Norges miljø- og biovitenskapelige universitet

Masteroppgave 2019 30 stp
Fakultet for realfag og teknologi

Solslyng i jernbanespor – En mulighetsstudie for å forutsi solslynghendelser ved hjelp av maskinlæring

Sun kinks in railway tracks – A feasibility study for predicting sun kinks using machine learning

Elise Tegnér
Geomatikk

Forord

Denne masteroppgaven er skrevet ved Fakultet for realfag og teknologi ved Norges miljø- og biovitenskapelige universitet i Ås våren 2019. Den markerer slutten på sivilingeniørstudiet i Geomatikk med spesialisering i GIS og datavitenskap.

Temaet i oppgaven ble utarbeidet under en sommerjobb sammen med andre studenter hos Geodata AS, og går ut på å bruke maskinlæring for å forsøke å forutsi solslynghendelser på jernbanen. Solslyng oppstår når temperaturen i stålet i jernbaneskinnene blir så høy at de "knekker ut" til siden. Dette kan i beste fall forårsake ristinger og ubehag for passasjerer på tog. I verste fall kan det føre til togavsporinger og ulykker.

Jeg vil først og fremst takke veilederne mine som har støttet meg i prosessen med skrivingen og hatt tro på oppgaven. Jeg vil takke sjefingeniør Alf Helge Løhren i Bane NOR for god hjelp til å sette seg inn i nødvendig kunnskap om jernbane og solslyng, førsteamanuensis Oliver Tomic og Kristian Hovde Liland for gode tips og hjelp i forhold til maskinlæringen, og førsteamanuensis Håvard Tveite for oppfølging av det GIS-faglige.

Jeg vil også takke de ansatte på "Teknologi og regelverk" i Bane NOR for god hjelp til å sette seg inn i jernbane-databasen og svar på diverse spørsmål om hvordan jernbanen er satt sammen og fungerer.

Videre vil jeg rette en spesiell takk til Geodata AS for temaet på oppgaven, kontorplass, lån av PC, samt hjelp i forhold til GIS og programvare innen GIS.

I tillegg vil jeg takke Alix Cordray for diskusjoner og støtte under skrivingen, samt familie og venner.

Oslo, 14. mai 2019

Elise Tegnér

Sammendrag

Hensikten med oppgaven er å undersøke og prøve å forutsi solslynghendelser på jernbanen ved hjelp av maskinlæring. Solslyng oppstår når temperaturen i stålet i jernbaneskinnene blir så høy at de ”knekker ut” til siden. Dette kan forårsake ristinger og ubehag for passasjerer på tog i beste fall. I verste fall kan det føre til togavsporinger og ulykker. Maskinlæring handler om å bruke ulike algoritmer til bruke kjent informasjon til å forutsi ny og ukjent informasjon. Det vil være til stor nytte for Bane NOR om det gikk an å varsle solslynghendelser *før* de skjer, noe som vil spare dem og det norske samfunnet for mye penger og kanskje unngå at menneskeliv går tapt.

Det ble ikke kommet fram til maskinlæringsmodeller som er gode nok til å kunne brukes i praksis til å predikere solslyng.

Oppgaven kan sies å være tredelt, der den første delen handler om datainnsamling, den neste om bearbeiding og utforsking av dataene som er funnet og den siste delen bruke maskinlæring til å predikere solslyng. Oppgaven er avgrenset til å undersøke solslyng på Gjøvik-banen, Roa-Hønefossbanen og Hovedbanen mellom Oslo S og Lillestrøm for sommerhalvårene 2017 og 2018. Dette utgjør 55 solslyng å 167 kilometer med jernbane over 366 dager.

På datainnsamlingsdelen ble det samlet inn og bearbeidet så mye data som mulig som kunne tenkes å ha innvirkning på risikoen for solslyng. Dette gjaldt blant annet vedlikehold og komponenter på jernbanen som har påvirkning på den sidevegs stivheten i sporgeometrien. Det ble også samlet inn og laget geografiske data, som blant annet radius i svingene på jernbanen og hvor mye solinnstråling som slipper til på de ulike delene av jernbanen i forhold til vegetasjon, bygninger og terreng. I tillegg er det samlet inn værdata som temperatur, nedbør, vindhastighet og skydekke.

I bearbeidingen og utforskingen av de innsamlede dataene ble det undersøkt om de ulike variablene som var laget kunne skille solslyng-tilfeller fra tilfeller hvor det ikke var solslyng. Det ble også brukt prinsipalkomponentanalyse for å undersøke om prinsipalkomponentene kunne skille ut solslyngene. Både ut fra variablene og ut fra prinsipalkomponentene ser det ut som at det er full overlapp mellom solslyngtilfellene og ikke-solslyngtilfellene.

Det ble testet ut 5 ulike maskinlæringsalgoritmer med ulike parametere. Disse var Random Forest, logistisk regresjon, Support Vector Classification (SVC), Multi-Layer Neural Network og Recurrent Neural network. I tillegg ble det gjort et forsøk med anomali-deteksjon med Isolation Forest. De beste resultatene ble oppnådd med Multi-Layer Neural Network-modell med en test F1-score på 0.04. Ved testing av denne modellen klarte den å finne 4 av 17 solslyngtilfeller, mens 263 ikke-solslyngtilfeller ble feilaktig klassifisert som solslyng. Dette sier seg selv at ikke er bra nok til å kunne brukes i praksis.

Oppgaven avsluttes med et kapittel med anbefalinger på hvordan denne oppgavens problemstilling kan utforskes videre, på bakgrunn av de erfaringene og den kunnskapen som har blitt tilegnet gjennom arbeidet med denne oppgaven.

Abstract

The purpose of this master's thesis is to do a feasibility study for predicting sun kinks in railway tracks using machine learning. Sun kinks happen when the temperature in the rail steel increases so much that the rails buckle. In the best case, this can cause shaking and discomfort for the train travelers. In the worst case, it can cause the train to derail and cause accidents. Machine learning is about using different algorithms to obtain knowledge from well known information and predict new and unknown information. The Norwegian company responsible for the national railway infrastructure, Bane NOR, would benefit from automatic prediction of sun kinks *before* they happen. This would save them and Norwegian society a significant amount of money and perhaps save human lives.

No machine learning models were found that were good enough to predict sun kinks in practice.

This thesis consists of three parts. The first part is about data collection, the next about processing and exploring the collected data, while the last part uses machine learning to predict sun kinks. This thesis considers the Gjøvik line, Roa-Hønefoss line and the Hoved line between Oslo Central Station and Lillestrøm. The time considered is April-September 2017 and 2018. The data is for 167 kilometers of railway tracks over 366 days, and contains 55 sun kinks.

During the data collection part, as much data as possible that could have an impact on the risk for sun kinks was collected. The data collected was about maintenance and different parts of the railway track which influence the lateral resistance to buckling. Geographic data, such as the radius of curves and how much solar radiation hits the tracks (influenced by vegetation, buildings and terrain) was also collected. In addition, different types of weather data, such as temperature, precipitation, wind speed and cloud area were collected.

The processing and exploration of the data consisted of an examination of whether the different variables could distinguish the sun kink samples from the non-sun kink samples. Principal component analysis was also used to see if the principal components could distinguish between the samples. From both the variables and the principal components it appears that the sun kink samples and the non-sun kink samples can not be separated.

5 different machine learning algorithms with different parameters where tested. Those were Random Forest, logistic regression, Support Vector Classification (SVC), Multi-Layer Neural Network and Recurrent Neural Network (RNN). In addition, the algorithm Isolation Forest was tried as an anomaly detection algorithm. The best result was obtained by a Multi-Layer Neural Network with a test F1-score of 0.04. By testing this model, it was able to find 4 out of 17 sun kink samples, while 263 non-sun kink samples were classified wrongly as sun kinks. This shows that the models are not accurate enough to be used in practice.

The thesis' ending chapter offers some recommendations on how the topic of this thesis can be explored further. The recommendations are based on the knowledge and experiences obtained through working with this thesis.

Innhold

Definisjoner

Baksefeil	En skinnestrengs avvik fra ideell beliggenhet i horisontalplanet
Ballast	Jernbanesporets underlag av pukk
Banedata	Bane NORs vedlikeholdssystem og infrastrukturdatabase
Banekartet	Applikasjon for visualisering av jernbanesporene i et kart
Befestigelse	Festeanordningen mellom skinner og sviller
Hovedspor	Hovedsporet på en jernbanestrekning, altså ikke sidespor, togspor i forbindelse med flere plattformer på stasjoner eller kryssningsspor.
jernbanekompetanse.no	Wiki-side med lærebøker i jernbaneteknikk
Km-segment	En bit av en jernbanestrekning med lengde ca 1 km, refererer også til et segment av det geografiske linjedatasettet ”km-segmenter”
Kryssvalidering (k fold)	K systematiske oppdelinger av et datasett i treningssett og valideringssett.
Målevognbilder	Bilder av jernbanesporet
Pilhøyde	Den horisontale avstanden mellom skinnens kjørekant og midtpunktet til en 10 eller 20 m lang korde. Kordens ender følger skinnens kjørekant.
Pilhøydefeil	Pilhøydens avvik i mm fra ideell pilhøyde
Skinne	Selve skinnestrengen
Skinneprofil	Type skinne, formen på tverrsnittet til skinnen
Solslyng	Pilhøydefeil på mer enn 25 mm, forårsaket av høy temperatur i skinnene
Spor	Sporstigen av skinner og sviller
Sporkilometer	I oppgaven er Bane NORs system for km-notasjon på jernbanesporene brukt. Alle linjer har en km-betegnelse for entydige plasseringer

	langs jernbanesporene, vanligvis antall kilometer fra Oslo S. For eksempel vil km 7.2 på Gjøvikbanen være jernbanesporet på Gjøvikbanen i en avstand fra Oslo S på ca 7.2 km.
Stokkskinneskjøt	Overgangen mellom skinne og begynnelsen av en sporveksel
Sville	Skinnenes tverrgående underlag i betong eller tre.
Supervised	At en maskinlæringsalgoritme bruker et testsett til å bestemme hvordan analysen eller læringen skal gjøres.
Teknisk Regelverk	Bane NORs tekniske regelverk for bygging, prosjektering og vedlikehold av jernbaneinfrastrukturen
Terrengmodell	Et raster-datasett med høydeverdier (moh) tilsvarende ”overflaten” til terrenget, det vil si tretopper og toppen annen vegetasjon, hustak osv.
Testsett	Datagrunnlaget som brukes til prediksjon
Treningssett	Datagrunnlaget som brukes for trening av en maskinlæringsmodell
Unsupervised	At en maskinlæringsalgoritme ikke bruker testsett til å bestemme hvordan analysen eller læringen skal foregå.

Kapittel 1

Innledning

1.1 Bakgrunn

Maskinlæring har de siste årene blitt mer og mer populært, og bruksområdet utvides stadig til nye problemstillinger. Sommeren 2018 var en tid preget av uvanlig høy temperatur, mye solskinn og dermed også uvanlig mange solslynghendelser på jernbanen. I en sommerjobb den sommeren satt et team av studenter og skulle finne opp et konsept på hvordan teknologi kan gjøre samfunnet bedre, "Tech for Society". Med stadige forsinkelser og buss for tog på jernbanen for en av studentene som pendlet, kom idéen opp om ikke maskinlæring også kan brukes til å forutsi solslynghendelser. Resultatet ble et konsept der ulike værdata, samt informasjon om jernbanen og terreng skulle kunne brukes til dette. Konseptet gikk også ut på lage et system for kontinuerlig prediksjon i sanntid, automatisk varsling og et samhandlingssystem mellom ulike aktører som blir berørt, som for eksempel passasjerer, lokførere og vedlikeholdsarbeidere.

For teamet av studenter forble ideen kun et konsept, men én av studentene gikk videre med dette som sin masteroppgave, som har resultert i denne oppgaven.

I tillegg til at solslyng bidrar til forsinkelser på jernbanen, kan i verste fall solslyng også forårsake tog-avsporinger som kan koste det norske samfunnet millioner av kroner. Solslyng er dessuten ikke alltid så lett å oppdage, da det i dag ikke finnes noe automatisk varsling for dette. Derfor vil det være av betydning for samfunnet å undersøke muligheter for bedre varslingssystemer i forhold til solslyng.

Maskinlæring handler blant annet om å forutsi ny ”ukjente” data basert på gamle og ”kjente” data. Når det gjelder solslyng passer maskinlæring godt inn i problemstillingen om prediksjon av solslynghendelser. Maskinlæringsalgoritmer kan trene på historiske data om solslyng, for så å predikere hvor og når solslyng evt vil oppstå i framtiden basert på ny data.

1.2 Mål og hensikt

Det overordnede samfunnsnyttige målet med oppgaven er som nevnt ovenfor å undersøke muligheter for et bedre system for å forutsi solslyng på jernbanen, enn det som finnes i dag. I denne oppgaven er det maskinlæring som er brukt som verktøy for å prøve å oppnå dette målet.

Målet med oppgaven blir derfor å undersøke om det er mulig å forutsi solslynghendelser på jernbanen ved hjelp av maskinlæring, gitt det datagrunnlaget som finnes tilgjengelig og er realistisk å få tak i innenfor rammene av en masteroppgave. Det er også interessant å undersøke eventuelt hvilke variabler som bidrar best til prediksjonen.

Om modellen skulle vise seg å gi gode resultater, var også målet å lage en modell som kan brukes av Bane NOR til kontinuerlig prediksjon og varsling av solslyng.

1.3 Fagområder

Denne masteroppgaven er tverrfaglig, i forhold til at både jernbane-faglig, GIS-faglig og datavitenskapelig kunnskap har blitt brukt.

Når det gjelder det jernbanefaglige har prosessen med datainnsamling handlet om å lese seg opp på teori om solslyng og de ulike komponentene på jernbanen. Det har vært nødvendig for å ta fornuftige valg i forhold til hvordan informasjonen om jernbanen kan utnyttes best mulig i maskinlæringen.

Kunnskapen om GIS som har blitt brukt har bestått mest i praktisk bearbeiding av geografiske data. Dette har blant annet bestått i utregning av geometriske størrelse, både euklidiske avstander, avstander langs kurver og retning i forhold til nord, samt koordinater for km-segmentene både i UTM sone 32N og WGS84. Det har også gått ut på bruk av en terrengmodell til å beregne høyder, mengde solinnstråling som treffer de ulike delene av jernbanen.

Av datavitenskapelig kunnskap er det eksplorativ analyse og maskinlæring som har vært mest brukt. Det har også vært nødvendig å bruke kunnskap fra statistikk for tolkning av resultatene fra svært ubalanserte datasett.

1.4 Avgrensning

Oppgavens avgrensing i rom (hvor mange kilometer jernbane) og tid (hvor mange dager), har blitt gjort med tanke på hvor godt datagrunnlag det er mulig å skaffe innenfor tidsrammen for en masteroppgave. For maskinlæringen sin del er det en fordel at datagrunnlaget er mest mulig balansert med tanke på solslyng. Dette vil si høyest mulig frekvens av solslyng pr. kilometer banestrekning og i forhold til antall dager. I tillegg vil det være en fordel å bruke nyest mulig data, da ulike komponenter på jernbanen kan bli byttet ut og det ikke vil gi så mye framtidig nytte å arbeide med jernbanestrekninger som ikke lenger finnes. Dessuten er kvaliteten på dataene gjerne bedre for nyere data enn for eldre data.

Basert på dette ble oppgaven i tid avgrenset til tidsperioden 1. april til 30. september 2017 og 1. april til 30. september 2018.

Som avgrensning i rom har 3 banestrekninger blitt valgt ut på grunn av høy frekvens av solslynghendelser i forhold til antall kilometer, se tabell 1.1.

Bane	**Stasjoner fra-til**	**Antall spor**	**Km fra**	**Km til**
Gjøvikbanen	Grefsen til Gjøvik	1	7.000	123.913
Hovedbanen	Ca Bryn til Lillestrøm	2	3.000	19.732
Roa-Hønefossbanen	Roa til Hønefoss	1	58.428	90.600

Tabell 1.1: Geografisk avgrensning av oppgaven.

Kun jernbanens hovedspor er tatt med, det vil si at alle sidespor, krysningsspor, togspor osv er utelatt. Det er valgt å starte på km 7 på Gjøvikbanen for å kun få en enkeltsporet strekning. Gjøvikbanen er definert til 123.913 i Banedata. Km 3 og utover på Hovedbanen er valgt, da det manglet data på km 2, og for å kun få strekning med dobbeltspor. Hovedbanen før Lillestrøm er definert til km 19.732 i Banedata. Roa-Hønefossbanen er definert fra km 58.428 til km 90.6 i Banedata. Sporveksler i forbindelse med stasjoner er tatt med om de ligger på hovedsporet.

1.4.1 Avgrensning i forhold til maskinlæring

I maskinlærings-problemstillinger må det alltid tas stilling til begrensninger med hensyn på tid og regnekraft. Jo flere ressurser man har i forhold til tid og regnekraft jo bedre muligheter har man for å finne gode maskinlæringsmodeller som passer til datasettet og problemstillingen. For å tilpasse oppgaven innenfor rammene av en masteroppgave, er maskinlæringen avgrenset til uttesting av følgende maskinlæringsalgoritmer:

- Random Forest
- Isolation Forest
- Logistisk regresjon
- Support Vector Classification
- Multi-Layer Neural Network
- Recurrent Neural Network (RNN)

1.5 Oppgavens oppbygning og struktur

Etter innledningen av oppgaven, er det et kapittel om det teoretiske grunnlaget for oppgaven. Dette kapittelet har til hensikt å redegjøre for den kunnskapen som er brukt og som ligger til grunn for de ulike valgene som er gjort i oppgaven. Deretter, i kapittelet ”Metode - verktøy”, er det skrevet om hvilke verktøy som er brukt i oppgaven. Dette gjelder først og fremst hvilke programvarer og APIer som er brukt til å løse oppgaven.

Videre er det i ”Metode - Datainnsamling” skrevet om hvordan datainnsamlingen har foregått, hvilke valg som er tatt og om dataene som er samlet inn.

I kapittelet ”Metode - Analyse” blir framgangsmåtene for å utføre maskinlæringen redegjort for. Det er også en del som omhandler hvordan det er utført eksplorativ analyse av dataene for å bygge opp bedre intuisjon og forståelse om strukturen i dataene. Dette var for å hjelpe og understøtte valg og tolkninger av maskinlæringen og resultatene fra maskinlæringen.

Resultatene av den eksplorative analysen og maskinlæringen blir gjennomgått i kapittelet ”Resultater”.

1.6 Tidsbruk

Det ble prioritert å lage så godt datagrunnlag som mulig, da prestasjonene til maskinlæringsalgoritmene automatisk vil speile kvaliteten på dataene. Derfor ble majoriteten av arbeidsinnsatsen og arbeidstiden satt inn på datainnsamlingsdelen.

Noe av tiden er brukt på eksplorativ analyse i etterkant av datainnsamlingen. De eksplorative analysene ble gjort for å få en bedre oversikt over datasettet og for å bygge opp intuisjonen når det gjelder valg av modeller og parametre i maskinlæringen.

Maskinlæringsdelen av oppgaven ble gjort helt på slutten og det ble prøvd ut så mange ulike maskinlæringsteknikker og maskinlæringsmodeller som mulig innenfor tidsrammen.

1.7 Begreper brukt i oppgaven

De fleste ord som trenger ytterligere forklaring står i ”Definisjoner” like etter innholdsfortegnelsen. Det har likevel blitt valgt å klargjøre bruken og betydningen bak noen mye brukte begreper her.

Når det er skrevet ”datasettet”, betyr det i denne oppgaven datasettet som er satt sammen av alle variablene som er resultat av datainnsamlingen og som er ferdig designet til å mates inn i en maskinlæringsmodell. Begrepet ”treningssett” er brukt om den delen av datasettet som maskinlæringsmodellen trener på, men ”testsettet” er den delen av datasettet som maskinlæringsmodellen prøver å predikere.

”Km-segment” referer til en strekning på ca 1 km på jernbanen, uavhengig av tid. Når det gjelder bruk av ordet ”et tilfelle”, refererer dette til et km-segment på en bestemt dato. Datasettet i denne oppgaven inneholder altså da informasjon om 167 km-segmenter (167 km med jernbane) og 61122 tilfeller (167 km-segmenter representert på 366 ulike dager). Et solslyng-tilfeller viser da til et km-segment på en spesifikk dag da det i Banedata er registrert solslyng eller tilløp til solslyng. Det motsatte gjelder for et ikke-solslyng-tilfelle.

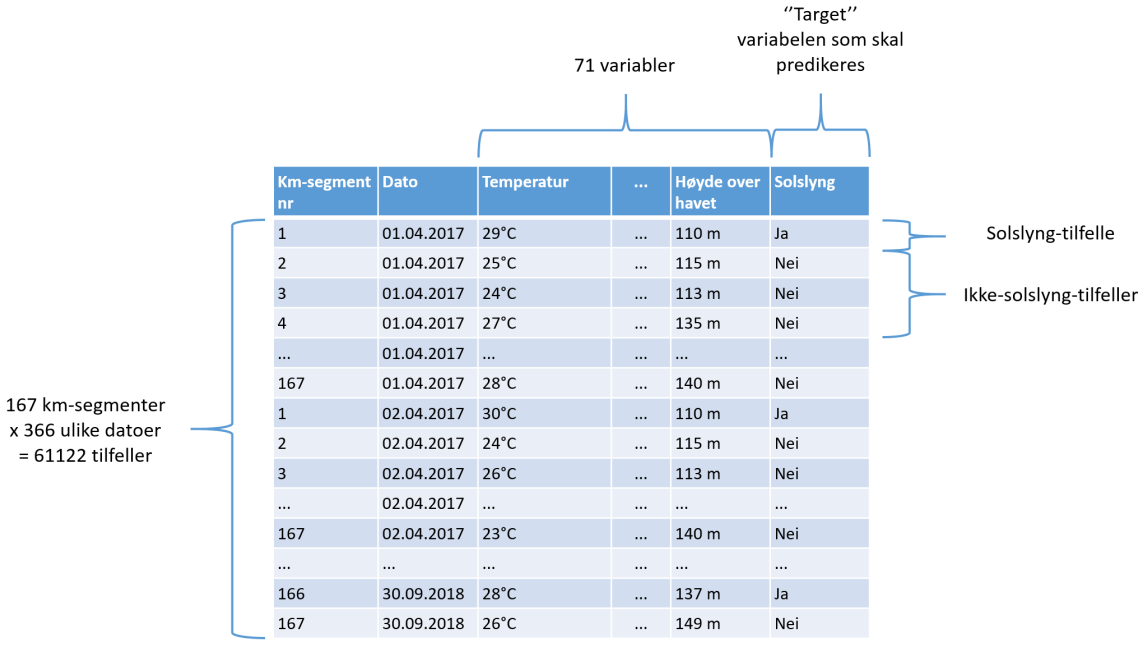
”Km-segment-datasettet” referer til et geografisk linjedatasett i shape-filformat, hvor hvert linje-segment representerer plasseringen til hver av de 167 km-segmentene.

Bane NOR har et eget sporkilometer-system for å definere hver enkelt del av jernbanen. Dette fungerer

ved at hver del av jernbanen har et eget km-nummer. Som regel svarer dette nummeret til omtrent antall kilometer langs jernbanen fra Oslo S. Km 4 på Gjøvikbanen vil da si en strekning på 1 km som ligger i en avstand langs jernbanen på 4-5 km fra Oslo S.

Når det gjelder maskinlæringen er begrepet maskinlæringsalgoritme brukt om en generell maskinlæringsmetode som gjerne er ferdig implementert og tilgjengelig via et API. Begrepet maskinlæringsmodell er brukt om en maskinlæringsalgoritme hvor de ulike parameterene er blitt definert, og som brukes til å trene på treningssettet, predikere på testsettet og kan også avgi et eller flere prestasjonsmål for å fortelle hvor tett opp til "fasiten" predikeringen ble.

En illustrasjon av utformingen til datasettet med begreper brukt i oppgaven vises i figur 1.1. Treningssettet blir da tilfeldige utvalgte rader fra datasettet som maskinlæringsmodellen skal trene på. Testsettet blir de resterende radene hvor "target", Solslyng-variabelen, er tatt ut. Maskinlæringsmodellens prediksjoner på testsettet sammenlignes med verdiene i Solslyng-variabelen for måling av maskinlæringsmodellens prestasjon og treffsikkerhet.



Km-segment nr	Dato	Temperatur	...	Høyde over havet	Solslyng
1	01.04.2017	29°C	...	110 m	Ja
2	01.04.2017	25°C	...	115 m	Nei
3	01.04.2017	24°C	...	113 m	Nei
4	01.04.2017	27°C	...	135 m	Nei
...	01.04.2017	...	...	...	...
167	01.04.2017	28°C	...	140 m	Nei
1	02.04.2017	30°C	...	110 m	Ja
2	02.04.2017	24°C	...	115 m	Nei
3	02.04.2017	26°C	...	113 m	Nei
...	02.04.2017	...	...	...	...
167	02.04.2017	23°C	...	140 m	Nei
...	...	...	...	...	...
166	30.09.2018	28°C	...	137 m	Ja
167	30.09.2018	26°C	...	149 m	Nei

Figur 1.1: Illustrasjon av datasettets utforming med eksempelverdier og -variabler, samt begreper brukt i oppgaven. Laget i MS Powerpoint.

Kapittel 2

Teoretisk grunnlag

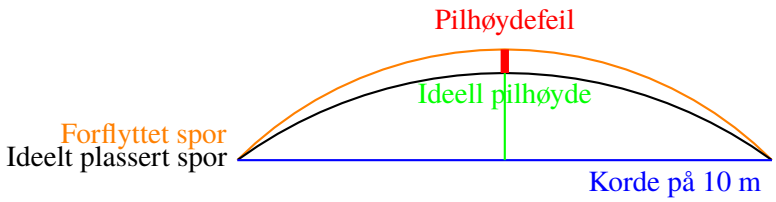
Det teoretiske grunnlaget for masteroppgaven har et praktisk og anvendelig fokus, og har til hensikt å gjøre rede for kunnskapen og forståelsen som ligger til grunn for de ulike valgene som er tatt i oppgaven. Da masteroppgaven er av en praktisk art, vil det for eksempel ikke bli presentert mange dype matematiske utregninger, men heller praktisk og anvendbar kunnskap for å understøtte de valgene som er gjort i oppgaven.

2.1 Om solslyng

Solslyng oppstår når jernbaneskinnene utvider seg så mye på grunn av varme, at skinnene ”bøyer seg” og forskyves lateralt (sidevegs). Solslyng måles ved å måle pilhøyden, hvor pilhøydefeil på større enn 25 mm klassifiseres som solslyng.

Pilhøyden i en skinne måles ved å legge en (vanligvis) 10 m lang korde hvor endene berører skinnen på ulike steder langs sporet. Pilhøyden er da avstanden mellom midtpunktet til korden og skinnen (Bane NOR mfl., 2019). I en kurve med konstant radius vil pilhøyden også være konstant. Pilhøydefeil er da avviket mellom faktisk pilhøyde i skinnen og ideell pilhøyde, se figur 2.1 hvor pilhøydefeilen er markert med rødt.

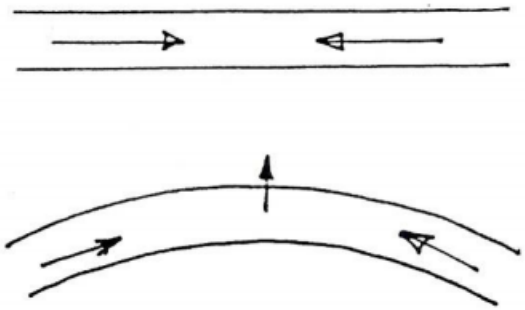
I Norge er det stor variasjon i temperatur gjennom året. Jernbanesporet må derfor tåle store temperatur-forskjeller. Stålet i jernbaneskinnene er følsomt for dette og vil derfor utvide seg og trekke seg sammen, for det meste aksialt (i lengderetningen). I dag er det aller meste av jernbanespor i Norge



Figur 2.1: Pilhøydefeil. Figur laget med TikZ (Open source, 2019)

helsveist, som vil si at skinnene er sveiset sammen i stedet for at de er skjøtet sammen med lasker. Dette gir ikke det samme rommet for skinnene til å forlenges og forkortes som følge av temperaturendringer som de gamle laskede sporene hadde. Helsveist spor kan derfor ha større risiko for solslyng og skinnebrudd enn lasket spor, men gir til gjengjeld høyere komfort for togreisende og er mye enklere å vedlikeholde (sst.).

De aksiale kreftene i skinnene vil variere gjennom et år og gjennom et døgn som følge av temperaturendringer. Faren for solslyng er størst i kurver, siden kraftretningen da vil peke utover i svingen, og ikke bare langs skinnen, som vist i figur 2.2.



Figur 2.2: Kraftretning i kurver versus kraftretning på rette strekninger. Figur hentet fra (Sørlie, 2008) side 10.

Det som skal hindre skinnene i å forskyve seg lateralt er tilstrekkelig sideforskyvningsmotstand i sporet sammen med korrekt nøytraltemperatur i skinnene. Nøytraltemperaturen i skinnene er den temperaturen hvor det ikke er noen aksiale spenninger i stålet, og skal i Norge ligge på ca 21 °C.
Sideforskyvningsmotstanden i sporet består av friksjon mellom sville og ballast, ballastskulderen, skinnenes stivhet sidevegs og dreiemotstanden i befestigelsen mellom skinne og sville (Sørlie, 2008). Om nøytraltemperaturen i skinnene er riktig og sporets komponenter, ballasten og underbygningen er i henhold til forskriftene skal det teoretisk sett ikke kunne oppstå solslyng gitt de temperaturforskjellene vi har i Norge. Grunnen til at solslyng likevel oppstår er derfor at nøytraltemperaturen er feil og/eller at

det ikke er tilstrekkelig sideforskyvningsmotstand i sporet.

Nøytraltemperaturen kan endres ved at sporet forflytter seg sidevegs, for eksempel som følge av ikke tilstrekkelig kontroll av sporets beliggenhet før og etter vedlikehold. Sporet kan også forlytte seg innover i kurver når skinnene trekker seg sammen ved lave temperaturer om vinteren om stabiliteten i underbygningen og i ballasten er for dårlig.

2.2 Eksplorativ analyse

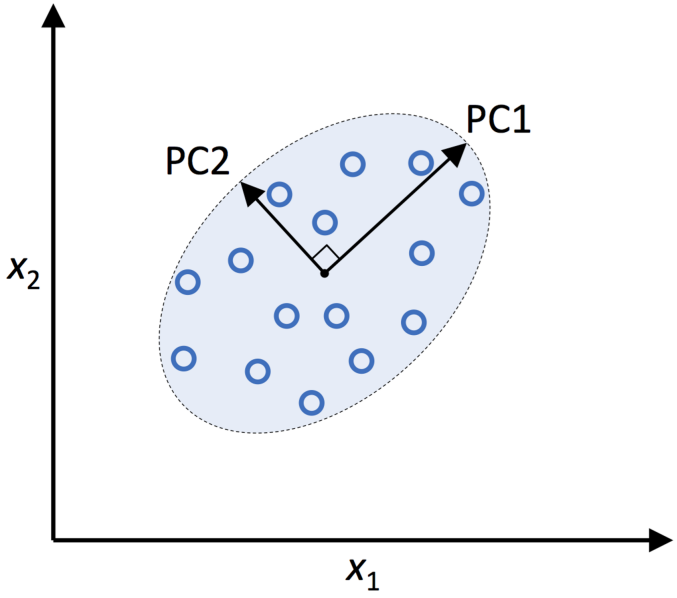
I eksplorativ analyse-delen av oppgaven er for det meste histogrammer og prinsipalkomponentanalyse (PCA) brukt.

2.2.1 Prinsipalkomponentanalyse (PCA)

Prinsipalkomponentanalyse (videre PCA) er en unsupervised metode og blir brukt mye både innenfor maskinlæring for dimensjonalitetsreduksjon (dimensionality reduction) og innenfor mer "tradisjonell" multivariat analyse for å undersøke blant annet trender og mønstre i multivariate datasett. Det at det er en unsupervised metode, vil si at man undersøker hele datasettet på en gang. Man deler ikke opp datasettet i trenings- og testsett og man har dermed ikke at noe testsett blir styrende for hvordan PCAen skal utføres.

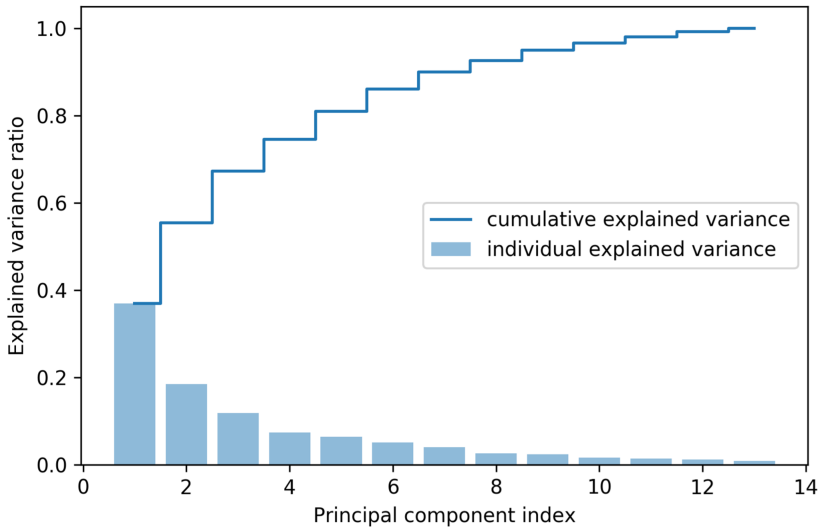
Ved en PCA blir variablene i datasettet transformert til prinsipalkomponenter som erstatter de opprinnelige variablene. En veldig forenklet forklaring på hva som skjer i en PCA er at man bruker datasettets kovarians-matrise og egenverdier slik at den største delen av variansen i datasettet blir forklart av den første prinsipalkomponenten, litt mindre i den neste komponenten osv. Fra et matematisk ståsted ville disse prinsipalkomponentene være datasettets egenvektorer hvorav egenvektorenes egenverdier i synkende rekkefølge bestemmer egenvektorenes rekkefølge som prinsipalkomponenter.

Grafisk kan dette illustreres slik som i figur 2.3, der X_1 og X_2 symboliserer to variabler i et datasett, $PC1$ den første prinsipalkomponenten som strekker seg langs aksen med størst varians i datasettet og $PC2$ den andre prinsipalkomponenten som skal stå 90 °på første prinsipalkomponent i retningen med størst varians.



