Emotion classification

Vegard Aa Albretsen, Erlend Vitsø - 15 nov

BESKRIV PROBLEMET

SCOPE

Prosjektets mål er å utvikle en maskinlæringsmodell som kan analysere tekstdata og klassifisere hvilken av de seks følelsene (sinne, frykt, glede, kjærlighet, trist og overrasket) som uttrykkes.

Denne modellen kan gi verdifull innsikt i hvordan brukere opplever og responderer på innhold, noe som for eksempel kan forbedre kundetilfredshet, optimalisere innholdstilpasning og støtte beslutningsprosesser i markedsføring og kundeservice ved å tilpasse responsen til den emosjonelle tonen i brukerens kommunikasjon.

Løsningen kan brukes til å analysere tilbakemeldinger fra kunder om et produkt eller tjeneste for å avgjøre hvilken følelse som uttrykkes i teksten. Dette gir kundebehandlere innsikt i kundens emosjonelle tilstand, noe som kan være avgjørende for kundetilfredshet og tidlig identifisering av problemer. Med en klarere forståelse av følelser som frustrasjon, glede eller forvirring kan organisasjonen tilpasse og forbedre kundekommunikasjonen og produkttilbudet.

I tillegg kan modellen bidra til å forbedre kundebehandleres svar ved å gi dem tilbakemeldinger på tonen i kommunikasjonen de sender ut. Dette vil bidra til at kundebehandlere kommuniserer på en måte som oppleves som profesjonell, passende, og med riktig følelsesmessig innstilling. For eksempel kan det sikre at svar på negative tilbakemeldinger oppleves som empatiske, mens positive tilbakemeldinger kan bli møtt med entusiasme.

Ved å løse problemet manuelt krever dette en god del arbeid, og ved behov innsikt fra flere personer.

Business metrics

Ytelsen til en maskinlæringsmodell for å hente følelser ut fra tekst kan måles ved å bruke følgende business metrics:

- **Customer Satisfaction Score (CSAT):** Måler kundetilfredshet før og etter implementeringen av modellen.
- **Net Promoter Score (NPS):** Vurderer hvor sannsynlig det er at kunder vil anbefale tjenesten til andre.
- **Churn Rate:** Overvåker om modellen bidrar til å redusere frafall av kunder.
- **Conversion Rate:** Ser på om modellen fører til høyere konverteringsrater, f.eks. flere salg eller registreringer.
- **Operational Efficiency:** Måler hvor mye tid og ressurser som spares ved å bruke modellen i stedet for manuelle prosesser.

Pipeline

Data Ingestion: Rådata fra innkommende e-poster blir samlet inn og strukturert. Dette inkluderer metadata (avsender, tidspunkt, emne) og selve innholdet i e-posten.

Forbehandling av data: Innholdet i e-postene gjennomgår tekstbearbeiding, som staving- og grammatikkorrigering, stemming, og fjerning av støy. Dette sikrer at dataene er klare for analyse og modellprediksjon.

Klassifisering og prediksjon (Maskinlæringsmodellen): Maskinlæringsmodellen klassifiserer e-poster basert på innhold, for eksempel etter prioritet, kategori (f.eks. teknisk støtte, fakturaspørsmål) eller sentimentanalyse (positiv, nøytral, negativ). Modellen kan også generere forslag til svar eller rute e-posten til riktig avdeling.

Integrasjon med e-postplattformen: Resultatene fra modellen integreres direkte i plattformen, hvor anbefalte handlinger (f.eks. automatiske svar eller flagging av viktige saker) vises til kundebehandlere i sanntid.

Kontinuerlig læring og oppdatering: Modellen oppdateres kontinuerlig basert på tilbakemeldinger fra kundebehandlere og resultater fra tidligere prediksjoner, noe som sikrer bedre ytelse over tid.

Stakeholders

1. Kundene (sluttbrukerne):
Kundene er hovedstakeholderne i prosjektet og består av sluttbrukerne som vil dra nytte av systemet. For en løsning som integreres i en e-postplattform for kundebehandling, inkluderer dette:

- **Kundeservicemedarbeidere**: De bruker systemet daglig for å håndtere forespørsler, noe som krever en intuitiv og pålitelig løsning. Modellen må levere presise klassifiseringer og anbefalinger for å forbedre deres arbeidshverdag.
- **Endekundene**: Disse er kundene til virksomheten som benytter e-postplattformen. Dedrar nytte av raskere, mer relevante svar og forbedret opplevelse.

