Obligatorisk innlevering 2b - Syntaks og semantisk klassifisering

Høst 2025

Innleveringsfrist: fredag 7. november kl. 23.59

Innlevering av oppgaven skjer i Devilry. Det er viktig at du setter deg inn i retningslinjene for obligatoriske oppgaver. Besvar de tre første deloppgavene i én fil som angir brukernavnet ditt slik:

`oblig2b_brukernavn.py`

Oppgave 4 skal besvares i en egen kodefil `oppgave4.py` som du også leverer. Du trenger ikke å levere mappen `NoReC`.

Besvarelsen må kunne kjøres med Python i terminalen, slik som tidligere. Ikke pakk filene inn i en ZIP-mappe.

Poeng

En perfekt løsning av denne innleveringen er verdt 100 poeng. Du må vise koden din i oppgave 2 og 4. Koden må kunne kjøres og må inneholde kommentarer som forklarer hva koden gjør.

1 Kontekstfrie grammatikker (eksamen våren 2017) (10 poeng)

Anta at vi har følgende grammatikk:

`S → NP VP`

`NP → Per | Kari | Ola | boka`

`NP → D N`

`D → en`

`N → bok`

`PP → P NP`

`P → med`

`VP → V NP`

`VP → V NP NP`

`VP → VP PP`

`V → gir`

`V → slår`

Setningen *"Per slår Ola med boka"* er strukturelt flertydig.

1. Beskriv de to ulike tolkningene med egne ord.
2. Den vedlagte grammatikken gir kun én analyse for denne setningen. Hvordan kan du utvide grammatikken slik at den kan representere flertydighet av denne typen?
3. Er grammatikken rekursiv? Hvis ja, på hvilken måte? Hvis nei, gi et eksempel på en rekursiv regel.

2 Chunking (15 poeng)

I denne oppgaven skal du lage en NP-chunker og evaluere den. For å få til dette skal du bruke `nltk.RegexpParser`, som beskrives i NLTK-boken, kapittel 7:

[ ]:
```
grammar = 'NP: {<DT>?<JJ>*<NN>}'
cp = nltk.RegexpParser(grammar)
```

Bruk minst fem mønstre som ikke forekommer i boken (eller utvid mønstrene fra boken på fem forskjellige måter). Mens du utvikler chunkeren din kan du teste den på treningskorpuset fra CoNLL2000. Dette korpuset er hentet fra Penn Treebank og bruker altså ordklassetaggsettet derfra (som er gjengitt og beskrevet kort på innsiden av omslaget foran i Jurafsky & Martin og her). Korpuset må først lastes ned, som tidligere, med `nltk.download("conll2000")`. Deretter kan det hentes inn i koden din slik:

[ ]:
```
from nltk.corpus import conll2000
training_chunks = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
```

Hvis du trenger inspirasjon til mønstrene dine kan du skrive ut treningskorpuset og inspisere elementene merket `NP`. Du evaluerer chunkeren din på treningskorpuset slik (gitt at `cp` er variabelen som inneholder chunkeren din og `training_chunks` er variabelen som inneholder treningskorpuset):

[ ]:
```
cp.evaluate(training_chunks)
```

Dokumenter chunkeren grundig og forklar hvordan den fungerer. Evaluer chunkeren ved å beregne dens nøyaktighet på **testkorpuset** fra CoNLL2000. Dette henter du ut slik:

[ ]:
```
test_chunks = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
```

Kopier output fra evalueringen inn i kodefilen din som en kommentar.

3 Naive Bayes klassifisering (15 poeng)

Vi ønsker å utvikle et system som automatisk skal angi om et dokument er positivt eller negativt. Vi har et lite korpus som er annotert med sentiment som presentert i tabellen under.