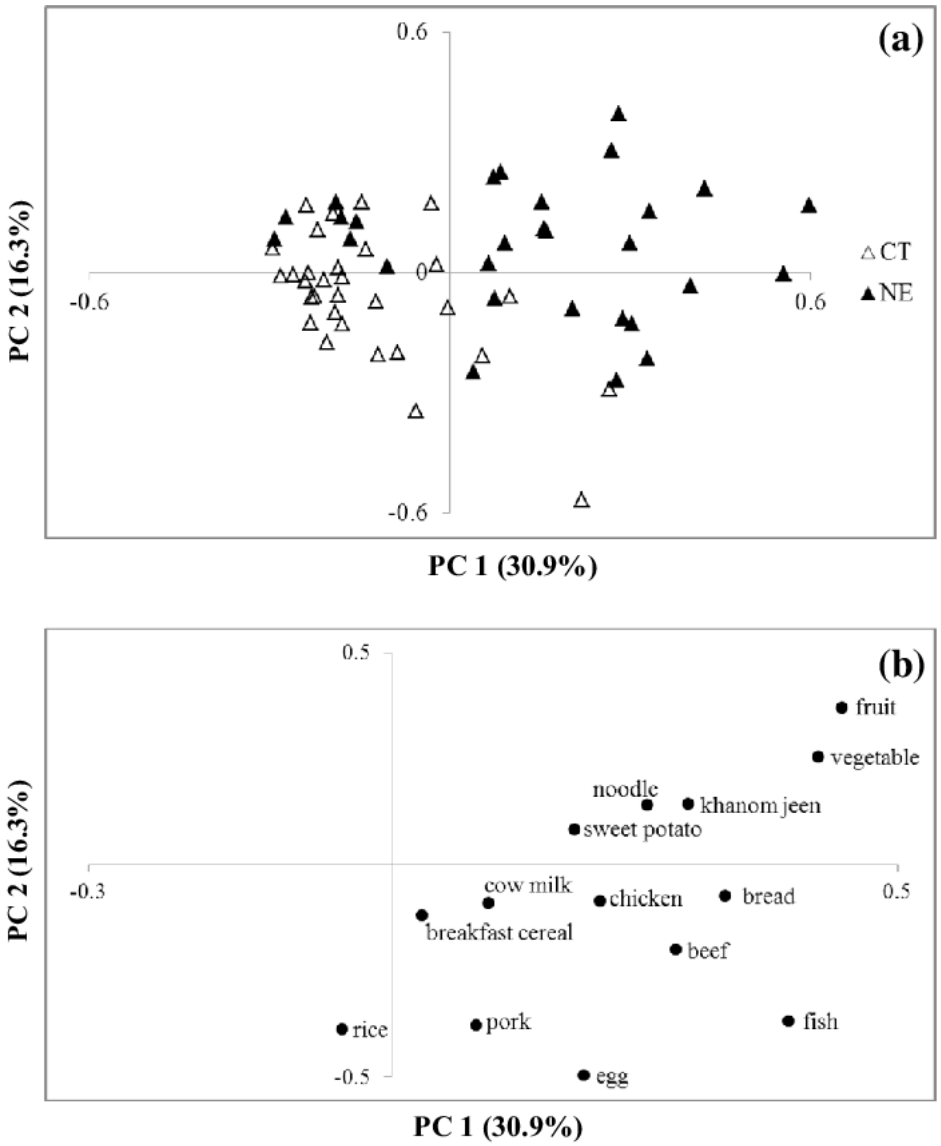
Figur 2.3: Fra variabler til prinsipal-komponenter. Figur hentet fra (Raschka og Mirjalili, 2017) side 143.

For å se hvor mye av variansen i datasettet prinsipalkomponentene forklarer, er det ofte nyttig å se på forklart varians for de første komponentene, slik som i figur 2.4. Der vises både kumulativt og individuelt forklart varians for hver komponent. Komponent-nummer ligger langs x-aksen mens andel forklart varians mellom 0 og 1 ligger langs y-aksen. Jo høyere andel varians de første komponentene forklarer (jo brattere kumulativ kurve), jo bedre modell kan vi si at PCAen er på det respektive datasettet. Om det kreves mange prinsipalkomponenter (slak kumulativ kurve) for å få forklart noen særlig del av variansen, vil man få mindre utbytte av PCA-modellen. Dette er fordi et mål med en PCA kan beskrives som å få mest mulig informasjon fra datasettet forklart i færrest mulig variabler (komponenter).



Figur 2.4: Kumulativt og individuelt forklart varians. Figur hentet fra (Raschka og Mirjalili, 2017) side 148.

Videre er det vanlig å sammenligne scorene og ladningene i egne plott fra en PCA-modell. Scorene er de ulike tilfellene (samplene) i det opprinnelige datasettet, projisert inn i vektorrommet som prinsipalkomponentene utgjør. Ladningene er de opprinnelige variablene projisert inn i det nye rommet. I figur 2.5 (La-ongkham mfl., 2015) vises scorene og ladningene i hvert sitt plott. Dette er resultatet av en PCA på et datasett om matvanene til friske barn i 2 forskjellige regioner i Thailand. I plott a), score-plottet, viser ufargede trekanter barn i sentral-Thailand, mens fylte trekanter viser barn i Nord-østlige Thailand. 1. og 2.-aksen er første og andre prinsipalkomponent, og trekantene viser datasettets tilfellers plassering i forhold til disse. I plott b), ladningsplottet, vises de ulike variablenes plassering i forhold til første og andre prinsipalkomponent.



Figur 2.5: PCA scoreplott (a) og ladningsplott (b). (sst.)

Ved å sammenligne plott a) og b) kan det for eksempel tyde på at det er en trend at barn fra det Nord-østlige Thailand spiser mer frukt og grønnsaker enn barn fra det sentrale Thailand. Man kan også slutte at for eksempel variabelen ”brød” blir mer forklart av første prinsipal-komponent enn den andre, siden den har en høyere verdi i forhold til denne (ligger lenger fra første prinsipalkomponents nullpunkt enn andre prinsipalkomponents nullpunkt). Ellers kan man generelt tolke score- og ladningsplott etter følgende prinsipper. Scorer (tilfeller) som er nær hverandre har lignende egenskaper. Ladninger (variabler) som er nær hverandre har høy korrelasjon, mens ladninger på motsatt side av origo for hverandre har negativ korrelasjon. Scorer øverst til høyre domineres av ladninger som er øverst til høyre i tilhørende ladningsplott (Mevik, 2007).

2.3 Om maskinlæring

Maskinlæring handler i denne oppgaven om å bruke maskinlæringsalgoritmer for å fortelle oss hvor og når det eventuelt blir solslyng. En slik algoritme trener først på kjente data, den blir altså matet med data og får vite når og hvor det var solslyng. Videre skal algoritmen prøve å predikere på ukjente data. Dette vil si at vi mater den med data uten å fortelle den når og hvor det var solslyng. Så skal den fortelle oss hvor og når den tror det var solslyng. Treffsikkerheten til maskinlæringsalgoritmene måles ved å sammenligne det maskinlæringsalgoritmen forteller oss om hvor og når den tror det var solslyng, med ”fasiten”, hvor og når det faktisk var solslyng.

Stegene i en typisk maskinlæringsprosess:

1. Skaffe data og kunnskap om dataene
2. Konstruere datasett tilpasset maskinlæring
3. Kjøre variabel-seleksjoner og variabel-mekking (feature engineering)
4. Dele opp datasette i trenings-, validerings- og testsett
5. Teste ulike modeller og stille inn parametere ved å trene algoritmen på treningssettet og kontinuerlig sjekke treffsikkerhet på valideringssettet
6. Kjøre den eller de beste modellene på testsettet
7. Repetere steg 3-6 mange ganger
8. Kjøre beste modell på ny data

Maskinlæringsproblemstillinger kan deles opp i tre ulike kategorier etter hva de er ment å predikere. Dette er kontinuerlig output, dvs regresjon hvor outputen er kontinuerlige verdier, binær klassifisering og multi-klasse klassifisering. I denne oppgaven er det valgt å bruke binær klassifisering, det vil si predikere om det blir solslyng på et tilfelle eller ikke.

Et av de aspektene det er viktigst å være klar over i arbeid med maskinlæring er overtilpasning (overfit) og undertilpasning (underfit). Overtilpasning vil si at algoritmen har lært seg så mange av strukturene i treningssettet (den delen av datasettet som den trener på), at den predikerer vesentlig dårligere på testsettet (den resterende delen av datasettet som den ikke har trent på) enn på treningssettet. Da har den rett og slett lært seg strukturer som kun gjelder for treningssettet, men ikke gjelder generelt også for testsettet. For å bruke maskinlæring i praksis er det viktig å lage robuste modeller som har tilegnet seg

mest mulig kunnskap som er felles for alle data den skal komme til å predikere på, og minst mulig kunnskap som kun gjelder for dataene den har trent på.

Undertilpasning vi si at maskinlæringsalgoritmen enda ikke har lært alle ”universelle” strukturer i treningssettet, og predikerer dårligere både på treningssettet og testsettet enn det er potensiale til.

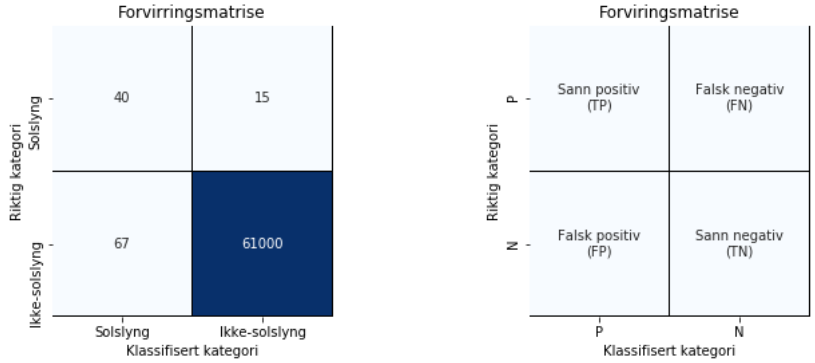
I de følgende avsnittene blir maskinlæringsalgoritmene brukt i masteroppgaven gjennomgått. Det er en veldig praktisk presentasjon av algoritmene, med fokus på intuisjon og anvendelser av dem. Det er lagt spesielt vekt på å beskrive aspekter av dem som har vært nyttige i arbeidet med oppgaven.

2.3.1 Prestasjonsmål (Performance metrics)

F1-score har blitt valgt for å måle maskinlæringsmodellenes prestasjon, altså hvor godt de klarer å predikere solslyngtilfellene og ikke-solslyngtilfellene i sine riktige klasse (solslyng/ikke-solslyng). Grunnen til dette redegjøres for i de følgende avsnittene.

Det vanligste prestasjonsmålet brukt til klassifisering i maskinlæring er nøyaktighet (accuracy), det vil si andelen riktig klassifiserte tilfeller på en skala fra 0 til 1. Da maskinlæringen i denne oppgaven bygger på et ekstremt ubalansert datasett, hvor forholdet mellom antall solslyng-tilfeller og ikke-solslyng-tilfeller er 55:61067, vil det gi mindre mening å bruke nøyaktighet som prestasjonsmål. Grunnen til dette er hvis maskinlæringsmodellen klassifiserer alle tilfellene som ikke-solslyngtilfeller, vil den få en nøyaktighet på $\frac{61067}{61122} \approx 0.9991$, noe som vanligvis ville vært en vanligvis bra score. I dette tilfellet, derimot, ville en så høy score ikke bety mer enn tilfeldig gjetning.

Ved veldig ubalanserte datasett er det da nyttig å se på andre måter å måle prestasjonen på. For å etablere forståelsen rundt hvordan prestasjonen i ubalanserte datasett kan måles, starter vi med å se på en typisk forvirringsmatrise. En forvirringsmatrise viser antall riktig og feilklassifiserte tilfeller etter hvilken klasse de tilhører, se figur 2.6.



Figur 2.6: Forvirringsmatrise. Figur til venstre viser et eksempel på et utfall av en maskinlæringsprediksjon. Laget med Seaborn (Waskom mfl., 2018)

Ut fra dette regnes nøyaktighet ut slik:

$$ACC = \frac{TP+TN}{FP+FN+TP+TN}$$

der ACC står for nøyaktighet og de resterende størrelsene jfr. figur 2.6.

I dette tilfellet ville nøyaktigheten blitt

$$ACC = \frac{40+61000}{15+67+61000+40} \approx 0.9987$$

Sett fra et generelt maskinlæringsperspektiv er dette en veldig høy score, men Bane NOR hadde fått 107 solslyngvarsler hvor kun $\frac{40}{40+67}100 \approx 37.38\%$ av varslene faktisk var solslyng.

Et annet og mye brukt prestasjonsmål for ubalansert klassifisering er F1-score, som også er valgt som prestasjonsmål i denne oppgaven. F1-scoren går fra 0 til 1, hvor 1 er perfekt klassifisering, og blir regnet ut på følgende måte.

Først beregnes størrelsene precision (PRE) og recall (REC)

$$PRE = \frac{TP}{TP+FP}$$

$$REC = \frac{TP}{FN+TP}$$

F1 score er en kombinasjon av precision og recall:

$$F1 = 2\frac{PRE \times REC}{PRE+REC}$$

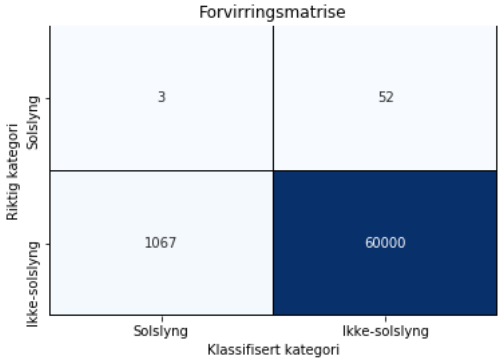
som i vårt tilfelle ville blitt

$$PRE = \frac{TP}{TP+FP} = \frac{40}{40+67} \approx 0.3738$$

$$REC = \frac{TP}{TP+FN} = \frac{40}{40+15} \approx 0.7273$$

$$F1 = 2\frac{PRE \times REC}{PRE+REC} = 2\frac{0.3738 \times 0.9989}{0.3738+0.7273} \approx 0.4938.$$

Altså en F1-score på ca 0.49.



Figur 2.7: Forvirringsmatrise. Laget med Seaborn (sst.)

Ved en dårligere klassifisering, slik som i figur 2.7, ville nøyaktigheten og F1-scoren blitt slik:

$$ACC = \frac{TP+TN}{FP+FN+TP+TN} = \frac{61000+3}{52+1067+61000+3} \approx 0.9817$$

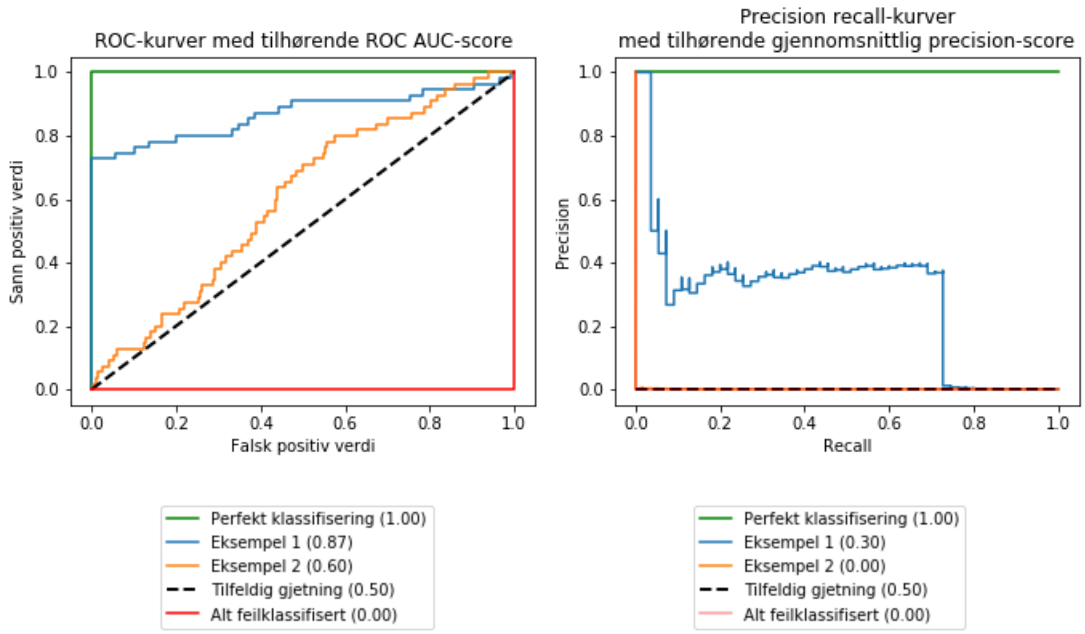
$$PRE = \frac{TP}{TP+FP} = \frac{3}{3+1067} \approx 0.0028$$

$$REC = \frac{TP}{TP+FN} = \frac{3}{3+52} \approx 0.0545$$

$$F1 = 2\frac{PRE \times REC}{PRE+REC} = 2\frac{0.0028 \times 0.0545}{0.0028+0.0545} \approx 0.0053$$

Selv med en nøyaktighets-score på 0.9817, ville Bane NOR ha fått 1070 varsler om solslyng, hvor kun $\frac{3}{1067} \times 100 \approx 0.28\%$ av dem faktisk var solslyng. Da er F1-scoren som gir 0.0053 mer fornuftig å bruke siden den gir et mer realistisk bilde av situasjonen.

En annen mye brukt metode for å måle prestasjon er receiver operating curve (videre ROC-kurve) med tilhørende prestasjonsmål ROC areal under kurven (videre ROC AUC-score). Precision recall-kurver og gjennomsnittlig precision-score er også mye brukt på ubalansert data. ROC-kurver og Precision-recall-kurver med tilhørende scorer for de to foregående eksemplene vises i figur 2.8. Tallene i parentes er arealet under kurvene, ROC AUC-scoren.



Figur 2.8: ROC-kurver for eksemplene i figur 2.3.1 (Eksempel 1) og 2.3.1 (Eksempel 2) med tilhørende ROC AUC-score i parentes. NB: Feil i tegnforklaringen til Precision recall-kurver. Tilfeldig gjetning skal være på 0.0009 og ikke på 0.50. Scoren på Eksempel 2 er tilnærmet lik 0. Laget med Matplotlib (Hunter, 2007)

ROC-kurven viser forholdet mellom falsk positiv verdi (videre FPR) og sann positiv verdi (videre TPR). FPR vil si sannsynligheten for at ikke-solslyngtilfellene blir klassifisert som solslyng, mens TPR vil si sannsynligheten for at solslyngtilfellene blir klassifisert som solslyng. FPR og TPR er definert slik:

$$FPR = \frac{FP}{FP+TN}$$

$$TPR = \frac{TP}{FN+TP}$$

F1-scoren og ROC AUC-scoren regnes ut på forskjellige måter og vil dermed gi litt forskjellige resultater etter hvordan forvirringsmatrisen blir seende ut. I vårt tilfelle kan dette oppsummeres i tabell 2.1. Eksempel 1 har eksempelvis en F1-score på 0.49, noe som er omtrent midt mellom laveste og høyeste score på henholdsvis 0 og 1. Samtidig har eksempel 1 en ROC AUC-score på 0.86, som er langt over middels, mellom laveste og høyeste score på henholdsvis 0.5 og 1 (evt. 0 og 1), selv om kun 37.38 % av tilfellene som ble klassifisert som solslyng faktisk var solslyng. Nøyaktighet viser seg å kunne være direkte misvisende på oppgavens datasett, da tilfeldig gjetning får høyere nøyaktighet enn det begge eksemplene får. 4 desimaler er brukt på nøyaktighet kun for å kunne skille verdiene fra hverandre.

Mange maskinlæringsalgoritmer er implementert med støtte for å få ut prediksjonene som

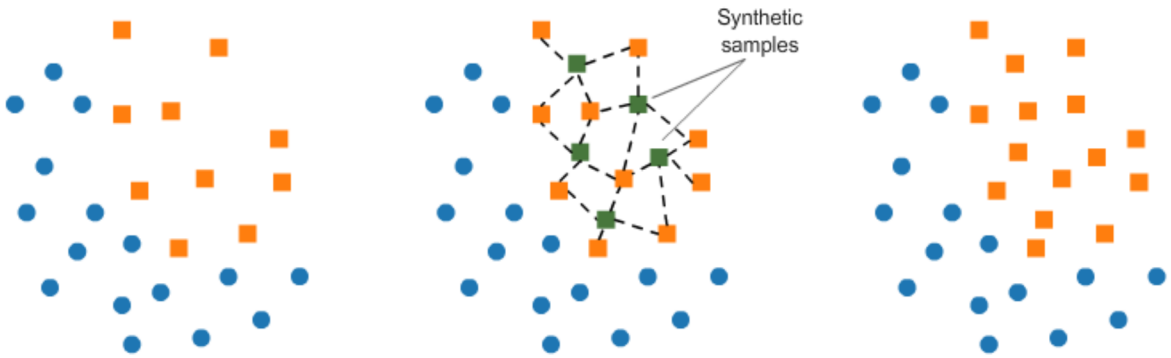
	F1-score	ROC AUC-score	Nøyaktighet (accuracy)
Perfekt klassifisering	1.00	1.00	1.0000
Eksempel 1	0.49	0.86	0.9987
Eksempel 2	0.01	0.60	0.9817
Tilfeldig gjetning	0.00	0.50	0.9991
Alt feilklassifisert	0.00	0.00	0.0000

Tabell 2.1: Sammenligning av F1- og ROC AUC-scorer og nøyaktighet (accuracy).

sannsynligheter for hvilken klasse hvert tilfelle hører hjemme i. For eksempel kan følgende vektor være et resultat fra en maskinlæringsmodell, med sannsynlighet for at tilfellene er solslyng: [0.9, 0.1, 0.4, 0.7, 0.3]. Ved å sette en terskel-verdi kan justere på hvor sannsynlig det må være for at et tilfelle er solslyng før det faktisk blir klassifisert som solslyng. En terskel-verdi på 0.6 her ville klassifisert to av tilfellene som solslyng, mens en terskel-verdi på 0.2 ville klassifisert alle unntatt ett tilfelle som solslyng. På denne måten kan man justere om det viktigste er å finne alle solslyngtilfellene selv om man risikerer "falsk alarm" enkelte ganger, eller om det er viktigere at tilfellene som blir klassifisert som solslyng, faktisk er solslyng, selv om man risikerer å ikke få med alle solslyngtilfellene.

2.3.2 Resampling

Det finnes ulike metoder for å håndtere ubalanserte datasett, som blant annet resampling. Resampling vil si at man enten upsampler, altså genererer flere fra minoritetsklassen (den klassen med færrest tilfeller), eller at downsampler, altså tar ut tilfeller fra majoritetsklassen (klassen med flest tilfeller), eller en kombinasjon av begge deler. Det finne ulike algoritmer og teknikker for å resample. For avgrensning av oppgaven ble det valgt å bruke én resamplings-teknikk som heter SMOTE (Synthetic Minority Over-sampling Technique) fra Python-biblioteket imbalanced-learn (Lemaître, Nogueira og Aridas, 2017). SMOTE går ut på å upsample minoritetsklassen, ved å generere nye tilfeller i minoritetsklassen basert på de allerede eksisterende tilfellene. Dette gjøres ved å tilfeldig velge et punkt fra minoritets-klassen, finne et antall nærmeste naboer innenfor samme klasse, og generere nye punkter mellom disse "naboene". Dette illustreres i figur 2.9.



Figur 2.9: Illustrasjon av hvordan resamplings-metoden SMOTE fungerer. Figur hentet fra (Alencar, 2018).

2.3.3 Random Forest

Random Forest er en ensemble-klassifikasjonsmetode basert på et ”ensemble” av beslutningstrær (decision trees). Et beslutningstre er et binærtre med den hensikt å skille ulike tilfeller fra hverandre basert på tilfellenes egenskaper. For eksempel, om solslyng kan forekomme ved temperaturer over 26 °C, kan rotnodens for eksempel to barn-noder skille tilfellene over og under 26 °C. Om solslyng forekommer et sted med antall soltimer i uka høyere enn 60, kan barnebarn-nodene inneholde tilfellene over og under 60 soltimer. Og slik fortsetter det helt til det kun er ett tilfelle i hver løvnode, eller til en eventuelt satt maksgrense for dybde på treet er nådd.

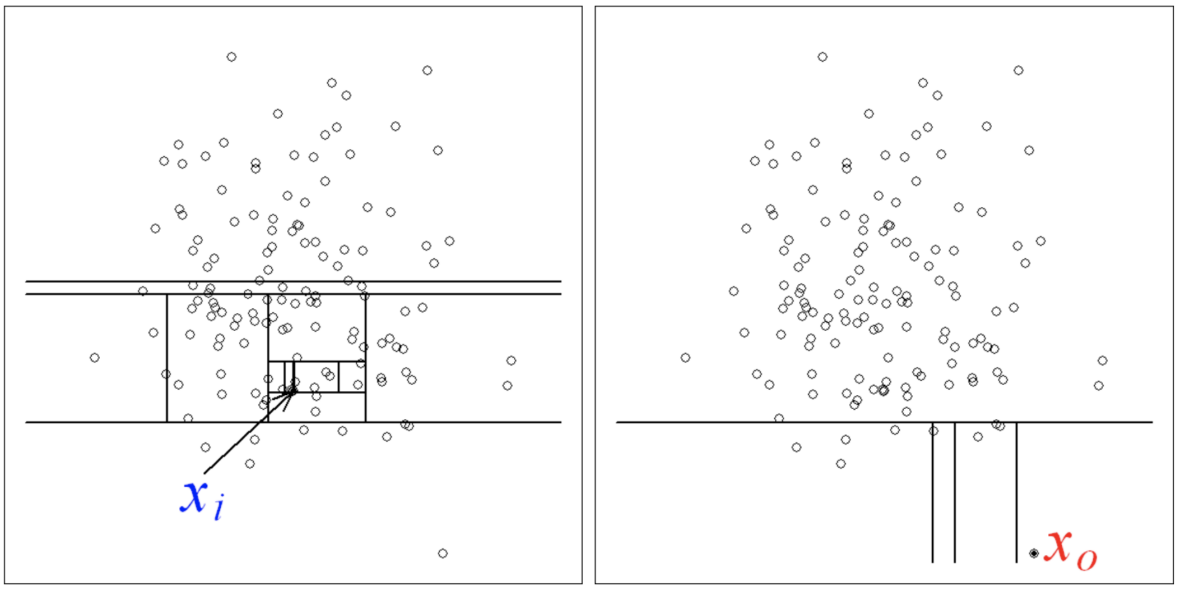
Beslutningene i hver splitt til hvert tre i Random Forest kan blant annet baseres på Gini impurity eller information gain. Gini impurity kan kort fortalt bli forstått som å sannsynligheten for å klassifisere feil, mens information gain måler entropi i nodene og prøver å få entropien i hver node lavest mulig (Raschka og Mirjalili, 2017). Lav entropi vil i denne sammenhengen si at det er flest mulig tilfeller av samme klasse i hver node.

Random Forest fungerer ved at hvert beslutningstre kommer fram til sin ”konklusjon” til hvordan å dele opp datasettet best mulig for å skille solslyng- og ikke-solslyngtilfellene. Så blir det utført en ”avstemning”, de beslutningstrærne og deres avgjørelser for forekommer flest ganger blir gjeldene.

Ved trening av en Random Forest-modell er det ofte hensiktsmessig å justere og prøve ut ulikt antall beslutningstrær og ulik maks dybde på trærne for å få så gode klassifiseringer som mulig.

2.3.4 Isolation Forest

Isolation Forest er en unsupervised maskinlæringsalgoritme ment til å finne uteliggere og anomalier i datasett. På samme måte som Random Forest er Isolation Forest bygget opp som en ensemble-algoritme av binær-trær. Den fungerer ved å velge ett tilfeldig tilfelle og videre partisjonere datasettet med tilfeldige oppdelings-verdier. Uteliggere som ligger mer alene i forhold til andre tilfeller, får da en partisjon for seg selv mye tidligere (nærmere rotnoden), enn et "vanlig" tilfelle som har mange andre tilfeller rett i nærheten (Lewinson, 2018). Dette er illustrert i figur 2.10.



Figur 2.10: Illustrasjon av virkemåten til Isolation Forest. Uteliggeren, x_0, får en partisjon for seg selv mye tidligere enn et "vanligere" tilfelle, x_i. Figur hentet fra (Lewinson, 2018).

Datasettet som er laget og brukt i denne oppgaven er ekstremt ubalansert med 55 solslyng-tilfeller mot 61067 ikke-solslyng-tilfeller. Det har derfor vært vurdert å snu problemstillingen på hodet: I stedet for å klassifisere solslyng-tilfeller fra ikke-solslyng-tilfeller, kan solslyngtilfellene sees på som anomalier. Dette er vanlig blant annet i maskinlæring for naturkatastrofer (som jo hender bare noen få ganger i året kanskje) eller for å detektere hackere fra vanlige internett-brukere (ved deres uvanlige nett-aktivitet). Ut fra dette er det naturlig å tenke seg at dette er en relevant metode å prøve ut også for denne oppgavens datasett.

2.3.5 Logistisk regresjon

I motsetning til Random Forest og Isolation Forest, handler det for mange maskinlæringsalgoritmer om å separere et datasett i de ønskede klassene ved hjelp av lineære funksjoner. Hvis dette lar seg gjøre på

et datasett, sier vi at datasettet er lineært separabelt. De fleste datasett har flere enn to variabler som brukes for å klassifisere tilfellene, slik at den lineære funksjonen i realiteten blir et hyperplan. Dette hyperplanet kalles også for beslutningsgrensen (decision boundary). Siden noen variabler vil tilføre mer nyttig informasjon til klassifiseringen enn andre, blir hver variabel tillagt en vekt som bestemmer hvor mye variabelen skal ha å si for valg av beslutningsgrense i forhold til de andre variablene.

Logistisk regresjon fungerer på denne måten og presterer veldig bra på lineært separable data (Raschka og Mirjalili, 2017). Logistisk regresjon er basert på oddsen for hvert enkelt tilfelles klassetilhørighet. Uten å gå altfor dypt inn i matematikken, presenteres det her en overfladisk beskrivelse av hvordan logistisk regresjon fungerer.

Variablene blir matet inn i en logistisk sigmoid-funksjon ($\phi(z)$) som konvergerer mot enten 0 eller 1 avhengig av størrelsen på verdien, z. Se 2.3.5.

$$\phi(z) = \frac{1}{1+e^{-1}}$$

Sigmoid-funksjonen er aktiveringsfunksjonen (activation function) til logistisk regresjon. Aktiveringsfunksjonen ”mapper” hvert tilfellet til en av klassene og de fleste maskinlæringsalgoritmer har en aktiveringsfunksjon.

Fordi sigmoid-funksjonen konvergerer mot enten 0 eller 1 fungerer algoritmen kun for binær klassifisering. Er det flere en to klasser, kjøres algoritmen flere ganger og predikerer en klasse av gangen mot resten av klassene.

Ut fra dette blir det regnet ut en kostfunksjon basert på oddsen for at hvert tilfelle tilhører hvilken klasse. kostfunksjonen er en funksjon av vektene og avstanden mellom sann klasseverdi (0 eller 1) multiplisert med logaritmen til aktiveringsfunksjonen. Jo større avstand til sann klasseverdi, jo høyere blir kost-verdien. Gradienten til kostfunksjonen brukes for å finne de vektene som gir lavest kost-verdi. Dermed blir vektene oppdatert til neste iterasjon algoritmen kjører. Dette prinsippet med gradient og kostfunksjon kalles gradient descent.

Regularisering

Logistisk har også funksjoner for regularisering. Regularisering er nyttig for å håndtere kolineariteter, filtrere ut støy (variabler som ikke tilfører nyttig informasjon i forhold til klassifiseringen) og dermed

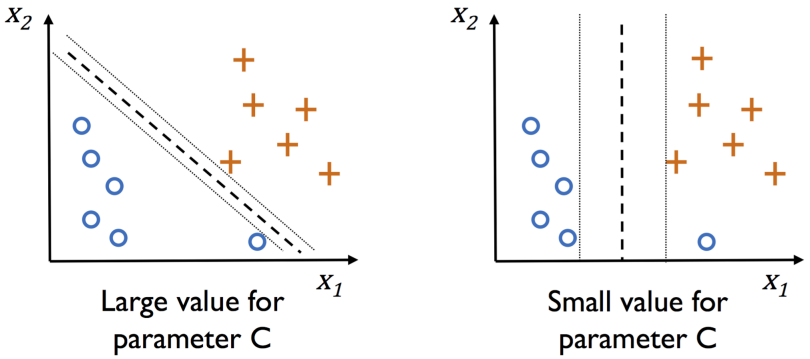
forhåpentligvis redusere risikoen for overtilpasning (sst.). Den vanligste formen for regularisering er såkalt L2-regularisering, hvor kvadratet av hver vekt blir multiplisert med en regulariseringsparameter. Dette blir så lagt til kostfunksjonen og graden av innvirkning som variablene har hver for seg på kostfunksjonen, blir regulert. En annen type regularisering er L1 regularisering, hvor kun absolutt-verdien av hver vekt blir multiplisert med regulariseringsparameteren. L1-regularisering vil dermed sette noen av variablene til 0, slik at de ikke blir med i klassifiseringen. Dette er spesielt nyttig på datasett med mange variabler, og L1-regularisering kan da fungere som en form for variabel-seleksjon.

