**Oblig 1b**  
Rapport

Justyna, Joanna, Adam

**Oppgave 1a**

I denne oppgaven skrev vi ferdig funksjonen **preprocess(sentences)** ved å lage en liste som heter korpus[], for å deretter bruke en for løkke som går gjennom setningene i korpuset og splitter de på mellomrom til ord ved hjelp av funksjonen **split()**. Etter disse operasjonene elementer blir lagret i listen korpus[].

Vi har også skrevet andre typer preprocessing blant annet downcase og uten special symbols.

For å oppnå **preprosess\_downcase(sentences)** funksjonen brukte vi **lower()** funksjonen for å gjøre om alle bokstavene til små bokstaver.

Når det gjelder **preprocess\_special\_symbosl(sentences)** så benyttet vi oss av regulære uttrykk for å ta med bare ordene, ikke noen spesiale symboler som punktum, eller noen tall.

**Oppgave 1b**

Her laget vi 3 variabler start\_a, end\_a, start\_b som styrer posisjonen til elementer i listen. For å oppnå dette brukte vi en if test som forebygger indeks error hvis start\_b er mindre enn 0.

Først tar vi elementer før «mål»-ordet, vi tar delen av listen som vi er interesert i, dvs. fra element på indeks pos – size (vindu størrelse) til indeksen til vårt ord. Den andre delen av ord vi er interesert i er begrenset mellom indeksen start\_a og end\_b.

Her ser vi eksempel hvor pos = 4 og size = 3.



**start\_b pos start\_a end\_a**

**Oppgave 1c**

I denne oppgaven skrev vi ferdig fit funksjonen **fit(self, setences)**

Vi begnte med å lage en ordbok som heter dict{}. Der brukte vi en for-løkke for å telle ord. Ordet legges i ordboken som nøkkel, mens verdi er satt til 1. Hvis ordet allerede eksisterer i ordboken så økes verdi med 1.

Etterpå sorterer vi ordboken, den transformeres til ei liste, og reverserer den. Slik får vi de mest frekvente ordene først.

Vi lagrer den begrenset antall av de mest frekvente ordene i et sett for å øke hastigheten av programmet.

Så lagret vi de som nøkkler i matrisen, mens verdien satte vi til et tom ordbok.

Så gikk vi til utfylling av matrisen. Vi kjørte for-løkker gjennom alle setninger, deretter alle ord og kalte på context\_window på ordet hvis den var en del av de mest frekvente. Så lagrer vi de kontekst ordene i matrisen, så lenge de også er en del av de mest frekvente.

I tillegg endret vi fit til å sjekke om vektoriseren er opprettet med normalize=True. Hvis det stemmer da blir **normalize\_vectors()** kalt. (Oppgave 2d)

**Oppgave 2a**

I denne delen fylte vi ut **vector\_norm()** funksjonen for å regne ut vektorlengden.

Vi summerer v^2 for hver verdi i vektoren i en enkel for-løkke.

Til slutt, for å få den faktiske vektorlengden, funksjonen returner en kvadrat rot av resultaten fra for løkken.

**Oppgave 2b**

Her fullførte vi funksjonen **euclidian\_distance()** som regner ut distansen mellom to ord.

Først begynte vi ved å lage to variabler p og q som holder vektorene for ord1 og ord2.

Først:

Vi brukte for-løkker som går gjennom nøkler i p.

Hvis nøkkelen fantes i q.keys() da ble p.value – q[key] beregnet og addert til den totalet summen. Hvis den ikke fantes da brukte vi 0 istedenfor q[key].

Deretter benyttet vi oss av de samme for-løkker til å beregne summen av ord som ikke fantes i p, men som var med i q, ved å bruke 0 istedenfor p[key].

Til slutt funksjonen returnerer kvadratroten av eu\_dist, altså den totale summen.

**Oppgave 2c**

I denne oppgaven implementerte vi **cosine\_similarity().**

For å oppnå dette går vi frem på samme måte som i euclidean\_distance(). Forskjellen er at vi ganger verdier sammen istedenfor å opphøyde de i to. Den andre forskjellen er at vi ikke bryr oss om 0 verdier, da de allikevel hadde vært lik null.

Funksjonen ble endret til å sjekke hvis vektorene er normalisert til å da beregne bare dott produktet, ellers returnere funksjonen dott produktet delt på lengde \* lengde av vektorene.

**Oppgave 2d**

I denne oppgaven skrev vi ferdig funksjonen **normalize\_vectors().**

For å oppnå dette brukte vi en for-løkke som går over elementer i matrisen, over alle verdier, og deler verdiene til ord i et setning på lengde av vektoren av gitte setning.

**Oppgave 2e**

I denne delen gjorde vi ferdig funksjonen **nearest\_neighbors().**

Vi brukte **cosine\_similarity()** og **sorted()** funksjonen for å utføre sortering.

Vi lagde ei sortert liste av ord og tilsvarende avstand til «mål»-ordet i ei liste, deretter reverserte vi den til å ha de høyeste verdiene først. Så returnerer vi de k første som brukeren vil.

**Oppgave 3**

