ML-Innlevering
Dat158

Sentiment analyse Filmanmeldelser

Marius V.Pedersen, Dato: 02.11.2025

Introduksjon og Problemstilling

Viktig: vennligst les .ipynb filen, den inneholder steg for steg av trening og analysering

I dette prosjektet lages en modell som kan automatisk analysere sentimentet basert på tekstbaserte filmanmeldelser. Dette tillater oss å veie om en filmanmeldelse er utrykt positiv eller negativ og gjør det mulig å kategorisere anmeldelser etter sentiment. Prosjektet er inspirert av et github-repo som jeg diskuterte med en venn.

Lenke Github-repo inspirasjon: Github-Repo-Inspo

Modellen skal trenes på et IMBD-datasett, som består av 50 000 anmeldelser med tilhørende sentiment etiketter (positivt eller negativ). Resultatet skal bli «*deployed*» gjennom en webapplikasjon som tillater brukere å veie tekstbaserte filmanmeldelser. Men riktige og tilpassede datasett kan modellen også trenes og brukes til kategorisering av sentiment i lignende relevante oppgaver.

Maskinlæring er en gunstig og skalerbar måte å løse problemet på, siden komplekse språklige mønstre er vanskelige å beskrive med manuelle regler. Maskinlæring tillater modeller å lære fra store mengder data, som tillater gjenkjenning av mønstre, ordbruk og tone.

Sentiment analyser kan gjennomføres på mange forskjellige måter. Det finnes bedre modeller som vil gi bedre resultater og kan se mer kontekst en hva denne modellen vil. En transformer modell vil for eksempel gi mer kontekst generelt som gjør at man vil få bedre svar, men er også mye mer avansert å bruke og sette opp.

Mål:

- Målet er å kunne klassifisere filmanmeldelser og gi en predikasjon på om de er positive eller negative

Løsning inneholder:

- API- grensesnitt: Via node.js og Express
- Frontend (Html, js og css)
- .py (python): fil som inneholder konvertering av json objekter til og lasting og bruk selve ml modellen som er ferdigtrent og skal brukes
- .joblib (biblotek for lagring av modell og vektorisering)
- .ipynb (jupiter notebook) dokument som går gjennom oppsett og trening av modellen

ML-Innlevering
Dat158

Metrikker

For å måle modellens ytelse bruker vi følgende ml-metrikker

- Accuracy
- Precision/Recall/ F1-score
- Confusion matrix

Burde kanskje blitt med:

- Brukeropplevelse
- Latency

Data

I dette prosjektet bruker vi et offentlig datasett som er hentet fra huggingface: https://huggingface.co/datasets/stanfordnlp/imdb

Datasettet består av 50 000 engelskspråklige filmomtalelser som er merket med et sentiment label. Data er delt opp i text og label, hvor text er en tekststreng mens label er en binær etikett, 0 (negativ) eller 1 (positiv).

Eksempel på data:

{

"label": 0,

"text": "Goodbye world2\n"

}

Datasettet er lastes direkte gjennom hugging face dataset-biblotek som vist i .ipynb filen Dette tillater automatisk håndtering av nedlastning og strukturering.

Datamengde og bruk

Datasettet er på forhånd delt inn i trening og testsett, men inneholder også usupererviset data som kan brukes:

- 25 000 anmeldelser for trening
- 25 000 for testing
- 50 000 unsupervised (Ikke brukt)

I prosjektet ble 20% av testsettet brukt som validering, slik at de endelige delene som ble brukt:

- Trening: 25 000
- Test: 20 000
- Validering: 5000

ML-Innlevering
Dat158

Dataprosessering

Scikit-learn forventer lister av tekster (X) og lister av labels (Y), så vi pakker derfor ut text og labels fra datastrukturen til Hugging Face:

Eksempel: (Vist i .pynb)

X_train_texts = raw_train["text"]

y_train_labels = raw_train["label"]

Tallrepresentasjon

For at teksten skal kunne brukes i maskinlæring må den konverteres numeriske verdier. Dette ble gjort ved hjelp av TF-IDF-vektorisering(Term Frequency – Inverse Document Frequency), implementert med TfidfVectorizer fra scikit-learn