2. Utviklingsteamet:
Dette teamet har ansvar for å implementere, teste og vedlikeholde systemet. Deres oppgaver inkluderer:

- Integrasjon av modellen på e-postplattformen.
- Optimalisering av ytelsen for å sikre rask respons og høy skalerbarhet.
- Overvåking og løsning av tekniske utfordringer som oppstår i produksjon.

3. Produktledere:
Produktlederne er ansvarlige for å definere prosjektets funksjonalitet og prioriteringer. Deres oppgaver inkluderer:

- Identifisere forretningsbehov og brukerkrav.
- Prioritere funksjoner som gir størst verdi for kundene.
- Koordinere mellom ulike team, inkludert utvikling, datavitenskap og ledelse.

4. Data Scientists:
Data Scientists spiller en nøkkelrolle i utviklingen og finjusteringen av maskinlæringsmodellen. Deres ansvar inkluderer:

- Samling og behandling av data som brukes til å trene modellen.
- Eksperimentering med ulike algoritmer og parameterjusteringer for å forbedre modellens ytelse.
- Evaluering av modellens nøyaktighet og effekt i praktiske scenarioer.
- Sikre kontinuerlig læring ved å oppdatere modellen med nye data.

5. Ledelsen:
Ledelsen har en strategisk rolle og bruker prosjektet for å oppnå organisatoriske mål. De er ansvarlige for:

- Ressurstildeling, inkludert budsjetter, personell og teknologier.

- Evaluere prosjektets innvirkning på selskapets mål, som kostnadsreduksjon, økt effektivitet eller kundetilfredshet.
- Beslutte fremtidig retning for prosjektet basert på suksess og tilbakemeldinger.

Tidslinje for prosjektet

Vi startet å se på prosjektet mandag den 21.oktober 2024 og hadde innleveringsfrist 15.november 2024. Tiden vil dermed omtales som uke 1-4 istedenfor datoer.

Vi regner (for enkelhets skyld) at en arbeidsuke er 36 timer, og at ⅓ skal brukes på DAT158. Dette blir dermed 12 timer. Så trekker vi fra 4 timer forelesninger per uke, og ender opp med 8 timer per uke per person. Når dette da også har gått parallellt med algoritme-modul 3, så er det åpenbart at vi har vært travel for å prøve å få lagt ned nok arbeid i dette prosjektet, noe vi ikke har fått til. Vi skulle nok ideelt sett hatt hele arbeidsuker og lengre tid for å få et ønsket resultat.

Grunnet HV-øvelse for et av medlemmene i uke 3 ender vi på 8*4 + 8*3 (timer*uker) = 56 totale arbeidstimer prosjektert.

Da ble tidslinjen omtrent slik:

1. **Prosjektplanlegging og databehandling:**

   Initielt måtte vi finne ut hva vi ville gjøre. Vårt første ønske var å gjøre bildegjenkjenning, men ettersom bare nedlastingen av alle bildene var mangfoldige gigabyte, noe som tok lang tid bare å laste ned, konkluderte vi med at vi ville få mer læringsutbytte ved å ta noe “simplere” sånn at vi fikk brukt mer tid på å finjustere.

   Det tok omtrentlig hele uke 1 før vi fikk bestemt oss for prosjektrammene og datasettet.

2. **Modellutvikling**

   Vi tenkte å finne baseline for å finne gode modeller og deretter forbedre de beste modellene. Dette står mer om i eget avsnitt.

   Dette brukte vi uke 2 og 3 på.

3. **Deployering og evaluering**

Vi startet i uke 4 med å deployere prosjektet lokalt med streamlit. Dette står også i eget avsnitt. Her synes det kjapt at modellen, selv om den hadde 90% accuracy, precision og recall, ikke gjorde det så godt i deployering. Vi fikk problemer med teksthåndteringen for brukerinput, og fikk ikke streamlit til å hente tokenizeringsmappene og ble derfor nødt til å bruke en modell som er trent på raw tekst for selve deployeringen.

Programvare og Biblioteker

- **Python:** Programmeringsspråket som brukes for utvikling og modellering.
- **Biblioteker:**
    - **Scikit-Learn:** For klassiske maskinlæringsmodeller som logistisk regresjon og SVC.
    - **NLTK eller SpaCy:** For naturlig språkprosessering, inkludert tokenisering og stemming.
    - **Pandas, MatplotLib, Seaborn & NumPy:** For datahåndtering og analyse.
    - **Streamlit:** For å bygge og implementere det interaktive brukergrensesnittet.

