IN2110 – Obligatorisk innlevering 1b

*Leveres innen **fredag 8. mars kl. 23.59** i Devilry.*

Pre-kode: `https://www.uio.no/studier/emner/matnat/ifi/IN2110/v19/obliger/in2110-oblig-1b-precode.py`

Oppsett

For IN2110 har vi laget et utviklingsmiljø for Python som inneholder programvare og data for de obligatoriske oppgavene. Om du sitter på en av IFI sine Linux maskiner kan du kjøre

```
$ ~eivinabe/in2110-shell
```

Om du sitter på egen maskin kan du logge deg inn på IFI sitt Linux-cluster og kjøre

```
$ ssh -Y <ditt-uio-bruker-navn>@login.ifi.uio.no
```

for og så starte `in2110-shell` derfra. Argumentet `-Y` gir deg grafisk grensesnitt via SSH om klienten støtter det. Det er også mulig å installere miljøet på egen maskin, men er på eget ansvar! Se `https://github.uio.no/eivinabe/in2110-shared` for mer info om installasjon på egen maskin. Om du allerede har installer miljøet på egen maskin må du kjøre

```
$ ./update.sh
```

for å oppdatere miljøet for du setter i gang med oppgaven.

Innleveringsformat

Navn	Accuracy (%)
System X	55.6
System Y	49.2
System Z	61.2

Tabell 1: Eksempel på tabell

Innleveringen skal bestå av én Python-fil med kode og en rapport i PDF-format. I rapporten skal dere beskrive hva dere har gjort og begrunne valgene dere har

gjort underveis. Rapporten skal også inneholde de tallene og figurene det spørres om, samt svar på spørsmål. Tall bør presenteres i en tabell, slik som illustrert i tabell 1. Det forventes at dere bruker hele setninger – stikkord er ikke nok. Python-filen skal inneholde de ferdig implementerte klassene og funksjonene i pre-koden, samt koden dere har brukt for å produsere de resultatene dere presenterer i rapporten.

Oppgave 1 Ordvektorer

For å lage ordvektorer trenger vi et korpus. Fordi dette er ikke-veiledet læring kan vi benytte oss av store mengder ikke-annotert tekst. Vi skal bruke Norsk aviskorpus[1] som består av tekst fra norske aviser i perioden 1998-2014. Korpuset inneholder mer enn én milliard tokener, så vi kommer kun til å jobbe med et lite utvalg, mer bestemt tekster på Nynorsk fra perioden 1998-2005.

a) Pre-prosessering

Funksjonen `aviskorpus_10_nn.sentences()` returnerer en generator over alle setningene i korpuset vi skal bruke. Setningene er ferdig tokenisert.

> Gjør ferdig implementasjonen av `preprocess()`. Den skal ta inn en generator over setningene i korpuset og returnere en liste med setninger, hvor hver setning er en liste med tokens. Legg til annen pre-prosessering som f.eks. downcase, fjerning av spesialsymboler, o.l. og begrunn valgene dere har tatt.
>
> Eksempelkjøring:

```
>>> list(aviskorpus_10_nn.sentences())[10]
'<s> " ... informasjon er ikkje kunnskap og kunnskap er ikkje visdom " . </s>'
>>> sentences = preprocess(aviskorpus_10_nn.sentences())
>>> sentences[10]
['<s>', '"', '...', 'informasjon', 'er', 'ikkje', 'kunnskap',
  'og', 'kunnskap', 'er', 'ikkje', 'visdom', '"', '.', '</s>']
```

b) Kontekst

Det er mange måter å definere ord-konteksten, men vi vil her definere den som et vindu rundt det ordet vi ser på. Antall ord i konteksten vil være 2 × vindustørrelse.

> Gjør ferdig funksjonen `context_window()`. Denne skal ta inn en setning, samt posisjonen til gjeldene ord og størrelse på vindu. Funksjonen skal returnere en liste som inneholder ordene i konteksten.

[1] `https://www.nb.no/sprakbanken/show?serial=sbr-4&lang=nb`

```
>>> context_window(sentences[10], 6, 3)
['informasjon', 'er', 'ikkje', 'og', 'kunnskap', 'er']
```

I starten og slutten av setningen vil kontekstvinduet havne delvis utenfor setningen. Funksjonen skal da returnere de delene av konteksten som befinner seg innenfor setningen.

```
>>> context_window(sentences[10], 0, 3)
['"', '...', 'informasjon']
```

c) Telling

For å lage ordvektorene skal vi telle over ordene i kontektsten for hvert ord. Fordi tellebaserte metoder resulterer i høy-dimensjonale vektorer med få aktive verdier (sparse vector) vil vi bruke Python dictionaries for å representere co-occurence matrisen. Hver dictionary skal kun inneholde de aktive verdiene, dvs. alle verdier som ikke er 0. For å begrense ordforrådet – og dermed størrelsen på matrisen – vil vi kun bruke de mest frekvente ordene.