Denne modellen beregner en vekt for hvert ord eller n-gram basert på

- Hvor ofte et ord forekommer i dokumentet (TF)
- Hvor unikt ordet er i alle tekstene

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

TF-IDF kombinerer de to slik at vanlige ord vektes lavt, og sjeldne (men viktige) ord vektes høyt

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

Resultatet er en vektor per dokument som beskriver hvilke ord eller utrykk som kjennetegner teksten. Sluttresultatet er en numerisk matrise hvor hver rad representerer en anmeldelse, og hver kolonne er et ord eller frase (ngram_range)

Parameter brukt i TF-IDF vektoriseringen

- Max_featurres: Begrenser hvor mange ord og utrykk som skal tas med i ordforrådet
- Ngram_range: bestemmer antall ord per feature
- Stop_word: Forhåndsdefinert liste med ord som ikke har verdi
- Lowercase: funksjon for å konvertere tekst til små bokstaver

Modell og trening

Vi har brukt TF-IDF vektorisering for å representere teksten som numeriske trekk

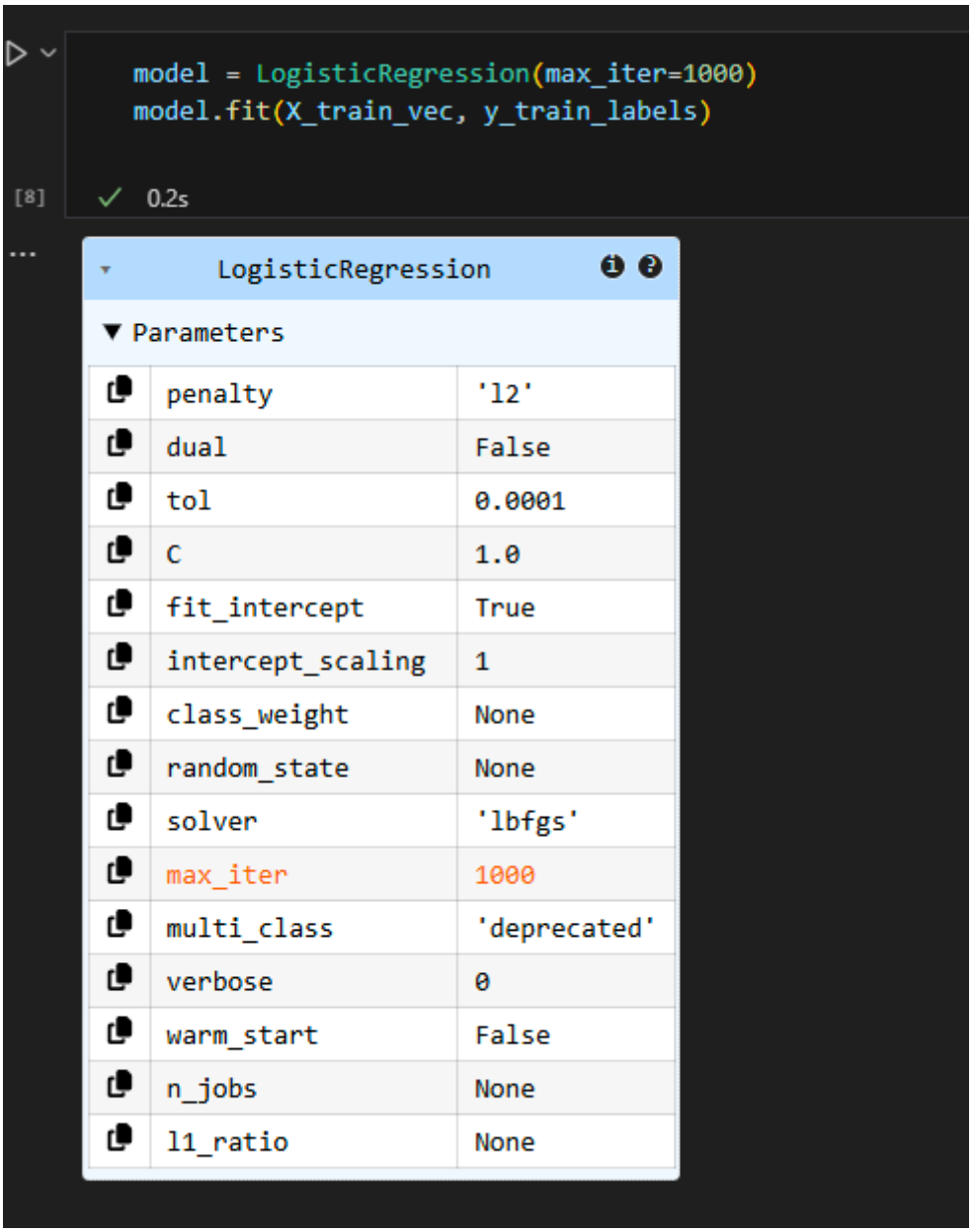
Vi bruker logistisk regresjon som klassifikasjonsmodell. Logisk regresjon passer bra til å predikere et utfall som kun har to verdier, som for eksempel positivt og negativt. Passer også godt til TF-IDF matrisen som er lineære vektorer. Logistisk regresjon estimerer sannsynligheten for at et gitt eksempel tilhører klassen (positiv/negativ), og gir oss en sannsynlighet som ligger mellom 0-1.