Regulariseringsparameteren er justerbar i scikit-learn, ved at man kan endre på parameteren C som er lik den inverse av regulariseringsparameteren (λ). På denne måten kan man justere hvor mye regularisering som skal foregå.

$$C = \frac{1}{\lambda}$$

2.3.6 Support Vector Machines (SVM)

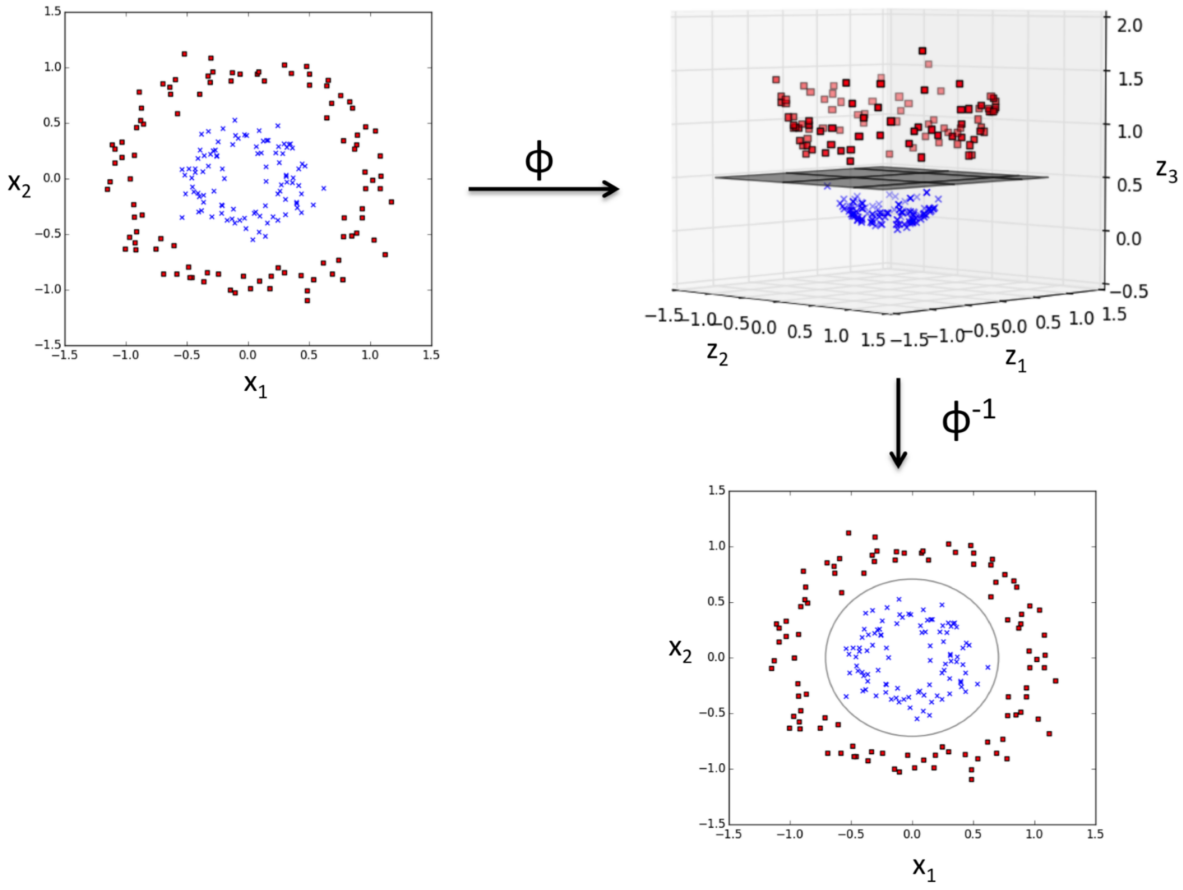
Det som er spesielt med support vektor machines (videre SVM) er at at de søker å maksimere bredden av beslutningsgrensen mellom to klasser. Dette er med på å avgjøre hvilket hyperplan som velges for klassifisering. Størrelsen på bredden justeres ved å justere på SVMs penalty parameter C. Jo større C vi velger, jo smalere blir bredden, og jo "strengere" blir algoritmen på å ikke feilklassifisere noen tilfeller (sst.). Jo mindre C, jo større bredde slik at algoritmen blir mer generaliserende og mer robust mot overtilpasning. Figur 2.11 illustrerer dette.



Figur 2.11: Illustrasjon av hvordan justering av parameteren C innvirker på valg av hyperplan i klassifiseringen. Figur er hentet fra (Raschka og Mirjalili, 2017) side 79.

Noen SVMer har støtte for kjerner. Radiell basis-funksjon er et eksempel på en slik kjerne. Den kan

projisere ikke-lineært separable data inn til høyere dimensjoner, slik at det blir lineært separabelt i det nye hyper-rommet.



Figur 2.12: Illustrasjon av hvordan radial basis funksjon fungerer. Et ikke lineært separabelt datasett blir projisert slik at det får en dimensjon til og blir da lineært separabel i den nye dimensjonen. Deretter blir datasettet projisert tilbake til det opprinnelige variabel-rommet (feature space), og beslutningsgrensen blir tatt vare på. Figur hentet fra (Raschka og Mirjalili, 2017) side 84.

2.4 Dyp læring (Deep learning)

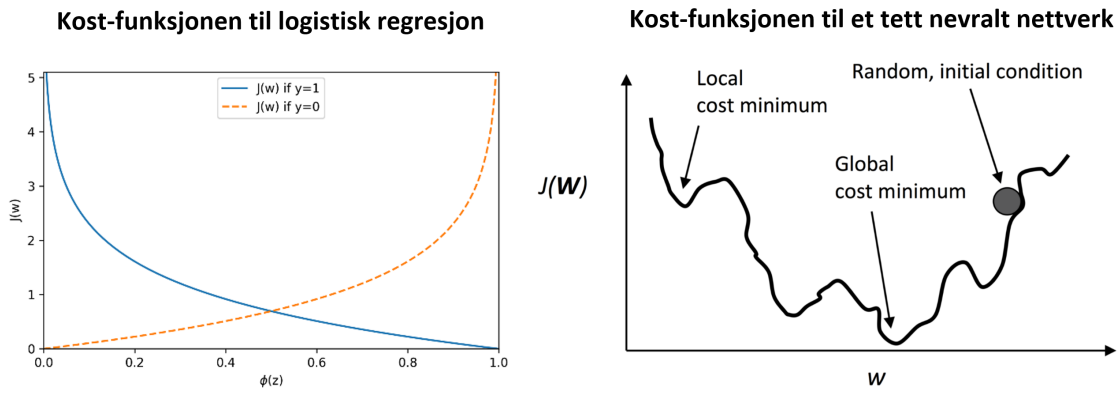
2.4.1 Multi-Layer Neural Networks

Multi-Layer Neural Networks (videre MLNN) er et eksempel på en maskinlæringsalgoritme innen dyp læring (deep learning) og kan sammenlignes med logistisk regresjon ved at vekter også her oppdateres basert på en aktiveringsfunksjon og en kostfunksjon. Forskjellen på MLNN er at dette kan foregå flere ganger i flere ulike lag, hvor det siste laget er et output-lag hvor prediksjonene kommer ut. Utformingen av disse lagene og antall lag kan velges og tilpasses nesten i det uendelige, og stort sett er det prøving å feiling som skal til for å finne den optimale modellen for hvert datasett. For hvert lag må det blant annet

velges antall noder, bortsett fra det første laget (input-laget) som må ha samme antall noder som antallet tilfeller i datasettet, og bortsett fra det siste laget (output-laget) som må ha samme antall noder som antallet klasser som skal predikeres.

For hvert lag må det også velges en aktiveringsfunksjon. Sigmoid-funksjonen brukes ofte i output-laget for binær klassifisering, mens på de andre skjulte lagene kan man velge andre funksjoner. En mye brukt aktiveringsfunksjon er Rectifier Linear Unit som kan skriver som $\phi(z) = max(0, z)$.

Det må også velges en optimizer, altså hvilken algoritme som skal få æren av å prøve å finne det globale minimumet av kostfunksjonen. Optimizeren har dessuten gjerne flere parametere som kan tilpasses, blant annet læringsrate. Læringsraten vil si hvor store steg optimizeren skal ta om gangen i jakten på det globale minimum. Siden kostfunksjonen gjerne er mye mer komplisert enn ved for eksempel logistisk regresjon kan det være nødvendig å gjøre justeringer på læringsraten slik at optimizeren for eksempel ikke mistolker lokale minimums-plasseringer for å være globale. Forskjellen på den logistiske kostfunksjonen og en kostfunksjon for tette nevrale nettverk vises i figur 2.13.



Figur 2.13: Ulike kostfunksjoner. Til venstre vises kostfunksjonen til logistisk regresjon, der $J(w)$ er kostfunksjonen, w er vektene, $\phi(z)$ er aktiverings-funksjonen, z er variabel-verdi og y er sann klasseverdi. Til høyre vises en illustrasjon av en potensiell kostfunksjon for et tett nevralt nettverk, "Random, initial condition" er tilfeldig start-verdi for en vekt. Figurene er hentet fra (Raschka og Mirjalili, 2017) side 65 og 417.

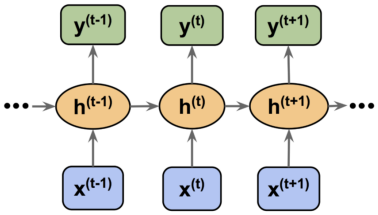
På grunn av de store tilpasningsmulighetene og muligheter for å lære veldig komplekse mønstre, presterer ofte dype nevrale nettverk bedre enn enklere maskinlæringsalgoritmer som de som er beskrevet i de foregående avsnittene.

MLNN har også støtte for at man kan putte inn såkalte dropout-lag. Det er lag som ekskluderer en tilfeldig andel av datasettet fra å bli trent på. Dette bremser prosessen med at maskinlæringsmodellen

lærer seg for mange strukturer i treningssettet som ikke er universelle, fordi den hele tiden må trene på litt forskjellige datasett, da en del av det hele tiden er tatt ut. Dette kan dermed bidra til å få mer robuste modeller og forhåpentligvis redusere overtilpasning.

2.4.2 Recurrent Neural Network (RNN)

Recurrent Neural Network (videre RNN) er bygd opp på omtrent samme prinsipp som MLNN, bortsett fra at de også kan ta sekvensielle strukturer i datasettet med i betraktning. Det vil si at den kan lære fra tidsserier eller annen sekvensiell data som ord og tekst, hvor rekkefølgen i datasettet spiller en rolle. I MLNN flyter informasjonen fra input-laget, via de skjulte lagene og til out-putlaget. I RNN går ikke informasjonen kun fra input-laget, men også fra de skjulte lagene for det forrige tidspunktet (om det er snakk om en tidsserie) og deretter til output-laget. Dette illustreres i figur 2.14, som viser en RNN med ett skjult lag. x er variablene for et tilfelle, h er det skjulte laget og y er output-laget. Her går informasjonsflyten både fra input-lagene og fra det skjulte laget fra det forrige "steget".



Figur 2.14: Illustrasjon av lagene i RNN. Figuren er et utklipp fra (Raschka og Mirjalili, 2017) side 542.

I RNN er det et problem at gradientene til kostfunksjonen kan bli mindre og mindre og til slutt bli ubetydelig på grunn av at den også tar output fra det forrige steget som input. Dette er det vanlig å løse med å bruke et LSTM-lag eller et GRU-lag. Disse lagene hjelper til med å "spare" på informasjonen slik at gradienten ikke "forsvinner" (the vanishing gradient problem).

I RNN kan man også bruke dropout-lag, men da er det vanligere å bruke en spesial-versjon av dropout kalt recurrent dropout. Bruker man vanlig dropout i rnn kan noe av informasjonen som har med det sekvensielle i datasettet å gjøre gå tapt, siden tilfeldige tilfeller blir tatt ut. Recurrent dropout sørger får at tilfeller blir tatt ut, uten at informasjonen i det sekvensielle går tapt.

Kapittel 3

Metode - Verktøy

3.1 Datainnsamling

Verktøyene brukt til datainnsamling av informasjon om jernbanen har vært Microsoft Excel, og Python 3.6 med Jupyter Notebook (Kluyver mfl., 2018) blitt brukt. For å uthenting og bearbeiding av geografiske data har blitt utført med Python 3.6 og ArcGIS Pro 3.2 (ESRI, 2018a). Værdataene har blitt lastet ned via et skript i MATLAB (MathWorks, 2019) og videre bearbeidet i Python og Jupyter Notebook.

3.2 Eksplorativ analyse

Til å utføre prinsipalkomponentanalyse (PCA) i forbindelse med eksplorativ analyse er Python-biblioteket Hoggorm (Tomic, 2017), blitt brukt. Ellers har Matplotlib (Hunter, 2007) blitt brukt til visualisering.

3.3 Maskinlæring

Python i Jupyter Notebook har blitt brukt til å implementere de ulike maskinlæringsmodellene. Maskinlæringsmodellene er hentet fra maskinlærings-APIet Scikit-Learn (Pedregosa mfl., 2011).

For deep learning har APIet Keras (Chollet mfl., 2015) blitt brukt. Dette er et høynivås nevrale nettverk API skrevet i Python (sst.) og er i denne oppgaven kjørt med Tensorflow backend.

Maskinlæringsalgoritmen Recurrent Neural Network (RNN) fra Keras-APIet er kjørt i Google Colaboratory (Google, 2019).

Andre Python-biblioteker som har blitt brukt til maskinlæringen er Seaborn (Waskom mfl., 2018) og Imbalanced-Learn (Lemaître, Nogueira og Aridas, 2017).

Kapittel 4

Metode - Datainnsamling

4.1 Om datagrunnlaget

4.1.1 Valg av variabler

Valg av variabler til datasettet er gjort med tanke på hva som kan ha påvirkning på faktorer som fører til solslyng. Siden solslyng forekommer som et resultat av både høy temperatur og redusert sideforskyvningsmotstand i sporet, er variabler som har med dette å gjøre blitt valgt ut.

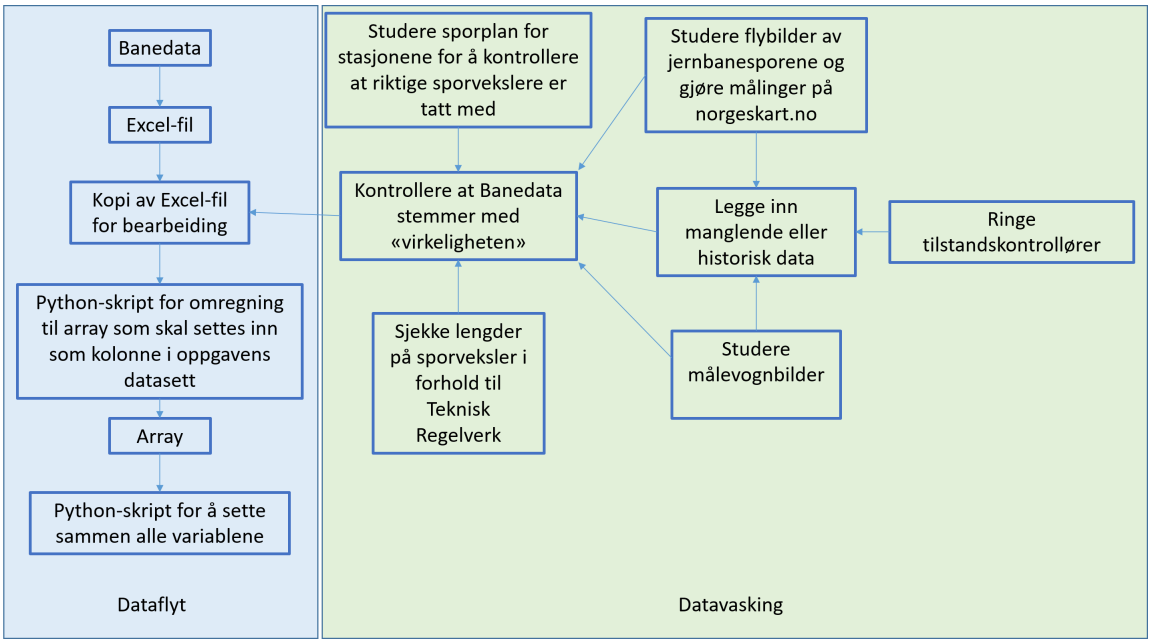
4.1.2 Jernbane-data

Banedata - innsyn (videre Banedata) er Bane NORs tjeneste for å hente ut data om jernbanen og har vært den viktigste kilden til data om jernbanen. Målevognbilder, bilder av jernbanen langs alle strekninger som taes to ganger i året har også blitt brukt for å kontrollsjekke opplysninger i Banedata, samt å skaffe ytteligere informasjon. Norgeskart.no samt Teknisk Regelverk har blitt brukt for å måle og finne lengder på sporveksler der dette ikke er oppgitt i Banedata. ”Networkstatement”, en tjeneste hos Bane NOR, samt Norgeskart.no har blitt brukt for å få oversikt over de forskjellige sporene og sporvekslene på stasjoner, slik at data om riktige spor og sporveksler har blitt tatt med i oppgaven.

En utfordring har vært inkonsistens i Banedata for data om jernbanen som ble hentet ut. Banedata har vært utsatt for endringer i de seneste årene, og skal være tilpasset mange forskjellige brukere på samme tid, både forskere, ingeniører og lokale vedlikeholdsarbeidere. Det har også vært ulik praksis for

hvordan data legges inn, slik at dataene som kommer ut stort sett ikke er standardisert. Dette gjorde at dataene som ble hentet fra Banedata måtte tilpasses og testes før det kunne brukes i maskinlæring. Dette innebar blant annet å gå gjennom hver meter av jernbanen og sjekke og korrigere for eventuelle strekninger hvor det manglet informasjon (huller) eller det var lagt inn ny data uten å fjerne eldre data slik at det ble overlapp. Historisk data var heller ikke tilgjengelig i Banedata. For noen tilfeller ble det derfor nødvendig å kontakte lokale banesjefer og tilstandskontrollører for å få ytterligere informasjon.

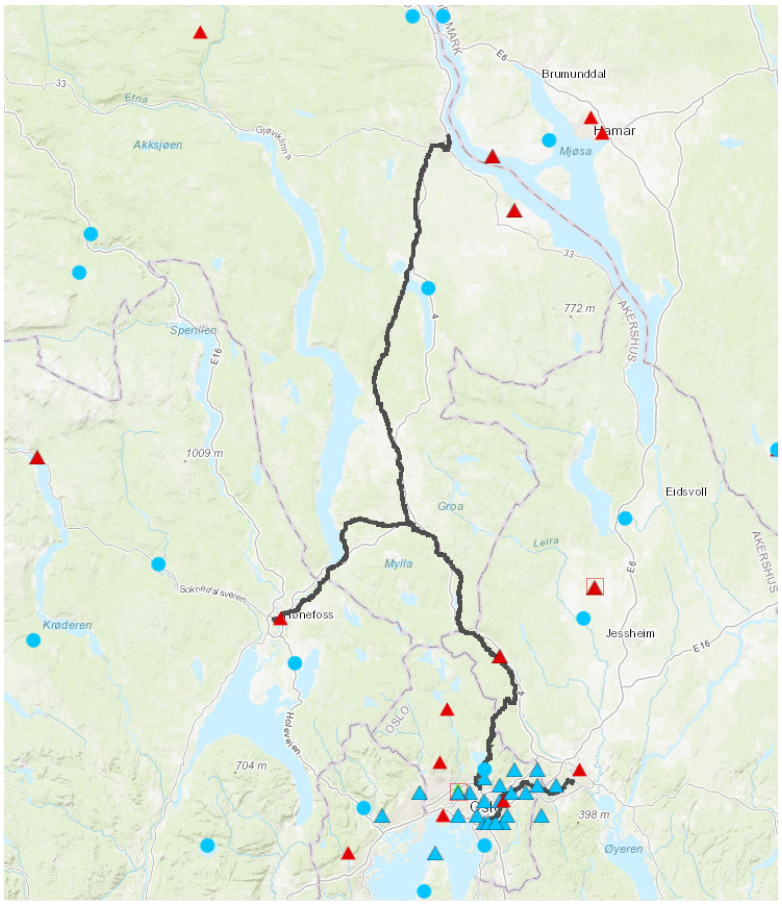
En illustrasjon av arbeidsflyten for å samle inn jernbane-data vises i figur 4.1.



Figur 4.1: Arbeidsflyt ved innsamling og vasking av jernbane-data.

4.1.3 Værdata

Når det gjelder værdata er det naturlig å henvende seg til Meteorologisk institutt sine gratis nedlastningstjenester. Krav til værdataene har vært at det må finnes data for alle deler av alle jernbanestrekningene som er innenfor oppgavens avgrensning både geografisk og i tid. Værdataene hos Meteorologisk institutt, samles inn på ulike værstasjoner rundt om i Norge. Det er derimot stor forskjell i tetthet mellom værstasjonene i forhold til den geografiske avgrensningen av denne oppgaven. Det er for eksempel mange værstasjoner langs Hovedbanen mellom Oslo S og Lillestrøm, men veldig få langs Gjøvikbanen og Roa-Hønefossbanen som vist i figur 4.2 hvor røde og blå symboler representerer plasseringene til ulike værstasjoner og svart linje jernbanestrekningene som er med i oppgaven.



Figur 4.2: Ulike værstasjoner (sirkler og trekanter) og jernbanestrekningene (svarte linjer). (Værstasjonene: Stefan Chapkanski, 2017. Jernbanen: Bane NOR SF, 2015. Bakgrunnskart: (ESRI, 2018a)

For å få gode værdata for alle deler av banestrekningene, må det derfor gjøres en interpolering mellom værstasjonene. Meteorologisk institutt har åpne tilgjengelige interpolerte data mellom værstasjonene som værvarsel, men ikke for historiske observasjoner. Siden maskinlæringsmodellen ideelt sett skal kunne forutsi solslyng i fremtiden, vil maskinlæringsmodellen måtte bli ”matet” med værvarsel-data. Derfor ble historisk, interpolert værsvarselsdata valgt ut til å representere datagrunnlaget for værdata i denne oppgaven. Disse dataene ble hentet ved hjelp av Meteorologisk institutts nedlastningstjeneste thredds.met.no.

Fra denne nedlastningstjenesten er dataene fra filene med navn ”meps_ mbr0_ pp_ 2_ 5km_ *.nc” brukt. * representerer dato og klokkeslett. Dette er post-prosessert værvarsel-data som blant annet brukes som basis for Yr.no. Dataene er predikert ved MetCoOp Ensemble Prediction System (MEPS), et ensemble-læringssystem med 10 ulike modeller (”members”) hvor en type gjennomsnitt av resultatene fra alle modellene utgjør de ferdig prosesserte dataene. Dataene er også interpolert ved ulike

interpolerings-algoritmer som Kriging og naboskap (Køltzow, 2017).

For perioden 12. mai til 28. mai 2018 var ikke disse dataene tilgjengelige, slik at filene med navn ”meps_mbr0_extracted_2_5km_ *.nc” ble brukt i stedet. Dette kan påvirke kvaliteten på værdatene noe, da disse dataene ikke er post-prosessert.

4.1.4 GIS-data

For generering av variabler til treningssettet ved hjelp av GIS-verktøy er det brukt følgende geografiske datasett.

Datasettet km-segmenter er laget ut fra filgeodatabasen ”Jernbane - Banenettverk - Lineære referanser” eid av Bane NOR. Denne inneholder linjedatasett for alle jernbanelinjer i Norge, samt punktdatasett for sporkilometere og tabeller for plasseringer til ulike objekter som tunneler, bruer og overganger. Det er et geografisk stedfestet topologisk nettverk av linjegeometri og noder (Bane NOR, 2018).

Km-segment-datasettet er laget ved å dele opp linjedatasettet av jernbanelinjer ved plasseringene til punktdatasettet med sporkilometere.* I tillegg er det gjort enkelte sammenslåinger av linjestykkene, slik at datasettet til slutt besto av 167 ulike linje-segmenter, ett segment for hver sporkilometer på jernbanen. Km-segmentenes attributter er generert ved en romlig kobling mot sporgeometri-settet.

Tunnel-datasettet er laget på følgende måte. Fra Bane NORs datasett ”Jernbane - Banenettverk - Lineære referanser” ble det opprettet et punktdatasett basert på tunnelers start- og slutt-kilometer på jernbanen. Dette punkt-datasettet ble brukt til å dele opp km-segment-datasettet. Det ble videre utført sletting av km-segmenter utenfor tunneler og sammenslåing av km-segmenter på strekninger med tunneler. Resultatet ble et linjedatasett, hvor hver linje-segment har geografisk plassering tilsvarene tunnelene på jernbanestrekningene.

Terrengmodellen som ble brukt til å generere enkelte av variablene er et raster-datasett lastet ned fra høydedata.no (Kartverket, 2019). Det er en digital overflatemodell (DOM) med oppløsning på 1 meter. For å redusere filstørrelsen på terrengmodellen, er en buffer på 2 km ut fra km-segment-datasettet brukt som klippe-polygon.

4.2 Om datasettet

4.2.1 Valg av datasettets utforming

I klassiske maskinlæringstilfeller har man ofte et antall forekomster av noe, for eksempel 200 hunder, som man vil plassere i en eller annen form for kategori, for eksempel hunderase. Da er det vanlig å samle så mye informasjon som mulig om alle hundene, samt informasjon om hvilken rase de er. Disse dataene organiseres så i en matrise, hvor hver rad representerer en hund og hver kolonne representerer en egenskap ved hunden. Den ene av kolonnene må inneholde ”fasiten”, nemlig hvilken hunderase hver hund er. Denne kolonnen kalles target, og er den som maskinlæringsalgoritmen skal prøve å predikere.

For maskinlæring på jernbanen ble problemstillingen litt annerledes, da jernbanen ikke er like klart definert i ulike tilfeller som for eksempel hundene. Det måtte da gjøres et valg for hvordan jernbanen skulle deles opp og hvordan hver jernbanedels egenskaper skulle representeres. Valget falt på å dele opp jernbanen i 1 km-segmenter etter Bane NORs sporkilometer-system. Jernbanen kan variere mye innenfor en kilometer, og det hadde sikkert blitt oppnådd mer enhetlige jernbane-segmenter hvis hvert segment var kortere. Likevel var det også et poeng å ha så lange segmenter som 1 km, for at datasettet ikke skulle bli enda mer ubalansert. Tilfellene med solslyng hadde blitt betydelig færre i forhold til ikke-solslyngtilfellene om km-segmentene hadde vært kortere.

Når det gjelder å beskrive hvert km-segments egenskaper, med en tall-verdi per variabel var det flere valg som måtte tas. Blant annet er det flere komponenter på jernbanen som ikke skifter når jernbanens sporkilometer skifter. Som regel er overgangen mellom ulike skinneprofiler for eksempel nesten aldri på grensen mellom 2 ulike sporkilometere. Det ble først drøftet å bare registrere den skinneprofilen som forekom mest på hvert km-segment, men dette tilførte ikke mye informasjon da det er et par skinneprofil-typer som dominerer mest på de aller fleste km-segmenter. Ofte er det heller de mindre vanlige og eldre skinneprofiler og overganger til disse som kan være relevante i forhold til solslyng. Derfor ble det i stedet besluttet å lage en kolonne for hver skinneprofil-type og for alle andre komponent-typer på jernbanen som er tilsvarende kontinuerlige. Hver kolonne inneholder da andelen av hvor mye akkurat denne forekommer på km-segmentet som en tall-verdi mellom 0 og 1. Det ble i tillegg talt opp antall overganger mellom de ulike typene for å prøve å beskrive jernbanen enda litt mer utførlig.

Disse og andre variabler laget ut fra data om jernbanen beskrives mer detaljert i seksjonen ”Variabler

fra Banedata".

Hvordan de geografiske dataene og værdataene er representert i datasettet beskrives i seksjonen "Variabler for vær og terreng".

4.2.2 Datasettets struktur

Datasettet som mates inn i maskinlæringsalgoritmene er en 2D matrise, hvor hver rad svarer til ett km-segment og hver kolonne til én variabel. I tillegg er det en kolonne for om det har oppstått solslyng på km-segmentet eller ikke.

Antall km-segmenter og antall solslyngtilfeller i treningssettet er som følger:

Bane	**Antall km-segmenter**	**Antall solslyng**
Gjøvikbanen	117	32
Hovedbanen	17	5
Roa-Hønefossbanen	33	18
Totalt	**167**	**55**

Datasettet har en tidsoppløsning på én dag, fra 1. april 2017 til 30. september 2017 og fra 1. april 2018 til 30. september 2018. Dette er 167 dager til sammen, slik at totalt antall rader i treningssettet er antall dager x antall km-segmenter, $366 \times 167 = 61122$.

Rekkefølgen på radene er første km-segment første dag, deretter andre km-segment første dag osv helt til siste km-segment første dag. Da kommer første km-segment igjen, men dag nummer to. Deretter andre km-segment for dag nummer to osv.

Antall rader med solslyng utgjør $55/61122 = 0.090\%$, som gjør at dette datasettet er veldig ubalansert.

Noen av variablene i datasettet varierer for de ulike dagene, som for eksempel temperatur og skydekke, mens andre variabler er konstante for hele perioden, slik som sporgeometri. De variablene som er konstante er derfor identiske for alle tidspunktene i treningssettet.

4.3 Variabler fra Banedata

4.3.1 Håndtering av kontinuerlige variabler

Med kontinuerlige variabler menes her variabler som finnes på hele og alle jernbanestrekninger. Dette gjelder for eksempel skinneprofil og om det er enkelt- eller dobbeltspor. Eksempler på variabler som ikke er kontinuerlige er tunnel og skinneskjøter, da disse kun finnes enkelte steder.

Skinneprofil, sviller og befestigelse er kontinuerlige variabler og har blitt behandlet spesielt med tanke på forekommende huller og overlapp i Banedata. Disse variablene kan være forskjellig på vanlige banestrekninger og på sporveksler. Av og til er sporveksler lagt inn med utstrekning, mens andre steder er kun km-punktet for stokkskinneskjøten lagt inn. Av og til har jernbanestrekningen på begge sider av en sporveksler blitt lagt inn med et «hull» der hvor sporvekselen ligger, andre ganger er jernbanestrekningen lagt inn også der sporvekselen befinner seg. Derfor er alle strekningene gått gjennom meter for meter og korrigert for overlapp, huller og mulige feil. Det er sjekket med målevognbilder, banekartet, målinger på Norgeskart.no samt informasjon om sporveksellengder på jernbanekompetanse.no og i Teknisk Regelverk for å understøtte korrigeringene.

Ved overlapp i Banedata har følgende rekkefølge blitt brukt for å bestemme hvilken data som skal brukes: sporveksler, nyeste data, nøyaktighet/antall desimaler notert. Der det er mulig å se, er målevognbildene og banekartet brukt for utvelgelse.

Alle korrigeringene er notert excel-filene i mappen ‘Framgangsmåte treningssett’.

Hull på under 100 m kan forekomme i treningssettet, men alle overlapp er korrigert for. På km-segmenter med hull er det gjort en ‘strekking’ av variablene for å jevne ut for hullet. For eksempel om det er 45 % Pandrol e/PR og 45 % Pandrol Fastclip på en strekning og 10 % hull, vil dette strekkes slik at det blir 50 % Pandrol og 50 % Fastclip på strekningen.

Et eksempel på hvordan dette har blitt utført vises i skript 9.1 ”Eksempel skinneprofil”.

4.3.2 Solslyng

Solslynghendelser er lagt inn på de dagene og de km-segmentene de oppsto på. Om flere solslyng oppsto på en strekning som går over mer enn ett km-segment, er solslyngen lagt inn på begge segmentene.

Banedata skiller mellom tilløp til solslyng (pilhøydefeil 0-25 mm), solslyng (pilhøydefeil på 25 mm eller mer) og baksefeil (generelt avvik i sporets plassering i horisontalplanet) som følge av høy temperatur. For å få flest mulige tilfeller av solsyng å trene på, er alle disse tilfellene med i solslyng-variabelen i treningssettet.

4.3.3 Skinneprofil

Ulike skinneprofiler har ulik form og tverrsnittsareal, som har påvirkning på stivheten i skinnene som igjen har påvirkning på hvor lett skinnene vil bøye seg som følge av høy temperatur. Kraft = Spenning x Areal, der kraften er konstant og spenningen endres proporsjonalt med arealet. Øker arealet, vil spenningen minke og motsatt. Derfor er de ulike skinneprofilene tatt med som variabler i treningssettet.

Likeledes vil et profilbytte, overgang mellom to ulike skinneprofiler, øke faren for solslyng fordi det kan bli en spenningstopp i punktet med ulike skinneprofiler (Løhren, 2019b).

I Banedata er skinneprofil oppgitt for høyre og venstre skinnestreng hver for seg. For Hovedbanen som består av dobbeltspor er derfor skinneprofil oppgitt fire ganger for alle deler av strekningen. Siden profilbytter på høyre og venstre skinnestreng ikke nødvendigvis ligger på samme sporkilometer er det gjort et gjennomsnitt av alle skinnestrengene for andel forekomst av hver profil. Det er også tatt et gjennomsnitt av antall profilbytter per spor om det er dobbeltspor.

4.3.4 Vedlikehold

Ofte har solslyng oppstått etter at det har vært utført vedlikeholdsarbeid på jernbanen. Grunner til dette kan være at skinnene kan ha flyttet på seg under vedlikeholdsarbeidet, om skinnenes plassering ikke har blir kontrollert før og etter at arbeidet ble utført. En annen grunn kan være at etter vedlikehold som ballastrensing og justering av sporet er sporet ofte ikke like stabilt som før vedlikeholdet. Dette skal det i utgangspunktet kompenseres for ved å bruke en sporstabilisator og/eller redusere hastigheten til togene, inntil nok brutto tonn har passert på stedet slik at sporet igjen har ønsket stabilitet (Bane NOR mfl., 2019).

Vedlikeholdsdata om maskinelt sporvedlikehold

Ved utforming av variabel for vedlikehold er ulike vedlikeholdstyper som har forekommet på banestrekningene fordelt i 4 kategorier:

- Sporjustering
- Ballastfordeling
- Skinnesliping
- Sporstabilisering

Sporjustering omfatter alle typer pakking av spor, mens ballastfordeling omfatter alt som har med ballast å gjøre, som ballastfordeling, ballastsupplering og ballastprofilering.

Siden solslyng kan oppstå etter at det har vært utført vedlikehold er antall dager siden sist vedlikeholdt tatt med. I treningssettet er det én kolonne for hver av de 4 kategoriene, som inneholder antall dager siden vedlikeholdstypene ble utført sist for hvert km-segment. For km-segmenter hvor det ikke er notert vedlikehold i Banedata, er det i treningssettet satt inn 100 000 dager siden sist vedlikeholdt.

Sporstabilisering, vibrering av skinnestigen for å øke stabiliteten i sporet etter vedlikehold, er først blitt mer vanlig de siste 5-10 årene, og er derfor "aldriblitt utført på de fleste km-segmenter.

Ballastrensing

Ballastrensing utføres når ballasten "ikke lenger fyller sin nødvendige funksjon på grunn av for høy andel materiale med liten diameter eller innhold av andre uønskede materialer"(sst.) som for eksempel jord eller planterester. Ballastrensing vil ha påvirkning på solslyng ved at dårlig ballast kan føre til redusert stabilitet på sporet, samt at etter utført ballastrensing tar det litt tid før ballasten får satt seg"og oppnår optimal stabilitet.

På grunn av mangelfull og lite pålitelige data om ballastrensing i Banedata, har dette blitt tilegnet egne kolonner for ballastrensing før 1990, ballastrensning fra 1990-2011 og ballastrensing i 2012 eller senere. For hvert segment utgjør andel av strekningen renset i de forskjellige tidsperiodene, verdiene i kolonnene.

