Multiklassificering af hadefulde ytringer med maskinlæring

Gustav Weber Kinch, Emma Rasmussen, Ask Harup Sejsbo & Joakim Hey Hinnerskov *

Roskilde Universitet
Gruppe nr. S2025625906
Anslag 133.176

Vejleder: Jens Ulrik Hansen

June 2, 2020

Abstract

This paper revolves around the development of an LSTM multiclass classifier, constructed using Keras as framework and CRISP-DM as project process, with the purpose of classifying natural language into varying degrees of toxicity. The model takes a starting point in an existing toxic comment classification challenge from Kaggle.com, and makes a first iteration, engineered towards the requirements in the challenge. In this first iteration, several measures are taken to avoid common pitfalls of neural networks. The model is then held up against principles of freedom of speech including The Harm Principle and The Offence Principle by John Stuart Mill and Joel Feinberg respectively. After evaluating upon the models performance in the light of these principles, a second iteration is constructed with some design changes. For reasons i.a. related to the dataset, this operation is less successful. The paper concludes that it is possible to make a good multiclassification tool for shallow NLP problems, but gets less efficient in later iterations as we try to apply it to more concrete purposes.

Keywords: *Natural Language Processing, LSTM, Hadefulde ytringer, Maskinlæring, Reccurent neural networks, Toxic comment classification*

*gwk@ruc.dk (64035), emmaras@ruc.dk (63801), harup@ruc.dk (64033), jheyh@ruc.dk (63759)

Indhold

Figurer

Tabeller

1 Indledning

Den offentlige diskurs bevæges i større omfang til online platforme, hvor sociale medier fungerer som kurator for den debat, der føres (Gillespie, 2010). Effekten af denne bevægelse medgiver et ansvar til sociale medier, som skal facilitere et forum, hvori diskursen kan føres inden for loven og etiske rammer. Vi ser dog en ændret attitude, når der ytres og debatteres online. I forbindelse med USA's præsidentvalg i 2016, gik amerikanske vælgere på sociale medier, for at udtrykke deres meninger gennem hadefulde beskeder og angreb på enkeltpersoner. Dette fik amerikanske politikere til at samle repræsentanter for medieplatformene for at diskutere en måde, hvoraf ekstreme holdninger kunne begrænses (Johnson, 2018).

I Danmark har Burkal og Zuleta (2017) udarbejdet en rapport sammen med IMR (Institut for Menneskerettigheder). Rapporten gennemgår data af kommentarer fra DR nyhederne og TV 2 Nyhedernes Facebook-sider. I rapporten sættes der fokus på hadefulde ytringer i den offentlige debat. Her pointeres det, at emnet er vigtigt at italesætte, da det dvæler mellem ytringsfrihed, som er et centralt element i et demokrati, og det at tage hensyn til udsatte eller bestemte grupper, som kan udsættes for racisme, diskrimination eller had. Rapporten opsummerer, at hver syvende kommentar indeholder hadefulde ytringer. Med hadefulde ytringer menes der kommentarer, som er stigmatiserende, nedsættende, krænkende eller truende, og som er rettet mod individer eller grupper af speciel køn, etnicitet, religion, handicap, seksuel orientering, politisk holdning eller social status. Rapporten anbefaler medier at operere med en gennemskuelig og konsekvent moderation af diskursen. Herunder en formulering af klare retningslinjer for den debat, som følger presseetiske regler med konsekvent håndhævelse. Derudover anbefaler rapporten, at regeringen udvikler en national handleplan mod hadefuld tale, som man har set det i Sverige og Norge (Burkal & Zuleta, 2017). Mens flere politikere i Danmark er bange for, at det vil føre til censur og en krænkelse af ytringsfrihed, ønsker Mette Frederiksen (Danmarks statsminister 2020) at afvæbne de sociale medier. Frederiksen mener, at medierne skal kunne stå til retsligt ansvar for, hvad der bliver delt, og ønsker at magten skal tildeles demokratiet og samfundet (Benner & Fejerskov, 2019). Herunder forudsættes det, at medieplatforme effektiviserer diverse værktøjer for i større omfang at forhindre hadefulde ytringer. Et af de værktøjer, der allerede bliver brugt til at håndtere hadefulde ytringer på sociale medier, er maskinlæring, eller kunstig intelligens (AI), som kan finde og klassificere de førnævnte typer af ytringer. Den store mængde af data, som kan samles på sociale medier, gør det til et oplagt emne at udføre maskinlæringsundersøgelser i. Omend ansvaret ligger hos medierne, regeringen eller brugerne, tegner der sig et billede af en problematik, som skal betragtes med nøjsomhed.

Problematikken er relevant i lyset af udviklingen indenfor maskinlæringsfeltet, hvor automatiseret censur bliver lettere end tidligere at implementere. Derfor søger dette projekt, at udvikle en maskinlæringsmodel, som kan klassificere uønsket indhold på samme måde, som man ville gøre det på et socialt medie, og holde resultater fra dette op imod de principper, som ytringsfrihed er bygget på.

1.1 Problematikken omkring hadefulde ytringer & censur

Problematikken omkring begrænsingen af hadefulde ytringer på sociale medier er svær at adressere af flere årsager, bl.a. fordi der ikke findes nogen global betegnelse for, hvad hadefulde ytringer består af. Derfor er det svært at sætte en fælles grænse for, hvad der skal betragtes som hadefuldt. Samtidig opstår problematikken omkring censur af hadefulde ytringer og hvordan dette påvirker ytringsfriheden. Ser vi eksempelvis på det sociale medie Facebook, er formålet her at skabe et rum, hvor der er plads til, at alle kan udtrykke sig. Facebook har dog erfaret, at dette rum misbruges til individers egne agendaer som terrorisme, trusler eller had. Mens Facebook tager ansvar for, at brugere kan føle sig trygge på deres medie, pointere de også, at balancen mellem, at alle kan ytre sig og sørge for deres sikkerhed, er svær (Zuckerberg, 2018).

I IMR's rapport viser det sig, at hver tredje, der deltager i debatter på Facebook, har oplevet at blive talt grimt til, herunder er det i højere grad kvinder, der oplever grove kommetarer, og at kommentarerne ofte er møntet på deres køn, sociale status eller politiske holdninger (Burkal & Zuleta, 2017, p. 83-86). Ca. halvdelen af debattørerne mener også, at tonen på Facebook generelt er blevet hårdere de seneste par år, og at tonen er med til at afholde dem fra at deltage i debatter. Her er det ligeledes kvinder, der oftere end mænd afholder sig fra debatterne. Den generelle holdning i undersøgelsen er desuden, at Facebook har et ansvar for at sikre en ordenlig debattone og at redigere eller fjerne hadefulde ytringer (Burkal & Zuleta, 2017, p. 88-94).

Der er altså en bred enighed om, at Facebook har et ansvar i at sikre, at brugerne føler sig trygge, når de bruger mediet, og at begrænse niveauet af hadefulde ytringer, så brugerne ikke afholdes fra at deltage. Her har Facebook før i tiden været afhængige af, at brugere anmeldte problematisk indhold, men dette betød også, at indhold ikke blev vurderet og evt. fjernet, før brugere havde set det (Zuckerberg, 2018). Det er først af nyere tid, at Facebook er begyndt at anvende AI til proaktivt at identificere og fjerne problematisk indhold. Her har de haft stor succes med at identificere visuelt indhold, men i forhold til hadefulde ytringer er det stadig kun omkring 88% af tilfældene, der identificeres og behandles (Facebook, 2020b). Dette skyldes bl.a. at teknologien ikke er tilstrækkelig, og at de derfor ikke har formået at træne deres system til en grad, hvor den kan yde mere præcist, men også fordi sprog er mere nuanceret og dynamisk og derfor sværere at kategorisere (Zuckerberg, 2018).

I dette projekt beskæftiger vi os, ligesom Facebook, med en teknologisk måde at gribe problemstillingen an på. Gennem maskinglæringsmetoder til sprogprocessering undersøger vi, hvordan maskinlæring kan være med til at klassificere kommentarer baseret på givne faktorer. Herunder undersøges det, hvordan disse platforme kan regulere på vilkår, der ikke begrænser ytringsfrihed, men samtidig tager afstand fra hadefuld retorik. Dette skyldes at ytringfriheden er en af de mest værdsatte kvaliteter ved det liberale samfund (van Mill, 2018). Det er derfor interessant for projektet at undersøge, hvorledes en maskinlæringsmodel går på kompromis med dette, og hvad det kan indebære.

1.1.1 Problemformulering

"Hvordan kan en maskinlæringsmodel konstrueres, til at kategorisere hadefulde ytringer, i overensstemmelse med teori omkring ytringsfrihed og hvilke udfordringer står denne model overfor?"

1.1.2 Arbejdsspørgsmål

1. Hvad er maskinlæring, og hvordan kan en maskinelæringsmodel designes?
2. Hvordan defineres hadefulde ytringer, og hvilke problematikker skaber de i den offentlige online debat?
3. Hvilke tekniske aspekter beror maskinlæring sig på, og hvordan kan dette overføres til hadefuld tale?
4. Hvordan konstrueres en maskinelæringsmodel, der kan multiklassificere hadefulde ytringer med afsæt i teorier om ytringsfrihed?

1.2 Semesterbinding

Dette projekt inddrager de tre HumTek dimensioner; Design & konstruktion, Teknologiske systemer og artefakter og Subjektivitet, teknologi og samfund. Nedenfor beskrives det, hvordan hver enkelt dimension spiller en rolle i projektet, og hvordan de alle bidrager med forskellige synspunkter og mulige vidensbidrag.

- **Design & konstruktion.** I den første dimension, Design & konstruktion, fokuserer vi på processen, hvori projektet udarbejdes. Her benytter vi den agile udviklingsmetode Scrum som arbejdsmetode, da denne giver god mulighed for fleksibilitet og iterativt arbejde med udviklingen af projektet. Den iterative udviklingsproces benyttes også under udviklingen af vores maskinlæringsmodel, hvor vi skaber en prototype af modellen, tester, evaluerer og gentager iterationen med den nye viden og erfaring. Gennem denne tilgang har vi altid en prototype, der virker og kan anvendes i løbet af projektforløbet.
- **Teknologiske systemer og artefakter.** I den anden dimension, Teknologiske systemer og artefakter, fokuserer vi på udviklingen af vores maskinlæringsmodel. Her går vi i dybden med at forstå, hvordan maskinlæring fungerer, hvordan modellen trænes, og hvilke fejl der typisk kan opstå, dermed kan vi være opmærksomme på disse og så vidt muligt forsøge at undgå dem. Ved at gå i dybden med disse emner opnår vi en større forståelse for, hvordan de forskellige komponenter i maskinlæring arbejder sammen for at opnå det overordnede mål. Dette kaldes også for det operationelle princip (Jørgensen, 2017, p. 29). Med større forståelse for maskinlærings indre teknologier får vi også en større forståelse for maskinlæring som teknologi, og hvilke muligheder den skaber.

- **Subjektivitet, teknologi og samfund.** I den tredje og sidste dimension, Subjektivitet, teknologi og samfund, fokuserer vi på ytringsfrihed, hadefulde ytringer og censur, samt hvilke begrundelser der kan være for at censurere ytringer. Ligeledes vil vi have fokus på den individuelle såvel som for samfundet, og om maskinlæring kan benyttes til at mindske hadefulde ytringer på sociale medier. Vi vil desuden diskutere nogle af de udfordringer, der kan opstå, når man træner en maskinlæringsmodel til at genkende og klassificere sprog. Herunder kan foranderligheden i sprog være en afgørende faktor i, hvor effektiv modellens kategoriseringer er.

1.3 Videnskabsteoretisk tilgang

I dette projekt tilgår vi problemet med den antagelse, at der findes en praktisk løsning, og at vi kan tilegne os viden gennem de undersøgelser, der står bag denne løsning. Det er vores indstilling, at relevant og værdifuld viden og funktionalitet kan opnås igennem systematisk indsamling af data og analysering af dette (Beck, 2011, p. 28). Denne videnskabsopfattelse er altså til dels positivistisk. Når der arbejdes med maskinlæring, trænes en model på et pågældende datasæt og derved justere parametre, for at øge nøjagtigheden af forudsigelser. Disse parameter og metrikker anser vi, i lys af vores videnskbsteoretiske tilgang, som værdifulde og brugbare. Det datasæt som modellen tilegnes med, antager vi, beskriver en akkurat repræsentation af verdenen, som den er på internettet. Vi tager udgangspunkt i noget konkret, hvilket definerer den positivistisk videnskabelse (Beck, 2011, p. 28). Ligesom klassisk positivistisk forskning tager vi udgangspunkt i en induktiv videnskabelse. Vi har udvalgt data, indsamlet fra sociale medier, og driver derefter analyse på denne data igennem vores maskinlæringsmodel. Modellen vil derefter igennem, hvad den kan erfare, udlede et sammenhæng eksempelvis imellem en sætning eller et brugt ord, og i hvilken grad dette stykke data er hadefuldt. Dette stemmer overens med, hvordan den induktive metode ellers foregår, hvor en forsker systematisk observerer og dokumenterer fænomener, hvorefter forskeren kan tegne et billede af verdenen (Beck, 2011, p. 30).

Når vi til dels arbejder positivistisk, tilegner vi os ligeledes antirealisme, som også forbindes med positivismen. Denne bunder i, at videnskabelse består af systematisering af sanseindtryk og ikke de teorier, der ligger bag dette (Beck, 2011, p. 32).

Vi forholder os til de praktiske udfordringer, der ligger i problemet med hadefulde ytringer, og de konsekvenser disse hadefulde ytringer har på mennesker. Med dette projekt mener vi altså, at den viden, som vi kan uddrage fra systematisk gennemgang af data, er værdifuld, og at den pragmatisk kan bidrage til løsningen af det problem, vi arbejder med.

Det er dog vigtigt at understrege, at vi, imodsætning til ærkepositivister, ikke forholder os, som om dette er den objektive sandhed. Vi anerkender, at der er mere til videnskaben end denne positivistiske tilgang til verdenen. Sprog er komplekst og foranderligt, og der er derfor mange videnskabelige udfordringer i at lade en computer definere, hvad der er hadefuldt, og hvad der ikke er. Vi anser dog vores projekt som en pragmatisk løsning på et problem. Vi mener ikke, at dette er den eneste kilde til erkendelse, og vi mener ikke, at sanserne per se er den eneste adgang til objektiv viden (Beck, 2011, p. 32-33). Vi anerkender ligeledes at krænkelse og forargelse, som følge af hadefulde ytringer, er en subjektiv sag,

men vi har dog alligevel valgt at forholde os positivistisk, da vi er nødt til at anvende et datasæt til træningen af vores maskinlæringsmodel. For at modellen kan lære af datasættet, må det betragtes som objektivt, da vi ellers ikke kan udlede objektiv viden fra dette.

2 Grundlæggende om maskinlæring

I dette afsnit beskrives indledende, hvad maskinlæring er og hvad det kan bruges til. Derudover beskrives der, hvad vi forventer, at vores egen maskinlæringsmodel kan bruges til i henhold til sociale medier, online debat og ytringsfrihed. Afsnittet danner grundlag for den gren af maskinlæring, som vi beskæftiger os med og i hvilket kontekst. Den specifikke tekniske teori, som er anvendt i konstruktionen af vores egen maskinlæringsmodel, kan findes i afsnit 5 og 6.

2.1 Hvad er maskinlæring?

Maskinlæring er baseret på et paradigme inden for computation, hvor man forsøger at udvikle algoritmer, der kan selvforbedres gennem træning. Computeren skal altså, gennem data, selv kunne skabe regler, som kan bruges til at processere data i stedet for, at en programmør, programmerer regler og giver dem til computeren. Computeren trænes altså i højere grad end den programmeres. Træningen består af, at computeren præsenteres for et træningsdatasæt, og herefter selv finder strukturer og sammenhænge mellem data og svar, som kan sammenfattes i en algoritme, der kan bruges, når computeren præsenteres for et nyt datasæt (Chollet, 2017, p. 5). Træningsdatatsættet består af data samt tilhørende svar, altså det der vil være det forventede output fra maskinlæringsmodellen. For eksempel kunne et træningsdatasæt bestå af en masse forskellige billeder, hvor dem, der indeholder en banan, er markeret som "banan". En maskinlæringsmodel bruger således et træningsdatasæt og en måde at måle, hvorvidt modellens output stemmer overens med det forventede output. Den sidst nævnte del er her, hvor læringsdelen finder sted i og med, at resultatet af målingen sendes retur til modellen, som herefter tilpasser algoritmen efter denne feedback (Chollet, 2017, p. 6). Når man giver et datasæt til en model for at træne den, løber datasættet oftest ikke kun igennem modellen en enkelt gang. Et enkelt gennemløb kaldes for en epoke, og oftest sker det, at man definerer et bestemt antal epoker, som dataet skal igennem. På denne måde kan modellen forbedre sig, når data gennemløbes igen og igen (Berry, Mohamed, & Yap, 2019, p. 3-6).

Fordelen ved en maskinlæringsmodel er derfor, at den evner at lære af data, og på den måde kan kompleksiteten af modellens opgaver øges sammenlignet med, hvad en programmør kunne specificere i et program bestående af trinvise processer og algoritmer. Ligeledes vil modellen blive ved med at lære og tilpasses efter den tages i brug. Her vil den nemlig hele tiden modtage ny data, som træner modellen, og på den måde kan modellen blive bedre og bedre, uden at en programmør aktivt bruger tid på at gøre den bedre (The Royal Society, 2017, p. 19).

Superviseret og ikke-superviseret læring

For mere præcist at kunne formulere formålet med denne opgave, vil vi komme ind på de to overordnede typer af maskinlæring, den superviserede og den ikke-superviserede. Dette projekt handler dog om en model, der anvender supervision, derfor er dette beskrevet mere udførligt.