Treningsdata

Kat.	Dokument
POS	en fortreffelig start på konserten
POS	et samspilt og førsteklasses orkester
POS	har aldri laget et ordentlig dårlig album
NEG	konserten viste seg å være én av de dårligste i mitt liv
NEG	oppskrytt orkester
NEG	kjedelig konsert
NEG	et ordentlig dårlig album

Testdata

Kat.	Dokument
?	førsteklasses artist men dårlig og kjedelig album
?	fortreffelig orkester og flott album

Vi ønsker å beregne den mest sannsynlige klassen k for hvert dokument i testdataen, gitt trekkene for dokumentet. Vi bestemmer oss for å løse denne oppgaven ved Naive Bayes-klassifisering, angitt ved følgende formel:

$$\hat{k} = \underset{k \in K}{\text{argmax}} P(k) \prod_{j=1}^{n} P(v_j|k)$$

1. Forklar hvordan Naive Bayes-formelen er utledet fra den opprinnelige sannsynligheten $P(k|v)$ og beskriv hvilke antagelser som er gjort. Husk at hovedantagelsen i Naive-Bayes er:

$$\hat{k} = \underset{k \in K}{\text{argmax}} P(k|v)$$

2. Beregn den mest sannsynlige klassen for hvert av testdokumentene *"førsteklasses artist men dårlig og kjedelig album"* og *"fortreffelig orkester og flott album"*. Anta en Naiv Bayes-klassifiserer og bruk "legg til én-glatting".

3. Er du enig med klassifiseringen av dokumentene? Hvis ikke, forklar og foreslå hva du tror må til for å få Naive Bayes til å angi riktig klasse.

4 Naive Bayes klassifisering med Python (60 poeng)

I denne oppgaven skal du bruke NLTKs `NaiveBayesClassifier` til å lage flere klassifiserere. Du skal også besvare et par teorispørsmål. Svarene på disse skriver du som kommentarer i kodefilen.

Her skal du bruke et begrenset utvalg av det såkalte NoReC-korpuset der du skal fokusere på bokanmeldelsene. Siden korpuset ikke er en del av NLTK har vi skrevet litt prekode som du kan bruke for å laste inn korpuset og bruke det slik korpora fra NLTK blir brukt. Du finner også kode som deler dataene dine inn i separate datasett for trening, utvikling og testing (`train_set`, `dev_set` og `test_set`). Du må laste ned korpuset fra emnesiden, og ha mappen `NoReC` i samme mappe som kodefilen din.

I denne oppgaven skal du trene flere Naive Bayes-klassifiserere og rapportere resultater for hver av dem. Bruk Python til å utføre beregningene dine. Du finner prekoden i filen `oppgave4.py`. Du skal også skrive din egen kode i denne filen. Les gjennom hele oppgaven før du begynner og gjør deg kjent med kodefilen og prekoden i den.

4.1 Lese inn dokumenter

Først leser du inn dokumentene og sorterer dem etter kategori (*pos* og *neg*). Vi ønsker her å lage to lister `pos_reviews` og `neg_reviews`. Disse skal inneholde en liste med ord for hvert dokument. Hver fil i korpuset tilsvarer ett dokument; én anmeldelse. Skriv koden i metoden `main()`. Dette er metoden som utføres når du kjører kodefilen. Pass på å være nøye med korrekt linjeinnrykk når du jobber med kode i metoder.

- Bruk metoden `load_corpus()` for å lese inn korpuset. Som argument bruker du strengen `'NoReC/'`.
- Iterer over dokumentene i korpuset. For hvert dokument må du avgjøre om det tilhører kategorien *pos* eller *neg* og legge til dokumentordene i `pos_reviews` eller `neg_reviews`.

4.2 1000 mest frekvente ord som trekk

Du skal nå fylle ut metoden `feature_extractor_top_1000()`. Denne tar inn en ordliste `document` som argument og returnerer en ordbok `features` som inneholder trekkene for dokumentet.