4.3.5 Sviller og befestigelse

‘Solslyngrapport – Bane NOR 2018’ (Skogan mfl., 2018) viser at en større andel av solslyngene i forhold til andel av Bane NORs jernbanelinjer har skjedd på strekninger med tresviller med Heyback befestigelse og betongsviller med Pandrol e/Pr befestigelse. Dette ga grunn for å legge inn sviller og befestigelse som variabler i treningssettet. Antall overganger mellom ulike typer sviller og befestigelser

er tatt med siden solslyng har vist seg å kunne oppstå i overganger mellom geometrier med forskjellig stivhet.

For å redusere antall variabler med sviller, er de forskjellige svilletypene delt inn i følgende kategorier med svilletyper i parentes:

- Betongsviller 2,6 m lange (JBV 60, JBV 60 "Dual rail", NSB 95, NSB 93)
- Betongsviller 2,4 m lange (JBV 54, JBV 54 "Dual rail", JBV 97, NSB 90, NSB Enhetssville, Brusville 97 type 2)
- Betongsviller 2,3 m lange(Betongsville Type 2 Pandrol)
- Betongsviller i sporveksler
- Tresviller i sporveksler
- Tresviller (furu, bøk)
- Brusviller i tre (brusville furu og bøk)

4.3.6 Tvangspunkter

Et tvangspunkt, også kalt fastpunkt, er et punkt eller en strekning på jernbanen hvor geometrien er spesielt stiv/sterk og hvor da sideforskyvningsmotstanden er vesentlig sterkere enn for resten av strekningen. Solslyng har vist seg å kunne oppstå ved overganger til tvangspunkter på jernbanen nettopp fordi spenningen i sporet utenfor tvangspunktet kan bygge seg opp akkurat i overgangen til tvangspunktet.

Tvangspunkter tatt med som variabel i treningssettet er stålbruer og andre bruer hvor skinnene er fastmontert på brua og det dermed ikke er gjennomgående ballast, moderne planoverganger som fungerer som tvangspunkter, og sporveksler. Det er talt antall tvangspunkter per km-segment, hvor én bru, én planovergang eller én sporveksel regnes som ett tvangspunkt. Det er tatt hensyn til idriftssatt dato når det gjelder bruer og sporveksler. Det er store mangler i idriftssatt dato på planoverganger så tidsdimensjonen er dermed ikke tatt hensyn til når det gjelder disse, antall planoverganger er konstant for alle tidspunkter.

4.3.7 Sporgeometri

Ifølge solslyngrapporten til Bane NOR oppsto 60 % av solslyngtilfellene i 2018 i «kurver med radius under 600 m, mens det er bare 17 % av Bane NOR sitt nett som består av kurver med R<600 m.» (Skogan mfl., 2018, s. 3). Radius ser derfor ut til å ha betydning for hvor solslyng oppstår og er derfor med som en variabel i treningssettet.

Den minste radiusen som forekommer på hvert km-segment er brukt som variabel. Radiusen er hentet fra datasettet om horisontal sporgeometri fra en applikasjon for beregning av hastigheter i Banedata. Det er ikke tatt hensyn til tidsdimensjonen for denne variabelen, da skinnenes sporgeometri ikke har endret seg i tidsperioden for denne oppgaven.

4.3.8 Tunnel

Siden jernbanestrekninger inne i tunneler vil være mindre eksponert for sol, er andel i % av hvert km-segment som befinner seg inne i tunnel tatt med som en variabel.

Det er ikke tatt stilling til dato når det gjelder tunneler, da ingen nye tunneler er kommet til i løpet av perioden 2017-2018.

4.3.9 Isolerte skinneskjøter

Dagens jernbaneskinner er stort sett helsveist, det vil si at skinnene er sveiset sammen i stedet for lasket slik at man får kontinuerlige skinnestrenger. Det går elektrisk strøm gjennom skinnene som brukes til signalanlegg for å vite hvor togene er. For å kontrollere hvor strømkretsene går legger man inn isoleringen mellom skinnene i stedet for å sveise dem sammen enkelte steder. Dette kalles isolerte skjøter, og er punkter langs skinnene som er svakere enn ellers langs skinnene. Dette er fordi kraftoverføringen skjer via friksjon mellom skinne og laskene og mellom selve laskene og laskeboltene og ikke bare over selve tverrsnittet til skinnene. Om det blir feil i disse skjøtene, kan kraftoverføringen svikte og man får en ekstra lateral kraftkomponent som får sporet til å slå ut (Løhren, 2019b).

Kurver langs jernbanen skal ha kontant krumning, eller lineært stigende eller synkende radius ved overganger til kurver. Dette er for å sikre en jevn lateral kraftkomponent som gir bedre komfort for passasjerer, samt jevn lateral belastning på sporet. Ved isolerte skjøter, derimot blir det ofte en geometrifeil, spesielt i kurver, i form av en kort rett linje. Dette skaper spenningstopper ved skjøtene, og

ved togpasseringer blir det en ekstra dynamisk lateral kraft på grunn av ujevnheten i sporet. Dette øker faren for at solslyng kan oppstå (Løhren, 2019b).

Antallet isolerte skinneskjøter per km-segment er derfor tatt med som en variabel i treningssettet. Data om skinneskjøter er hentet fra Banedata og følgende tilfeller av skinneskjøter er ikke tatt med:

- Uisolerte skjøter
- Tilfeller med tomme felter om hva slags type skjøt det er

Det kan derfor forekomme isolerte skjøter som ikke har kommet med i treningssettet på grunn av manglende informasjon i Banedata.

Det er ikke tatt hensyn til tidsdimensjonen når det gjelder skinneskjøter og trafoer. Treningssettet er derfor identisk for alle tidspunktene.

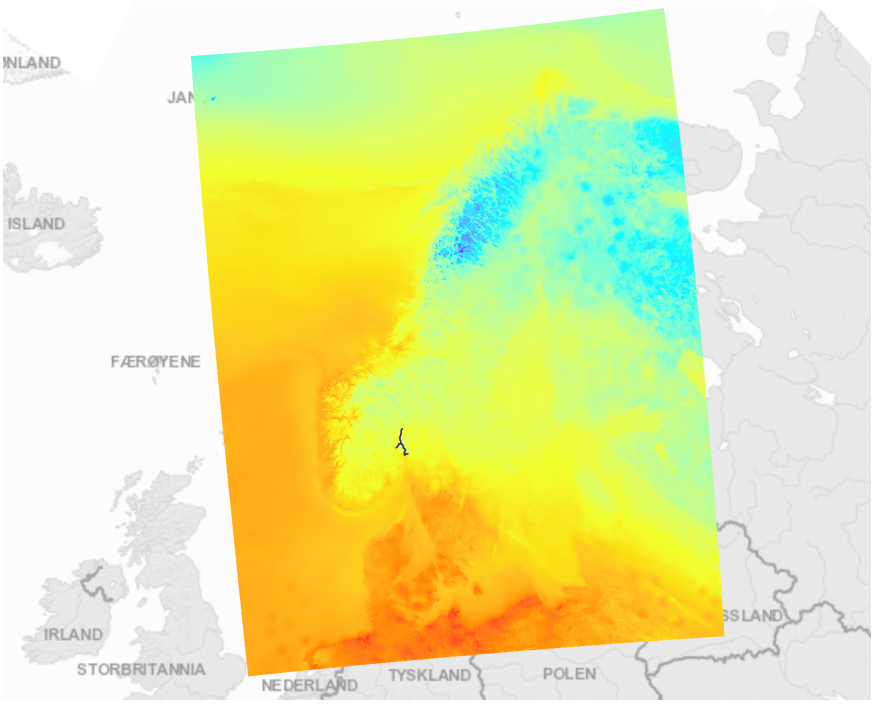
Det er talt antall skinneskjøter og trafoer for hvert km-segment. På hovedbanen hvor det er dobbeltspor, er antallet skinneskjøter og trafoer delt på 2.

4.4 Variabler for vær og terreng

For innsamling av data for vær og terreng er de geografiske datasettene som er omtalt i ”Metode - Datainnsamling / Om datagrunnlaget / GIS-data” brukt.

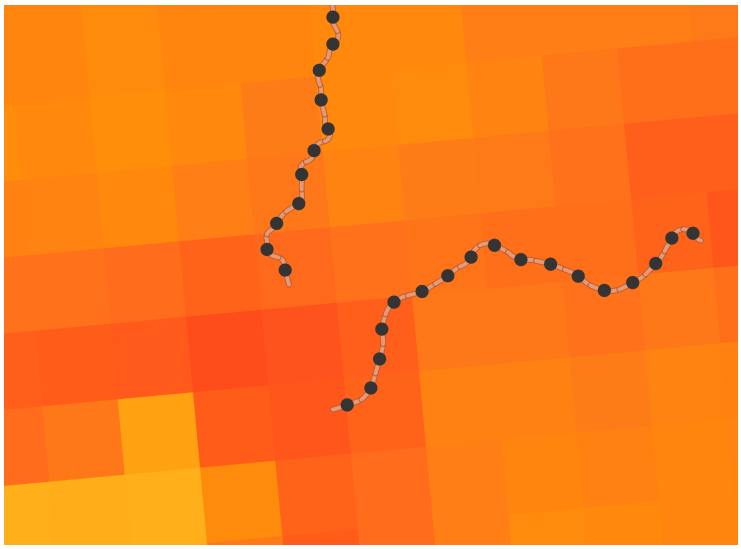
4.4.1 Værdata

Værdata i oppgaven er hentet fra Meteorologisk institutt sin nedlastningstjeneste thredds.met.no. Værdataene ligger lagret som NetCDF-filer som kan sees på som multidimensjonale rasterdatasett. Hver vær-variabel har dimensjonene tid x høyde x Y-koordinat x X-koordinat. Tidsoppløsningen er per time for 67 timer, fra tidspunktet i filnavnet og 67 timer framover. Høyden er antall meter over bakke-nivå, som regel med bare en høyde å velge mellom (2 meter over bakken). Koordinatene er i WGS84 med 2,5 km avstand. Rutenettet som disse koordinatene utgjør har størrelse 929x869 og har utstrekning over hele Skandinavia. Figur 4.3 viser et eksempel av NetCDF-filen med temperaturen 2 meter over bakken, ved midnatt natt til 1. april 2017. De svarte linjene i Oslo-området er jernbane-strekningene som er med i denne oppgaven.



Figur 4.3: Temperatur-raster ved midnatt natt til 1. april 2017, med km-segmentene som svarte linjer i Oslo-området. Værdataene er hentet fra thredds.met.no, kartet er laget i ArcGIS Pro og bruker ESRIs bakgrunnskart (sst.).

For å koble værdataene til km-segmentene, ble det generert midtpunkter på alle km-segmentene i ArcGIS Pro. Videre ble disse midtpunktenes lengde- og breddekoordinater eksportert til en tabell. Via et matlab-skript ble det iterert over koordinatene i tabellen mot koordinatene til en vær-variabel. Korteste euklidiske avstand mellom hvert koordinatpar fra tabellen og vær-variabelen ble styrende for hvilken ”celle” i NetCDF-filen som hører til hvilket km-segment. Figur 4.4 viser et området rundt Oslo hvor oransje linjer viser km-segmentenes plassering (Hovedbanen og sørligste delen av Gjøvikbanen) og de sorte punktene viser midtpunktenes plassering. Bakgrunnen viser temperaturen på ettermiddagen 2. april 2017. Her er det mulig å se NetCDF-filens ”rutestørrelse” på 2,5 x 2,5 km og hvordan disse faller sammen med midtpunktene på km-segmentene.



Figur 4.4: Hovedbanen og sørlige del av Gjøvikbanen, med km-segmentenes midtpunkter mot et temperatur-raster på en NetCDF-fil. Værdataene er hentet fra thredds.met.no og kartet er laget i ArcGIS Pro (ESRI, 2018a).

Oppgavens geografiske og tidsmessige avgrensing har krevd nedlasting av data for 167 ulike plasseringer, 366 ulike dager og 6 ulike vær-variabler. For å kunne hente ut dette på en rask og effektiv måte falt valget på tilgang til NetCDF-filene via en OPeNDAP-protokoll (Støylen, 2016) som gjør det mulig å jobbe direkte mot datastrømmen uten at man trenger å laste ned hele filer. Via funksjonen ”ncread” i MATLAB ble de riktige utvalgene av data som trengtes i oppgaven hentet via et MATLAB-skript.

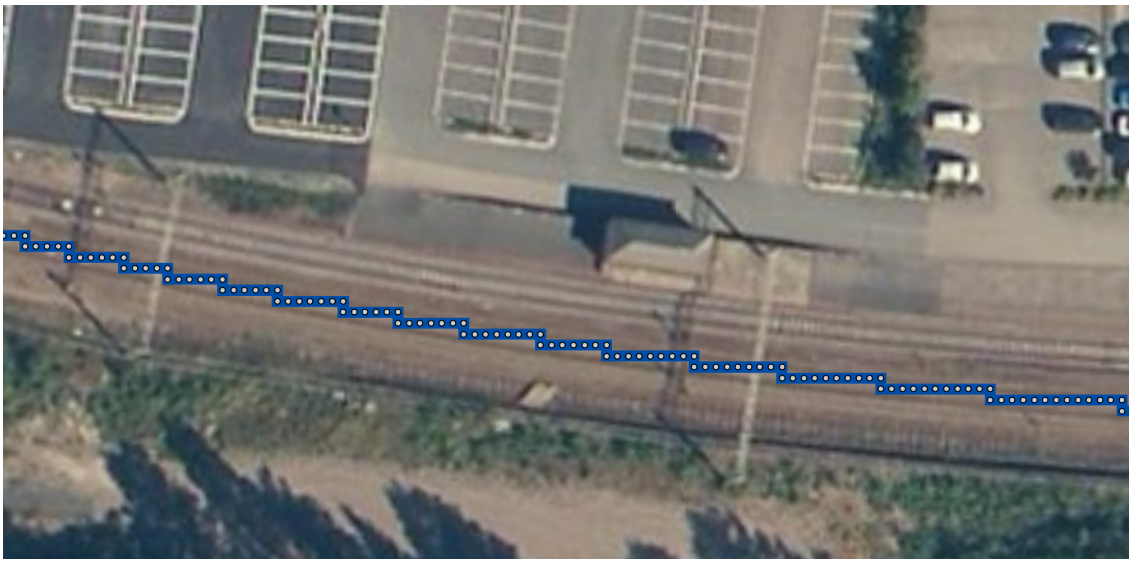
For hvert midtpunkt for hver dag ble det hentet ut værdata fra filen med dato *dagen før*, for så å hente ut værdata fra kl 22 dagen før til kl 21 samme dag. Både temperaturene natten før og hvor raskt temperaturen faller samme ettermiddagen har innvirkning på risikoen for solslyng (Løhren, 2019b). Det ble videre regnet ut minimums-, maksimums-, gjennomsnitts- og mediantemperatur av dette, samt standardavvik som da utgjør 5 kolonner i datasettet.

Vær-variablene som ble hentet ut er luft-temperatur 2 meter over bakken (”air_ temperature_ 2m”), akkumulert nedbørmengde (”precipitation_ amount_ acc”), sky-areal-andel, tåke-areal-andel (”fog_ area_ fraction”) og vindhastighet i kastene (”wind_ speed_ of_ gust”).

5 variabler med 5 ulike statistiske størrelser utgjør da 25 ulike kolonner i datasettet.

4.4.2 Høyde over havet

Det ble regnet ut et gjennomsnitt av høyde over havet for hvert km-segment for å lage variabelen ”Hoyde” i datasettet. Dette ble gjort på følgende måte. Brukte verktøyet ”Extract by Mask” i ArcGIS Pro til å beskjære terrengmodellen etter den romlige utstrekningen til km-segment-datasettet. Verktøyet ”Raster to points” brukt for å konvertere den beskjærte høydemodellen til punkter. Punktene beholdt høydemodellens rasterverdier (høyde over havet) som attributter. Et utsnitt fra Nittedal stasjon på Gjøvikbanen av den beskjærte høydemodellen (i blått) og de genererte punktene (i lysegrått) vises i figur 4.5. Der kan man tydelig se mønsteret av ”raster-rutenettet”.



Figur 4.5: Beskjært terrengmodell i blått og genererte punkter i lysegrått på Nittedal stasjon. Kartet er laget i ArcGIS Pro og bruker ESRIs bakgrunnskart (sst.).

Videre ble det kjørt en romlig kobling mellom de genererte punktene og km-segmentdatasettet, slik at punktenes høyde-attributt blir lagt inn i attributtene til km-segmentene. Da det er mange punkter til hvert km-segment, ble det valgt gjennomsnitt som løsning for hvilke punktverdier som skulle legges til attributtene til km-segmentene.

4.4.3 Retning

Siden sola hele tiden har en posisjon relativt til jernbanesporet, kan retning på hvert km-segment være interessant å utforske i forhold til solslyng.

Retningen er lagret som azimut, vinkel i radianer med klokka i forhold til nord.

For hvert km-segment er retningen regnet ut i ArcGIS Pro på følgende møte:

1. Brukte funksjonen 'Calculate Geometry' for å beregne hvert segments start- og endekoordinater i UTM sone 32N.
2. Brukte funksjonen i kodeblokk 4.1 for å beregne vinkelen på linjene fra km-segmentenes startpunkt til sluttpunkt i forhold til nord.

```
import math as m

def direction(x_start, x_end, y_start, y_end):

    # Calculate euclidian lengths
    dx = x_end - x_start
    dy = y_end - y_start

    # Calculate angel
    azimuth = m.atan(dx / dy)

    # Angle in the right quadrant
    if dy < 0:
        azimuth += m.pi
    elif dx < 0 and dy > 0:
        azimuth += 2 * m.pi

    return azimuth
```

Listing 4.1: Funksjon for beregning av retningen til km-segmentene

4.4.4 Sinuosity

Sinuosity er et mål på hvor kurvet eller rett en linje er. Det er definert ved kurvens lengde dividert på euklidisk lengde mellom kurvens start- og endepunkt. Jo høyere sinuosity, jo mer kurvet er linja. Som nevnt tidligere oppstår solslyng ofte i kurver med liten radius, derfor er det tenkt at graden av sinuosity kan ha innvirkning på forekomst av solslyng.

For beregning av sinuosity er følgende funksjon brukt på km-segmentene i ArcGIS Pro.

```
import math as m

def sinuosity(x_start, x_end, y_start, y_end, length):

    # Calclulate euclidian length
    euc_l = m.sqrt( (x_end - x_start)**2 + (y_end-y_start)**2 )

    return length / euc_l
```

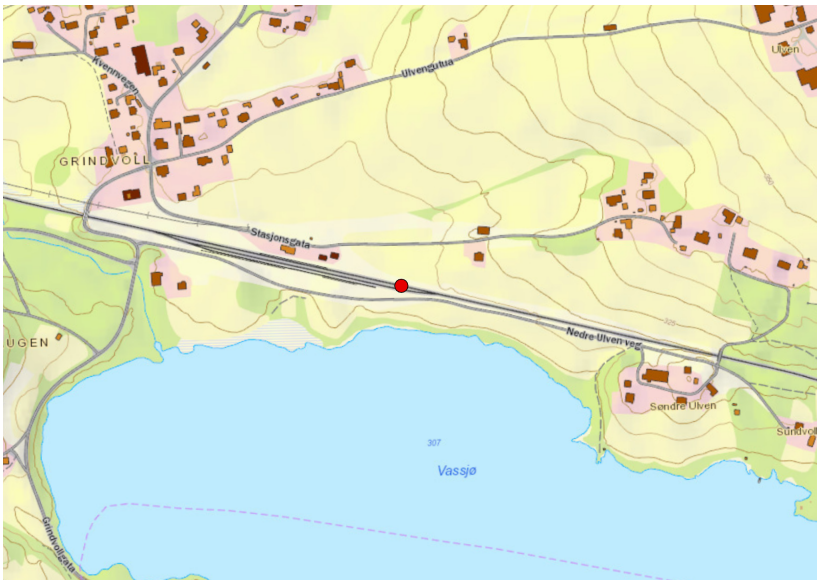
Listing 4.2: Funksjon for beregning av sinuosity

Km-segmentenes koordinater er i og segmentenes lengde er beregnet med funksjonen ”Calculate Geometry”.

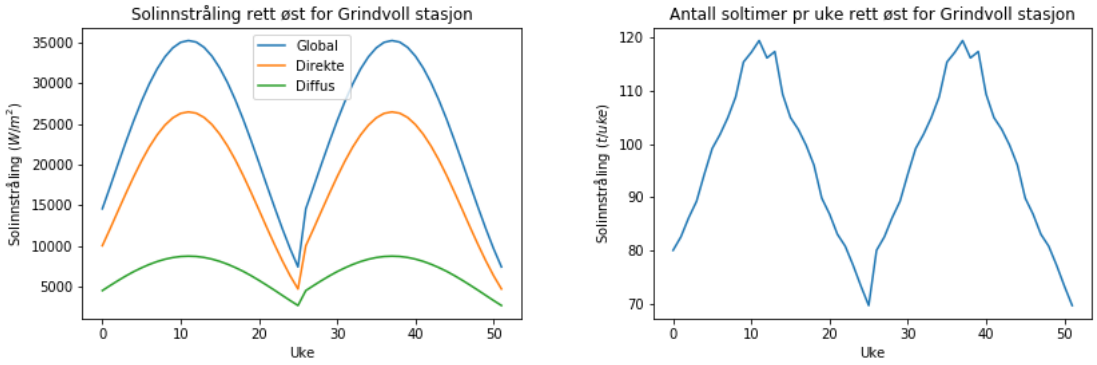
4.4.5 Solinnstråling

Siden stråling fra sola har stor innvirkning på temperatur i stål i tillegg til luft-temperaturen, ble det laget variabler basert på mengde solinnstråling langs jernbanen. Fire variabler ble generert, antall soltimer, mengde direkte stråling (W/m^2), mengde diffus stråling og global stråling som er summen av direkte og diffus stråling. Beregningene av solinnstråling bygger på metoder fra hemisfærisk utsyn-algoritme utviklet av Rich et al. og videre utviklet av Fu og Rich (ESRI, 2018b). Solinnstrålingen og antall soltimer ble beregnet på punkt-plasseringer med ca 100 meters mellomrom langs jernbanen. Videre er det regnet ut gjennomsnitts-, minimums- og maksimumsverdi for hvert km-segment basert på disse ti punktene.

Figurene 4.6 - 4.9 viser solinnstrålingen på to forskjellige steder langs jernbanen. Figur 4.6 viser et punkt med mye sol. Der ligger i et åpent jordbrukslandskap og det er kun et vann sør for punktet og en åsside nord for punktet. Figur 4.8 viser et punkt med lite sol. Det ligger nord for en ås, rett utenfor en tunnel-inngang. Grafene viser mengde solinnstråling og antall soltimer gjennom hele tidsperioden til treningssettet, april-september i 2017 og 2018.

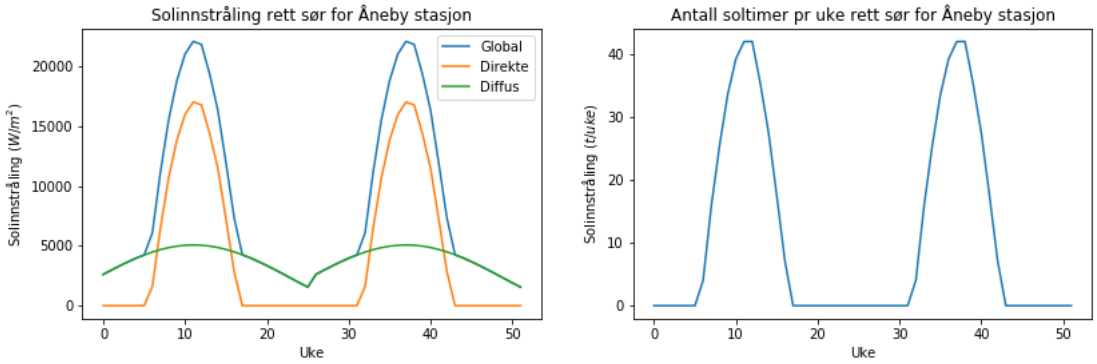


Figur 4.6: Et punkt (rød sirkel) med mye solinnstråling på km 65 på Roa-Hønefossbanen, rett før Grindvoll stasjon. Kartet er laget i ArcGIS Pro med bakgrunnskart av ESRI (ESRI, 2018a).



Figur 4.7: Et punkt med mye solinnstråling på km 65 på Roa-Hønefossbanen, rett før Grindvoll stasjon. Laget med Matplotlib (Hunter, 2007)

Figur 4.8: Et punkt (rød sirkel) med lite solinnstråling på km 27 på Gjøvikbanen, rett før Åneby stasjon. Kartet er laget i ArcGIS Pro med bakgrunnskart av ESRI (ESRI, 2018a).



Figur 4.9: Et punkt med lite solinnstråling på km 27 på Gjøvikbanen, rett før Åneby stasjon. Man kan se at blant annet de første seks ukene er det ingen direkte solinnstråling her og antall soltimer er null. Laget med Matplotlib (Hunter, 2007)

Detaljert beskrivelse av beregningen av solinnstrålingen i ArcGIS Pro

Genererte punkter pr 100 meter langs km-segmentene, inkl endepunkter, med verktøyet ”Generate Points Along Lines”.

Gjorde en romlig seleksjon av punktene basert på overlapp med linje-datasettet med tunneler, med søkeradius på 1 meter. Slettet så de valgte punktene.

Kjørte en romlig kobling mot km-segment-datasettet for å få banenummer og sporkilometer med som attributt på punkt-datasettet. Søkeradius på 1 meter.

Kjørte verktøyet ”Points Solar Radiation” med standard parametre, bortsett fra dagsintervall på 7 dager (kjør ny beregning for hver uke) og timeintervall på 1 time (beregn solinnstrålingen hver time og kjør et integral over dette). Terrengmodellen med oppløsning på 1 meter ble brukt som overflatemodell.

Videre ble punktenes attributt-tabell eksportert til python og omformet til treningssett med gjennomsnitts-, minimums- og maksimumsverdier for hvert km-segment.

Av en eller annen grunn har kolonnen med minimumsverdier av global solinnstråling (”Solar_ radiation_ min”) ikke kommet med i treningssettet. Sannsynligvis har ikke dette hatt så mye å si i maskinlæringen, da denne variabelen uansett kun er en sum av variablene direkte og diffus stråling.

Kapittel 5

Metode - Analyse

5.1 Eksplorativ analyse

Når det kjøres trening og prediksjon med maskinlæringsalgoritmer er det en fordel å ha god kjennskap til datagrunnlaget. Utforsking av datagrunnlaget kan bidra til bedre intuisjon i forhold til valg av parametre og modeller for maskinlæring, og kan dessuten kanskje gi en pekepinn til hva man kan forvente seg. Er for eksempel datasettet lineært separabelt for en eller flere variabler, det vil si at det går an å skille solslyngtilfellene fra tilfellene hvor det ikke var solslyng kun med en lineær funksjon, vil det holde å bruke enklere maskinlæringsalgoritmer som Perceptron og Adaline. Er ikke treningssettet lineært separabelt, trengs det mer avanserte algoritmer som i for eksempel SVM og nevrale nettverk.

Det ble dermed undersøkt om fordelingen av de ulike variablene varierte for tilfellene med solslyng og tilfellene hvor det ikke var solslyng. Resultatene av dette ble vist i histogrammer, hvor stolpebredden er forsøkt satt, slik at histogrammet gjengir strukturen i datasettet best mulig. Siden antallet solslyngtilfeller kun utgjorde 55 av 61122 tilfeller, ble solslyngtilfellene ganget opp med $61122/55 \approx 1111.3$, slik at begge fordelingene består av like mange tilfeller. De viste fordelingene må derfor tas med en liten klype salt, da fordelingen med solslyngtilfeller bygger på over 1000 ganger så få tilfeller.

Prinsipalkomponentanalyse

Prinsipalkomponentanalyse (videre PCA) har i denne oppgaven blitt brukt både til eksplorativ analyse og til dimensjonalitetsreduksjon i noen av maskinlæringsmodellene som ble testet ut.

PCA i eksplorativ analyse øyemed, ble utført for å undersøke om det er noen systematisk varians i datasettet. Det ble undersøkt om prinsipalkomponentene kunne skille solslyngtilfellene fra ikke-solslyngtilfellene og hvor mye av variansen i datasettet de første komponentene viser. Det ble også undersøkt hvilke variabler som blir mest forklart av de første komponentene og selvfølgelig hvilke variabler som ser ut som er viktige for solslyng. Python-biblioteket Hoggorm (Tomic, 2017) ble brukt for å generere scorer, ladninger og kumulativt forklart varians, med 50 komponenter og kryssvalidering med 4 folder som parametere.

PCAen utført på data som allerede var standardisert med funksjonen ”preprocessing.StandardScaler” i scikit-learn (Pedregosa mfl., 2011).

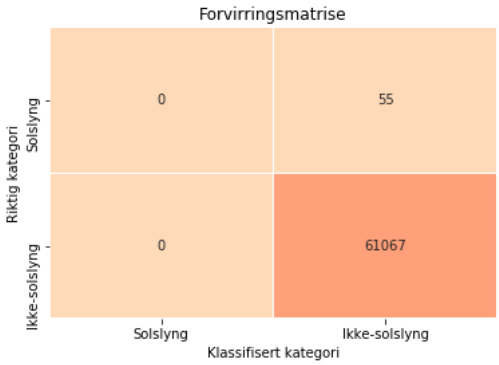
5.2 Maskinlæring

5.2.1 Valg av prestasjonsmål

Som tidligere nevnt i 2.3.1 Prestasjonsmål, så er det i maskinlæring ikke nødvendigvis likegyldig hvilket prestasjonsmål som velges, spesielt ikke når det gjelder ubalanserte data. F1-scoren ble valgt som prestasjonsmål, da denne nyanserer prestasjonen mye bedre på ubalanserte data enn det ROC AUC-scoren eller treffsikkerheten (% riktig klassifiserte) gjør. F1-scoren var også enkel å implementere som kostfunksjon og prestasjonsmål for MLNN og RNN, som gjør at alle maskinlæringsmodellene kan vurderes etter samme prestasjonsmål som jo er en fordel når de skal sammenlignes.

5.2.2 Baseline

Før man går i gang med maskinlæringen er det nyttig å etablere en baseline. En baseline er den treffsikkerheten man ville fått ved å klassifisere tilfellene helt tilfeldig. Når man senere måler maskinlæringsmodellenes prestasjoner måles dette i forhold til baselinen. I denne oppgaven vil det si å predikere alle tilfellene til å ikke være solslyng-tilfeller. Da ville man fått en treffsikkerhet på $1 - \frac{55}{61122} \times 100 \approx 99.91\%$ riktig klassifiserte tilfeller. Tilsvarende ville dette gitt følgende forvirringsmatrise og F1-score:



Figur 5.1: Forvirringsmatrise ved tilfeldig klassifisering. Laget med Seabron (Waskom mfl., 2018).

$$PRE = \frac{TP}{TP+FP} = \frac{0}{0+0} = 0$$

$$REC = \frac{TP}{TP+FN} = \frac{0}{0+55} = 0$$

$$F1 = 2\frac{PRE \times REC}{PRE+REC} = 2\frac{0 \times 0}{0+0} = 0$$

Modellenes prestasjoner ble målt i forhold til en baseline som besto av F1-score = 0.

5.2.3 Dimensjonalitetsreduksjon

Som dimensjonalitetsreduksjon i maskinlæringen ble PCA utført på treningssettet ved trening. Videre ble testsettet transformert etter modell fra PCAen til treningssettet. Modulen ”decomposition.PCA” i biblioteket scikit-learn (Pedregosa mfl., 2019) ble brukt med ti komponenter. Eksempel på kode med en maskinlæringsmodell som bruker PCA som dimensjonalitetsreduksjon vises i listing 5.1, linje 33-36.

```
1  from sklearn.model_selection import train_test_split
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.decomposition import PCA
4  from sklearn.svm import SVC
5
6  # Dataset
7  X = samples
8  y = target
9
10 # Parameters
11 n_components = 10
12 c = 10
13 kernel = 'rbf
14 classweight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}
15 random_states = range(10)
16
17 # Lists to store the results
18 train_accuracy = []
19 test_accuracy = []
20
```

```
21
22 for rs in random_states:
23
24     X_train, X_test, y_train, y_test = train_test_split(X, y,
25                                                         test_size=0.3,
26                                                         stratify=y,
27                                                         random_state=rs)
28     # Standardize the data
29     sc = StandardScaler()
30     X_train_std = sc.fit_transform(X_train)
31     X_test_std = sc.transform(X_test)
32
33     # Decompose the data by PCA
34     pca = PCA(n_components=n_components, random_state=rs)
35     X_train_std_pca = pca.fit_transform(X_train_std)
36     X_test_std_pca = pca.transform(X_test_std)
37
38     # Define model
39     svc = SVC(C=c, kernel=kernel,
40               class_weight=classweight(y_train),
41               random_state=rs)
42
43     # Fit and predict model
44     svc.fit(X_train_std_pca, y_train)
45     y_train_pred = svc.predict(X_train_std_pca)
46     y_test_pred = svc.predict(X_test_std_pca)
47
48     # Save the results
49     train_accuracy.append(svc.score(X_train_std_pca, y_train))
50     test_accuracy.append(svc.score(X_test_std_pca, y_test))
```

Listing 5.1: Eksempel-kode for dimensjonalitetsreduksjon med PCA i maskinlæring

5.2.4 Resampling

På noen av maskinlæringsmodellene ble datasettet resamplet ved bruk av Python-biblioteket imbalanced-learn sin modul SMOTE (Lemaître, Nogueira og Aridas, 2017). Dette ble gjort ved å upsample treningssettet etter standardiseringen og før treningen av modellen, altså samme plassering som PCAen ble utført. Derfor viser eksempelkode 5.2 kun de linjene som har med SMOTE å gjøre.

```
1  from imblearn.over_sampling import SMOTE
2
3  # Define SMOTE instance
4  sm = SMOTE(sampling_strategy=1/3, # Upsampling minority
5                                    # class to 1/3 of
6                                    # majority class
7             n_jobs=-1,
8             random_state=2)
9
10 # Resample training data
11 X_train_std_res, y_train_res = sm.fit_sample(X_train_std, y_train)
```

Listing 5.2: Eksempel-kode for upsampling med SMOTE i maskinlæring

For alle modellene som ble trent med resamplede data ble SMOTE med samme parametere brukt, slik at modellene blir mest mulig sammenlignbare. Å upsample solslyngtilfellene (minoritetsklassen) til 1/3-dels antall av antallet ikke-solslyng-tilfeller ble valgt på følgende grunnlag. Siden datasettet er så enormt ubalansert, var det med hensikt å gjøre resamplingen så varsom som mulig, uten å endre for mye på strukturen i datasettet. Derfor ble antallsforholdet 1:3 valgt som en mellomting mellom å få datasettet mest mulig balansert, og å ikke resample for mye.

