IN2110 våren 2020 – obligatorisk innlevering 1a

*Leveres innen **fredag 21. februar kl. 23.59** i Devilry.*

Det er en god idé å lese gjennom hele oppgavesettet før du setter i gang. Har du spørsmål så gå på gruppetime, spør på Piazza eller send epost til `in2110-hjelp@ifi.uio.no`.

Oppsett

For IN2110 har vi laget et utviklingsmiljø for Python som inneholder programvare og data for obligene. For mer informasjon om bruk av miljøet på IFI's Linux maskiner eller via SSH, eller for installasjon på egen maskin, se;
`https://github.uio.no/IN2110/in2110-shared`

I oppgavene under skal dere bygge videre på pre-kode som er tilgjengelig under mappen `obliger/1a/` i kurs-repoet for gruppetimer og obliger:
`https://github.uio.no/IN2110/in2110-lab/`

Bakgrunn

For denne innleveringen skal vi jobbe med datasettet *Norwegian Review Corpus* (NoReC) som består av anmeldelser hentet fra en rekke norske nettaviser. NoReC består av over 35.000 dokumenter fordelt på 9 tematiske kategorier. Vi skal kun se på 3 av disse kategoriene: 'games', 'restaurants' og 'literature'. I oppgavene som følger skal vi skal jobbe med å (*i*) pre-prosessere tekstene, (*ii*) representere dem i en vektorrom-model og så (*iii*) lage en klassifikator for å predikere hvilken kategori en gitt anmeldelse tilhører.

Når vi jobber med klassifikasjon er det viktig at vi setter til side en del av dokumentene slik at vi kan bruke disse til å *evaluere* klassifikatoren. NoReC er delt i tre deler: 'train', 'dev' og 'test'. Det er god praksis å trene med treningssettet ('train'), evaluere underveis med valideringssettet ('dev' for *development*) og spare testsettet ('test') helt til slutt. Utviklingsmiljøet inneholder funksjonene `norec.train_set()`, `norec.dev_set()` og `norec.test_set()` som dere kan bruke for å hente ut dataene. Disse funksjonene returnerer en iterator over dokumenter i NoReC. Hvert dokument er et objekt som inneholder attributtene `text` og `metadata` som inneholder henholdsvis dokumentteksten og en dictionary med metadata for dokumentet.

Innleveringsformat

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.

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

Tabell 1: Eksempel på tabell

Oppgave 1 Data og pre-prosessering

a) Data

I pre-koden er det en funksjon kalt `prepare_data()`. Denne tar inn en iterator over dokumenter og skal returnere to lister: en liste med dokument-tekstene og en liste med den respektive kategorien for hvert av dokumentene.

> Skriv ferdig `prepare_data()`. Husk at vi kun ønsker dokumentene i kategoriene 'games', 'restaurants' og 'literature'.

b) Pre-prosessering

Pre-koden inneholder funksjonen `tokenize()` som splitter på mellomrom. Vi ønsker en bedre tokenisering av dokumentene. Om du gikk glipp av første gruppetime eller trenger en oppfriskning på tokenisering, se mer informasjon på: `https://github.uio.no/IN2110/in2110-lab/tree/master/lab1`

- Tokeniser trenings-settet med den originale funksjonen og rapporter hvor mange tokens og hvor mange ordtyper du får.
- Endre funksjonen til å bruke en bedre tokenizer, f.eks. `word_tokenize` i NLTK. Rapporter antall tokens og ordtyper for denne også.
- Prøv andre typer pre-prosessering, som f.eks. å gjøre om alle ord til små bokstaver, normalisering av tall, eller normalisering av tokens som betyr det samme (som f.eks. forskjellige typer hermetegn). Rapporter antall tokens og ordtyper.
- Hvilken tokenisering gir lavest antall ordtyper?

c) Statistikk

- Beregn antall dokumenter per kategori ('games', 'restaurants' og 'literature') i trenings-settet.
- Diskuter kort fordelingen mellom kategoriene.

