IDATT2502 Anvendt Maskinlæring med Prosjekt

Anvendelse av Maskinlæringsmodeller for Identifikasjon av Språk

Simon Jensen, Stian Fjæran Mogen, and Lars Brodin Østby

Norges teknisk-naturvitenskapelige universitet (NTNU)

November 25, 2021

Abstract

Denne rapporten tar for seg arbeidet gjort i faget "IDATT2502 Anvendt Maskinlæring med Prosjekt". Oppgaven har gjennom undersøkelsene foretatt i prosjektet, utviklet seg til å bli en øvelse i både anvendelse av metoder innenfor maskinlæring, samt et forsøk på å redegjøre for valgene og betraktningene gjort underveis. Rapporten viser hvordan RNN-modeller med hidden size 512 gir svært gode resultater for klassifisering av store datasett. Multinomial Naive Bayes er på sin side anvendbar på mindre datasett, men med overlegen klassifikasjon gitt liten input. Rapporten gir et helhetlig bildet av utfordringene knyttet mot klassifisering og differensiering av språk i tekstlig data. Modellene som presenteres kan skille mellom flere hundre språk. I hvilken grad er det ulike problemer og behov knyttet mot klassifiserigng av beslektede språk, og hvordan kan man på best mulig måte svare på disse utfordringene diskuteres i dette dokumentet. Målet med denne rapporten er ikke å være en fasit på hva som til en hver tid vil være den beste metoden for klassifisering av språk. Derimot skal denne rapporten være et nyttig dokument som presenterer fordeler og ulemper med de forskjellige metodene som den tar for seg. Samtidig som den presenterer resultater og tilhørende betrakninger, som ble brukt til å løse gruppens spesfikke problemstilling.

1 Introduksjon

Språkidentifikasjon er en relevant problemstilling innenfor informatikk, og maskinlæring spesifikt. I den moderne teknologiske verden er det overalt rundt oss. Din telefon retter på deg når du skriver feil i tekstmeldinger. Den foreslår hvilke ord du skal skrive, før du i det hele tatt vet det helt selv. Om du går inn på en nettside kjenner den automatisk igjen et fremmedspråk, og foreslår i stedet å oversette til et språk du kan.

I dette prosjektet skal gruppen undersøke modellering for klassifisering av språk, ved hjelp av flere forskjellige verktøy innenfor maskinlæringsfeltet. Oppgavens hensikt blir å drøfte og reflektere rundt metoder for å lage gode identifikasjonsmetoder. Ved å se på tidligere forskning og prosjekter skal gruppen lage egne modeller, og sammenligne disse opp mot hverandre. Målsetningen er at dette skal lede til en samling av nyttig data og resultater, som til slutt skal resultere i en ferdig modell som tar i bruk konklusjonene fra prosjektets undersøkelser. Dette skal forhåpentligvis kunne klassifisere med høy presisjon, fra et stort utvalg av språk og dialekter.

2 Tidligere Relevant Arbeid

Gruppens arbeid og prestasjoner er i stor grad et resultat av tidligere arbeider gjort på språkidentifikasjon, som har banet vei for videre arbeid og forskning innenfor feltet. Spesielt har Reccurent Nueral Network og dens utvidelser oppnådd utmerkede resultater på oppgaver innenfor Natural Language Processing. Språkmodellering basert på sannsynlighetsmodell Naive Bayes er også svært aktuelt arbeid verdt å anerkjenne før vi ser på gruppens resultater [16].

2.1 Naive Bayes

Naive Bayes har vist seg å være svært aktuelt i klassifisering av tekstlig data. Spesielt er "Naive Bayes, An evaluation of naive Bayesian anti-spam filtering" [5], verdt å trekke frem. I publikasjonen ble det vist at Naive Bayes i stor grad utkonkurrerte den enkle nøkkelord-baserte filtreringen som på det tidspunktet var i bruk.

2.2 Reccurent Nueral Nettverk i Natural Learning Problemer

I 1997 utga S. Hochreiter og J. Schmidhuber Long Short-Term Memory [13], og introduserte en løsning som for første gang effektivt kunne håndtere problemer med behov for langtidsminne i Recurrent Neural Network. Publikasjonen beskrev kun et potensiale for å løse anvendelse i språk identifikasjon, men har med tiden vist seg å være et essensielt verktøy i NLP-problemer.

I tiden etter utgivelsen av verket, har nytten av LSTM blitt vist i flere publikasjoner. I 2005 viste Graves og Schmidhuber at lagring av informasjon over tid gjorde LSTM til et egnet verktøy for blant annet språkidentifikasjon. [4]. Dette ble utgangspunktet til Alex Graves i hans bok "Supervised Sequence Labelling with Recurrent Nueral Networks" fra 2012 [3]. Her demonstreres blant annet hvordan Sequence Labeling i NLP kan gjennomføres ved hjelp av RNN. Også den spesifikke anvendelsen av LSTM blir beskrevet i verket. Graves viser her hvordan den håndterer vanishing gradient problemet for overlegne resultatet i hastighet, samt noe høyere presisjon.

Et nyere alternativ til LSTM ble foreslått i 2014, av K. Cho [6]. Gated Recurrent Unit (GRU) skal være en mer effektiv gated løsning i RNN enn LSTM, som oppnår like gode resultater i maskinoversettelse.

3 Metode

3.1 Datasett

For trening, test og validering i dette prosjektet brukes “WiLI-2018 wikipedia dataset” [14]. Dette er det utvidede, originale datasettet fra hva som ble foreslått i oppgavebeskrivelsen. Datasettet består av 235 språk, med 1000 paragrafer hver. Det er delt i 50-50 trening- og test set. Hver med to tilhørende filer med beskrivelser av språkene, samt link til paragrafen på wikipedia. Ettersom datasettet var stort er det hensiktsmessig å endre på ratio mellom train-test. [15]. Datasettet ble splittet på forholdet 80, 10, 10 på henholdsvis training-, validation- og testsett. Filene markert med x er setninger, mens y er de tilhørende språkene. Selv om det er stor variasjon mellom flere av språkene er det også andre beslektede språk med flere likheter. Eksempelvis inkluderer datasettet både bokmål og nynorsk, samt seks ulike former for tysk.

For å bruke datasettet, leses filene for trening validering og test til hver sin liste. Siden setningene og tilhørende språk har samme plassering i sine respektive filer, vil indeksene til listene kunne brukes til å trene og teste modellen.

3.1.1 CUDA

RNN modellene laget i Pytorch er tidkrevende å trene. Det er derfor nødvendig å ta i bruk CUDA. Dette er en parallell beregningsplattform og modell utviklet av NVIDIA. Denne gjør det mulig å dra nytte av skjermkortet sitt potensiale til å gjøre store matematiske beregninger. [8]

3.1.2 Minibatches

Ettersom datasettet var stort og vi benyttet oss av skjermkort til RNN modellen ville det ikke vært mulig å laste inn hele datasettet samtidig til minne. For å minimere minnebruken under trening deles datasettet i mindre batches som sendes inn sekvensielt til grafikkortet. På denne måten trenger en kun holde orden på feilverdier i en èn batch av gangen. Modellen blir oppdatert etter hver batch.

For å sende en batch inn i modellen må alle setninger være like lange. Dette løses ved å padde alle paragrafer mindre enn største paragraf i hver batch. Vi kutter også av paragrafer som er over en gitt grense for at enkelttilfeller ikke skal gjøre hele batchen for stor. Paragrafene sorteres før de deles i batches. På denne måten må det paddes mindre ettersom paragrafene blir mer lik i størrelse.

3.2 Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) algoritmen er en sannsynlighetsbasert læringsmetode som blir brukt i Natural Language Processing (NLP). Naive Bayes er basert på Bayes teoremet, som forteller om den betingede sannsynligheten for at et utfall inntreffer, basert på et tidligere utfall. En Naive Bayes klassifiserer er en kolleksjon av flere algoritmer. Felles for disse algoritmene er at ingen to features i klassifiseringen er relatert eller vil påvirke hverandre, derav navnet ‘naiv’. [7]

Utvinning av features og utvalg er de viktigste deloppgavene i mønsterklassifisering. En feature burde være meningsfull for problemet, skalerbar og den må inneholde nok informasjon for å kunne skille mellom ulike mønstre. For klassifikasjon av språk kan man bruke Bag of

Words med n-gram av ulik n-antall ord for feature seleksjon.