Personell og Kompetanse

- **Maskinlæringsstudenter**
    - Vi er to studenter i faget DAT158 uten andre forutsetninger enn det dataingeniør-studiet og faget har gitt oss.

METRIKKER

For at prosjektet skal kunne betegnes som vellykket, må systemet oppnå målbare resultater som reflekterer de overordnede forretningsmålene. Disse målene inkluderer forbedret effektivitet i kundebehandlingen, økt brukeropplevelse for både kundeservicemedarbeidere og sluttkunder, samt en merkbar reduksjon i svartid. For å måle suksess, kan vi sette minimumskrav til ytelse basert på maskinlærings- og software-metrikker:

1. Treffsikkerhet (Accuracy):

- ***Mål****: Systemet skal ha en treffsikkerhet på minst* ***85 %*** *for klassifisering av e-poster etter kategori (f.eks. teknisk støtte, fakturaspørsmål).*
- ***Forretningsrelevans****: En høy treffsikkerhet sikrer at kundebehandlere mottar riktige anbefalinger, noe som reduserer tid brukt på feilretting og øker effektiviteten.*

2. Precision og Recall:

- ***Precision****: Minimum* ***90 %****, for å sikre at anbefalte handlinger (f.eks. svar eller kategorisering) faktisk er relevante.*
- ***Recall****: Minimum* ***80 %****, for å sikre at systemet ikke overser relevante alternativer.*
- ***Forretningsrelevans****: En balanse mellom precision og recall gir en pålitelig løsning som både minimerer feilaktige anbefalinger og maksimerer de riktige, noe som bygger tillit hos brukerne.*

3. Gjennomsnittlig svartid (Latency):

- ***Mål**: Responsen fra systemet må være innen **500 ms** for 95 % av forespørslene.*
- ***Forretningsrelevans**: En rask responstid er avgjørende for å opprettholde en god brukeropplevelse og sikre at kundebehandlere kan håndtere forespørsler uten forsinkelser.*

6. Brukeropplevd kundetilfredshet (Customer Satisfaction):

- ***Mål**: En kundetilfredshetsvurdering på minst **4 av 5** basert på spørreundersøkelser blant sluttbrukerne.*
- ***Forretningsrelevans**: Direkte tilbakemeldinger fra brukere måler løsningens opplevde verdi og brukervennlighet.*

Sammenheng med forretningsmål

*Disse metrikker er direkte knyttet til prosjektets **business objectives**:*

1. ***Effektivitet i kundebehandling**: Høy treffsikkerhet og rask svartid gir kundebehandlere bedre støtte, noe som reduserer tid brukt på hver forespørsel.*
2. ***Brukeropplevelse**: Precision og recall sikrer at sluttbrukerne får relevante løsninger, mens lav latency og høy throughput bidrar til en sømløs opplevelse.*
3. ***Strategisk verdi**: Konsistent ytelse over tid (gjennom kontinuerlig læring og oppdatering) gir grunnlag for videre utvikling og skalerbarhet av systemet.*

Ved å oppnå disse minimumskravene, vil prosjektet ikke bare tilfredsstille tekniske spesifikasjoner, men også levere målbar verdi til kundene og bedriften som helhet.

DATA

Datasettet vi skal bruke er hentet fra Kaggle og hver enkelt dataregistrering er hentet fra X (tidligere Twitter).

Datasettet har 416809 ulike datapunkter, med denne fordelingen:

Feature id	Feature definisjon	Antall
0	sadness	121187
1	joy	141067

2	love	34554
3	anger	57317
4	fear	47712
5	surprise	14972

Datapreprossesering

Det er ingen datapunkter i datasettet som ikke inneholder noen verdi. Det gjør at vi slipper å ta hensyn til hva vi skal gjøre med ufullstendige data. Dette må vi derimot ta hensyn til ved deployeringen, dvs. input som den trente modellen skal predikere på.
Vi la merke til at datasettet var “class imbalanced” altså at det var vesentlig mye mer entries med target 0 enn andre. Modeller som Logistisk regresjon forsøker å maksimere sannsynligheten for riktig prediksjon basert på alle dataene. Når en klasse dominerer, vil modellen fokusere på å forbedre prediksjonene for den overrepresenterte klassen, på bekostning av den underrepresenterte. Derfor var det hensynsfullt å prøve å balansere datasettet. Vi valgte å Resample med en UnderSampler.
Det ser ut som at teksten i datasettet allerede er redusert til formen A-Z og kun små bokstaver, men for å være sikker så gjøres dette også før noe mer blir gjort.

```
import re
data['text'] = data['text'].apply(lambda x: re.sub(r'[^a-zA-Z\s]',
'', x).lower())
```