5.2.5 Maskinlæringsalgoritmer

Maskinlæringsalgoritmene som ble testet ut er Random Forest, Isolation Forest, logistisk regresjon, Support Vector Classification (videre SVC). Innenfor dyp læring (Deep learning) ble algoritmene Multi-Layer Neural Network (videre MLNN) og Recurrent Neural Network (RNN) testet ut.

5.2.6 Oppdeling av datasettet

I maskinlæringssammenheng blir datasett vanligvis oppdelt i tre deler, ett treningssett og et valideringssett som brukes til trening av modellen og tilpasning av parametere. Fordi datasettet i denne oppgaven kun inneholdt 55 solslyngtilfeller, ble det besluttet å kun dele opp datasettet i to deler, hvor 70 % av datasettet er treningssett og de resterende valideringssett for å få så mange solslyngtilfeller å trene på som mulig. Det at datasettet ikke ble oppdelt i et tre, med et test-sett for å sjekke prestasjonen til modellen helt til slutt, gjør at modellene som er trent i oppgaven har større risiko for å være overtilpasset. Det vil si at det er større risiko for at modellene vil prestere dårligere på eventuelt nye datasett, fordi det har tilpasset seg strukturene i oppgavens datasett i for stor grad.

For enkelhets skyld blir oppgavens valideringssett omtalt som treningssett videre.

Oppdelingen foregikk med scikit-learn-funksjonen ”model_ selection.train_ test_ split”, med jevnest mulig fordeling av solslyngtilfellene i treningssettet og testsettet med hensyn på antall tilfeller i de to settene, som ble spesifisert ved parameteren ”stratify=y”. Dette er for å sikre at ca 30 % av solslyngtilfeller blir med i testsettet.

Hver modell ble kjørt ti ganger med ti ulike oppdelinger av datasettet, hvor gjennomsnittet av resultatene fra de ti oppdelingene ble bevart. På denne måten blir resultatene mer pålitelige og mindre sensitive for påvirkning fra tilfeldig variasjon i klassifiseringen som følge av tilfeldige start-vekter.

Oppdeling av datasettet for RNN blir redegjort for på slutten av kapittelet.

5.2.7 Standardisering

Treningssettet ble for hver oppdeling standardisert med scikit-learn-funksjonen ”preprocessing.StandardScaler”. Videre ble testsettet standardisert etter modell fra treningssettet.

5.2.8 Klasse-vekting (Class-weight)

De fleste maskinlæringsmodeller i scikit-learn har støtte for å påvirke de tilfeldige startvektene til treningssettet. Dette er en metode som kan hjelpe treningen av ubalanserte datasett, da modellene skal vekte tilfellene i minoritetsklassen høyere enn tilfellene i majoritetsklassen. Dette gjør at tilfellene i minoritetsklassen får høyere prioritet.

Klasse-vektene ble spesifisert med funksjonen i kode-blokken 5.3. Denne funksjonen tilegner tilfeller i klasse 1 (solslyng) vekter lik antallet tilfeller i klasse 0 (ikke-solslyng). Tilfellene i klasse 0 får tillagt vekter lik antall tilfeller i klasse 1.

```
1 class_weight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}
```

Listing 5.3: Funksjon for utregning av klasse-vekter for maskinlærings-modellene

Alle modeller bortsett fra Isolation Forest ble kjørt med klasse-vekting. RNN fikk klasse-vektene definert på en egen måte som er redegjort for på slutten av kapittelet.

5.2.9 Prestasjonsmål og kostfunksjon

For alle maskinlæringsmodellene ble F1-score brukt som presisjonsmål.

For implementering av F1 kostfunksjon i MLNN og RNN i Keras ble en deriverbar versjon av F1-scoren brukt som kostfunksjon Denne ble hentet fra (Haltuf, 2018). Koden til kostfunksjonen vises i kode 5.4. Linje 3 har jeg lagt til selv, for å illustrere hva variabelen ”K” er for noe.

```
1 import tensorflow as tf
2
3 from keras import backend as K # Min modifikasjon
4
5 def f1(y_true, y_pred):
6     y_pred = K.round(y_pred)
7     tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
8     tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
9     fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
```

```
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)

def f1_loss(y_true, y_pred):

    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return 1 - K.mean(f1)
```

Listing 5.4: Funksjon for å regne ut F1-score. Beregnet for bruk som kostfunksjon i Keras-modeller. Koden er hentet fra (Haltuf, 2018)

Denne F1-kostfunksjonen og prestasjonsmålet F1-score ble satt som parametere på følgende måte i Keras-modellen.

```
def build_model():
    # Build model
    model = models.Sequential()
    model.add(layers.Dense(1000, activation='relu',
                           input_shape=(X.shape[1],)))
    model.add(layers.Dense(1, activation='sigmoid'))

    # Compile model
    model.compile(optimizer=optimizers.SGD(lr=0.01)
                  loss=f1_loss,     # Custom loss function
                  metrics=[f1])     # Custom metric

    return model
```

Listing 5.5: Funksjon for å bygge et tett nevralt nettverk. kostfunksjonen (loss function) og prestasjonsmålet (metrics) er egendefinert.

Fordi kostfunksjonen og prestasjonsmålet i disse modellene da blir de samme, ble kun F1-scoren brukt til å måle prestasjonen.

5.2.10 Parameter-tuning

For Random forest ble det undersøkt hvordan antall beslutningstrær og maksimum tredybde innvirket på prestasjonen. Da antallet beslutningstrær ikke viste seg å ha så mye å si, annet enn at det måtte være mange av dem (ca over 100), så ble det maksimum dybde som ble mest undersøkt med antall trær konstant satt til 200.

For logistisk regresjon var det parameteren C som ble undersøkt med ulike kombinasjoner av ”Saga” eller ’Liblinear” som ”solver” og L1 og L2 som regulariseringer. Saga og Liblinear ble valgt ut fordi begge disse har støtte for L1-regularisering (Pedregosa mfl., 2019)..

Ved maskinlæring ved SVC ble også parameteren C undersøkt. Det ble brukt radiell basis-funksjon som kjerne, for å undersøke hvordan prestasjonen ble med muligheter for opp-dimensjonering av datasettet, se 2.3.6 Support Vector Machines (SVM).

5.2.11 Spesielt for Isolation Forest

Siden Isolation Forest ikke er en supervised algoritme, ble treningen av algoritmen kjørt på hele datasettet. Målingen av prestasjon består da kun av én F1-score og fra fra denne treningen. Isolation Forest er heller ikke kjørt på resamplet datasett, da dette er en algoritme for å skille uteliggere fra resten av datasettet. I dette ligger det av datasettet skal være ubalansert. I scikit-learn er det for Isolation Forestikke støtte for å få ut sannsynlighetene for klassetilhørighet, slik at man ikke får justert terskelen for følsomhet ovefor solslyng.

Prestasjonen til Isolation Forest ble undersøkt ved ulikt antall beslutningstrær.

5.2.12 Spesielt for Multi-Layer Neural Network

Når det gjelder MLNN, er det utrolig mange muligheter å velge i når man skal prøve ut forskjellige modeller. Det ble valgt en strategi basert på et grid-search-prinsipp. Det vil si velge ut hvilke parametere og hvilke parameterverdier man vil teste ut, og deretter teste alle mulige kombinasjoner av forskjellige parametere og parameter-verdier. Det ble derfor gjort en brainstorming på alle mulige kombinasjoner av modeller som kunne være nyttige å teste ut. Dette viste seg å bli over 300 modeller, så det ble nødvendig å gjøre en utvelgelse av disse til et antall modeller som var realistisk å få til innenfor rammene av denne oppgaven. Følgende modeller ble valgt ut til testing:

- 3 MLNN-modeller med ett skjult lag og antall noder på henholdsvis 10, 100, og 1000 noder.
- 3 MLNN-modeller med to skjulte lag og antall noder på henholdsvis 10, 100, og 1000 noder.
- 3 MLNN-modeller med tre skjulte lag og antall noder på henholdsvis 10, 100, og 1000 noder.

Videre ble de modellene som fikk høyest F1-score og resulterte i stor overtilpasning valgt ut og testet på nytt, men da med et eller flere dropout-lag lagt til. Det ble lagt til et dropout-lag pr skjulte lag med en dropout-verdi på 0.5.

Neste steg ble å velge ut de modellene som hittil hadde prestert best, og teste ut med SMOTE resampling og dimensjonalitetsreduksjon med PCA.

Rectified linear unit (ReLU) er brukt som aktiveringsfunksjon i de skjulte lagene, mens sigmoid-funksjonen ble brukt i output-lagene siden dette er binær klassifisering.

MLNN ble trent og validert på samme måte som de andre maskinlæringsalgoritmene, med ti ulike oppdelinger i trenings- og testsett for hver modell. Det var nødvendig å prøve ut ulike optimizere og learning rate for at modellen skulle ha noen prestasjon som var høyere enn baseline på 0.0. Stochastic gradient descent ble valgt på bakgrunn av dette. Justering av decay og momentum ble også testet ut, men så ikke ut til å gi noen særlig bedre påvirkning enn å kun justere læringsraten og ble derfor ikke brukt i maskinlæringsmodellene brukt i oppgaven.

Alle modellene ble kjørt med så mange epoker at det til slutt ble en tydelig over- eller undertilpasning. Deretter ble prestasjonen målt ved å se på høyeste F1-score for test-settet.

Eksempel-kode for en MLNN-modell med to dense-lag og to dropout-lag vises i kodeblokk 5.6.

```
from keras import layers
from keras import models
from keras import optimizers

def build_model():
    model = models.Sequential()

    # Add dense layer with 1000 nodes
    model.add(layers.Dense(1000, activation='relu',
                           input_shape=(X.shape[1],)))

    # Add dropout-layer with dropout rate = 0.5
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(1000, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(1, activation='sigmoid'))
```

```
17
18      # Compile. Custom optimizer and learning rate
19      model.compile(optimizer=optimizers.SGD(lr=.001),
20                    loss=f1_loss,
21                    metrics=[f1])
22
23      return model
```

Listing 5.6: Eksempel på MLNN-modell med to lag med 100 noder hver i tillegg til to dropout-lag.

5.2.13 Spesielt for Recurrent Neural Network (RNN)

For RNN er det også utrolig mange mulige kombinasjoner av modeller å velge mellom, men her ble det prøvd ut en annen strategi, ved å teste ut én modell og ut fra resultatet av denne, endre modellen litt og se om det blir bedre. Dette gjøres da flere ganger. Framgangsmåten ble inspirert av eksempelet i en Notebook av François Chollet (Chollet, 2017).

I Chollets eksempel gjøres det prediksjon av temperatur-data, basert på flere ulike vær-variabler. I datasettet til Chollet representerte vær rad et gitt tidspunkt, mens vær kolonne representerte en vær-variabel. Det var på denne måten et sekvensielt datasett med en tids-variabel. Treningssettet ble generert ved tilfeldig utvalgte perioder med værdata og maskinlæringsmodellen skulle basert på dette predikere temperaturen for en gitt periode som lå foran i tid i forhold til treningssettet.

Datasettet i denne masteroppgaven har ikke bare en tidsdimensjon, men også en romlig dimensjon basert på km-segmentenes rekkefølge i rommet. Det har ikke lyktes å finne noen metode for å bruke RNN på datasett med to slike sekvensielle dimensjoner, så dette krevde en spesial-løsning. Det ble bestemt å snu om på treningssettet, ved å la alle tidspunktene for et km-segment komme etter hverandre. Variablene utgjorde fortsatt kolonnene, mens rekkefølgen på radene ble da første dag/første segment, andre dag/første segment, tredje dag/første segment osv. På denne måten ble tiden representert ved radene, ikke som to sommerhalvår som datasettet egentlig består av, men som $167 \times 2 = 334$ ”sommerhalvår” etter hverandre. Den romlige dimensjonen i datasettet fikk ikke komme fram på denne måten, men datasettet ble i hvert fall tilpasset slik at det gikk an å kjøre det med en RNN-algoritme.

Dette førte til svært lave F1-scorer eller F1-score = 0, så det ble bestemt å ta ut alle km-segmentene som aldri hadde hatt solslyng, og kun kjøre på de 38 km-segmentene hvor det har vært solslyng minst én gang innenfor oppgavens tids-avgrensning. Dermed ble det $38 \times 2 = 76$ sommerhalvår å trene og predikere på. En argumentasjon for at det også kan være fornuftig å prøve ut et sånt utvalg fra datasettet,

er at det kan være aktuelt i realiteten å kun predikere solslyng på de områdene som har større risiko for at solslyng oppstår. Å redusere utvalget har flere fordeler, ved at det går raskere både å samle inn data og raskere for maskinlæringsmodellen å trene og predikere fordi man får færre kilometer med jernbane å arbeide med.

Det ble først testet ut en modell identisk med eksempelet til Chollet, for så å gjøre endringer og se om det gikk an å oppnå høyere F1-score. Endringene besto i å legge til dropout og redusere antall noder for å se om det gikk an å redusere overtilpasningen.

I noen av modellene ble det brukt dropout og recurrent dropout. Disse ble begge satt til 0.2, da dette utgjorde en merkbar forskjell

Maskinlæringen med RNN ble utført i Google Colaboratory (Google, 2019). I motsetning til de andre maskinlæringsalgoritmene ble det kun kjørt fem ulike oppsplittinger av datasettet i stedet for 10, på grunn av lang kjøretid (running time).

Klasse-vekting ble gjort ved å at solslyngtilfellene fikk vekt lik antallet ikke-solslyngtilfeller, mens ikke-solslyngtilfellene fikk vekt lik antallet solslyngtilfeller.

5.2.14 Evaluering av modellene i forhold til praktisk nytte i prediksjon av solslyng

Den modellen med høyest F1-score ble kjørt på nytt med de parameterene som ga best resultat, slik at det var mulig å få ut selve prediksjonene og se på forvirringsmatrise, sensitivitet og spesifisitet. Prediksjonene ble skrevet ut med sannsynlighet for hvilken klasse modellen mente de tilhørte. Deretter ble det testet ut ulike terskel-verdier og undersøkt antall riktig predikerte solslyng, antallet solslyng som ikke ble oppdaget, antallet ikke-solslyng-tilfeller som ble klassifisert som solslyng osv.

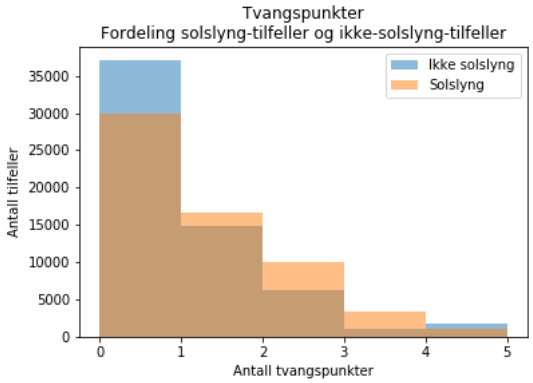
Fra Alf Helge Løhren i Bane NOR ble det beskrevet to scenarioer som kunne være ønskelig å få til. Det første var å predikere solslyngene mest mulig nøyaktig, slik at tiltak kan konsentreres på de mest utsatte stedene. Det neste ble å finne steder med en viss risiko for solslynghendelse, hvor det da kan utføres preventive tiltak (Løhren, 2019a). Metoden med terskling og forvirringsmatrise blir da en måte å prøve å etterkomme dette ønsket.

Kapittel 6

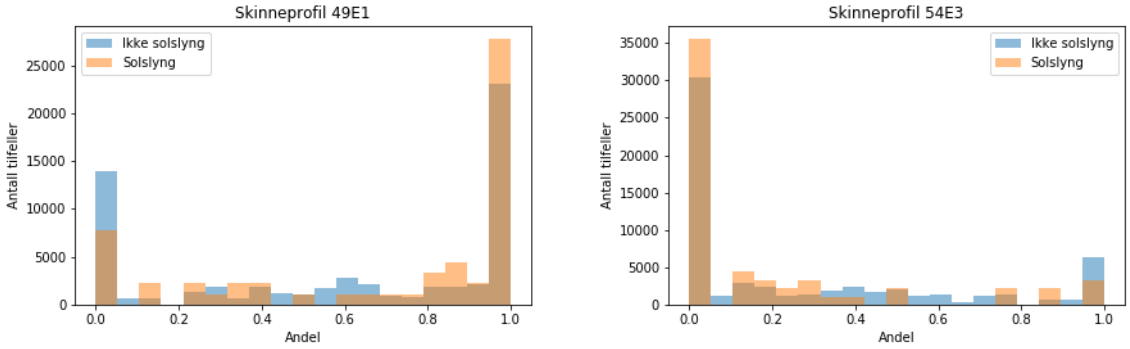
Resultater

6.1 Eksplorativ analyse

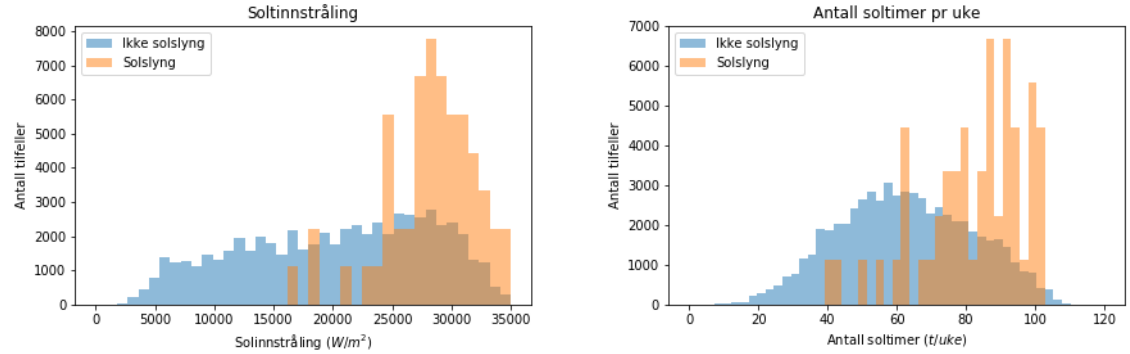
Følgende figurer viser resultatet av datautforskningen. Det ble hovedsakelig undersøkt om fordelingen av de ulike variablene varierte for tilfellene med solslyng og tilfellene hvor det ikke var solslyng. De variablene med tilsynelatende høyest ulikhet i fordelingene er vist nedenfor i histogrammer, hvor blå stolper viser ikke-solslyng-tilfeller og oransje stolper viser solslyngtilfellene. solslyngtilfellene er multiplisert opp slik at det er like mange tilfeller i begge klassene. Stolpene er delvis gjennomsiktige, så det går an å se fordelingene mot hverandre. Kommentarer står skrevet i figur-tekstene.



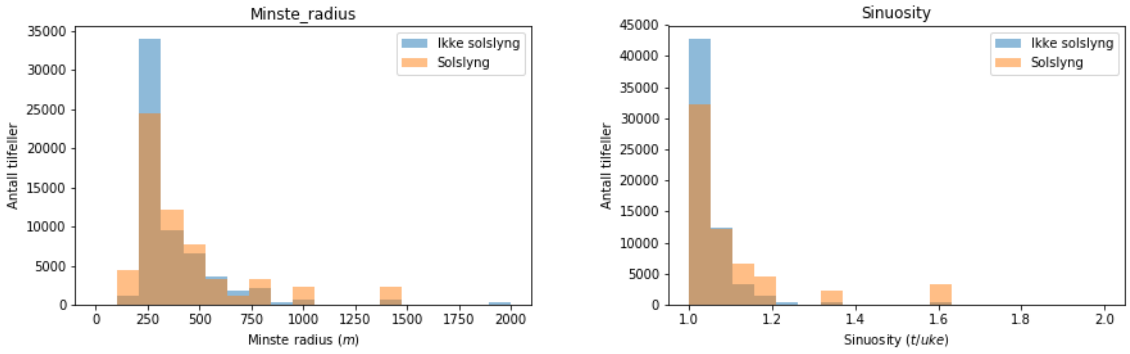
Figur 6.1: Fordelingen av antall tvangspunkter for tilfellene med og uten solslyng. Det er mulig å se en liten tendens til at det er flere tvangspunkter der det har vært solslyng, men forskjellen er så liten at dette kan være tilfeldig. NB: Solslyngtilfellene er multiplisert opp slik at antallet tilfeller i de to fordelingene blir like. Laget med Matplotlib (sst.)



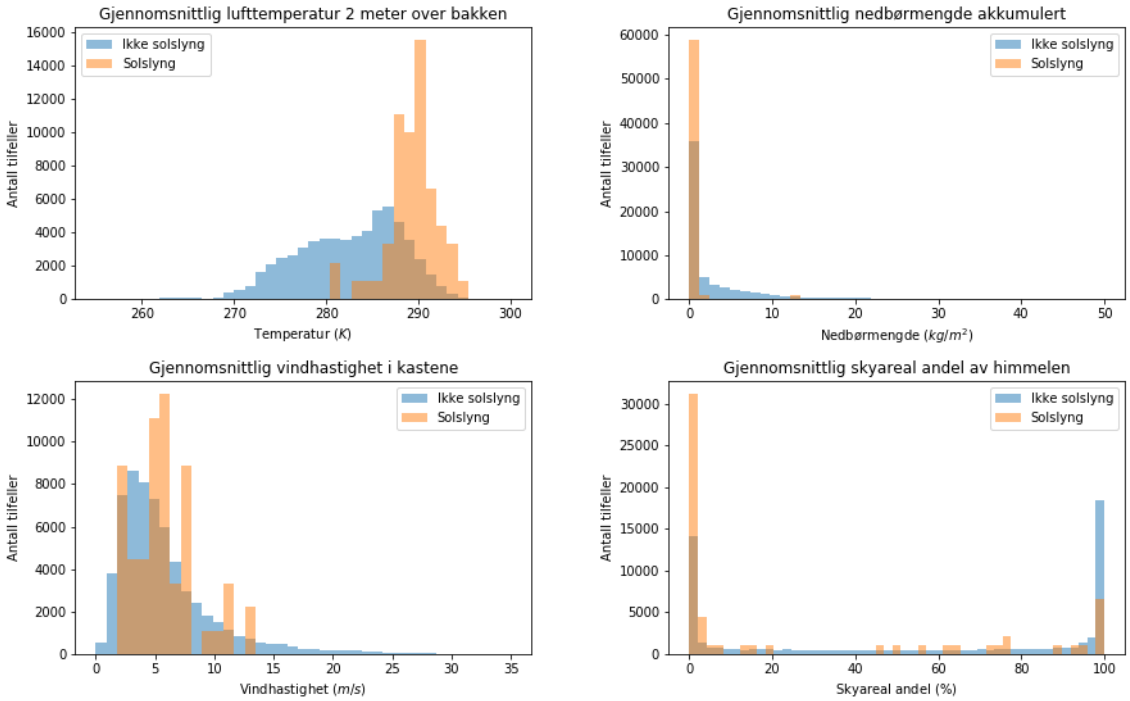
Figur 6.2: Forskjeller i fordelingene mellom skinneprofil 49E1 og 54E3. Skinneprofil 54E3 er en nyere type med høyre bøyemotstand. Det er derfor som forventet at man kan se en liten overvekt solslyngtilfeller der hele strekningen har profil 49E1 i forhold til 54E3. Søylene i histogrammet har bredde lik 0.05. NB: Solslyngtilfellene er multiplisert opp slik at antallet tilfeller i de to fordelingene blir like. Laget med Matplotlib (Hunter, 2007)



Figur 6.3: Her kan man tydelig se at det mest sannsynlig er en forskjell mellom fordelingene. NB: Solslyngtilfellene er multiplisert opp slik at antallet tilfeller i de to fordelingene blir like. Laget med Matplotlib (sst.)



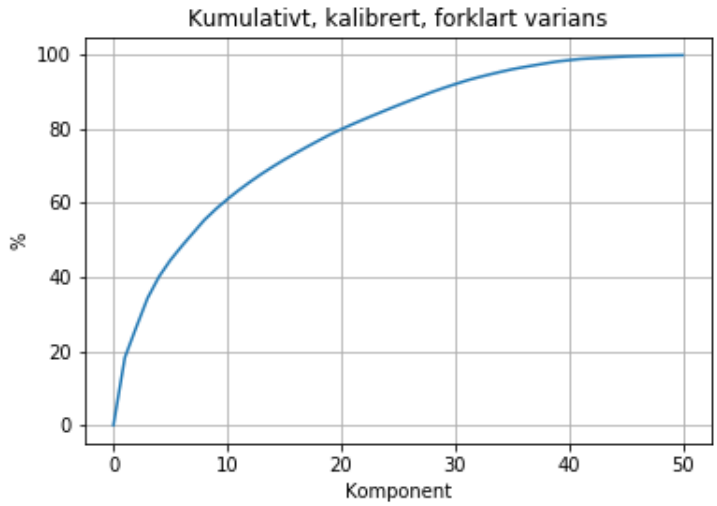
Figur 6.4: Noe forskjell, men ikke mye når det gjelder kurvatur på jernbanen. NB: Solslyngtilfellene er multiplisert opp slik at antallet tilfeller i de to fordelingene blir like. Laget med Matplotlib (sst.)



Figur 6.5: Fordelingene for luft-temperatur minner litt om fordelingen av solinnstråling. Det er også ut til å være forskjeller i fordelingene for nedbørmengde og skyareal. NB: Solslyngtilfellene er multiplisert opp slik at antallet tilfeller i de to fordelingene blir like. Laget med Matplotlib (sst.)

6.2 Prinsipalkomponentanalyse

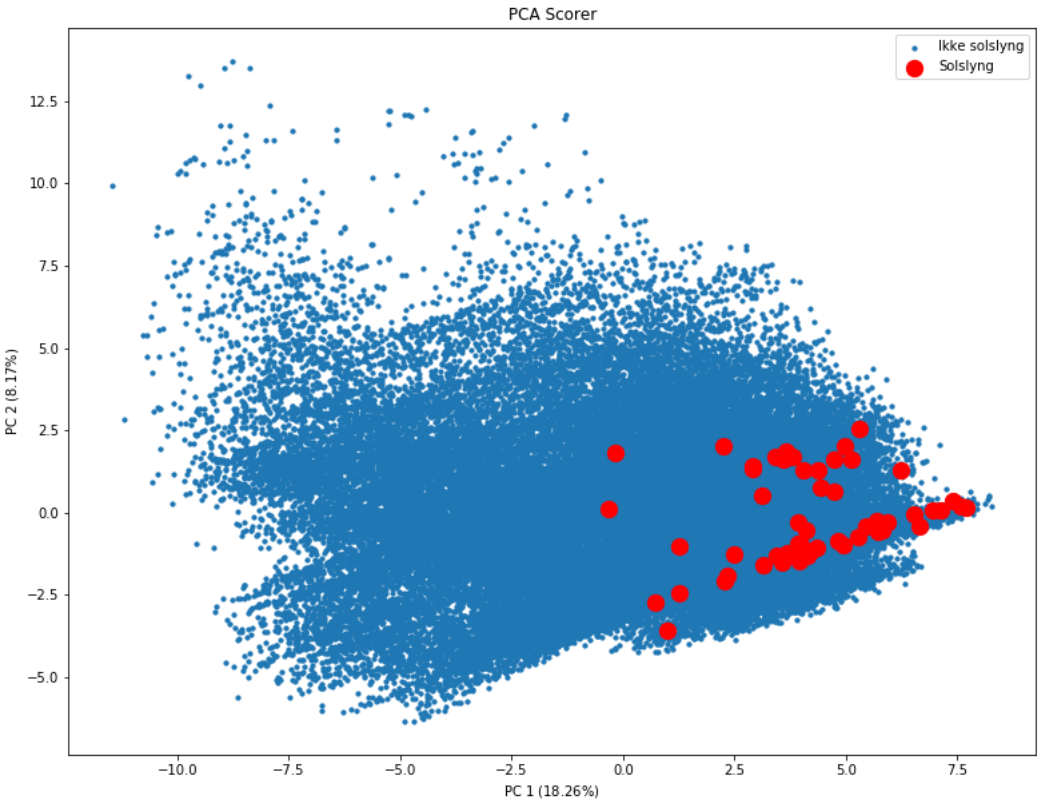
De følgende figurene viser de viktigste resultatene fra prinsipalkomponentanalysen (videre PCA). Det har blitt valgt å bare vise de fire første komponentene, da det her er mulig å se en forskjell i varians for solslyngtilfellene og ikke-solslyngtilfellene. For komponent 4 og utover har de to gruppene av tilfeller i stor grad lik utstrekning i score-plottene. De to første komponentene forklarer ca 26.4 % av variansen i datasettet, mens de første fire komponentene forklarer ca 40.1 % av variansen, jfr figur 6.6.



Figur 6.6: Kumulativt forklart varians. Laget med Matplotlib (Hunter, 2007)

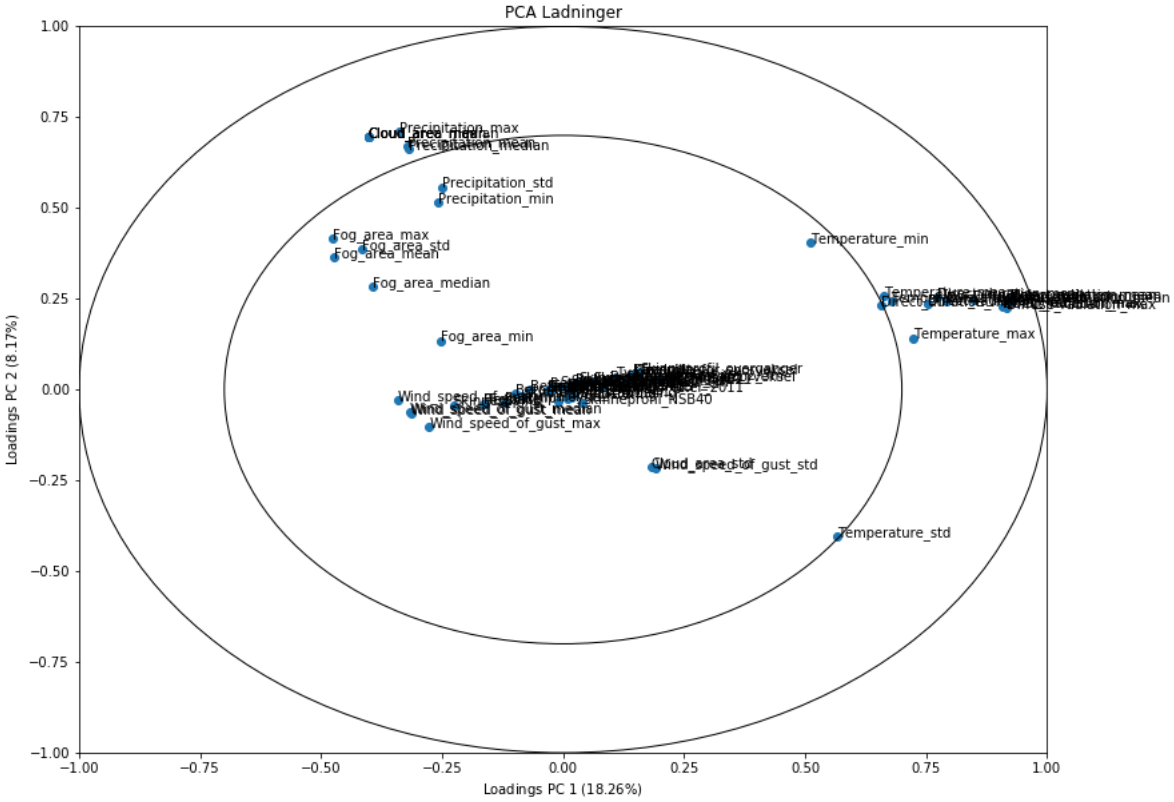
Ut fra figur 6.6 skal det omlag 15 komponenter til for å forklare 70 % av variansen. PCA beskriver altså ikke veldig mye av variansen i datasettet med de første komponentene.

I figur 6.7 vises score-plottet for første og andre komponent, blå punkter for ikke-solslyngtilfeller og røde sirkler for solslyng-tilfeller. Man kan klart se at PCAen har klart å samle alle solslyngtilfellene i den nedre og venstre delen av plottet, men at det er full overlapp mellom begge gruppene av tilfeller. solslyngtilfellene har ikke blitt separert fra resten av tilfellene.