3.2.1 N-gram

N-gram handler om inndeling av tekstlig data, og betyr helt enkelt en sekvens med N-tegn/ord. [2] Ta følgende setning til betraktning; "Studenter som elsker maskinlæring". En måte å dele denne setningen er ord for ord, dette blir et unigram (1-gram). Et bigram (2-gram), deles inn på samme måte, dog med to og to ord om gangen. 1-gram og 2-gram på nevnte setning blir henholdsvis:

"Studenter", "som", "elsker", "maskinlæring" (1-gram)

"Studenter som", "som elsker", elsker maskinlæring" (2-gram)

På samme måte som ord kan deles opp i n-gram kan dette gjøres med tegn. Resultatet for et enkelt ord "Student" delt opp i trigram (3-gram) vil bli følgende.

"Stu", "tud", "ude", "den", "ent" (3-gram)

Her er det verdt å merke seg at Hvis X er antall ord i en setning K, vil antallet n-gram beskrives slik.

$$ngram_k = X - (N - 1)$$

3.2.2 Sannsynlighet med N-gram

Når vi jobber med N-gram, tar vi utgangspunkt i et stort korpus. Dette er en stor samling av tekstlig data som kan brukes i en modell. I en bigram modell er sannsynligheten for at ord a etterfølges av ord b definert på følgende måte.

antall(ord a, ord b) / antall(ord a)

Antallet her baseres seg på korpuset vi har, ved at antall ganger ord a etterfølges av ord b, deles på den totale forekomsten av ord a. Vi ser for oss at ordet "elsker" forekommer tre ganger i korpuset, hvorav en av de blir etterfulgt av "maskinlæring". Dette vil resultere i en sannsynlighet på 0.33 for at ordet "elsker" etterfølges av "maskinlæring" i N-gram modellen.

3.2.3 Bag of Words

En av de vanligste måtene å bryte ned naturlig språk til n-gram, er ved bag of words. Her ser vi helt bort i fra grammatikk og rekkefølge. Det blir derfor naturlig å se på representasjonen som en sekk med ord, hvor antall forekomster av hvert ord er det viktige. Denne representasjonen kan eksempelvis lagres på JSON format, med ordet som nøkkel, og antallet som verdi. Det er også bruke sekvenser av tegn og telle forekomster av disse.

3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) brukes ofte i Natural Language Processing. I stedet for å ta inn en stor samling av setninger, tar vi i RNN inn en og en bokstav/tegn som input. Dette tillater en større grad av fleksibilitet for lengde på setninger. For klassifisering av språk i tekstlig data, bruker vi "Mange til en" arkitektur i RNN. Dette betyr enkelt at vi tar inn mange inputs, som gir oss en enkelt output.

3.3.1 Dictionary

I RNN modeller sender vi som nevnt inn et og et element. Dette kan vi håndtere ved hjelp av et dictionary. Blant totalt 235 språk, vil det naturligvis være et stort array tilhørende tegn. Vi ønsker også å unngå duplikater av tegn, slik at vi kun lagrer et element én gang. Vi oppretter to "ordbøker" for ulike tegn og språkkoder. I dictionary klassen gjennomgåes alle paragrafer og språk fra treningsdatasettet. Alle unike tegn og språk i de to ordbøkene får en unik tallverdi som representasjon av tegnet/språket før vi benytter data i treningen. Når denne prosessen er ferdig sitter vi igjen med en fullstendig liste av unike karakterer, alle med en egen unik indeks.

For å bruke validering eller testsettet går en gjennom disse ordbøkene og undersøker om tegnene/språkene fra disse settene eksisterer fra før. Dersom de gjør det lagres tallrepresentasjonen av tegnet/språket. Dersom de ikke eksisterer i ordboken sløyfes dette tegnet/språket. Ettersom modellen er trent opp utifra encodingen i ordbøkene må disse lagres på lik linje med modellen dersom den skal eksporteres.

3.3.2 Vanishing Gradient

Et problem som oppstår med RNN er dens utfordring med å videreføre tidligere tillært informasjon over en lengre periode. Man sier gjerne at RNN har korttidshukommelse som følge av 'the vanishing gradient problem'. Dette er et problem som oppstår når gradienten over tid i løpet av en treningsøkt blir svært liten, etterhvert som man back propargater gjentatte

ganger. Ettersom gradient blir brukt til å oppdatere nettverkets vekter, så vil etterhvert lag med lav gradientverdi slutte å oppdateres og lære. Dette vil da alltid gjelde lagene tidlig i prosessen. Dette er et problem som for eksempel kan oppstå hvis man skal prosessere en lengre paragraf med tekst.

3.3.3 Hidden State

Hidden state gir oss muligheten til å håndtere input av variabel lengde, ved hjelp av en gitt vekt som beskriver hvordan vi skal bruke en gitt hidden variabel fra et tidligere state. Matematisk har vi at for alle verdier av t som ikke er null, vil vi oppdatere hidden state ved en sigmoid funksjon med input forrige hidden state og current state. Figur 1 viser hvordan dette tradisjonelt regnes ut. *ht*

Figure 1: Hidden state formell

$$\mathbf{h}_t = \begin{cases} 0, & t = 0 \\ \phi\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right), & \text{otherwise} \end{cases} \qquad \mathbf{h}_t = g\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

[1]

3.4 Utvidelser av RNN

LSTM og GRU er utvidelser av Recurrent neural network som løser Vanishing Gradient ved å implementere aktiverings funksjons lag, ofte kalt gates/porter, hvor hver port har et eget bruksområde. Disse aktiverings funksjonene er hovedsakelig sigmoid funksjoner, men også tanh i enkelte tilfeller.

Utvidelsene LSTM og GRU bruker porter, som har som jobb å regulere flyten av informasjon. De vil ikke bare mate outputet fra en celle til en annen, men bruker portene til å huske og videreføre viktig informasjon. Portene lærer seg hvilken sekvens med informasjon som er viktig å ta vare på, og hva den kan kaste til side. Informasjonen den ønsker å ta vare på vil bli lagret i en vektor kalt celle state.

Celle state er selve langtidshukommelsen, og er hva som skiller de fra en alminnelig RNN. Vektoren holder på informasjon som har blitt vurdert til å være nyttig i senere prediksjoner. Den gjør det mulig å benytte seg av tillært informasjon som ikke nødvendigvis kom direkte fra et nytt lag. I hver celle kan celle state bli oppdatert, før den blir sendt videre til neste celle. Så i motsetning til RNN som bare sender hidden state, videresendes både hidden state og celle state [9], [10].

3.4.1 Long Short Term Memory

Long Short Term Memory (LSTM) sin langtidshukommelse kommer som sagt fra implementeringen av de ulike portene. De ulike portene er som følger:

1. (f) Forget gate/remember vektor - Denne porten bestemmer hvilken tidligere informasjon vi skal kaste vekk fra celle state. Porten bruker sigmoid funksjon for å predikere, og gir et tall mellom 0 og 1 som forteller hvorvidt informasjonen skal ta vares på eller ikke.

2. (i) Input gate/ save vektor - Denne gaten bestemmer hvilken informasjon som skal legges inn i celle state, og bli en del av langtidshukommelsen. Igjen så brukes sigmoid aktivering funksjonen for å få et resultat fra 0 til 1 for å veie hvor viktig denne nye tillærte informasjonen er.

3. (g) Input modulation gate - Her lages en vektor med kandidatverdier vi kan legge til i celle state. Siste steg for inputen er en pointwise operasjon som kombinerer de to input gatene for å lage en oppdatering til celle state.

4. (o) Output gate- Basert på celle state vil outputen, altså neste hidden state, bli generert av output gate.

Strukturen til en LSTM celle bak torch.nn.LSTM() brukt i vår modell kan matematisk formuleres som:

Figure 2: Parametere i LSTM

$$\begin{aligned} i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\ f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\ g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\ o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

[12]

.

Hvor: i(t), f(t), g(t) og o(t) er enkeltvis i rekkefølge: input gate, forget gate, input modulation gate og output gate. h(t) = hidden state, c(t) = celle state, x(t) = input. σ = sigmoid og $\odot$ = Hadamard product/ pointwise operasjon. t referer til en variables tilstand i gitt tid t. t-1 vil da referere til en tidligere tilstand til de ulike variablene (Eksempel: input gate bruker den innsendte hidden state i sin utregning, altså h(t-1)).