Videre så vil dataen ‘tokeniseres’ og stemmes. Tokenisering vil si at vi deler teksten opp i individuelle deler, ofte ord, for å gjøre det enklere å analysere hvert ord individuelt. Stemming fjerner bøyninger fra ord, slik at de blir til en slags stammeform. For eksempel blir "running", "ran", og "runs" alle til "run". Dette kan være litt røft, fordi stammer ikke alltid gir meningsfulle ord.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
data['tokens'] = data['text'].apply(lambda x: [stemmer.stem(word) for word
in word_tokenize(x)])
```

```
from sklearn.model_selection import train_test_split
# Del opp datasettet i trenings- og testdata
X_train_text, X_test_text, y_train, y_test =
train_test_split(data['text'], data['label'], test_size=0.2,
random_state=42)

train_data = pd.DataFrame({'text': X_train_text, 'label': y_train})
test_data = pd.DataFrame({'text': X_test_text, 'label': y_test})

train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)
```

Deretter bør man gjøre tokenene om til tall på en eller annen måte. To vanlige måter er “Bag-Of-Words”. Det vil se på hvert unike ord i teksten og gjøre setninger om til en vektor som sier noe om hvilke ord som er til stede i hvilke setning. Ulempen her er at den ikke fanger opp rekkefølge eller kontekst.
Den andre metoden jeg fant var “Term Frequenzy - Inverse Document Frequency (TF-IDF)”. TF-IDF justerer for hyppigheten av ord i en tekst og tar hensyn til hvor vanlig ordet er i hele datasettet. Dette gir høyere vekting til ord som er unike for en tekst og lavere vekting til vanlige ord som "er" eller "og".

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sett opp TF-IDF-omformer og tilpass på treningsdata
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train_text).toarray()
X_test = vectorizer.transform(X_test_text).toarray()
```

Modellen ble først trent på treningssettet etter å ha gjennomgått nødvendige forbehandlingstrinn for både data og målvariabel. Dette inkluderte blant annet vektlegging av klassene for å håndtere ubalanse i datasettet. I treningsfasen justerte vi hyperparametere for å optimalisere modellens ytelse.

Hyperparameter-tuning med GridSearch

Vi gjennomførte en GridSearch for å finne de mest optimale hyperparametrene for modellen. GridSearch er en metode der vi definerer et søkeområde for ulike hyperparametere og evaluerer modellen på tvers av alle kombinasjonene. Dette ble gjort ved bruk av krysvalidasjon

for å sikre at resultatene var robuste og generaliserbare. De beste kombinasjonene av hyperparametere ble deretter brukt i den endelige modellen.

Evaluering av modellen

Etter at modellen var trent med de optimale hyperparametrene, ble testsettet brukt til å evaluere ytelsen. Vi predikerte testsettet og sammenlignet de predikerte verdiene med de faktiske verdiene. For å kvantifisere ytelsen brukte vi **accuracy** som en overordnet måling.

For ytterligere innsikt i modellens ytelse, spesielt med tanke på hvordan den håndterte ulike klasser, beregnet vi en **confusion matrix**. Denne matrisen gir en detaljert oversikt over antallet sanne positive, sanne negative, falske positive og falske negative prediksjoner. Ved å analysere confusion matrix kan vi identifisere om modellen har bias mot en spesifikk klasse og vurdere behov for ytterligere forbedringer.

Ved å kombinere disse metodene sikret vi at modellen var så presis og robust som mulig i møte med både balanserte og ubalanserte datasett.

MODELLERING

Etter å ha gjort litt research på teksklassifiseringsmodeller og testing (ref. Tabell under) endte vi opp med å velge *Logistic Regression* og *Linear SVC*. Disse er relativt enkle modeller som skal gi solide resultater i tekstklassifisering (Géron, 2017).