- **Superviseret læring** foregår som andre typer modeller ved, at man videregiver sit datasæt til modellen. I en superviseret model bliver modellen præsenteret for input, der er kategoriseret korrekt. På baggrund af den kategoriserede data, kan denne type model lære at kategorisere fremtidigt input korrekt. Dette vil sige, at såfremt modellen får et sæt af billeder af pærer og bananer, der er kategoriseret korrekt, er den selv i stand til at udlede en algoritme, der kan kende forskel på et billede af en pære eller banan i et fremtidigt billede. Altså hvilke instanser der er af kategorien pære, og hvilke der er af kategorien banan. Herefter vil maskinlæringsmodellen være i stand til at kigge på en fremtidig instans og selv tage en beslutning om, hvilken af de to frugter der er tale om. Superviserede modeller kan også udføre regression og forsøge at forudsige en bestemt feature for en test-observation. I pære og banan eksemplet kan input bestå af farven af billedernes pixels, den procentvise sammensætning af farver eller måske de former, som farvens sammensætning kan danne på et billede. En observations feature bliver oftest angivet ved numeriske eller kategoriske datatyper. Datasættets output er i denne case en angivning af, hvorvidt instansen er en pære eller banan. Når en superviseret model bliver anvendt, kender den altså både input og målet for sit træningsdatasæt, hvilket betyder, at for en veltrænet model vil vi kunne returnere og forudsige det korrekt associerede output for et givet input (Berry et al., 2019). Denne type model, der kan kategorisere mellem forskellige klasser, hedder en klassifikationsmodel.

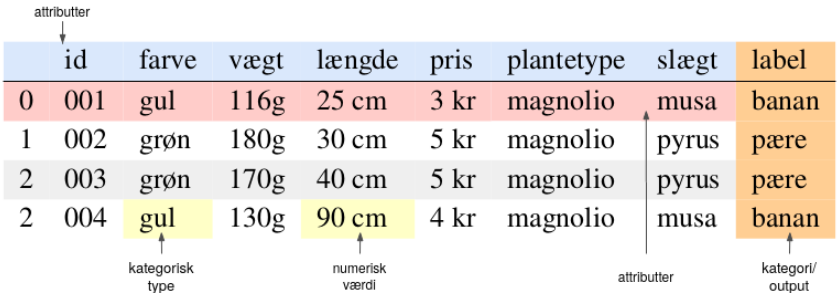
- **Ikke-superviseret læring** foregår, modsat supervision, ved at træningssættet, som gives til modellen, ikke indeholder de korrekte klasser. Siden disse typer modeller ikke har klasser som outputs i sit datasæt, er det også andre typer opgaver, som de kan løse i forhold til de superviserede modeller. Ikke-superviserede modeller kan eksempelvis selv udføre 'clustering' og dele et datasæt op i klasser, baseret på datasættets egenskaber. Altså vil den, hvis den fungerer som den skal, kunne sortere pærer og bananer op i to klasser (Berry et al., 2019, p. 10-13). Ofte bliver denne type ikke-superviserede modeller brugt til at definere klasserne for inputtet til en klassifikationsmodel (Berry et al., 2019, p. 4). Alternativt kan ikke-superviserede modeller også finde interessante nye, førhen uopdagede, associationsregler i datasæt. Dette er ofte anvendt i analyse af supermarkedskvitteringer (Chollet, 2017, p. 94).

Datasæt-terminologi i superviseret læring

I superviseret læring er der lidt terminologi, som kan anvendes til at gøre det lettere at adressere, hvilke elementer i et datasæt vi taler om. Ovenfor anvendte vi et eksempel vedrørende

en algoritme, som skulle klassificere mellem billeder af pærer og bananer, altså et binært klassificeringsproblem.

Af figur 1 fremgår et eksempel på et datasæt. Her er attributterne *farve*, *vægt*, *længde*, etc. *Gul* er en kategorisk værdi, *25 cm* er en numerisk værdi og *pære* og *banan* er kategorier / klasser.



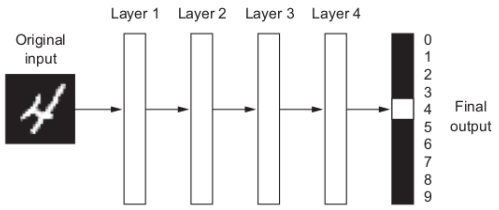
Figur 1: Datasæt eksempel

I forhold til typer af data opererer datasættet med to typer, den ene er numerisk, og den anden er kategorisk, som grupperer observationerne ind i kategorier. En kategorisk datatype kan f.eks. være en farve, men kan også være numeriske værdier, som er grupperet inden for intervaller. I programmering kan man ofte repræsentere en kategorisk type ved det, man kalder en "enum", en type som kan have én ud af et par foruddefinerede værdier. Et datapunkt kan også være numerisk og simpelthen være udgjort af et tal (Google Inc., 2020).

Neurale netværk

I dette projekt arbejder vi med neurale netværk (NN), som især de senere år er blevet en af de mest anvendte metoder indenfor maskinlæring. NN fungerer ved at stable en række lagvise repræsentationer oven på hinanden. Disse lag består af noder eller neuroner. Neuroner er matematiske funktioner, der taget et eller flere input og giver et output(Nwankpa, Ijomah, Gachagan, & Marshall, 2018, p. 2). I NN kan der være en lang række lag, men som udgangspunkt findes der et input lag, hvor rå data bliver videregivet til modellen, nogle gemte lag som fortolker og finder mønstre og til sidst et output lag (Chollet, 2017, p. 7-8). I figur 2 er denne proces visualiseret ved et NN, som skal klassificere et billede af et firtal. Igennem disse lag leder modellen altså efter mønstre og kommer med en forudsigelse om, hvilket tal den tror billedet viser. I superviseret læring foregår denne proces ved at give billeder til modellen, som er angivet med et label for, hvilket tal billedet repræsentere. Processen kaldes *træning*, og det er her modellen lærer at finde mønstre, som den associerer med et firtal. Disse mønstre kan være pixels på forskellige områder, farver, former mm. Når et NN har været gennem denne træningsproces for mange forskellige billeder af tal, kan man derefter give modellen en række billeder, den ikke kender til. Derefter køres billederne gennem disse lag for at finde mønstre, som modellen kan associere med tal. Til sidst kom-

mer netværket med et output, som er det tal, netværket forudsiger, at billedet repræsentere. Chollet (2017) beskriver denne proces som en distilleringsproces af information, hvor information går gennem mange efterfølgende lag og kommer renere ud, end da den kom ind (Chollet, 2017, p. 9). I eksemplet i figur 2 kommer billedet af et firtal ind, og den digitale repræsentation af firtallet kommer ud.



Figur 2: MNIST eksempel
(Chollet, 2017, p. 9).

Hvordan ved vi, om et neuralt netværk forudsiger korrekt?

Den neurale netværksmodel, der er beskrevet i denne rapport, er en superviseret klassificeringsmodel, og derfor vil netværkets output-lag ofte have en node, der svarer til hver klasse. Da modellen er superviseret, har den derfor et ønsket resultat, som vi kan bruge til at måle en loss-gradient på. En loss-gradient kan udregnes på flere måder, men et eksempel er at udregne output − forventet ouput. Output kan f.eks. være mellem 0 og 1, hvor en værdi desto tættere på 1 angiver, hvor 'meget' netværket mener at inputtet tilhører den pågældende klasse

$$Loss\ gradient\ = output - forventet\ output$$

Aktuelt findes der flere typer af loss-funktioner, som laver denne udregning på forskellige måder. Efter en loss-gradient er opnået, kan modellen arbejde sig baglæns gennem netværket og opdatere alle forbindelserne / vægtene mellem noder med den nye loss-gradient. Denne algoritme kaldes 'backpropagation', og er en måde, hvorpå et netværk lærer (Chollet, 2017, p. 51-53).

Efter træningen af en model, eller ved senere tests, kan man også måle på et netværks nøjagtighed *(eng. accuracy)*, ved rent faktisk at måle succesraten for at klassificere korrekt i forhold til den fulde mængde klassifikationer.

$$Accuracy = \frac{Antal\ klassifikationer}{Totale\ klassifikationer}$$

Denne udregning giver et billede af, hvor ofte netværket klassificerer korrekt. Loss-gradienten bruges altså som udgangspunkt til, at netværket kan lære af sine forudsigelser, mens accuracy giver et helhedsbillede af, hvor god modellen er til at forudsige.

Natural Language Processing

Natural Language Processing (NLP) handler, modsat billedegenkendelse, om maskinlæring i forbindelse med ord og sætninger. NLP finder mønstre for at kunne klassificere i forskellige sammenhænge, men sætter krav til, at disse input af ord kan associeres med hinanden for at finde semantiske ligheder (Chollet, 2017, p. 180). NN, som beskæftiger sig med NLP, kan bruges til en række forskellige ting, heriblandt færdiggørelse af sætninger eller til kategorisering af sproglige udtalelser, som vi arbejder med i dette projekt.

2.2 Hvad kan maskinlæring bruges til?

Maskinlæring er ikke noget, man hører så meget i daglig tale, men det er en teknologi, som findes i en bred vifte af applikationer, som mange af os anvender dagligt. F.eks. anvendes maskinlæring ofte til at tagge billeder, herunder på sociale medier. En model kan være i stand til at analysere et billede og associere billedets indhold med information, så modellen f.eks. genkender en kat i et billede, selvom den ikke har set det specifikke billede før. Billedegenkendelse kan også anvendes til at læse håndskrift og konvertere det til digital tekst. Et andet eksempel på maskinlæring i hverdagen er virtuelle personlige assistenter som Siri, Alexa og Google Assistant, som benytter sig af NLP. Disse er i stand til at forstå tale og konvertere det til tekst, udføre handlinger baseret på den talte ordre og svare med tekst eller tale. Et tredje og mere simpelt eksempel kunne være spamfiltret i vores e-mail, som ofte består af en maskinlæringmodel, der er blevet trænet til at udpege og klassificere karakteristika, som en spammail ofte besidder. Det er dog ikke altid, at spamfilterede får fjernet de rigtige mails, og derfor finpudses modellen hver gang, vi markerer noget som spam eller ikke-spam (The Royal Society, 2017, p. 21-23).

Selvom maskinlæring har udviklet og forbedret sig i en imponerende hastighed henover det sidste årti, og i nogle opgaver performer bedre end mennesker (The Royal Society, 2017, p. 19), skal det dog påpeges, at maskinlæring ikke er løsningen til alverdens problemer. Maskinlæring er godt til at finde mønstre i data, og repræsentere data på meningsfulde måder, men maskinlæring kan ikke anvendes til noget, der kræver rationelt ræsonnement. Man skal altså på nuværende tidspunkt være forsigtig med at forvente for meget af maskinlæring og menneskelignende AI, da denne type teknologi kun kan tage data og repræsentere det på andre måder, og selvom den kan blive rigtig god til dette, betyder det ikke, at den forstår dataet, som den arbejder med (Chollet, 2017, p. 325). Altså ved den ikke nødvendigvis, hvad en kat er, selvom den kan identificere en kat i et billed, eller hvad et smil symboliserer, selvom den kan genkende et smil.

Hvad skal vores model kunne?

Vores model trænes med et datasæt, der indholder forskellige typer af kommentarer, hvorefter programmet lærer at repræsentere kommentarerne alt efter, hvor hadefulde de er. Herefter er programmet altså i stand til at bedømme, om en ny kommentar, der ikke stammer fra træningssættet, er stødende eller ej, og kan kategorisere disse. Målet med projektet er

ikke at implementere vores model på et socialt medie, men nærmere at skabe en maskinelæringsmodel, der kan kategorisere strenge af ord, sætninger og kommentarer, således at en funktionalitet videre kan anvendes på disse kategoriseringer. Det er altså ikke vigtigt for os, at vores maskinelæringsmodel kan gøre noget funktionelt, som at fjerne kommentarer udover at afgøre, hvorledes og i hvilken grad disse består af hadefulde ytringer. I dette projekt betragter vi selve denne kategorisering som det udfordrende og den faktiske brug af disse kategoriseringer som noget trivielt. Når vi har formået at skabe en maskinlæringsmodel, der kan opnå denne kategorisering tilfredsstillende, vil vi altså betragte det tekniske arbejde som udført.

3 Metode

Gennem dette projekt har der været fokus på iterative processer, da ingen af gruppens medlemmer har tidligere erfaring med at konstruere en maskinlæringsmodel. Vi har derfor struktureret projektet på en iterativ måde, så vi i forbindelse med konstruktionen af en model og projektskrivningen har haft mulighed for at genbesøge tidligere arbejde.

Afsnittet starter med at beskrive, hvordan vi har brugt den agile udviklingsmetode Scrum, og hvordan denne relaterer sig til projektskrivning ift. systemudvikling. Herefter beskrives vores fremgangsmåde med prototyper i arbejdet med maskinlæring, og hvordan denne metode har bidraget til en iterativ udviklingsproces, som gør, at vi kan lære gennem erfaringer omkring arbejdet med maskinlæring. Til sidst beskrives CRISP-DM som model til at strukturere et maskinlæringsprojekt. Denne fungerer som framework til et projektforløb og sætter fokus på de vigtigste faktorer, når der arbejdes med data og maskinlæring.

3.1 Agile udviklingsmetoder i projektskrivning

Behovet for agil softwareudvikling og processer har længe været anerkendt, og i 1990'erne tog begrebet fart med udviklingen af ideer omkring agile metoder, f.eks. ekstrem programmering og Scrum. Denne epoke af softwareudvikling havde fokus på hurtig levering, for at imødekomme udviklingen, der fandt sted i tech-industrien. I takt med at nye teknologier tog form, blev softwareudviklingsprojekter ofte forældede, før de var færdige. Der var derfor brug for en nytænkende måde at udvikle software på, som stod i kontrast til den langsommelige plan-drevne metode. En af de metoder, som har vundet stort indpas, er Scrum. I Scrum bliver udviklingsprocessen håndteret inkrementelt, hvor hvert inkrement skal forstås som en ny udvidelse til et givet system, der bliver gjort tilgængeligt for brugerne hver 2-3 uge (Sommerville, 2016, p. 72-74).

Agile udviklingsmetoder er kendt for at inddrage brugerne i udviklingsprocessen og have mindre fokus på dokumentation. Kravspecifikationer bliver mere flydende med udviklingsprocessen, der er ingen detaljeret systemspecifikation, og brugerkrav bliver betragtet som den vigtigste ramme for systemet. Derudover bliver der lagt stor vægt på automation og målbarhed. Herunder bliver der udformet en plan for hver periode, så udviklere kan arbejde sammen og vægte opgaver, der skal fuldføres indenfor dette inkrement.

Metoden til at organisere og planlægge et produkt kan overføres til projektskrivning. Herunder arbejdes der ud fra nogle begreber, som gør sig gældende ved udvikling af software såvel som et projekt. Antag at sektioner i et projekt er inkrementelle udvidelser, der udvikles på indenfor en given periode. Denne periode kaldes et *sprint* og forløber sig gerne over to uger. I starten af et sprint samles gruppen og definerer opgaverne, der skal indgå i hele projektet eller for det kommende sprint. Disse opgaver defineres i fællesskab og tilføjes til en *product backlog*, hvor opgaver efterfølgende tildeles et sprint. Inden sprintet starter, nedbrydes disse opgaver i mindre håndgribelige opgaver, og der gives et tidsestimat, som skal indikere, hvor lang tid det tager at løse opgaven. Dette betyder, at en *velocity* kan udregnes, som viser, hvor langt man er for at komme i mål med det nuværende sprint. Altså en måde hvorpå man kan måle sin effektivitet og dermed skabe overblik over hele processen (Sommerville, 2016, p. 84-87).

Til at danne dette overblik tildeles rollen som *Scrum master*, der holder styr på et *Scrum board*, hvor sprints, opgaver og velocity kan findes. Denne planlægning suppleres med daglige møder, kaldet et *daily*, som er et dagligt møde på 15 minutter, hvor hvert medlem har nogle minutter til at forklare, hvad de arbejdede på dagen før, og hvad der skal arbejdes på den pågældende dag, samt om der er behov for hjælp, opklarende spørgsmål, mm. (Sommerville, 2016, p. 93-97).

Ved at arbejde agilt har vi mulighed for at tilpasse os ny viden og gennemarbejde vores produkt gennem iterationer. En af grundende til, at vi har valgt denne metode, beror sig på ideen om at skabe viden gennem udvikling af vores egen maskinlæringsmodel, hvor man netop arbejder ud fra, at en evaluering af modellen kan føre til ændringer, som skaber nye opdagelser. Samtidig har vi fra tidligere projekter erfaret os, at arbejdet med sprint, en backlog og dailies giver et fuldendt overblik og styrker kvaliteten af projektets indhold.

3.2 Iterative udviklingsprocesser i maskinlæring

I sammenhæng med Scrum har vi i dette projekt arbejdet iterativt med udviklingen af vores maskinlæringsmodel ved hjælp af prototyping. Prototyping er en hurtig måde at skabe et program, der virker, og som kan testes og eksperimenteres med tidligt i processen. En prototype er dermed også en meget tidlig version af programmet, som gennem tests og eksperimenter er med til at opdage hvordan et program vil agere i givne situationer. Test af en prototype kan være med til at validere tidligere antagelser, men kan også bruges til at opdage fejl og mangler i den nuværende prototype. Derfor er en prototyping og en iterativ udviklingsproces essentiel, når et nyt program skabes, da det giver mulighed for løbende at opdage fejl og udbedre disse, inden de skaber større og mere komplekse problemer i programmet. Ligeledes giver det også mulighed for hurtigt at ændre og tilpasse programmet til nyopdagede krav og behov uden, at programmet behøver at undergå en større modificering (Sommerville, 2016, p. 62). Formålet med en prototype bør være fastlagt fra starten af sprintet. Dette er for at undgå misforståelser og for at drage flest muligt fordele af resultaterne fra de test som udgøres. En iteration i udviklingsprocessen består altså af at definere prototypens formål og funktionalitet, skabe prototypen og til sidst teste og evaluere prototypen. Herefter kan iterationen gentages, med ny viden tilføjet fra evalueringen, i en iterativ

udviklingsproces (Sommerville, 2016, p. 63).