3.5 Gated Recurrent Unit

Gated Recurrent Unit (GRU) er en utvidelse av RNN, som ønsker å forenkle noe av kompleksiteten ved LSTM, uten å ofre presisjon. I motsetning til LSTM har vi kun to porter, en "reset gate" og en "update gate". Update gate bestemmer om cell state skal bli oppdatert med den aktuelle verdien. Reset gate bestemmer om den forrige cell staten er viktig eller ikke. GRU virker på følgende måte. [9]

- Hvis reset gate har verdi nære 0, så ignoreres hidden state og irrelevant informasjon droppes.
- Hvis update gate er nære 1, så kopierer vi informasjonen til bruk gjennom mange steps.
- Update bestemmer hvor mye tidligere states betyr

Figure 3: Parametere i GRU

$$
\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}
$$

[11]

1. Både reset- og update gate regnes ut med sigmoid funksjonen på parameter- matrise og vektorer.
2. Kandidatvektoren bruker tahn funksjonen på de samme parameterne, og tar også hensyn til den nye reset-gaten.
3. Outputvektoren er en verdi som nå regnes ut ved hjelp av update gate og kandidatvektoren.

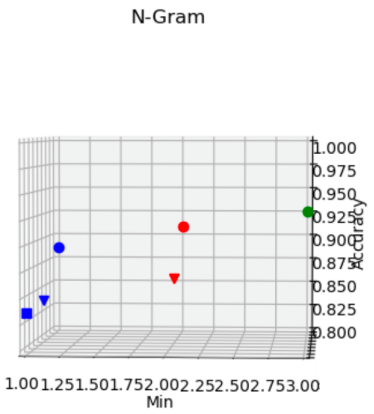
4 Resultat

Underveis i prosjektet ble ulike modeller utviklet for å prøve å oppnå en så best egnet modell til språkidentifikasjon som mulig. Forskjellige modeller egner seg til forskjellige type arbeid og input. Denne delen vil inneholde resultatene fra disse undersøkelsene, samt den generelle progresjonen til prosjektet.

4.1 Multinomial Naive Bayes

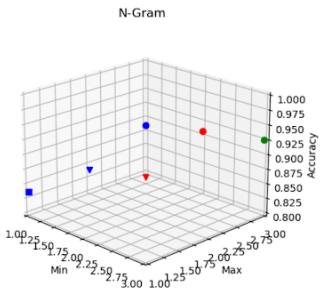
N i n-gram kan enten være karakterer eller ord. I dette prosjektet ble begge versjonene testet for å se hva som ville resultere i de beste modellen. Figur 4 og 5 viser resultatene fra kjøringene grafisk. Tabell 1 gir en tabelloversikt av resultatene for de samme kjøringene. De første tre radene av tabellen har N-gram ved bruk av karakterer. Siste rad er resultater for n-gram med hele ord.

Figure 4:



N-gram med n av 1 ,2 og 3

Figure 5:



N-gram med n av 1 ,2 og 3

Table 1: N-gram accuracy

	1	2	3
1	0.837	0.839	0.886
2		0.862	0.908
3			0.925
Ord			
	0.928		

(x,y) = (N-max/N-min)

I figur 4 og figur 5 ser man en visuell representasjon av naive bayes resultatene med 235 features. Fra grafene kan man se en relativt lineær økning basert på både minste- og største verdi av n i n-gramene. Laveste treffsikkerhet ser man naturligvis når en bruker uni-gram i modellene. Denne tar en og en bokstav,

dette begrenser antallet kombinasjoner. Man ser likevel at n-gram som med minste verdi n = 1 og største verdi n = 2, har virtuelt null differanse sammenlignet med kun uni-gram. Derimot får man en betraktelig økning ved inkludering av kombinasjoner av bokstaver opptil n = 3 i lengde. Sammenlignet med enkel 2-gram, får man også at modellen med 3-gram fortsatt gir bedre presisjon

Modellen med 2- og 3-gram, som utelater 1-gram har enda bedre accuracy enn dette igjen, og er dessuten den første modellen med treffsikkerhet over 90 prosent. Den siste modellen med kun tri-gram, blir også den beste så langt, med 92.5 prosent treffsikkerhet. Modellen som tar inn et og et ord får veldig sammenlignbare resultater, med 92.8 prosent. Kun basert på disse kan det være naturlig å anta at stigningen fortsetter ved økning av maksimal n-verdi, samtidig som man begrenser laveste verdi av n. Det er derimot nødvendige betraktninger som må presenteres for å kunne ta en kvalifisert avgjørelse for anvendelse av denne typen modeller.

4.1.1 N-gram vektorisering

Først og fremst tas det utgangspunkt i teorien presentert tidligere i rapporten. Tri-gram er et svært godt utgangspunkt for å identifisere og klassifisere språk, men har den naturlige svakheten ved at ord med mindre enn tre bokstaver ikke kan sees på isolert. En kan argumentere at fullstendige setninger tilnærmet alltid vil ha flere ord med tre eller flere bokstaver. Likevel er det verdt å ta dette til betraktning dersom man ønsker å velge den beste modellen for en spesifikk type anvendelse.

Et annet punkt som er verdt å notere seg er måten modellen i Naive Bayes med N-grams lagrer informasjon. Siden metoden lagrer alle kombinasjoner av ord basert på minste og høyeste verdi av n, får man store økninger av den fysiske størrelsen på modellen. Tabell 2 er en oversikt over størrelsen på de ulike modellene (i gigabyte).

Table 2: Modellstørrelse i GB

	1	2	3
1	0.04	1.48	6.35
2		1.44	6.31
3			4.85
Ord			
	6.91		

(x,y) = (N-max/N-min)

Fra tabell 2 ser man hvor store økninger man får for hver økning av maksimal n. En økning i fysisk lagringsplass er her relativt intuitiv. For hver økning får man også et stort antall nye mulige kombinasjoner av bokstaver som må lagres i modellen. Som følge av at presisjonen i stor grad økte parallelt med økningen av maksimal n, ville en økning fra maks n fra 3 til 4 være det naturlige neste steg. Denne markante økningen av modellstørrelse fører blant annet til at å sette maksimal n til 4, ikke lot seg gjøre på grunn av begrensninger i tilgjengelige hardware til kjøring av modellen. Det er likevel verdt å notere seg at dette eventuelt kan bidra til en forbedring av modellens accuracy. Men på grunn av begrensninger ses dette derimot på som et mindre gjennomførbart alternativ enn andre mer aktuelle alternativer.

4.1.2 Begrense antall features

Per nå så har datasettet 235 forskjellige språk som klassifiseres, datasettet inneholder tusen ulike setninger per språk, til totalt 235 000 setninger. Av disse blir 80 prosent, altså 188 000 setninger brukt til trening. Ved å sette ned antall språk til for eksempel de 20 mest brukte språkene i verden, vil vi ha svært mye mindre data å lagre. Dermed vil det være mulig å kutte ned modellstørrelsen betraktelig. Spørsmålet i dette tilfellet blir hvorvidt dette er hensiktsmessig overfor problemstillingen. Som nevnt i introduksjonen er ønsket å kunne skille på selv nært beslektede språk og dialekter. For eksempel vil man med en slik tilnærming ikke kunne ha noen mulighet til å skille på norsk bokmål og nynorsk. Fra denne rasjonalen, ble det bestemt at å fjerne store mengder språk ikke er et alternativ som på en tilfredsstillende måte lar den spesifikke problemstillingen bli løst. Det kan likevel være verdt å se nærmere på hvordan dette vil påvirke både størrelsen og treffsikkerheten på modellene.

4.1.3 Justert datasett

For å skalere ned datasettet tas det et utvalg av 23 vilkårlige språk, som tilsvarere omtrent ti prosent av det originale sine totalt 235

forskjellige språk. Det resulterer i totalt 23 * 1000 * 0.80 = 18 400 setninger til trening. Dette tillater å kjøre høyere verdier av n, da minnebruken blir lavere. Tabell 3 med resultatene fra kjøringen:

Table 3: N-gram justert datasett

	1	2	3
1	0.972	0.976	0.981
2		0.978	0.982
3			0.983
Ord			
	0.976		

(x,y) = (N-max/N-min)

Først og fremst viser tabellen et stort hopp i accuracy fra forrige kjøring. Den beste modell fra denne kjøringen har 98.3 prosent treffsikkerhet, sammenlignet med 92.5 prosent fra den tilsvarende 3-gram modellen trent og testet på flere språk. Dette betyr naturligvis ikke at det er en bedre modell, men det forteller litt om hvordan man eventuelt kan anvende naive bayes på en mer effektiv måte. Som nevnt brukes et utvalg av vilkårlige språk som er kun 10 prosent av det originale datasettet. Som konsekvens vil modellen ha færre språk å gjette mellom, samtidig som man begrenser antall relaterte dialekter som kan være enkle å misforstå. For eksempel vil det være færre tilfeller av forvirring mellom relativt like språk som norsk og nynorsk. Om man eksempelvis kun ønsker å bruke modellen til å skille mellom mellom verdens 20 mest brukte språk uten behov for å skille mellom dialekter, kan dette være et godt alternativ. Problemet med store plasskrevende modeller har også blitt adressert gjennom denne endringen.

Den største modellen fra kjøringen med alle språkene var 3-1 modellen, med hele 6.35GB. Etter endringen har den tilsvarende modellen på færre språk, fått en størrelse på 54.2MB, eller 0.054GB. Dette er under ett prosent av den originale størrelsen. Dette gir mening, da den enorme størrelsen skyldes lagring av en eksponentielt økende matrise med kombinasjoner. En økning av antall språk med en faktor av 10, vil øke størrelsen på matrisen tilsvarende 10^2. Det er derfor verdt å merke seg at modellstørrelsen er mye mer håndterlig på mindre datasett.

Avslutningsvis er det verdt å merke seg at det nå er en mulighet å utforske for høyere verdier av n, da minnebruken naturligvis blir lavere. Ved kjøring med n av 4, 5, 6 og 7 blir det følgende resultater:

Table 4: Modeller med n av 4

	1-4	2-4	3-4	4-4
Accuracy	0.985	0.985	0.985	0.987
Størrelse i GB	0.235	0.235	0.229	0.182

I tabell 4 kan man se at en økning av n fortsetter å gi bedre resultater på presisjon. Samtidig fortsetter det det å være hensiktsmessig å kun inkludere en enkel type n-gram i modellen, slik at minste og høyeste verdi er den samme.

Table 5: Modeller med n av 5, 6 og 7

	4-4	5-5	6-6	7-7
Accuracy	0.987	0.989	0.989	0.990
Størrelse i GB	0.182	0.446	0.791	1.11

Tabell 5 viser at presisjonen til modellene divergerer mot 99 prosent. For dette spesifikke datasettet gir det en marginal økning med høyere verdi av n enn fire. Samtidig ser man at modellene raskt øker i størrelse, slik at de etter hvert blir sammenlignbare med modellene som trenes på det store datasettet.

4.2 Recurrent Neural Networks

4.2.1 Loss og Læringsrate

Innledningsvis ble det brukt en konstant læringsrate på 0.001, noe som er standardverdien for Adam optimalisering. Modellene ble kjørt med GRU og LSTM, og det det forsøkt med ulike hidden size. Her ble de beste resultatene på valideringssettet målt til 95.2 prosent presisjon for LSTM og 94.6 prosent for GRU. Gruppen oppdaget også at resultatene utover i kjøringene fra de respektive modellene, økte i nøyaktighet samtidig som de fikk økt tap.

Epoke	Accuracy	Loss
6	0.948	0.2081
7	0.949	0.2160
8	0.950	0.2188
9	0.949	0.2340

LSTM, Hidden Size 512

Accuracy og loss epoke- 6 til 11

Til tross for relativt god presisjon er det ønskelig å forbedre ytterligere, spesielt med

tanke på den uventede økningen av tap. Det implementeres derfor en læringsrateplanlegger som får læringsrate til å avta utover i treningsrundene. På grunn av tidsperspektiv med kjøringer som tok flere timer, valgte gruppen å ta i bruk en enkel planlegger som sank læringsraten med 1/10 hver femte runde. Det ble forsøkt med ulike startverdier, hvorav startverdi 0.001 var det som ga raskest og best resultat. Flere ulike planleggere kunne vært undersøkt ved videre arbeid, men blir ikke vektlagt ytterligere i denne rapporten.

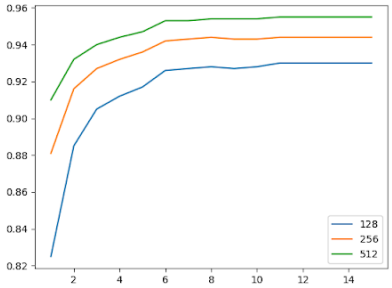
4.2.2 Valg av Hidden Size

Å finne ut hvilken hidden size som resulterer i best resultat for en gitt modell er ingen eksakt vitenskap. En generell tommelfingerregel (riktignok ikke alltid utrolig hjelpsom) er at den optimale størrelsen på hidden layer vanligvis er mellom størrelsen på input og output. Det ble undersøkt med en rekke hidden sizes for å finne de som ga best resultat. Noe annet som er verdt å ta inn i betrakting er at ved multiplisering på moderne skjermkort lønner det seg å ha en hidden size og kan deles på 16. Vi benyttet oss av hidden size i toer-potens, mer spesifikt de tre størrelsene 2^7, 2^8 og 2^9. Man trenger ikke nødvendigvis å belage seg på toer-potensener, men disse tallene vil alltid deles på 16. Her er det verdt å merke seg at $2^7 = 128 <$ output size (235). Dette var noe vi prøvde ut for å undersøke om det var mulig å benytte seg av mindre størrelser enn det tommelfingerregelen sa. I tillegg gir større hidden størrelser flere parametere i treningen og økt vram minnebruk under kjøring.

4.2.3 Presisjon og tap med variabel Hidden Size

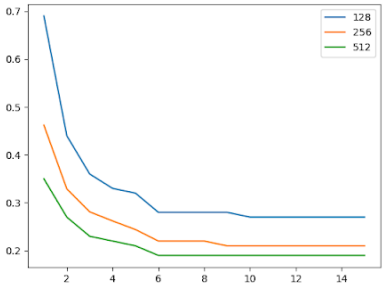
Figur 6 og 7 presenteres resultater fra unidirectional LSTM, begge med tre grafer hver. Det første plottet viser en oversikt over nøyaktigheten til modellen over x-antall episoder. I plottene er det tegnet grafer for hver enkelt verdi av hidden size. Verdiene er henholdsvis 128, 256 og 512. Et raskt blikk på grafene i figur 6, viser en klar klar sammenheng mellom mellom accuracy og hidden size. Denne trenden går igjen i alle utvidelsene av RNN modellene, dette kommer vi tilbake til. Figur 7 gir en grafisk representasjon av tap i modellen igjen for ulik hidden size. Også i dette tilfelle ser vi en sammenheng mellom endring av hidde size og vår måling av loss. Høyere hidden size gir lavere loss, som er gunstig for en bedre- og mer nøyaktig modell uten overfitting. Basert på denne dataen er det åpenbart at den høyeste

Figure 6: Accuracy Hidden Size



Accuracy LSTM, Hidden Size 128, 256, 512

Figure 7: Loss Hidden Size



Loss LSTM, Hidden Size 128, 256, 512

verdien for hidden-size, 512, er mest fordelaktig for presisjon og loss i dette tilfellet.

Det er også interessant å se på punktet den henholdsvise stigningen og nedgangen avtar. Denne endringer begynner i epoke tre, og man kan se grafene begynner å divergere fra epoke seks. Det er i disse punktene læringsraten avtar (etter implementasjonen av planlegger), slik at modellen i mindre grad tar til seg endringer. På denne måten holder loss seg stabil også etter epoke seks, i motsetning til resultatene som ble presentert ved konstant læringsrate.

4.2.4 Bidirectional vs Unidirectional RNN

I tillegg til variabel læringsrate og hidden-size, implementeres en toveis/bidirectional RNN model. For kjøring av begge de nevrale nettverkene ble det testet med både uni- og bidirectional, samt med ulike hidden layer størrelser slik som ved tidligere kjøringer. Uni- og bidirectional handler om minnet til modellen under læring. Som navnet unidirectional tilsier, så ser unidirectional bare en retning. Den ser kun på fortiden og hvor den kom fra. Bidirectional derimot lar oss kjøre input i to retninger, både fra "fortiden" til "fremtiden", og "fremtiden" til "for-

Figure 8: Accuracy LSTM vs BLSTM

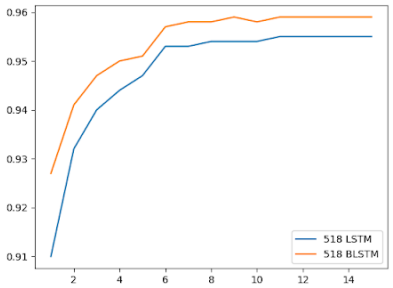
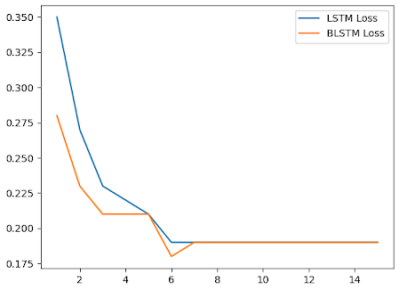


Figure 9: Loss LSTM vs BLSTM



tiden". Dette gjøres ved å kjøre LSTM baklengs i tillegg til den normale fremadkjøringen, slik at modellen lærer informasjon fra "fremtiden". Når disse to hidden statene senere kombineres kan man for hvilket som helst punkt i tid peke ut både hva som kommer før og etter det gitte punktet.
Tabell 6 viser beste presisjon på valideringssettet under trening for hver hidden size. Som vi ser i tabellen gir en hidden size på 512 best resultat for både uni- og bidirectional i både LSTM og GRU. Man kan da tenkte at en enda høyere hidden size ville gitt et enda bedre resultat igjen, men med hardware gruppen hadde tilgjengelig så ble vi begrenset i utforskingen av større hidden sizes. Vi ser også at modellene med toveis minne ga høyere nøyaktighet enn uten. Det tok i gjengjeld åpenbart betraktelig lenger tid å trene opp disse modellene, ettersom man gjør dobbelt opp med arbeid.

Table 6: Unidirectional vs Bidirectional

Modell	Uni-GRU	Bi-GRU	Uni-LSTM	Bi-LSTM
128 acc	0.923	0.939	0.930	0.943
128 loss	0.2967	0.2385	0.2711	0.2211
256 acc	0.942	0.957	0.944	0.955
256 loss	0.2267	0.2053	0.2115	0.1865
512 acc	0.953	0.957	0.955	0.959
512 loss	0.2025	0.2092	0.1855	0.1869

acc = Accuracy

For å tydeliggjøre forskjellen i resultat mellom uni- og bidirectional RNN, kan vi se på differansen mellom LSTM med hidden-size 512.

Fra figur 8 ser man tydelig at BLSTM modellen gir høyere accuracy gjennom hele kjøringen. Det er derimot en relativt marginal økning sammenlignet med for eksempel økningen ved endring av hidden size. De to beste resultatene i LSTM og BLSTM er henholdsvis 95.5- og 95.9 prosent accuracy. Sammenligner vi loss mellom de to modellen, ser vi at det noe overraskende er LSTM som har lavest loss ved høyest accuracy. Denne differansen er også minimal, i den grad det ikke er synlig på figur 9 når vi runder av til to desimaler. Noe interessant er det at BLSTM starter med både høyere accuracy, og lavere loss. Dette jevner seg derimot ut i det grafene divergere ved epoke 6.

4.2.5 LSTM vs GRU overordnet

Resultatene fra testene har gitt et godt grunnlag for å sette LSTM og GRU opp mot hverandre. De beste resultatene fra undersøkelsen ble oppnådd med hidden size satt til 512. Samtidig ble det tydelig at en variabel læringsrate som ble mindre utover kjøring minimerte økning av loss. Hvorvidt ordinær- eller bidirectional RNN er mest hensiktsmessig kan diskuteres. Bidirectional gir noe høyere presisjon, men marginal økning av loss der hidden size er 512. Modellen blir også større med en faktor av 2. For å sammenligne LSTM og GRU, tar vi derfor både uni- og bidirectional til betraktning, sammen med de forbedringene som ga best resultater.

Table 7: Unidirectional vs Bidirectional 512

Modell	Uni-GRU	Bi-GRU	Uni-LSTM	Bi-LSTM
Accuracy	0.953	0.957	0.955	0.959
Loss	0.2025	0.2092	0.1855	0.1869

Hidden Size = 512

Resultat fra valideringssett

Table 8: Resultater fra uavhengig testsett

Modell	Uni-GRU	Bi-GRU	Uni-LSTM	Bi-LSTM
Accuracy	0.953	0.954	0.954	0.958
Loss	0.2076	0.2142	0.1918	0.1878

Hidden Size = 512

Resultat fra uavhengig testsett

Fra resultatene i tabell 7 kan vi at det kun skiller 0.2 prosent mellom henholdsvis uni- og

bidirectional GRU og LSTM på valideringssettet. For å bekrefte resultatene, ser vi tilsvarende verdier fra prediksjonene på det uavhengige testsettet. Her går de samme trendene igjen for loss og presisisjon (se figur 8).

4.2.6 LSTM vs GRU Kjøretid

En begrensende faktor for gjennomføring av prosjektet, har vært hvor tidkrevende det er å trene RNN-modellene. For å sørge for at modellene har nådd sin tilnærmede maksimale presisjon, har samtlige RNN-modeller blitt kjørt 15 epoker.

Table 9: Epoke 10-15

Epoke	Train acc	Valid acc	Train loss	Valid loss
10	0.995	0.958	0.0220	0.1890
11	0.996	0.959	0.0182	0.1875
12	0.996	0.959	0.0173	0.1874
13	0.996	0.959	0.0168	0.1874
14	0.997	0.959	0.0163	0.1875
15	0.997	0.959	0.0159	0.1877

LSTM, Hidden Size 512

Accuracy og Loss epoke- 10 til 15

Resultatene i tabell 9 viser begrensede endringer i presisjon fra og med epoke 10. Basert på dette kan man diskutere i hvilken grad det er hensiktsmessig å fortsette kjøringen helt til epoke 15. Det er likevel en ikke triviell differanse i loss for både trenings- og valideringssettet. En reduksjon fra 0.0220 til 0.0159 og 0.1890 til 0.1877 henholdsvis.

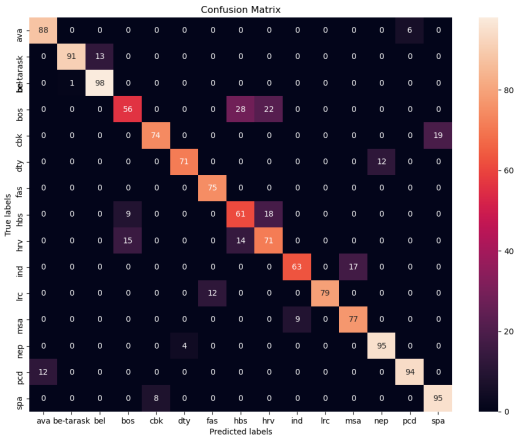
Table 10: Kjøretid LSTM

Modell	256 Uni	256 Bi	512 Uni	512 Bi
LSTM	2263	3751	4466	7851
GRU	2228	3491	3678	6359

Kjøretid i sekunder

Når det kommer til kjøretid viser resultatene fra tabell 10 først og fremst at å doble hidden size, betraktelig øker kjøretid. Det kan være verdt å merke seg at slike type resultater vil variere avhengig av hardware og hvordan kjøringen gjennomføres. En tilnærmet 60 prosent økning for LSTM, og 100 prosent økning for GRU er likevel svært stor. Overgangen fra unidirectional til bidirectional gir også en forventet økning, som er tilnærmet lik en dobling i kjøretid. GRU opp mot LSTM er også interessant. Som forventet tar det kortere tid å kjøre GRU enn LSTM, men forskjellen er ikke like markant som for de tidligere nevnte endringene.

Figure 10: Confusion matrix fra språk med høyest avvik



4.2.7 Confusion Matrix

Så langt har rapporten i stor grad kun fokusert på presisjon og loss for å avgjøre styrkene og svakhetene til maskinlæringsmodellene, og i noe mindre grad fysisk størrelse og kjøretid. Det kan derimot være verdt å gå litt mer i dybden, for å se hvorfor vi "kun" får rundt 95 prosent presisjon når modellene trener på samtlige språk. For dette kan det brukes en confusion matrix. En confusion matrix, eller forvirringsmatrise, er en tabell som viser hva modellen predikerte opp mot hva som var det faktiske svaret. Representasjonen viser oss i hvilke, og hvor mange tilfeller modellen vår har valgt feil.

Ettersom datasettet inneholder 235 forskjellige språk, er det mer hensiktsmessig å vise en representasjon med de mest interessante observasjonene. Av den grunn dannet vi en confusion matrix fra språk med minst ett avvik over 10, se figur 10. Med avvik menes at modellen predikerer språk X når det egentlig er språk Y.

Forvirringsmatrisen i figur 10 ble skrevet basert på resultater fra testsettet. Denne ble tilfeldig splittet. Av den grunn er det nødvendigvis ikke like mange instanser av hvert språk. Av sannsynlighetsmessige årsaker vil det uansett være omtrent 100 avsnitt totalt fra hvert språk.

Table 11: Tilfeller av store avvik

Predikert språk	Faktisk språk 2	Feil
Avarisk	Pennsylvaniatysk	12
Hviterussisk	Taraškievica	13
Bosnisk	Kroatisk	15
Høysorbisk	Kroatisk	14
Høysorbisk	Bosnisk	28
Kroatisk	Høysorbisk	18
Kroatisk	Bosnisk	22
Lurisk	Persisk	12
Malay	Indisk	17
Nepali	Doteli	12
Spansk	Chavacano	19

Feil fra Confusion Matrix, LSTM 512, testdatasett

Fra tabell 11 leser vi blant annet at modellen ofte tar feil på de slaviske språkene; bosnisk, høysorbisk og kroatisk. Dette passer med vår intuisjon. Disse språkene er sterkt relatert, og vil ha mange likheter som gjør at de er vanskeligere å skille.

4.2.8 Justert datasett

Basert på observasjoner fra confusion matrix og modellenes kjøretid ved mange epoker, er det ønskelig å se på prestasjonen til modellene med et justert datasett. Samme metode som for Multinomial Naive Bayes blir brukt til å redusere datasettet. Ved å ta et utvalg av 23 tilfeldige språk, får vi resultater fra et mindre antall features. For sammenlignings skyld blir disse satt opp mot resultater fra vår mest presise modell, BLSTM med hidden size 512 (tabell **??**. De målte resultatene stemmer overens med hypotesen. Det oppnås naturligvis høyere accuracy, og kortere kjøretid.

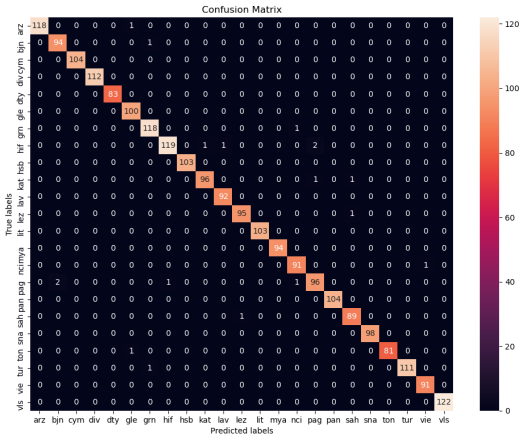
Table 12: Sammenligning Justert Datasett

Modell-features	Accuracy	Loss	Kjøretid
BLSTM 235	0.959	0.1869	7851
BGRU 235	0.957	0.2092	6359
BLSTM 23	0.989	0.051	893
BGRU 23	0.989	0.053	744

Kjøretid i sekunder, resultater fra valideringssett
Hidden Size = 512

4.2.9 Confusion Matrix Justert Datasett

Figure 11: Confusion matrix med justert data

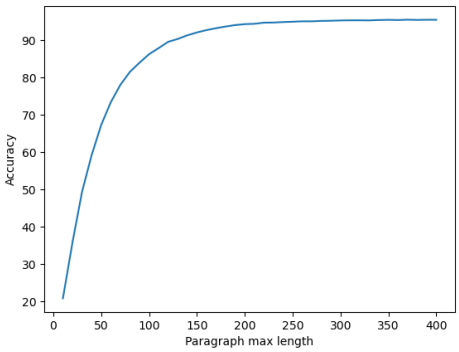


Som konsekvens av et utvalg på under 10 prosent av det originale datasettet, er denne forvirringsmatrisen (figur 10) lesebar selv med alle 23 språkene. For å bekrefte eller avkrefte gruppens antagelse om at dette vil gi en lavere forekomst av store avvik mellom språk, kan matrisen brukes for å se hvor modellen slår feil. I motsetning til det store datasettet, er det svært få tilfeller av et stort antall feiltagelser. Et raskt overblikk forteller at modellen aldri predikerer feil mer enn to ganger for to språk. Dette er en klar forbedring fra modellen med det fullstendige datasettet.

4.2.10 Justert testsett

Det vil være hensiktsmessig å se i hvilken grad modellene kan predikere setninger når lengden på inputdataen begrenses. Så langt har resultatene som er blitt presentert vært for paragrafer av ulik lengde. Det har dog så langt ikke blitt presentert forsøk som eksplisitt viser modellens evne til å klassifisere setninger av begrenset og variabel lengde.

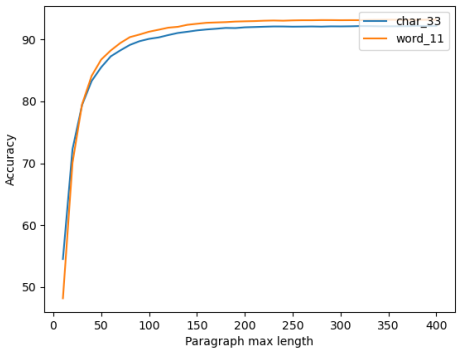
Figure 12: Presisjon på variabel setningsslengde



BLSTM 512 Presisjon testdata, variabel inputlengde

Presisjonen på testsettet ble testet med variabel lengde av inputen (se figur 12). Lengden langs x-aksen er antall karakterer som modellen predikerer. Den første målingen er med maks ti karakterer. Dette betyr at alle paragrafer med mer enn ti karakterer blir kuttet av. Her får vi naturligvis svært lav presisjon, rett i overkant av 20 prosent. Det blir derimot en svært rask eksponensiell økning, som stabiliserer seg på 95 prosent allerede på en maks lengde 200 karakterer. Avhengig av setning og språk, tilsvarer dette omlag 30-50 ord for å oppnå den tilnærmede maksimale presisjonen på 95 prosent for samtlige 235 språk.

Figure 13: Presisjon på variabel setningsslengde



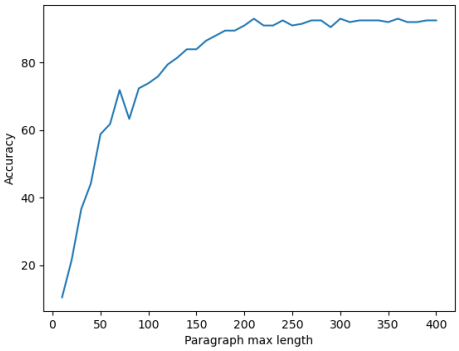
N-gram Char 3-3 og Ord
Presisjon testdata, variabel inputlengde

Om samme test kjøres for MNB-modellene, kan man trekke ut noen interessante observasjoner. Først og fremst ser man fra figur 13 at modellen som tar inn bokstaver, får høyere presisjon enn modellen med ord for prediksjon av korte setninger. De to divergerer til slutt på sammenlignbar presisjon. Sammenlignet med RNN-modellen som divergerte først etter paragraflengde på 200 karakterer, når n-gram modellene 90 prosent accuracy allerede før lengde 100. For korte setninger vil altså n-gram modellen være mer presis.

4.2.11 Resultater fra beslektede språk

Det er allerede vist at en utfordring for modellen er når den er nødt til å skille på beslektede språk. Avslutningsvis kan vi med justert lengde på testsettet, vise hvordan presisjonen til modellen utvikler seg basert på inputlengden. Et slikt tilfelle vil være for prediksjon mellom norsk og nynorsk.

Figure 14: Presisjon Norsk vs Nynorsk



Norsk vs Nynorsk
Presisjon testdata, variabel inputlengde

Grafen i figur 14 deler mange likheter med tilsvarende graf for testing på det totale datasettet (figur 12). Den blir som antatt ikke like nøyaktig. Likevel ser vi en releltivt høy presisjon etter 150 karakterer, før modellen til slutt jevner seg ut på 92.4 prosent.

5 Diskusjon

I dette prosjektet har en rekke modeller blitt utviklet for å teste hvilken type maskinlæringsmodell som gir oss best resultater i språkidentifikasjon. Rapporten har testet et utvalg forskjellige modeller, med ulik grad av tilknytning. MNB og RNN-modellene bruker helt forskjellige metoder for å løse samme problemstilling. GRU og LSTM bruker sammenlignbar fremgansmåte med noen viktige forskjeller. Prosjektet har også presentert forskjellige varianter av samme modelltype, ved å endre ulike parameter som gjør utslag

og gir oss et annet resultat. I denne seksjonen vil de presenterte resultatene bli diskutert, og fordelene og ulempene ved de forskjellige modellvariantene blir belyst.

MNB ga noen gode resultater, men viste seg å være svært plasskrevende for lagring av mange kombinasjoner på store datasett. De beste resultatene fra n-grammene på karakterer, ble helt åpenbart for modellene som inkluderte til og med kombinasjoner på tre og tre karakterer. Det var derimot mindre åpenbart hvorvidt det var hensiktsmessig å inkludere både 3-gram og 2-gram i modellen, eller om en ren tri-gram modell var veien å gå. Begge n-gram variantene ga merkbart høyere accuracy enn de andre kombinasjonene som ble testet (de to eneste over 90 prosent), men også forskjellen i presisjon mellom disse to spesifikke modellen var merkbar. 90.8- og 92.5 prosent for henholdsvis 2-3-gram og 3-gram er på ingen måte trivielt. Derimot er det verdt å merke seg hvordan n-gram modeller fungerer før vi kan gi en konklusjon på hvilken som er best egnet.

Ved å skrive ut vektoriseringen av to setninger: “hei på deg” og "hei du", kan man hente ut nyttig informasjon ved å konvertere de til en matrise med forekomster av ulike ngram. For å introdusere konseptet, ser vi først resultatet fra ord-unigram:
"hei": 0, "på": 1, "deg": 2, "du": 3. Setningene over kan beskrives gjennom forekomster av de ulike ngrammene. [1, 1, 1, 0] [1, 0, 0, 1]
En kan gjøre noe tilsvarende med bokstavngram. Bi- og trigram blir:
’hei’: 0, ’ei ’: 1, ’i p’: 2, ’ på’: 3, ’på ’: 4, ’å d’: 5, ’ de’: 6, ’deg’: 7, ’eg ’: 8, ’g ’: 9, ’i d’: 10, ’ du’: 11, ’du ’: 12, ’u ’: 13, ’he’: 14, ’ei’: 15, ’i ’: 16, ’ p’: 17, ’på’: 18, ’å ’: 19, ’ d’: 20, ’de’: 21, ’eg’: 22, ’g ’: 23, ’du’: 24, ’u ’: 25
3-gram:
’hei’: 0, ’ei ’: 1, ’i p’: 2, ’ på’: 3, ’på ’: 4, ’å d’: 5, ’ de’: 6, ’deg’: 7, ’eg ’: 8, ’g ’: 9, ’i d’: 10, ’ du’: 11, ’du ’: 12, ’u ’: 13

Først og fremst viser dette at 3-2 modellen lagrer flere kombinasjoner. Dette er ikke noe nytt, da det tidligere i rapporten ble presentert at 3-2 gram modellen tok betydelig større fysisk lagringsplass enn 3-gram modellen. Intuitivt er det fort gjort å tenke at dette nødvendigvis vil føre til høyere accuracy, noe resultatene fra forsøkene har motbevist. Dette blir forsterket ved at 3-1 gram modellen får enda lavere presisjon enn begge disse modellen. Det viser seg her at å lagre kombinasjoner av to og to bokstaver ikke gir positivt utslag når vi lagrer kombinasjoner på tre og tre. De beste resultatene ga i underkant av 93 prosent presisjon på de store datasettene, som avhengig av bruksområdene kan beskrives som under tilfredsstillende. Til tross for disse manglene, var MNB svært rask i treningen. Metoden er naiv, og derav vil ingen to features påvirke hverandre. Samtidig viste MNB seg effektiv på det reduserte datasettet, hvor de begrensende faktorene som lagringsplass og lavere accuracy ble eliminert. Den mest presise modellen ga her en accuracy på over 98 prosent, samtidig som den totale lagringsplassen ble redusert med en faktor på 100.

Gruppen ønsket å utforske om den effekten med økt presisjon ved økning av kombinasjoner opp til både fire og fem karakterer. Manglende minnekapasitet på hardware ble dessverre en begrensende faktor for videre utforskning av hypotesen på det originale datasettet. Løsningen gruppen fant for denne problemstillingen ble å redusere datasettet, slik at modellen trente og testet på kun 23 vilkårlige utvalgte språk av de opprinnelige 235. Ved å redusere antall features, ble det mulig å undersøke hvilken effekt det vil gi å øke antall n-grams utover n=3. Her ble det vist at n=4 ga enda et merkbart hopp, fra 98.3 prosent til 98.7 prosent. Med det nye datasettet var det derimot ingen tydelig forbedring etter dette punktet. For å øke presisjonen med over 0.1 prosent, måtte n økes fra 4 til 7. Basert på dette er det rimelig å anta at en økning av n vil gi bedre resultater for det store datasettet, men det er usikkert i hvor stor grad den vil forbedre seg etter n=4. Her er det spesielt viktig å merke seg at økningen i fysisk størrelse for den trente modellen vil øke proporsjonalt, som også var grunnen til at det ikke var mulig å teste dette på det originale datasettet med nåværende hardware.

RNN modellen ga gode resultater for presisjon fra begynnelsen. Gruppen merket seg derimot en uforventet trend for tapet under kjøringen. Resultatene viste at valideringstapet til å begynne med sank samtidig som presisjonen hadde økt. Lengre ut i treningen begynte derimot tap å øke igjen, samtidig som nøyaktigheten fortsatte å øke. Dette kommer av at to fenomener skjer samtidig. Modellen begynner med “overfitting”, at den tilpasser modellen i for høy grad til treningssettet, samtidig som den fortsatt drar læring datasettet. Crossentropyloss, som er tapsmetoden vi brukte i modellen, fungerer på en slik måte at èn veldig gal prediksjon kan få tapresultatet til å eksplodere. Tapsmetoden vi brukte kan

på mange måter sees på som et slags mål på overraskelse i prediksjonene. Resultatene kan forklares på denne måten:

La oss si at modellen predikerer følgende:
[0.1, 0.9, 0.9001, 0.8]
Dersom vi ser på presisjon velger vi maks verdi:
[0,0,1,0]
Forventet resultat:
[0,1,0,0]

Tap og presisjon kan defineres på følgende måte:
Tap - Samlet feil mellom y og ypred
Presisjon - Hvorvidt y og max(ypred) er lik

Her vil metoden ha lav loss men lav nøyaktighet. Dersom nøyaktigheten økte samtidig som tap kan dette gi en indikasjon på at en gal prediksjon blir enda "verre" samtidig som modellen blir bedre fordi den lærer og gir flere riktige prediksjoner.

GRU og LSTM deler mange likheter. Den tradisjonelle recurrent unit erstatter activation. Den nye verdien vil også være et resultat av beregning med current input og en tidligere hidden state. Både GRU og LSTM benytter reset state og update gate. Derimot har LSTM den ekstra kompleksiteten ved at den har ytterligere gates som GRU ikke har. Den ekstra kompleksiteten til LSTM vil også gjøre den vanskeligere å implementere. GRU på sin side har derimot ingen internminne slik som LSTM. GRU tar færre treningsparametere og vil til gjengjeld bruke mindre minne og kjøre raskere enn LSTM. GRU kan predikerer bedre enn LSTM hvis datasettet er begrenset. Fra prosejekts resultater for begrenset datasett, ser vi derimot marginal til ingen forskjell i presisjon. Det LSTM taper på mengden minne og hastighet gjør den opp i med å være mer nøyaktig på prediksjoner på de større og fyldige datasettene enn GRU. Dette stemmer med de oppnådde resultatene for gruppen, hvor LSTM har hatt noe bedre presisjon enn GRU ved enhver kjøring. Hvorvidt denne økningen er verdt den ekstra kompleksiteten og minnebruken som internminnet til LSTM presenterer, må tas til betraktning for hvert enkelt tilfelle. Her vil tilgjengelig hardware og tid være utslagsgivende for en beslutning. Den samme problemstillingen vil også være aktuelle når en avgjør om en vil ta i bruk bidirectional RNN, fremfor den vanlige enveismetoden. Her er forskjellen i presisjon mindre sammenlignet med differansen mellom GRU og LSTM, samtidig som de fortsatt er betraktelig mer tidkrevende.

Presisjonen på det reduserte datasettet blir tilnærmet identisk på tvers av RNN modellene, og Naive Bayes. Ved å analysere forvirringsmatrisene har rapporten avdekket tydelige forskjeller ved klassifiseringen av det fullstendige datasettet opp mot det reduserte. Det er tydelig når man ser på tilfellene av feil for datasettet med 235 språk, at det er enkelte språk som er spesielt vanskelig å skille mellom. Modellene predikerer riktig i de aller fleste tilfeller, dog vil presisjonen bli dratt ned spesielt mye i de tilfellene hvor den konsekvent gjetter feil av to språk. For eksempel kan man trekke frem Høysorbisk, som modellen feilaktig predikerte 29 og 19 ganger for henholdsvis Kroatisk og Bosnisk. Slike særtilfeller er med på å dra presisjonen ned. Ved å redusere antall språk fra 235 til 23, ser vi at presisjonen øker til 99 prosent. En av faktorene er naturligvis at antallet mulige alternativer modellen kan gjette, reduseres betraktelig. Ved å se nærmere på forvirringsmatrisen, ser man at maksimalt antall forekomstert av feilaktig predikering mellom to språk nærmer seg null. Dette betyr i praksis at modellen ikke lenger påvirkes av at nært beslektede språk trekker ned presisjonen i samme grad.

Det var også gode konklusjoner som kunne trekkes etter testing av modellene på testsettet med variabel inputlengde. Først og fremst så var det en rask eksponensiell økning for både Naive Bayes og RNN. Testen på BLSTM modellen med 512 i accuracy, divergerte i det testdataen ble opp til 200 karakterer lange. Det er derfor tydelig at man er avhengig av å ha setninger eller paragrafer på mellom 30-50 ord før man kan forvente de beste mest presise resultatene. Selv om MNB her også hadde lavere accuracy enn RNN, kom det likevel frem at den kunne benyttes på en hensiktsmessig måte. Allerede før karaktergrensen nådde 100, hadde n-gram modellen 90 prosent presisjon. I tilfeller hvor en ønsker å predikere kortere setninger, viste derfor MNB seg å være overlegen i vår implementasjon.

Konklusjon

Denne rapporten har tatt for seg en empirisk vurdering av Multinomial Naive Bayes (MNB), Long Short Term Memory (LSTM) og Gated Recurrent Unit (GRU) for klassifisering av språk. Det er også gjort rede for relevant teori som ligger til grunne for anvendelsen av de ulike modellene. Rapporten har gått dypere inn på RNN, og sett på både på uni- og bidirectional

implementasjon av LSTM og GRU. Rapporten svarer også på hvordan modellene påvirkes av antall språk i klassifiseringen, samt begrensing av inputlengde.

For det fullstendige datasettet var det RNN-modellene som viste seg å være det overlegne alternativet. Hidden size 512 ga klart beste resultater fra hva som ble testet. Videre utprøvning av enda høyere verdier vil være hensiktsmessig, det lot seg derimot ikke gjøre med tilgjengelig maskinvaren i dette prosjektet. Kjøringene var tidkrevende, og kjøretid og minnebruk økte betraktelig sammen med økt hidden size. Likevel har resultatene vist at å øke hidden size fra 256 til 512 absolutt er hensiktsmessig, dersom man har tilstrekkelig hardware. Bidirectional implementasjon ga noe økning i accuracy, men ingen merkbar forskjell i loss. Selv om denne modellen ga noe bedre resultater, er det fortsatt uklart om en fordobling i kjøretid er verdt den marginale økning i presisjon. LSTM presterte best av de to RNN-modellene, selv om GRU sin presisjon var veldig sammenlignbar med sin mer tidkrevende motpart.

MND måtte ikke bare se seg slått på presisjon, men modellene som oppnådde høyest presisjon brukte også svært stor fysisk lagringssplass. På det reduserte datasettet viste derimot MND seg som svært anvendelig. Her ga den nær identisk presisjon som RNN-modellene, samtidg som problemene knyttet opp mot lagringssplass og minnebruk ble løst. Dette gjorde det også mulig å endre n til verdier større enn 3, hvor spesielt $n = 4$ ga god økning. Ved enda kraftigere hardware tilgjengelig, kunne det vært interessant å se om denne trenden fortsatt for det fullstendige datasettet. En siste kjøring på modellene med varierende input lengde viste at MND var langt bedre på å predikere korte setninger, enn hva resultatene fra RNN-modellene viste.

Hensikten med prosjektet har i stor grad utviklet seg til å utforske hvordan en kan anvende maskinlæringsmodeller for å oppnå gode resultater på forskjellige områder av språkidentifikasjon. Rapporten har bevist at alle modellene som er presentert, har sine respektive områder de kan anvendes. En interessant utvidelse av oppgaven kan være å se hvordan en kombinasjon av de forskjellige metodene sine styrker, kan brukes for lage en overlegen klassifiksjonsmodell. MNB sin evne til å klassifisere språk på selv relativt korte setninger, kombinert med RNN sin overlegne presisjon på det store datasettet. Det ville også vært svært interessant å utforske en løsning for tap av presisjon ved nært beslektede språk. Uavhengig av potensialet til en slik anvendelse, har denne studien presentert en detaljert beskrivelse av flere utfordringer og løsninger i språkidentifikasjon.

Anerkjennelser

Martin Johannes Nilsen sitt bidrag kan ikke gå unevnt. Hans kompetanse og jevnlige veiledning har vært uvurderlig for gruppens resultater i prosjektarbeidet.

References

[1] D. AI. Recurrent nueral networks. URL http://d2l.ai/chapter_recurrent-neural-networks/rnn.html.

[2] K. Ganesan.

[3] A. Graves. Supervised sequence labelling with recurrent neural networks. studies in computational intelligence. URL https://www.cs.toronto.edu/~graves/preprint.pdf.

[4] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. neural networks, 18(5-6):602–610. URL https://pubmed.ncbi.nlm.nih.gov/16112549/.

[5] K. C. G. P. I. Androutsopoulos, J. Koutsias and C. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. URL https://dl.acm.org/doi/abs/10.1145/345508.345569.

[6] . G. B. B. D. F. B. H. S. Y. B. K. Cho, B. van Merrienboer. Learning phrase representations using rnn encoder-decoder for statistical machine translation. URL https://arxiv.org/abs/1406.1078.

[7] G. Learning. Multinomial naive bayes explained. URL https://www.mygreatlearning.com/blog/multinomial-naive-bayes-explained.

[8] Nvidia. Cuda. URL https://developer.nvidia.com/accelerated-computing-training.

[9] M. H. Pedamallu. Rnn vs gru vs lstm. URL https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573.

[10] T. D. S. M. Phi.

[11] Pytorch. Gru, . URL https://pytorch.org/docs/stable/generated/torch.nn.GRU.html.

[12] Pytorch. Lstm, . URL https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[13] J. S. S. Hochreiter. Long short-term memory. neural computation, 9(8):1735–1780. URL https://www.researchgate.net/publication/13853244_Long_Short-term_Memory.

[14] M. Thoma. Wili-2018. URL https://zenodo.org/record/841984#.YZz5pL3MI6E.

[15] N. M. Towards Data Science. Splitting a dataset. URL https://towardsdatascience.com/splitting-a-dataset-e328dab2760a.

[16] H. Zhang. Naive bayes text classifier. URL https://ieeexplore.ieee.org/abstract/document/4403192.