	Linear SVC	Naive Bayes	Logistic Regression
Accuracy	0.899	0.81	0.895
Precision	0.899	0.836	0.895
Recall	0.899	0.81	0.895
F1 Score	0.899	0.817	0.895

Vi prøvde også å teste *Random Forest* og *K-Nearest Neighbors*, men etter at de hadde kjørt i 25 min droppet vi dem.

Feil-prediksjoner og Feature Importance

Ved feilprediksjoner er vi nødt til å analysere disse nærmere for å forstå hvilke følelser som forveksles mest, og deretter justere pre-prosessering eller modellarkitektur for å fokusere på disse kategoriene.

For å undersøke feature importance bruker vi Seaborn og Matplotlib til å visualisere hvilke ord eller fraser modellen tillegger mest vekt i klassifiseringen. Dette krever flere steg for å trekke ut og klargjøre funksjonene. Visualiseringene har allerede hjulpet oss med å identifisere mønstre og skjevheter i modellens beslutningsprosess, noe som vil være avgjørende for å forbedre nøyaktigheten i fremtidige iterasjoner.

DEPLOYMENT

I denne tidlige fasen av prosjektet har vi implementert et enkelt brukergrensesnitt ved hjelp av **Streamlit**, et Python-basert verktøy for rask utvikling av webapplikasjoner. Grensesnittet består av en **input-tekstboks** der brukeren kan skrive inn en tekst, og en **"Analyze Text"**-knapp som utløser modellens prediksjon. Når brukeren klikker på knappen, klassifiserer modellen teksten og presenterer en sortert prosentvis oversikt over sannsynligheten for hver av følelsene, der den mest sannsynlige følelsen vises øverst som modellens prediksjon.

Dette enkle oppsettet lar oss raskt teste og demonstrere modellens funksjonalitet, samt samle inn tilbakemeldinger fra testbrukere.

Hvordan prediksjonene brukes

Prediksjonene fra modellen brukes til å gi en klassifisering av den følelsesmessige tonen i teksten som brukeren skriver inn. Resultatene vises umiddelbart i Streamlit-grensesnittet, noe som gir brukeren en direkte opplevelse av modellens ytelse. I denne tidlige fasen er formålet primært å:

- **Teste modellens ytelse** på ulike typer input, spesielt tekster som avviker fra treningsdataene.
- **Avdekke svakheter** i modellen, som forveksling av følelser eller vektlegging av irrelevante ord.
- **Validere ideen** gjennom tilbakemeldinger fra brukere.

Monitorering og vedlikehold

Selv i denne tidlige fasen er det viktig å legge grunnlaget for monitorering og vedlikehold. Våre tiltak inkluderer:

1. **Logging av input og output**: Ved hjelp av Streamlit kan vi logge alle brukerinteraksjoner for senere analyse.
2. **Analyse av feilprediksjoner**: Identifisere hvilke typer input modellen har problemer med, og bruke dette som grunnlag for å forbedre preprosesseringen eller justere modellens arkitektur.
3. **Enkel evaluering**: Integrere evaluering av metrikker som treffsikkerhet (accuracy) direkte i grensesnittet, slik at vi kan måle ytelse på testdata parallelt med nye brukertester.

Planer for forbedring

Etter at modellen er testet gjennom grensesnittet, planlegger vi følgende forbedringer:

1. **Utvidelse av datasettet**: Legge til mer varierte tekster, inkludert større variasjon i casing og formuleringer, for å forbedre generaliseringsevnen.
2. **Finjustering av modellen**: Basert på feilprediksjoner fra Streamlit-grensesnittet, vil vi iterere over modellen og justere parametere eller arkitekturen.
3. **Utvidet funksjonalitet i Streamlit**:
    - Vise feature importance direkte i grensesnittet, slik at brukerne kan forstå hvilke ord som har påvirket prediksjonen.
    - Mulighet for brukere å gi tilbakemelding på modellens resultater, som kan brukes til videre forbedringer.
4. **Forberedelse for produksjon**: Når modellen er mer robust, vil vi utvide til en API-basert løsning som kan integreres i større systemer, med støtte for skalerbarhet og sanntidsbruk.

REFERANSER

Kaggle dataset: https://www.kaggle.com/datasets/nelgiriyewithana/emotions

Vi bruker generativ KI til prosjektet: https://www.myaistudentassistant.com/. Den har hjulpet oss med å finne idé til prosjekt.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (1st ed.). O'Reilly Media.