Gennem dette projekt har vi lavet prototyper af vores maskinlæringsmodel i og med, at den løbende er blevet trænet med træningsdatasættet og herefter justeret, for at kunne nærme sig et ønsket resultat. Når modellen opnår et tilfredsstillende resultat, testes den med ny data eller andre parametre, for at afdække eventuelle svagheder og muligheder for forbedring. Modellen indgår altså i en iterativ udviklingsproces, hvor den for hver iteration kommer nærmere de rigtige resultater i forhold til formålet.

3.3 CRISP-DM projektproces

CRISP-DM (CRoss Industry Standard Process for Data Mining) er et framework baseret på praktiske erfaringer fra rigtige projekter. I slut 1990'erne var der ikke nogen specifik model for udvikling af Data mining projekter, og derfor blev der nedsat et konsortium af de ledende virksomheder indenfor feltet. Projektet var sponsoreret af Europa-kommision under ESPRIT programmet og havde til formål at danne et sådan framework. Data mining er en kreativ proces, som kræver viden og erfaringer, men der var hidtil ikke noget framework eller fremgangsmåde, som kunne skabe ensartethed i projekterne. Dette betød, at projekterne var dybt afhængige af enkelte personer eller teams samt deres evne til at praktisere et sådan projekt. Derfor var et framework søgt, som kunne definere en standard fremgangsmåde og hjælpe til at oversætte forretnignsproblemer til Data mining opgaver, herunder datatransformation, teknikker og evalueringsprocess. CRISP-DM projektet adresserer disse problematikker ved at definere en proces-model til at udføre Data mining projekter, som både er individuelle fra industri og sektor men også ifm. teknologibrug. CRISP-DM sigter efter at gøre store Data mining projekter pålidelige, ensartede, håndterbare, hurtigere og mindre kostfulde (Wirth & Hipp, 2000).

CRISP-DM har sin oprindelse indenfor Data mining, men er i dag også anset som den mest brugte fremgangsmåde indenfor maskinlæring (Clark, 2018). Modellen følger en række faser, beskrevet af Wirth and Hipp (2000), som sikrer, at der arbejdes med de faktorer, som kan afgøre udfaldet af et succesfuldt projekt, herunder;

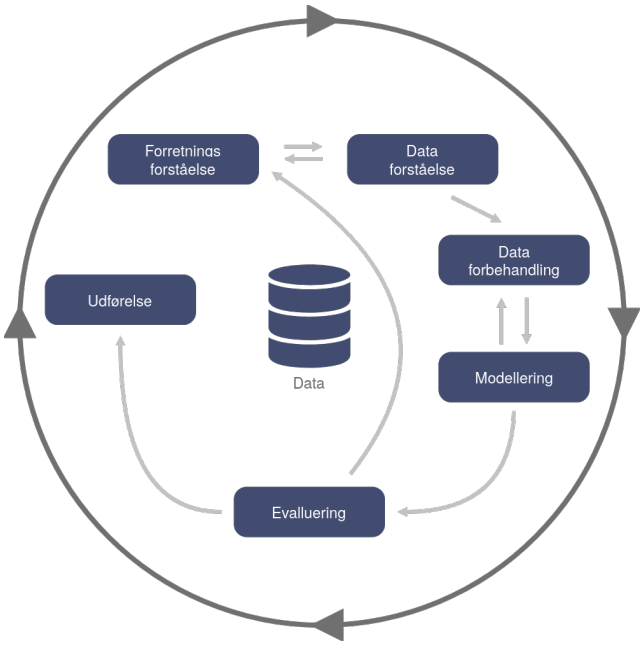
- **Forretningsforståelse**. Fasen adresserer forståelse for projektets mål og krav fra et forretnignsperspektiv, og skal lede til, at disse mål og krav konverteres til et Data mining problem. Herunder brugssituationer og hvilke problemer modellen skal løse.

- **Dataforståelse**. Fasen starter indledende med dataindsamling og det at blive familiær med den indsamlede data. Fasen er med til at sikre kvaliteten af den indsamlede data, men bidrager også til hypotesedannelse. At få en forståelse for data er afhængig af forretnignsforståelsen, og uden denne forståelse vil det ikke være muligt at afgøre kvaliteten af data.

- **Dataforbehandling**. Fasen handler om at klargøre data til at blive videregivet til modellen. Fasen kan være meget tidskrævende alt efter, hvor struktureret den indsamlede data er. Fasen indeholder bl.a. dataoprydning, opsætning og attributselektering, og forventes besøgt flere gange gennem projektets proces.

- **Modellering**. I denne fase bliver modeller og teknikker udvalgt til, herunder optimering af parametre. Der findes ofte flere forskellige modeller til samme problemstillinger, og det er derfor også en fase med afprøvning og tests. Parametres indstillinger bliver genbesøgt og finjusteret.
- **Evaluering**. Når en model, som imødekommer den ønskede kvalitet, er konstrueret, evalueres den. Hvert step bliver genovervejet, og udfald bliver holdt op mod de egentlige forretningsmæssige mål og krav. Det er her vigtigt at identificere, hvorhvidt alle problematikkerne er tilstrækkeligt dækket.
- **Udførelse**. I sidste fase foretages udførelsen, som dækker over faktisk implementering og eller rapportskrivning. Denne fase er i forretningsøjemed tiltænkt brugeren af produktet. Ligemeget hvem der udfører fasen, er det vigtigt at tilegne sig forståelse for, hvad der skal til for, at modellen praktisk kan anvendes.

CRISP-DM er en iterativ model, og det er vigtigt at understrege processen som en cyklus. I figur 3 fremgår CRISP-DM modellen og dens faser illustreret. Wirth and Hipp (2000) pointerer, at pilene indikerer de vigtigste og hyppigste afhængigheder mellem faser, men at det kommer an på det enkelte projekt og udfaldet af en fase, hvilken fase der er den efterfølgende.



Figur 3: CRISP-DM proces model
(Wirth & Hipp, 2000)

CRISP-DM som maskinlæringsproces

Clark (2018) argumenterer for brugen af CRISP-DM til maskinglæringsprojekter, og pointerer specifikt vigtigheden af valget af maskinlæringsalgoritme, herunder den valgte algoritmes fortolkningsmulighed i forbindelse med de forretningsmæssige behov. Derudover pointeres vigtigheden omkring at undersøge om data er tvedelt i trænings- og testdata. Dette hjælper mod *overfitting*, altså at en algoritme matcher et specifikt datasæt for meget. Herunder anbefales et 80/20 split, hvor 80 % af datasættet fungerer som træningsdata, og 20 % fungerer som testdata. Samtidig pointeres evaluering som en nøgle faktor, hvor nøjagtighed ikke er ensbetydende med kvalitet. Her skal også undersøges kulturel bias i modellen, og hvorhvidt modellen forfølger en selvforstærkende bias f.eks. i forbindelse med race variabler (Clark, 2018).

I dette projekt vil CRISP-DM modellen bliver brugt til at konstruere en maskinlæringsmodel, hvoraf forretning udskiftes med problemtikken omkring hadefulde ytringer i online debat. Første step er altså en generel forståelse for, hvad hadefulde ytringer er, hvorfor de opstår og problematikken derom. Med denne forståelse vil vi være i stand til at udarbejde et maskinglæringsproblem, der danner rammer for udviklingen af modellen. Efterfølgende betragtes modellens principper i maskinlæringsregi med en iterativ tilgang, hvor sidste step, udførelse, dog undlades i og med, at vi ikke implementerer modellen på et socialt medie.

4 Konceptualisering af hadefulde ytringer og ytringsfrihed

I dette afsnit beskrives, hvad ytringsfrihed betyder, og hvilke grunde der kan være til at censurere eller frigøre ytringer. Her inddrager vi "The Harm Principle"og "The Offence Principle"som eksempler på, hvorfor censur i nogle tilfælde kan være nødvendigt. Herudover kommer vi også ind på, hvilke samfundsmæssige, men også personlige, konsekvenser det kan have at blive udsat for hadefulde ytringer, og hvorfor det er vigtigt at tage aktion.

4.1 Ytringfrihed og hadefulde ytringer

Da vi i dette projekt arbejder med konstruktion af en maskinlæringsmodel, der kan kategorisere hadefuldeytringer, er det vigtigt at overveje, hvilke implikationer dette kan have. Fjerner man kommentarer eller beskeder, laver man restriktioner på, hvad der er tilladt at sige i det givne forum. Det er derfor vigtigt at overveje, hvordan disse moderationer kan påvirke ytringsfriheden, samt hvorvidt moderationerne, som vores model kan lave, er retfærdiggjort.

Hvad er ytringsfrihed?

Ytringsfrihed er et omtalt debatområde og en frihed, der er værdsat af mange mennesker, samfund og nationer. Der er dog, som påpeget af van Mill (2018), et lidt paradoksalt element ved denne frihed. Ytringfriheden vil nemlig i sig selv altid være begrænset - ultimativ ytringsfrihed findes ikke. Dette skyldes, at ytringsfrihed altid udspilder sig i en kontekst af

konkurrerende værdier. Dette kan klargøres, når to tilfælde af anvendt ytringsfrihed mødes, og det skal besluttes, hvilken en ytring der skal prioriteres over den anden (van Mill, 2018). Eksempelvis kunne én persons ytringer sætte en praktisk stopper for en anden persons ytringer, for eksempel hvis en persons råben overdøver en politisk tale. Her skal der altså tages et valg om, hvilken ytringsfrihed der skal forbeholdes (van Mill, 2018). Ytringsfrihed er diffust i og med, at det som sådan ikke eksisterer, i hvert fald ikke som kompromisløs ytringsfrihed hvor alt og alle må ytre sig hele tiden. Det altså essentielt at definere grænserne for ytringfriheden. Sociale medier er ligeledes tvunget til at vælge, hvem der skal have sin ytringsfrihed forbeholdt, når de modererer fora, billedeopslag eller kommentarfelter.

The Harm Principle

Van Mill (2018) nævner John Stuart Mill's "Harm Principle"som et udgangspunkt for en grænse for ytringfriheden, og bruger dette princip til at udforske forskellige grader af ytringsfrihed. Van Mill understreger, at J.S. Mill mener, at alle har ret til at udtale sig om hvad som helst, og han bruger hertil følgende J.S. Mill citat til at understrege dette: *"If all mankind minus one were of one opinion, and only one person were of the contrary opinion, mankind would be no more justified in silencing that one person than he, if he had the power, would be justified in silencing mankind. (1978, 16)"* (van Mill, 2018). Men hvilke grænser skal der så være for ytringsfriheden? Dette er, hvor J.S. Mills "Harm Principle"kommer ind i billedet. Ifølge dette princip er den eneste grund, der kan være til at fratage en anden persons ytringsfrihed, hvis denne magtudøvelse vil forhindre skade på andre (van Mill, 2018). Hvad J.S. Mill har ment med skade, er et stærkt debatteret emne, men i van Mill's essay, som vi tager udgangspunkt i, bliver skade forstået som en direkte konsekvens af det, der bliver ytret. Dette princip giver derfor ytringfriheden meget frie tøjler, da det er få udsagn, der kan siges at forsage direkte skade på andre (van Mill, 2018). Eksempelvis er det acceptabelt at beskylde en person for at være medskyldig i at sulte de fattige, men det er ikke okay at beskylde en person for at sulte de fattige, hvis der står en vred forsamling af mennesker, der er klar til at gå amok, ude foran den beskyldtes hus, da dette kunne være en direkte grund til, at vedkommende ville lide skade (van Mill, 2018).

The Offence Principle

En kritik af J.S. Mills "Harm Principle"kommer fra Joel Feinberg som går på, at princippet er uegnet til tilstrækkeligt at dække ytringsfrihed, da Mill's princip giver grønt lys til en for bred vifte af udtalelser. Et alternativ er Feinbergs eget princip "The Offence Principle". Dette princip er bygget på den forståelse, at nogle ytringer er så offensive og forargende, at de kan forbydes. Feinbergs princip evaluerer ytringer på basis af, hvorvidt de krænker eller forarger andre end den aktør, der udtaler udsagnet (van Mill, 2018). En forargelse er mindre seriøs end direkte skade, og straffen for en overtrædelse af dette skal derfor ikke være specielt omfattende. Feinberg fastholder dog, at det er statens opgave at dømme, hvorledes det, der bliver ytret, er forargende eller offensivt (van Mill, 2018).

Dette princip kan dog være svært at håndhæve, netop fordi det at forarge eller krænke

kan være svært at definere. Det kan sagtens tænkes, at to personer vil betragte en ytring som henholdsvis ekstremt forargende og slet ikke forargende (van Mill, 2018). Så selvom det måske intuitivt giver mening at censurere efter, i hvor høj grad en udtalelse er forargende, kan det være svært at definere præcist, hvornår en ytring skal censureres for overtrædelse af dette princip. Feinbergs princip behøver desuden ikke at forbyde ytringer, hvis disse er nemt undgåelige, det vil sige, at hadefulde ytringer eksempelvis kan foregå i privat regi, uden at dette behøver at forbydes (van Mill, 2018).

Hadefulde ytringer, censur og Facebook

Hadefulde ytringer har som udgangspunkt ikke skade som direkte konsekvens. Det bringer sjældent dem, som det bliver brugt imod, i fare, og det kan som så ikke siges at forsage direkte skade på dem. Derfor bliver det svært at bruge J.S. Mill's princip på hadefulde ytringer, og vi må derfor delvist tillade hadefulde ytringer, hvis det kun er dette princip, vi tilslutter os. Van Mill understreger, at der for Feinbergs "Offence Principle"kun skal forbydes hadefulde ytringer, hvis det hadefulde udsagn er uundgåeligt for den, der bliver forarget (van Mill, 2018). For sociale medier bliver dette kompliceret, da der tit kan findes hadefulde ytringer i både opslag og kommentarspor. Da den forargerede dybest set bare kan forlade sin computer eller telefon, slukke den og forsætte med sit liv, kan det siges, at Feinbergs princip ikke gælder her. Det er dog ikke tænkeligt, at dette er i de sociale mediers interesse. Et eksempel på at løse dette problem er Facebooks fællesskabsregler. Disse regler er efter sigende baseret på eksperters rådgivning omkring teknologi, offentlig sikkerhed og menneskerettigheder. Disse regler skal være med til at gøre plads til forskellige synspunkter og holdninger, og Facebook udtrykker, at målet med disse fællesskabsregler altid har været at skabe et trygt sted at komme til udtryk og bruge som talerør (Facebook, 2020a). Disse regler dækker over en række forskellige områder, men vi vil hovedsageligt fokusere på de regler, der omhandler hadefulde ytringer.

Facebook udtrykker, at de ikke tillader hadefuld ytringer, fordi det skaber et intimiderende og ekskluderende miljø, og at hadefulde ytringer i nogle tilfælde kan føre til vold i virkeligehen. Hadefulde ytringer defineres her som et direkte angreb på, hvad Facebook kalder for beskyttede karakteristika: "race, etnicitet, national oprindelse, religiøst tilhørsforhold, seksuel orientering, kaste, køn, kønsidentitet og alvorlig sygdom eller handicap"(Facebook, 2020a). Her er det altså tydeligt, at Facebook har taget et forbehold, der viser, at de prioriterer nogle ytringer over andre. Facebook kan siges både at gøre brug af J.S. Mills princip, idet de forhindrer hadefulde ytringer, fordi det i sjældne tilfælde kan forsage virkelig fysisk vold, og de gør også brug af Feinbergs princip, da de ikke tillader offensive ytringer eller ekspression på deres platform. Facebook er tydeligvis ikke interesseret i at skabe et ekskluderende miljø, hvori de brugere, der potentielt kan blive forarget over hadefulde ytringer, bliver nødt til at søge væk, og derfor tillader de at censurere og moderere i de opslag og kommentarer, der blive lagt op på deres platform.

4.2 Konsekvenser af hadefulde ytringer

Når vi arbejder med hadefulde ytringer og kategorisering og censurering af samme, er det også vigtigt, at vi i dette projekt danner os en idé om, hvad konsekvenserne af sådan hadefulde ytringer er. Kategorisering og censur virker overdrevet, hvis der ikke er en omfattende konsekvens af ikke at foretage nogen tiltag.

Hvad er på spil, og hvorfor bør der tages aktion?

Den tidligere nævnte rapport, som IMR har lavet vedrørende hadefulde ytringer på sociale medier med fokus på DR og TV2's platforme på Facebook, kommer også ind på konsekvenserne af hadefulde ytringer. Her bliver det gjort klart, at hadefulde ytringer både kan have konsekvenser for samfundet som helhed, men også for den enkelte bruger af det sociale medie. Når sociale grupper af eksempelvis køn, etnicitet eller seksuel orientering bliver tildelt egenskaber, der får dem til at fremstå som mindreværdige borgere i samfundet, kan det være med til at forstærke fejlagtige og skadelige stereotyper. Denne cementering kan være med til at legitimere yderligere diskriminerende adfærd (Burkal & Zuleta, 2017, p. 26-27). Derudover kan det hadefulde sprogbrug være med til at ekskludere de omtalte mennesker, samt fremprovokere angst og frygt hos dem, der bliver omtalt. Det er derfor ikke overraskende, at IMR's rapport peger på norske erfaringer, der antyder, at hadefulde ytringer svækker demokratiet, fordi der med hadefulde ytringer kommer en begrænset deltagelse i den offentlige debat. Derudover peger IMR på en anden undersøgelse fra Politihøjskolen i Norge, der viser en sammenhæng mellem hadefulde ytringer og vilje til vold. Hadefulde ytringer bliver af IMR set som et demokratisk problem, der svækker den offentlige debat ved at reducere antallet af deltagere (Burkal & Zuleta, 2017, p. 27).