Gjør ferdig implementasjonen av `fit()`. Funksjonen skal bruke `window_size` og `vocab_size` som satt i konstruktøren. Bruk `context_window()` for å hente konteksten for hvert ord i setningene. Den ferdige matrisen skal lagres i `self.matrix`.

Tren ordvektorer ved å kalle `fit()`. Velg selv verdier for `vocab_size` og `window_size`. Størrelse på vokabularet skal begrense modellen til å kun bruke de mest frekvente ordtypene i korpuset. NB: Ved veldig stor `vocab_size` vil tellingen ta veldig lang tid og matrisen vil kreve veldig mye minne. Start med en lavere verdi, som f.eks. 1000, så kan dere heller øke senere.

Sørg for at `transform()` fungerer som den skal. Den skal ta inn en liste med ord og returnere en liste med ordvektorer.

Eksempel:

```
>>> vec = WordVectorizer(5000, 5)
>>> vec.fit(sentences)
>>> vec.matrix['nynorsk']['bokmål']
144
```

Oppgave 2 Lengde, avstand og likhet

a) Vektorlengde

$$||v|| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Fullfør funksjonen `vector_norm()`. Funksjonen skal ta inn et ord og returnere lengden til ordvektoren.

Eksempel på output (tallene dere får er nok noe annerledes):

```
>>> vec.vector_norm("nynorsk")
1267.0745834401382
```

b) Avstand

$$\mathrm{d}(p, q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Fullfør funksjonen `euclidean_distance()`. Funksjonen skal ta inn to ord og beregne den euklidske avstanden mellom vektorene for de to ordene.

Eksempel på output (tallene dere får er nok noe annerledes):

```
>>> vec.euclidean_distance("bokmål", "nyorsk")
476.92347394524415
```

c) Likhet

$$\text{similarity}(p, q) = \frac{p \cdot q}{||p|| \times ||q||} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \times \sqrt{\sum_{i=1}^{n} q_i^2}}$$

Fullfør implementasjonen av `cosine_similarity()`. Funksjonen skal ta inn to ord og bergne cosine similarity mellom vektorene for de to ordene. Fordi vektorene er sparse kan vi her spare litt regnekraft ved å utnytte det at å gange med 0 gir 0.

Eksempel på output (tallene dere får er nok noe annerledes):

```
>>> vec.cosine_similarity("bokmål", "nynorsk")
0.9283819574714427
```

d) Normalisering

$$\hat{v}_i = \frac{v_i}{||v||}$$

Gjør ferdig `normalize_vectors()`. Denne funksjonen skal normalisere alle vektorene slik at de blir enhetsvektorer.

Endre `fit()` slik at den kaller `normalize_vectors()` om vektoriseereren er opprettet med `normalize=True`.

Endre `cosine_similarity()` til å kun beregne dot-product hvis vektorene er normalisert.

Eksempel:

```
>>> vec.normalize_vectors()
>>> vec.vector_norm("nynorsk")
0.9999999999999809
```

NB: Verdien blir ikke nøyaktig 1 pågrunn av avrundingsfeil.

e) Naboskap

Vi ønsker å finne de nærmeste naboene til en gitt ordvektor. Dette kan vi gjøre ved hjelp av enten `cosine_similarity()` eller `euclidean_distance()`.

Gjør ferdig `nearest_neighbors()`. Denne skal ta inn et ord og returnere de k nærmeste naboene til ordet. Dere kan selv velge om dere vil bruke cosine similarity eller euclidean distance, men tenk på hvor mye regnekraft som trengs. Begrunn valget deres.

Eksempel (med cosine similarity):

```
>>> vec.nearest_neighbors("nynorsk", k=4)
[('nynorsk', 0.9999999999999951),
  ('jorda', 0.957391577933915),
  ('deira', 0.9553782853654251),
  ('utanfor', 0.9549852291610347)]
```

Oppgave 3 Visualisering

Nå som dere har implementert et fullstendig vektorrom kan dere bruke funksjonene dere har skrevet til å utforske ordvektorene. Bruk `nearest_neighbors()` til å søke dere rundt. Finn to klynger med ord som befinner seg et stykke unna hverandre. Bruk funksjonen `visualize_word_vectors()` for å visualisere de utvalgte ordvektorene:

```
>>> visualize_word_vectors(vec, ["nynorsk", "bokmål"])
```

Legg ved visualiseringen i rapporten.