart-parser for denne grammatikken og kall den *parser_basic*. Du kan da teste den som i

```
>>> parser_basic = nltk.ChartParser(grammar)
>>> for n in parser_basic.parse("Mary ran".split()): print n
```

Vi bruker `nltk.Chartparser()` og ikke f.eks. `nltk.RecursiveDescentParser()` fordi den første kan behandle alle typer CFG-er, og vi er ikke interessert i parsingprosedyren foreløpig.

Merk også at du må bruke dokumentasjon for 2.utgave av NLTK-boka her. Kommandoene (og metodene) er noe endret og har fått nye navn siden 1.utgaven.

Innlevering: `basic.cfg`

Oppgave 5: PP-ledd (10 poeng)

Utvid `basic.cfg` til en grammatikk med PP-ledd, kall den `pp.cfg`, og lag chart-parser *parser_pp*. Du skal ha med minst 10 forskjellige preposisjoner.

Du kan teste grammatikken din med funksjonen *test* fra *test_cfg.py* (som ligger i mappen *cfg* på fellesområdet) og samme eksempelfil *norsk_testset.txt* som i innleveringssett 1.

Denne sjekker bare om det finnes minst en analyse for hver av de grammatiske setningene. Det er et poeng med kontekstfrie grammatikker at vi kan få frem flertydigheter og at de reflekteres i trær. Se på setningene:

a) Dyret i huset ved vannet sov.
b) Kari sov i huset ved vannet.
c) Kari likte huset ved vannet.
d) Kari likte dyret i huset ved vannet.

Hver av setningene (a-c) har to analyser. Sjekk at du får dette. For hver av de tre eksemplene forklar med en til to setninger intuitivt forskjellen mellom de to analysene. Hvor mange analyser får (d)?

Innlevering: Fila som inneholder *pp.cfg*. Kjøringseksempler som viser trærne du får for setningene (a-d). Forklaringene det spørres etter.

Oppgave 6: En kontekstfri grammatikk til (10 poeng)

I innleveringssett 1, oppgave 2, lagde vi en DFA og et regulært uttrykk for språket L2 med alfabet A = {a, b} med følgende to egenskaper:

- Et velformet uttrykk skal inneholde et odde antall *a*-er, altså 1, 3, 5, 7, eller … 2n+1, …
- *b*-ene skal komme i par, dvs. enhver *b* skal ha en annen *b* rett til venstre for seg, eller rett til høyre for seg, men ikke på begge sider

Lag nå en kontekstfri grammatikk for det samme språket.

Innlevering: Den kontekstfrie grammatikken.

SLUTT