Konsekvenser for den individuelle

Set ud fra den individuelles perspektiv, kan hadefulde ytringer på nettet have fatale konsekvenser. Dette er et af omdrejningspunkterne i bogen "Online Hate and Harmful Content"af Keipi, Näsi, Oksamen, and Räsänen (2017), der blandt andet undersøger, hvad eksponering til online had kan gøre for en persons velbefindende. Bogen understreger, at hadefulde ytringer online har haft døden til følge for bl.a. Jessica Laney, Amanda Todd, Ryan Halligan og David Molak, der alle begik selvmord, angiveligt som konsekvens af online mobning (Keipi et al., 2017, p. 1). I en af undersøgelserne i bogen er forfatterne interesseret i subjektets velbefindende, ved at undersøge den individuelles nuværende livssituation igennem følgende emner: glæde, økonomisk befindende, tillid og selvtillid (Keipi et al., 2017, p. 79). I de undersøgelser, bogen gør brug af, er de adspurgte blevet spurgt om, hvor de ser sig selv på en skala fra 1 (minimalt) til 10 (maksimalt) i de ovennævnte emner (Keipi et al., 2017, p. 80). Studiet viser en markant forskel på to grupper mennesker. Folk der har været udsat for "cyberhate"(had på internettet), og folk der ikke har været udsat for det. Det er bemærkelsesværdigt, at den gruppe, der har været udsat for cyberhate, har rapporteret konsekvent lavere resultater end den modstående gruppe. Specielt er der lavere tal inden for selvtillid og glæde (Keipi et al., 2017, p. 81). Keipi et al. (2017) påpeger, at dem, der er i størst farezone

for at blive påvirket, er de unge, der i forvejen har det svært, og som måske i mindre grad har ressourcer til at kunne håndtere ubehagelige oplevelser online (Keipi et al., 2017, p. 85).

4.3 Fra problem til løsning

Det er altså klart for os, at der er virkelige konsekvenser som følge af hadefulde ytringer. Dette har vi set i form af undersøgelser fra Keipi et al. (2017), der understøtter, at mennesker, der er udsat for hadefulde ytringer online, har nedsat velfærd. Derudover er der historiske eksempler på unge, hvis liv er endt delvist som følge af onlinehad. Dette giver en indikation om, at der er et håndfast problem, at folk lider under online had, og at det ville være gavnligt at udvikle værktøjer til at løse dette problem. Det er også blevet klart, at det ikke nødvendigvis er problematisk at indsnævre ytringsfrihed på baggrund af dette. Vi har ligeledes set, at der er flere tilgange til at bedømme, hvornår man skal tage aktion i forhold til at begrænse ytringer i form af J.S. Mill's "Harm Principle"og Joel Feinbergs "Offence Principle". Disse vil vi senere kunne bruges til at vurdere, hvordan vores maskinelæringsmodel kategoriserer sætninger.

5 Terminologi og anvendte metoder til maskinlæring

Gennem projektet har vi arbejdet med maskinlærings biblioteket 'Keras', som udstiller en række prædefinerede funktioner. Keras kan bruges til at definere og træne en stor variation af maskinelæringsmodeller herunder arbejdet med NLP. Biblioteket er brugervenligt og nemt at implementere, samt giver mulighed for at begynde en maskinlæringsproces uden at have videre matematisk forståelse for hvordan neurale netværk fungerer (Chollet, 2017, p. 61). I dette afsnit vil vi dog give en generel forståelse for terminologien og de anvendte metoder vi har benyttet os af.

Afsnittet tager udgangspunkt i kerne elementerne for en maskinlæringsmodel og starter med at beskrive typen af model vi arbejder med. Efterfølgende beskrives hvordan sådan en type af model modtager input og hvilke tiltag der skal gøres for at modellen kan forstå input i form af kommentarer. Derefter beskrives hvordan ord kan indeles i et opslagsværk til at danne semantiske relationer. I afsnittet omkring dense lag, beskrives hvordan neuroner er forbundet og hvordan modellens output defineres. Med en gennemgang af den centrale teori til den maskinlæringsmodel vi arbejder med, beskrives hefter hvordan neurale netværk kan optimeres og hvilke processer der er relevante at gennemgå. Afslutning vises beskrives hvordan data håndteres og hvilke tiltag der er gode at tage i forbindelse med et maskinlærings projekt.

5.1 Recurrent Neural Networks

Der findes flere overordnede typer af neurale netværksmodeller, som almindeligvis bliver anvendt. Generelt fungerer neurale netværk ved, at input bliver kørt igennem modellen som

enkeltstående datapunkter. Processeres der eksempelvis billeder, giver det mening, at behandle et billede ad gangen. Dette er dog et problem, hvis man befinder sig i et domæne, hvor det er afgørende, hvordan elementer af input er organiseret i forhold til hinanden. Dette er især gældende ved behandling af naturlige sprog og tale (Chollet, 2017, p. 196). Da vi netop arbejder med naturlige sprog, er det vigtigt, at vores model processerer sætninger sammenhængende. Derfor har vi valgt at arbejde med Recurrent Neural Network (RNN), altså et tilbagevendende neuralt netværk. RNN er typisk anvendt ved NLP problemstillinger, fordi naturlige sprog skal processeres sammenhængende. Grunden til, at RNN oftest bliver anvendt til NLP, begrundes af den måde RNN arbejder med sekvenser af elementer og processerer samlede sekvenser. Lad os antage sekvensen $x =$ "I study at Roskilde University". Så vil denne sætning blive brudt op i følgende bidder, hvor hvert ord repræsenteres med specifikke ord-vektorer.

x_1 = "I", x_2 = "study", x_3= "at", x_4 = "Roskilde", x_5 = "University"

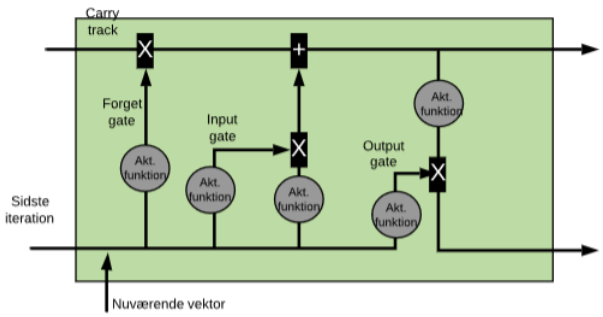
I stedet for at hver vektor bliver processeret igennem modellen hver for sig, sker processeringen iterativt per sekvens. Dermed kan ords sammensætninger tages i betragtning, som må siges at være vigtig i arbejdet med sprog (John McGonagle, n.d.).

Long-Short-Term-Memory

LSTM (Long-Short-Term-Memory) er en RNN-type som er velegnet til klassificerings problemer. LSTM differentierer sig fra typiske RNN, som er stærkt disponeret for *"The vanishing/exploding gradient problem"*. Dette problem opstår ved backpropagation, hvor forbindelserne mellem neuronernes vægte bliver opdateret. Her kan der enten ske det, at eventuelle tendenser i gradienten vil blive forstærket i takt med, at netværkets vægte bliver opdateret baglæs gennem lagene. Her kan en gradient som er en smule for høj resultere i vægte, der eksploderer, eller på den anden side kan en lav gradient resultere i vægte, der bliver for små, og derfor ikke kan bruges til at forudsige noget (Chollet, 2017, p. 202). Denne problematik opstår ofte, når man vil anvende RNN på tekst, da ord, der i deres mening ofte er tætte på hinanden, kan være i hver deres ende af en lang sætning. Derfor vil analyser på meget lange sætninger resultere i mange lag i modellen, hvilket vil gøre backpropagationen usikker (John McGonagle, n.d.).

LSTM-modellen kan netop undgå disse to faldgruber, da modellens neuroner har en anderledes arkitektur. LSTM løser problemet med de lange sekvenser ved tilføjelsen af et såkaldt CT (Carry Track).

I figur 4 fremgår det hvordan CT'et løber igennem cellen, og hvordan den er forbundet til det aktuelle sekvenselement ved hjælp af 'gates'. CT'et løber igennem hver enkelt neuron og gemmer information fra den nuværende sekvens således, at hvis et relateret ord viser sig senere i sekvensen, så kan ord, der er placeret lang fra hinanden, associeres med hinanden. CT har tre forskellige 'gates'; 'forget gate', 'input gate' og 'output gate'. I disse 'gates' bliver det besluttet, hvorvidt den nuværende vektor, som består af information fra sidste gennemløb, og en vektor som repræsenterer det nuværende element i sekvensen,



Figur 4: LSTM celle
(Kumar, 2020)

skal gemmes i det mere permanente CT eller ej. Sammenlagt bliver disse to informationer passeret igennem hver deres gate, som består af non-linære funktioner. Ved at evaluere på outputtet af disse 'gates' kan man ved hvert ord vurdere, hvorvidt et ord skal gemmes globalt. I det, der hedder 'output gate', bliver den information, som vi har i vores CT, taget med i cellens output, ved at CT's værdi bliver multipliceret med med $(x_i + x_i - 1)$. Dermed kan vi holde på noget global viden, som kan blive båret rundt i behandlingen af alle vektorer i alle sekvenser (Chollet, 2017, p. 202-207).

5.2 Vektorer som input

Når en bestemt type observation skal videregives til et neuralt netværk, kan man ikke blot tage observationen, som den er, og direkte videregive den. Man er derimod nødt til at repræsentere data på en måde, som modellen forstår. Dette kan gøres med vektorer (Chollet, 2017, p. 180). Vektorer er simpelt set en liste med værdier, som korrelerer med en bestemt attribut i den relevante observation. En vektor, som indeholder information om kun én attribut, kaldes en endimensionel vektor af den årsag, at den kun indeholder et tal. Hvis en vektor derimod indeholder tal, som beskriver to attributter, så er den todimensionel, tre tal giver en tredimensionel vektor, etc. Vektorer bliver ofte repræsenteret grafisk i et koordinatsystem som pile, der peger i en retning og kan have n dimensioner (Chollet, 2017, p. 35).

I dette projekt har vi brugt en indkodningsteknik, der går ud på, at hvert ord fra vores ordbog (alle ord som indgår i datasættets sætninger) svarer direkte til en numerisk værdi. På den måde kan hvert ord repræsenteres som en endimensionel vektor. Når en sætning eller et datasæt så processeres, skal alle ordene i sætningerne konverteres til den numeriske værdi, som står ud for det pågældende ord i vores ordbog (van Aken, Risch, Krestel, & Löser, 2018, p. 1-4). Denne metode er mere pladseffektiv end andre teknikker og gør det også lettere at repræsentere rækkefølgen af ord i sætninger. Et problem ved teknikken er dog, at de ord, som i ordbogen står ved siden af hinanden, ikke nødvendigvis er semantisk eller kontekstuelt relaterede, og hvis ord, som er nært beslægtede, ikke har vektorværdier, som er tæt på hinanden, så vil dette have en negativ effekt, når vi træner vores model. Dog

kan dette undgås ved at indlejre ord gennem metoden *Word embedding*. Metoden fungerer ved, at man træner et neuralt netværk til at associere ordene. Netværket tager et datasæt med 2D tensors som input, og associerer hvert indeks i vores ordbog med en vektor af en højere densitet, som mere præcist kan beskrive et ords relative placering (Chollet, 2017, p. 186).

5.3 Word-Embedding

Word-Embedding (WE) bliver udført af forskellige årsager, men oftest for at reducere arbejdet, der ligger i et opslagsværk, der er blevet one-hot encoded. Her skaber man binære vektorer, der er i mange dimensioner (Chollet, 2017, p. 181). Dette gøres, for at inddele ord i et oplagsværk med "one-hot encoding", ved at omdanne ordet til en vektor, der har lige så mange dimensioner, som der er ord i opslagsværket. Denne vektor vil have n dimensioner, og alle elementerne i vektorerne vil være 0 undtagen et nummer, der vil være 1. Dette vil således resultere i n vektorer, der har n dimensioner (Chollet, 2017, p. 181). Alternativt kan man, ved at anvende WE, komme ned på markant færre dimensioner. Dette opnår man ved at lære sin WE direkte fra ens træningsdatasæt. WE findes ofte i 256, 512 eller 1024 dimensioner, i tilfælde hvor ordforrådet, som bliver brugt, er meget stort. Når man bruger one-hot word-encoding, er det ikke unormalt at sidde med vektorer på 20.000 dimensioner eller mere (Chollet, 2017, p. 184). Når WE anvendes, er en mulig tilgang, at man sammensætter WE med modellens primære opgave, og så lærer WE i takt med, at vægtene i netværket bliver trænet (Chollet, 2017, p. 184). Først vil man typisk initialisere de vektorer, som skal associere til hvert ord, tilfældigt. Dette er dog ikke nemt for neurale netværk at finde rundt i. Derfor træner modellen således, at de geometriske figurer i vores vektor-rum vil give semantisk mening (Chollet, 2017, p. 185). Efter WE er foregået, kan outputtet være af en dimensionalitet, der ikke er kompatibelt med det input, som vores netværk forventer. I det tilfælde kan man bruge en metode, der hedder *'pooling'*. Pooling bruges ofte til at kunne nedskalere en tensor (som er en liste med n dimensioner) fra n dimensioner, til $n-1$ dimensioner eller mere. På den måde kan pooling både bruges til at gøre input mere pladseffektivt, men også til at kunne give den rigtige type som modellen forventer (Chollet, 2017, p. 127-130).

5.4 Dense lag

I vores model anvender vi såkaldte *'Dense lag'*. Et lag, der er tæt eller 'dense', er den mest almindelige type lag i neurale netværk. I et dense lag er alle input neuroner for laget direkte forbundet med alle output-neuroner for outputlaget. Det vil sige, at når modellen propagerer fremad, skal flere inputs fra et lag altså kombineres til én værdi. Til dette bruger man en aktiveringsfunktion, som kan tage flere input, og give et output til neuronet i det næste lag. Her kan anvendes både lineære og ikke-lineære funktioner (Chollet, 2017, p. 320).

Aktiveringsfunktioner i dense lag

Neurale netværk består af neuroner, som originalt er inspireret af den menneskelige hjerne. Disse neuroner skal aktiveres og sende et signal videre i nettet, ellers skal de ikke aktiveres og derved slukkes. Da output fra en neuron sjældent er netop 1 eller netop 0, er det brugbart at gøre brug at matematiske funktioner, der manipulerer output til at give et tal der eksempelvis er tættere på 0 eller 1. Dette er hvad aktiverings funktioner kan, de er tilknyttet til hver neuron i neurale netværk og normaliserer et input alt efter hvilken aktiverings funktion der anvendes. Aktiveringsfunktioner fungerer som et matematisk led mellem input til en neuron og output til næste lag. Aktiveringsfunktioner kan være så simple som at slukke neuronens output fra og til. Men funktionerne kan også fungere ved at mappe input signaler til output signaler der skal bruges for at det neurale netværk kan fungerer (Nwankpa et al., 2018, p. 3-6).

Linære aktiveringsfunktioner

Linære aktiveringsfunktioner tager form af en linær funktion, deraf navnet. Funktionen tager input og ganger det med vægtene for hver neuron og videregiver derefter output. Ved at gange neuronerne kan man altså få forskellige outputs. Dog er det ikke muligt, at bruge backpropagation i linære funktioner, da funktionen er konstant og ikke har nogen relation til input (Nwankpa et al., 2018, p. 4-5).

Ikke-linære aktiveringsfunktioner

Modsat linære aktiveringsfunktioner, muliggøre ikke-linære aktiveringsfunktioner at modellen kan mappe input mellem netværkets input og output. Derfor er det muligt backpropagere, da funktionen er en afledt funktion som er relateret til input. Derudover tillader ikke-linære aktiveringsfunktioner at stable lag oven på hinanden, som er et behov når modellen skal lære komplekse datasæt. Indenfor ikke-linære aktiveringsfunktioner findes en række typiske funktioner som ofte anvendes. Her vil vi kort beskrive de to som vi har brugt i forbindelse med dette projekt.

- **Sigmoid** givet et ouput mellem 1 og 0 samt normalisere outputtet for hver neuron. Sigmoid er kendt for at give klare forudsigelser, da det bringer den outputtede værdi meget tæt på 1 eller 0. Funktionen er dog kendt ved problematikken omkring "Vanishing gradient"hvor meget høje eller meget lave værdier af x ikke ændres og dermed resultere i at modellen ikke lærer videre eller er for langsom til, at nå til nøjagtige forudsigelser (Nwankpa et al., 2018, p. 5-7).
- **ReLU** er kendt for at være en hurtig aktiveringsfunktion, da den ligner og agere som en linær funktioner men faktisk er en afledt funktion, så der er mulighed for backpropagation (Nwankpa et al., 2018, p. 8).