Oppgave 2 Dokumentrepresentasjon

Vi ønsker å representere dokumentene som vektorer som vi så kan bruke som input til å trene en klassifikator som lar oss predikere hvilken kategori et gitt dokument tilhører. For å gjøre dette skal vi bruke *scikit-learn*. De klassene og funksjonene dere trenger er allerede importert i pre-koden. Bruk gjerne litt tid på å se på veiledningene på `http://scikit-learn.org`. I denne seksjonen skal vi jobbe med å lese inn dokumentene og lage vektorrepresentasjoner, så skal vi jobbe med selve klassifikatoren i neste seksjon.

a) Vektorisering

I `Vectorizer` fra prekoden skal vi bruke `CountVectorizer` fra scikit-learn for å lage *bag-of-words*-representasjoner av dokumentene. Den tar inn en iterator over dokumenter og returnerer en dokumentvektor for hvert dokument. Fordi ordforrådet er så stort – over 500 000 ordtyper for hele NoReC – blir hver enkelt dokumentvektor 500 000-dimensjonale hvis vi bruker alle ordene. Vi kommer til å begrense oss til de 5000 mest frekvente ordene. Dette reduserer både minnebruk og kjøretid.

`CountVectorizer` tar veldig mange argumenter, men dere kan ignorere de fleste av dem; de eneste som er viktige for oss er `lowercase`, `tokenizer` og `max_features`. Vi ønsker å bruke vår egen tokenizer, ikke den innebygde i `CountVectorizer`, og vi ønsker heller ikke å la den gjøre om til *lowercase* – vi kan heller gjøre om til små bokstaver i `tokenize()` om vi skulle ønske det. Bruk argumentet `max_features` til å begrense ordforrådet. For å kunne ta i bruk `CountVectorizer` må vi først identifisere ordtypene som skal inngå i ordforrådet / vokabulæret til modellen. Det kan vi gjøre med metoden `fit()`. Det er viktig at vi kun bruker trenings-settet for dette. Etter vi har identifisert vokabulæret kan vi anvende vektorisereren med `transform()`; dette vil altså opprette dokumentvektorene. For å spare litt kjøretid kan vi ev. bruke `fit_transform()` som gjør begge operasjoner i ett.

> Endre pre-koden slik at vi kan bruke `Vectorizer.vec_train()` for å lage dokumentvektorer for treningsdataene, og `Vectorizer.vec_test()` for å lage vektorer for 'nye' dokumenter i validerings- og test-dataene. Dere må gjøre endringer i konstruktøren, `vec_train()` og `vec_test()`. (Variabelen `vec_tfidf` kan ignoreres for nå.)

b) Visualisering

Om du kjører miljøet på egen maskin vil visualiseringen vises i eget grafisk vindu. Om du kjører via SSH vil grafen lagres som en PNG-fil. Du kan også bruke `ssh -Y` *for å få grafisk grensesnitt over SSH, hvis du er dreven på Linux. Du kan lagre grafen til disk fra det grafiske grensesnittet ved å trykke på diskett-ikonet.*

Nå har vi vektorisert dokumentene, men vi vet ikke om disse vektor-representasjonene er gode eller ikke. For å få litt mer innsikt skal vi visualisere vektorene i treningssettet. Vi har laget en ferdig funksjon som dere kan bruke:

```
>>> from in2110.oblig1 import scatter_plot
>>> scatter_plot(vectors, labels)
```

hvor `vectors` er de vektoriserte dokumentene og `labels` er listen over kategorien til hvert av dokumentene. Visualiseringen viser en prikk for hvert dokument, med farge avhengig av hvilken kategori det tilhører. Ideelt sett skulle vi sett at hver kategori var en helt separat klynge, men i praksis er det alltid noe overlapp mellom kategorier.