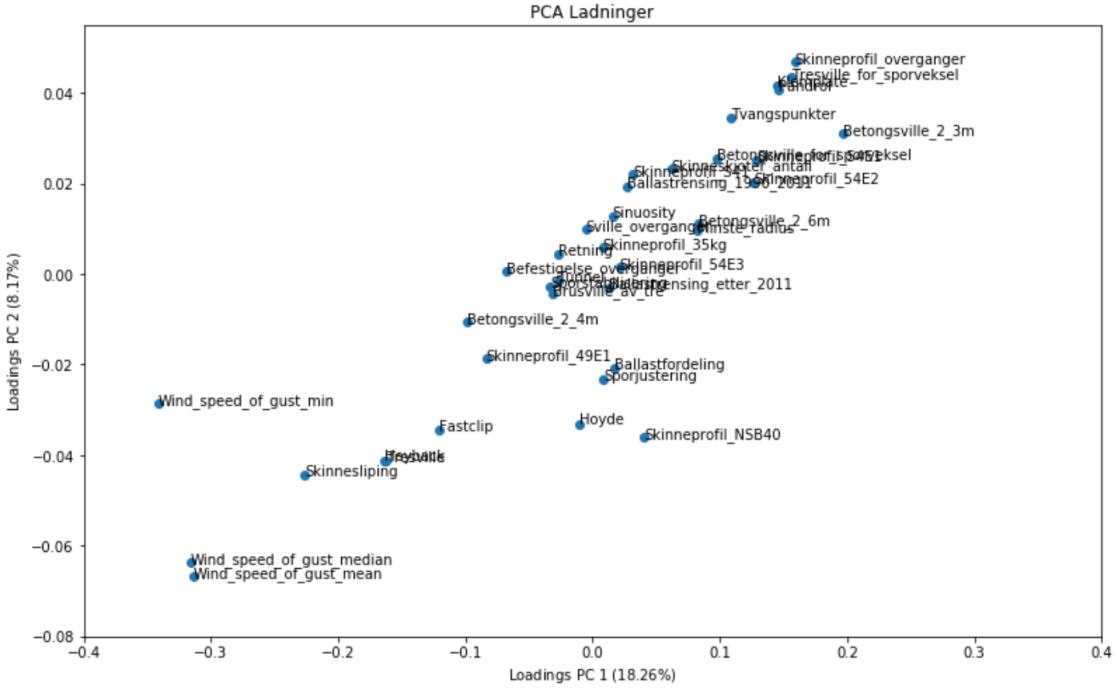


Figur 6.7: Score-plott med første og andre prinsipalkomponent. Laget med Matplotlib (Hunter, 2007)

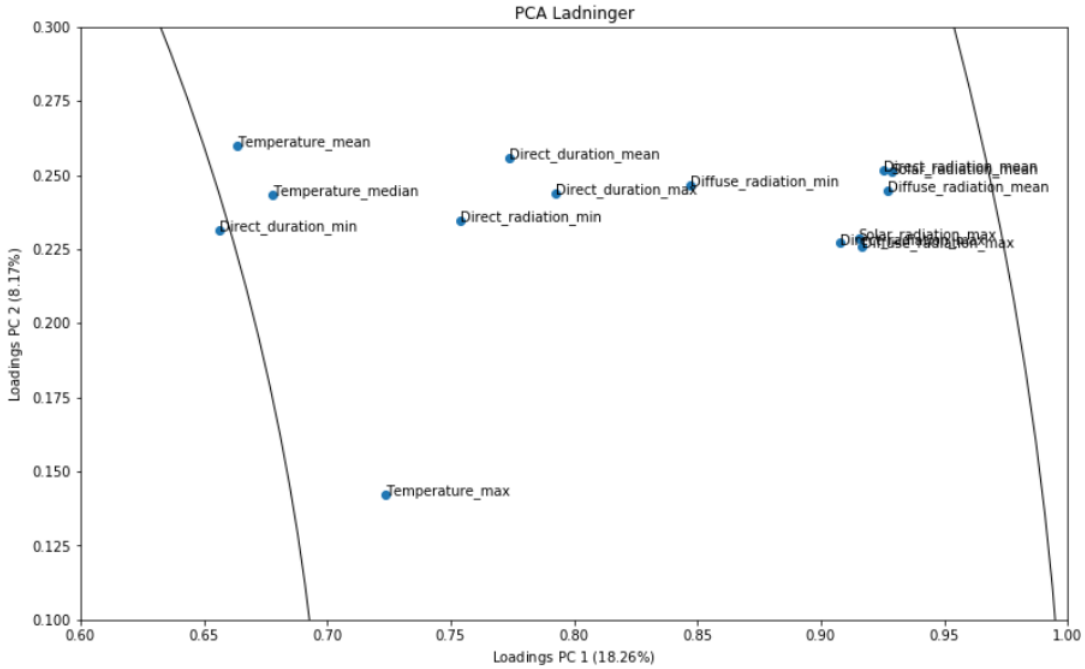
Figur 6.8 viser ladningsplottet med alle variablene med forstørrede utsnitt av de tette variabel-klyngene i figur 6.9 og 6.10 for at det skal gå an å se hvilke variabler som ligger der. Det er helt klart at variasjonen i værdataene og solinnstrålingsdataene blir mest forklart av de to første komponentene, mens banedataene og geometri-dataene ikke blir forklart i noen stor grad.



Figur 6.8: Ladnings-plott med første og andre prinsipalkomponent. Laget med Matplotlib (Hunter, 2007)



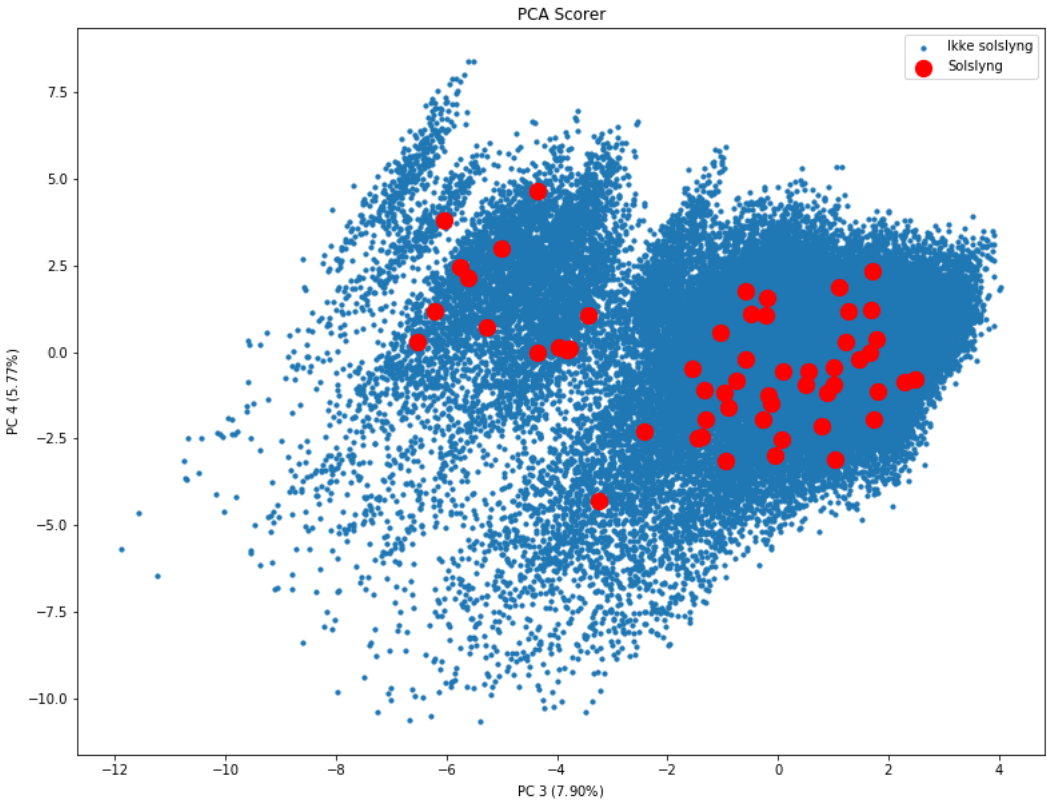
Figur 6.9: Forstørret utsnitt av ladningsplottet i figur 6.8. Laget med Matplotlib (sst.)



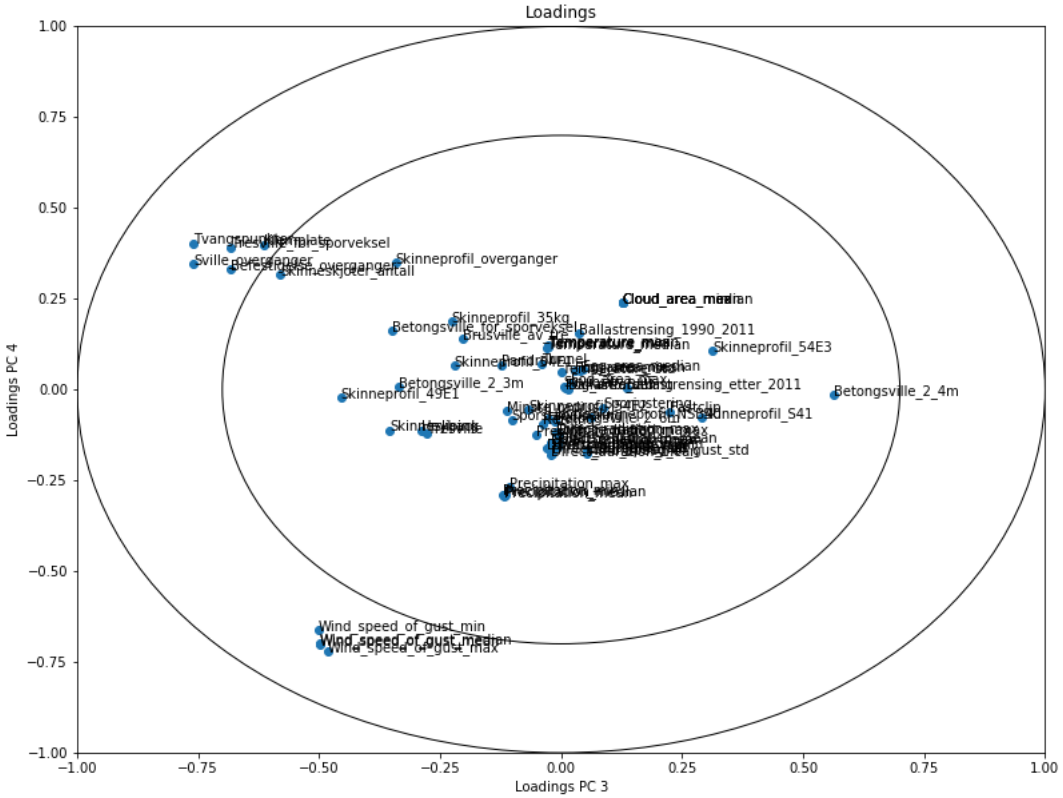
Figur 6.10: Forstørret utsnitt av ladningsplottet i figur 6.8. Laget med Matplotlib (sst.)

Sammenlignes score-plottet med ladnings-plottet tyder det på at temperatur-dataene og solinnstrålings-dataene dominerer hos solslyngtilfellene. Det kan også påpekes at standardavvikene til temperaturene ”Temperature_ std” ligger litt for seg selv mot det nedre, høyre hjørne hvor det også er mye solslyng-tilfeller i det tilsvarende score-plottet.

I figur 6.11 og 6.12 vises score-plottet og ladningsplottet til 3. og 4. prinsipalkomponent. Det er det verdt å påpeke at vind-dataene, ”Wind_ speed..osv” i ladningsplottet ligger ca på motsatt side av solslyngtilfellene i score-plottet. Utenom vind-dataene er det mest enkelte av banedata-variablene som blir forklart mest av tredje og fjerde komponent.



Figur 6.11: Score-plott med tredje og fjerde prinsipalkomponent. Laget med Matplotlib (Hunter, 2007)



Figur 6.12: Ladnings-plott med tredje og fjerde prinsipalkomponent. Laget med Matplotlib (sst.)

6.3 Resultater fra maskinlæringen

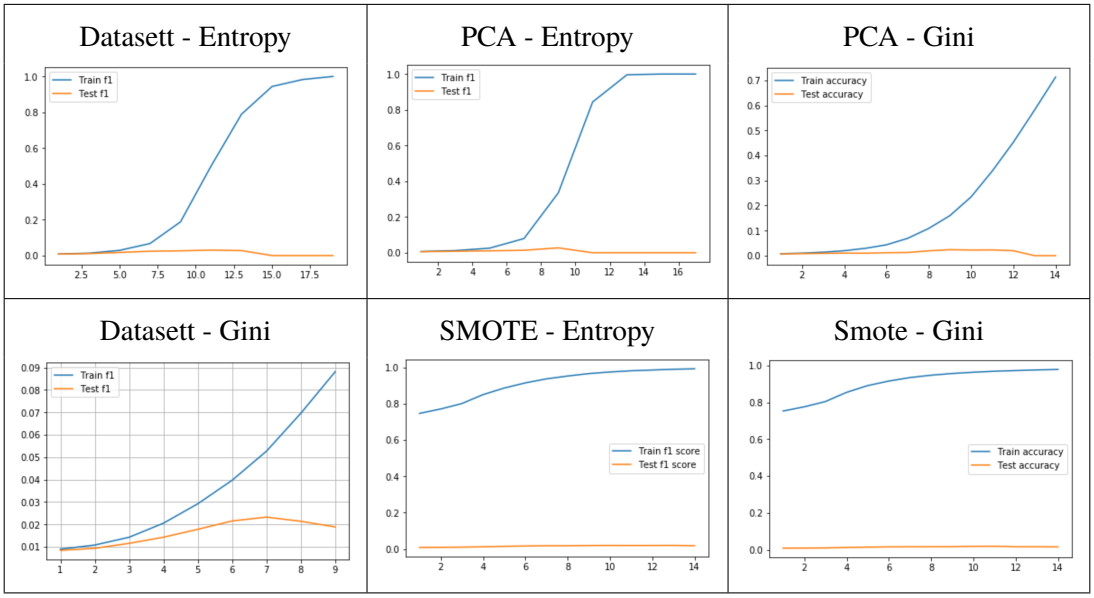
De neste avsnittene presenterer resultatene fra de forskjellige modellene.

6.3.1 Random Forest

Tabell 6.1 og 6.5 viser resultatene fra Random Forest. Alle modellene er kjørt med 200 beslutningstrær, da antallet trær ikke så ut til å ha noen særlig innvirkning, så lenge de var flere enn rundt 100 stykker.

Utvalg	Kriterium	Maks dybde	F1-score trening	F1-score test
Datasett	Entropy	11	0.499	0.030
PCA	Entropy	9	0.335	0.027
PCA	Gini	9	0.160	0.024
Datasett	Gini	7	0.053	0.023
SMOTE	Entropy	13	0.990	0.019
SMOTE	Gini	11	0.969	0.019

Tabell 6.1: F1-scorer for Random Forest, alle modeller kjørt med 200 beslutningstrær.



Tabell 6.2: Plott av trenings- og testscorer for Random Forest. Blå linjer viser treningsscorene, mens oransje linjer viser testscorene. X-aksen er antall epoker og y-aksen er F1-score.

6.3.2 Logistisk regresjon

Test F1-scorene fra logistisk regresjon var så lave at det ble prioritert å bruke mer resurser på å teste modeller av de andre maskinlæringsalgoritmene.

Utvalg	Regularisering	Solver	C	F1-score trening	F1-score test
SMOTE	L2	Liblinear	1000	0.902	0.014
Fullt datasett	L2	Liblinear	1000	0.023	0.013
SMOTE	L2	Saga	1	0.847	0.012
Fullt datasett	L1	Saga	1086	0.011	0.009
Fullt datasett	L2	Saga	15	0.012	0.009

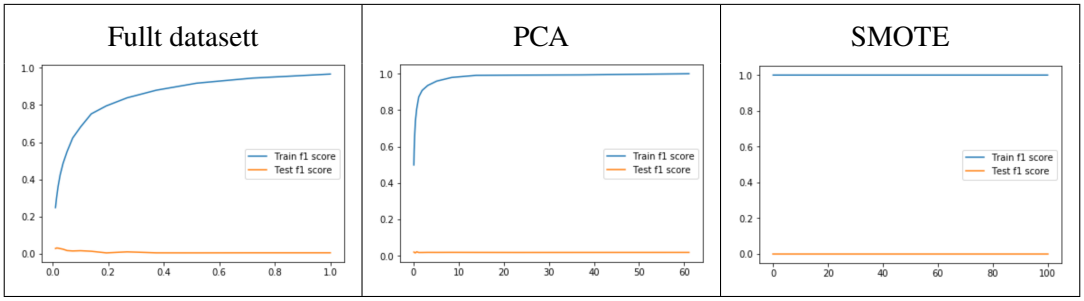
Tabell 6.3: F1-scorer for logistisk regresjon, der C er regulariseringsparameteren.

6.3.3 Support Vector Classification (SVC)

Alle SVC-modeller ble kjørt med radiell basis-funksjon som kjerne. Penalty-parameteren C er ikke oppgitt for modellen med resampling med SMOTE, da treningsscoren endte opp på 1 og test-scoren på 0 for alle verdier av C.

Utvalg	C	F1-score trening	F1-score test
Fullt datasett	0.01	0.301	0.031
PCA	0.7	0.514	0.022
SMOTE	ingen betydning	1.000	0.000

Tabell 6.4: F1-scorer for SVC, der C er regulariseringsparameteren.



Tabell 6.5: Plott av trenings- og testscorer for SVC. Blå linjer viser treningsscorene, mens oransje linjer viser testscorene. X-aksen er regulariseringsparameteren C og y-aksen er F1-score.

6.3.4 Isolation forest

Isolation Forest nådde en F1-score på ca 0.003 på PCA og 0.002 på fullt datasett.

Utvalg	Antall trær	F1-score
PCA	1	0.003
Fullt datasett	100	0.002

Tabell 6.6: F1-scorer for Isolation Forest.

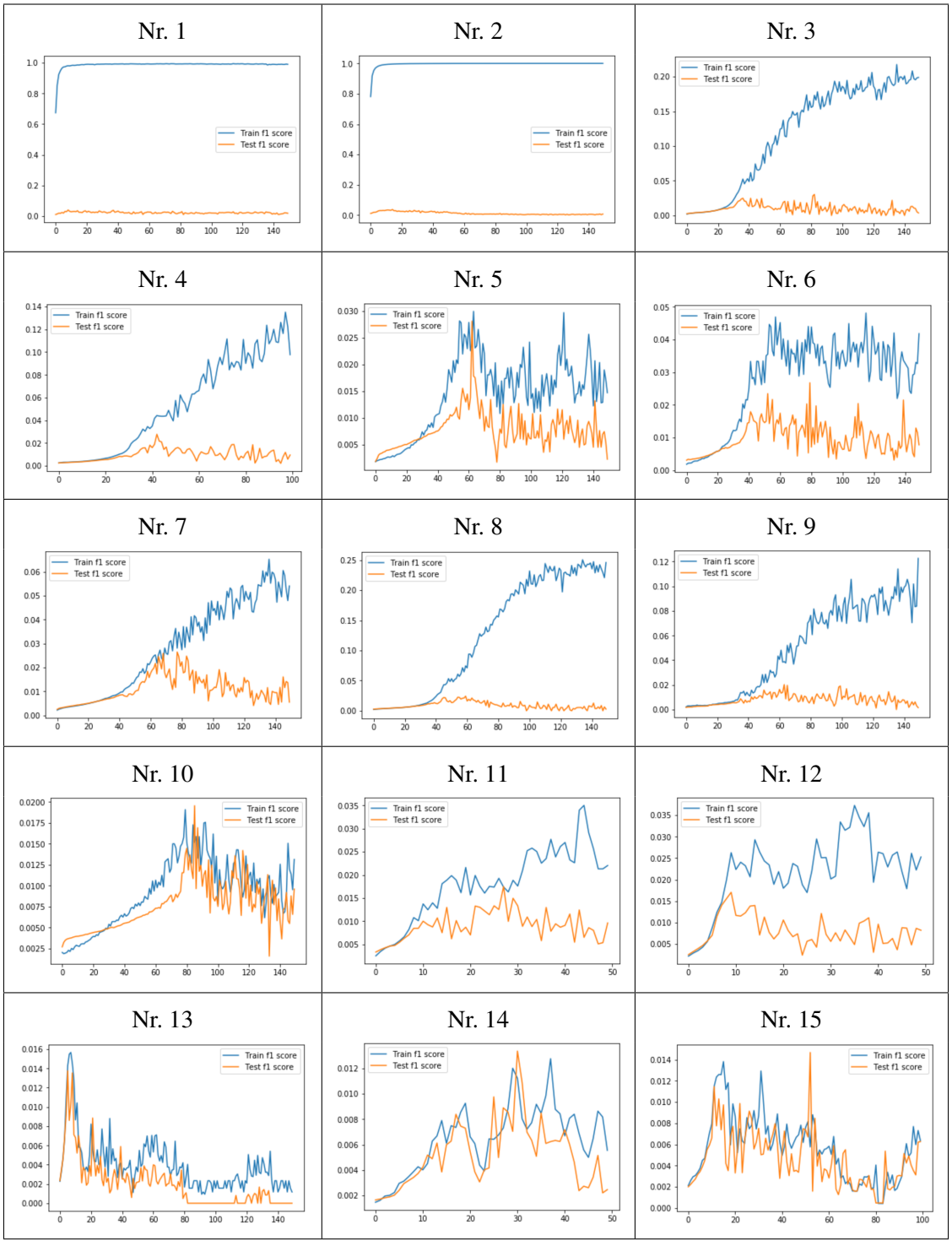
6.3.5 Multi-Layer Neural Network

Resultatene fra MLNN-modellene, sortert og nummerert etter F1 test-scoren, vises i tabell 6.7. På modellene med resampling måtte læringsraten justeres ned for at modellene i det hele tatt fikk en test F1-score som ikke sto på stedet hvil eller kun var lik 0. Tabell 6.8 viser trenings- og testscorene i forhold til antall epoker kjørt for alle MLNN-modellene.

For modell nr 15 er beste test F1-scoren tatt fra området antall epoker mindre enn 40, da den egentlig høyeste F1-scoren ser ut til å kun være et tilfeldig avvik når man sammenligner med F1-scorene ved siden av. Dette kan sees i tabell 6.8.

Nr	Utvalg	Læringsrate	Antall lag	Antall noder	Antall dropout	Antall epoker	F1-score trening	F1-score test
1	SMOTE	0.0001	3	1000	3	9	0.981	0.040
2	SMOTE	0.00001	2	1000		15	0.996	0.038
3	Datasett	0.001	2	1000		83	0.173	0.030
4	Datasett	0.001	3	1000		42	0.042	0.028
5	Datasett	0.001	3	1000	3	63	0.024	0.028
6	Datasett	0.001	2	1000	2	80	0.036	0.027
7	PCA	0.001	2	1000		78	0.028	0.026
8	Datasett	0.001	1	1000		60	0.075	0.024
9	Datasett	0.001	3	100		64	0.039	0.020
10	PCA	0.001	3	1000		86	0.015	0.020
11	Datasett	0.01	1	10		28	0.017	0.018
12	Datasett	0.01	1	100		9	0.020	0.017
13	Datasett	0.01	2	100		6	0.014	0.014
14	Datasett	0.01	3	10		31	0.011	0.013
15	Datasett	0.01	2	10		12	0.011	0.011

Tabell 6.7: F1-scorer for MLNN. For modeller med dropout-lag er antallet skjulte lag i modellen ”Antall lag” + ”Antall dropout”.



Tabell 6.8: Plott av F1 trenings- og testscorer for MLNN. Blå linjer viser treningsscorene, mens oransje linjer viser testscorene. X-aksen er antall epoker og y-aksen er F1-score.

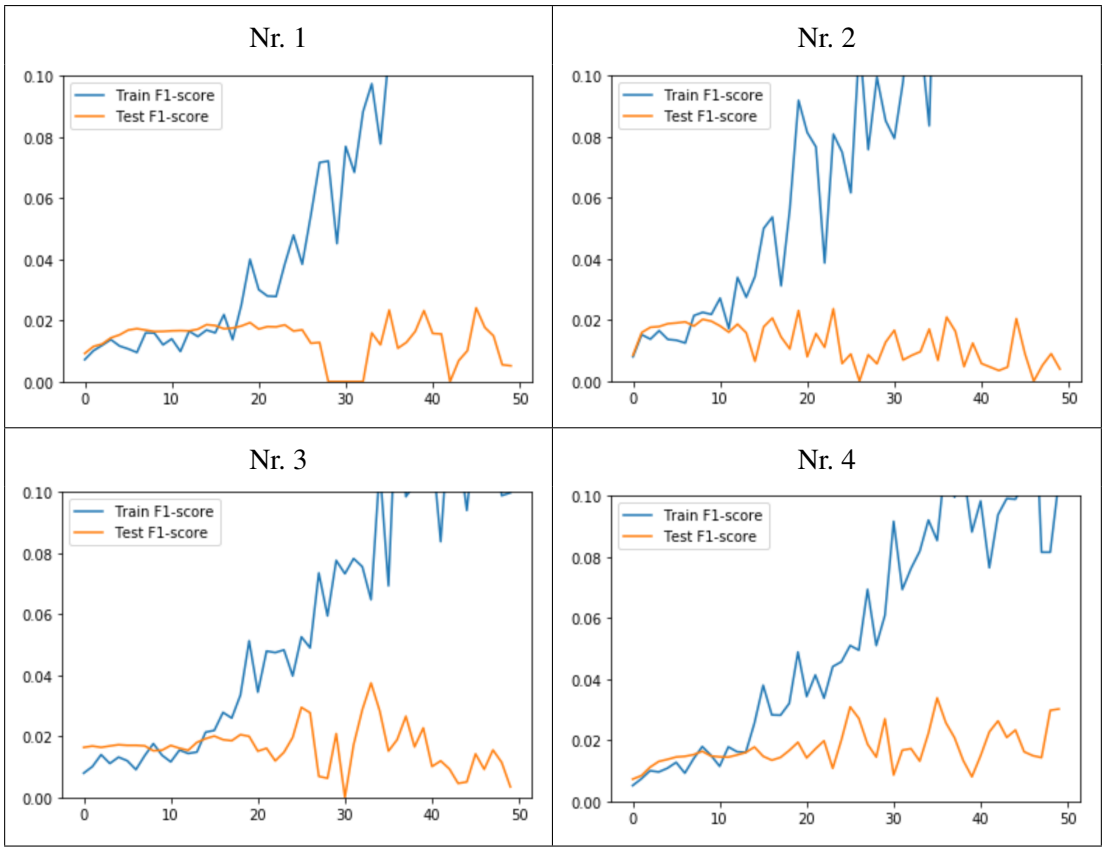
6.3.6 Recurrent Neural Network (RNN)

De følgende figurer viserer resultatene fra de 4 RNN-modellene som ble prøvd ut. De ble prøvd ut hver for seg etter hverandre, ved at parameterne spesifisert på modell 2 er valgt på bakgrunn av resultatene fra modell, parameterne på modell 3 er valgt på bakgrunn av resultatene fra modell 2 osv.

Alle RNN-modellene ble kjørt på et redusert datasett, der alle tilfellene på km-segmenter hvor det ikke har forekommet solslyng innenfor oppgavens tidsavgrensning var tatt ut. Det reduserte datasettet besto da av data for 38 km-segmenter i stedet for 167 km-segmenter.

Modell	Læringsrate	Antall lag	Antall noder	Dropout	Antall epoker	F1-score trening	F1-score test
Modell 3	0.001	1	16	0.2	34	0.065	0.037
Modell 4	0.001	1	8	0.2	36	0.085	0.034
Modell 2	0.001	1	32	0.2	24	0.081	0.024
Modell 1	0.001	1	32		46	0.155	0.024

Tabell 6.9: F1-scorer for RNN.

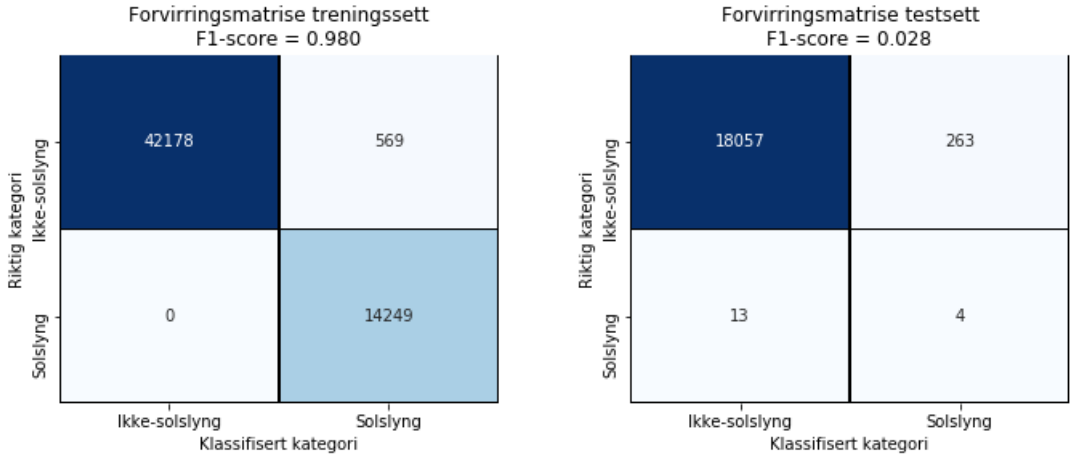


Tabell 6.10: Plott av F1 trenings- og testscorer for RNN. Blå linjer viser treningsscorene, mens oransje linjer viser testscorene. X-aksen er antall epoker og y-aksen er F1-score. Samme x- og y-verdier vises på alle plottene, slik at det blir lettere å sammenligne dem.

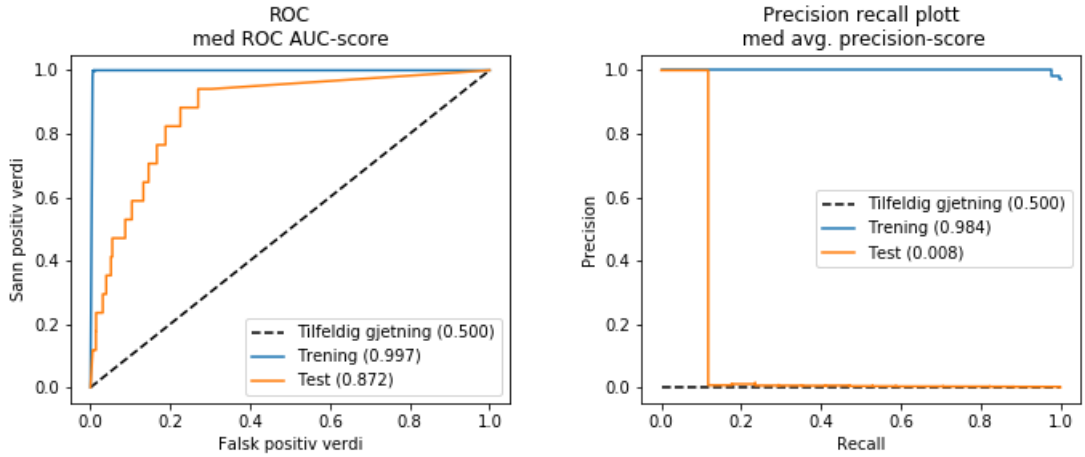
Gjennomsnittlig kjøretid (running time) på disse modellene var på rundt 90 minutter.

6.3.7 Evaluering i forhold til praktisk nytte

Figur 6.13 og 6.14 presenterer resultatene fra en trening og prediksjon av den modellen som scoret høyest på test F1-score, med en F1-score på 0.040. Dette var en MLNN-modell med 3 dense-lag med 1000 noder i hvert lag i tillegg til 3 dropout-lag. Treningssettet ble i tillegg resamplet med SMOTE og treningen ble kjørt i 9 epoker. F1-scoren på testsettet er på 0.028 og avviker fra høyeste F1-score oppnådd. Det er på grunn av naturlig variasjon som følge av bland annet tilfeldige startvekter.



Figur 6.13: Forvirringsmatrise for trenings- og testsettet for modellen som scoret høyest på test F1-score. NB: Klassefordelingen i treningssettet er 1:3, mens i testsettet er den på 55:61067. Laget med Seaborn (Waskom mfl., 2018)

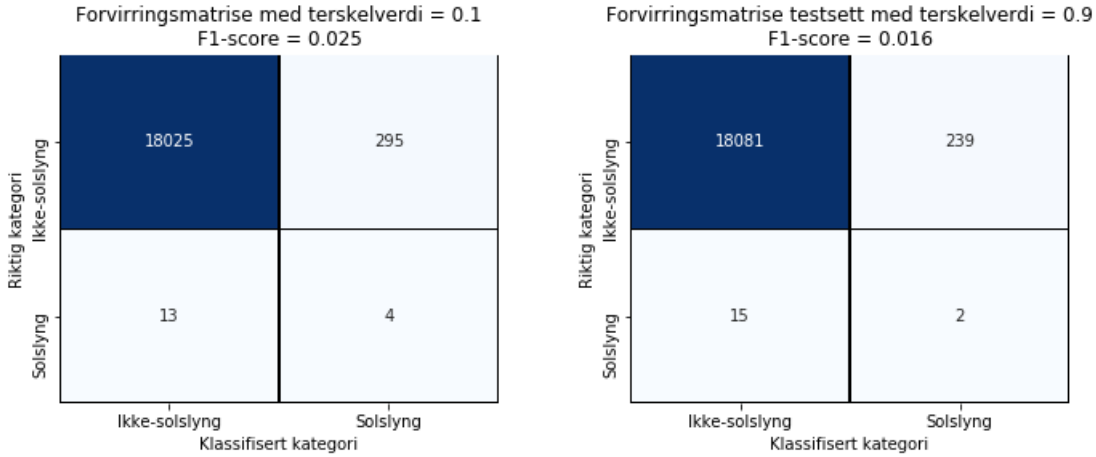


Figur 6.14: ROC og precision recall plott for trenings- og test-score. Laget med Matplotlib (Hunter, 2007)

Figur 6.14 viser ROC og precision recall-kurver for den samme modellen.

Det ble også prøvd ut ulike terskel-verdier for å justere følsomheten for solslyng-tilfeller i klassifiseringen. Alle tilfeller som blir predikert med sannsynlighet over terskel-verdien blir da

klassifisert som solslyng. Figur 6.15 viser forvirringsmatriser til testsettet med to ulike terskel-verdier på henholdsvis 0.1 og 0.9.



Figur 6.15: Forvirringsmatriser for to ulike terskelverdier. Laget med Seaborn (sst.)

Kapittel 7

Diskusjon

7.1 Eksplorativ analyse

Det ble funnet noen forskjeller i fordelingene for solslyng-tilfeller og ikke-solslyng-tilfeller. Forskjellene var i stor grad ventet, som for eksempel at luft-temperaturen og mengde solinnstråling har innvirkning på risikoen for solslyng. Ellers var det full overlapp mellom de to fordelingene, noe som også gjaldt for prinsipalkomponentene fra prinsipalkomponentanalysen. Det ser ikke ut til at noen variabler klarer å skille ut solslyngtilfellene fra resten av tilfellene.

7.2 Maskinlæring

Det var tydelig at F1-scoren var et bedre prestasjonsmål på oppgavens datasett enn ROC AUC-score og nøyaktighet (accuracy). Dette var på grunn av at datasettet er svært ubalansert med et klasseforhold på 55:61067. Precision-recall-kurven illustrerte og resultatene mer ”intuitivt” i forhold til hvor gode resultatene er i forhold til praktisk nytte enn det receiver operation-kurven (ROC) gjorde.

Aller lavest F1-score fikk Isolation Forest-modellene. Dette kan sees i sammenheng med resultatene fra prinsipalkomponentanalysen, der det viste seg å være fullstendig overlapp mellom solslyngtilfellene og ikke-solslyngtilfellene. Histogrammene som viste de to fordelingene solslyng og ikke-solslyng mot hverandre, bekrefter også at det er full overlapp. Isolation Forest er en unsupervised algoritme, som baserer prediksjonen på at uteliggerne faktisk *er* uteliggere. I oppgavens datasett ser det derfor ut til at

solslyngtilfellene ikke er uteliggere.

Nest dårligste F1-scorer var det logistisk regresjon som oppnådde. Logistisk regresjon fungerer som kjent godt på lineært separable datasett. At datasettet er lineært separabelt krever at det ikke er overlapp mellom de to klassene (solslyng / ikke-solslyng). Ut fra dette er det kanskje som forventet at logistisk regresjon ikke gjorde det like bra som algoritmene som kan håndtere ikke-lineært separable datasett bedre.