5.5 Optimering af neurale netværk

Noget af det, der er essentielt for, at ens neurale netværk kan "lære", er, hvordan ens vægte er sat. Disse vægte er matricer, der indeholder information, som ens model har lært ved eksponering til det træningsdata, der er valgt. Vægtene i en model er som udgangspunkt initialiseret med tilfældige værdier - hvilket ikke har meget værdi i sig selv. Modellen skal være i stand til at ændre på disse vægte, og det er denne justering af modellens vægtmatricer, der kaldes for træning (Chollet, 2017, p. 46). Når man skal træne sin model, bliver det af Chollet (2017) delt op i fire trin, som skal gentages så længe, det er nødvendigt.

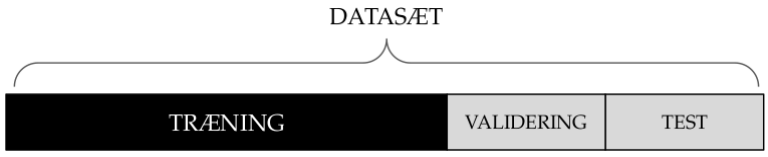
1. Lav en samling af træningsprøver x og tilsvarende y.
2. Kør modellen på x og kom med forudsigelser om y.
3. Find ud af hvor stort et mismatch der er imellem forudsigelserne og det faktiske resultater. Modellens loss-funktion.
4. Opdater alle vægtmatricerne i modellen, så modellens mismatch bliver mildt reduceret.

(Chollet, 2017, p. 46).

Denne proces kan se magisk ud, men når det kommer til stykket, er det en samling af tensoroperationer, der foretages, som giver det ønskede resultat.

5.6 Håndtering af træningsdata

Når data er indsamlet, vil det oftest ligge i én fil (f.x. csv), som man refere sin kode til at læse fra. Der er dog behov for en opdeling af data, så nøjagtigheden af en models kategorisering kan måles. En typisk opdeling vil foregå i tre grupperinger; et trænings-, valideringsog testsæt. Af figur 5 fremgår en sådan gruppering, hvoraf mængden af data i de enkelte grupperinger er bestemt af formålet.



Figur 5: Datasæt inddelings eksempel

Intervallet, der hedder træning, bruges til at træne modellen. Ved træningen tilpasser modellen sig træningssættet. I den superviserede model vil dette træningssæt altså bestå af data, der er blevet tildelt labels, og modellen vil heraf vide, hvilke kategorier dataen skal deles op i (Cios, 2007).

Modellen træner med træningssættet, hvoraf valideringssættet anvendes ved hver epoke i træningen, og validerer modellen løbende. Fra gennemløbet med valideringen bliver lossgradienten udregnet. Ved hjælp af et "loss"-indeks kan man få en ide om, hvor godt en

model fungerer på andet end træningssættet. Efter modellen er trænet, videregives testsættet til modellen. Her er det vigtigt, at modellen ikke kender observationernes klasser, og at testsættet altså er fri for klasser. Ud fra testsættet kategoriserer modellen altså input, og giver et bud på en klassificering ud fra det, den har lært fra træningssættet og valideringssættet (Chollet, 2017, p. 97-98).

6 Typiske problemstillinger i maskinlæring

Følgende afsnit vil adressere et udsnit af fejlkilder, og hvilke tiltag der kan foretages til at håndtere disse. Først beskrives begrebet overfitting og hvordan denne problematik kan håndteres. Derefter beskrives, hvordan et dårligt balanceret datasæt kan justeres til at give modellen lige præmisser at træne på.

6.1 Overfitting

En af de mest almindelige fejlkilder betegnes ved *overfitting*. Ved overfitting forstås, at modellen tilpasses træningsdatasættet i for høj grad. Denne grundlæggende udfordring spænder mellem optimering og generalisering. Optimering er den proces, der foregår, når modellen tilpasses for at opnå bedst ydeevne på vegne af træningsdata. Generalisering refererer til den trænede models ydeevne på ukendt data. Det er dog kun muligt at tilpasse modellen på vegne af træningsdata, og det er derfor ikke en mulighed at tilpasse generalisering.

Overfitting handler altså om, at modellen er for optimeret til et bestemt datasæt. Det modsatte kan dog også være et problem. I dette tilfælde vil modellen være for generaliseret. Ved generalisering har modellen ikke identificeret alle relevante mønstre, og modellen vil betegnes med begrebet underfitting. Når træningsdata har været igennem tilpas mange epoker eller iterationer af træningssættet, stopper træningen med at være gavnlig. Modellen vil begynde at overfitte og finde mønstre, som er specifikke for det enkelte datasæt, men som vil være irrelevante eller vildledende for nye datasæt.

For at undgå at modellen begynder at lære fra metrikker, der ikke er relevante for datasættet, er det vigtigt at gøre sig overvejelser omkring overfitting. Her er der en række tiltag, som kan foretages for at undgå fejlkilderne. Herved foretager man altså regulering af træningsdataen (Chollet, 2017, p. 107).

At diagnosticere og undgå over- og underfitting

En god indikator på, at en model overfitter, er, at vi kan observere gode metrikker på træningsdata, men så snart et datasæt valideres, som modellen ikke er trænet på, begynder metrikkerne at blive dårligere (Chollet, 2017, p. 107-109). Ved overfitting sker det altså, at modellen er så specifikt tilpasset træningsdataet og mønstrene heri, at den fejlkategoriserer, så snart observationerne kommer fra et andet sæt.

Det mest typiske indgreb, man kan gøres sig i kampen mod overfitting, er at øge mængden af træningsdata. Hvilket vil medføre flere mønstre, der kan afdækkes, og mere diversitet i sættet. Modellen vil altså være i stand til at generalisere bedre med mere data. Hvis det ikke

er en mulighed at øge træningssættet, og man opererer med et begrænset sæt, kan man også begive sig ud i at begrænse størrelsen af sin model (Chollet, 2017, p. 104-105). En model med færre lag, og færre neuroner per lag, vil også have en mindre hukommelseskapacitet, som den kan huske meget specifikke mønstre med.

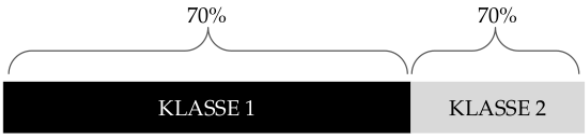
Dropout er en anden metode, hvorpå man kan reducere overfitting, og går som udgangspunkt ud på, tilfældigt at nulstille output resultater. Dette vil sige, at hvis man tilføjer dropout til et lag i sit neurale netværk, så vil en del af outputtet, alt efter hvor stor ens dropout er, blive nulstillet (Chollet, 2017, p. 109). Nulstilling af output kan reducere overfitting af netværket ved at introducere støj, der vil forebygge konspirationer. Konspirationer er tilfældige ikke signifikante mønstre, som netværket ellers ville begynde at memorere (Chollet, 2017, p. 110).

6.2 Metoder i kampen mod et dårligt balanceret datasæt

En anden fejlkilde, som kan opstå ud fra datasættet, handler om, hvorvidt sættet er balanceret. Hvis vi står med et binært klassificeringsproblem, hvor vi skal klassificere mellem to klasser, så ville det ideelle scenarie være, at man havde lige mange observationer af hver klasse, således at hver klasse var repræsenteret lige godt, eksemplificeret i Figur 6.

Figur 6: Datasæt 50/50 eksempel

Med dette ideelle datasæt vil modellen træne lige meget på hver af de to klasser. Datasæt fra den virkelige verden er dog sjældent perfekte, og vil ofte være dårligt balancerede. Fordelingen af et datasæt vil dermed ofte være langt dårligere balanceret (Chollet, 2017, p. 104-107).



Figur 7: Datasæt 70/30 eksempel

Et datasæt kan f.eks. have en balance som i figur 7, her er den første klasse stærkt overrepræsenteret. Hvorvidt, dette er et problem eller ej, afhænger af formålet for modellen. Såfremt målet for modellen er at kunne klassificere hen imod den dårligst repræsenterede klasse, så kan det være et problem, at en model hovedsageligt træner på den klasse, som er mest forekommende. Når dette er tilfældet, er der nogle forskellige teknikker, man kan anvende for at korrigere. Det er dog kun i nogle tilfælde, at dette vil være meningsfuldt. Et

eksempel kunne være en model, som har til formål at finde fejl i et vilkårligt system. Hvis datasættet, som denne model skal trænes med, repræsenterer virkeligheden, så vil observationer, som tilhører klassen for fejl-instanser, højst sandsynligt være kraftigt underrepræsenteret. Når man så kører tests, vil man kunne observere en høj succesrate, eftersom modellen blot vil rapportere instanserne som værende tilhørende klassen, som ikke har nogen fejl. I det tilfælde er det et stort problem, at fejl-klassen er underrepræsenteret. Med andre ord så giver det bedst mening at korrigere for en minoritets-klasse, når klassen er vigtig (Seif, 2018).

Balancering af vægte

En typisk teknik til korrigering er at justere på den vægt, som hver af klasserne bærer på i træningen af modellen, dette vil føre til en større eller mindre effekt på tabs-indekset, alt efter hvordan vægtene er justeret. Hvis vi tager udgangspunkt i figur 7, hvor klasse 1 er overrepræsenteret med tyve procentpoint, kan vi opjustere vægten for klasse 2, således at vi udligner forskellen mellem de to (Seif, 2018). I Keras kan det gøres ved at definere et associativt array, eller dictionary, som det hedder i Python, på følgende måde:

```
>>> class_weight = {1: 1.0, 2: 1.66667}
```

Med denne vægtning vil vores anden klasse altså "vægte" mere i udregningen af tabs-indekset, og vi har nogenlunde korrigeret for forskellen. Efter vægtningen er italesat, kan vi blot passere det associative array som et argument i den metode, som træner modellen, og således vil disse vægtninger være initialiseret i vores model med vores valgte værdier.

```
>>> model.fit(X_train, Y_train, epochs=10, batch_size=32,
class_weight=class_weight)
```

Med denne metode kan man altså sætte en højere træningsintensitet på den del af træningssættet, som er dårligt repræsenteret (Seif, 2018).

7 Maskinlæringsproces med CRISP-DM

I dette afsnit beskrives konstruktionen af vores egen maskinlæringsmodel, som klassificere kommentarer alt efter hvor hadefulde de er. Afsnittet beskrives gennem faserne fra CRISP-DM projektprocessen, herunder hvilke overvejelser, opdagelser og konklusioner vi har kunne drage fra hver af faserne. Først opsummeres vores arbejde, med en forståelse for problematikken omkring hadefulde ytringer og censur, hvilket leder til en definition af et maskinlæringsproblem. Efterfølgende beskrives hvordan vi har dannet forståelse for den data vi arbejder med og hvilke ting vi har været opmærksomme på, grundet denne forståelse. Efter en gennemgang af dataforståelse, beskrives hvordan vi har forbehandlet vores datasæt, fasen har primært fokus på at håndtere data på en måde som modellen kan forstå.

I fasen modellering, beskrives hvordan den egentlig model er konstrueret, hvor vi tager afsæt i vores viden fra afsnit 5 & 6. Til sidst evalueres modellen og fremtrædende faktorer opsummeres i en delkonklusion, som tages med videre i anden iteration af modellen.

7.1 Forståelse for problematikken

Første step i CRISP-DM er at forstå problematikken, som vi ønsker at løse med vores maskinlæringsmodel. I afsnit 4 fandt vi ud af, at ytringsfrihed ikke er ultimativ, og at det er essentielt at sætte nogle grænser for ytringer, også i online rum. Her foreslår The Harm Principle, at det kun er ytringer, der har skade på andre som direkte konsekvens, der bør censureres. Det mere begrænsende Offence Principle foreslår, at ytringer, som kan krænke eller forarge andre og samtidig er uundgåelige, bør censureres. Dette er dog ikke optimalt både da krænkelse og forargelse varierer fra person til person, men ligeledes fordi ytringer på sociale medier altid er uundgåelige i og med, at man blot kan logge af mediet. Samtidig er det ikke i mediernes interesse at brugere fravælger at deltage i en debat, fordi andre brugere benytter hadefulde ytringer.

Vi har igennem vores undersøgelser fundet frem til, at hadefulde ytringer på sociale medier er et problem, da disse kan skabe et ekskluderende og intimiderende miljø, som svækker fællesskabet, opbygger stereotyper og potentielt skader demokratiet. Desuden kan det også have konsekvenser for den individuelle, der udsættes for hadefulde ytringer, som generelt oplever mindre selvtillid og livsglæde, hvilket potentielt kan føre til selvskade og selvmord.

Vores maskinlæringsmodel skal altså kunne læse og identificere forskellige typer af hadefulde ytringer, så den potentielt kunne implementeres på et socialt medie. Succeskriterierne for vores model er , at den kan identificere hadefulde ytringer blandt mange ytringer, et såkaldt multiklassificerings problem. Problemet, vores model skal løse, beror sig dermed på at kategorisere, hvad der er hadefuldt. Herunder forventes det, at disse klassificeringer kan kategoriseres med førnævnte principper. Vi bestræber os altså efter en model, som direkte kan klassificere forhold som må betegnes som hadefulde ytringer.

7.2 Dataforståelse

Når maskinlæringsproblemet er defineret, skal der tages et kig på det data, der arbejdes med. Dette gøres først og fremmest for at sikre sig, at det er den korrekte data man arbejder med, men også for at se hvordan dataet er bearbejdet. I denne rapport tages der udgangspunkt i et datasæt fra Kaggle.com, der indeholder observationer, bestående af kommentarer fra Wikipedia. Observationerne i vores datasæt falder indenfor seks klasser, der alle er forskellige typer af hadefulde ytringer; 'toxic', 'obscene', 'insult', 'identity hate', 'severe toxic' og 'threat'. Herunder kan threat (trussel), som den mest åbenlyse, føres under The Harm Principle, idet en anden persons sikkerhed trues. Selvom skade ikke nødvendigvis er en direkte konsekvens af trusler, så ligger der stadig en sikkerhedsmæssig overvejelse i begrundelsen for at fjerne denne type kommentarer, idet det først kan siges, at der var skade som konsekvens af ytringen, efter skaden er sket. Derudover kan det tænkes at opfordring til selvskade

også må høre inde under dette princip. Denne type kommentar kan findes i datasættet under flere forskellige kategorier. Følgende er et eksempel på en sådan kommentar: *"(...) do the world a favour and kill yourself, ok? the world will be a better place without you. or if you can't bring yourself to do it, get someone else to do it, see if i care. (...)"* (train_csv, id: a04b778673ce16fe). Denne type kommentarer kunne også tænkes at blive censureret på baggrund af The Harm Principle. Tilsluttede vi os kun The Harm Principle, ville vi kun kunne argumentere for, at det kun var meget grove kommentarer bestående af trusler og opfordringer til selvskade, der bliver censureret, da de resterende kommentarer sjældent kunne tænkes at medføre direkte skade. Tilslutter vi os derimod The Offence Principle og ser bort fra princippet om, at det kun er ytringer, der er uundgåelige der censureres, vil vi kunne kategorisere og potentielt censurere langt flere hadefulde ytringer.

Rent praktisk indeholder datasættet 223.549 observationer, der som nævnt er opdelt i seks hadefulde klasser og en klasse; 'clean', som er de observationer, der ikke slår ud på noget negativt indhold. Datasættet er frit tilgængeligt og kan hentes på `kaggle.com`. Her findes 'train.csv' som er træningsættet vi har arbejdet med, 'test.csv' som er testsættet vi anvender vores model på og 'test_labels.csv' som har referencer til testsættet med klassificeringer. Nedenfor i tabel 1 kan en opgørelse over fordelingen af kommentarer i de forskellige klasser i sættet ses.

clean	201.081
toxic	21.384
severe toxic	1.962
obscene	12.130
insult	2.117
threat	689
identity hate	2.117

Tabel 1: Fordeling af kommentarer

Da vi arbejder med et offentligt tilgængeligt datasæt, har vi ikke selv struktureret den indsamlede data. Det er dermed en fordel, at kunne betragte den data vi arbejder med i *Jupyter notebook*, da store datasæt kræver en del af computeren at håndtere. Ved at kalde nedenstående funktion fremvises hovedet af datasættet. Hovedet indeholder de fem første datapunkter med dertilhørende attributter.

```
>>> train_csv.head()
```

I tabel 2 fremgår et udsnit af det output, som sendes tilbage til os efter, funktionen er kaldt. Her kan vi først læse indekseringen af de forskellige kommentarer. Indekseringen starter ved nul og går op til fire, altså de første fem datapunkter. Efterfølgende vises et id på den specifikke kommentar, men også starten af kommentaren. Vi kan altså læse os frem til, at datsættet er indekseret, hver kommentar har et id, og vi kan læse starten af kommentarerne.

	id	comment_text	toxic	severe_toxic	obscene	threat
0	0000997932	Explanation\nWhy the edits made...	0	0	0	0
1	000103f0d9	D'aww! He matches this backgro...	0	0	0	0
2	000113f07e	Hey man, I'm really not trying to...	0	0	0	0
3	0001b41b1c	"\nMore\nI can't make any real s...	0	0	0	0
4	0001d958c5	You, sir, are my hero. Any chance...	0	0	0	0

Tabel 2: Et første kig på data

Derudover bliver vi gjort bekendt med, at kommentarerne er på engelsk og har attributterne *"toxic"*, *"severe_toxic"*, *"obscene"*, *"threat"* etc.