- Hvis resultat > 0.5 -> positiv
- Hvis resultat < 0.5 -> negativt

Baseline:

- Basert på prosjektet vil vi få en «naiv» baseline på 50% korrekte siden datasettet er balansert (50% positive og 50% negative).
- Basert på github-repo som inspirerte denne testen som er en transformer-modell ligger baselinen på 84% riktige. Hvis tolket riktig tar dette til rette for klassifiseringen og ikke selvsikkerheten til modellen. Ut ifra andre tester jeg så og kanskje tolket riktig var selvsikkerhet til denne modellen mye mer sikker når den gjettet.

Modellen ble trent på trening settet bestående av 25 000 anmeldelser.



Validerings settet tillater at man unngår å bruke testsettet til «tuning». Ikke brukt her men brukt med forbehold til videre utvikling og testing.

```
val_pred = model.predict(X_val_vec)
val_acc = accuracy_score(y_val_labels, val_pred)

print("Validation accuracy:", val_acc)
print(classification_report(y_val_labels, val_pred, target_names=["neg", "pos"]))
```

✓ 0.3s Python

```
Validation accuracy: 0.888
              precision    recall  f1-score   support

         neg       0.90      0.88      0.89      2513
         pos       0.88      0.90      0.89      2487

    accuracy                           0.89      5000
   macro avg       0.89      0.89      0.89      5000
weighted avg       0.89      0.89      0.89      5000
```

Endelig test på test settet hvor vi klarte å gjette riktig på ca 88% av tilfellene

```
test_pred = model.predict(X_test_vec)
test_acc = accuracy_score(y_test_labels, test_pred)

print("Test accuracy:", test_acc)
print(classification_report(y_test_labels, test_pred, target_names=["neg", "pos"]))
```

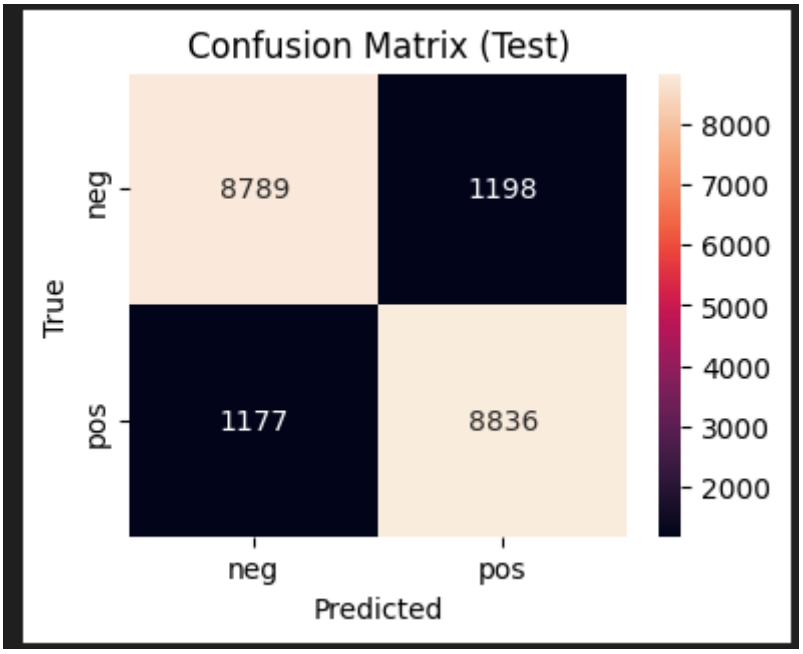
✓ 1.4s Python

```
Test accuracy: 0.88125
              precision    recall  f1-score   support

         neg       0.88      0.88      0.88      9987
         pos       0.88      0.88      0.88     10013

    accuracy                           0.88     20000
   macro avg       0.88      0.88      0.88     20000
weighted avg       0.88      0.88      0.88     20000
```