Random Forest fikk omtrent samme test F1-scorer som Support Vector Classification (SVC). F1-scorene lå i områdedet fra ca 0.02 til 0.03. Det er ikke store forskjeller på dette i forhold til resultatene fra logistisk regresjon, men dette kan tyde på at å bruke algoritmer som ikke nødvendigvis trenger å dele opp datasettet lineært, kan gi noe bedre resultater.

Både SVC og Random Forest oppnådde stor overtilpasning. For SVC ble det best resultat for små verdier av penalty parameteren C. Dette betyr at bredere og mer generaliserende beslutningsgrenser ga bedre test-score og tyder på at dette bidro til å redusere overtilpasning.

Resultatene fra Multi-Layer Neural Network (videre MLNN) kan tyde på at de mest kompliserte modellene, med 1000 noder og 2-3 skjulte lag i tillegg til eventuelle dropout-lag gjorde det best. Resampling med Smote på disse mest komplekse modellene ga best test F1-score, men veldig stor overtilpasning. Det er mulig å se at for de enklere modellene, modellene 10 til 15 ble det mer undertilpasning, mens det for de mer avanserte modellene ble det overtilpasning. Unntaket er modell 8 og 9, som også ble undertilpasset. Disse hadde også med dropout-lag, noe som tyder på at dropout-lagene bidro til å redusere overtilpasningen.

Det var ikke store forskjeller i test F1-score på Recurrent Neural Network-modellene. Det er likevel mulig å se at overtilpasningen er noe redusert i modell fra modell 1 til modell 4, ved at grafen for treningsscorene går noe lavere. Det at modell 3 og 4 oppnådde høyere maks F1 test-score i forhold til modell 1 og 2 ser ut til å komme av større varians i scorene.

7.3 Evaluering av praktisk nytte

Den høyeste F1-scoren på testsett som ble oppnådd ble på 0.04 med en Multi-Layer Neural Network-modell (videre MLNN) med resampling (SMOTE).

Modellen har stor overtilpasning, som forvirringsmatrisene i figur 6.13 viser. Den klarte å finne alle solslyngtilfellene i treningssettet, og kun feilklassifisere en liten del av ikke-solslyngtilfellene som solslyng. Derimot klarte den kun å finne 4 av 17 av solslyngtilfellene i testesettet og feilklassifisere 263 ikke-solslyng-tilfeller som solslyng. Her er det nyttig å påpeke at siden modellen ble trent med resampling, er klassefordelingen i treningssettet på 1:3, mens i testsettet er den på 55:61067. F1-scoren på testsettet på 0.028 avviker noe fra høyeste F1-score som ble oppnådd. Dette avviket kommer av naturlig variasjon som følge av blant annet tilfeldige startvekter og tilfeldig oppdeling av datasettet i trenings- og testsett. Forskjellene mellom 0.040 og 0.028 er likevel så liten at den ikke vil ha noen stor praktisk betydning.

I figur 6.14 kan man se at precision recall-kurven gir et mer intuitivt riktig bilde av resultatet, ved at test-settet vises med så lav score som den gjør. I plottet med ROC ser grafen for test-settet ikke riktig så ille ut, selv om vi vet at bare 4 av 17 solslyng-tilfeller ble klassifisert riktig, og at 263 tilfeller ble feilaktig klassifisert som solslyng.

I dette tilfellet gjorde det ikke prediksjonen for vårt formål noe særlig bedre å justere på terskel-verdiene. Den fant ingen flere solslyngtilfeller ved å sette terskelverdien ned til 0.1. Kun 2 av 17 solslyng ble funnet ved en terskelverdi på 0.9, mens antallet tilfeller som feilaktig ble klassifisert som solslyng kun sank med 24 tilfeller. Dette forteller dermed at ingen av oppgavens modeller vil være gode nok til å kunne brukes i praksis til å predikere solslyng.

Kapittel 8

Konklusjoner

På bakgrunn av diskusjonen i forrige kapittel kan det konkluderes med at oppgaven ikke har lyktes med å predikere solslynghendelser på jernbanen så godt at modellene kan brukes i praksis. Antakelig skyldes dette i stor grad at ingen av variablene klarte å skille solslyngtilfellene fra ikke-solslyngtilfellene. Det var full overlapp mellom de to fordelingene.

Oppgaven viste seg også å bli en liten studie av fornuftige prestasjonsmål for ubalanserte datasett. Det lyktes å finne et prestasjonsmål, F1-score, og en grafisk visning, precision-recall-kurve, som nyanserte resultatet på en god og intuitivt realistisk måte i forhold til hvor gode modellene var i forhold til praktisk nytte.

Oppgaven har kun dreid seg om å predikere solslyng på en liten del av jernbanen og over kun to sommerhalvår. Oppgaven utelukker derfor ikke at ikke går an å predikere solslyng ved hjelp av maskinlæring om det skaffes et større datagrunnlag. Oppgavens datasett er også bare én måte av mange mulige måter det går an å designe og organisere variablene på, da en kontinuerlig jernbanestrekning ikke har like naturlige oppdelinger i tilfeller, som for eksempel ulike hunder. Det er derfor store valgmuligheter i hvordan egenskaper ved jernbanen kan beskrives. I oppgaven er det dessuten kun testet ut noen utvalgte maskinlæringsalgoritmer på et begrenset utvalg av ulike parametere. Det er dermed mulig at det kan gå an å komme fram til bedre resultater om dette settes inn flere ressurser til å prøve flere ulike modeller og flere ulike kombinasjoner av parametere. På denne måten er problemstillingen om det går an å bruke maskinlæring til å predikere solslyng ikke enda fullt og helt besvart. Oppgaven

må sees på som et første forsøk på å få dette til og som eventuelt kan bane vei for videre utforsking og testing av problemstillingen.

Kapittel 9

Videre anbefalinger

Mot slutten av arbeidet med oppgaven har det dukket opp flere ulike momenter som det kan settes spørsmålstegn ved om oppgaven ble utført på den beste måten. Hensikten med dette kapittelet blir derfor å presentere noen refleksjoner om hva som kunne vært gjort annerledes og komme med noen anbefalinger i forhold til videre arbeid med temaet.

Da variablene i oppgaven viste seg å ikke skille solslyngtilfellene fra ikke-solslyngtilfellene, at det var full overlapp mellom disse to fordelingene, kan det være en god idé å finne nye måter å utforme variablene på. For eksempel er det ingen av variablene som forteller *hvor* innad på et km-segment noe befinner seg. Det være seg temperatur, solinnstrålingen, skinneprofil, sviller, tvangspunkter, skarpe kurver osv. Alle variablene er designet slik at de sier noe om hvor mye forekomst det er på hvert km-segment, men ikke akkurat *hvor* de ligger. For eksempel kan ett km-segment ha en skarp kurve ett sted, mye solinnstråling et annet sted og en eldre skinneprofil et tredje sted. Et annet km-segment kan ha en skarp kurve, mye solinnstråling og en eldre skinneprofil på akkurat samme sted, noe som ville økt risikoen for solslyng betraktelig, da alle risikofaktorene forekommer på samme sted. Slik datasettet er utformet i denne oppgaven, blir disse to km-segmentene representert helt likt, selv om risikoen for solslyng er forskjellig. Det å ta med mer informasjon om hvor ting ligger i forhold til hverandre innad på et km-segment tror jeg derfor vil kunne gi bedre resultater i maskinlæringen.

En anbefaling fra Alf Helge Løhren i Bane NOR har vært å ta med en variabel med skjæring og fylling langs jernbanen. Dette ble det dessverre ikke tid til å få til i denne oppgaven. Skjæring vil si et sted der

jernbanetraséen er skåret ut i terrenget, for eksempel gjennom en fjellknaus eller i en bratt skråning. Fylling vil si at det har blitt fylt på i terrenget, slik at jernbanetraséen går høyere enn opprinnelig bakkenivå.

Andre variabler som kan være nyttige å ha med, som ikke ble med i denne oppgaven er en binær variabel med 0 for enkeltsporede og 1 for dobbeltsporede strekninger. En variabel med km-segment-nummer og en med dag-nummer kunne også vært interessant å teste ut, da dimensjonene i tid og rom blir representert. Temperaturene om vinteren har påvirkning på om og hvor mye skinnene kan ”trekke seg sammen” som følge av lave temperaturer. Skinner som har trukket seg sammen vil ha lavere nøytral-temperatur og dermed få større risiko for solslyng. Antakelig vil da en variabel med ”vintertemperaturer” hjelpe i predikeringen.

Værdataene som er brukt i oppgaven har en oppløsning på 2.5 km. Å bruke værdata med høyere oppløsning, for eksempel ved å plassere ut værstasjoner langs jernbanen kan tenkes å bidra til bedre resultater fra maskinlæringen.

Det går også an å ta med variabler som har med det rullende materiale (togene) å gjøre, for eksempel strekninger med mye oppbremsing eller akselerasjon samt vertikal stigning. Antallet brutto tonn med rullende materiell som kjører på de forskjellige strekningene kunne også vært interessant å teste ut.

I oppgaven har jernbanestrekningene blitt delt inn i 1 kilometer lange segmenter. Det går at å se om lengre eller kortere segmenter vil ha noen innvirkning på prediksjonen. Med lengre segmenter vil man få færre tilfeller å jobbe med, og datasettet vil bli mer balansert, men samtidig blir det større variasjon innad langs segmentet og prediksjonene blir mindre presise når det gjelder akkurat *hvor* det blir solslyng. Kortere segmenter vil gi et mindre balansert datasett, men til gjengjeld vil man få mer enhetlige tilfeller med mindre variasjon langs segmentet innad.

Det er mye spennende som skjer innen GIS og maskinlæring nå om dagen. Maskinlæring blir anvendt mer og mer på geografiske data og blir stadig mer tilgjengelig og lettere å bruke. Her kan det finnes spennende muligheter, blant annet for å utnytte at datasettet både har en romlig dimensjon og en tids-dimensjon.

Bibliografi

Bane NOR (2018): *Jernbane - Banenettverk - Lineære referanser*. Referanselinjer for jernbanestrekninger i Norge. Tilgjengelig fra: https://kartkatalog.geonorge.no/metadata/bane-nor-sf/jernbane-banenettverk-lineaere-referanser/dc4b671f-f354-4e1f-a23b-dd717ae48e1b. (lest 28.04.2019).

Bane NOR mfl. (2019): *Lærebøker i jernbaneteknikk*. Tilgjengelig fra: https://www.jernbanekompetanse.no/wiki/Forsidee. (lest 30.01.2019).

Chollet, François (2017): *Advanced Usage of Recurrent Neural Networks*. Tilgjengelig fra: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb. (lest 05.05.2019).

Chollet, François mfl. (2015): *Keras*. (Versjon 2.2.4). Programvare. Tilgjengelig fra: https://keras.io.

ESRI (2018a): *ArcGIS Pro*. (Versjon 2.3.0) Programvare. Tilgjengelig fra: https://pro.arcgis.com/en/pro-app/. (lest 03.05.2019).

— (2018b): *How solar radiation is calculated*. Tilgjengelig fra: https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-solar-radiation-is-calculated.htm. (lest 28.04.2019).

Google (2019): *Google Colaboratory*. Programvare. Tilgjengelig fra: https://colab.research.google.com.

Haltuf, Michal (2018): *Best loss function for F1-score metric*. Tilgjengelig fra: https://www.kaggle.com/rejpalcz/best-loss-function-for-f1-score-metric. (lest 03.05.2019).

Hunter, J. D. (2007): "Matplotlib: A 2D graphics environment". I: *Computing In Science & Engineering* 9.3, s. 90–95. DOI: 10.1109/MCSE.2007.55. (Versjon 3.0.3). Programvare.

Kartverket (2019): *Høydedata*. Tjeneste for å laste ned åpne høydedata. Tilgjengelig fra: https://hoydedata.no/LaserInnsyn/. (lest 13.03.2019).

Kluyver, Thomas mfl. (2018): *Jupyter Notebook*. (Versjon 5.7.4). Programvare. Tilgjengelig fra: https://jupyter-notebook.readthedocs.io/en/stable/index.html. (lest 15.04.2019).

Køltzow, Morten (2017): *MetCoOp Ensemble Prediction System (MEPS)*. Tilgjengelig fra https://drive.google.com/file/d/0B-SaEtrDE91WWEJoNkJiUm5TNzg/view. (lastet ned 21.03.2019).

Lemaître, Guillaume, Fernando Nogueira og Christos K. Aridas (2017): "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". I: *Journal of Machine Learning Research* 18.17, s. 1–5. URL: http://jmlr.org/papers/v18/16-365.html. (Versjon 0.4.3). Programvare.

Lewinson, Eryk (2018): *Outlier Detection with Isolation Forest*. Tilgjengelig fra: https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e. (lest 04.05.2019).

Løhren, Alf Helge (2019a): *Sensitivitet/Spesifisitet solslyng*. (e-post til Elise Tegnér 27. april 2019).

— (2019b): *Status masteroppgave solslyng*. (e-post til Elise Tegnér 11. april 2019).

MathWorks (2019): *MATLAB*. (Versjon R2019a). Programvare. Tilgjengelig fra: https://se.mathworks.com/products/matlab.html.

Mevik, Bjørn-Helge (2007): *PCA for Analysis of Complex Multivariate Data*. Tilgjengelig fra: https://www.uio.no/studier/emner/matnat/math/nedlagte-emner/STK4040/h07/undervisningsmateriale/PCA-pluss_p%c3%b8lseeksempel.ppt (lest 03.05.2019).

La-ongkham, Orawan mfl. (feb. 2015): "Distinct gut microbiota of healthy children from two different geographic regions of Thailand". I: *Archives of Microbiology* 197. DOI: 10.1007/s00203-015-1089-0.

Open source (2019): *pgf – A Portable Graphic Format for TeX*. (Versjon 3.1.1) Programvare. Tilgjengelig fra: https://pgf-tikz.github.io/. (lest 03.05.2019).

Pedregosa, F. mfl. (2011): "Scikit-learn: Machine Learning in Python". I: *Journal of Machine Learning Research* 12, s. 2825–2830. (Versjon 0.20.3). Programvare.

— (2019): "Generalized Linear Models". I: (Lest 12.05.2019).

Raschka, Sebastian og Vahid Mirjalili (2017): *Python Machine Learning*. 2. utg. Birmingham: Packt Publishing.

Skogan, Pål mfl. (2018): *Solslyngrapport - Bane NOR 2018*. Teknisk avdeling (TA) ved Bane NOR.

Støylen, Eivind (2016): *Downloading subset of a NetCDF-file from thredds*. Tilgjengelig fra: `https://drive.google.com/file/d/1Mmt_l88x74wXbui1lgCVEI56egtLcW01/view`. (lest 07.05.2019).

Sørlie, Per Herman (2008): *Krefter i helsveist spor*. Tilgjengelig fra: `https://brage.bibsys.no/xmlui/bitstream/id/239444/Krefter`. (lastet ned 08.04.2019).

Tomic, Oliver (2017): *Hoggorm: Python-bibliotek for eksplorativ multivariat statistikk*. (Versjon 0.12.0). Programvare. Tilgjengelig fra: `https://hoggorm.readthedocs.io/en/latest/`. (lest 30.04.2019).

Waskom, Michael mfl. (2018): *mwaskom/seaborn: v0.9.0 (July 2018)*. DOI: `10.5281/zenodo.883859`. URL: `https://doi.org/10.5281/zenodo.883859`. (Versjon 0.9.0). Programvare.

Tabeller

Figurer

Alle figurer uten oppgitte kilder er selvillustrert.

Listings

Alle koder uten oppgitte kilder er laget selv.

Vedlegg

Vedlegg 1 - Eksempler fra datainnsamlingen

Vedlegg 2 - Kode-eksempler på maskinlæringsmodellene

Vedlegg 3 - Variablene i datasettet

Vedlegg 1 - Eksempler fra datainnsamlingen

Eksempel på bearbeiding av data fra Banedata i Excel

Eksempel på bearbeiding av data fra Banedata

Om denne filen:

Dette er en kopi av en original Excel-fil Banedata.
Excel-filen inneholder informasjon om skinneprofilen til høyre skinnestreng på høyre hovedspor på Hovedbanen.
De kolonnene som ikke har vært av betydning for oppgaven er tatt ut.
Alle endringer som har blitt gjort i forhold til original-filen fra Banedata er merket med rødt.
Ved overlapp i datasettet, det vil si at ulik informasjon er lagt inn to ganger for samme strekning, har følgende rekkefølge blitt brukt for sletting av data:
Sporveksler er 1. prioritet. Nyere data prioriteres fremfor eldre data. Det som ikke prioriteres blir slettet.

Forklaring for hver kolonne:

Beskrivelse	H-hsp H-side	Høyre hovedspor, høyre skinnestreng
	SPV-101a	Sporveksel og sporveksel-nummer. Alle sporvekslere er nummerert.
Fra og Til	Sporkilometeranvisning til strekningen informasjonen gjelder for	
Hulltest	Kolonne lagt til av meg, for å sjekke huller og overlapp. Den viser differansen mellom 'Til'-feltet i raden over og 'Fra'-feltet på samme rad	
Kommentar	Redegjørelse for endring.	
Rødt var	Original-verdi som var i stedet for verdien som er merket med rødt.	
Idriftssatt dato	Dato skinnen ble idriftssatt. Om denne datoen faller etter eller innenfor oppgavens tidsavgrensning, har det vært nødvendig å hente historiske data fra lokal tilstandskontrollør. Den historiske dataen har da blitt lagt til manuelt på slutten når det genereres data for alle datoene.	
Skinneprofil	Skinneprofil, vist med forskjellige farger for å lett skille skinneprofilene fra hverandre.	
Lengde	Kolonne lagt til av meg for å se lengden av hver strekning. Denne har blitt brukt mest for å kontrollere at sporvekslere er lagt inn med en fornuftig lengde. Dvs rundt 25 - 60 m, eller standard sporveksel-lengder fra jernbanekompetanse.no som er på henholdsvis ca 27, 33, 34, 42, 54, 65 og 95 meter. Ved tvil er lengden av sporvekslene også målt ut fra flybilder på norgeskart.no.	

Beskrivelse	Fra	Til	Hulltest	Kommentar	Rødt var	Idriftssatt dato	Skinneprofil	Lengde
Skinne, H-hsp H-side, Oslo S - Bryn	2,590	3,649	0	Tatt ut overlapp med samme profil		01.01.1981 00:00:00	49-E1	1,059
Skinne, SPV-101a, Bryn stasjon	3,649	3,682	0			01.01.1971 00:00:00	49-E1	0,033
Skinne, H-hsp H-side, Bryn stasjon	3,682	3,749	0			01.01.1981 00:00:00	49-E1	0,067
Skinne, SPV-103b, Bryn stasjon	3,749	3,776	0			01.01.1978 00:00:00	49-E1	0,027
Skinne, H-hsp H-side, Bryn stasjon	3,776	3,846	0			01.01.1981 00:00:00	49-E1	0,07
Skinne, SPV-105b, Bryn stasjon	3,846	3,879	0			01.01.1978 00:00:00	49-E1	0,033
Skinne, H-hsp H-side, Bryn stasjon	3,879	4,205	0			01.01.1978 00:00:00	49-E1	0,326
Skinne, SPV-107b, Bryn stasjon	4,205	4,238	0			01.01.1972 00:00:00	49-E1	0,033
Skinne, SPV-109a, Bryn stasjon	4,238	4,271	0			01.01.1976 00:00:00	49-E1	0,033
Skinne, SPV-111a, Bryn stasjon	4,271	4,304	0			01.01.1971 00:00:00	49-E1	0,033
Skinne, H-hsp H-side, Bryn stasjon	4,304	4,731	0			01.01.1978 00:00:00	49-E1	0,427
Skinne, H-hsp H-side, Bryn - Brobekk	4,731	5,400	0	Fjernet 4. desimal	4,7311	01.01.1980 00:00:00	49-E1	0,669
Skinne, H-hsp H-side, Brobekk stasjon	5,400	5,585	0			01.01.1931 00:00:00	49-E1	0,185
Skinne, SPV-195a, Brobekk stasjon	5,585	5,626	0			01.01.1991 00:00:00	49-E1	0,041
Skinne, H-hsp H-side, Brobekk stasjon	5,626	5,885	0			01.01.1931 00:00:00	49-E1	0,259
Skinne, H-hsp H-side, Brobekk stasjon	5,885	7,037	0			01.01.1992 00:00:00	54-E3	1,1515
Skinne, H-hsp V-side, Brobekk - Aker	7,037	7,500	0	Skinneprofil tatt fra ventre skinnestreng pga manglet på høyre	7,036	01.01.1973 00:00:00	49-E1	0,4635
Skinne, H-hsp H-side, Brobekk - Aker	7,500	7,542	0			01.01.1987 00:00:00	49-E1	0,042
Skinne, H-hsp H-side, Brobekk - Aker	7,542	7,756	0			01.01.1966 00:00:00	49-E1	0,214
Skinne, H-hsp H-side, Aker stasjon	7,756	8,324	0			01.01.1981 00:00:00	49-E1	0,568
Skinne, H-hsp H-side, Aker stasjon	8,324	8,339	0			01.01.1976 00:00:00	49-E1	0,015
Skinne, H-hsp H-side, Aker stasjon	8,339	8,670	0	Tatt ut overlapp med samme profil og stålkvalitet		01.01.1983 00:00:00	49-E1	0,331
Skinne, SPV-102b, Aker stasjon	8,670	8,703	0			01.01.2000 00:00:00	54-E1	0,033
Skinne, SPV-115b, Aker stasjon	8,703	8,736	0			01.01.2000 00:00:00	54-E1	0,033
Skinne, SPV-114a, Aker stasjon	8,736	8,769	0			01.01.2000 00:00:00	54-E1	0,033
Skinne, H-hsp H-side, Aker stasjon	8,769	8,841	0			01.01.1999 00:00:00	49-E1	0,072
Skinne, H-hsp H-side, Aker stasjon	8,841	9,056	0			01.01.1966 00:00:00	49-E1	0,215
Skinne, H.hsp, Aker - Grorud	9,056	10,113	0			01.01.1966 00:00:00	49-E1	1,057
Skinne, SPV-121b, Grorud stasjon	10,113	10,154	0			01.09.2007 00:00:00	54-E3	0,041
Skinne, SPV-123a, Grorud stasjon	10,154	10,181	0			01.09.2008 00:00:00	54-E3	0,027
Skinne, H.hsp, Grorud stasjon	10,181	10,200	0			01.01.1966 00:00:00	49-E1	0,019
Skinne, S49, Grorud stasjon	10,200	10,306	0	"Laget plass" til sporveksel	10,98	01.01.1976 00:00:00	49-E1	0,106
Skinne, SPV-127b, Grorud stasjon	10,306	10,333	0			01.09.2008 00:00:00	54-E3	0,027
Skinne, S49, Grorud stasjon	10,333	10,836	0	"Laget plass" til sporveksel	10,2 og 10,98	01.01.1976 00:00:00	49-E1	0,503
Skinne, SPV-129a, Grorud stasjon	10,836	10,877	0			17.08.2003 00:00:00	54-E3	0,041

Skinne, S49, Grorud stasjon	10,877	10,963	0	"Laget plass" til sporveksel	10,85	09.06.2011 00:00:00	49-E1	0,086
Skinne, S49, Grorud stasjon	10,963	10,974	0	"Laget plass" til sporveksel	10,2 og 10,98	01.01.1976 00:00:00	49-E1	0,011
Skinne, SPV-133b, Grorud stasjon	10,974	11,015	0			17.08.2003 00:00:00	54-E3	0,041
Skinne, S49, Grorud stasjon	11,016	12,940	0,001	Hull opp til 100 m er ok		01.01.1974 00:00:00	49-E1	1,924
Skinne, S49, Grorud - Lørenskog	12,940	13,608	0		13,62	01.01.1985 00:00:00	49-E1	0,668
Skinne, SPV-135a, Lørenskog stasjon	13,608	13,641	0			01.01.1981 00:00:00	49-E1	0,033
Skinne, S49, Lørenskog stasjon	13,641	13,748	0	"Laget plass" til sporveksel	13,62 og 13,8	01.01.1981 00:00:00	49-E1	0,107
Skinne, SPV-137b, Lørenskog stasjon	13,748	13,775	0			01.01.1981 00:00:00	49-E1	0,027
Skinne, S49, Lørenskog stasjon	13,775	13,807	0	"Laget plass" til sporveksel	13,62 og 13,8	01.01.1981 00:00:00	49-E1	0,032
Skinne, SPV-139, Lørenskog stasjon	13,807	13,840	0			01.01.1981 00:00:00	49-E1	0,033
Skinne, S49, Lørenskog stasjon	13,840	13,940	0		13,81	01.01.1975 00:00:00	49-E1	0,1
Skinne, S49, Lørenskog stasjon	13,940	14,110	0			01.01.1954 00:00:00	49-E1	0,17
Skinne, S49, Lørenskog stasjon	14,110	14,250	0			01.01.1975 00:00:00	49-E1	0,14
Skinne, S49, Lørenskog stasjon	14,250	14,305	0	"Laget plass" til sporveksel	14,37	01.01.1981 00:00:00	49-E1	0,055
Skinne, SPV-104, Lørenskog stasjon	14,305	14,332	0			01.01.1983 00:00:00	49-E1	0,027
Skinne, S49, Lørenskog stasjon	14,332	14,338	0	"Laget plass" til sporveksel	14,25 og 14,3	01.01.1981 00:00:00	49-E1	0,006
Skinne, SPV-141b, Lørenskog stasjon	14,338	14,365	0			01.01.1983 00:00:00	49-E1	0,027
Skinne, S49, Lørenskog stasjon	14,365	14,370	0	"Laget plass" til sporveksel	14,25	01.01.1981 00:00:00	49-E1	0,005
Skinne, S49, Lørenskog stasjon	14,370	14,400	0			01.01.1988 00:00:00	49-E1	0,03
Skinne, S49, Lørenskog stasjon	14,400	14,443	0	"Laget plass" til sporveksel	14,46	01.01.1976 00:00:00	49-E1	0,043
Skinne, SPV-143a, Lørenskog stasjon	14,443	14,476	0			01.01.1983 00:00:00	49-E1	0,033
Skinne, S49, Lørenskog stasjon	14,476	14,730	0	"Laget plass" til sporveksel	14,46	01.01.1983 00:00:00	49-E1	0,254
Skinne, S49, Lørenskog - Strømmen	14,730	14,957	0	Fjernet 4. desimal	14,7301	01.01.1983 00:00:00	49-E1	0,227
Skinne, S49, Lørenskog - Strømmen	14,957	15,551	0			01.01.1975 00:00:00	49-E1	0,594
Skinne, UIC54E, Lørenskog - Strømmen	15,551	17,204	0			01.01.1986 00:00:00	54-E2	1,653
Skinne, UIC54E, Strømmen stasjon	17,204	17,470	0	Fjernet 4. desimal	17,2041	01.01.1987 00:00:00	54-E2	0,266
Skinne, S49, Strømmen stasjon	17,470	17,479	0			01.01.1995 00:00:00	49-E1	0,009
	17,479	17,481	0	Fra tilstandskontrollør			54-E3	0,002
Skinne, SPV-147a, Strømmen stasjon	17,481	17,512	0			01.01.1980 00:00:00	49-E1	0,031
	17,512	17,517	0	Fra tilstandskontrollør			54-E3	0,005
Skinne, SPV-145b, Strømmen stasjon	17,517	17,551	0			01.01.1979 00:00:00	49-E1	0,034
	17,551	17,558	0	Fra tilstandskontrollør			54-E3	0,007
Skinne, SPV-149a, Strømmen stasjon	17,558	17,594	0			01.09.2007 00:00:00	49-E1	0,036
	17,594	17,692	0	Fra tilstandskontrollør			54-E3	0,098
Skinne, S54, Strømmen stasjon	17,692	18,002	0			01.01.2008 00:00:00	54-E3	0,31
	18,002	18,066	0	Fra tilstandskontrollør			54-E3	0,064
Skinne, SPV-153b, Strømmen stasjon	18,066	18,098	0			01.09.2007 00:00:00	49-E1	0,032
	18,098	18,106	0	Fra tilstandskontrollør			54-E3	0,008
Skinne, SPV-155a, Strømmen stasjon	18,106	18,139	0			01.01.1979 00:00:00	49-E1	0,033
	18,139	18,226	0	Fra tilstandskontrollør			54-E3	0,087
Skinne, SPV-157b, Strømmen stasjon	18,226	18,258	0			01.01.1979 00:00:00	49-E1	0,032
Skinne, S49, Strømmen stasjon	18,258	18,331	0			01.01.1973 00:00:00	49-E1	0,073
Skinne, S49, Strømmen stasjon	18,331	18,432	0			01.01.1973 00:00:00	49-E1	0,101
Skinne, S49, Strømmen - Lillestrøm	18,432	18,441	0	Fjernet 4. desimal	18,4321	01.01.1973 00:00:00	49-E1	0,009
Skinne, S49, Strømmen - Lillestrøm	18,441	18,579	0			01.01.1973 00:00:00	49-E1	0,138
Skinne, S54, Strømmen - Lillestrøm	18,579	18,815	0			01.10.2014 00:00:00	54-E3	0,236
Skinne, S54, Strømmen - Lillestrøm	18,815	19,415	0			28.05.2015 00:00:00	54-E3	0,6
Skinne, S49, Strømmen - Lillestrøm	19,415	19,581	0			01.01.1973 00:00:00	49-E1	0,166
Skinne, S54, Strømmen - Lillestrøm	19,581	19,732	0			01.01.1996 00:00:00	54-E3	0,151

Eksempel på Python-skript som konverter bearbeidet Excel-fil til variabel til masteroppgavens datasett

```
## Eksempel paa kode for omgjoering av banedata til variabler til
## masteroppgavens datasettt

"""
Dette skriptet henter en bearbeidet Excel-fil med data om skinneprofil
paa Hovedbanen. Det er dobbeltspor paa strekningen, saa Excel-filen
inneholder 4 ark, ett for hver av skinnestrengene.

Skriptet henter informasjonen fra Excel-filen og putter det inn i en
DataFrame. Deretter kjoeres det en funksjon som sjekker om det mangler
data. Info om den manglende dataetn skrives til fil. Deretter
''strekkes'' hver rad i DataFrame'n slik at det ikke blir noen
''huller''.

Deretter telles det antall profilbytter paa hvert km-segment, foer det
regnes ut et gjennomsnitt av skinneprofiler og profilbytter for hver
skinnestreng. Dette skrives til fil, som variabler som skal inn i
masteroppgavens datasett.


Se foroevrig avsnittet ''Metode - Datainnsamling / Variabler fra
Banedata / Haandtering av kontinuerlige variabler''

"""

import pandas as pd
import numpy as np
import copy
import os


# Funksjon for aa sjekke om det er mangel i dataene
hull = []

def hulltest(df):

    for i, row in df.iterrows():

        # Summerer opp radene, om ikke summen = 1 er det en mangel
        # paa raden
        if np.sum(row) != 1:

            hull.append((i, np.sum(row)))




# Funksjon for aa sette inn skinneprofilene i output-datasettet
def insert_profil(df):

    columns = ['Skinneprofil_49E1', 'Skinneprofil_54E3',
               'Skinneprofil_60E1_60E2', 'Skinneprofil_35kg',
               'Skinneprofil_NSB40', 'Skinneprofil_S41',
```

```
                'Skinneprofil_54E2', 'Skinneprofil_54E1',
                'Skinneprofil_overganger']

    index = list(range(3, 20))

    skinneprofil = pd.DataFrame(columns=columns, index=index).fillna(0)

    for i, row in df.iterrows():

        fra = max(3, row['Fra'])
        fra_ceil = np.ceil(fra) if fra % 1 != 0. else fra + 1
        til = row['Til']
        til_ceil = np.ceil(til) if fra % 1 != 0. else til + 1
        profil = 'Skinneprofil_' + row['Skinneprofil'].replace('-', '')

        km = list(range(max(3, np.floor(fra).astype(int)),
                                        int(til_ceil)))

        if len(km) == 1:

            skinneprofil.loc[km[0], profil] += til - fra

        else:

            skinneprofil.loc[km[0], profil] += fra_ceil - fra

        if len(km) > 1:

            skinneprofil.loc[km[-1], profil] += til % 1

        if len(km) > 2:

            skinneprofil.loc[km[1:-1], profil] += 1

    hulltest(skinneprofil)

    return skinneprofil.round(3) # Stoler ikke paa noeyaktigheten mer
                                 # enn tre desimaler


# Mangel paa data som er paa mindre enn 0.1 pr km ignoreres.
# Dataene ''strekkes''.
def stretch(df):

    df_s = copy.copy(df)

    for i, row in df_s.iterrows():

        sum_ = np.sum(row.values)

        if sum_ != 1.0:

            df_s.loc[i, :] = np.round(row * (1/sum_), 3) # runder av
                                                         # til 3 des.
        assert np.sum(df_s.loc[i, :]) == 1.0

```

```
110     return df_s
111
112
113 # Funksjon for aa telle antall profilbytter
114 def overganger(sk_orig, sk_km):
115
116     sk_overgang = copy.copy(sk_km)
117
118     forrige = '49-E1'
119
120     for i, row in sk_orig.iterrows():
121
122         km = np.floor(row['Fra'])
123         profil = row['Skinneprofil']
124
125         if profil != forrige:
126
127             sk_overgang.loc[km, 'Skinneprofil_overganger'] += 1
128
129             forrige = profil
130
131     return sk_overgang
132
133
134 # Funksjon for aa regne ut gjennomsnitt av to skinnestrenger
135 def snitt_h_v(skinne_h, skinne_v):
136
137     columns = ['Skinneprofil_49E1', 'Skinneprofil_54E3',
138                'Skinneprofil_60E1_60E2', 'Skinneprofil_35kg',
139                'Skinneprofil_NSB40', 'Skinneprofil_S41',
140                'Skinneprofil_54E2', 'Skinneprofil_54E1',
141                'Skinneprofil_overganger']
142
143     index = list(range(3, 20))
144
145     skinneprofil_snitt = pd.DataFrame(
146                          (skinne_h.values + skinne_v.values) * (1/2),
147                           columns=columns, index=index)
148
149     for i, row in skinneprofil_snitt.iterrows():
150
151         assert np.sum(row[:-1]).round(3) == 1.0 # Siste celle,
152                                                 # skal ikke vaere med i
153                                                 # summasjonen.
154
155     return skinneprofil_snitt
156
157
158 if __name__ == '__main__':
159
160     # Last inn data
161     s = pd.read_excel('skinne_tren_til_ipynb_HO.xlsx',
162                       sheet_name=None) # Les inn alle ark i
163                                        # Excel-dokumentet
164
165
```

```
166     # Liste med Excel-arkene, en skinnestreng per ark
167     sett = [skinnestreng for skinnestreng in s]
168     skinner = []
169 
170     for skinnestreng in sett:
171 
172         print(skinnestreng)
173 
174         skinne = s[skinnestreng]
175 
176         skinneprofil = insert_profil(skinne)
177 
178         skinneprofil_stretched = stretch(skinneprofil)
179 
180         skinneprofil_overganger = overganger(skinne,
181                                              skinneprofil_stretched)
182 
183         skinner.append(skinneprofil_overganger)
184 
185     skinneprofil_snitt_h = snitt_h_v(skinner[0], skinner[1])
186     skinneprofil_snitt_v = snitt_h_v(skinner[2], skinner[3])
187 
188     skinneprofil_snitt = snitt_h_v(skinneprofil_snitt_h,
189                                    skinneprofil_snitt_v)
190 
191 
192     # Skriv de ferdige variablene til fil
193     skinneprofil_snitt.round(3).to_csv('skinneprofil_Hovedbanen.csv',
194                                        index_label='Km')
195 
196 
197     # Skriv liste med huller til fil
198     np.savetxt('hull.csv', hull)
```

Listing 9.1: Fra bearbeidet data om skinneprofil til variabel til oppgavens datasett.