Vi skal nu angive, hvilken data der skal bruges til træning, og hvilken der skal bruges til at teste på. Her er viden omkring de forskellige attributter vigtig, da vi skal kunne angive til modellen, hvor selve kommentarerne er, og hvor de andre attributter er.

```
>>> X = train_csv['comment_text']
>>> y = train_csv[['severe_toxic', 'obscene', 'threat', 'insult',
'identity_hate','clean']].values
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)
```

For at kunne tilgå den enkelte værdi, som attributterne indeholder, gemmes disse i ovenstående variabler. Her gemmes altså selve kommentaren i en x værdi til x-aksen og labels i en y værdi til y-aksen. Til sidst splittes træningsættet op, så vi har 80 % træningsdata og 20 % test data.

```
>>> y_train.shape
(159571, 6)
```

Ved at tjekke formen kan vi se, at vi har 159571 datapunkter med 6 y værdier, hvilket stemmer overens med det, vi ved om datasættet. Vi betragter altså 6 forskellige kategorier, som kommentarerne kan klassificeres som. Det er dog vigtigt at nævne, at en kommentar godt kan være medlem af flere af klasserne. De 6 forskellige forhold vil blive mærkeret med 0 og 1 alt efter, om de passer til klassificeringen.

7.3 Dataforbehandling

Grundet at maskinlæringsalgoritmer, som udgangspunkt, kun forstår heltal eller numre, bliver vi nødt til at forbehandle vores data. Kommentarerne er repræsenteret med tekst og ikke som numre eller heltal, derfor bliver vi nødt til at konvertere tekst, så modellen kan arbejde med den. Derudover er indeholdet af kommentarerne af forskellig længde, mens modellen forventer input af samme længde, og derfor skal kommentarerne også konverteres til samme længde. Først skal hvert ord dog deles op i sin egen unikke tekststreng, så modellen kan

betragte de enkelte ord i en kommentar. Dette gøres ved at splitte ord op *(eng. tokenize)*. I sidste afsnit gemte vi kommentarer i en *x* variabel, som nu kan bruges til at lave en liste af ord.

```
>>> max_features = 2000
>>> tokens = Tokenizer(num_words=max_features)
>>> tokens.fit_on_texts(list(X_train))
>>> tokenized_train = tokens.texts_to_sequences(X_train)
>>> tokenized_test = tokens.texts_to_sequences(X_test)
```

Først sættes det maksimale antal af ord i en sætning,som vi ønsker at gemme. Det vil derfor kun være de 2000 hyppigste ord, som modellen gemmer i sin hukommelse. Her kan andre faktorer som f.eks. filtre også defineres. Herefter gemmes de hyppigste ord i en variabel tokens, og til sidst bliver ordene konvereret til en sekvens af heltal.

Padding af kommentarer

Med hver kommentar splittet kan vi nu få et indblik i, hvordan modellen vil betragte datasættet som heltal i stedet for tekst. Her kan en kommentar fra *x* variablen på hvilket som helst index printes, hvorefter den opdelte og konverterede kommentar kan printes som i nedenstående.

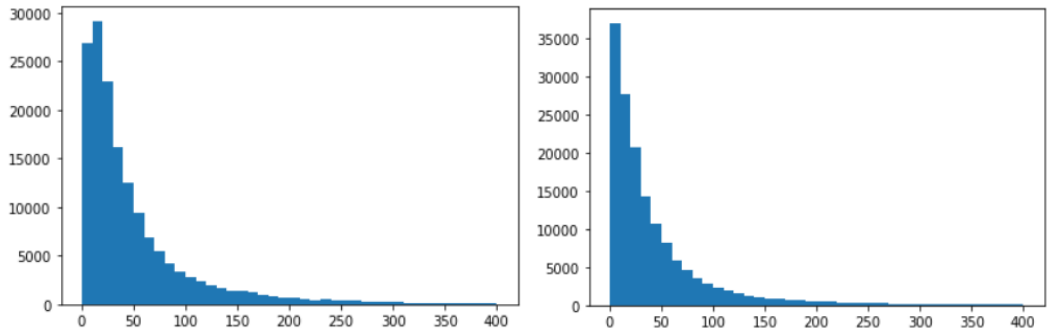
```
>>> print(X[0])
>>> print(token_test[0])

Of course, it is always a pleasure to see you.
[3, 438, 11, 8, 382, 5, 3940, 2, 63, 6]
```

Her kan man se, at "Of"er angivet som "3"og "course"som "438". Men det er stadig vigtigt at angive en fælles længde for kommentarer. Det er vigtigt at være opmærksom på, at en kort maks længde kan føre til at vigtige ord udelades fra en kommentar, mens en lang maks længde kræver mere computer kraft. Det kan derfor være en fordel at betragte hele datasættet og længden af kommentarer, hvilket kan gøres ved at løbe igennem listen af kommentarer og arrangere dem i et søjlediagram.

```
>>> total = [len(comment) for comment in tokenized_train]
>>> plt.hist(total,bins = np.arange(0,410,10))
>>> plt.show()
```

Outputtet fremgår af figur 8a & 8b som søjlediagrammer. Her kan vi se, at der er en del kommentarer med længden 0-50 og meget få over 200. Det betyder altså, at vi skal vælge en længde, så vi er sikre på at fange meningen af kommentaren, men vælger vi en længde på 50 vil dette nok ikke være tilfældet, hvorimod vælger vi en længde på 200, vil alle



(a) Test (b) Train

Figur 8: Data ordlængde

kommentarer ikke være lige lange. Derfor benyttes metoden padding, hvor de kommentarer, som ikke har en længde lig med maks længden, bliver udfyldt med 0 værdier. Dermed sikre vi os, at alle kommentarer er lige lange, uden at for meget af kommentaren går tabt.

Når man benytter sig af padding giver diagrammet i figur 8a & 8b en fin indikator for, hvor vi får mest udbytte ift. computerkraft. Her har vi valgt at bruge en maks længde på 200, som har vist sig at være fint til den cloud-baserede service fra "Kaggle kernals", som vi anvender. Efterfølgende printer vi træningsættet fra indeks 0 til 10 for at bekræfte, at der er sket en padding.

```
>>> max_len = 200
>>> padded_train = pad_sequences(tokenized_train, maxlen = max_len)
>>> padded_test = pad_sequences(tokenized_test, maxlen = max_len)
>>> padded_train[:10]
array([[  0,   0,   0, ..., 143,  73,  89],
       [  0,   0,   0, ...,  94,  38, 182],
       [  0,   0,   0, ...,   1,  93,   1],
       ...,
       [  0,   0,   0, ...,   6,  44,  16],
       [  0,   0,   0, ..., 159,  21,  10],
       [  0,   0,   0, ...,   2, 142,   3]], dtype=int32)
>>>
```

Printer vi de første 10 kommentarer, kan vi se at denne padding er udført. Hermed er vi færdige med at forbehandle vores trænings- og testsæt, og kan gå i gang med at skabe vores model og de individuelle lag.

7.4 Modellering

I modelleringsfasen startes der med at definere input. Som vist tidligere tager modellen sætninger, som er vektoriserede med en max længde på 200 ord.

```
>>> input_model = Input(shape = (max_len, ))
```

Herefter skal embedding laget passere ord til de definerede vektorpladser, som er afhængige af afstanden til andre ord i en sætning. Dette gøres ved først at definere vektor rummet med variablen *embed_size*, hvilket gøres for at reducere modellens størrelse og dimensioner. Outputtet af embedding laget er en liste af tal, der indikerer, hvor ord i vektor rummet kan findes. Afstanden mellem koordinater repræsenterer semantisk relation f.eks. mellem 'gul' og 'grøn', fordi begge er farver.

```
embed_size = 128
>>> x = Embedding(max_features, embed_size)(input_model)
```

Efter embedding laget defineres det, hvordan tensorer passeres til LSTM laget. Det første argument er *"units"*, som angiver formattering af hver LSTM celles vektor-output. I andet argument defineres den ønskede output sekvens, som er (*Parti størrelse*, *Iterationer*, *Antal input*).

```
>>> x = LSTM(60, return_sequences = True, name='lstm_layer')(x)
```

Herefter benyttes et *Global Max pooling lag*, for at reducere størrelsen af data. Dette gøres todimensionelt, da vores efterfølgende lag forventer en 2D tensor. Herefter tilføjes vores første *Dropout lag*, som frakobler noder, så de næste lag dermed skal forholde sig til den manglende data. Dette gøres for at opnå en bedre generalisering og mindske overfitting.

```
>>> x = GlobalMaxPool1D()(x)
>>> x = Dropout(0.1)(x)
```

Med en 2D tensor og et Dropout lag aktivieret, passeres tensoren gennem et tæt sammenkoblet lag. Først defineres det ønskede output rum, hvorefter en aktiveringsfunktion tilføjes. Først bruges ReLU , som er en hurtig og effektiv funktion, der summerer motivations noden positivt eller negativt, hvilket er en fordel ved binære problemstillinger. Efterfølgende tilføjes endnu et Dropout lag. Og til sidst tilføjes sigmoid-funktionen, som giver et output mellem 0 og 1, alt efter om en kommentar kan klassificers indenfor en klasse.

```
>>> x = Dense(50, activation = 'relu')(x)
>>> x = Dropout(0.1)(x)
>>> x = Dense(6, activation = 'sigmoid')(x)
```

Herefter defineres input modellen og output, og når modellen kompileres sættes loss-funktionen til "binary_crossentropy", da vi ønsker et binært output om hvorhvidt en kommentar klassificeres i de enkelte klasser. Derudover bruges "adam"som optimerings metode til at optimere loss-funktionen. I tilfælde af loss vil "adam"ændre attributer i netværket.

```
>>> model = Model(inputs = input_model, outputs = x)
>>> model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
```

Til sidst trænes modellen. Først defineres *"batch_size"*, som er antallet af sætninger, der bliver kørt før modellen er opdateret. Herefter defineres epoker, altså antallet af gennemkørsler gennem datasættet. Derudover har vi valgt at splitte datasættet med 10 % til validering.

```
>>> batch_size = 32
>>> epochs = 2
>>> History = model.fit(padded_train, y_train, batch_size = batch_size,
epochs = epochs, validation_split = 0.1)
```

7.5 Evaluering

Når modellen bliver trænet med fit-metoden, giver træningen os nogle metrikker, som vi kan starte med at måle vores model på. Modellen trænede på 114.890 observationer, og udførte validation på 12.766. Her er resultaterne fremstillet i tabel 3.

Metrics	**loss**	**accuracy**	**validation loss**	**validation accuracy**
Epoch 1/2	0.0793	0.9758	0.0579	0.9805
Epoch 2/2	0.0556	0.9808	0.0567	0.9808

Tabel 3: Evaluering af træning

Det er vigtigt at nævne, at man med to epoker ikke kan forvente at få de mest præcise metrikker som output, men vi vil alligevel kommentere på de eventuelle mønstre, som denne mængde data præsenterer for os.

Loss for træningssættet starter med at være 0.0793, og bliver formindsket til 0.0556 i anden epoke, og falder altså med 0.0237. Vi kan se det samme mønster i accuracy for træningssættet, hvor præcisionen går fra 97.5%, og stiger med et halvt procentpoint til 98%.

I validation ser forbedringen fra epoke til epoke dog betydeligt mere konservativ ud. Loss falder kun med 0.0012. Accuracy i valideringen bliver ligeledes kun forbedret med 0.0003 procentpoint.

Alle metrikkerne bliver forbedret efter en epoke, dog ser vi en større forbedring fra epoke 1 til 2 hos træningssættet end i validationen. Disse metrikker er vi umiddelbart meget tilfredse med. På disse tal virker det umiddeblbart hverken som om vores model overfitter eller underfitter. Hvis vi havde flere epoker i modellen, kunne man forestille sig at valida-tionssættet ville begynde at performe dårligere.

For i højere grad at sikre os at vores model fungerer som vi vil have det til, skal vi også evaluere på vores testsæt, til dette har keras en funktion der hedder 'evaluate', som kan bruges til netop dette.

Evaluering af testsæt

Testsættet er et udklip af det totale datasæt, som ikke er indgået i træningen af modellen på nogen måde. Når vi skal teste vores model på testsættet, kalder man evaluate på følgende måde, hvor evalueringen foretages med et præprocesseret sæt. Som argumenter gives "x", som er observationer uden labels og "y", som er de labels, der passer til observationerne i "x".

```
>>> metrics = model.evaluate(x = padded_test, y = y_test, verbose = 1)
```

Når evaluate er kaldt, vil vi have et loss og en accuracy for dette testsæt.

Metrikker	**loss**	**accuracy**
Evaluering	0.0559	0.981

Tabel 4: Test på modellen

Disse to metrikker harmonerer umiddelbart fint med metrikkerne fra træningen, og tyder umiddelbart ikke på forekomst af overfitting.

Anvendelse af model

Når modellen anvendes, kaldes funktionen "predict", hvorefter et testdatasæt gives. Predict returnerer ikke samme metrikker som evaluate funktionen, men fungerer netop som en anvendelse der kan køres på et vektoriseret datasæt. Altså klassificere instanser. Det er nu interessant at observere modellens klassificeringer på et datasæt, som ikke er kendt af modellen.

```
>>> pred = model.predict(padded_test,verbose=1)
```

Når modellen har foretaget sin forudsigelse, gemmes output i et array. I dette array kan kommentarer tilgås ved indeksering. Lad os starte med at kigge på en tilfældig kommentar i datasættet.

```
>>> print(X_test[1836])
"truth is linux fanboys are a fucking waste of air.
die all of you mafukas"
```

Kommentaren ved indekseringen 1836 må kunne klassificeres som "toxic", men lad os se hvad modellen siger. Her tilgås indekseringen af kommentaren i den liste som modellen har forudset på.

```
>>> print(pred[1836])
[0.9923614  0.1998196  0.95110023 0.04139983 0.8072933  0.11299966]
```

Da vi bruger en sigmoid-funktion, er værdierne uafhængige af hinanden, og outputtet er en værdi for hver klassificering ['toxic' 'severe_toxic' 'obscene' 'threat' 'insult' 'identity_hate']. Når værdien er mere end 0.5, betyder det, at modellen klassificerer kommentaren i den pågældende kategori. Tættere på 1 betyder en kraftigere klassificering. I eksemplet er det tydeligt, at kommentaren klassificeres som 'toxic', men modellen slår også kraftigt ud på 'insult' hvilket måske er grundet den direkte fornærmelse over for "linux fanboys". Derudover kategoriseres kommentaren som 'obscene', der kan have forskellige betydninger alt efter sammenhæng. I dette eksempel klassificeres kommentaren muligvis sådan grundet "die all of you mufakas", hvilket kan betragtes som 'over grænsen'. Det er selvfølgelig ikke muligt at påpege præcis, hvad modellen slår ned på, men denne sammenligning giver et forslag, hvoraf der er fin sammenhæng mellem klassificeringerne.

```
>>> print(X_test[152637])
>>> print(pred[152637])
What about The Lion King, The Hunchback of Notre Dame and Hercules?
These three films do not belong to Paramount Pictures Corporation,
those are all Disney movies.
[0.01618179 0.00012992 0.00347115 0.00016462 0.00483669 0.00089261]
```

I kontrast til den foregående kommentar er ovenstående i en anden genre. Læser man kommentaren, kan den ikke umiddelbart klassificeres under nogen af de labels, vores model har arbejdet med. I outputtet af forudsigelsen kan man se, at modellen heller ikke klassificere kommentaren under nogen af klasserne, og kommentaren er derfor uden hadefulde ytringer.

Forbedringer

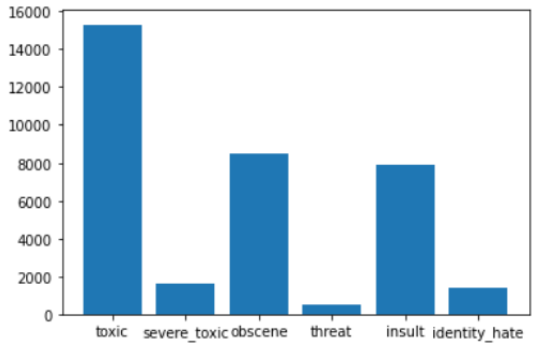
Trods gode metrikker og umiddelbart korrekte klassificeringer, opdager vi dog nogle mønstre i modellen, som kan vise sig at være problematiske.

```
>>> print(X_test[152539])
>>> print(pred[152539])
```

```
His career aspiration is to become a Christian Preacher"" what a faggot!!!
[0.95114475 0.06208223 0.8375022  0.02093048 0.55480343 0.06636483]

>>> print(X_test[152569])
>>> print(pred[152569])
Adrian Hes Gay
[0.9199189  0.1024749  0.6191639  0.02996691 0.5243157  0.09867583]
```

I ovenstående eksempel er der en stærk klassificering ved 'toxic', dog må disse beskeder betegnes som rettet mod seksuelle orienteringer og i vores forstand være klassificeret som 'identity_hate', hvilket ikke er tilfældet for modellen. Vores model er altså hurtig til at klassificere, hvorhvidt en kommentar kan betegnes som 'toxic', hvor andre klassificeringer mister opmærksomhed. Derfor tager vi et kig på distribueringen af træningssættet, og hvordan disse er klassificeret.



Figur 9: Antal klassificeringer pr. klasse

Python gør det heldigvis let for os at konstruere et plot diagram, hvor vi kan se distribueringen af de forskellige klasser. I figur 9 er der en tydelig overvægt af 'toxic' klassificerede kommentarer. Vi vil derfor i næste afsnit beskrive, hvilken betydning det kan have, og hvilke muligheder vi har for at få et bedre udfald på de klassificeringer, der er færre af i træningssættet.

7.6 Delkonklusion

I denne iteration var vi først og fremmest i stand til at opnå en høj accuracy både i validation og test. Det mest dominerede resultat kom som en konsekvens af, at klassen 'toxic' var en stærk majoritets-klasse. Vores prædiktioner havde derfor en meget stærk tendens til at blive klassificeret som 'toxic'. I forlængelse af dette blev de mere grove minoritetsklasser sjælendt klassificeret. Derudover var kategorien 'clean' ikke en klasse, som indgik i klassificeringen, men vi kunne blot observere den ved fraværet af de andre klasser. For fremtiden

kunne det være hjælpsomt at kunne klassificere denne type. Vores model står altså overfor en problemstilling, hvor der ikke er nok data til, at modellen kan finde mønstre i kategorier som 'threat' og 'identity_hate'.