> Visualiser dokumentvektorene for trenings-settet. Lagre grafen som en fil og legg ved i rapporten. Beskriv og diskuter hva du ser.

c) Vekting

For å få bedre representasjoner kan det være lurt å vekte trekkene i ordvektorene, slik at de trekkene som er mer informative gis høyere vekt. Vi skal se på en type vekting kalt *term frequency–inverse document frequency (tf-idf)*. Scikit-learn har en innebygd klasse for tf-idf kalt `TfidfTransformer` som vi skal bruke. Den tar dokumentvektorer som input og gir ut nye vektorer som output. I likhet med `CountVectorizer` må denne tilpasses trenings-settet med `fit()` eller `fit_transform()` før vi kan ta den i bruk (fordi vi først må gjøre de nødvendige frekvenstellingene i korpuset).

- Legg til tf-idf vekting i `Vectorizer`. `vec_train()` og `vec_test()` skal returnere vektorer med og uten tf-idf.
- Visualiser trenings-vektorene som er vektet med tf-idf. Ser du noen forskjell mellom vektorene med og uten tf-idf?

Oppgave 3 Klassifisering

Vi skal bruke k-NN for å predikere hvilken kategori dokumentene i valideringssettet og testsettet tilhører ved hjelp av `KNeighborsClassifier` fra scikit-learn. Klassifikatoren trenes med `fit()` på treningsdataene og kan brukes for å predikere med `predict()` på testdata. For best resultat er det viktig at trenings- og

test-dataene er pre-prosessert på samme måte. F.eks. vil en klassifikator trent uten tf-idf gi dårlige resultater for dokumentvektorer med tf-idf. Ved å endre på verdien av k kan man tilpasse klassifikatoren. Avhengig av datasett, pre-prosessering og vekting vil forskjellige verdier av k kunne gi forskjeller i ytelse.

a) k-NN

Gjør ferdig `create_knn_classifier()`. Den skal ta tre argumenter: en liste med dokumentvektorer, en liste med kategorier for dokumentene og verdien av k. Funksjonen skal returnere klassifikatoren.

b) Evaluering

Et vanlig brukt mål får å evaluere en klassifikator er *accuracy*. Vi kan beregne dette ved hjelp av funksjonen `accuracy_score()` fra scikit-learn. Den tar to argumenter: en liste med *sanne merkelapper* og en med *predikerte merkelapper*. De sanne merkelappene er listen over kategorier for hvert dokument som vi fikk fra NoReC (altså såkalte 'gull-standard' merkelapper), mens de predikerte er modellens egne som vi får fra `predict()`. Ved å evaluere prediksjonene på valideringssettet kan vi finne den verdien av k som gir best resultat.

- Lag dokumentvektorer for dokumentene i valideringssettet.
- Tren en klassifikator for forskjellige verdier av k med og uten tf-idf, og beregn accuracy for hver av dem. Du kan f.eks teste med k fra 1 til 20. Accuracy vil ofte øke i starten, men så avta etterhvert som k øker.
- Inkluder en tabell med resultetene i rapporten.
- Diskuter resultatene dine. Hvilken kombinasjon av vekting og k-verdi gir best accuracy?

c) Testing

Vi har tilpasset parameterene til klassifikatoren vår til *valideringss*settet og nå ønsker vi å gjøre en siste evaluering på *test*settet. For å få mest mulig realistiske og representative resultater er det viktig at vi venter med dette helt til slutt, slik at vi ikke tilpasser klassifikatoren testsettet. Fordi vi vil teste hvor bra klassifikatoren vår kan generalisere til nye data må vi være nøye på at det reserverte testsettet er nettopp dette – nye data som ikke har påvirket de valgene vi har tatt for klassifikatoren vår.

- Lag dokumentvektorer for testsettet.
- Tren en klassifikator med samme valg for k-verdi og vekting (tf-idf eller ikke) som den med høyest accuracy i forrige deloppgave.
- Hvordan er ytelsen på testsettet sammenlignet valideringssettet?