INF 2820 V2016: Innleveringsoppgave 2

- **Besvarelsene skal leveres i devilry innen fredag 3.3 kl 18.00**
- **Det blir 5 sett med innleveringsoppgaver. Hvert sett gir inntil 20 poeng. Til sammen kan en få inntil 100 poeng. For å kunne gå opp til eksamen må du ha oppnådd minst 60 poeng til sammen på de 5 settene.**
- **For å oppnå poeng er det viktigere at du prøver å løse en oppgave enn at løsningen er helt riktig. Vektleggingen er slik sett annerledes enn på eksamen.**
- **Innleveringsfrister utsettes kun i forbindelse med sykdom (ved egenmelding eller legeerklæring), og omlevering vil ikke være mulig**
- **Vi går ut i fra at du har lest og er kjent med IFIs reglementet for obligatoriske oppgaver: http://www.mn.uio.no/ifi/studier/admin/obliger/index.html og også kjenner rutinene for behandling av fuskesaker http://www.uio.no/om/regelverk/studier/studier-eksamener/fuskesaker/**
- **Dette er en individuell oppgave. Det skal ikke leveres felles besvarelser.**
- **Du kan levere om igjen i Devilry inntil fristen, men inkluder i så fall alt du vil levere i den siste innleveringen.**
- **Filene det vises til blir ikke lagt ut i vortex. Du finner dem på IFIs cluster under**
  - **/projects/nlp/inf2820/fsa**
  - **/projects/nlp/inf2820/scarrie**

Forutsetninger: Klasser i Python

Mange av de praktiske oppgavene i dette emnet er i Python og NLTK. Siden Python er et objektorientert språk og NLTK gjør stor bruk av klasser, trenger alle å gjøre seg litt kjent med hvordan klasser i Python virker og spesielt med noen av klassene NLTK bruker. De av dere som tok INF1001 i høst, bør være godt kjent med Pythons klasser og for dere som er vant med klasser i Java, bør det ikke være vanskelig å lese Python-klasser. Du finner en innføring i Pythons bruk av klasser i kap. 15 i http://openbookproject.net/thinkcs/python/english3e/.

På den andre siden legger ikke NLTK opp til at vi selv skal lage klasser og NLTK-boka inneholder ikke eksempler som konstruerer klasser. I løsning av oppgavene står du fritt i valget mellom å bruke klasser eller, som NLTK-boka, skrive prosedyrer som tar klasser som parametre.

Oppgave 1: NFA, algoritme for gjenkjenning, klokkeslettuttrykk (5 poeng)

På forelesningen 3. februar gikk vi gjennom en smart algoritme for gjenkjenning med NFA, og vi ga en enkel Python-implementasjon. Denne finner du i filen *nfa_smart.py*. Vi har gjort et par utvidelser i forhold til forelesningen. Vi tillater NFA-er med forkortelser og med epsilon-transisjoner. Selve algoritmen, den delen du trenger å forholde deg til, er nrec(tape, nfa, trace=0). Her kan *tape* være en liste eller en streng, mens *nfa* er et objekt av klassen NFA. Det er også et opsjonelt argument *trace*, der *trace*=1 gir utskrift av prosessen underveis. Denne prosedyren svarer *True* hvis tapen er i språket og *False* ellers.

Automater kan skrives i en fil som vist i *template.nfa*. Her uttrykker # en epsilontransisjon (hoppekant). Symboler uten anførselstegn er forkortelser for endelig mange symboler, f.eks. er DET en forkortelse for *'a'* og *'the'*. Kanten *0 DET 2* betyr at du fra tilstand 0 går til tilstand 2 hvis du ser *'a'* eller *'the'* som neste symbol i tapen, men *DET* er ikke et lovlig tape-symbol. At *DET* er en forkortelse

for '*a*' og '*the*' står under *ABRS*. Programmet er ganske lite fleksibelt med hensyn til hvordan automatfilen ser ut. Her bør du følge *template.nfa* med hensyn til linjeskift o.l.

I ukeoppgave 3 var en oppgavene å lage et nettverk for norske klokkeslettuttrykk. Vi skal nå teste nettverket med dette programmet. (Hvis du ikke alt har gjort det:) Lag en NFA for norske klokkeslettutrykk. Du kan bruke 12 timers klokke og "gammel tellemåte", eks. '*ti over halv fire*'. Du kan også ta med uttrykksmåten '*femten førti*', men det er frivillig

Eksempler:

Skal være med	Skal ikke være med
tre	halv
halv fem	kvart på halv ni
ti på sju	fem over kvart på ti
fem over halv åtte	ti på halv
kvart over ni	halv over to
kvart på ti	halv på tolv

Det skal være minst ett uttrykk for alle minutter gjennom dagen. NFA-en skal skrives på samme format som *template.nfa* og testes med *nfa.py* på eksemplene over.

Innlevering: Filen med NFA og kjøringseksempel med eksempeluttrykkene både de som skal være med og de som ikke skal være med.

Oppgave 2: Stemming (3 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"? Hvordan vil de to stemme "goldfishes"?

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

b) Porter-stemmer er langt på vei blitt en standard stemmer for engelsk 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')
```

Vi skal sammenlikne Porter-stemmer og Snowball-stemmer. Bruk teksten 'grail.txt' som kan lastes som en liste av ord med kommandoen

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

Finn deretter 6 forskjellige ordformer hvor Snowball-stemmeren og Porter-stemmeren gir forskjellig resultat. Vi regner ikke forskjeller mellom stor og liten bokstav som forskjeller i denne sammenheng.. (Hint: Finn alle ordforekomster hvor de to gir forskjellig resultat, og bruk set( ) til å plukke forskjellige ordformer fra disse.)

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 3: Leksikon og morfologisk analyse (12 poeng)

Vi skal arbeide med morfologi og leksikon for norsk. Først finner vi et tilgjengelig leksikon for norsk, og så bearbeider vi det for våre formål. Det kan være greit å vite at Språkbanken, som ligger ved Nasjonalbiblioteket, har en del tilgjengelige ressurser for norsk språk, bl.a. leksika, se http://www.nb.no/Tilbud/Forske/Spraakbanken. Vi har lastet ned SCARRIE-leksikonet, som ble utviklet av Victoria Rosén og Koenraad De Smedt ved Universitetet i Bergen, og Torbjørn Nordgård ved Norges Teknisk-Naturvitenskapelige Universitet. Leksikonet distribueres i XML (se utsnitt i appendiks). For å gjøre det lettere å redistribuere, 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 (se appendiks). 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/maskin.

Du kan arbeide interaktivt med leksikonet i Python 3 (eller ipython 3). Kjør programmet *read_lexicon_py3.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 alle ordformene. Leksikonet består av mer enn 300 000 forskjellige ordformer. Hver ordform består av selve formen og to typer trekk, et morfologisk trekk og et morfosyntaktisk trekk. (De to trekkene er til dels overlappende.) Vi har laget en klasse for slike ordformer i Python. Vi kan plukke ut et tilfeldig ord og inspisere nærmere.

```
>>> sclex = ScaryLexicon()
>>> wf2 = sclex.words[‘w99263’]
>>> wf.form
'glupskest'
>>> wf2.morf_feat
'Adj,indef,sg'
>>> wf2.syn_feat
'Adj_mfn_sup_indef_sg'
>>> wf2.ident
‘w99263’
```

Hvert ord (ordform) har altså en unik identifikator, her ‘w99263’. 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 'w140165', 'w140116', 'w140106'. 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.

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

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

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 prosedyre, 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 ny dictionary, som vi så kan bruke etter behov.

a) 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 ['w140165', 'w140116', 'w140106'].

Lag deretter en funksjon som til en form gir alle mulige analyser, f.eks. 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'*.

b) Verdien av de to trekkene *wf.morf_feat* og *wf.syn_feat* er ganske kompakte og mer beregnet for lesing av maskiner enn av mennesker. Forklar hva du mener verdien av disse trekkene for analysene av 'kaster' sier. Analyser dem bit for bit. (Du kan ha nytte av å se på flere eksempler før du svarer.)

Innlevering: Svar på spørsmålene.

c) Vi vil nå endre analyserutinen til å gi mer forståelig informasjon. For words['w140116'] kan det se ut som noe slikt

```
kaster
cat:  N
gen:  neut
num:  pl
def:  def
```

Alle ordene i leksikonet er tilordnet en ordklasse eller kategori (eng: "part of speech", "catagory") så dette trekket skal med for alle ord. For noen ordklasser er dette alt som finnes. For andre ordklasser, er det en god del mer. Prøv å tolke og skrive ut all informasjon for ordene som tilhører klassene N, V, og Det.

Innlevering: Kode+resultatet av å analysere *'kaster'*, *'kastet'*, *'noen'* og *'et'*.

d) Vi vil også at analysen vår skal gi grunnformen, som også kalles lemmaet, til de ulike ordformene. 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 laget en klasse Lexeme. Et objekt i denne klassen inneholder en unik identifikator og en liste ordidentifikatorer som sier hvilke ordformer som er i dette leksemet. Vi har laget en dictionary som til leksemidentifikatorer gir leksemet med denne identifikatoren.