8 Anden iteration

På baggrund af den første iteration har vi i vores anden iteration lavet nogle markante ændringer i vores datasæt og vores model, der på overfladen kunne synes at stride imod vores videnskabelige tilgang til projektet. Disse ændringer har vi besluttet med afsæt i vores teori omkring ytringsfrihed og censur. Når vi ændrer i vores datasæt og er det ikke fordi vi tvivler på, hvorledes vores model kategorisere korrekt, men fordi vi i dette projekt ikke mener, at de valgte klassificeringer stemmer overens med vores problemstilling og hvad vi gerne vil opnå qua principperne i afsnit 4. Den viden, der bliver dannet efter denne ændring er ikke mere eller mindre objektiv.

8.1 Maskinlæringsproblem genbesøgt

Efter at have foretaget den første iteration er der nogle ting, der er blevet klarere for os. Bl.a. at mange af vores prædiktioner slog meget stærkt ud på kategorien 'toxic', hvilket med høj sandsynlighed hænger sammen med, at denne klasse i vores datasæt også er den mest velrepræsenterede af alle kategorierne. Dette kan hænge sammen med, at kommentarerne er multikategoriseret, og at de hver især kan have flere end et mærkat på sig. Da 'toxic' er den nemmeste kategori at ende i, giver det meget god mening, at mange af de kommentarer, der er mere 'seriøse' markeres som "toxic".

Mange af de problematikker vi har undersøgt i forbindelse med hadefulde ytringer, heriblandt "The Harm Principle"og "The Offence Principle"er svært at koble til mange af de kommentarer, der kun er katagorisrert som "toxic". For at en kommentar skal markeres og potentielt fjernes, mener vi, at den skal overtræde en af de to principper, som vi arbejder med. Herunder er det dog svært at se, hvordan de kommentarer, der kun er markeret "toxic", overtræder disse princippers grænser. Derfor er det fremadrettet vigtigt, at vi i større grad fokuserer på de andre klassificeringer. Herunder mener vi, at "threat", "obscene"og "identity_hate"er de kategorier, der er vigtigst at identificere, da disse oftest kan ses som dybt forargende eller som noget, der kunne have sikkerhedsmæssige konsekvenser for den udsatte. Vores model skal derfor i højere grad kunne finde frem til netop disse kategoriseringer og udføre dem korrekt.

Vores maskinlæringsproblem kan altså specificeres ned til følgende; "Hvordan kan vi korrekt kategorisere ytringer, der er stærkt forargende eller kan føre til direkte skade".

8.2 Dataforbehandling genbesøgt

Når vi går ind i anden iteration af vores model, er vi blevet klogere på en ujævn distribuering af kommentarer. Derfor ønsker vi at blive klogere på forholdet mellem klassificerede

kommentarer og kommentarer, som ikke bliver klassificeret med nogen labels, altså kommentarer uden hadefulde ytringer.

Datasættet har dog ikke et label, som mærkere kommentarer uden hadefulde ytringer, hvilket derfor er vores første ændring i led af en ny forbehandlingsfase. Til at opnå dette bruger vi pandas 'iloc'-funktion, som kan håndtere data på x og y akser.

Funktionen bruges til at summere alle x værdier og angiver, at dette skal ske fra anden indeksering og fremefter. Den summerede værdi gemmes i en variabel kaldet 'col'. For at kunne identificere kommentarer, som ikke har en værdi i nogen af kategorierne oprettes en ny variabel 'row', som summere værdierne af x aksen for hver kommentar. Herefter angives en ny dimension, 'clean', i array'et, der også indeholder en boolean værdi, der tjekker om den summerede værdi er 0. Efterfølgende laves en appender, som kan sætte en variabel til 1, hvis den er sand og 0 hvis falsk. Til sidst mappes alle clean kategorierne, og hvis row er lige med 0, er den sand, og dermed mappes værdien i 'clean' til 1, hvis row er falsk, altså ikke holder værdien 0, mappes værdien i 'clean' til 0.

```
>>> col = train_csv.iloc[:, 2:].sum()
>>> row = train_csv.iloc[:,2:].sum(axis=1)
>>> train_csv['clean'] = (row == 0)
>>> appender = {True : 1, False : 0}
>>> train_csv["clean"] = train_csv["clean"].map(appender)
```

Vi forventer nu, at kommentarer uden hadefulde ytringer kategoriseres under clean og har en værdi på 1. Vi tager derfor et kig ved bruge af funktionen `train_csv.sample(5)` på et tilfældigt stykke data for at tjekke.

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	clean
6274e93...	GODDAMIT MAN LISTEN TO ME FOR...	1	0	1	0	1	0	0
9c22fa7...	"\n\n To check for socks \nGo to Wikipe...	0	0	0	0	0	0	1
e079de...	"\n\nRegarding edits made on User tal...	1	0	0	0	0	0	0
199912...	FUCK YOU, YOU ATHEIST CUNT! FU...	1	1	1	0	1	0	0
fa5ca7...	Romanian legislative elections results...	0	0	0	0	0	0	1

Tabel 5: Data eksemepl m. clean klassificering

Som forventet ser vi nu en klassificering 'clean', hvoraf kommentarer, hvor den summerede x værdi er 0 angives. Dette kan hjælpe os i retning af at skelne mellem kommentarer, men er ikke en løsning på overvægt af kommentarer, der er klassificeret som 'toxic'.

Kategorien toxic

'Toxic' kategorien opfylder som nævnt ikke kriterierne for de principper vi har arbejdet med. Når en kommentar kun ligger i kategorien 'toxic', er det altså sjældent, at der ud fra "The Harm Principle"eller "The Offfence Principle"kan argumenteres for, at den skal censureres.

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	clean
bdc082dfc9224b82	You are 22 years older than your husband! You are an old cougar! You are an old cougar! You are an old cougar!	1	0	0	0	0	0	0
891d6b977a505519	This article is simply nonsense through and through. Delete it before many people can see this crap! By the way the used English is absolutely terrible!!	1	0	0	0	0	0	0
f5dafde1e3d59e04	This is your final warning if you persist in vandalism your account will be deleted and further legal action will be taken against you.	1	0	0	0	0	0	0

Tabel 6: Data eksemepler kun klassificeret som toxic

I tabel 6 fremgår kommentarer, der kan betragtes som offensive, men som ikke falder ind under kategorien hadefulde ytringer. Det er altså ikke disse kommentarer, vi i dette projekt vil have fokus på. Det er dog stadig de kommentarer, der var en markant overvægt af i træningsættet. Vores fokus er på de kommentarer, som minimum falder ind under vores to anvendte principper, "The Harm Principle"og "The Offence Principle", og vi må derfor arbejde hen imod bedre at kunne kategorisere de kategorier, der falder ind under disse, hvilket er 'obscene', 'threat' og 'identity_hate". Vi har derfor valgt at fjerne "toxic"kategorien med følgende kode:

```
train_csv = train_csv.drop(['toxic'], axis = 1).
```

8.3 Modellering genbesøgt

Efter at have fjernet kategorien 'toxic' og tilføjet 'clean', er det nu muligt at rette fokus på de kategorier, som kan betegnes ud fra principperne vi har arbejdet med. I tabel 7 fremgår distribueringen af kommentarer i det nye datasæt.

severe_toxic	1595
obscene	8449
threat	478
insult	7877
identity_hate	1405
clean	143346

Tabel 7: Fordeling af kommentarer efter toxic er fjernet

Udover denne fordeling ønsker vi også at påvirke modellen til at slå hårdere ned på kommentarer, som kan kategoriseres som 'threat', 'obscene' og 'identity_hate'. Derfor vælger vi at justere vægtningen kategorierne 'threat' og 'identity_hate', så vi i højere grad kan sikre os, at modellen slår ud på disse minoritetsklasser. Denne vægtning tager også højde for antallet kommentarer i hver klassificering, som fremgår af figur 7. Derfor er vægten for 'obscene' ikke justeret. Til at gøre dette oprettes en Python 'dictionary', hvor hver element i listen 'class_weights' repræsenterer en klasse, og elementets værdi er den værdi, som klassen skal tilskrives:

```
>>> class_weight = {0: 1,
                    1: 1,
                    2: 1.75,
                    3: 1,
                    4: 1.75,
                    5: 1,}
>>> History = model.fit(padded_train, y_train, batch_size = batch_size,
epochs = epochs, validation_split = 0.1, class_weight = class_weight)
```

8.4 Evaluering genbesøgt

Når vi genevaluerer vores model, er vi ikke interesserede i, hvor meget selve modellen har forbedret sig, men hvordan modellen har ændret sig i forhold til vores problemstilling. Dog tager vi først et kig på modellens metrikker for at bekræfte, at modellen stadig har korrekte kategoriseringer.

Metics	**loss**	**accuracy**	**validation_loss**	**validation_accuracy**
Epoch 1/2	0.0866	0.9759	0.0647	0.9800
Epoch 2/2	0.0613	0.9806	0.0623	0.9804

Tabel 8: Evaluering af træningsæt iteration v2

Som forventet er metrikkerne i tabel 8 næsten det samme som i første iteration. Ligeledes lander evalueringen af testsættet på 0.05 i loss og 0.9805 i accuracy. Mere interessant er de kommentarer, som vi forventede ville slå ud på 'identity_hate', som vi har justeret op.

comment	**Iteration**	**toxic**	**severe_toxic**	**obscene**	**threat**	**insult**	**identity_hate**	**clean**
His career aspiration is to become a	v1	0.951	0.062	0.837	0.020	0.554	0.066	x
Christian Preacher what a faggot!!!	v2	x	0.126	0.628	0.103	0.541	0.251	0.073
Adrien Hes Gay	v1	0.919	0.102	0.619	0.0229	0.524	0.098	x
	v2	x	0.164	0.390	0.182	0.478	0.292	0.176

Tabel 9: Evaluering af kommentarer v1 & v2

I tabel 9 fremgår klassificeringer af nogle udvalgte kommentarer for model version 1 og 2. Alle modellens forudsigelser findes kan findes i bilag 1. Justeringen af modellen viser først og fremmest, at kommentarerne kan klassificeres som hadefulde ytringer. Derudover slår modellen mindre ned på klassificeringen 'obscene', mens 'threat' og 'identity_hate' får en højere værdi i version 2. En klassificering betegnes når værdien er over 0.5, derfor slår modellen stadig ikke nok ned på disse to kommentarer, da de er under 0.5 i 'threat' og 'identity hate'. I forhold til opjusteringen af vores to minoritetesklasser har vi dog en forbedring, og modellen agerer nogenlunde som vi havde forventet.

8.5 Delkonklusion

I denne iteration tog vi nogle designbeslutninger baseret på konklusionen fra første iteration. Som det første tilføjede vi klassen 'clean', som skulle kunne klassificeres. Derefter fjernede vi klassen 'toxic' baseret på en vurdering af, at denne ikke faldt ind under nogen af vores to principper fra afsnit 4. Vi opjusterede også vægte for to af vores minoritetsklasser, på baggrund af en beslutning om at prioritere disse klasser. Dette virkede i forhold til, at de to klasser havde en højere værdi. De to eksempler, som vi forsøgte at anvende, slog dog ikke nok ud til at blive klassificeret inden for klassen 'identity_hate', som vi ellers betragtede den som. Derfor kan vi konkludere, at justering af vægte ikke er nok ift. vores problemstilling. I et fremtidigt arbejde med denne model vil vi anbefale et træningssæt med flere repræsentationer i kategorien 'identity_hate'. Vi mener ikke, at en ydereligere justering af vægte vil give en retmæssig repræsentation af data, og derfor forventes det, at der vil ske overfitting på klassen. Altså skal vi formodentlig bruge et datasæt, hvor disse typer er bedre repræsenteret. Et andet alternativ kunne være at se, hvordan modellen påvirkes, hvis fylde ord fjernes fra kommentarerne. Dette tiltag vil vi komme nærmere ind på i vores diskussion i afsnit 9.2.

9 Diskussion

I dette afsnit vil vi diskutere nogle af de udfordringer og problematikker, der kan opstå, når en maskinlæringsmodel skal lære at klassificere sprog og ytringer, samt hvordan de eventuelt kan løses. Derudover vil vi diskutere, hvorvidt andre former for præprocessering af vores data kunne have udledt andre resultater i modellen, samt om andre typer af maskinlæringsmodeller ville have været brugbare til projektets formål.

9.1 Udfordringer ved sprogforandring og maskinlæring

Som det kan ses i evalueringen af vores maskinlæringsmodel, performer den med en accuracy på 0.98, hvilket altså betyder, at den kategoriserer 98 % af kommentarerne korrekt i forhold til antallet af kommentarer, der er i vores testsæt. Dette er en udmærket performance, men det skal dog påpeges, at der er nogle aspekter, som vi bevidst ikke har taget højde for, da disse ville komplicere maskinlæringsprocessen en del, og derfor er det muligt, at vores model ikke ville performe lige så godt, hvis den blev implementeret på et socialt medie. F.eks. ses det som tidligere nævnt, at Facebook benytter AI til at identificere og fjerne hadefulde ytringer, men denne finder kun 88% af det indhold, der identificeres som hadefuldt på Facebook. De resterende 12% rapporteres stadig manuelt af brugerne, og fjernes derfor reaktivt i stedet for proaktivt (Facebook, 2020b).

Dette kan begrundes med, at sproget er dynamisk og hele tiden ændrer sig (Worsøe, 2014, p. 13). Når vi skaber en maskinlæringsmodel, der skal kategorisere hadefulde ytringer, afhænger vi i høj grad af det træningsæt, vi træner modellen på, og det stadie sproget er i på det givne tidspunkt. Vores model er altså optimeret efter et statisk datasæt, hvilket står i stærk kontrast til den virkelige verden, hvor mennesker kommunikerer og ytrer sig på

dynamisk og foranderlig vis. Men hvis sproget er i konstant forandring, står vores maskinelæringsmodel over for et omfattende problem. Hvordan skal den korrekt kunne kategorisere hadefulde ytringer, hvis sproget selv er i konstant forandring? F.eks. kan ord skifte betydning fra deres originale betydning, nye forkortelser og slang kan opstå lige så hurtigt, som de kan forsvinde igen, og sprog og ord kan benyttes som kreativ udtryksform ved opfindelse af nye ord (Worsøe, 2014, p. 41-42). Denne fleksibilitet og dynamik kan gøre det svært at træne en maskinlæringsmodel i og med, at ord nærmest kan skifte betydning fra den ene dag til den anden, og nye ukendte ord opstår. Et ultimativt datasæt ville være umuligt at få fat i, da sproget hele tiden ændrer sig. Derfor skulle modellen ofte trænes med nye datasæt, der tog højde for sprogændringer, for at kunne følge med den sprogmæssige udvikling. Dette problem er dog ikke en umulig opgave, da der er tale om mindre forandringer i og med, at store sprogforandringer, der gør sprog uforståeligt fra det, vi har i dag, kan tage flere århundrede (Aitchison, 2001, p. 4). Det er derfor et langsigtet problem, der i grunden kan løses med regelmæssige træning med træningssæt, der på forhånd er klassificeret, ligesom vi løste dette problem med det nuværende sprog, i vores første iteration.

Falske positiver og falske negativer

Ligesom der er sprogforandringer, er der også forskel på at bruge et ord og referere til et ord, eksempelvis i forbindelse med at skabe opmærksomhed på problematikken omkring specielle hadefulde ytringer, eller at bruge et hadefuldt ord om sig selv i forbindelse med generobring af ordet (Facebook, 2020b). Disse faktorer er altså med til, at indhold på sociale medier nogle gange misklassificeres og på denne måde både risikerer at skabe falske positiver (indhold markeret som hadefuldt, men som ikke er det) og falske negativer (indhold markeret som ikke-hadefuldt, men som er det). Her er spørgsmålet, hvad der er at foretrække. Hvis der skabes falske negativer, kan brugere, der har ytret sig hadefuldt, gå under radaren og dermed sprede sine hadefulde holdninger, indtil det eventuelt rapporteres af andre brugere. Modsat hvis der skabes falske positiver, kan brugere, der ikke har ytret sig hadefuldt, få fjernet deres indhold, og får på denne måde ikke mulighed for at sprede deres budskab, og deres ytringsfrihed begrænses derfor på baggrund af en forkert klassifikation.

Under anden iteration af vores maskinlæringsmodel valgte vi bl.a., at vi hellere så modellen skabe falske positiver fremfor falske negativer, idet hadefulde ytringer kan have store konsekvenser både for den individuelle og for samfundet. Derfor ser vi hellere, at modellen kan fange flere muligt hadefulde ytringer i stedet for at lade eventuelle hadefulde ytringer slippe ubemærket igennem. Blev dette eksempelvis implementeret på et socialt medie, ville det formodentligt hurtigt blive upopulært, at eventuelle ikke-hadefulde kommentarer blev markeret hadefulde og fjernet. Til dette har Facebook dog en løsning, idet brugerne har muligheden for at appellere indhold, der er blevet fjernet for at overtræde deres fællesskabsregler om hadefulde ytringer. Her har brugeren altså mulighed for at få revurderet, hvorvidt der var tale om hadefulde ytringer eller ej, og hvis det viser sig, at der var tale om en falsk positiv, kan brugeren få genoprettet indholdet (Zuckerberg, 2018). Det viser sig dog, at ud af 9.600.000 indhold, som Facebook mellem januar og marts 2020 fjernede pga. hadefulde ytringer, er det kun 63.600 appellerede sager ud af 1.300.000, der genoprettes. Dette svarer

til blot 0,66% af det fjernede indhold (Facebook, 2020b). At opprioritere at få indhold markeret som hadefuldt, men også at være åben for at brugeren kan få genoprettet sit materiale kan altså være en måde, hvorpå denne problematik kan tilgås, da censurering dermed ikke vil ske uberettiget.