- Begynn med å finne de 1000 mest frekvente ordene. Dette kan du gjøre med et `Counter`-objekt. Tidligere har vi brukt metoden `most_common()` for å hente ut de mest frekvente ordene tuppelvis med tilhørende frekvens, men her er vi kun interesserte i ordene. For å hente ut en liste med de `n` hyppigste ordene, uten tilhørende frekvens, fra et `Counter`-objekt `fd`, kan du bruke følgende "list comprehension":

```
[ ]: frequent_words = [word for word, count in fd.most_common(n)]
```

- Iterer over de mest frekvente ordene. For hvert ord ønsker vi å legge til et trekk i `features` som sier at dokumentet inneholder det ordet. Dersom ordet f.eks. er "fantastisk", kan trekket beskrives med strengen `'contains(fantastisk)'`. Dette blir nøkkelen. Verdien setter du til `True`. Til slutt returneres `features`.
- Nå skal du bruke `feature_extractor_top_1000()` i `main()` til å hente ut trekk. Du må lage to ny lister `pos_features` og `neg_features`. Disse skal tuppelvis inneholde en ordbok med trekk som representerer hvert dokument, og strengen `'pos'` for positive doku-

menter eller `'neg'` for negative dokumenter. Iterer over anmeldelsene i hhv. `pos_reviews` og `neg_reviews` og hent ut trekk.

- Del opp dataen i `train_set`, `dev_set` og `test_set`. Dette kan du gjøre med den ferdigdefinerte metoden `split_data()`.
- Du kan nå trene en klassifiserer og beregne nøyaktigheten på `dev_set` slik:

```
[ ]: classifier = NaiveBayesClassifier.train(train_set)
     accuracy = classify.accuracy(classifier, dev_set)
```

- Rapportér resultatene oversiktlig med utskrifter til terminalen. Inkluder hvilken modell du rapporterer for ("1000 mest frekvente ord" i dette tilfellet), lengden på `train_set`, `dev_set` og `test_set`, nøyaktigheten `accuracy` og til slutt de 30 mest informative trekkene i modellen. Disse skrives ut med `classifier.most_informative_features(30)`.
- I denne modellen har vi ikke luket ut tegnsetting og stoppord. Hva er et stoppord? Forklar og gi minst 3 eksempler.
- Hvordan tror du håndtering av tegnsetting og stoppord kunne påvirket modellen?

4.3 Flere typer trekk

Du skal nå fylle ut flere metoder som henter ut trekk:

- `feature_extractor_bow()` med "bag of words" som trekk.
- `feature_extractor_bow_bigrams()` med "bag of words" og bigram som trekk.
- `feature_extractor_bow_bigrams_trigrams()` med "bag of words", bigram og trigram som trekk.

Disse fungerer på samme måte som `feature_extractor_top_1000()` ved at de tar inn en ordliste `document` og returnerer en ordbok med trekk `features`. Denne ordboken er i hver metode strukturert på samme måte med nøkler, som `'contains(ord)'`, og verdier `True`.

- Til "bag of words" må duplikate ord fjernes med `set()`, og tegnsetting og stoppord lukes ut med `clean_words()`. Til n-grammene ønsker vi ikke å gjøre dette.
- Iterer over ordene (og ev. n-grammene) og legg til trekk i `features`.
- Bruk `train_set` til å trene en ny modell med hver av de nye "trekkmetodene" og rapporter resultatet til terminalen, slik du gjorde for modellen med de 1000 mest frekvente ordene som trekk. Husk å vise de mest informative trekkene også! Dette gjør du i `main()`.

4.3.1 "Bag of words"

Her må du for hvert ord i dokumentet lage et trekk som sier at dokumentet inneholder det ordet.

4.3.2 "Bag of words" og bigram

Som "bag of words", men med bigram i tillegg. Her kan trekkene med bigram bli seende f.eks. slik ut: `'contains(("et", "flott"))'`

Du trenger ikke å tenke på setningsgrenser og markører for setningsstart og setningsslutt her.

4.3.3 "Bag of words", bigram og trigram

Som "bag of words" og bigram, men med trigram i tillegg. Her kan trekkene med trigram bli seende f.eks. slik ut: `'contains(("et", "flott", "verk"))'`

Du trenger ikke å tenke på setningsgrenser og markører for setningsstart og setningsslutt her.

4.4 Evaluering

- Hvilken av modellene gir høyest nøyaktighet på `dev_set`? Test denne modellen til slutt på `test_set` og rapporter nøyaktigheten i en utskrift til terminalen.
- Kan du tenke deg andre trekk eller preprosesseringer som kunne ha forbedret den beste modellen ytterligere?