```
>>> lexemes['x29482'].word_ids
 ['w140165', 'w140166', 'w140167', 'w140168', 'w140169']
```

Og ser vi nærmere etter står dette for:

w140165 kaster	N,sg,indef	N_m_sg_indef
w140166 kastere	N,pl,indef	N_m_pl_indef
w140167 kasteren	N,sg	N_m_sg_def
w140168 kasterer	N,pl,indef	N_m_pl_indef
w140169 kasterne	N,pl	N_m_pl_def

I dette tilfellet er den naturlige siteringsformen, den du vil finne i en ordbok, *'kaster'*. Den kommer først, men det gjør den ikke for andre leksem (se f.eks. lexemes['x34351']).

Du skal nå skrive en kodebit som til en leksemidentifikator returnerer den ordformen som er den mest naturlige siteringsformen, for eksempel til x29482 gir den *'kaster'*, og til x29471 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.

e) 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 rutinen fra pkt (c) slik at den i tillegg returnerer et lemma. For words['w140116'] kan det se ut som noe slikt

```
kaster
cat:  N
lemma:      kaste
gen:  neut
num:  pl
def:  de
```

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

f) Vi skal gjøre noen tellinger over leksikonet. Hvor mange leksemer og hvor mange ord er det i leksikonet?

Vi er interessert i å vite litt mer om flertydighet. Hvor mange forskjellige ordformer er det i leksikonet? Her vil altså *'kaster'* telle som én, selv om flere ord har denne formen, mens *'kaste'* og *'kaster'* vil telle som to, selv om de tilhører samme leksem.

Vi vil nå bare være interessert i ordklasser, ikke i annen morfologisk informasjon. Da vil *'kaster'* telle som to, en gang som V og en gang som N, men *'kaster'* vil ikke telle som tre. Hvor mange slike par er det i leksikonet?

(OBS: Disse tellingene kan gi et litt skjevt bilde. Det kan være en del dubletter i leksikonet. Spesielt finner vi de samme ordene i kategorien PossDet og i Det. Skal vi bruke leksikonet videre, bør vi fjerne dubletter.)

Innlevering: Kode som gir svar på de fire spørsmålene og resultat fra kjøring.

g) Vi vender tilbake til ordklassene (kategoriene) fra pkt. (c). Hvor mange slike ordklasser er det, og hvor mange ord i hver klasse?

Innlevering: Kode som gir svar på spørsmålene og resultat fra kjøring.

h) Bonusoppgave (for de som vil undersøke mer:) Hvilke ordformer i norsk er de mest flertydige ifølge leksikonet og hvor mange analyser har de? (ikke poengtrekk for ikke å gjøre denne.)

Innlevering: Kode som gir svar på spørsmålene og resultat fra kjøring.

- SLUTT -

Appendiks

Utdrag fra main i formatet som leses av read_python_py3.py

```
Lexeme
glupende;Adj,pos;Adj_mfn_pos_e
glupende;Adj,pos,indef,sg;Adj_mfn_pos_indef_sg
Lexeme
glupsk;Adj,pos,indef,sg;Adj_mfn_pos_indef_sg
glupske;Adj,pos;Adj_mfn_pos_e
glupskere;Adj,sgpl;Adj_mfn_comp_defindef_sgpl
glupskest;Adj,indef,sg;Adj_mfn_sup_indef_sg
glupskeste;Adj,sgpl;Adj_mfn_sup_def_sgpl
Lexeme
gluten;N,sg,indef;N_n_sg_indef
glutenet;N,sg;N_n_sg_def
```

Utdrag fra originalt Scarrie-leksikon

```
<LexicalEntry><Lemma />
<WordForm>
 <feat att="writtenForm" val="glupende" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,pos" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_pos_e" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glupende" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,pos,indef,sg" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_pos_indef_sg" />
</WordForm>
</LexicalEntry>

<LexicalEntry><Lemma />
<WordForm>
 <feat att="writtenForm" val="glupsk" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,pos,indef,sg" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_pos_indef_sg" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glupske" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,pos" />
 <feat att="replacement" val="" />
```

```
 <feat att="synFeat" val="Adj_mfn_pos_e" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glupskere" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,sgpl" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_comp_defindef_sgpl" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glupskest" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,indef,sg" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_sup_indef_sg" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glupskeste" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="Adj,sgpl" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="Adj_mfn_sup_def_sgpl" />
</WordForm>
</LexicalEntry>

<LexicalEntry><Lemma />
<WordForm>
 <feat att="writtenForm" val="gluten" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="N,sg,indef" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="N_n_sg_indef" />
</WordForm>
<WordForm>
 <feat att="writtenForm" val="glutenet" />
 <feat att="corrStyle" val="N" />
 <feat att="morSynFeat" val="N,sg" />
 <feat att="replacement" val="" />
 <feat att="synFeat" val="N_n_sg_def" />
</WordForm>
</LexicalEntry>
```