9.2 Alternativer i præprocessering

I vores model anvender vi nogle af keras' præprocesseringsressourcer bl.a. "tokenization" for at kunne udelade alle mellemrum i træningen af vores model. På den måde sparer vi vores model for at udføre behandling af de, som regel, semantisk meningsløse mellemrum, der optager en betydelig mængde plads i de fleste sætninger. En anden ting, vi gør, som sparer vores model for belastning, er, at vi begrænser mængden af de ord, som vi vil vektorisere, til 2000. Det betyder dog også, at de ord, som ikke befinder sig i intervallet af de 2000 mest brugte ord i datasættet, ikke vil figurere eller have indflydelse i træning af modellen.

Keras' tokenizer hjælper os altså med at udføre den slags basale præprocessering. Der findes dog midlertidigt også andre præprocesserings repositorier, som kan hjælpe med andre og muligvis mere avancerede operationer. Dette kan eksempelvis være anvendelsen af såkaldte stopord *(eng. stopwords)*. Stopord er ofte hyppigt anvendte ord, som for det meste ikke har en afgørende betydning for en sætnings semantiske betydning. På engelsk er ordene "the", "a", "an", "in" gode eksempler på stopord. Disse ord kan man under præprocessering filtrere ud af datasættet. Et populært repositorie, som kan hjælpe med lige præcis dette, er 'nltk', som indeholder mange forskellige hjælpemidler til behandling af NLP. Bl.a. indeholder nltk en samling af prædefinerede stopord i en række forskellige sprog. Her er et eksempel på, hvor simpelt en filtrering af disse ord kan være (nltk.org, n.d.).

```
>>> import nltk
>>> from nltk.corpus import stopwords
>>> filtered_words = [word for word in X_train if word not in
stopwords.words('english')]
```

Hvis vi tog beslutningen om, at disse stopord ikke er vigtige for sætningers mening, så vil det altså være relativt nemt at udelade disse. Man kan altså fokusere sin model henimod bestemte ord, som man bedømmer som værende af mindre betydning. Der findes dog et lingvistisk ræsonnement for at være tilbageholdende med denne teknik, da disse stopord ofte binder sætningerne sammen og giver dem mening. Lad os bruge ordet 'is', som er et af de mest indlysende stopord:

"X is an idiot "

Hvis stopordet *'is'* bliver processeret ud her, så kunne det følgende lige så godt have stået der, da *'isn't'* også er et stopord:

"X isn't an idiot "

Modellen ville altså ikke kunne kende forskel på disse to sætning, hvis stopord blev filtreret fra. Når man arbejder med en RNN, som analyserer på relationen mellem ord i sætninger, begynder nogle af disse stopord altså at være af afgørende betydning for sætningernes semantiske værdi Singh (n.d.).

Udover denne teknik findes der også flere typer af præprocessering, som vi kunne benytte os af, men det er vigtigt at overveje, hvordan man konfigurerer sin processering i forhold til det specifikke problem, man arbejder med, og hvilket domæne det befinder sig indenfor.

9.3 Forskellige typer af modeller

Denne rapport beskriver implementeringen af et neuralt netværk af typen LSTM, og evaluerer på denne models ydelse. LSTM er en meget specifik type model, som næsten altid bliver anvendt til analyse af sætninger i tekst, og er i stand til at forbinde ord, som i lange sætninger er semantisk relaterede, men langt fra hinanden i selve sætningen. Det er dog ikke en selvfølge, at denne model virker bedst på lige netop vores type problem, og derfor er det relevant at se på litteratur, der undersøger netop dette.

En oversigtsartikel fra Stanford University undersøger spørgsmålet og sammenligner de forskellige typer af modeller med en såkaldt f_1-score, hvor man ikke kun tager hensyn til accuracy, men også en metrik der hedder 'recall', hvor man forsøger at tage forbehold for falske positiver (Khieu & Narwal, n.d.). Oversigtsartiklen gennemgår det datasæt, som vi arbejder med i denne rapport.

I artiklen fra Khieu and Narwal (n.d.) konkluderes det, at LSTM-modeller performer en del bedre end Convolutional Neurale Netværk (CNN) og Multi Layer Perceptron-netværk (MLP). Når modellerne skal foretage analyser binært, på ordnivaeu, har LSTM-modellen en f_1-score på 0.886, hvor CNN og MLP kun præsterer henholdsvis 0.871 og 0.852. Hvis man derimod beslutter sig for at analysere på bogstav-niveau, ser det anderledes ud, her præsterer LSTM en f_1-score på 0.669 og CNN 0.800. MLP blev ikke brugt til bogstav-niveau. Det er dog først, når problemet bliver gjort til et multiklasse-problem, at LSTM virkeligt præsterer. Her opnår CNN en f_1-score mellem 0.176-0.197, hvorimod LSTM-netværket kommer helt op på 0.706 (Khieu & Narwal, n.d.).

Det lader altså til, at LSTM præsterer langt bedre end andre typer modeller, især når det kommer til en multiklassificeringsmodel. Til gengæld bliver det dog hurtigt mere relevant at overveje andre typer modeller, når modellen skal løse andre typer problemer. Eksempelvis når vi vil klassificere binært, eller hvis vi har en årsag til at udføre vores analyse på bogstav-niveau (Khieu & Narwal, n.d.). Dette understreger en pointe, som vi i denne rapport har været inde på før, at definitionen af maskinlæringsproblemet er vigtigt, da vi skal vide præcis, hvad vi vil have vores model til at gøre, og om det eventuelt kan være relevant at foretage analyser på bogstav-niveau.

10 Konklusion

Vi har gennem projektet skabt et neuralt netværk baseret på en LSTM maskinlæringsmodel, til at tackle et NLP multiklassificeringsproblem, for at kategorisere hadefulde ytringer. Vi har derudover arbejdet med principperne "The Harm Principle"og "The Offence Principle", der definerer, hvornår det kan være nødvendigt at indskærpe ytringsfrihed. Vi har brugt disse principper til at undersøge resultaterne af vores maskinlæringsmodel og til at arbejde videre på nye iterationer af modellen.

I første iteration af vores model, kom vi frem til et tilfredsstillende resultat; 98% accuracy. Dette gjorde vi ved hjælp af keras framworket og grundlæggende teori inden for maskinelæring. Dog havde vores model en stærk tendens til at klassificere kommentarer som 'toxic', grundet at denne kategori var stærkt overrepræsenteret i vores datasæt, hvilket var problematisk i og med, at kategorien ikke levede op til de to principper vi arbejdede med.

I anden iteration af modellen var målet i højere grad at komme tættere på, hvad vi anså som værende meningsfuldt at have fokus på. Vi rettede fokus mod kommentarer fra datasættet, der lå indenfor kategorien 'obscene', 'threat' eller 'identety_hate' da netop disse tre kategorier var dem, der faldt ind under "The Harm Principle"og "The Offence Principle", da de potentielt kunne medfører skade som direkte konsekvens, eller at de kunne være groft forargende. Efter en moderation af datasættet og en finjustering af de forskellige vægte i vores model måtte vi anerkende, at vi ikke ville komme meget tættere på et ønsket resultat, end hvad vi opnåede i den første iteration af modellen.

Vi kan derfor konkludere, at vi har lavet en fyldestgørende maskinelæringsmodel til klassifikation af NLP. Dog er yderligere indsnævring af, hvad den skal have fokus på, mere udfordrende end forventet. Moderering af datasæt og finjustering af vægte viste sig ikke at være effektivt nok til at give betydeligt bedre resultater i anden iteration af modellen. En indsnævring af problemet ville bedre kunne opnås med et mere fyldestgørende datasæt, der havde en større repræsentation af kommentarer, der faldt ind under de grovere kategorier af kommentarer: 'obscene', 'threat' eller 'identety_hate'.

Derudover findes der også et problem i sprogets foranderlighed, som i projektet ikke har haft nogen indvirkninger, men som på sigt kunne gøre modellen utilstrækkelig. Dog konkluderer vi, at problemet er langsigtet og at træning på regelmæssigt opdaterede datasæt ville kunne løse dette problem.

11 Perspektivering

Gennem dette projekt har vi arbejdet med et datasæt, der er indsamlet fra en ekstern hjemmeside. Trods datasættets størrelse på 223.549 kommentarer er dette blot et marginalt udsnit af det data, der findes på internettet. Sætter vi vores projekt i perspektiv, har vi altså kun beskæftiget os med et lille hjørne af internettets debatkultur. Vi kan derfor ikke være sikre på, at tonen i datasættet, vi har brugt, er repræsentativt for al kommunikation på internettet generelt, da vi formoder, at der både findes hjemmesider med blødere og hårdere debattoner. Det er derfor svært at udbrede vores konklusioner til at gælde internettet bredt, da vores datasæt kun er en lille del af den virkelighed, som er kommunikation på nettet.

I vores anden iteration af modellen forsøgte vi at opprioritere to kategorier, som er særligt grove men også underrepræsenterede. Dette viste sig at være en udfordring, og vi var ikke i stand til at producere tilfredsstillende resultater med den metode. Af denne årsag kunne en spændende fremtidig udvikling bestå i at træne en model på nye datasæt, som ikke kun var baseret på data fra Wikipedia, men fra flere forskellige sociale medier, hvor et bredere udsnit af internetkulturen kunne komme til udtryk. På platforme, som er bygget til, at brugere kan kommunikere direkte med hinanden, kunne man forvente at flere personlige henvendelse og angreb ville tage sted, hvorimod Wikipedia er et medie, som ikke er designet til person til person kommunikation. Det ville især være interessant at observere træningen af en model på data fra medier, hvor offentlig debat i højere grad udfolder sig.

Ved at diversificere kilden til data kunne det blive muligt at brede vores konklusion ud med større sikkerhed, og vi vil også være bedre sikret mod lingvistiske forskelligheder, som kan variere på tværs af internettets forskellige platforme. Et datasæt med større diversitet og antal kommentarer ville altså potentielt kunne forbedre vores resultater markant. Hvis vi tog dette videre i sin fulde udstrækning og forestillede os alle kommentarer, på alle sociale medier, indsamlet og brugt til maskinlæringsmodeller, er der dog en række etiske spørgsmål, der rejser sig vedrørende privatliv. Hvis alt, der udtales i den offentlige debat, skal danne præcedens for, hvad der er tilladt at sige i den offentlige debat, hvor mange ville så udføre selvcensur, og hvor mange vil føle sig overvågede?

Litteratur

Aitchison, J. (2001). *Language change: progress ir decay?* Cambridge University Press.

Beck, A. (2011). *Videnskab i virkelighed.* Frederiksberg : Samfundslitteratur.

Benner, T., & Fejerskov, J. (2019). Mette frederiksen erklærer krig mod mastodonter som facebook, google og amazon. *Politikken*. Retrieved 2020-03-28, from https://politiken.dk/kultur/medier/art7142944/Mette-Frederiksen-erkl\T1\aerer-krig-mod-mastodonter-som-Facebook-Google-og-Amazon

Berry, M. W., Mohamed, A., & Yap, B. W. (2019). *Supervised and unsupervised learning for data science*. Springer International Publishing.

Burkal, R., & Zuleta, L. (2017). *Hadefulde ytringer i den offentlige online debat*. Institut for Menneskerettigheder.

Chollet, F. (2017). *Deep learning with python*. Manning.

Cios, K. J. (2007). *Data mining a knowledge discovery approach* (1st ed. 2007. ed.). New York, NY: Springer US.

Clark, A. (2018). The machine learning audit—crisp-dm framework. *ISACA Journal*, *1*, 29-39.

Facebook. (2020a). *Facebook fællessakbsregler.* Retrieved 2020-05-07, from https://www.facebook.com/communitystandards/

Facebook. (2020b). Facebook transparency report. Retrieved 2020-05-24, from https://transparency.facebook.com/community-standards-enforcement#hate-speech

Gillespie, T. (2010). The politics of 'platforms'. *New Media Society*, *12*(3), 347–364. doi: 10.1177/1461444809342738

Google Inc. (2020). *Machine learning glossary.* Retrieved from https://developers.google.com/machine-learning/glossary

John McGonagle, J. K., Christopher Williams. (n.d.). *Recurrent neural network.* Retrieved 2020-29-205, from https://brilliant.org/wiki/recurrent-neural-network/

Johnson, B. (2018). Tolerating and managing extreme speech on social media. *Internet Research*, *28*(5), 1275–1291. doi: https://doi.org/10.1108/IntR-03-2017-0100

Jørgensen, N. (2017, 11). Teknologiers indre mekanismer og processer. eksemplificeret ved digital signatur. *RUC*, 0-40.

Keipi, T., Näsi, M., Oksamen, A., & Räsänen, P. (2017). *Online hate and harmful content*. Taylor Francis.

Khieu, K., & Narwal, N. (n.d.). Cs224n: Detecting and classifying toxic comments. *stanford*. Retrieved from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/reports/6837517.pdf

Kumar, R. (2020). Cerebral lstm: A better alternative for single- and multi-stacked lstm

cell-based rnns. *SN Computer Science*.

nltk.org. (n.d.). *nltk corpus documentation.* Retrieved from https://www.nltk.org/book/ch02.html

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, *abs/1811.03378*. Retrieved from http://arxiv.org/abs/1811.03378

Seif, G. (2018). Handling imbalanced datasets in deep learning. *towardsdatascience.com*, 0-8. Retrieved from https://towardsdatascience.com/handling-imbalanced-datasets-in-deep-learning-f48407a0e758

Singh, G. (n.d.). Why you should avoid removing stopwords. *towardsdatascience.com*. Retrieved from https://towardsdatascience.com/why-you-should-avoid-removing-stopwords-aa7a353d2a52

Sommerville, I. (2016). *Software engineering*. Pearson Education Limited.

The Royal Society. (2017). *Machine learning: the power and promise of computers that learn by example.*

van Aken, B., Risch, J., Krestel, R., & Löser, A. (2018). Challenges for toxic comment classification: An in-depth error analysis. *arXiv preprint arXiv:1809.07572*.

van Mill, D. (2018). Freedom of speech. In E. N. Zalta (Ed.), *The stanford encyclopedia of philosophy* (Summer 2018 ed.). Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2018/entries/freedom-speech/.

Wirth, R., & Hipp, J. (2000). Crisp-dm: Towards a standard process model for datamining. *Wirth, R., Hipp, J. (2000, April). CRISP-DM: Towards a standard process model for data mining. In Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, 29-39.

Worsøe, L. B. (2014). Nye ord på nye måder: nyorddannelse belyst fra et dynamisk sprog- og kognitionsperspektiv. *Det Humanistiske Fakultet, Københavns Universitet*.

Zuckerberg, M. (2018). A blueprint for content governance and enforcement. *Facebook*. Retrieved 2020-05-23, from https://www.facebook.com/notes/mark-zuckerberg/a-blueprint-for-content-governance-and-enforcement/10156443129621634/

12 Kode appendix

```
from keras.layers import Input, GlobalMaxPool1D, Dropout,
from keras.layers import Dense, Flatten, LSTM, Embedding,
from keras.models import Model, Sequential
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
import sys
import re
import csv
import codecs
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import keras
from keras import initializers, regularizers, constraints,
optimizers, layers
Activation
Bidirectional

# Read data
train_csv = pd.read_csv('../input/kag-data/train.csv')
test_csv = pd.read_csv('../input/kag-data/test.csv')

# Divide dataset, add 'clean'
col = train_csv.iloc[:, 2:].sum()
row = train_csv.iloc[:, 2:].sum(axis=1)
train_csv['clean'] = (row == 0)
appender = {True: 1, False: 0}
train_csv["clean"] = train_csv["clean"].map(appender)

# Remove toxic
train_csv = train_csv.drop(['toxic'], axis=1)
train_csv.sample(10)

# Define axises
X = train_csv['comment_text']
y = train_csv[['toxic', 'severe_toxic', 'obscene',
               'threat', 'insult', 'identity_hate']].values
X_testv2 = test_csv['comment_text']
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=0)

# Tokenizer
max_features = 2000
tokens = Tokenizer(num_words=max_features)
tokens.fit_on_texts(list(X_train))
tokenized_train = tokens.texts_to_sequences(X_train)
tokenized_test = tokens.texts_to_sequences(X_test)
tokenized_testv2 = tokens.texts_to_sequences(X_testv2)

# Padding
max_len = 200
padded_train = pad_sequences(tokenized_train, maxlen=max_len)
padded_test = pad_sequences(tokenized_test, maxlen=max_len)
padded_testv2 = pad_sequences(tokenized_testv2, maxlen=max_len)

# Modeleing
input_model = Input(shape=(max_len, ))
embed_size = 128
x = Embedding(max_features, embed_size)(input_model)
x = LSTM(60, return_sequences=True, name='lstm_layer')(x)
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(6, activation='sigmoid')(x)

# Balance weights
class_weight = {0: 1,
                1: 1,
                2: 1.75,
                3: 1,
                4: 1.75,
                5: 1, }

# Define model
model = Model(inputs=input_model, outputs=x)
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
# Fit model to dataset
batch_size = 32
epochs = 2
History = model.fit(padded_train, y_train, batch_size=batch_size,
                    epochs=epochs, validation_split=0.1,
                    class_weight=class_weight)

# Predict divided test set
pred = model.predict(padded_test, verbose=1)

# Evaluate model on divided test set
metrics = model.evaluate(x=padded_test, y=y_test, verbose=1)

# Predict on seperate test set
new_pred = model.predict([padded_testv2], verbose=1)
```