En Confusion Matrix som viser hvor ofte positive og negative tekster ble feilklassifisert. I dette tilfellet var true positives og true negativs relativt jevnfordelt som betyr at modellen ikke er kraftig skjev mot en klasse, men kan være litt usikker på midt på treet-omtaler.



Ved å teste enkelt tilfeller kan vi se at modellen ikke er veldig flink til å tolke:

- Blandende sentiment, både bra og dårlig i samme anmeldelse
- Høflige negativ kritikk og sarkasme
- Menneskelig tale som «ønsker» som kan være negativt eller positivt ladet, men språklig forsiktig

```
Review: I absolutely loved this movie. It made me cry in a good way.
 → Positive (0.5304902815985154)

Review: What a waste of time. Boring, predictable, badly acted.
 → Positive (0.532291157301089)

Review: It was okay I guess, not terrible but not great either. i wish it had more action
 → Positive (0.7184250872334396)


Review: I absolutely loved this movie. It made me cry in a good way.
 P(neg)=0.470, P(pos)=0.530

Review: What a waste of time. Boring, predictable, badly acted.
 P(neg)=0.468, P(pos)=0.532

Review: It was okay I guess, not terrible but not great either. i wish it had more action
 P(neg)=0.282, P(pos)=0.718
```

Deployment

Etter modellen ble ferdig trent og evaluert, ble det gjort klar for praktisk bruk ved å lagre den som to .joblib filer:

- tfidf_vectorizer.joblib: inneholder ordforråd og TF–IDF-vekter
- sentiment_model.joblib: inneholder den ferdig trente logistiske regresjonsmodellen

Systemet består av tre lag:

1. Modell-api i python
2. web-server med Node.js (express)
3. Frontend(HTML,CSS , JS)

Predikamentene kan brukes til å:

- Klassifisere filmanmeldelser som positiv eller negativ og kanskje nøytral
- Gi et terningkast (Vil ikke være relevant, men kan være gøy å legge til)
- Kan brukes til filtrering av anmeldelser

Videre arbeid:

- Re-trene og teste om man kan lage en bedre modell med å blant annet teste:
    - Ordvekter: (inspisere hvilke ord og utrykk som strekt dytter modellen
    - n-gram: teste rekkevidde og lengre fraser om det gir bedre resultater
    - endringer av reguleringsstyrke C:
    - justere beslutning terskelen: 0.5 passer bra til positiv/negativ, øking gjør ikke modellen bedre men kan gir mer nyttig og presis data
- Rensing av data: Nå blir deler av prosessering gjort i parameter som eksempel: tolowercase(). Dataprosessering kan bli bedre selv om det ser relativt greit ut nå: eksempelvis: didn't blir didn.
- Nettsiden tar nå inputs og vurderer om de positiv negativ eller nøytrale, dette gir mange nøytrale anmeldelser selv om de egentlig vekter mot en grad. Selve predikamentet er riktig (positiv/negativ) men modellen er ikke god nok til å vekte «karakter/terningkast», må i så fall jobbes videre med.

Referanser:

Chatgpt: Brukt til beskrivelse av konsepter og kodegenerering

Inspirasjon til oppgave:
https://github.com/LH01t/inf265_project_3/blob/main/01_encoder_sentiment_classifier/imdb_sentiment_encoder.ipynb

Datasett: https://huggingface.co/datasets/stanfordnlp/imdb