Nedlastning av værdata fra thredds.met.no

```
%% Extracting data from thredds.met.no via OPENDAP

% This script downloads data from thredds.met.no via the OPENDAP protocoll.
% It returns a 2D array of shape 61122 x 25, where the row represents all
% km-segments for the whole time period and the columns represents the
% weather variables.

% The data is extracted by a OPENDAP-path consisting of date and weather
% variable and constraints for time,
% height and x and y indexes. Example:
% "https://thredds.met.no/thredds/dodsC/meps25epsarchive/2017/07/27/
% meps_mbr0_pp_2_5km_20170727T00Z.nc?
% fog_area_fraction[0:1:0][0:1:0][0:1:0][0:1:0]"

% More info about OPENDAP and how to extract data from thredds can be found
% here: https://drive.google.com/file/d/1Mmt_l88x74wXbuillgCVEI56egtLcW01/view


%% Constraints
X = "[254:270]"; % indexes east-west, to limit the amount of data loaded
Y = "[307:347]"; % indexes north-south
time_span = "[21:44]"; % Kl 22-21 UTC+1
height = "[0:1:0]";
variables = ["air_temperature_2m", ...
             "precipitation_amount_acc", ...
             "fog_area_fraction", ...
             "cloud_area_fraction", ...
             "wind_speed_of_gust"];


%% Array of dates
ar2017 = datetime(2017, 3, 31) : datetime(2017, 9, 29);
ar2018 = datetime(2018, 3, 31) : datetime(2018, 9, 29);
dates = horzcat(ar2017, ar2018)';


%% Getting the indexes for the locations of all km-segments
indexes = nc_indexes;


%% Loop through the dates and extract data

% Defining path
path_start = 'https://thredds.met.no/thredds/dodsC/meps25epsarchive/';
path_mid = '/meps_mbr0_pp_2_5km_';
path_end = 'T00Z.nc?'; % Kl 00 UTC

% Ouput array, 167 = number of km-segments, 5 = number of statistics
% to calculate
temp_out = nan(167*length(dates), 5*length(variables));

% Start clock
tic

for i = 1:length(dates)

        % Date strings to place in the OPENDAP-path
    date1 = datestr(dates(i), 'yyyy/mm/dd');
    date2 = datestr(dates(i), 'yyyymmdd');
    disp(dates(i));

    for j = 1:length(variables)

        disp(variables(j));

        % Two of the variables do not have height dimension
        if (variables(j) == "cloud_area_fraction" || ...
                        variables(j) == "wind_speed_of_gust") & (i < 226 || i > 242)

            constraint = strcat(variables(j), time_span, Y, X);
        else
            constraint = strcat(variables(j), time_span, height, Y, X);
        end

                % Putting all parts of the OPENDAP-path together
        path = strcat(path_start, date1, path_mid, date2, path_end, constraint);

        file_found = 0;
        while file_found == 0
            try
                % Load file
                            nc = rot90(ncread(path, variables(j)));
                file_found = 1;
            catch

                % If file not found, try the next file from 6 hours later
                path_ch = char(path); % string to char to be able to index
                new_time = double(string(path_ch(94:95))) + 6; % Add 6 hours.
                path_1 = path_ch(1:93);
                path_2 = path_ch(96:end);

                path = strcat(path_1, pad(string(new_time), 2, 'left', '0'), path_2);

                subset_start = strfind(path, '[');
                subset_mid = strfind(path, ':');
                subset_end = strfind(path, ']');

                path_ch = char(path);
                subset_from = double(string(path_ch(subset_start(1) + 1 : subset_mid(2) - 1))) - 6;

                subset_to = double(string(path_ch(subset_mid(2) + 1 : subset_end(1) - 1))) - 6;
```

```
                path_1 = path_ch(1:subset_start(1));
                path_2 = path_ch(subset_end(1):end);

                            % New path with new file
                path = strcat(path_1, string(subset_from), ':', string(subset_to), path_2);

                            % If no file found for the date, try the extracted
                            % dataset instead
                if new_time >= 24
                    disp('File not found! Using extracted dataset instead.');
                    path_extracted = strcat(path_start, date1, '/meps_mbr0_extracted_2_5km_', date2, path_end, constraint);

                    disp(path_extracted);
                    nc = rot90(ncread(path_extracted, variables(j)));

                    break
                end

            end
        end


        for k = 1:length(indexes)


            var = nc(indexes(k, 1), indexes(k, 2), 1, :);
            [~, ~, ~, time_len] = size(var);

            var = reshape(var, time_len, 1);

            temp_out(k + (i-1)*length(indexes), ...
                                    (1:length(variables)) + (j-1)*length(variables)) = ...
                                    [min(var), max(var), mean(var), std(var), median(var)];

        end

    end
end

toc
```

Listing 9.2: Nedlastning av værdata fra thredds.met.no.

Funksjon for kobling av NetCDF-indekser til km-segmentene

```
% Denne funksjoner kobler X- og Y-indeksene i NetCDF-filene til riktig
% km-segment ved hjelp av korteste euklidiske avstand.

% X- og Y-indeksene i NetCDF-filene er lengde- og breddegrader i WGS84.

function [indexes] = nc_indexes()

% Bredde-koordinater
nc_name_lat = "https://thredds.met.no/thredds/dodsC/meps25epsarchive/2017/04/01/meps_mbr0_pp_2_5km_20170401T00Z.nc?...
                                latitude[307:347][254:270]";
lat = rot90(ncread(nc_name_lat, 'latitude'));

% Lengde-koordinater
nc_name_lon = "https://thredds.met.no/thredds/dodsC/meps25epsarchive/2017/04/01/meps_mbr0_pp_2_5km_20170401T00Z.nc?...
                                longitude[307:347][254:270]";
lon = rot90(ncread(nc_name_lon, 'longitude'));

% Km-segmentenes lengde- og breddekoordinater (midtpunkter)
rail = readmatrix('midtpunkter_lat_lon.xlsx');
rail = rail(:, 3:end);

num_segments = length(rail);
[len_y, len_x] = size(lat);

indexes = zeros(167, 2);

for i = 1:num_segments

    shortest_dist = 99999999999;

    for j = 1:len_y

        for k = 1:len_x

            % Euklidisk avstand
            dist = sqrt( (rail(i, 1) - lat(j, k))^2 + (rail(i, 2) - lon(j, k))^2 );

            if dist < shortest_dist

                shortest_dist = dist;
                Y = j;
                X = k;

            end
        end
    end

    indexes(i, :) = [Y X];

end

end
```

Listing 9.3: Kobling av NetCDF-indekser til km-segmentene

Personer kontaktet for historisk og/eller manglende data i Banedata

Navn	Tlf	E-post	Stilling	Bane
Roar Dyrud	916 79 999	dyrroa@banenor.no	Tilstandskontrollør	Gjøvikbanen
Per Edvin Fjeldbu	456 30 837	fjeldp@banenor.no	Tilstandskontrollør	Roa-Hønefossbanen
Ulf Rudolf Reffsgård	916 69 442	reftru@banenor.no	Drift og prosjektstøtte jernbanefag	Hovedbanen

Vedlegg 2 - Kode-eksempler for maskinlæringsmodellene

Random Forest

```
## Random Forest with resampling

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

from imblearn.over_sampling import SMOTE


# Load data
path = 'Treningssett/'
X = pd.read_csv(path + 'X.csv', index_col=0)
y = pd.read_csv(path + 'y.csv', index_col=0)


# Parameters
n_estimators = 200
max_depth = list(range(1, 15))
criterion = 'entropy'
random_states = range(10)
class_weight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}


# Lists of results
train_f1 = []
test_f1 = []


for depth in max_depth:

    print('Training model #', max_depth.index(depth)+1, 'of',
                                                    len(max_depth))

    train_f1_temp = []
    test_f1_temp = []

    for rs in random_states:

        # Split in train and test set
        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    stratify=y,
                                                    random_state=rs)
        # Scale and standardize data
        sc = StandardScaler()
        X_train_std = sc.fit_transform(X_train)
```

```
53          X_test_std = sc.transform(X_test)
54
55          # Resample training data
56          sm = SMOTE(sampling_strategy=1/3,
57                     n_jobs=-1,
58                     random_state=2)
59
60          X_train_std_res, y_train_res = sm.fit_sample(X_train_std,
61                                                       y_train)
62
63          # Define model and predict
64          rf = RandomForestClassifier(n_estimators=n_estimators,
65                                      criterion=criterion,
66                                      max_depth=depth,
67                                      class_weight=class_weight(y_train_res),
68                                      random_state=rs, n_jobs=-1)
69
70          rf.fit(X_train_std_res, y_train_res)
71          y_train_pred = rf.predict(X_train_std_res)
72          y_test_pred = rf.predict(X_test_std)
73
74          # Store results
75          train_f1_temp.append(f1_score(y_train_res, y_train_pred))
76          test_f1_temp.append(f1_score(y_test, y_test_pred))
77
78      train_f1.append(np.mean(train_f1_temp))
79      test_f1.append(np.mean(test_f1_temp))
80
81
82  # Plot results
83  plt.plot(max_depth, train_f1)
84  plt.plot(max_depth, test_f1)
85  plt.legend(['Train f1 score', 'Test f1 score'])
86  plt.show()
87
88
89  # Save results to file
90  np.savetxt('test.csv', test_f1)
91  np.savetxt('train.csv', train_f1)
```

Listing 9.4: Random Forest

Isolation Forest

```
## Isolation Forest with PCA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
from sklearn.decomposition import PCA


# Load data
path = 'Treningssett/'
X = pd.read_csv(path + 'X.csv', index_col=0)
y = pd.read_csv(path + 'y.csv', index_col=0)
X.head()


# Parameters
n_comp = 10
n_estimators = [1, 5, 10, 25, 50, 75, 100, 250, 500, 750, 1000]
r_state = 1

# Scale and standardize data
sc = StandardScaler()
X_std = sc.fit_transform(X)

# Decompose by PCA
pca = PCA(n_components=n_comp, random_state=1)
X_std_pca = pca.fit_transform(X_std)


# List of results
f1 = []


for n_est in n_estimators:

    print('Training model #', n_estimators.index(n_est)+1, 'of',
                                            len(n_estimators))

    # Fit and predict model
    isf = IsolationForest(n_estimators=n_est, random_state=r_state,
                                                    n_jobs=-1)
    isf.fit(X_std_pca)
    y_pred = isf.predict(X_std_pca)

```

```
56      # From list of -1 and 1 to list of 0 and 1
57      y_pred = np.abs((y_pred - 1) / 2)
58
59      # Store result
60      f1.append(f1_score(input_y.values, y_pred))
61
62
63  # Plot results
64  plt.plot(n_estimators, f1)
65  plt.legend(['F1 score'])
66  plt.show()
67
68
69  # Save results to file
70  np.savetxt('f1.csv', f1)
```

Listing 9.5: Isolation Forest

Logistisk regresjon

```
## Logistic Regression

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score


# Load data
path = 'Treningssett/'
X = pd.read_csv(path + 'X.csv', index_col=0)
y = pd.read_csv(path + 'y.csv', index_col=0)
X.head()


# Parameters
Cs = list(np.logspace(-3, 3.5, 15))
penalty = 'l2'
solver = 'saga'
classweight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}
random_states = range(10)


# Lists for results
train_f1 = []
test_f1 = []


for c in Cs:

    print('Training model #', Cs.index(c)+1, 'of', len(Cs))

    train_f1_temp = []
    test_f1_temp = []

    for rs in random_states:

        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.3,
                                                            stratify=y,
                                                           random_state=rs)

        # Scale and standardize data
        sc = StandardScaler()
```

```
56          X_train_std = sc.fit_transform(X_train)
57          X_test_std = sc.transform(X_test)
58
59          # Fit and predict
60          lr = LogisticRegression(C=c, penalty=penalty, solver=solver,
61                                  class_weight=classweight(y_train),
62                                  random_state=rs, n_jobs=-1)
63
64          lr.fit(X_train_std, y_train)
65          y_train_pred = lr.predict(X_train_std)
66          y_test_pred = lr.predict(X_test_std)
67
68          # Store results
69          train_f1_temp.append(f1_score(y_train, y_train_pred))
70          test_f1_temp.append(f1_score(y_test, y_test_pred))
71
72      train_f1.append(np.mean(train_f1_temp))
73      test_f1.append(np.mean(test_f1_temp))
74
75
76  # Plot results
77  plt.plot(Cs, train_f1)
78  plt.plot(Cs, test_f1)
79  plt.legend(['Train f1 score', 'Test f1 score'])
80  plt.show()
81
82
83  # Save results to file
84  np.savetxt('test.csv', test_f1)
85  np.savetxt('train.csv', train_f1)
```

Listing 9.6: Eksempel-kode for dimensjonalitetsreduksjon med PCA i maskinlæring

Support Vector Classifier (SVC)

```
# Support Vector Classifier (SVC)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import f1_score


# Load data
path = 'Treningssett/'
X = pd.read_csv(path + 'X.csv', index_col=0)
y = pd.read_csv(path + 'y.csv', index_col=0)


# Parameters
Cs = list(np.logspace(-2, 0, 15))
kernel = 'rbf'
classweight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}
random_states = range(10)


# Lists for results
train_f1 = []
test_f1 = []


for c in Cs:

    print('Training model #', Cs.index(c)+1, 'of', len(Cs))

    train_f1_temp = []
    test_f1_temp = []

    for rs in random_states:

        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.3,
                                                            stratify=y,
                                                           random_state=rs)

        # Scale and standardize data
        sc = StandardScaler()
        X_train_std = sc.fit_transform(X_train)
        X_test_std = sc.transform(X_test)

        # Fit and predict
        lr = SVC(C=c, kernel=kernel, class_weight=classweight(y_train),
                                                    random_state=rs)
        lr.fit(X_train_std, y_train)
        y_train_pred = lr.predict(X_train_std)
        y_test_pred = lr.predict(X_test_std)
```

```
56
57          # Store results
58          train_f1_temp.append(f1_score(y_train, y_train_pred))
59          test_f1_temp.append(f1_score(y_test, y_test_pred))
60
61      train_f1.append(np.mean(train_f1_temp))
62      test_f1.append(np.mean(test_f1_temp))
63
64
65  # Plot results
66  plt.plot(Cs, train_f1)
67  plt.plot(Cs, test_f1)
68  plt.legend(['Train f1 score', 'Test f1 score'])
69  plt.show()
70
71
72  # Save results to file
73  np.savetxt('test.csv', test_f1)
74  np.savetxt('train.csv', train_f1)
```

Listing 9.7: Support Vector Classifier

Multi-Layer Neural Network

```
## Dense Neural Network (Dense NN)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score

from keras.callbacks import Callback
from keras import layers
from keras import models
from keras import backend as K
from keras import optimizers

import tensorflow as tf


# Functions for F1 as loss function and metric
def f1(y_true, y_pred):
    """
    Code copied from
    'https://www.kaggle.com/rejpalcz/
    best-loss-function-for-f1-score-metric'
    (read 12.05.2019)
    Author: Michal Haltuf
    """
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)

    return K.mean(f1)

def f1_loss(y_true, y_pred):
    """
    Code copied from
    'https://www.kaggle.com/rejpalcz/
    best-loss-function-for-f1-score-metric'
    (read 12.05.2019)
    Author: Michal Haltuf
    """
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

```

```
    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return 1 - K.mean(f1)


# Load data
path = 'Treningssett/'
X = pd.read_csv(path + 'X.csv', index_col=0)
y = pd.read_csv(path + 'y.csv', index_col=0)


# Parameters
classweight = lambda y: {1.: y.shape[0] - np.sum(y), 0.: np.sum(y)}
n_splits = 10
num_epochs = 50


# Lists for results
train_all_f1_histories = []
val_all_f1_histories = []


# Define model
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(100, activation='relu',
                           input_shape=(X.shape[1],)))

    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizers.SGD(lr=.01),
                          loss=f1_loss, metrics=[f1])
    return model


K.clear_session()

for i in range(n_splits):
    print('processing split #', i+1)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                             test_size=0.3, stratify=y, random_state=i)

    # Scale and standardize data
    sc = StandardScaler()
    X_train_std = sc.fit_transform(X_train)
    X_test_std = sc.transform(X_test)

    # Build model and fit
    model = build_model()
    history = model.fit(X_train_std, y_train,
                          validation_data=(X_test_std, y_test),
                          epochs=num_epochs, batch_size=1000,
                          class_weight=classweight(y_train.values),
```

```
                          verbose=0)

    # Get results
    train_all_f1_histories.append(history.history['f1'])
    val_all_f1_histories.append(history.history['val_f1'])


# Calculate average results for each epoch
train_average_f1_history = [np.mean([x[i]
                             for x in train_all_f1_histories])
                             for i in range(num_epochs)]

val_average_f1_history = [np.mean([x[i]
                           for x in val_all_f1_histories])
                           for i in range(num_epochs)]


# Plot results
plt.plot(range(num_epochs), train_average_f1_history)
plt.plot(range(num_epochs), val_average_f1_history)
plt.legend(['Train f1 score', 'Test f1 score']);
plt.show()


# Save results to file
np.savetxt('test.csv', val_average_f1_history)
np.savetxt('train.csv', train_average_f1_history)
```

Listing 9.8: Multi-Layer Neural Network

Recurrent Neural Network (RNN)

```
## Recurrent Neural Network (RNN)
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score

import tensorflow as tf

import keras
from keras.models import Sequential
from keras import layers
import keras.backend as K
from keras import optimizers


# Connect to drive
from google.colab import drive
drive.mount('/content/drive/')


# Load data
Xy_RNN = pd.read_csv('drive/My Drive/Master/Colab/data/Xy_RNN.csv',
                                                        index_col=0)


# Scale and standardize data
sc = StandardScaler()
X_std = sc.fit_transform(Xy_RNN.iloc[:, :-1])
data = pd.concat([pd.DataFrame(X_std), Xy_RNN.iloc[:, -1]], axis=1)


# Define generator
def generator(data, lookback, delay,
              shuffle=False, batch_size=128, step=6):
    """
    Code copied and modified from
    'https://github.com/fchollet/deep-learning-with-python-notebooks/
    blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb'
    (read 12.05.2019)
    Author: Francois Chollet
    """
    min_index = 0
    max_index = data.shape[0] - delay - 1

    if max_index is None:
        max_index = len(data) - delay - 1
    i = min_index + lookback

    while 1:

        if shuffle:
```

```
56              rows = np.random.randint(
57                  min_index + lookback, max_index, size=batch_size)
58          else:
59              if i + batch_size >= max_index:
60                  i = min_index + lookback
61              rows = np.arange(i, min(i + batch_size, max_index))
62              i += len(rows)
63
64          samples = np.zeros((len(rows),
65                              lookback // step,
66                              data.shape[-1]))
67
68          targets = np.zeros((len(rows),))
69          for j, row in enumerate(rows):
70              indices = range(rows[j] - lookback, rows[j], step)
71              samples[j] = data[indices]
72              targets[j] = data[rows[j] + delay][-1]
73
74          yield samples, targets
75
76
77  # Functions for F1 as loss function and metric
78  def f1(y_true, y_pred):
79      """
80      Code copied from
81      'https://www.kaggle.com/rejpalcz/
82      best-loss-function-for-f1-score-metric'
83      (read 12.05.2019)
84      Author: Michal Haltuf
85      """
86      y_pred = K.round(y_pred)
87      tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
88      tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
89      fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
90      fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)
91
92      p = tp / (tp + fp + K.epsilon())
93      r = tp / (tp + fn + K.epsilon())
94
95      f1 = 2*p*r / (p+r+K.epsilon())
96      f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
97
98      return K.mean(f1)
99
100 def f1_loss(y_true, y_pred):
101     """
102     Code copied from
103     'https://www.kaggle.com/rejpalcz/
104     best-loss-function-for-f1-score-metric'
105     (read 12.05.2019)
106     Author: Michal Haltuf
107     """
108     tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
109     tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
110     fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
111     fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)
```

```
112
113     p = tp / (tp + fp + K.epsilon())
114     r = tp / (tp + fn + K.epsilon())
115
116     f1 = 2*p*r / (p+r+K.epsilon())
117     f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
118     return 1 - K.mean(f1)
119
120
121 # Function for splitting data in train and test set
122 # It also takes out the non-solslyng segments
123 def train_test_split():
124
125     km_segmenter = {i + 1: data.iloc[i * 366 : (i+1) * 366].values
126                     for i in range(167)}
127
128     # Dict for only km-segments with sun kink
129     km_segmenter_sol = {}
130
131     for km in km_segmenter.keys():
132
133         if 1. in km_segmenter[km][:, -1]:
134
135             km_segmenter_sol[km] = km_segmenter[km]
136
137     # Find the keys for shuffling:
138     sol_ix = np.fromiter(km_segmenter_sol.keys(), dtype=int)
139     np.random.seed(1)
140     np.random.shuffle(sol_ix)
141
142     # Obtaining test/train-size
143     n_segmenter = len(km_segmenter.keys())
144     test_frac = 0.3
145     test_size = int(n_segmenter * test_frac)
146     train_size = n_segmenter - test_size
147     n_sol = 38
148     n_sol_test = int(n_sol * test_frac)
149     n_sol_train = n_sol - n_sol_test
150
151     # Insert into test and train data
152     km_segmenter_sol_train = {km: km_segmenter_sol[km]
153                               for km in sol_ix[:n_sol_train]}
154
155     km_segmenter_sol_test = {km: km_segmenter_sol[km]
156                              for km in sol_ix[-n_sol_test:]}
157
158     # Final train array
159     train = np.zeros((len(km_segmenter_sol_train.keys()) * 366, 72))
160
161     # Shuffle train data
162     train_ix = np.array( list(km_segmenter_sol_train.keys()))
163     np.random.shuffle(train_ix)
164
165     for i, km in enumerate(train_ix):
166
167         train[i * 366 : (i+1) * 366, :] = km_segmenter_sol_train[km]
```

```
168
169     # Final test array
170     test = np.zeros((len(km_segmenter_sol_test.keys()) * 366, 72))
171
172     # Shuffle test data
173     test_ix = np.array(list(km_segmenter_sol_test.keys()))
174     np.random.shuffle(test_ix)
175
176     for i, km in enumerate(test_ix):
177
178         test[i * 366 : (i+1) * 366, :] = km_segmenter_sol_test[km]
179
180
181     return train, test
182
183
184 # Define generators
185 def define_generators(train, test):
186     """
187     Code copied and modified from
188     'https://github.com/fchollet/deep-learning-with-python-notebooks/
189     blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb'
190     (read 12.05.2019)
191     Author: Francois Chollet
192     """
193     lookback = 200
194     step = 1
195     delay = 20
196     batch_size = 128
197
198     train_gen = generator(train,
199                           lookback=lookback,
200                           delay=delay,
201                           shuffle=True,
202                           step=step,
203                           batch_size=batch_size)
204
205
206     test_gen = generator(test,
207                          lookback=lookback,
208                          delay=delay,
209                          step=step,
210                          batch_size=batch_size)
211
212
213     # This is how many steps to draw from `test_gen`
214     # in order to see the whole test set:
215     test_steps = (test.shape[0] - 1 - lookback) // batch_size
216
217     return train_gen, test_gen, test_steps
218
219
220 # Parameters
221 n_splits = 5
222 num_epochs = 50
223
```

```
224 # For class 1. (solslyng), calculating number of non-solslyng samples
225 classweight = {1.: 38*366 - 55, 0.: 55}
226
227
228 # Lists for results
229 train_all_f1_histories = []
230 val_all_f1_histories = []
231
232
233 K.clear_session()
234
235 for i in range(n_splits):
236
237     print('processing split #', i+1)
238
239     # Split data
240     train, test = train_test_split()
241
242     # Define generators
243     train_gen, test_gen, test_steps = define_generators(train, test)
244
245
246     # Define model
247     model = Sequential()
248     model.add(layers.GRU(16, input_shape=(None, data.shape[-1]),
249                          dropout=0.2, recurrent_dropout=0.2))
250
251     model.add(layers.Dense(1, activation='sigmoid'))
252
253     model.compile(optimizer=optimizers.SGD(lr=.001), loss=f1_loss,
254                                          metrics=[f1])
255
256     history = model.fit_generator(train_gen,
257                                   steps_per_epoch=100,
258                                   epochs=num_epochs,
259                                   validation_data=test_gen,
260                                   validation_steps=test_steps,
261                                   class_weight=classweight)
262
263     # Store results
264     train_all_f1_histories.append(history.history['f1'])
265     val_all_f1_histories.append(history.history['val_f1'])
266
267
268 # Calculation mean results for each epoch
269 train_average_f1_history = [np.mean([x[i]
270                              for x in train_all_f1_histories])
271                              for i in range(num_epochs)]
272
273 val_average_f1_history = [np.mean([x[i]
274                            for x in val_all_f1_histories])
275                            for i in range(num_epochs)]
276
277
278 # Plot results
279 plt.plot(range(num_epochs), train_average_f1_history, 'bo',
```

```
                                        label='Training score')
plt.plot(range(num_epochs), val_average_f1_history, 'b',
                                        label='Validation score')
plt.title('Training and validation score')
plt.legend()
plt.show()


# Save results to file
np.savetxt('test.csv', val_average_f1_history)
np.savetxt('train.csv', train_average_f1_history)
```

Listing 9.9: Recurrent Neural Network

Vedlegg 3 - Variablene i datasettet

Ballastfordeling

Format: Heltall | **NULL-verdi:** 100000
Beskrivelse: Antall dager siden ballastfordeling sist ble utført på km-segmentet. Omfatter også ballastsupplering og ballastprofilering.

Ballastrensing_1990_2011

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet hvor det ble utført ballastrensing fra og med år 1990 til og med år 2011.

Ballastrensing_etter_2011

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet hvor det ble utført ballastrensing etter år 2011.

Befestigelse_overganger

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Antall ganger type befestigelse skifter innad på et km-segment. Et gjennomsnitt per spor for strekninger med dobbeltspor, det vil si antall delt på 2.

Betongsville_2_3m

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av betongsviller med 2.3 meters bredde. Omfatter svilletypen Betongsville Type 2 Pandrol.

Betongsville_2_4m

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier **Beskrivelse:** Andel av km-segmentet som består av betongsviller med 2.4 meters bredde. Omfatter svilletypene JBV 54, JBV 54 "Dual rail", JBV 97, NSB 90, NSB Enhetssville og Brusville 97 type 2.

Betongsville_2_6m

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier **Beskrivelse:** Andel av km-segmentet som består av betongsviller med 2.6 meters bredde. Omfatter svilletypene JBV 60, JBV 60 "Dual rail", NSB 95 og NSB 93.

Betongsville_for_sporveksel

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av betongsviller for sporveksler.

Brusville_av_tre

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av brusviller av furu eller bøk.

Cloud_area_max

Format: Desimaltall [0, 100] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum skydekke i % fra kl 22 dagen før til kl 21 samme dag.

Cloud_area_mean

Format: Desimaltall [0, 100] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig skydekke i % fra kl 22 dagen før til kl 21 samme dag.

Cloud_area_median

Format: Desimaltall [0, 100] | **NULL-verdi:** ingen manglende verdier **Beskrivelse:** Median skydekke i % fra kl 22 dagen før til kl 21 samme dag.

Cloud_area_min

Format: Desimaltall [0, 100] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum skydekke i % fra kl 22 dagen før til kl 21 samme dag.

Cloud_area_std

Format: Desimaltall [0, 100] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Standardavvik skydekke i % fra kl 22 dagen før til kl 21 samme dag.

Diffuse_radiation_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum diffus solinnstråling i W/m^2.

Diffuse_radiation_mean

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig diffus solinnstråling i W/m^2.

Diffuse_radiation_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum diffus solinnstråling i W/m^2.

Direct_duration_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum antall timer med solinnstråling på en uke.

Direct_duration_mean

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig antall timer med solinnstråling på en uke.

Direct_duration_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum antall timer med solinnstråling på en uke.

Direct_radiation_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksium direkte solinnstråling i W/m^2.

Direct_radiation_mean

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig direkte solinnstråling i W/m^2.

Direct_radiation_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum direkte solinnstråling i W/m^2.

Fastclip

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av Pandrol Fastclip befestigelse.

Fog_area_max

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum tåke-andel fra kl 22 dagen før til kl 21 samme dag.

Fog_area_mean

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig tåke-andel fra kl 22 dagen før til kl 21 samme dag.

Fog_area_median

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Median tåke-andel fra kl 22 dagen før til kl 21 samme dag.

Fog_area_min

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum tåke-andel fra kl 22 dagen før til kl 21 samme dag.

Fog_area_std

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Standardavvik tåke-andel fra kl 22 dagen før til kl 21 samme dag.

Heyback

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av Heyback befestigelse.

Hoyde

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Høyde over havet i meter ved km-segmentets midtpunkt.

Klemplate

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av Klemplate befestigelse.

Minste_radius

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Radius i meter for km-segmentets skarpeste kurve.

Pandrol

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av Pandrol e/PR befestigelse.

Precipitation_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum akkumulert nedbørmengde i kg/m^2 fra kl 22 dagen før til kl 21 samme dag.

Precipitation_mean

Format: Desimaltall| **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig akkumulert nedbørmengde i kg/m^2 fra kl 22 dagen før til kl 21 samme dag.

Precipitation_median

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier **Beskrivelse:** Median akkumulert nedbørmengde i kg/m^2 fra kl 22 dagen før til kl 21 samme dag.

Precipitation_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum akkumulert nedbørmengde i kg/m^2 fra kl 22 dagen før til kl 21 samme dag.

Precipitation_std

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Standardavvik akkumulert nedbørmengde i kg/m^2 fra kl 22 dagen før til kl 21 samme dag.

Retning

Format: Desimaltall [0, 2π> | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Retningen på linjen fra km-segmentets start-punkt til slutt-punkt i radianer, med klokka, i forhold til nord.

Sinuosity

Format: Desimaltall [1, -> | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Km-segmentets sinuosity, lengden til segmentet delt på euklidisk lengde mellom start- og sluttpunkt.

Skinneprofil_35kg

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet 35 kg.

Skinneprofil_49E1

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet 49 E1.

Skinneprofil_54E1

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet 54 E1.

Skinneprofil_54E2

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet 54 E2.

Skinneprofil_54E3

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet 54 E3.

Skinneprofil_60E1_60E2

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilene 60 E1 eller 60 E2.

Skinneprofil_NSB40

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet NSB 40.

Skinneprofil_overganger

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Antall skinneprofilbytter innad på et km-segment. Et gjennomsnitt per spor for strekninger med dobbeltspor, det vil si antall profilbytter delt på 2.

Skinneprofil_S41

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av skinneprofilet S 41.

Skinneskjoter_antall

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Antall isolerte skinneskjøter innad på et km-segment. Et gjennomsnitt per spor for strekninger med dobbeltspor, det vil si antall skinneskjøter delt på 2.

Skinnesliping

Format: Heltall | **NULL-verdi:** 100000
Beskrivelse: Antall dager siden skinnesliping sist ble utført på km-segmentet.

Solar_radiation_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum solinnstråling i W/m^2.

Solar_radiation_mean

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig solinnstråling i W/m^2.

Sporjustering

Format: Heltall | **NULL-verdi:** 100000
Beskrivelse: Antall dager siden sporjustering sist ble utført på km-segmentet. Omfatter alle typer pakking av spor.

Sporstabilisering

Format: Heltall | **NULL-verdi:** 100000
Beskrivelse: Antall dager siden sporstabilisering sist ble utført på km-segmentet.

Sville_overganger

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Antall svillebytter innad på et km-segment. Et gjennomsnitt per spor for strekninger med dobbeltspor, det vil si antall svillebytter delt på 2.

Temperature_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimumstemperatur i kelvin fra kl 22 dagen før til kl 21 samme dag.

Temperature_mean

Format: Desimaltall| **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittstemperatur i kelvin fra kl 22 dagen før til kl 21 samme dag.

Temperature_median

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier **Beskrivelse:** Median temperatur i kelvin fra kl 22 dagen før til kl 21 samme dag.

Temperature_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimumstemperatur i kelvin fra kl 22 dagen før til kl 21 samme dag.

Temperature_std

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Standardavvik temperatur i kelvin fra kl 22 dagen før til kl 21 samme dag.

Tresville

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av tresviller i furu eller bøk.

Tresville_for_sporveksel

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som består av tresviller i sporveksler.

Tunnel

Format: Desimaltall [0, 1] | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Andel av km-segmentet som befinner seg inni tunnel.

Tvangspunkter

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Antall tvangspunkter innad på et km-segment. Et gjennomsnitt per spor for strekninger med dobbeltspor, det vil si antall tvangspunkter delt på 2.

Wind_speed_of_gust_max

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Maksimum vindhastighet i kastene i m/s fra kl 22 dagen før til kl 21 samme dag.

Wind_speed_of_gust_mean

Format: Desimaltall| **NULL-verdi:** ingen manglende verdier
Beskrivelse: Gjennomsnittlig vindhastighet i kastene i m/s fra kl 22 dagen før til kl 21 samme dag.

Wind_speed_of_gust_median

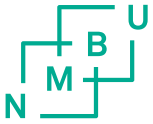
Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Median vindhastighet i kastene i m/s fra kl 22 dagen før til kl 21 samme dag.

Wind_speed_of_gust_min

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Minimum vindhastighet i kastene i m/s fra kl 22 dagen før til kl 21 samme dag.

Wind_speed_of_gust_std

Format: Desimaltall | **NULL-verdi:** ingen manglende verdier
Beskrivelse: Standardavvik vindhastighet i kastene i m/s fra kl 22 dagen før til kl 21 samme dag.



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway