



Faculteit Bedrijf en Organisatie

Vergelijkende studie van ARIMA-, LSTM- en Prophet-voorspellingsmodellen van univariate en multivariate,
seizoensgebonden en niet-seizoensgebonden tijdreeksen

Emiel Declercq

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Johan Decorte
Co-promotor:
Stijn Lievens

Instelling: Hogeschool Gent

Academiejaar: 2020-2021

Eerste examenperiode

Faculteit Bedrijf en Organisatie

Vergelijkende studie van ARIMA-, LSTM- en Prophet-voorspellingsmodellen van univariante en multivariate,
seizoensgebonden en niet-seizoensgebonden tijdreeksen

Emiel Declercq

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Johan Decorte
Co-promotor:
Stijn Lievens

Instelling: Hogeschool Gent

Academiejaar: 2020-2021

Eerste examenperiode

Woord vooraf

Deze bachelorproef werd geschreven als eindwerk voor het afronden van de opleiding Toegepaste Informatica aan de Hogeschool Gent met specialisatie e-business.

Daarnaast wil ik ook nog ir. Johan Decorte bedanken voor de begeleiding van deze bachelorproef en Stijn Lievens voor het opnemen van het co-promotorschap en de uitgebreide feedback.

Mijn ouders wil ik ook hartelijk bedanken voor alle steun, niet enkel tijdens deze scriptie maar gedurende mijn volledige studietraject. Zeker tijdens deze lockdown kan ik mij voorstellen dat dit op sommige momenten heel wat stress heeft veroorzaakt. Ook mijn zus, Edith Declercq verdient een woordje van dank voor al het naleeswerk tot in de late uurtjes en de resem van aangereikte tips bij het schrijven van een paper. Tenslotte wil ik ook zeker mijn vriendin bedanken voor alle steun bij het schrijven van deze bachelorproef, ik denk niet dat ik dit zonder haar had kunnen bolwerken.

Samenvatting

In deze bachelorproef zullen verschillende voorspellingstechnieken voor verschillende types tijdreeksen geanalyseerd worden. Er zal ook getest worden op verschillende types tijdreeksen. Het onderscheid tussen deze types wordt gemaakt op basis van 2 criteria namelijk seizoensgebondenheid en het aantal onafhankelijke variabelen. Deze zullen dan onderverdeeld worden in een al dan niet seizoensgebondenheid en een enkele of meerdere onafhankelijke variabelen. In totaal zullen er dus voorspellingen gemaakt worden voor 4 verschillende types tijdreeksen namelijk.

- Tijdreeksen met enkel de tijd als onafhankelijke variabele en 1 afhankelijke variabele zonder seizoensgebonden verband
- Tijdreeksen met enkel de tijd als onafhankelijke variabele en 1 afhankelijke variabele met een seizoensgebonden verband
- Tijdreeksen met 2 onafhankelijke variabelen waarvan 1 de tijd en 1 afhankelijke variabele zonder een seizoensgebonden verband
- Tijdreeksen met 2 onafhankelijke variabelen waarvan 1 de tijd en 1 afhankelijke variabele met een seizoensgebonden verband

De eerste techniek die zal toegepast worden is een ARIMA/VARMAX model. Als tweede zal een recurrent neuraal netwerk van het type LSTM (Long Term Short Memory) op dezelfde data toegepast worden en tenslotte zal ook Prophet gebruikt worden om voorspelingen te maken. Voor elk van deze 4 types tijdreeksen werd bepaald welk modeltype de laagste MAE behaalde bij gebruik van cross validation. Dit werd toegepast een dataset van de pooljdsdikte waaruit deze resultaten behaald werden:

- Bij de univariate niet-seizoensgebonden tijdreeks zal het ARIMA-model het beste resultaat behalen.

- Bij de univariate seizoensgebonden tijdreeks presteert het SARIMA-model met random walk differentiatie het best.
- Bij de multivariate niet-seizoensgebonden tijdreeks behaalt het VARMAX-model de beste resultaten.
- Bij de multivariate seizoensgebonden tijdreeks zal het LSTM model de beste prestatie leveren.

Tenslotte dient zeker nog vermeld te worden dat dit resultaat zal afhangen van tijdreeks tot tijdreeks en deze modellen niet bij elke reeks het meest optimale resultaat zullen behalen.

Inhoudsopgave

1	Inleiding	15
1.1	Probleemstelling	16
1.2	Onderzoeks vraag	16
1.3	Onderzoeksdoelstelling	16
1.4	Opzet van deze bachelorproef	17
2	Stand van zaken	19
2.1	Regressieanalyse	19
2.1.1	Lineaire Regressie	19
2.1.2	Polynomiale Regressie	21
2.1.3	Logistische Regressie	22
2.1.4	Meervoudige lineaire regressie	23
2.1.5	Toepassing	23

2.2 ARIMA	23
2.2.1 Stationariteit	23
2.2.2 Seizoenseffect	24
2.2.3 Differentiatie	25
2.2.4 Toeliching ARIMA	26
2.2.5 Varianten op ARIMA	30
2.3 Long Short Term Memory	31
2.3.1 Theoretische toelichting	31
2.3.2 Praktische toelichting	36
2.4 Prophet	43
2.5 Validatietechnieken	44
2.5.1 Foutmaten	44
2.5.2 Cross-validation	45
3 Methodologie	47
3.1 Voorbereiding	47
3.1.1 Datavoorbereiding	47
3.2 Univariate niet-seizoensgebonden	56
3.2.1 Algemene methodes	56
3.2.2 Stationariteit	57
3.2.3 Cross validation	59
3.2.4 ARIMA	59
3.2.5 LSTM	63
3.2.6 Prophet	69
3.2.7 Evaluatie	73

3.3 Univariate seizoensgebonden	75
3.3.1 Algemene methodes	75
3.3.2 Stationariteit	76
3.3.3 ARIMA	78
3.3.4 SARIMA	78
3.3.5 LSTM	80
3.3.6 Prophet	81
3.3.7 Evaluatie	82
3.4 Multivariate niet-seizoensgebonden	82
3.4.1 Stationariteit	85
3.4.2 VARMAX	85
3.4.3 LSTM	87
3.4.4 Evaluation	89
3.5 Multivariate seizoensgebonden	89
3.5.1 VARMAX	90
3.5.2 LSTM	90
3.5.3 Evaluation	93
4 Conclusie	95
A Onderzoeksvoorstel	97
A.1 Introductie	97
A.2 Stand van zaken	97
A.3 Methodologie	98
A.4 Verwachte resultaten	98

A.5	Verwachte conclusies	98
B	Broncode	99
B.1	Broncode datavoorbereiding	99
B.2	Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij univariate, niet-seizoensgebonden tijdreeksen	104
B.3	Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij univariate, seizoensgebonden tijdreeksen	118
B.4	Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, niet-seizoensgebonden tijdreeksen	144
B.5	Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, seizoensgebonden tijdreeksen	156
	Bibliografie	175

Lijst van figuren

2.1	Grafische weergave sum of least squares (Brown, 2020)	21
2.2	Grafische weergaven voorbeelddata voor lineaire regressie (Pant2019) 22	
2.3	Voorbeeldtijdreeksen (Rob J Hyndman, 2018)	24
2.4	Voorstelling van een neuron die invoer ontvangt van 3 andere neuro- nen (Lievens, 2018)	32
2.5	Cel in een neuraal netwerk dat gegevens van een andere cel gebruikt om een nieuw signaal uit te sturen (Olah, 2015)	34
2.6	Gedetailleerdere figuur van een cel uit een neuraal netwerk dat in- formatie interpreteert uit een andere cel van het neurale netwerk (Olah, 2015)	34
2.7	Grafische weergave van de forget layer binnen het neuron (Olah, 2015)	35
2.8	Grafische weergave van de input gate layer binnen het neuron (Olah, 2015)	35
2.9	Grafische weergave van het 2 ^{de} deel van de input gateway (Olah, 2015)	35
2.10	Grafische weergave van het onderdeel van de cel waarin de input vermenigvuldigd wordt met de tanh van de celtoestand om de output- waarde te bekomen (Olah, 2015)	36
2.11	Parabolische functie	38

2.12 Een stacked LSTM (Brownlee, 2017c)	40
2.13 Weergave van een bidirectioneel LSTM model (Colah2015)	41
2.14 (a) Gewone LSTM cell, (b) Weergave van een ConvLSTM (Syed Ashiqur Rahman, 2019)	41
2.15 Encoder-Decoder Model (Brownlee, 2017a)	43
2.16 Grafische weergave van underfitting en overfitting (Johan Decorte, 2019)	44
2.17 Grafische weergave van de structuur van een cross-validation tijdreeks (Rob J Hyndman, 2018) waarbij 1 tijdreeks zal opgedeeld worden in 20 train- en testreeksen	46
 3.1 Ruwe invoerdata	49
3.2 Grafische weergave jaarlijkse temperatuur	49
3.3 Resultaat van data formatting	50
3.4 Ruwe data van de jaarlijkse ijsdiktes	52
3.5 Grafische weergave van de maandelijkse ijsdiktes	53
3.6 Data van de maandelijks ijsdiktes	53
3.7 Grafische weergave van de ijsdikte en de temperatuur van de laatste 2 jaar	55
3.8 Grafische weergave van de ijsdikte en de temperatuur	55
3.9 Resultaat test stationarity	58
3.10 Resultaat test stationarity na random walk differencing	58
3.11 Resultaat finale iteratie ARIMA	62
3.12 Grafische weergave verschil in MAE's bij verschillende hyperparameters met MAE op de y-as	66
3.13 Resultaat finale iteratie LSTM	68
3.14 Resultaat finale iteratie Prophet	71
3.15 Resultaat van de univariate non-seasonal analyse	73
3.16 Grafische weergave van de laatste voorspellingen van de univariate non-seasonal analyse	74
3.17 Stationariteit van de originele univariate seizoensgebonden data	76
3.18 Stationariteit van de data na seizoendifferentiatie	77
3.19 Stationariteit van de data na random walk differentiatie	77
3.20 Resultaat van ARIMA bij random walk differentiatie	78

LIJST VAN FIGUREN	13
-------------------	----

3.21 Resultaat van ARIMA bij seisoendifferentiatie	79
3.22 Resultaat SARIMAX bij random walk differentiatie	79
3.23 Resultaat SARIMAX bij seisoendifferentiatie	80
3.24 Resultaat LSTM bij seisoendifferentiatie	80
3.25 Resultaten de voorspellingen voor univariate seisoensgebonden data met Prophet	81
3.26 Resultaten van de univariate seisoensgebonden voorspellingen	82
3.27 Grafische weergaven van de laatste voorspellingen van univariate seisoensgebonden data (deel 1)	83
3.28 Grafische weergaven van de laatste voorspellingen van univariate seisoensgebonden data (deel 2)	84
3.29 Grafische weergave multivariate tijdreeks zonder seizoenseffect	85
3.30 Grafische weergave gedifferentieerde multivariate tijdreeks zonder seizoenseffect	86
3.31 Resultaat multivariate voorspelling VARMAX	86
3.32 Resultaat multivariate voorspelling LSTM	88
3.33 Resultaten multivariate voorspelling	89
3.34 Grafische weergaven van de laatste voorspellingen van de VARMAX en LSTM modellen	89
3.35 Resultaat VARMAX bij random walk differentiatie	90
3.36 Resultaat VARMAX bij seisoendifferentiatie	91
3.37 Resultaat LSTM bij random walk differentiatie	91
3.38 Resultaat LSTM seisoendifferentiatie	92
3.39 Resultaten multivariate seisoensgebonden voorspellingsmodellen	93
3.40 Grafische weergaven van de laatste voorspellingen van multivariate seisoensgebonden voorspellingsmodellen	94

1. Inleiding

De hoeveelheid data die men kan verwerven in het digitale tijdperk waarin we de dag van vandaag leven is gigantisch zeker met de opkomst van het internet der dingen beter gekend onder de noemer *Internet of Things*. Deze data kan enorm uiteenlopend zijn, zo wordt onder andere de buitentemperatuur bijgehouden, maar ook de waarde van de aandelen van een bedrijf op de beurs, het aantal bezette plaatsen in een parking, het aantal mensen dat positief getest heeft op het coronavirus, ...

Zo kan het lijstje nog een hele tijd aangevuld worden, maar de net opgenoemde gegevens zijn niet enkel voorbeelden van data. Deze zaken variëren ook nog eens doorheen de tijd. Om dit in contrast te stellen met een ander voorbeeld zou een naam of een postcode van je geboorteplaats niet variëren en dus niet tijdsgebonden zijn. Een sequentie van data die tijdsgebonden is wordt benoemd als een tijdreeks ofwel een *time series*.

Een voorbeeld hiervan zou het aantal dagelijks gewandelde kilometers van het afgelopen jaar zijn. Doordat dit een tijdreeks is zouden we het aantal dagelijks gewandelde kilometers van het volgende jaar kunnen voorspellen met behulp van bepaalde modellen. Daaruit zouden we dan bijvoorbeeld kunnen vaststellen dat er in de winter minder gewandeld zal worden. Het nut hiervan zou dan kunnen zijn dat men inziet dat men te weinig lichaamsbeweging zal hebben en daar dan zal op inspelen door een indoorschport te beoefenen zodat men toch nog voldoende lichaamsbeweging heeft. Bij dit voorbeeld is het verband zeer simpel en kan men zich afvragen waarvoor het opstellen van een model nuttig zou zijn aangezien dit vrij makkelijk af te leiden valt met het blote oog. Maar soms zal het verband een pak complexer zijn dan een seizoen waardoor het moeilijk te vatten valt voor het menselijk brein maar praktischer is om te identificeren door middel van een wiskundig model.

De modellen die onderzocht zullen worden zijn: ARIMA/SARIMAX/VARMAX-modellen, LSTM-modellen en Prophet-modellen. Daarnaast zal ook nog rekening gehouden met het aantal invoerparameters. Zo zullen modellen die gebruik maken van 1 invoerparameter benoemd worden als univariate modellen, modellen die gebruik maken van meerdere invoerparameters daarentegen zullen benoemd worden als multivariate modellen. Deze 2 types modellen kunnen dan nog eens onderverdeeld worden in seizoensgebonden data en niet-seizoensgebonden modellen.

1.1 Probleemstelling

Voorspelde tijdreeksen kunnen zeer belangrijke data zijn om de richting waarin bepaalde beslissingen genomen moeten worden aan te geven. Een brandend actueel voorbeeld hiervan zou een voorspellingsmodel van de coronacijfers zijn wanneer er zo'n model beschikbaar is met een betrouwbaarheid van 100% wat zou er in principe geen discussie meer mogen zijn over de ernst van de situatie en het treffen van de correcte maatregelen zou een pak praktischer zijn. Dit is echter vrij onwaarschijnlijk aangezien er in dit geval tal van factoren een invloed hebben op die cijfers waarvan er velen zeer moeilijk te registeren vallen. Gelukkig zijn er ook problemen waarvan de factoren makkelijker registreerbaar zijn zoals het verminderen van de ijsoppervlakten aan de polen. Zo zal onderzocht worden welk model de beste voorspellingen maakt voor univariate niet-seizoensgebonden tijdreeksen, univariate seizoensgebonden tijdreeksen, multivariate niet-seizoensgebonden tijdreeksen en multivariate seizoensgebonden tijdreeksen. De conclusies en de broncode van dit onderzoek zal kan kunnen dienen als basis voor het opstellen van voorspellingsmodellen van andere tijdreeksen.

1.2 Onderzoeksvraag

Vergelijkende studie van ARIMA-, LSTM- en Prophet-voorspellingsmodellen van univariante en multivariate, seizoensgebonden en niet-seizoensgebonden tijdreeksen.

1.3 Onderzoeksdoelstelling

De doelstelling van deze bachelorproef is om te achterhalen welk model van de volgende drie:

- Autoregressie
- LSTM
- Prophet

de beste resultaten halen voor data van een voorbeeldtijdreeks waarbij een of meerdere

onafhankelijke variabelen beschikbaar zijn en/of een seizoensgebonden verband aanwezig is. Dit onderzoek zal ook kunnen dienen als basis voor het opstellen van een voorspellingsmodel van andere tijdreeksen.

Er dient echter wel vermeld te worden dat deze resultaten altijd zullen variëren van tijdreeks tot tijdreeks en een optimaal model voor de ene tijdreeks niet altijd de beste keuze zal zijn voor een andere tijdreeks.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomain, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeks vragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeks vragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

In dit hoofdstuk zullen de 3 gebruikte voorspellingstechnieken toegelicht worden.

2.1 Regressieanalyse

Regressieanalyse is een voorspellende modelleringstechniek die de relatie tussen een afhankelijke en onafhankelijke variabele onderzoekt. Deze techniek kan ook toegepast worden indien er meerdere onafhankelijke variabelen beschikbaar zijn. Dit betekent dus dat er een relatie gezocht wordt tussen de afhankelijke en de onafhankelijke variabelen om de best passende lijn of regressievergelijking op te stellen. Op basis van deze regressievergelijking waarbij de onafhankelijke variabele(n) als invoerwaarde gebruikt worden, zal de afhankelijke variabele voorspeld worden.

In het geval van tijdreeksen zal het tijdsverloop altijd de onafhankelijke variabele zijn bij univariate modellen en een van de invoervariabelen bij multivariate modellen. Er zijn verschillende manieren om aan regressie te doen de voornaamste technieken zijn: lineaire regressie, polynomiale regressie en logaritmische regressie. Deze zullen dan ook verder toegelicht worden.

2.1.1 Lineaire Regressie

De meest eenvoudige vorm van regressie is lineaire regressie. Dit type regressie wordt vaak gebruikt en wordt best toegepast op continue data (**Pant2019**). Dit is data die gelijkmatig stijgt of daalt en zal grafisch weergegeven worden als een rechte. Bij realistische

voorbeelden zullen deze waarden vrijwel nooit op een rechte lijn liggen maar zullen er altijd enkele uitschieters zijn. Indien we dit grafisch zouden weergeven krijgen we een rechte te zien die de gegeven waarden optimaal zal benaderen.

Een voorbeeld van zo'n lineaire data is weergegeven op de tabel 2.1. Op basis van de cijfers valt dit niet echt makkelijk waar te nemen maar op figuur 2.1 is er duidelijk stijgende tendens zichtbaar in de prijs naargelang de oppervlakte van de leefruimte groter is.

Hieronder staat de basisformule voor een lineaire regressie.

$$y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \quad (2.1)$$

Het eerste deel van deze formule meerbepaald de eerste en de tweede term van de optelling, zal de lineaire component weergeven terwijl de resterende derde term rekening zal houden met de willekeurige fout.

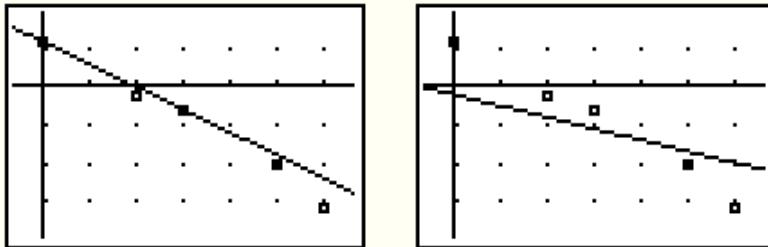
De lineaire component is in principe een eerstegraadsfunctie die de basis zal vormen voor de formule. De y_i zal de afhankelijke variabele weergeven, dit zal dus de te voorspellen waarden weergeven. β_0 zal het intercept weergeven ofwel de waarde van de afhankelijke variabele wanneer de onafhankelijke variabele nul is. Of eenvoudigweg de waarde van de functie waarbij de curve de y-as snijdt. β_1 wordt benoemd als de richtingscoëfficient, deze zal bepalen hoe sterk de rechte stijgt of daalt. Een hogere richtingscoëfficient zal tot een steilere curve leiden terwijl een richtingscoëfficient die 0 benadert vlakker zal zijn, wanneer deze waarde effectief 0 is zal de afhankelijke waarde constant zijn de onafhankelijke variabele die weergegeven wordt door X_i heeft hier dus geen invloed op aangezien deze geneutraliseerd wordt door de 0. Indien de richtingscoëfficient negatief is zal deze curve ook dalen in plaats van stijgen.

De Random Error component dient om alle ruis te vatten. Ruis zijn eigenlijk willekeurige afwijkingen die een verband zouden beïnvloeden waardoor een curve van reële data vrijwel nooit perfect lineair zal zijn. Wanneer deze modellen in de praktijk toegepast worden wordt hier echter vrijwel geen rekening mee gehouden. Dit aangezien verondersteld wordt dat deze afwijkingen elkaar zullen neutraliseren en zo zou de gemiddelde afwijking teweeggebracht door ruis in de curve 0 zijn. Dus ε_i zal een factor zijn die de ruis weergeeft in de formule van lineaire regressie maar valt te verwachten in deze context.

Wanneer we de onafhankelijke variabelen invoeren in de formule zouden we dus de afhankelijke variabelen moeten kunnen voorspellen. Om deze formule in te vullen moeten echter waarden ingevoerd worden voor β_0 en β_1 . De meest optimale waarden voor β_0 en β_1 zullen dus waarden zijn die er voor zorgen dat de curve zo nauw mogelijk aansluit bij de data zelf. Dit zal dan ook betekenen dat de opgetelde afstand tussen de datapunten en de curve die onze lineaire regressie weergeeft zo klein mogelijk zal zijn. Deze methode wordt ook wel benoemd als *the sum of the least squares*. Om dit grafisch even voor te stellen valt op te merken dat in figuur 2.1 de eerste curve nauwer aansluit bij de data dan de tweede en we kunnen ook wel vaststellen dat de som van de afstanden tussen de datapunten en de curve bij de eerste figuur lager ligt.

Hierdoor kan dus gesteld worden dat door β_0 en β_1 te variëren en te kijken welke parametercombinatie voor de curve zal zorgen waarvan de afstand van de datapunten en die curve

Figuur 2.1: Grafische weergave sum of least squares (Brown, 2020)



het laagst is, bepaald kan worden welke waarden voor β_0 en β_1 optimaal zijn.

De accuraatheid van een curve kan dus weergegeven aan de hand van formule 2.2.

$$\text{Error} = \sum (\text{actual output} - \text{predicted output})^2 \quad (2.2)$$

Nu rijst de vraag hoe de parameters zodanig kunnen evolueren zodat ze de kleinst mogelijke foutmarge benaderen. Dit wordt gerealiseerd aan de hand van gradient descent een uitgebreide uitleg van gradient descent komt aan bod bij het bespreken van de LSTM techniek maar het komt er op neer dat gradient descent er zal voor zorgen dat de optimale parameters zo nauwkeurig mogelijk bepaald zullen kunnen met een zo laag gebruik van middelen.

Tabel 2.1: Voorbeelddata voor lineaire regressie (Pant2019)

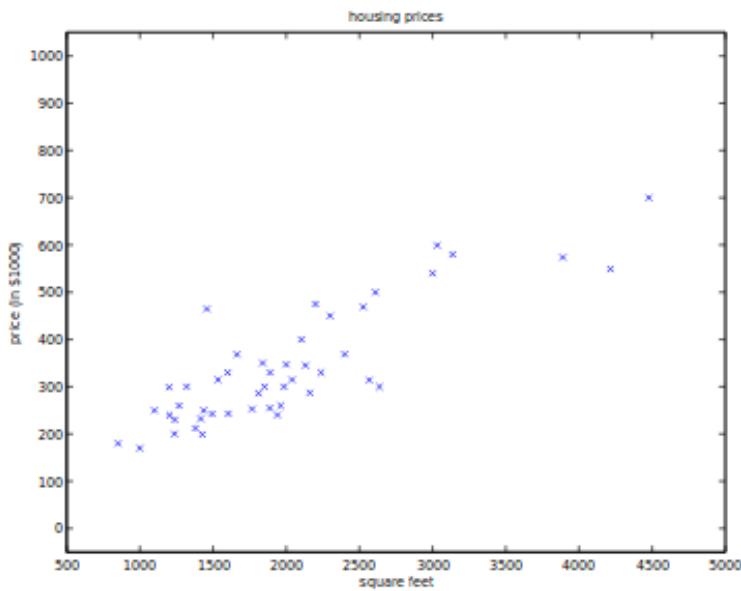
Living area (feet ²)	Price (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540

2.1.2 Polynomiale Regressie

Soms zal de foutmarge bij data hoog blijven bij lineaire regressie ook al is er een verband zichtbaar, dit verband zal dan waarschijnlijk niet te vatten vallen onder een gewone rechte. In zo'n geval kan polynomiale regressie hulp bieden, dit zal een complexere functie aan de data proberen fitten. Formule 2.3 is dan formule van een 2^{de} graadsfunctie

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2^2 \quad (2.3)$$

Dit zorgt ervoor dat de algemene formule herleid wordt naar de formule 2.4 indien we een polynomiale vergelijken wensen toe te passen bij de regressieanalyse.

Figuur 2.2: Grafische weergaven voorbeelddata voor lineaire regressie (**Pant2019**)

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_m x^m + \text{residual error} \quad (2.4)$$

Ook bij polynomiale regressie wordt gebruik gemaakt van gradient descent om de minimale kost te berekenen aangezien dit in principe een uitbreiding is op lineaire regressie.

Dus polynomiale regressie zal de beste gemiddelde inschatting kunnen maken van de relatie tussen de afhankelijke en onafhankelijke variabelen. De curve die gefit moet worden zal zeer complexere vormen dan een simpele rechte kunnen aannemen en meer dan enkel lineaire verbanden kunnen deduceren uit de data. Het grootste nadeel van polynomiale regressie is dat enkele uitschieters de resultaten sterk kunnen beïnvloeden terwijl er bij lineaire regressie meer validatietechnieken beschikbaar zijn voor het achterhalen van uitschieters (**Pant2019**).

2.1.3 Logistische Regressie

Logistische regressie is gelijkaardig aan lineaire regressie maar in tegenstelling tot lineaire regressie zal dit voornamelijk gebruikt worden voor classificatieproblemen, waarbij dus geen continue waarden dienen voorspeld te worden maar nominale waarden. Onder zijn meest simpele vorm zal dit type algoritme nuttig zijn bij het voorspellen van binaire waarden bijvoorbeeld indien men al dan niet aan een aandoening zal lijden. Dit wordt benoemd als binaire logistische regressie. Wanneer er 3 of meer categoriën die dienen voorspeld te worden waarvan de ene niet duidelijk hoger of lager is dan de andere zoals bijvoorbeeld type huisdier wordt dit benoemd als multinomial logistic regression. Indien er wel een logische ordening in zit zoals een aantal sterren op een filmbeoordeling wordt dit benoemd als ordinale logistische regressie. Aangezien in deze paper enkel continue

waarden voorspeld zullen worden, zal hier niet dieper op in gegaan worden (Swaminathan, 2018).

2.1.4 Meervoudige lineaire regressie

Indien er meerdere onafhankelijke variabelen beschikbaar zijn die een lineaire invloed zouden hebben op de afhankelijke variabele kunnen deze ook in rekening gebracht worden. Dit model zal dan opgesteld worden aan de hand van meervoudige lineaire regressie ofwel multiple linear regression. Formule 2.5 geeft weer hoe de onafhankelijke waarden bij meervoudige lineaire regressie berekend zullen worden.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_n x_{in} + \varepsilon_i \quad (2.5)$$

Net zoals bij enkelvoudige lineaire regressie zal y_i de afhankelijke variabele weergeven en β_0 het intercept. Het verschil zit hem in de termen die volgen want aangezien er rekening gehouden wordt met meerdere onafhankelijke variabelen, zullen ook deze verwerkt worden in de formule door middel van een waarde te voorzien voor elke onafhankelijke variabele weergegeven door X_i met bijhorend intercept weergegeven door β . De n in de formule geeft dan het aantal onafhankelijke variabelen weer. De ε_i zal ook in dit geval een factor zijn die rekening zal houden met de meetfouten (Kenton, 2020).

2.1.5 Toepassing

Deze regressietechnieken zullen dienen als basis voor andere meer uitgebreide technieken zoals ARIMA maar de verbanden die ze weergeven zijn te eenvoudig om effectief te gebruiken voor het opbouwen van modellen om aan extrapolatie te doen. (Sinha, 2019) Daarom zullen de klassieke regressiemethodes niet toegepast worden bij deze bachelorproef.

2.2 ARIMA

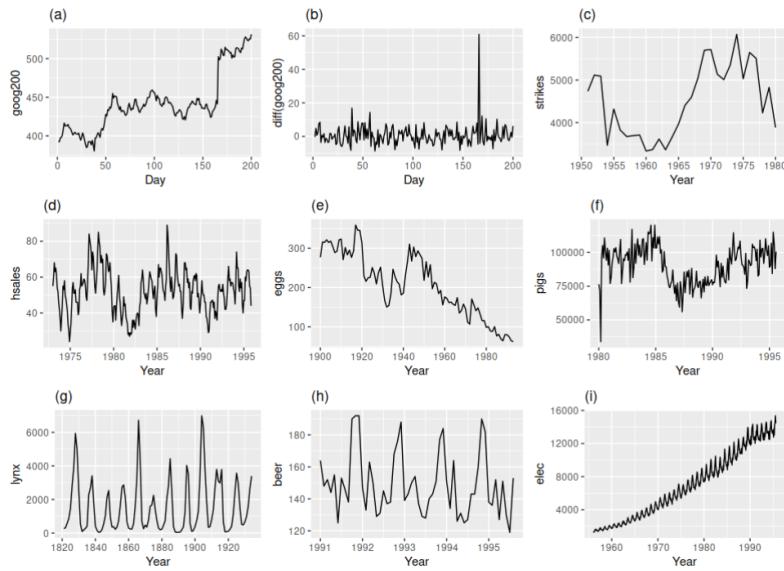
ARIMA is een modeleringstechniek die voorspellingen maakt op basis van stationaire en gedifferentieerde data. Daarom worden deze begrippen eerst toegelicht alvorens de techniek zelf te bespreken.

2.2.1 Stationariteit

Een stationaire tijdreeks is een tijdreeks waarvan de eigenschappen niet afhankelijk zijn van het tijdstip waarop de reeks wordt geobserveerd (Rob J Hyndman, 2018). Indien er een duidelijke trend dus een algemene stijging of daling aanwezig is zal de tijdreeks niet

stationair zijn. Ook wanneer er een seizoensverband aanwezig is die lijkt op een sinus of cosinusfunctie wordt de data als niet stationair beschouwd. Wanneer er echter een cyclisch verband aanwezig is kan er een speciaal type ARIMA model gebruikt worden dat rekening zal houden met deze seizoenscomponent genaamd SARIMA deze zal zodadelijk toegelicht worden. In figuur 2.3 zijn enkele voorbeeldtijdreeksen zichtbaar dit toont aan dat het niet zo evident is om te bepalen welke tijdreeks stationair is.

Figuur 2.3: Voorbeeldtijdreeksen (Rob J Hyndman, 2018)



De cijfers in de figuur omschrijven de volgende data:

- (a) De aandeelkoersen van Google voor 200 opeenvolgende dagen
- (b) De dagelijkse veranderingen in de aandeelkoersen van Google voor 200 opeenvolgende dagen
- (c) Het jaarlijkse aantal stakingen in de Verenigde Staten
- (d) Maandelijkse verkoop van nieuwe gezinswoningen in de Verenigde Staten
- (e) Jaarlijkse prijs van een dozijn eieren in de Verenigde Staten (rekening houdend met inflatie)
- (f) Maandelijks aantal varkens die geslacht worden in Victoria, Australië
- (g) Jaarlijks aantal lynxen die opgesloten worden in het McKenzie River district van noordwest Canada
- (h) Maandelijkse Australische bierproductie
- (i) Maandelijkse Australische elektriciteitsproductie

2.2.2 Seizoenseffect

Bij de figuren (d), (h) en (i) is er een duidelijk seizoenseffect zichtbaar. Er vallen duidelijke trends waar te nemen op figuren (a), (c), (e), (f) en (i). Ook in figuur (g) lijkt er een duidelijk seizoeneffect te zijn, maar deze veranderingen zijn aperiodisch dus op lange termijn valt dit niet te voorspellen aldus is deze tijdreeks stationair.

2.2.3 Differentiatie

Aan de hand van deze grafieken kan ook een ander begrip toegelicht worden dat een belangrijk gegeven is bij het opstellen van een ARIMA model namelijk differentiatie. Zo wordt op figuur (a) de aandeelkoers van google weergegeven voor 200 opeenvolgende dagen, maar hier is een duidelijk stijgende trend zichtbaar. Hierdoor kan de data niet op deze manier geïnterpreteerd worden door een ARIMA model. Maar we kunnen de data op een andere manier voorstellen namelijk door de koerswisselingen zoals weergegeven wordt op figuur (b) waardoor er geen trend meer aanwezig zal zijn en nog steeds geen zichtbare seizoensgebondenheid aanwezig is.

Random walk differentiatie

Bij random walk differencing wordt de data gedifferenstieerd door het verschil te nemen tussen een tijdstap en de tijdstap die ervoor komt. Dit zal de evolutie van de tijdreeks weergeven en aangeven hoeveel een waarde precies verschilt van de tijdstap ervoor. Deze differentiatie kan uitgedrukt worden als

$$y'_t = y_t - y_{t-1}. \quad (2.6)$$

De gedifferentieerde reeks zal 1 waarde minder hebben dan de originele serie omdat er geen waarde voor de eerste y'_1 komt om deze te differentiëren.

Wanneer de gedifferentieerde reeks ruis is kan het model uitgedrukt worden als

$$y_t - y_{t-1} = \epsilon_t, \quad (2.7)$$

Hierbij zal ϵ_t de ruis weergeven. Door deze formule te hervormen bekomen we het "random walk model".

Deze modellen worden vaak gebruikt voor niet-stationaire data, voornamelijk bij financiële en economische toepassingen.

De voorspellingen van dit model zijn gelijk aan dit de laatste observatie.

Tweedegraadsdifferentiatie

In sommige gevallen zal gedifferencieerde data nog steeds niet stationair zijn dan is het eventueel mogelijk om de gedifferentieerde nogmaals te differentiëren. Dit kan als volgt geformuleerd worden.

$$\begin{aligned}
 y_t^n &= y'_t - y'_{t-1} \\
 &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\
 &= y_t - 2y_{t-1} - y_{t-2}
 \end{aligned} \tag{2.8}$$

Op deze manier kan een verandering in de veranderingen gedetecteerd worden. Uiteraard zou men nog een stap verder kunnen gaan maar dit blijkt in praktijk zo goed als nooit nodig te zijn.

Seizoensdifferentiatie

Een laatste manier om data te differentiëren is seizoensdifferentiatie. Hierbij zal telkens het verschil genomen worden van een tijdstap en diezelfde tijdstap van de vorige seizoenscyclus. Dit kan als volgt weergegeven worden,

$$y'_t = y_t - y_{t-m}. \tag{2.9}$$

Hier zal m slaan op het aantal observaties in een seizoen. Indien deze data, die seisoendifferentiëerd is, ruis blijkt te zijn dan zou een geschat model voor de originele data kunnen geformuleerd worden als

$$y_t - y_{t-m} = \varepsilon_t, \tag{2.10}$$

Om seizoensdifferentiaties en gewone differentiaties uit elkaar te houden worden gewone differentiaties ook benoemd als eerste differentiaties.

Het is mogelijk dat deze verschillende soorten differentiaties gecombineerd moeten worden om tot een stationaire dataset te komen. Een dataset waarbij het waarschijnlijk nodig zal zijn om eerste differentiaties en seizoensdifferentiaties gecombineerd moeten worden is die van de maandelijkse Australische elektriciteitsproductie zichtbaar op figuur 2.3 (i). Aangezien er een seizoenscyclus en een stijgende trend aanwezig is.

2.2.4 Toeliching ARIMA

ARIMA is een acroniem gevormd door AutoRegressive Integrated Moving Average wat vrij vertaald kan worden als het autoregressie geïntegreerde voortschrijdend gemiddelde. Deze benaming geeft ook onmiddellijk de sleutelaspecten van dit model weer.

Zo staat autoregressief voor het modelleren van een volgende stap in een sequentie als een lineaire functie op basis van de voorgaande waarden in deze sequentie.

Daarnaast is ARIMA ook een geïntegreerd model wat inhoudt dat niet de cumulatieve waarden gebruikt worden om voorspellingen te maken maar er gekeken wordt naar het

verschil tussen die cumulatieve waarden.

Ten slotte wordt ook gekeken naar het voortschrijdend gemiddelde in plaats van naar de ruwe data zelf dit zal ervoor zorgen dat extreme waarden veroorzaakt door anomaliteiten geneutraliseerd worden. Hierdoor zal de globale trend beter waarneembaar zijn.

ARIMA modellen zijn theoretisch gezien, de meest algemene klasse van modellen om een tijdreeks te voorspellen die stationair/stationary gemaakt kan worden.

Elk van deze componenten is geparameteriseerd en kan dus aangepast worden deze parameters worden benoemd als ($p, d & q$)

p : Het aantal verlate observaties die worden opgenomen in het model. Deze worden in het Engels benoemd als lag observations of lag order.

d : Zal bepalen hoeveel keer ruwe observaties van elkaar afgetrokken worden dit kan ook benoemd worden als de mate van differentiatie of the degree of differencing in het Engels.

q : Deze parameter zal het aantal tijdstappen waarvan het voortschrijdend gemiddelde genomen wordt aangeven. Deze wordt in het Engels benoemd als the order of moving average.

Een lineair regressiemodel wordt geconstrueerd met data waarbij er een graad van differentiatie is zodat deze stationair is. Dit houdt in dat trend en seizoensstructuren die het regressiemodel op een negatieve manier beïnvloeden verwijderd worden.

Bij elke parameter kan ook de waarde 0 opgegeven worden zodat dit aspect horende bij die parameter buiten beschouwing gelaten wordt en dit model effectief gebruikt kan worden als gelijk welke combinatie zoals bijvoorbeeld een autoregressief en een geïntegreerd model al dan niet gebruikmakend van het voortschrijdend gemiddelde (Brownlee, 2018a).

d zal dus de differentiatiegraad weergeven wanneer we hier rekening mee houden kunnen we stellen dat de formule voor ARIMA er als volgt uit zal zien (Lau, 2020):

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} - \dots - \theta_q \varepsilon_{t-q} \quad (2.11)$$

Om de werking van ARIMA te schetsen zullen kort enkele basisparametercombinaties toegelicht worden.

ARIMA(1,0,0)

Dit zal een autoregressief model van de eerste orde zijn ofwel een first-order autoregressive model. Deze parametercombinatie zal gebruikt worden als de reeks stationair en autogecorreleerd is. Dit zou dus voorspeld kunnen worden door de eigen waarde te nemen en deze op te tellen met een constante. De voorspellingsvergelijking zal er als volgt uit zien:

$$\hat{y}_t = \mu + \phi_1 y_{t-1} \quad (2.12)$$

Dit stelt voor dat er regressie uitgevoerd en kan benoemd worden als een ARIMA(1,0,0) model. Wanneer het gemiddelde van y gelijk is aan nul, zal de constante term achterwege gelaten worden. Indien de hellingscoëfficient ϕ_1 positief maar minder dan één is zal het model de waarde van de volgende periode voorspellen als ϕ_1 keer zo ver van het gemiddelde dan de waarde van deze periode.

Indien men een autoregressief model van de 2^{de} orde wil voorstellen zal er bij formule (2.12) rechts een y_{t-2} term toegevoegd worden. Wanneer men een model van nog hogere orde wenst voor te stellen dient men analoog termen toe te voegen. Afhankelijk van de tekens en de grootte van de coëfficiënten zal een ARIMA(2,0,0) model een systeem onderschrijven waarvan de gemiddelde omkering sinusoïdaal oscillerend zal zijn, wat te vergelijken valt met de beweging van massa aan een veer die onderhevig is aan willekeurige uitwijkingen.

ARIMA(0,1,0)

Deze vorm van ARIMA past de meest simpele vorm van differentiatie toe namelijk “random walk”, hierbij zal de eerste differentiatie die zonet besproken is in de sectie differentiatie toegepast worden op de data. Door middel van deze differentiatie kan niet-stationaire data omgevormd kunnen worden naar stationaire data indien dit nog steeds niet het geval is kan deze data in een hogere graad gedifferencierd worden. Dit kan weergegeven onder de vorm van deze formule:

$$\hat{y}_t = \mu + y_{t-1} \quad (2.13)$$

Hier zal de constante term μ de gemiddelde verandering van periode tot periode weergeven dit kan ook benoemd worden als de “long term drift” in y . Aangezien het enkel een niet-seizoensgebonden verschil en een constante term omvat wordt het geklassificeerd als een “ARIMA(0,1,0) model met constante”. Een random walk zonder drift model zou een ARIMA(0,1,0) model zonder constante zijn.

ARIMA(1,1,0)

Deze ARIMA configuratie staat voor een gedifferencierd autoregressief model van de eerste graad. Indien de fouten van een random walk model autogecorreleerd zijn is het mogelijk dat dit verholpen kan worden door een vertraging ofwel lag toe te voegen aan de afhankelijke variabele van de voorspellingsvergelijking. Zo zou dus regressie uitgevoerd worden op het eerste verschil van y en zichzelf, vertraagd met 1 tijdstap. Hierbij zal de vergelijking er als volgt uitzien:

$$\hat{y}_t = \mu + y_{t-1} + \phi_1(y_{t-1} - y_{t-2}) \quad (2.14)$$

ARIMA(0,1,1)

Hier wordt de laatste parameter van ARIMA voor het eerst aangesproken de q . Hierdoor zal er nu gewerkt worden met een voortschrijdend gemiddelde van de data. Dit zal er voor zorgen dat ruis iets meer geneutraliseerd wordt.

$$\hat{y}_t = \mu + y_{t-1} - \theta_1 e_{t-1} \quad (2.15)$$

De constante μ kan eventueel achterwege gelaten worden, deze zal er voor zorgen dat de voorspelling op lange termijn eerder hellend zal zijn hierbij zal het ook mogelijk zijn om een negatieve coëfficient toe te laten. Deze constante zorgt er ook voor dat het ARIMA model een SES model wordt. Indien hier niet voor gekozen wordt zal de voorspelling op lange termijn vlak zijn. Zonder constante wordt dit type model benoemd als simple exponential smoothening indien er een constante gebruikt wordt hier growth aan toegevoegd, wat wijst op de stijgende trend op lange termijn waarvan de helling gelijk is aan μ .

ARIMA(0,2,1) of ARIMA(0,2,2)

Deze configuraties zonder constante worden ook benoemd als lineaire exponentiële afvlakking. Hier zal de 2^{de} afgeleide genomen worden omwille van de waarde voor d .

$$\hat{y}_t = 2y_{t-1} - y_{t-2} - \theta_1 e_{t-1} - \theta_2 e_{t-2} \quad (2.16)$$

De θ_1 en θ_2 zullen staan voor de MA(1) en MA(2) coëfficienten. Dit zal overeenkomen met Holt's model en in speciale gevallen met Brown's model. Het zal gebruik maken van exponentiële gewogen voortschrijdende gemiddeldes om het lokale niveau en lokale trend (local level and local trend) te schatten. Voorspellingen op lange termijn zullen naar een rechte lijn toe convergeren wiens helling zal afhangen van de gemiddelde trend naar het einde van de reeks toe.

ARIMA(1,1,2)

Dit wordt benoemd als damped-trend linear exponential smoothing en wordt weergegeven door de formule:

$$\hat{y}_t = y_{t-1} + \phi_1(y_{t-1} - y_{t-2}) - \theta_1 e_{t-1} - \theta_2 e_{t-2} \quad (2.17)$$

Dit model zal de lokale trend aan het einde van de reeks extrapoleren maar zal afvlakken naar het einde toe.

Bij het opstellen van een ARIMA mode wordt aangeraden om een van de parameters p of

q niet groter te laten worden dan 1 aangezien er een hoge waarschijnlijkheid is dat dit zal leiden tot overfitting of andere fouten.

Een manier om de corelaties en een eventueel vertraagd effect te identificeren tussen de waarden rekening houdend met trend, seasonality en het residu is een ACF ofwel een auto-correlation function (**Salvi2020**).

Een aanvulling daarop is een PACF ofwel een partial auto-correlation function deze zal eventuele correlaties wanneer de effecten na eerdere vertragingen verwijderd worden (door middel van ACF) weergeven.

Een goede vuistregel bij het opstellen van een ARIMA model is het verhogen van de p indien er een positieve correlatie is dit houdt in dat de afhankelijke variabele zal stijgen als de onafhankelijke variabele stijgt. Wanneer er een negatieve corelatie is en de onafhankelijke variabele stijgt zal de afhankelijke variabele dalen en in dit geval is het beter om de q te verhogen.

Een stappenplan om ARIMA toe te passen op een dataset zal er als volgt uit zien:

1. Plot de data en identificeer ongewone waarden
2. Transformeer de data om de variantie te stabiliseren indien nodig
3. Als de data niet stationair is neem eerste differentiaties van de data totdat deze stationair is.
4. Onderzoek de ACF/PACF en kijk of een ARIMA($p,d,0$) of ARIMA($0,d,q$) model gebruikt moet worden
5. Test de modellen uit en gebruik AICc (een scoringstechniek) om het model te verbeteren.
6. Bekijk de residuen van het gekozen model door ze te plotten samen met de ACF en een portmanteau test van de residuen te doen. Indien deze er niet uitzien als ruis is het aangeraden een nieuw model te proberen
7. Wanneer de residuen er uitzien als ruis kunnen voorspellingen gemaakt worden

2.2.5 Varianten op ARIMA

SARIMA

In sommige gevallen blijkt er een seisoensgebondenheid aanwezig te zijn in de data. Deze seisoensgebondenheid zal zorgen voor constante fluctuaties. Deze manier van beïnvloeden is iets waar bij een gewoon ARIMA model geen rekening mee wordt gehouden. Daarom is er een variant op het ARIMA model genaamd SARIMA waarbij de toegevoegde S zal staan voor de seisoensgebondenheid. Dit zal in rekening gebracht worden door extra parameters toe te voegen. Zo zullen de parameters bij ARIMA (p,d,q) zijn, terwijl de parameters bij SARIMA $(p,d,q)(P,D,Q)m$ zijn, waarbij m zal staan voor het aantal tijdsstappen binnen een seisoenscyclus. Zo zal bij een jaarlijkse cyclus m gelijk zijn aan 12 als de tijdspanne tussen de observaties 1 maand bedraagt. Daarnaast zal de p staan voor autoregressieve seizoensorde ofwel de seasonal autoregressive order, d

voor de gedifferencieerde seisoensorde ofwel de seasonal difference order en q voor de seisoensorde van het voortschrijdend gemiddelde ofwel de seasonal moving average order (Brownlee, 2018b).

VARMAX

Wanneer er meerdere tijdsreeksen voorspeld moeten worden waarbij er een verband mogelijk zou kunnen zijn tussen de verschillende tijdsreeksen kan de VARMAX techniek gebruikt worden. Hierbij zal de V staan voor vector wat duidt op het interpreteren en voorspellen van vectoren. Dit type model zal dan ook gebruikt worden bij het voorspellen van multivariate of meervoudige tijdsreeksen AR wijst andermaal op autoregressief. MA slaat ook weer op het voortschrijdend gemiddelde. De X in VARMAX staat voor de invloed van exogene factoren op de factoren die voorspeld moeten worden.

2.3 Long Short Term Memory

Als tweede methode die gebruikt zal worden om tijdsreeksen te voorspellen wordt gekozen voor een recurrent neuraal netwerk met gebruik van Long Short Term Memory modellen wat afgekort wordt als LSTM.

Een recurrent neuraal netwerk zal uitgebreid toegelicht worden in de volgende secties maar dit kan kort omschreven worden als een structuur die gebruik maakt van trainingsdata om te “leren” en op basis van dit leerproces voorspellingen kunnen maken van nieuwe data. Om dit met een simpel voorbeeld te schetsen zou een neuraal netwerk net zoals wij als mens leren dat er bij veel bewolking kans zal zijn op regen. Om dit te realiseren zullen alle parameters echter numeriek doorgegeven moeten worden aan het neuraal netwerk in dit geval zou een soort van bewolkinggraad aangewezen zijn. Op basis daarvan zal het neuraal netwerk dan een kans op regen kunnen aangeven.

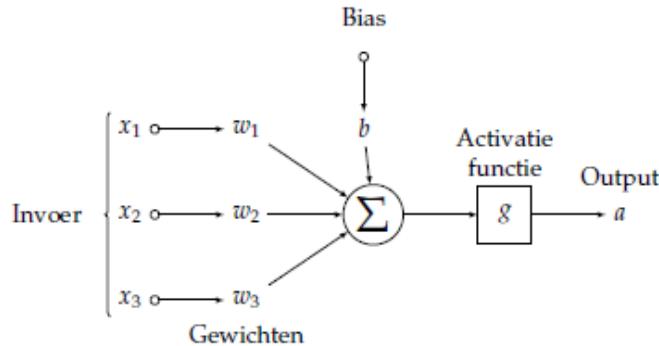
Een LSTM is een type recurrent neuraal netwerk dat gebruikt wordt bij sequentiële voorspellingsproblemen. Dit betekent dat de volgorde van de data een rol zal spelen en dat bij het voorspellen van een waarde in een reeks rekening zal gehouden worden met een aantal waarden die vlak voor de te voorspellen waarde komen.

2.3.1 Theoretische toelichting

Een neuraal netwerk valt makkelijkst te vergelijken met een menselijk brein. Zo zullen wij bijvoorbeeld het cijfer 1 herkennen aan zijn vormen. Door de verschillende pixels, gebruikt voor het weergeven van ons cijfer 1 als inputwaarden voor een neuraal netwerk te gebruiken en de outputwaarden gelijk te stellen met cijfers 0 tot en met 9 kan het netwerk aangeleerd worden deze cijfers te herkennen.

Dit gebeurt niet vanzelf maar door een combinatie van eenvoudige rekeneenheden die

Figuur 2.4: Voorstelling van een neuron die invoer ontvangt van 3 andere neuronen (Lievens, 2018)



verbonden zijn en waartussen er verbindingen zijn die beschreven worden met een reëel getal dat de sterkte van een verbinding aangeeft (Lievens, 2018). Zo'n combinatie van eenvoudige rekeneenheden die ook benoemd worden als neuronen wordt net zoals bij het menselijk brein een neuraal netwerk genoemd dit wordt weergegeven op figuur 2.4.

Zo bevinden zich tussen de input en de outputlaag verschillende lagen die benoemd worden als verborgen lagen. In deze lagen bevinden zich knooppunten (neuronen) waarin waarden worden berekend die een interpretatie geven van de inputwaarden waarmee ze verbonden zijn. Wanneer we dit toepassen op ons voorbeeld zou in een knooppunt bijvoorbeeld kunnen bekijken worden of er in het midden van de te interpreteren pixels een lijn staat. Wanneer dit het geval zou zijn zou er in dit neuron een grote waarde weergegeven worden en zal deze een sterk signaal sturen. Wanneer we voor dit voorbeeld met 1 verborgen laag zouden werken zou het neuron dat een sterk signaal zendt bij het herkennen van een centrale verticale lijn een belangrijke waarde zijn voor het herkennen van een 1 en een 4. De sterke van dit signaal zal dus een grote invloed hebben op de kans dat het resultaat een 1 of een 4 is. Wanneer we hier dieper op ingaan kunnen we aan elk neuron een formule toewijzen die er zo uitziet:

$$n = w_1 i_1 + w_2 i_2 \quad (2.18)$$

W staat voor de gewichten, i staat voor de inputwaarde. Stel dat bij dit neuron de bovenste 3 pixels genomen worden en we voor de eenvoud werken met een 3×3 afbeelding, dan zal de formule er als volgt uitzien:

$$n = w_1 i_1 + w_2 i_2 + w_3 i_3 \quad (2.19)$$

n zal dan staan voor gewogen gemiddelde, door hier een bias aan toe te voegen b net zoals bij lineaire regressie en dit als invoer te gebruiken voor een activatiefunctie g zal de sterkte van het uitvoersignaal bekomen worden.

Indien een neuron effectief geactiveerd wordt zal bepaald worden door de volgende

formule.

$$a = g(n + b) \quad (2.20)$$

Een bias b zal bij een neuraal netwerk zorgt voor een correcte neutrale toestand binnen een neuraal netwerk wanneer alle features nul zijn net zoals bij lineaire regressie.

Door het gebruik van een activatiefunctie g zal dit netwerk ook meer beginnen lijken op de werking van neuronen zoals ze voorkomen in de natuur.

Een activatiefunctie zal gebruikt worden om voor non-lineariteit te zorgen binnen het model. Wanneer we de activatiefunctie zouden laten wegvalLEN zou het volledige neurale netwerk kunnen herleid worden tot één lineaire functie waardoor er enkel weer lineaire verbanden gelegd kunnen worden. Het toevoegen van een activatiefunctie zal er voor zorgen dat de lineariteit gebroken wordt en er binnenin het netwerk nieuwe, abstracte features gegenereerd kunnen worden die kunnen resulteren in een betere interpretatie van de invoerdata.

Toepassing

Bij dit voorbeeld zijn we nog steeds op zoek naar een 1 en kennen een hoge waarde toe aan donkere kleuren opdat het model de afbeelding zou kunnen interpreteren. Dit zou betekenen dat i_1 en i_3 een lage waarde moeten hebben en i_2 een hoge waarde. In dit geval zullen de gewichten voor w_1 en w_3 gelijkgesteld worden aan -1 en het gewicht voor w_2 aan 1. Daardoor zal n een hoge waarde krijgen als de centrale pixel zwart is en de buitenste pixels wit. Dit wordt de propagation function genoemd. Om deze waarden om te zetten naar een waarde die interpretabel is doorheen het volledige model wordt het resultaat van de propagation function nog eens gebruikt als invoerwaarde voor een activatiefunctie. In dit geval zou de hyperbolic tangent (tanh) een goede keuze zijn, maar dit hangt af van de toepassing. Bij dit voorbeeld zou dat signaal dan naar de outputlaag gestuurd worden die op zijn beurt ook weer diezelfde formule toepast maar dan niet met de invoerwaarden maar met de resultaten uit de verborgen laag. Hieruit zal dan een score komen die aangeeft hoe waarschijnlijk het is dat de waarden die de pixels weergeven uit de invoerlaag het cijfer vormen dat toegekend is aan dit neuron in de uitvoerlaag.

Deze toepassing waarbij een cijfer wordt herkend is een voorbeeld van een toepassing van voorwaards gerichte neurale netwerken omdat de data slechts in 1 richting verloopt, een resultaat wordt niet beïnvloed door voorgaande inputwaarden. Bij recurrente neurale netwerken is dit wel het geval. Hierbij wordt rekening gehouden met de beoordelingen van voorgaande inputwaarden een voorbeeld hiervan is het beoordelen van zinnen de volgorde van de ingevoerde woorden zal een rol spelen. Zo zal ‘eet ik’ geïnterpreteerd kunnen worden als een vraag aan de hand van de woordanalyse. Terwijl ‘ik eet’ zal beoordeeld worden als een statement.

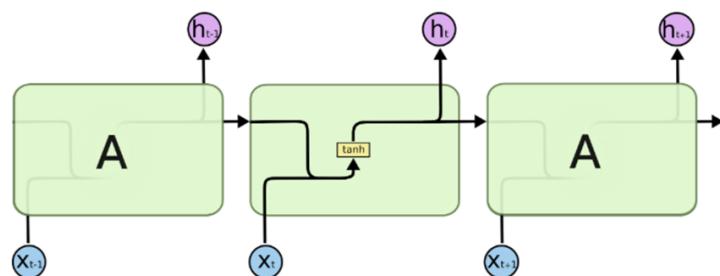
Dit zal louter gebeuren op basis van de woordvolgorde waarbij connecties tussen verschillende cellen teruglopend kunnen zijn onder elkaar en de vorige inputwaarden dus een

invloed zullen hebben op de beoordeling van de volgende inputwaarden (Lievens, 2018).

Dit is dus een gelijkaardige werking aan voorwaarts gerichte neurale netwerken maar de signalen verlopen niet allemaal in dezelfde richting binnen het netwerk. Zo kan er een signaal van de 2^{de} verborgen laag teruggestuurd worden naar de 1^{ste} verborgen laag. Op die manier hebben de waarden van vorige iteraties een impact op de volgende resultaten en dit is wat we willen bereiken.

Een lus van eenzelfde stuk uit het neurale netwerk die informatie van vorige iteraties doorgeeft aan zichzelf dit wordt weergegeven op Figuur 2.5.

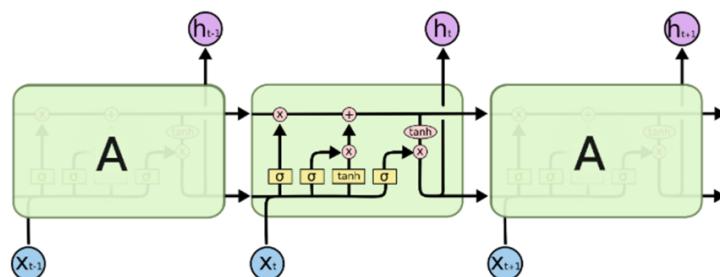
Figuur 2.5: Cel in een neurale netwerk dat gegevens van een andere cel gebruikt om een nieuw signaal uit te sturen (Olah, 2015)



Om het verloop van een tijdreeks te voorspellen moet ook rekening gehouden worden met het tijdsverloop. Zo zal het verloop van de vorige dagen een invloed hebben op het verloop van de volgende dagen. Dus voor deze toepassing zullen recurrente neurale netwerken gebruikt worden aangezien voorwaarts gerichte neurale netwerken geen rekening houden met de voorgaande inputwaarden.

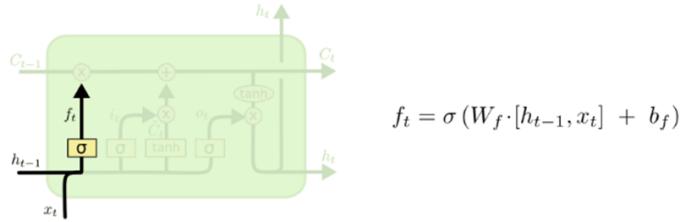
Het probleem bij gewone recurrente neurale netwerken is dat deze moeilijk patronen herkennen op lange termijn. Hiervoor biedt de long short term memory variant van recurrente neurale netwerken een oplossing. Dit zorgt ervoor dat enkel de data die bijgehouden moet worden uit vorige iteraties een impact zal hebben op nieuwe resultaten. De werking van een LSTM wordt voorgesteld op figuur 2.6.

Figuur 2.6: Gedetailleerdere figuur van een cel uit een neurale netwerk dat informatie interpreteert uit een andere cel van het neurale netwerk (Olah, 2015)

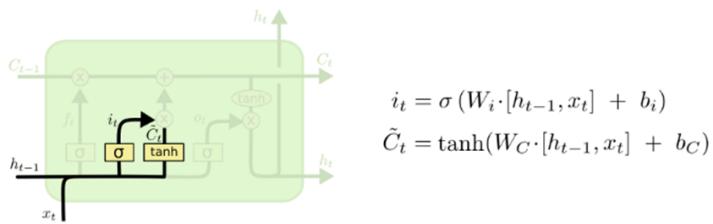


Dit aan de hand van 4 stappen:

Figuur 2.7: Grafische weergave van de forget layer binnen het neuron (Olah, 2015)



Figuur 2.8: Grafische weergave van de input gate layer binnen het neuron (Olah, 2015)

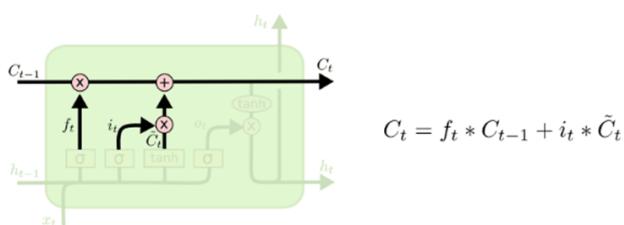


Stap 1 Eerst wordt bepaald welke info weggegooid mag worden in de “forget layer”. Deze zal een waarde tussen 0 en 1 uitvoeren waarbij een 0 zal betekenen dat deze waarde volledig genegeerd mag worden voor elke component van de cell state en waarbij een 1 zal betekenen dat alles behouden zal moeten worden.

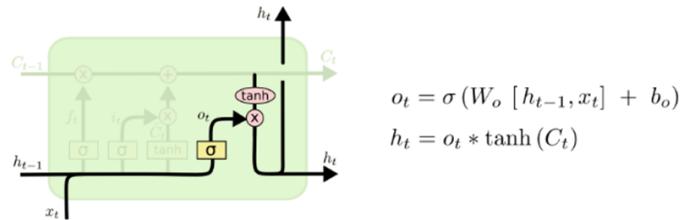
Stap 2 Voor de volgende stap zal in de “input gate layer” bepaald worden welke waarden geüpdateert moeten worden aan de hand van een sigmoïde funtie die hier zal resulteren in i_t . Daarna zal de vector geconstrueerd worden met de potentiële waarden \tilde{C}_t .

Stap 3 In deze stap zullen we de voorgaande celwaarde vernieuwen. Zo zullen door C_{t-1} te vermenigvuldigen met f_t de overbodige waarden vergeten worden. En door het product van i_t en \tilde{C}_t hierbij op te tellen zullen de achterhalde waarden geüpdateet worden.

Stap 4 Als laatste zal de uitvoer bepaald worden. Dit is een gefilterde versie van de celtoestand. Die filtering gebeurt aan de hand van een sigmoïde functie. Het resultaat hiervan wordt dan vermenigvuldigd met de tanh van de celtoestand. Met deze formule

Figuur 2.9: Grafische weergave van het 2^{de} deel van de input gateway (Olah, 2015)

Figuur 2.10: Grafische weergave van het onderdeel van de cel waarin de input vermeigvuldigd wordt met de tanh van de celtoestand om de outputwaarde te bekomen (Olah, 2015)



verkrijgen we de uitvoerwaarde die de cyclus kan verderzetten.

2.3.2 Praktische toelichting

Deze praktische toelichting is gebaseerd op het artikel van Jason Brownlee (Brownlee, 2018c) en zal dienen als basis voor het opstellen van een LSTM netwerk.

Univariate LSTM modellen

Univariate LSTM models maken voorspellingen voor data waarbij er slechts 1 afhankelijke variabele is. Hierbij bestaan de problemen uit een reeks observaties. Op basis van vorige waarden in een reeks zal de volgende waarde in diezelfde reeks voorspeld worden. In deze subsectie zullen de voorgestelde modellen toegepast worden op eenstaps univariate tijdreeksen maar deze kunnen makkelijk aangepast worden om gebruikt te worden bij andere soorten tijdreeksen.

Datavoorbereiding Een tijdreeks zelf bestaat uit een sequentie van waarden zoals hieronder bijvoorbeeld.

Listing 2.1: Originele databasequentie

[10, 20, 30, 40, 50, 60, 70, 80, 90]

Om deze waarden te gebruiken voor het trainen van het neuraal netwerk zal deze tijdreeks omgevormd moeten worden naar samples. De dimensies van deze samples zullen bepalen hoeveel inputwaarden en outputwaarden zullen gebruikt worden bij het model. Aangezien we bij dit voorbeeld met eenstapsvoorspellingen werken zal elk sample over 1 outputwaarde beschikken. Bij dit voorbeeld worden 3 inputwaarden genomen. Dit zal als gevolg hebben dat de samples van de gegeven databasequentie er als volgt zullen uitzien.

Listing 2.2: Samples van de gegeven databasequentie

X,	y
10, 20, 30	40

20 , 30 , 40	50
30 , 40 , 50	60
...	

Vanilla LSTM

Een Vanilla LSTM is een LSTM model dat over één enkele verborgen laag beschikt en een uitvoerlaag die een voorspelde waarde zal aangeven. In deze verborgen laag zullen zich 50 nodes bevinden. De hoeveelheid te kiezen nodes is afhankelijk van de data. Zo zullen meer nodes de foutmarge verkleinen maar een kleiner aantal nodes zal meer gaan generalizeren, wat het eindresultaat ook zou kunnen verbeteren. Uiteindelijk moet hier een evenwicht in gevonden worden.

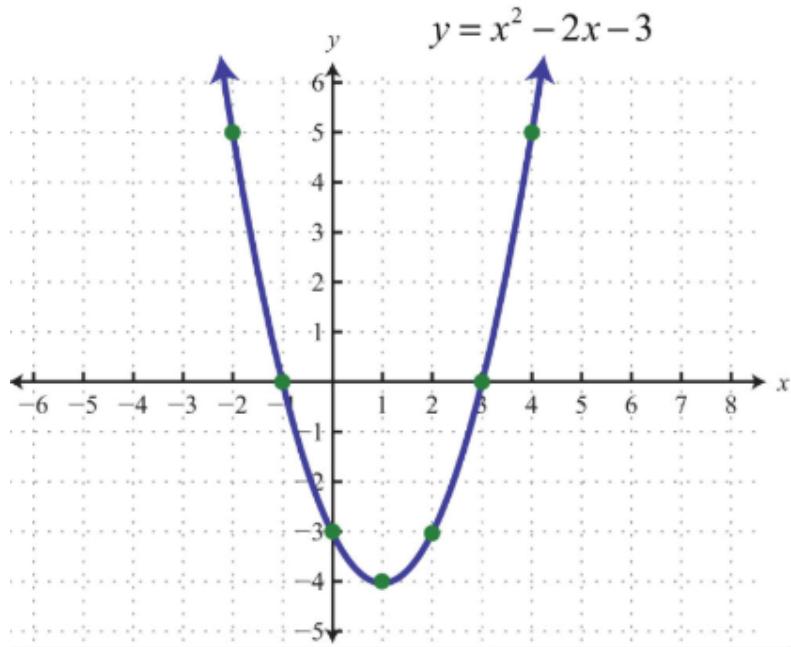
Om te bepalen of een signaal binnен een neuron effectief sterk genoeg zal zijn om te activeren en er zelf een te sturen zal de relu activatiefunctie gebruikt worden. Dit is de meest gebruikte activatiefunctie in neurale netwerken (Liu, 2020) en heeft als voordeel dat het er weinig complexe wiskunde aan te pas komt bij de berekening waardoor het performant is. Daarnaast convergeert het ook sneller. Door de lineariteit zal de helling niet afvlakken wanneer x vergroot. Tenslotte is deze functie ook ijl wat inhoudt dat er altijd iets van activatie is, specifiek bij de relu functie alle waarden onder 0 liggen. Ijlheid komt de efficiëntie van het model ten goede omdat hierdoor bepaalde neuronen enkel geactiveerd zullen worden bij specifieke prikkels. Zo zou een netwerk voor image recognition een bepaald neuron kunnen hebben dat zal activeren indien er een kattenoor waar te nemen valt, maar het zou niet wenselijk zijn moest dit neuron actief zijn indien er een afbeelding van een gebouw wordt getoond.

De gewichten van relaties tussen de neuronen in het neuraal netwerk zullen bepaald worden aan de hand van de Adam versie van stochastische gradient descent. Om stochastische gradient descent te definiëren zal eerst toegelicht moeten worden wat gradient descent precies inhoudt. Gradient staat voor de afgeleide van de verlies functie en descent betekent afdaling. De combinatie van deze 2 begrippen houdt in dat men zal af dalen tot het laagste punt bereikt wordt (Srinivasan, 2019).

We kunnen dit principe makkelijkst voorstellen aan de hand van een parabool. Deze wordt grafisch weergegeven op figuur 2.11. Het minimum van deze parabolische functie wordt bereikt wanneer x een waarde van 1 heeft. Als startpunt wordt een willekeurige waarde op de curve genomen waarbij dan nagegaan wordt in welke richting bewogen moet worden om dichter bij het minimum te komen. Wanneer het minimum gezocht wordt zou een stapgrootte gedefinieerd kunnen worden en dan telkens met even grote stappen afgedaald kunnen worden. Wanneer deze stapgrootte bepaald moet worden zou bij een kleine stapgrootte het minimum nauwkeuriger gedefinieerd zijn maar zou dit veel meer berekeningen vergen. Bij een grote stapgrootte zou het minimum snel berekend kunnen worden, maar het zou onnauwkeurig zijn.

Bij gradient descent wordt deze stapgrootte dynamisch bepaald op basis van het product van de hellingsgraad en de learning rate, een vooraf bepaalde waarde die moet voorkomen

Figuur 2.11: Parabolische functie



dat de stapgrootte te groot wordt. Wanneer de hellingsgraad laag zal zijn betekent dit dat het minimum wordt benaderd en dus kleinere stappen genomen zullen worden om dit zo nauwkeurig mogelijk te bepalen. De kans dat 0 effectief wordt bereikt is zeer klein, daarom stopt het algoritme wanneer een vooraf bepaalde minimale stapgrootte wordt bereikt. Daarnaast kan ook een maximum aantal stappen ingesteld worden waarbij het algoritme stopt.

Gradient descent kan duidelijk toegelicht worden aan de hand van deze 5 stappen, bij elke stap moet de waarde van de afgeleide opnieuw berekend worden:

1. Neem de afgeleide van de verliesfunctie voor elke parameter in deze functie
2. Kies willekeurige waarden voor deze parameters
3. Voer de parameterwaarden in in de afgeleiden (in praktijk zal de gradiënt numerisch bepaald worden)
4. Bereken de stapgroottes: Stapgrootte = helling * learning rate
5. Bereken de nieuwe parameters: Nieuwe parameter = oude parameter - stapgrootte

Herhaal stappen 3 tot 5 tot het minimum bereikt wordt.

Bij dit voorbeeld zou gewoon de afgeleide van deze parabool kunnen genomen worden om zo tot het minimum te komen, maar bij complexere functies is dit niet mogelijk en biedt gradient descent hiervoor een oplossing.

Bij gebruik van stochastic gradient descent wordt niet telkens alle data gebruikt wanneer de parameters herberekend worden, maar een willekeurige subset uit de dataset om het

aantal berekeningen te reduceren wat deze methode performanter maakt zonder al te veel nauwkeurigheid te verliezen (Starmer, 2019).

Dan rest enkel Adam nog verklaard te worden. Dit is een variant op de klassieke vorm van gradient descent waarbij de learning rate aangepast voor elk gewicht binnen het netwerk en doorheen het trainingsproces.

Deze aanpassingen worden door de auteurs van Adam omschreven als het combineren van 2 andere uitbreidingen op gradient descent namelijk:

Adaptive Gradient Algorithm (AdaGrad) die er voor zorgt dat een per-parameter learning rate die performantie zal verbeteren van de gradiënt.

Root Mean Square Propagation (RMSProp) deze uitbreiding op gradient descent zorgt voor aanpassingen in de learning rates op basis van de schaal van de laatste gradiënten kortom hoe snel deze veranderen (Brownlee, 2017b).

En zal geoptimaliseerd worden op basis van de 'mse' verliesfunctie.

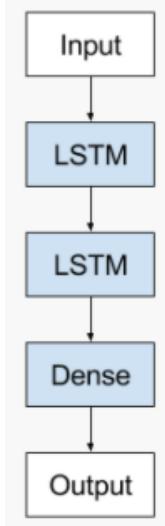
Listing 1: Datavoorbereiding dataset ijsexpansie

```

1 # univariate lstm voorbeeld
2 from numpy import array
3 from keras.models import Sequential
4 from keras.layers import LSTM
5 from keras.layers import Dense
6
7 # Opsplitsen van univariate sequentie in samples
8 def split_sequence(sequence, n_steps):
9     X, y = list(), list()
10    for i in range(len(sequence)):
11        # vinden van het einde van dit patroon
12        end_ix = i + n_steps
13        # nagaan of we ons na de sequentie bevinden
14        if end_ix > len(sequence)-1:
15            break
16        # verwerven van invoer en uitvoerdelen van het patroon
17        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
18        X.append(seq_x)
19        y.append(seq_y)
20    return array(X), array(y)
21
22 # Bepalen van de sequentie
23 raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
24
25 # bepalen van het aantal tijdsstappen
26 n_steps = 3
27
28 # onderverdelen in samples
29 X, y = split_sequence(raw_seq, n_steps)
30
31 # reshape from [samples, timesteps] into [samples, timesteps, features]
32 n_features = 1

```

Figuur 2.12: Een stacked LSTM (Brownlee, 2017c)



```

33 X = X.reshape((X.shape[0], X.shape[1], n_features))
34
35 # definieer model
36 model = Sequential()
37 model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
38 model.add(Dense(1))
39 model.compile(optimizer='adam', loss='mse')
40
41 # fit model
42 model.fit(X, y, epochs=200, verbose=0)
43
44 # demonstreer voorspelling
45 x_input = array([70, 80, 90])
46 x_input = x_input.reshape((1, n_steps, n_features))
47 yhat = model.predict(x_input, verbose=0)
48 print(yhat)
  
```

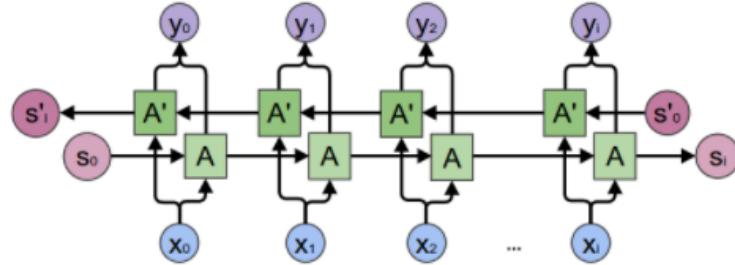
Stacked LSTM

Een stacked LSTM is een variant op het Vanilla LSTM waarbij er meerdere LSTM lagen na elkaar geplaatst worden. Het toevoegen van een extra laag zal voor een langere uitvoeringstijd zorgen. Het voordeel hiervan is dat hierdoor meerdere onderdelen makkelijker geregistreerd kunnen worden en in verband met elkaar gebracht kunnen worden. Het stapelen van LSTM lagen zal ervoor zorgen dat deze lagen ook onderling uitvoerwaarden zullen uitwisselen die een sequentie van tijdsstappen zal beschrijven en niet gewoon 1 enkele tijdstap (Brownlee, 2017c).

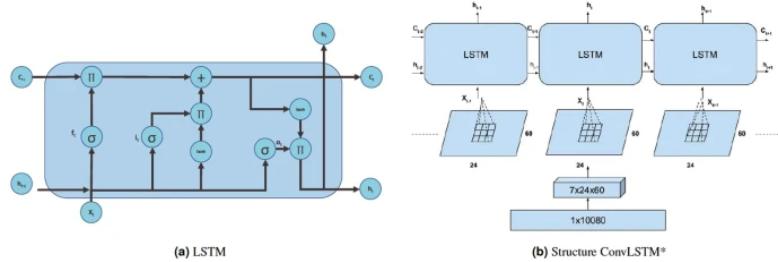
Bidirectionele LSTM

Bidirectionele LSTMs kunnen getraind worden gebruik makend van alle beschikbare invoerinformatie uit het verleden en de toekomst binnen een specifieke tijdspanne. Dit door middel van de toestandsneuronen op te splitsen van een normaal Recurrent Neuraal Netwerk in een deel dat verantwoordelijk is voor de positieve tijdsdirectie en een deel voor de negatieve tijdsdirectie. (Brownlee, 2017a)

Figuur 2.13: Weergave van een bidirectioneel LSTM model (Colah2015)



Figuur 2.14: (a) Gewone LSTM cell, (b) Weergave van een ConvLSTM (Syed Ashiqur Rahman, 2019)



Bidirectionele LSTMs worden voornamelijk gebruikt bij language processing omdat de definitie van een woord afgeleid kan worden uit de context. De begrippen die bepalend zijn voor de definitie kunnen zich zowel voor als achter het woord zelf bevinden dus moet de zin vanuit 2 richtingen geanalyseerd kunnen worden om de definitie te achterhalen.

CNN LSTM

Een convolutioneel neuraal netwerk ook wel een CNN genoemd wordt voornamelijk gebruikt bij het analyseren van tweedimensionale grafische data. Een CNN is ook sterk in het extraheren en aanleren van features van eendimensionale data. Een CNN model kan ook gecombineerd worden met een LSTM model om een CNN-LSTM te vormen. Hierbij wordt het CNN gebruikt om subreeksen te interpreteren die op zich dan weer zullen geïnterpreteerd worden als een reeks door het LSTM model.

ConvLSTM

ConvLSTM is ook een soort van CNN-LSTM waarbij het convolutionele deel rechtstreeks is ingebouwd binnen het LSTM gedeelte. Dus ook dit CNN-LSTM werd ontwikkeld voor het interpreteren van tweedimensionale grafische data maar kan aangepast worden voor gebruik bij univariabele tijdreeksvoorspellingen.

Multivariabele LSTM modellen

Bij multivariabele tijdreeksen zullen er in tegenstelling tot univariabele tijdsreeksen wel meerdere observaties zijn per tijdseenheid. Deze worden dan nog eens onderverdeeld in multiple input series en multiple parallel series.

Meerdere invoerreeksen

Bij de variant met meerdere invoerreeksen zullen er meerdere invoerwaarden zijn voor elk tijdstip. Hierbij zal slechts voor 1 van de invoerreeksen een voorspelling gemaakt worden.

Meerdere parallelle reeksen

Bij de LSTM variant waarbij er meerdere parallelle reeksen zijn zullen er meerdere invoerwaarden zijn voor elk tijdstip. Hier zal er voor elke reeks een waarde voorspeld worden.

Multi-Step LSTM modellen

Tot nu toe werden enkel maar one-step LSTM modellen toegelicht waarbij de invoerreeks slechts zal resulteren in 1 uitvoerwaarde. Bij multi-step LSTM modellen zal er meer dan 1 uitvoerwaarde voorspeld worden.

Vector output model

Bij dit type multi-step LSTM model zal de uitvoer onder de vorm van een vector weergegeven worden die de resultaten van de volgende tijdstappen zal weergeven en niet enkel van de volgende tijdstap.

Encoder-Decoder Model

Dit model is ontwikkeld om sequenties met variabele uitvoer te voorspellen (Brownlee, 2017a). Het is ook ontworpen om problemen waarbij er zowel invoer-als uitvoersequenties zijn te behandelen. Deze problemen worden ook wel benoemd als sequence-to-sequence of seq2seq-problemen. Deze techniek wordt voornamelijk gebruikt bij vertalingstoepassingen maar kan ook toegepast worden op tijdreeksen. Het model bestaat uit 2 hoofdonderdelen de encoder en de decoder.

De encoder is verantwoordelijk voor het lezen en interpreteren van de invoerreeks. De encoder zal dan een vector met een vaste lengte uitvoeren die een interpretatie zal geven van de invoerreeks. Normaal zal de encoder een Vanilla LSTM model zijn, maar er kan even goed een stacked, bidirectioneel of CNN model gebruikt worden.

De decoder zal de uitvoer van de encoder dan gebruiken als invoer. Die zal dan voor elke tijdsstap een uitvoerwaarde zal geven. Deze uitvoer zal dan nog eens geïnterpreteerd worden door een verborgen laag alvorens dit volledig model de multi-step uitvoerreeks zal teruggeven.

Multivariate Multi-Step LSTM Models LSTM modellen

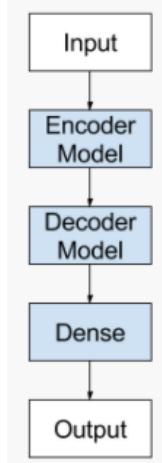
Multiple input Multi-Step Output

Wanneer men over een dataset beschikt met meerdere invoerparameters maar slechts 1 enkele parameter voor meerdere tijdsstappen dient te voorspellen spreekt men over multiple input multi-step output.

Multiple Parallel Input and Multi-Step Output

Wanneer de dataset waarvan een voorspelling gemaakt moet worden meerdere invoerparameters heeft maar er ook meerdere voorspeld moeten worden zal dit benoemd worden als een multiple parallel input and multi-step output LSTM.

Figuur 2.15: Encoder-Decoder Model (Brownlee, 2017a)



2.4 Prophet

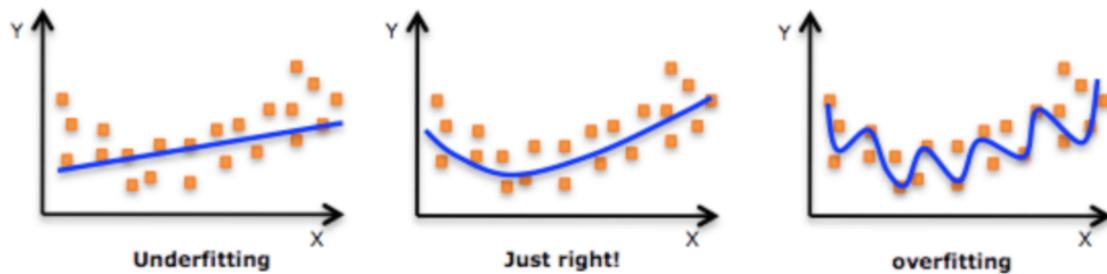
Prophet is een procedure om tijdreeksen te voorspellen ontwikkeld door Facebook. („Forecasting at scale.” 2020) Deze procedure baseert zich op een additief model waar niet-lineaire trends gefit worden aan jaarlijkse, wekelijkse en dagelijkse seizoensgebondenheid hierbij zal ook rekening gehouden worden met zogenaamde “holiday effects”, fluctuaties in de data die veroorzaakt worden door vakantieperiodes. Deze techniek zal best werken met tijdreeksen die een sterk seizoensgebonden effect hebben en waarvan ook meerdere seizoenen aan historische data beschikbaar zal zijn. Daarnaast houdt Prophet ook rekening met ontbrekende data, verschuivingen in de trend en zal het ook rekening houden met uitschieters.

Het is eenvoudig om te gebruiken maar er zijn ook optimalisaties mogelijk indien men het model wil finetunen. Prophet is zowel beschikbaar in R als in Python.

Hier zullen voornamelijk 2 parameters gebruikt worden om dit te optimaliseren. De eerste parameter changepoint_prior_scale zal aangeven in hoeverre de trend aan de data gefit zal worden. Hierbij zal een te hoge waarde voor overfitting zorgen. Dit houdt in dat het model te hard gefixeerd is op de trainingsdata en de verbanden in de trainingsset te nauwgezet volgt waardoor het de testdata moeilijk zal kunnen voorspellen. Een te lage waarde zal voor underfitting zorgen waarbij trendlijn de data te onnauwkeurig zal representeren. Het concept van underfitting en overfitting wordt weergegeven op Figuur 2.16.

De andere parameter die geoptimaliseerd zal worden is de seasonality_prior_scale. Deze parameter zal weergeven hoe strikt een seizoenstrend gevuld zal worden dit . Deze parameter zal enkel gebruikt worden wanneer er met seizoensgebonden data gewerkt wordt.

Figuur 2.16: Grafische weergave van underfitting en overfitting (Johan Decorte, 2019)



2.5 Validatietechnieken

Om de kwaliteit van de modellen te beoordelen zal er telkens een deel van de dataset achtergehouden worden die niet zal gebruikt worden bij het opstellen van het model. Deze achtergehouden dataset wordt de testset genoemd. In het geval van een tijdreeks zullen dit altijd de laatste waarden zijn. Het model zal dan trachten deze laatste waarden te voorspellen waarna de voorspelde waarden vergeleken zullen worden met de waarden van de testset. Op basis hiervan kunnen we achterhalen hoe groot de fout is bij elk model, het model met de laagste fout zal dan de meest accurate voorspelling maken. In deze sectie zullen enkele verschillende methodes om de fout te berekenen toegelicht worden (Rob J Hyndman, 2018).

2.5.1 Foutmaten

MAE

MAE staat voor mean absolute error. Deze zal berekend worden door het verschil te nemen tussen elke voorspelde tijdstap en de verwachte waarde voor deze tijdstap, hier de absolute waarde van te nemen en dan het gemiddelde te nemen van al deze absolute waarden. De formule voor de gemiddelde absolute fout wordt hieronder weergegeven.

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (2.21)$$

Dit is de meest eenvoudige foutmaat en wordt vaak gebruikt al moet er echter mee rekening gehouden worden dat deze fout eenheidsgebonden is en kan deze niet gebruikt worden om tijdreeksen te vergelijken die gebruik maken van andere eenheden. Dit heeft wel als voordeel dat deze fout makkelijk te interpreteren valt. Zo zou bijvoorbeeld een fout van 1 bij een tijdreeks over temperatuur in graden Celsius betekenen dat de gemiddelde fout bij het voorspellen van de temperatuur 1 graad Celsius is bij dat model.

RMSE

RMSE staat voor root mean squared error. Deze wordt berekend door het verschil te nemen tussen elke voorspelde tijdstap en de verwachte waarde bij deze tijdstap, dit te kwadrateren, het gemiddelde te nemen van deze kwadraten en dan de vierkantswortel te nemen van dit gemiddelde. Dit kan ook als volgt geformuleerd worden:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}} \quad (2.22)$$

Ook bij RMSE zal de eenheid gelijk blijven. Het verschil met MAE is echter dat een groot verschil tussen de voorspelde en effectieve waarde zwaarder zal doorwegen bij RMSE omdat deze eerst gekwadrateerd worden voordat het gemiddelde genomen wordt (Kampakis, 2020).

MAPE & RMSPE

MAPE staat voor mean absolute percentage error en RMSPE voor root mean square percentage error. Dit zijn procentuele fouten, dit type fout kan vergeleken worden tussen tijdreeksen met een verschillende eenheid. Dit zijn variaties op respectievelijk MAE en RMSE maar waarbij deler toegevoegd wordt bij het berekenen van het verschil tussen de voorspelde en de effectieve waarde. De formules van deze fouten worden hieronder weergegeven.

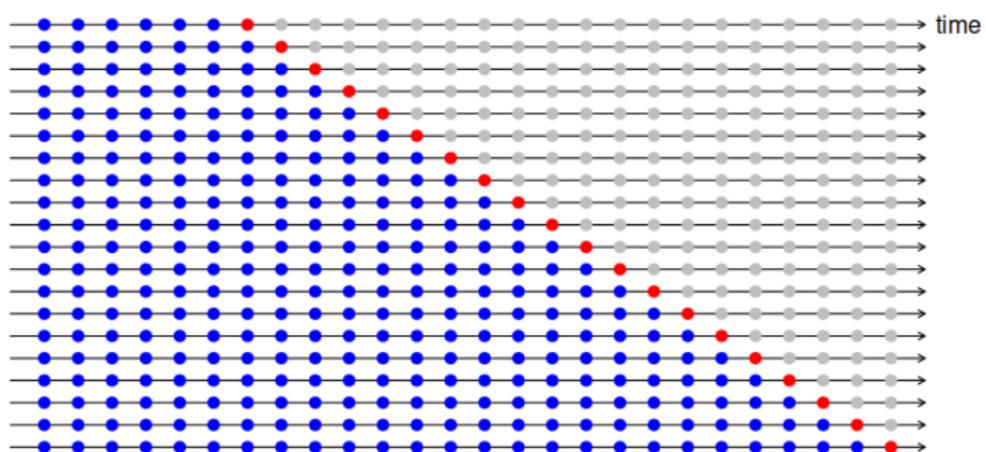
$$MAPE = \frac{\sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|}{n} \quad (2.23)$$

$$RMSPE = \sqrt{\frac{\sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{y_i} \right)^2}{n}} \quad (2.24)$$

2.5.2 Cross-validation

Om een tijdreeks nog beter te benutten kan ook cross-validation gebruikt worden. Hierbij zal dezelfde tijdreeks meerdere malen gebruikt worden om het model te trainen, maar zal deze telkens uitgebreid worden zoals zichtbaar in figuur 2.11. De accuraatheid van het model zal dan bepaald worden door de het gemiddelde te nemen van de accuraatheid voor elke opgestelde testset. Er moet echter rekening mee gehouden worden dat deze nieuwe training en testsets altijd nieuwe observaties moeten bevatten en de observaties van de trainingsset altijd moeten voorkomen voor deze van de bijhorende testset (Shrivastava, 2020).

Figuur 2.17: Grafische weergave van de structuur van een cross-validation tijdreeks (Rob J Hyndman, 2018) waarbij 1 tijdreeks zal opgedeeld worden in 20 train- en testreeksen



3. Methodologie

Om te onderzoeken welke types modellen best presteren zal er voor elke combinatie van seizoensgebonden of niet-seizoensgebonden en univariate of multivariate data een notebook opgesteld worden. Op het einde van elke notebook zal dan aan elk model een foutscore toegekend zijn. Deze foutscore wordt bepaald door het gemiddelde te nemen van de mean average errors van elke partitie bij de datasetverdeling door cross-validation. Het model met de laagste foutscore zal de meest accurate voorspelling gemaakt hebben bij het type invoerdata bij dit deelprobleem voor de gebruikte data.

Er dient echter vermeld te worden dat dit in zekere mate altijd zal afhangen van de invoerdata en dit type model niet noodzakelijk de beste prestatie zal leveren bij dit type invoerdata.

3.1 Voorbereiding

3.1.1 Datavoorbereiding

Dataset 1, temperatuur

Hier zal het deel van de dataset die de temperatuur weergeeft geformateerd worden naar een dataset met 1 kolom en een index waarin de maand en het jaartal weergegeven worden van het jaar 1979 tot 2018.

Listing 2: Datavoorbereiding dataset temperatuur

```

1 # Source: https://www.kaggle.com/rainbowgirl/climate-data-toronto-19372018
2 tt = pd.read_csv('./data/Toronto_temp.csv')
3 tt = tt[tt['Day'] == 1]
4 tt['Year'] = tt['Year'].replace({'2,013':'2013',
5 '2,014':'2014',
6 '2,015':'2015',
7 '2,016':'2016',
8 '2,017':'2017',
9 '2,018':'2018'})
10 # tt.groupby('Year').count()
11 tt = tt[(tt['Year'] != '1937')]
12 ttt = tt.groupby('Year').count()
13 #ttt.head(50)
14 #ttt.groupby('Year').count().tail(50)
15 meantt = tt.groupby('Year').mean()['Mean Temp (C)']
16 #meantt.index
17 #meantt
18 meantt.sort_index(inplace=True)
19
20 plt.xlabel('Years')
21 plt.ylabel('Temperature (C)')
22 plt.xticks(np.array(range(0,meantt.size,10)))
23 plt.scatter(meantt.index, meantt)
24
25 print('start : ' + meantt.index[0])
26 print('end : ' + meantt.index[-1])
27
28 new_row = pd.Series({'Mean Temp (C)' : 0.555556, 'Year': '2018', 'Month':12})
29 tt = tt.append(new_row, ignore_index=True)
30 tt['Year'] = tt['Year'].astype(int)
31 mean_temp_monthly = tt[['Year', 'Month', 'Mean Temp
   ↵ (C)']].set_index(['Year', 'Month']).sort_index()
32 # mean_temp_monthly
33 mean_temp_monthly =
   ↵ mean_temp_monthly[mean_temp_monthly.index.get_level_values(0).astype(int) >=
   ↵ 1979 ]
34 mean_temp_monthly

```

Op listing 2 wordt de code weergegeven die de ruwe dataset zal omzetten naar een dataset die bruikbaar zal zijn om te gebruiken voor deze bachelorproef.

De delen die in commentaar staan geven een tussentijdse weergave van de dataset, maar worden niet telkens weergegeven om het aantal tabellen binnen deze sectie overzichtelijk te houden.

Zo wordt op lijn 2 het originele csv-bestand ingelezen deze ruwe data zal er uit zien als deze zichtbaar op figuur 3.2.

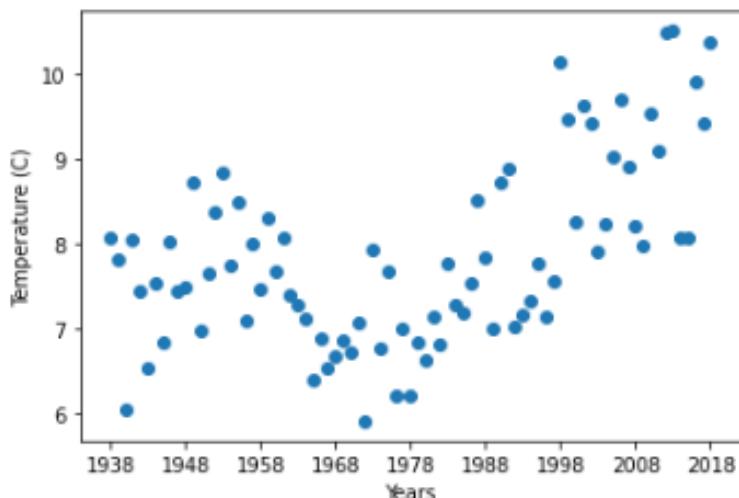
Op lijn 3 worden enkel de waarden van de eerste dag van de maand behouden in de dataset. Daarna worden van lijn 4 tot lijn 9 de jaartallen aangepast zodat deze overeenkomen met

Figuur 3.1: Ruwe invoerdata

Date/Time	Year	Month	Day	Mean Temp (C)	Max Temp (C)	Min Temp (C)	Total Rain (mm)	Total Snow (cm)	Total Precip (mm)	season
0	01-Jan-18	2018	1	1.0	-15.000000	-9.0	-21.0	0.0	0.0	0.0 Winter
1	01-Jan-17	2017	1	1.0	0.000000	2.0	-3.0	0.0	0.0	0.0 Winter
2	01-Jan-16	2016	1	1.0	-2.000000	0.0	-4.0	0.0	1.0	1.0 Winter
3	01-Jan-15	2015	1	1.0	-5.000000	-2.0	-8.0	0.0	0.0	0.0 Winter
4	01-Jan-14	2014	1	1.0	-13.000000	-10.0	-15.0	0.0	0.0	0.0 Winter
...
967	01-Dec-41	1941	12	1.0	-1.500000	1.9	-4.8	24.9	16.3	41.1 Winter
968	01-Dec-40	1940	12	1.0	-3.600000	0.6	-7.7	68.1	14.5	82.6 Winter
969	01-Dec-39	1939	12	1.0	NaN	NaN	NaN	NaN	NaN	NaN Winter
970	01-Dec-38	1938	12	1.0	-2.500000	1.1	-6.1	13.7	13.7	27.4 Winter
971	NaN	2018	12	NaN	0.555556	NaN	NaN	NaN	NaN	NaN NaN

972 rows × 11 columns

Figuur 3.2: Grafische weergave jaarlijkse temperatuur



de rest van de dataset.

Op lijn 10 wordt het aantal rijen per jaar weergegeven.

Op lijn 11 zal het eerste jaar uit de dataset verwijderd worden aangezien deze data onvolledig is.

Op lijn 12 wordt het aantal jaren toegekend aan de variabele *ttt*

Op lijn 15 wordt het gemiddelde van de temperaturen van de eerste dagen van de maand per jaar berekend.

Op lijn 18 wordt dit gemiddelde gesorteerd.

Van lijn 20 tot lijn 23 worden de labels en ijkingen gedefinieerd van de grafiek die de data zal weergeven en zichtbaar is op figuur 3.2 die op lijn 24 getekend zal worden.

Op lijn 25 en 26 worden de start en de eindjaren van deze dataset afgeprint om ze te kunnen vergelijken met de start en de eindjaren van de andere dataset.

Op lijn 28 wordt de laatste waarde voor het jaar 2018 toegevoegd aangezien deze nog niet in de dataset zat.

Figuur 3.3: Resultaat van data formatting

		Mean Temp (C)
Year	Month	
1979	1	-7.700000
	2	-10.800000
	3	1.600000
	4	5.300000
	5	11.400000

2018	8	24.000000
	9	21.000000
	10	10.000000
	11	6.000000
	12	0.555556

480 rows × 1 columns

Op lijn 29 wordt deze nieuwe waarde toegevoegd.

Op lijn 30 wordt de waarde voor het jaar geconverteerd naar een integer.

Op lijn 31 wordt de dataset geïndexeerd op jaar en maand.

Op lijn 33 worden de jaren die voor 1979 komen uit de dataset gefilterd omdat de waarden voor de andere dataset starten vanaf 1979.

Op lijn 34 wordt het resultaat van de dataformatting uitgevoerd zichtbaar op figuur 3.3.

Dataset 2, ijsdikte

Hier zal het deel van de dataset die de ijsdikte weergeeft geformateerd worden naar een dataset met 1 kolom en het jaartal van het jaar 1979 tot 2018 als index zal dienen.

Listing 3: Datavoorbereiding dataset temperatuur

```

1 #source: https://nsidc.org/arcticseaicenews/sea-ice-tools/
2 ice2 = pd.read_csv('./data/seaside2.csv')
3 # ice2
4 ice2_mean = ice2.mean()[1:-2]
5 # ice2_mean
6 ice2_mean.index = ice2_mean.index.values.astype(int)
7
8 plt.title('Yearly ice extent')
9 plt.scatter(ice2_mean.index,ice2_mean)
10 plt.xlabel('Years')
11 plt.ylabel('Extent')
12 plt.show()
13
14 # ice2['2018']
15 #
16 → pd.concat([ice2['2016'],ice2['2017'],ice2['2018'],ice2['2019']]).reset_index()[0]
17 # ice2[['2018']].append(ice2[['2019']])
18 ice2.rename(columns={'Unnamed: 0' : 'Month', 'Unnamed: 1' : 'Day'}, inplace =
19 → True)
20 ice2.drop([' ', '1981-2010', 'Day', '1978', '2020'],axis=1,inplace=True)
21 values = ice2.values
22 i = 0
23 for row in values :
24     if type(row[0]) != str :
25         values[i][0] = month
26     else:
27         month = row[0]
28         i = i +1
29 # ice2.columns.values
30 ice2_clean = pd.DataFrame(values)
31 ice2_clean.columns = ice2.columns.values
32 # ice2_clean.head(5)
33 ice2_monthly_mean =
34     → ice2_clean.set_index('Month').astype(float).groupby('Month',sort=False).mean()
35 # ice2_monthly_mean
36 # ice2_monthly_mean.T.stack().index.get_level_values(0)
37 #
38     → ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
39 ice2_monthly_mean_chron =
40     → ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
41 # ice2.columns.size
42 plt.title('Monthly ice extent')
43 plt.plot(ice2_monthly_mean_chron.values)
44 plt.xticks(np.array(range(0,500,75)))
45 plt.xlabel('Cumulative month')
46 plt.ylabel('Extent')
47 plt.show()
48
49 # np.unique(ice2_monthly_mean_chron.index.values).size*12

```

Figuur 3.4: Ruwe data van de jaarlijkse ijsdiktes

	Unnamed: 0	Unnamed: 1	1978	1979	1980	1981	1982	1983	1984	1985	...	2013	2014	2015	2016	2017	2018	2019	2020
0	January	1	NaN	NaN	14.200	14.256	NaN	14.253	NaN	NaN	...	12.959	13.011	13.073	12.721	12.643	12.484	12.934	13.102
1	NaN	2	NaN	14.997	NaN	NaN	14.479	NaN	14.103	14.045	...	12.961	13.103	13.125	12.806	12.644	12.600	12.992	13.075
2	NaN	3	NaN	NaN	14.302	14.456	NaN	14.306	NaN	NaN	...	13.012	13.116	13.112	12.790	12.713	12.634	12.980	13.176
3	NaN	4	NaN	14.922	NaN	NaN	14.642	NaN	14.237	14.240	...	13.045	13.219	13.051	12.829	12.954	12.724	13.045	13.187
4	NaN	5	NaN	NaN	14.414	14.435	NaN	14.494	NaN	NaN	...	13.065	13.148	13.115	12.874	12.956	12.834	13.147	13.123
...	
361	NaN	27	14.383	NaN	NaN	13.953	NaN	13.664	13.394	NaN	...	12.693	12.967	12.680	12.291	12.291	12.325	12.721	NaN
362	NaN	28	NaN	14.101	14.172	NaN	14.144	NaN	NaN	13.571	...	12.870	12.930	12.745	12.484	12.235	12.344	12.712	NaN
363	NaN	29	14.500	NaN	NaN	14.128	NaN	13.855	13.494	NaN	...	12.897	12.936	12.762	12.525	12.223	12.523	12.780	NaN
364	NaN	30	NaN	14.092	14.093	NaN	14.159	NaN	NaN	13.701	...	12.804	13.038	12.800	12.617	12.273	12.569	12.858	NaN
365	NaN	31	14.585	NaN	NaN	14.224	NaN	13.907	13.789	NaN	...	12.826	13.046	12.735	12.553	12.397	12.621	12.889	NaN

366 rows × 47 columns

```

45 print('from ' + ice2_monthly_mean_chron.index.values[0] + ' until ' +
46   ↪ ice2_monthly_mean_chron.index.values[-1])
46 ice2_monthly_mean_chron =
47   ↪ ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
47 ice2_monthly_mean_chron.columns = ['ice_extent']
48 ice2_monthly_mean_chron

```

Op lijn 2 zal de ijsdataset ingelezen worden en zal er uitzien zoals zichtbaar op figuur 3.4
 Op lijn 4 zal het gemiddelde genomen worden van alle kolommen tussen eerste en de voorlaatste. De eerste, de voorlaatste en de laatste kolom worden uit de dataset gelaten omdat deze irrelevant zijn voor dit onderzoek.

Op lijn 6 worden de indexwaarden geconverteerd naar integers.

De code van lijn 8 tot lijn 12 zal ervoor zorgen dat de jaarlijkse data grafisch weergegeven wordt.

Op lijn 17 zullen de kolomnamen hernoemd worden en op lijn 18 worden de overbodige kolommen verwijderd.

De code van lijn 18 tot lijn 26 zal er voor zorgen dat de maandkolom aangevuld wordt omdat deze tot hiertoe nog onvolledig zal zijn.

Op lijn 28 wordt een nieuwe dataframe geïnitialiseerd waarvan de kolommen op lijn 29 aangevuld worden.

Op lijn 31 wordt hier het maandelijks gemiddelde genomen van deze nieuwe dataframe en opgeslaan in de variabele en op lijn 35 worden deze in chronologische volgorde geplaatst.

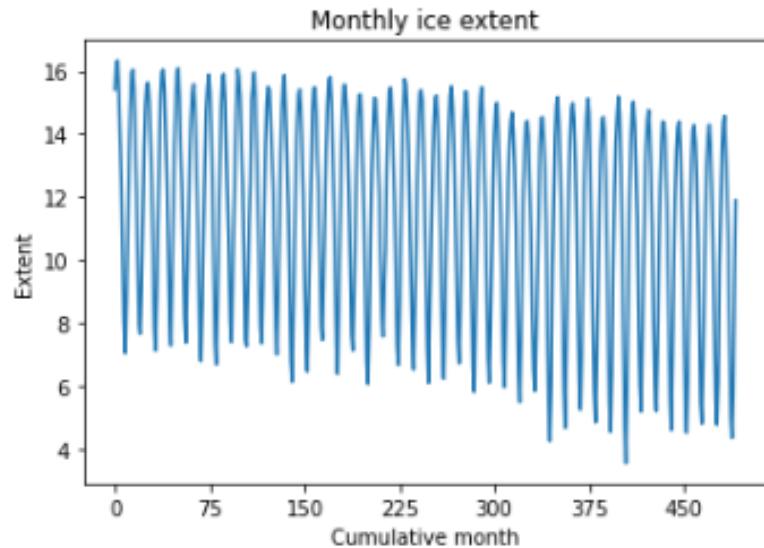
Het deel code van lijn 37 tot en met lijn 42 zal zorgen voor de grafische weergave van de maandelijkse ijsdikte weergegeven op figuur3.5.

Op lijn 45 worden de start en eindjaartallen uitgeprint.

De code op lijn 46 zal ervoor zorgen dat de maandindex verwijderd wordt.

Op lijn 47 zal de kolom hernoemd worden en op lijn 48 zal het resultaat uitgeprint worden, dit zal er uit zien zoals zichtbaar op figuur3.6.

Figuur 3.5: Grafische weergave van de maandelijkse ijsdiktes



Figuur 3.6: Data van de maandelijks ijsdiktes

ice_extent	
1979	15.414000
1979	16.175286
1979	16.341938
1979	15.446800
1979	13.856867
...	...
2019	5.026323
2019	4.363900
2019	5.734903
2019	9.352833
2019	11.903097

492 rows × 1 columns

Combineren van de datasets

In dit deel van de datavoorbereiding zullen de datasets gecombineerd worden.

Listing 4: Datavoorbereiding dataset temperatuur

```

1 ice2_monthly_mean_chron_cut = ice2_monthly_mean_chron[:-12]
2 # ice2_monthly_mean_chron
3 # ice2_monthly_mean_chron_cut
4 # mean_temp_monthly
5 # ice2_monthly_mean_chron_cut
6 combined = mean_temp_monthly[mean_temp_monthly.index.get_level_values(0) >= 1979]
7 combined['ice_extent'] = ice2_monthly_mean_chron_cut.values
8 # combined
9 combined.rename(columns={'Mean Temp (C)': 'mean_temp'}, inplace=True)
10 dataframe_monthly = combined
11 # dataframe_monthly
12 # dataframe_monthly[['mean_temp']]
13 plt.plot(dataframe_monthly[['mean_temp']].values[-24:], label='temperature')
14 plt.plot(dataframe_monthly[['ice_extent']].values[-24:], label='ice extent')
15 plt.legend()
16 plt.show()
17 dataframe_yearly = combined.groupby('Year').mean()
18 # dataframe_yearly
19 # dataframe_monthly[['mean_temp']].values
20 plt.plot(dataframe_monthly[['mean_temp']].values, label='temperature')
21 plt.plot(dataframe_monthly[['ice_extent']].values, label='ice extent')
22 plt.legend()
23 dataframe_monthly.to_csv('./data/dataframe_monthly.csv')
24 dataframe_yearly.to_csv('./data/dataframe_yearly.csv')

```

Op lijn 1 worden de laatste 12 waarden van de gemiddelde maandelijkse chronologische dataset in een nieuwe variabele gestoken.

Op lijn 6 wordt een nieuwe dataset geïnitialiseerd waar de maandelijkse gemiddelde temperaturen van 1979 ingevoerd worden.

Op lijn 7 worden de ijsdiktes hieraan toegevoegd.

Op lijn 9 wordt de kolom hernoemd op de lijn erna wordt de dataset nog eens toegevoegd aan een duidelijker variabelenaam.

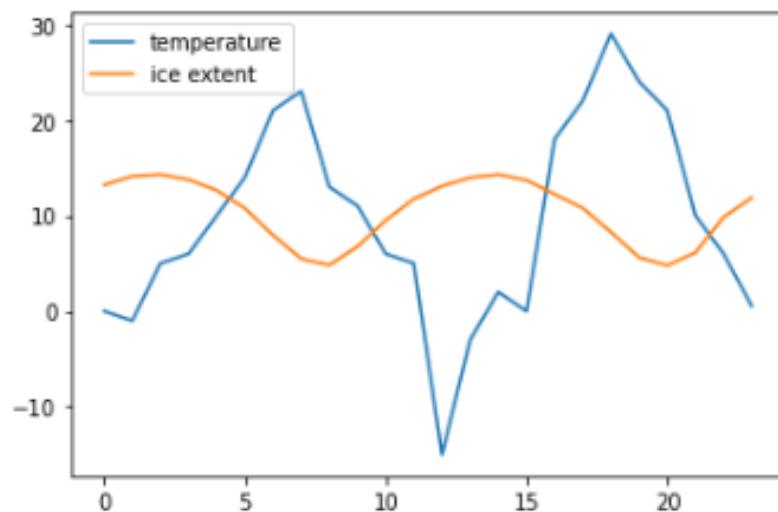
De code op lijn 13 tot 17 zorgt voor de grafische weergave van de gecombineerde dataset van de laatste 2 jaren weergegeven op figuur 3.7. Hier valt op te merken dat de ijsdikte zal vergroten wanneer de temperatuur laag is. Dit is logisch aangezien het ijs zal smelten bij hogere temperaturen.

De code op lijn 17 zal het jaarlijks gemiddelde nemen.

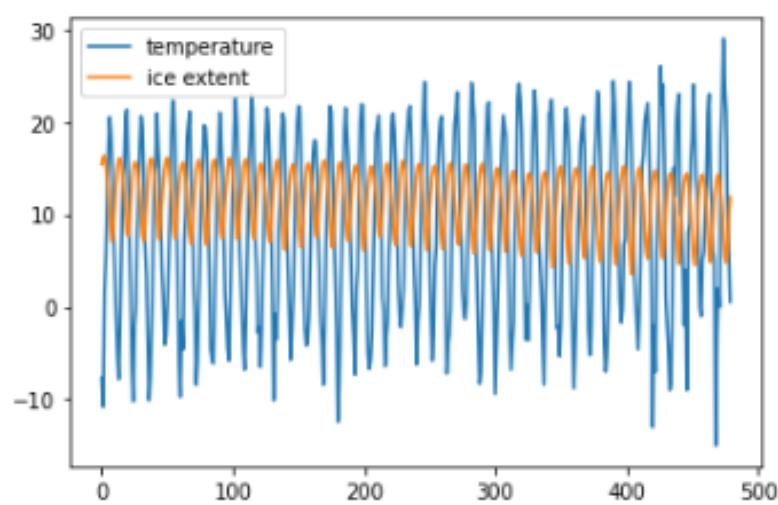
Op lijn 20 tot 23 wordt de code weergegeven die ervoor zal zorgen dat het volledige maandelijk gemiddelde weergegeven zal worden van 1979 tot 2018. Dit valt te beschouwen op figuur 3.7. Hierop kan vastgesteld worden dat de temperatuur lichtjes zal dalen terwijl de ijsdikte zal stijgen.

Op lijn 23 en 24 worden de dataframes weggeschreven naar csv-bestanden zodat ze kunnen gebruikt worden bij het opstellen van de modellen.

Figuur 3.7: Grafische weergave van de ijsdikte en de temperatuur van de laatste 2 jaar



Figuur 3.8: Grafische weergave van de ijsdikte en de temperatuur



3.2 Univariate niet-seizoensgebonden

In dit deel zullen de voorspellingsmodellen voor univariate niet-seizoensgebonden data opgesteld worden. De modeltypes die bekijken zullen worden zijn ARIMA, LSTM en Prophet. Om dit te onderzoeken zullen er voorspellingen gemaakt worden van de gemiddelde jaarlijkse temperatuur.

3.2.1 Algemene methodes

Hier worden enkele algemene methodes gedefinieerd namelijk de full_graph methode die een grafiek zal weergeven van de ingevoerde gedifferentieerde tijdreeks die ook als invoerparameter gebruikt zal worden samen met de grafietitel.

Daarnaast wordt ook nog de methode revert_diff gedefinieerd waar de gedifferentieerde voorspellingen terug omgezet worden naar voorspellingen in miljoenen vierkante kilometers. Hiervoor zijn de gedifferentieerde voorspellingen en de originele dataset nodig als invoerparameters.

Listing 5: Opstellen van algemene methodes

```

1 # define functions used throughout the notebook
2
3 # define function for plotting last prediction and the actual data
4 def full_graph(predicted_diff, title):
5
6     # format predictions by adding NaN values in front
7     predictionsArray = np.asarray(revert_diff(predicted_diff, ts))
8     zerosArray = np.zeros(ts.values.size-len(predictionsArray).flatten())
9     cleanPrediction =
10        pd.Series(np.concatenate((zerosArray,predictionsArray))).replace(0,np.NaN)
11     cleanPrediction.index = ts.index.values
12
13     # plot
14     plt.title(title)
15     plt.plot(ts, marker='o', color='blue',label='Actual values')
16     plt.plot(cleanPrediction, marker='o', color='red',label='Last 4 year prediction')
17     plt.ylim([0,15])
18     plt.legend()
19     plt.show()
20
21     # define function for reverting a differenced dataset
22     def revert_diff(predicted_diff, og_data):
23
24         # retrieve last value
25         last_value = og_data.iloc[-predicted_diff.size-1][0]
26
27         # initialize reverted array
28         predicted_actual = np.array([])
29
30         # add each value in the differenced array with the last actual value
31         for value_diff in predicted_diff:
32             actual_value = last_value + value_diff

```

```

33 predicted_actual = np.append(predicted_actual, actual_value)
34 last_value = actual_value
35
36 return predicted_actual

```

3.2.2 Stationariteit

Dan zal nagegaan worden in hoeverre deze dataset stationair is met gebruik van de hieronder omschreven test_stationarity methode. Dit is noodzakelijk voor het opstellen van het ARIMA model.

Listing 6: Test stationarity

```

1 # define method to visualise the stationarity of a time series
2 def test_stationarity(timeseries):
3
4     #Determining rolling statistics
5     rolmean = timeseries.rolling(12).mean()
6     rolstd = timeseries.rolling(12).std()
7
8     #Plot rolling statistics:
9     orig = plt.plot(timeseries, color='blue',label='Original')
10    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
11    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
12    plt.legend(loc='best')
13    plt.title('Rolling Mean & Standard Deviation')
14    plt.show(block=False)
15
16 # check stationarity of time serie
17 test_stationarity(ts)

```

Het resultaat van de test_stationarity methode wordt weergegeven op figuur 3.9. Daar kan opgemerkt worden dat de data niet stationair is maar er een dalende trend aanwezig is. Om deze trend te neutraliseren en de data stationair te maken zal het random walk difference genomen worden en nogmaals de stationariteit testen.

Listing 7: Test stationarity bij random walk differencing

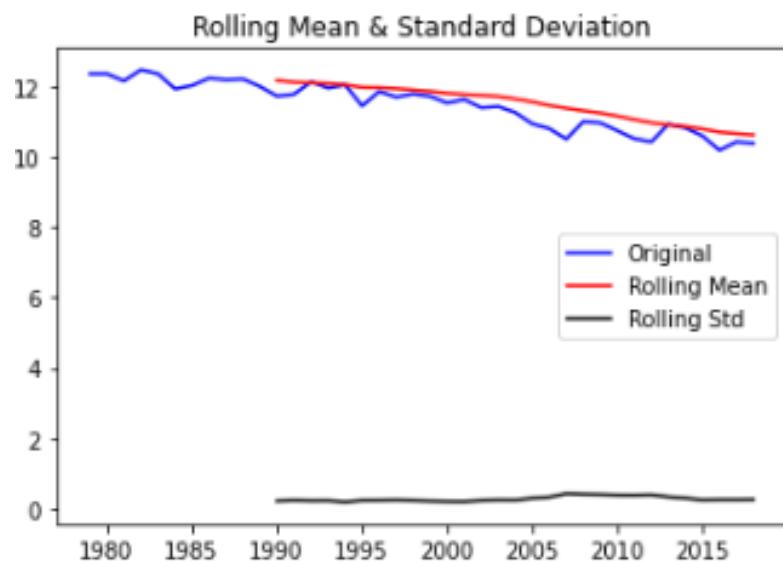
```

1 # take the random walk difference of the time serie
2 ts_diff = ts - ts.shift(1)
3 ts_diff = ts_diff.dropna()
4
5 # display stationarity of the newly differenced time serie
6 test_stationarity(ts_diff)

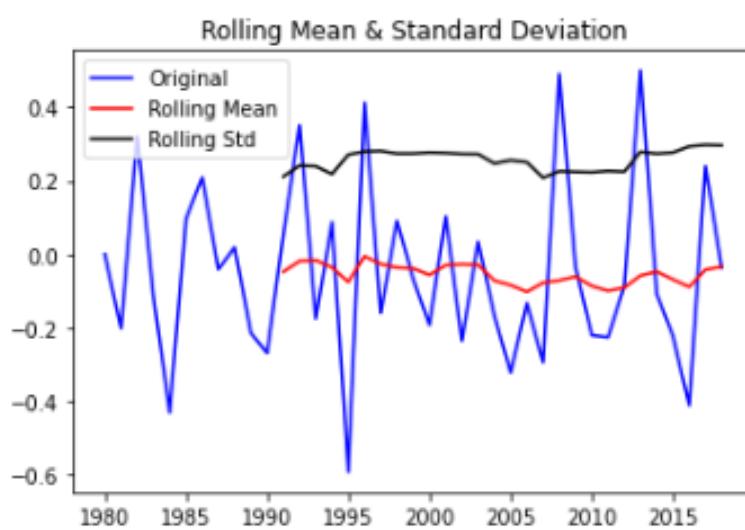
```

Door hiervan nogmaals de stationariteit te testen wordt figuur 3.10 bekomen. Hierop valt af te lezen dat de data nu wel stationair is.

Figuur 3.9: Resultaat test stationarity



Figuur 3.10: Resultaat test stationarity na random walk differencing



3.2.3 Cross validation

Om aan cross validation te doen moet de tijdreeks opgesplitst worden in verschillende reeksen waarbij de testset van de vorige reeks telkens toegevoegd wordt aan de trainingsset van de huidige reeks. Bij de univariate niet-seizoensgebonden tijdreeks wordt telkens een testgrootte van 4 genomen. De waarden worden ook enkel uitgeprint indien de testset groter is dan 20 om een minimale testset te garanderen. De grootte van de train-en testset zal ook geprint worden bij het uitvoeren van dit stuk code.

Listing 8: Code voor het opstellen van cross-validation

```

1 # initialize TimeSeriesSplit object
2 tscv = TimeSeriesSplit(n_splits = 8)
3
4 # loop trough all split time series that have a trainingsset with more than 20
4   ↳ values
5 for train_index, test_index in tscv.split(ts_diff):
6     if train_index.size > 20:
7
8       # initialize cross validation train and test sets
9       cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
10
11      # visualize cross_validation structure for reference
12      print("TRAIN:", train_index.size)
13      print("TEST:", test_index.size)
14      print()

```

3.2.4 ARIMA

In dit stuk code worden de hyperparameters bepaald die de beste resultaten zullen behalen. Zo worden alle mogelijke parametercombinaties binnen het ARIMA model getest op de data met cross validation. De best presterende parameters die dus zorgen voor de laagste MAE score worden behouden en zullen gebruikt worden voor de finale voorspelling.

Listing 9: Bepalen van de hyperparameters

```

1 %%time
2 # ARIMA
3 from statsmodels.tsa.arima_model import ARIMA
4 import itertools
5 import warnings
6 import sys
7 from sklearn.metrics import mean_absolute_error
8
9 # Define the p, d and q parameters to take any value between 0 and 2
10 p = q = range(0, 5)
11 d = range(0,3)
12
13 # Generate all different combinations of p, q and q triplets
14 pdq = list(itertools.product(p, d, q))
15

```

```
16 # initialize variables
17 best_pdq = pdq
18 best_mean_mae = np.inf
19
20 # specify to ignore warning messages to reduce visual clutter
21 warnings.filterwarnings("ignore")
22
23 # loop through all possible parameter combinations of pdq
24 for param in pdq:
25     print(param)
26
27 # some parametercombinations might lead to crash, so catch exceptions and
28 # → continue
29 try:
30
31     # initialize the array which will contain the mean average errors
32     maes = []
33
34     # loop through all split time series that have a trainingsset with more
35     # → than 20 values
36     for train_index, test_index in tscv.split(ts_diff):
37         if train_index.size > 20:
38
39             # initialize cross validation train and test sets
40             cv_train, cv_test = ts_diff.iloc[train_index],
41             # → ts_diff.iloc[test_index]
42
43             # build model
44             model = ARIMA(cv_train, order=(param))
45
46             # fit model
47             model_fit = model.fit()
48
49             # make predictions
50             predictions = model_fit.predict(start=len(cv_train),
51             # → end=len(cv_train)+cv_test.size-1, dynamic=False)
52
53             # renaming for clarity
54             prediction_values = predictions.values
55             true_values = cv_test.values
56
57             # error calculation this part of the cross validation
58             maes.append(mean_absolute_error(true_values, prediction_values))
59
60             # error calculation for this parameter combination
61             mean_mae = np.mean(maes)
62             print('MAE: ' + str(mean_mae))
63
64             # store parameters resulting in the lowest mean MAE
65             if mean_mae < best_mean_mae:
66                 best_mean_mae = mean_mae
67                 best_maes = maes
68                 best_pdq = param
69                 best_predictions = prediction_values
70
71
```

```

68     except Exception as e:
69         print(e)
70         continue
71
72     # logging
73     print()
74     print('Best MAE = ' + str(best_mean_mae))
75     print(best_pdq)

```

Hieruit blijkt dat de beste parametercombinatie voor een bereik van 0 tot 5 voor de p en q waarden en 0 tot 2 voor de d waarden (3,0,0) zijn. De d waarden blijven zo beperkt omdat een waarde hoger dan 1 in praktijk niet voorkomt zoals beschreven in de literatuurstudie. Dit zijn dan ook de parameterwaarden die gebruikt zullen worden voor de finale iteratie van ARIMA. Een hoger bereik zou tot een beter resultaat kunnen leiden maar ook tot overfitting en zal zeker zorgen voor een hogere uitvoeringstijd omwille van deze redenen is het bereik van p en q beperkt tot 5.

Listing 10: Finale iteratie ARIMA

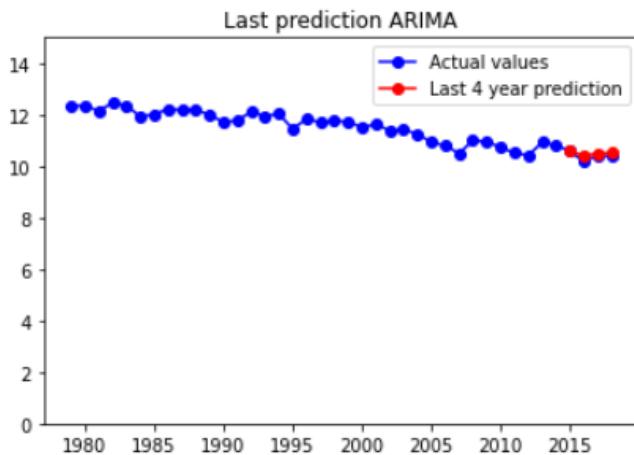
```

1 start_time = timeit.default_timer()
2
3 # specify to ignore warning messages
4 warnings.filterwarnings("ignore")
5
6 print("----")
7
8 # initialize the array which will contain the mean average errors
9 maes = []
10
11 # loop trough all split time series that have a trainingsset with more than 20
12 # values
13 for train_index, test_index in tscv.split(ts_diff):
14     if train_index.size > 20:
15
16         # initialize cross validation train and test sets
17         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
18
19         # build model
20         arima = ARIMA(cv_train, best_pdq).fit(start_ar_lags=1, disp=False)
21
22         # make predictions
23         predictions = arima.forecast(steps=4)
24         prediction_values = predictions[0]
25         true_values = cv_test.values
26
27         # error calc
28         maes.append(mean_absolute_error(true_values, prediction_values))
29
30         # last actual prediction
31         last_prediction_ARIMA = prediction_values
32
33         print("I", end="")

```

Figuur 3.11: Resultaat finale iteratie ARIMA

```
Mean MAE: 0.171 x 1 000 000 km2
MAE of last prediction: 0.103 x 1 000 000 km2
Execution time: 0.360 seconds
```



```
Mean average errors
[0.12511530847905092, 0.2752061283732373, 0.18026943891615582, 0.1025190381166726]
```

```
34 # store results to variables
35 time_ARIMA = timeit.default_timer() - start_time
36 mae_mean = np.mean(maes)
37 MAE_ARIMA = mae_mean
38 last_MAE_ARIMA = maes[-1]
39
40 # logging
41 print()
42 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_ARIMA)
43 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_ARIMA)
44 print('Execution time: %.3f seconds' % time_ARIMA)
45 full_graph(last_prediction_ARIMA, 'Last prediction ARIMA')
46 print('Mean average errors:')
47 print(maes)
```

De bovenstaande code zal leiden tot de uitvoer die zichtbaar is op figuur 3.11. Hier wordt de gemiddelde MAE overeen de verschillende iteraties bij cross validation weergegeven alsook de MAE van de laatste voorspelling. Daarnaast wordt ook de uitvoeringstijd weergegeven en ook de voorspelde waarden van de laatste partitie van de cross validation ten opzichte van de originele waarden. Ook de reeks met de MAEs wordt weergegeven.

3.2.5 LSTM

Het volgende model dat getest zal worden is een LSTM model. Aangezien dit een neuraal netwerk is zijn er op voorhand enkele methodes gedefinieerd om het overzicht te bewaren. De eerste methode die gedefinieerd wordt is de split_sequence dit zal er voor zorgen dat de univariabele tijdreeks opgesplitst wordt in samples zodat deze gebruikt kunnen worden als invoerwaarden voor een LSTM netwerk. De originele tijdreeks dient ingegeven te worden bij sequence, het aantal invoerstappen en het aantal uitvoerstappen dienen ook meegegeven te worden.

Daarnaast wordt ook de methode build_model gedefinieerd die het model zal opstellen. Dit zal gebeuren door de structuur van het LSTM model op te stellen met gebruik van het aantal features, het aantal neuronen in de LSTM laag, de dropout rate en de batchgrootte deze parameters diennen ook ingegeven te worden. Na het opstellen van het model zal het gefit worden aan de trainingsdata.

Ten slotte wordt ook nog de functie predict opgesteld. Deze functie verwacht de trainings-set, het model en het aantal features als invoerwaarde en zal het verdere verloop van de tijdreeks trachten te voorspellen.

Listing 11: Functies voor het opstellen van een LSTM model

```

1  from keras.layers import Dropout
2  # split a univariate sequence into samples
3  def split_sequence(sequence, n_steps_in, n_steps_out):
4      X, y = list(), list()
5      for i in range(len(sequence)):
6          # find the end of this pattern
7          end_ix = i + n_steps_in
8          out_end_ix = end_ix + n_steps_out
9
10         # check if we are beyond the sequence
11         if out_end_ix > len(sequence):
12             break
13
14         # gather input and output parts of the pattern
15         seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
16         X.append(seq_x)
17         y.append(seq_y)
18     return array(X), array(y)
19
20 def build_model(raw_seq, n_steps_in, n_steps_out, n_features, n_neurons, dropout,
21                 batch_s):
22
23     # split into samples
24     X, y = split_sequence(raw_seq.values.flatten(), n_steps_in, n_steps_out)
25
26     # reshape from [samples, timesteps] into [samples, timesteps, features]
27     X = X.reshape((X.shape[0], X.shape[1], n_features))
28
29     # define model
30     model = Sequential()
31     model.add(LSTM(n_neurons, activation='relu'))
32     model.add(Dropout(dropout))
33     model.add(Dense(n_steps_out))

```

```

33     model.compile(optimizer='adam', loss='mae')
34
35     # fit model
36     model.fit(X, y, batch_size=batch_s, epochs=100, verbose=0)
37
38     return model
39
40
41 def predict(x_input, model, n_features):
42     n_features = 1
43
44     # reshape data
45     x_input = x_input.reshape((1, n_steps_in, n_features))
46
47     # predict
48     yhat = model.predict(x_input, verbose=0)
49
50     return yhat

```

Ook bij LSTM dienen de hyperparameters geoptimaliseerd te worden. Dit gebeurt door middel van onderstaande code. Waarbij een reeks mogelijk waarden gedefinieerd wordt op lijn 15 tot 18 waarvan alle mogelijk combinaties getest worden en waarvan de best presterende combinatie bijgehouden wordt.

Listing 12: Bepalen van de hyperparameters

```

1 %%time
2
3 # Disabled tf warning because of visual clutter
4 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
5
6
7 # constant variables
8 n_steps_in = 4
9 n_steps_out = 4
10 n_features = 1
11 maes = []
12 global_maes = []
13
14 # optimizable variables
15 n_neurons_array = [1,10,20]
16 dropout_array = [0,0.5,0.99]
17 batch_size_array = [1,8]
18
19
20 # initialize values
21 best_MAE = 100
22 best_n_neurons = 0
23 best_activation = 'none'
24 best_dropout = 0
25 best_batch_size = 0
26
27 # loop over all possible parameter combinations
28 for n_neurons in n_neurons_array:

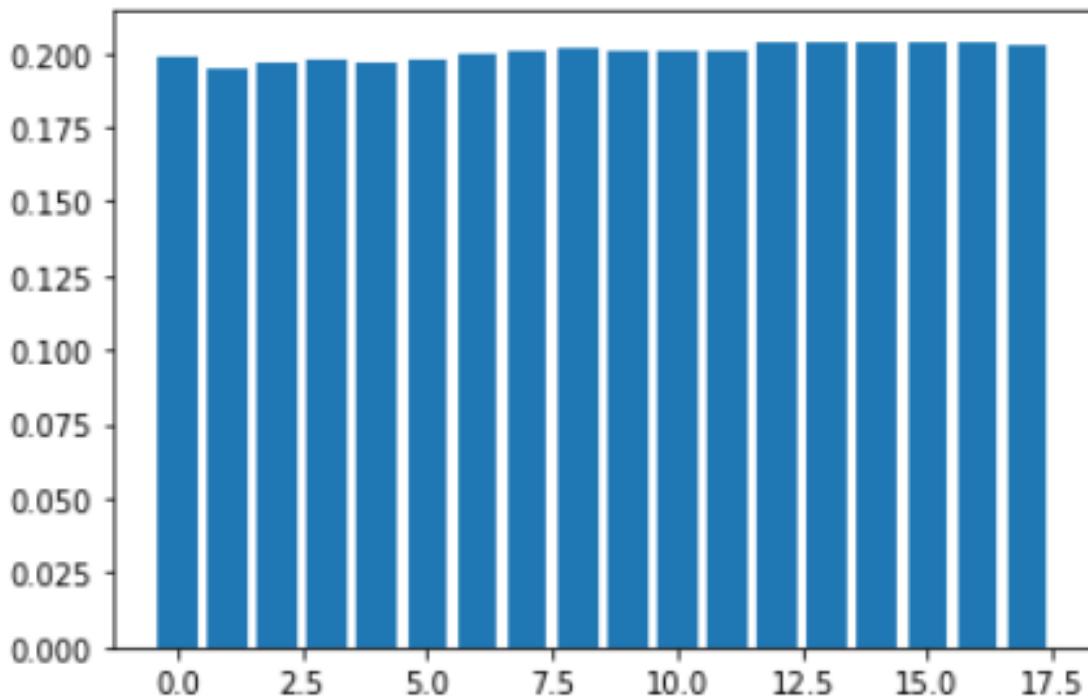
```

```

29     for dropout in dropout_array:
30         for batch_size in batch_size_array:
31
32             print("----")
33
34             # loop trough all split time series that have a trainingsset with
35             # more than 20 values
36             for train_index, test_index in tscv.split(ts_diff):
37                 if train_index.size > 20:
38
39                     # initialize cross validation train and test sets
40                     y_train, y_test = ts_diff.iloc[train_index],
41                     # ts_diff.iloc[test_index]
42
43                     # build model
44                     lstm_model = build_model(y_train, n_steps_in, n_steps_out,
45                     # n_features, n_neurons, dropout, batch_size)
46
47                     # make predictions
48                     x_input = array(y_test)
49                     y_predicted = predict(x_input, lstm_model,
50                     # n_features).flatten()
51                     y_actual = y_test.values
52
53                     # error calculation this part of the cross validation
54                     maes.append(mean_absolute_error(y_actual, y_predicted))
55
56                     print("I", end="")
57
58                     # last actual prediction
59                     last_prediction_LSTM = y_predicted
60
61                     # error calculation for this parameter combination
62                     MAE_LSTM = np.mean(maes)
63                     last_MAE_LSTM = maes[-1]
64                     global_maes.append(MAE_LSTM)
65
66                     # store parameters resulting in the lowest mean MAE
67                     if best_MAE > MAE_LSTM:
68                         best_n_neurons = n_neurons
69                         best_dropout = dropout
70                         best_batch_size = batch_size
71                         best_MAE = MAE_LSTM
72
73                     # log values for parameter combination
74                     print()
75                     print(n_neurons)
76                     print(dropout)
77                     print(batch_size)
78                     print(MAE_LSTM)
79                     print()
80
81                     # log parameter combination with best result
82                     print('Best:')
83                     print('N neurons')
84                     print(best_n_neurons)

```

Figuur 3.12: Grafische weergave verschil in MAE's bij verschillende hyperparameters met MAE op de y-as



```

81 print('Dropout rate')
82 print(best_dropout)
83 print('Batch size')
84 print(best_batch_size)
85 print('MAE')
86 print(best_MAE)
87 plt.bar(range(0, len(global_maes)), global_maes)

```

Na het uitvoeren van deze code blijkt er nauwelijks verschil te zijn bij het aanpassen van de hyperparameters. Dit valt af te leiden uit de MAE's die weergegeven worden door de plot op lijn 87. Die figuur 3.12 als resultaat zal hebben. Dit blijkt ook wanneer deze code meerdere malen doorlopen wordt aangezien er verschillende resultaten bekomen worden. Om verder te gaan zullen de waarden 1 voor het aantal neuronen gebruikt worden, 0 voor de beste dropout rate en 8 voor de batch size.

Listing 13: Finale iteratie LSTM

```

1 %%time
2
3 start_time = timeit.default_timer()
4
5 # Disabled tf warning because of visual clutter
6 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
7
8

```

```

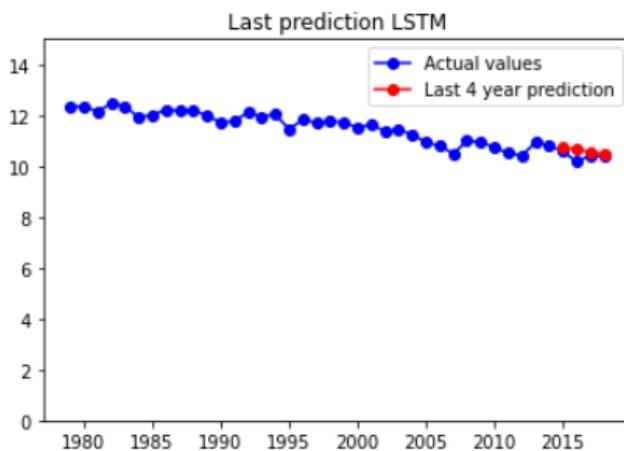
9  # constant variables
10 n_steps_in = 4
11 n_steps_out = 4
12 n_features = 1
13 maes = []
14
15
16 # optimizable variables
17 n_neurons = best_n_neurons
18 dropout = best_dropout
19 batch_s = best_batch_s
20
21 print("----")
22
23 # loop trough all split time series that have a trainingsset with more than 20
24 # values
24 for train_index, test_index in tscv.split(ts_diff):
25     if train_index.size > 20:
26         # initialize cross validation train and test sets
27         y_train, y_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
28
29         # build model
30         lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features,
31         # n_neurons, dropout, batch_s)
32
33         # make predictions
34         x_input = array(y_test)
35         y_predicted = predict(x_input, lstm_model, n_features).flatten()
36         y_actual = y_test.values
37
38         # error calc
39         maes.append(mean_absolute_error(y_actual, y_predicted))
40
41         print("I",end="")
42
43         # last actual prediction
44         last_prediction_LSTM = y_predicted
45
46         # store variables
47         time_LSTM = timeit.default_timer() - start_time
48         MAE_LSTM = np.mean(maes)
49         last_MAE_LSTM = maes[-1]
50
51         # visualisation
52         print()
53         print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
54         print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
55         print('Execution time: %.3f seconds' % time_LSTM)
56         full_graph(last_prediction_LSTM, 'Last prediction LSTM')
57         print('Mean average errors')
58         print(maes)

```

De bovenstaande code zal leiden tot de uitvoer die zichtbaar is op figuur???. Ook hier kan de gemiddelde MAE overeen de verschillende iteraties bij cross validation weergeven worden alsook de MAE van de laatste voorspelling. Daarnaast wordt ook de uitvoeringstijd

Figuur 3.13: Resultaat finale iteratie LSTM

Mean MAE: $0.200 \times 1\ 000\ 000 \text{ km}^2$
MAE of last prediction: $0.215 \times 1\ 000\ 000 \text{ km}^2$
Execution time: 5.480 seconds



Mean average errors
[0.1527431455020163, 0.23552654839108333, 0.19871178303483195, 0.21470190819788382]

weergegeven en ook de voorspelde waarden van de laatste partitie van de cross validation ten opzichte van de originele waarden. Ook de reeks met de MAEs wordt weergegeven. Net zoals bij ARIMA.

3.2.6 Prophet

Als laatste dient ook Prophet nog bekijken te worden voor het voorspellen van de tijdreeks. Hier zal eerst de data opnieuw geformateerd moeten worden aangezien prophet een bepaalde structuur hantereert.

Listing 14: Code voor het formatteren van de data voor Prophet

```

1 # formatting dataframe
2 ts_formated_prophet = ts_diff.reset_index().rename(columns = {'Year' : 'ds',
3   ↪ 'ice_extent' : 'y'})
4 ts_formated_prophet['ds'] =
5   ↪ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds']).astype(str),
6   ↪ format='%Y'))

```

Daarna kan de hyperparameter voor changepoint_prior_scale die zal staan voor de frequentie van het aanduiden changepoints (punten waar de data van een trend zal afwijken) bepaald worden met behulp van onderstaande code.

Listing 15: Code voor het bepalen van de hyperparameters

```

1 # Python
2 import itertools
3 import numpy as np
4 import pandas as pd
5
6 # define dataframe
7 df = ts_formated_prophet
8
9 param_grid = {
10   'changepoint_prior_scale': [0.001, 0.01, 0.1, 1, 2, 5, 10, 15, 20, 25],
11 }
12
13 # Generate all combinations of parameters
14 all_params = [dict(zip(param_grid.keys(), v)) for v in
15   ↪ itertools.product(*param_grid.values())]
16
17 # initialize variables
18 maes = []
19 global_maes = []
20 best_MAE_prophet = np.inf
21
22 # Use cross validation to evaluate all parameters
23 for params in all_params:
24
25   # loop trough all split time series that have a trainingsset with more than
26   ↪ 20 values
27   for train_index, test_index in tscv.split(ts_formated_prophet):
28     if train_index.size > 20:
29
30       # initialize cross validation train and test sets
31       train = ts_formated_prophet.iloc[train_index]
32       y_test = ts_formated_prophet.iloc[test_index][[ 'y']].values.flatten()

```

```

31     X_test = ts_formated_prophet.iloc[test_index][['ds']]
32
33     # Fit model with given params
34     model = Prophet(**params, weekly_seasonality=False,
35                      daily_seasonality=False)
36     model = model.fit(train)
37
38     # make predictions
39     forecast = model.predict(X_test)
40     y_pred = forecast['yhat'].values
41
42     # last actual prediction
43     last_prediction_prophet = y_pred
44
45     # error calculation this part of the cross validation
46     maes.append(mean_absolute_error(y_test, y_pred))
47
48     # error calculation for this parameter combination
49     MAE_prophet = np.mean(maes)
50     last_MAE_prophet = maes[-1]
51     global_maes.append(MAE_prophet)
52
53     # logging
54     print('changepoint_prior_scale: ' + str(params['changepoint_prior_scale']))
55
56     # store parameters resulting in the lowest mean MAE
57     if best_MAE_prophet > MAE_prophet:
58         best_params = params
59         best_MAE_prophet = MAE_prophet
60
61     # log optimal result
62     print('changepoint_prior_scale: ' + str(best_params['changepoint_prior_scale']))
63     print(best_MAE_prophet)

```

Hieruit zal blijken dat de optimale waarde voor de hyperparameter 2 is. Wanneer dit ingevoegd wordt bij het uitvoeren van de onderstaande code wordt het resultaat dat zichtbaar is op figuur 3.14 verkregen.

Listing 16: Finale iteratie Prophet

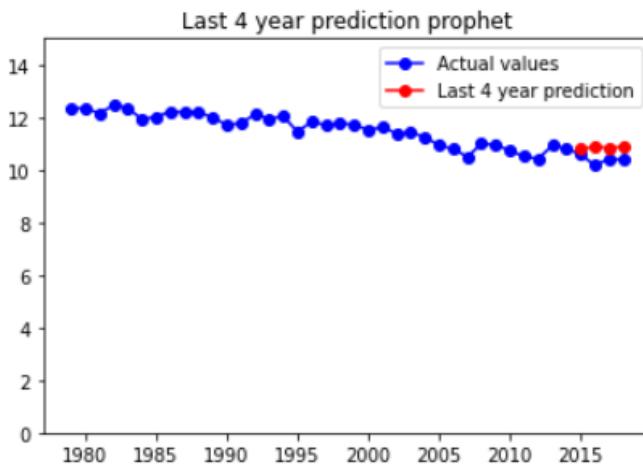
```

1 %%time
2
3 # Disabled tf warning because of clutter
4 warnings.filterwarnings("ignore") # specify to ignore warning messages
5
6 start_time = timeit.default_timer()
7
8 # initialize variables
9 maes = []
10
11 for train_index, test_index in tscv.split(ts_formated_prophet):
12     if train_index.size > 20:
13
14         # initialize cross validation train and test sets

```

Figuur 3.14: Resultaat finale iteratie Prophet

```
Mean MAE: 0.209 x 1 000 000 km2
MAE of last prediction: 0.245 x 1 000 000 km2
Execution time: 8.557 seconds
```



```
Mean average errors
[0.126584592587182, 0.2432586886102241, 0.22013313821990807, 0.24456896573145737]
```

```

15      train  = ts_formated_prophet.iloc[train_index]
16      y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
17      X_test = ts_formated_prophet.iloc[test_index][['ds']]
18
19      # build model
20      model = Prophet(**best_params, weekly_seasonality=False,
21                      daily_seasonality=False)
22      model.fit(train)
23
24      # make predictions
25      forecast = model.predict(X_test)
26      y_pred = forecast['yhat'].values
27
28      # error calc
29      maes.append(mean_absolute_error(y_test, y_pred))
30
31      # last actual prediction
32      last_prediction_prophet = y_pred
33
34  # store results
35  time_Prophet = timeit.default_timer() - start_time
36  MAE_Prophet = np.mean(maes)
37  last_MAE_Prophet = maes[-1]
38
39  # visualize results
40  print()
41  print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_Prophet)
42  print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_Prophet)
43  print('Execution time: %.3f seconds' % time_Prophet)
```

```
44 full_graph(last_prediction_prophet, "Last 4 year prediction prophet")
45 print('Mean average errors')
46 print(maes)
```

Figuur 3.15: Resultaat van de univariate non-seasonal analyse

	Mean MAE (x 1 000 000 km ²)	Execution time (s)	Last MAE (x 1 000 000 km ²)
ARIMA	0.171	0.360	0.103
LSTM	0.200	5.480	0.215
Prophet	0.252	8.891	0.308

3.2.7 Evaluatie

Wanneer deze resultaten gecombineerd worden wordt de tabel die zichtbaar is op figuur 3.15 verkregen. Om dit resultaat grafisch te schetsen worden op figuur 3.16 de laatste voorspellingen voor elk modeltype weergegeven.

Hieruit kunnen dus geconcludeerd worden dat ARIMA het beste gemiddelde resultaat zal halen bij cross validation aangezien het hier een fout van $0.171 \times 1\,000\,000 \text{ km}^2$ behaalt. Ook de uitvoeringstijd is het laagst bij ARIMA namelijk 0.360 seconden. Dit is een beter dan LSTM met een fout van $0.200 \times 1\,000\,000 \text{ km}^2$ en een uitvoeringstijd van 5.480 seconden. De voorspelling van Prophet is nog een pak minder accuraat met een fout van $0.252 \times 1\,000\,000 \text{ km}^2$ en een uitvoeringstijd van 8.891 seconden.

Listing 17: Code voor het weergeven van de resultaten

```

1 # formatting
2 results =
3     [[MAE_ARIMA,time_ARIMA,last_MAE_ARIMA],[MAE_LSTM,time_LSTM,last_MAE_LSTM],[MAE_Prophet,time_Prophet]]
4 # display results
5 pd.DataFrame(results, columns=['Mean MAE (x 1 000 000 km\u00b2)', 'Execution time (s)', 'Last MAE (x 1 000 000 km\u00b2)'], index=['ARIMA', 'LSTM', 'Prophet']).round(decimals=3)

```

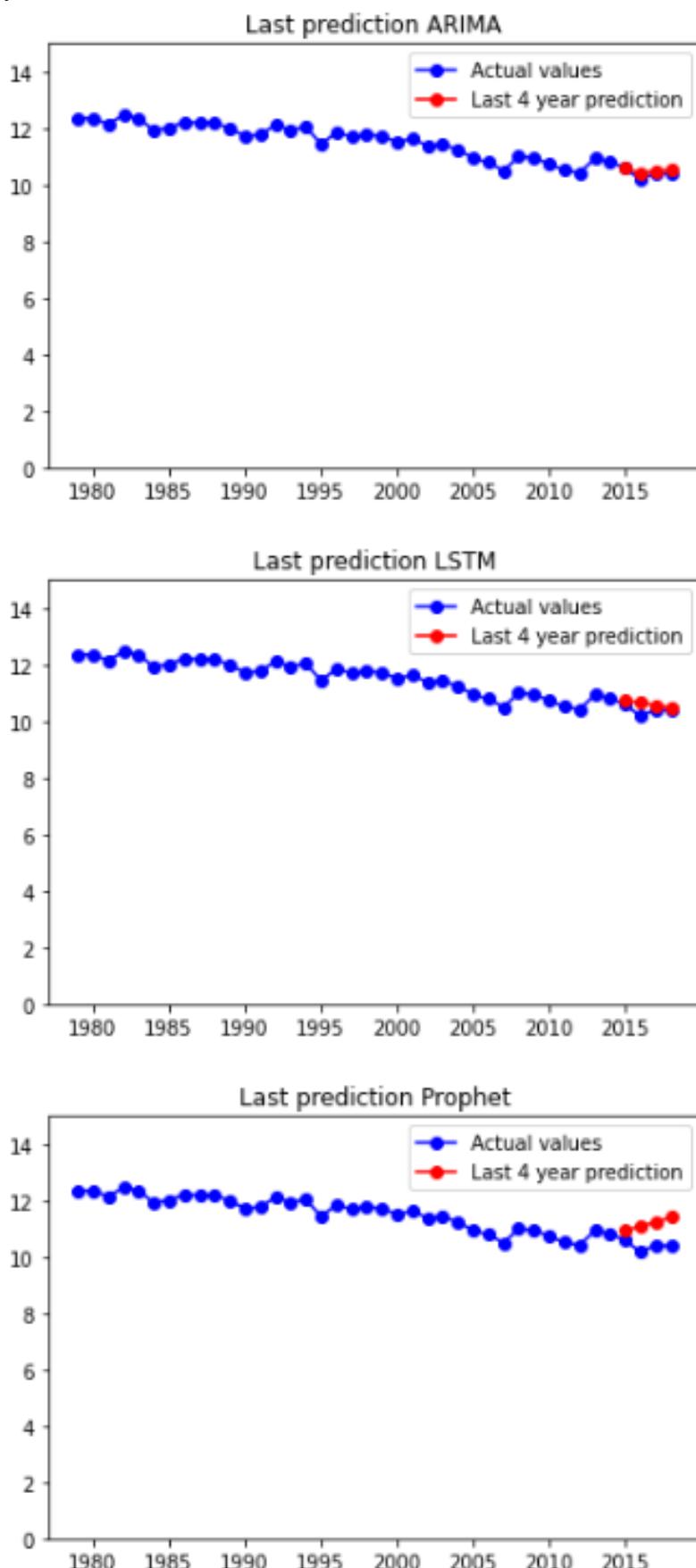
Listing 18: Code voor het grafisch weergeven van de resultaten

```

1 # visualize results of last prediction
2 full_graph(last_prediction_ARIMA, "Last prediction ARIMA")
3 full_graph(last_prediction_LSTM, "Last prediction LSTM")
4 full_graph(last_prediction_prophet, "Last prediction Prophet")

```

Figuur 3.16: Grafische weergave van de laatste voorspellingen van de univariate non-seasonal analyse



3.3 Univariate seisoensgebonden

In deze sectie zal onderzocht worden welk type model de beste voorspellingen zal treffen voor data met 1 variabele waar er een duidelijk seisoensverband zichtbaar is. Ook hier zal crossvalidation gebruikt worden maar het aantal minimum waarden in de trainingsset zal hier niet 20 zijn maar 300.

3.3.1 Algemene methodes

Vooraleer de dataset effectief gebruikt zal worden dienen er eerst nog enkele functies gedeclareerd te worden.

Zo herkennen kunnen de test_stationarity, full_graph en revert_diff methodes van de voorgaande sectie herkend worden. Naast deze methodes wordt hier ook nog de revert_seasonal_diff_recursion methode gedefinieerd. Deze wordt benut bij de seisoensgebonden variant van de revert_diff methode namelijk revert_seasonal_diff. Ook deze methode zal dienen om een gedifferentieerde data terug om te zetten naar data die vergeleken kan worden met de effectieve waarden.

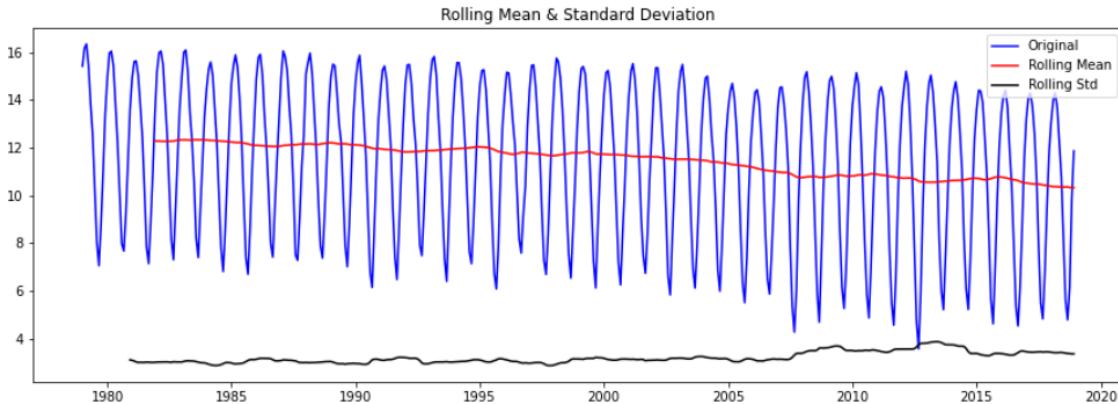
Listing 19: Algemene methodes

```

1 def test_stationarity(timeseries):
2
3     #Determining rolling statistics
4     rolmean = timeseries.rolling(36).mean()
5     rolstd = timeseries.rolling(24).std()
6
7     #Plot rolling statistics:
8     orig = plt.plot(timeseries, color='blue',label='Original')
9     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
10    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
11    plt.legend(loc='best')
12    plt.title('Rolling Mean & Standard Deviation')
13    plt.show(block=False)
14
15 def full_graph(predicted, og_dataset, title):
16     zerosArray = np.zeros(og_dataset.values.size-len(predicted.flatten()))
17     cleanPrediction =
18         pd.Series(np.concatenate((zerosArray,predicted))).replace(0,np.NaN)
19
20     # plot
21     plt.title(title)
22     plt.plot(og_dataset.index, og_dataset.values,marker='o',
23             color='blue',label='Actual values')
24     plt.plot(og_dataset.index, cleanPrediction,marker='o',
25             color='red',label='Last 2 year prediction')
26     plt.ylim([0,20])
27     plt.legend()
28
29     plt.show()
30
31 def revert_diff(predicted_diff, og_data):
32     last_value = og_data.iloc[-predicted_diff.size-1][0]

```

Figuur 3.17: Stationariteit van de originele univariate seisoensgebonden data



```

30     predicted_actual = np.array([])
31     for value_diff in predicted_diff:
32         actual_value = last_value + value_diff
33         predicted_actual = np.append(predicted_actual, actual_value)
34         last_value = actual_value
35     return predicted_actual
36
37 def revert_seasonal_diff_recursion(last_seasons_value, diff_value):
38     return last_seasons_value + diff_value
39
40 def revert_diff_seasonal(predicted_diff, og_data):
41     prediction_size = predicted_diff.size
42
43     history = ts[:-prediction_size].values.flatten()
44     for value_diff in predicted_diff[-prediction_size:]:
45         new_value = revert_seasonal_diff_recursion(history[-12], value_diff)
46         history = np.append(history, new_value)
47     return history[-prediction_size:]

```

3.3.2 Stationariteit

Ook wanneer we hier de stationariteit van de data gaan testen krijgen we een licht dalende trend. Deze trend die zichtbaar is op figuur 3.17. Deze loopt gelijk met de trend van de niet seisoensgebonden data aangezien die data het jaarlijks gemiddelde is en dit maandelijkse waarden zijn.

Aangezien hier met seisoensgebonden data gewerkt wordt is het hier naast random walk differentiatie een optie om seisoendifferentiatie te gebruiken. Dit zal dan ook getest worden.

Listing 20: Code voor seisoendifferentiatie

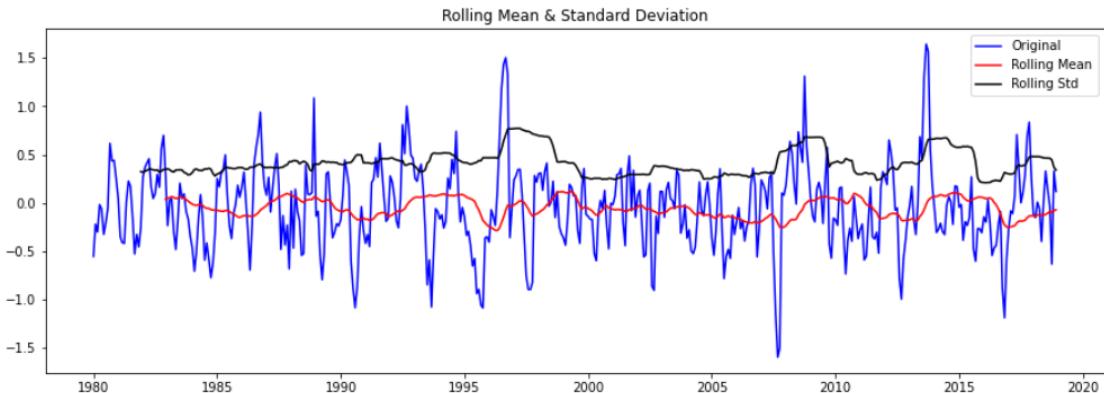
```

1 ts_diff_seasonal = ts - ts.shift(12)
2 ts_diff_seasonal = ts_diff_seasonal.dropna()
3 test_stationarity(ts_diff_seasonal)

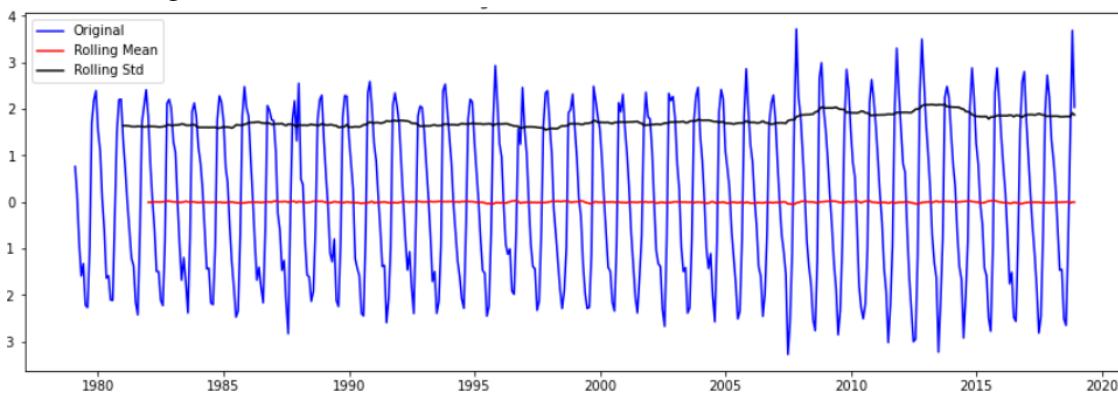
```

Wanneer seisoendifferentiatie toegepast wordt zal de stationariteit voorgesteld kunnen

Figuur 3.18: Stationariteit van de data na seizoendifferentiatie



Figuur 3.19: Stationariteit van de data na random walk differentiatie



worden zoals zichtbaar op figuur 3.18. Dit is wel stationair maar fluctueert wel behoorlijk naar het einde toe.

Daarnaast kan ook nog gewone random walk differentiatie toegepast worden op de data. De stationariteit van deze gedifferentieerde data wordt afgebeeld op figuur 3.19. Hier wordt vastgesteld dat het gemiddelde een horizontale lijn is. Dit houdt in dat de data zeer stationair is.

Listing 21: code voor differentiatie

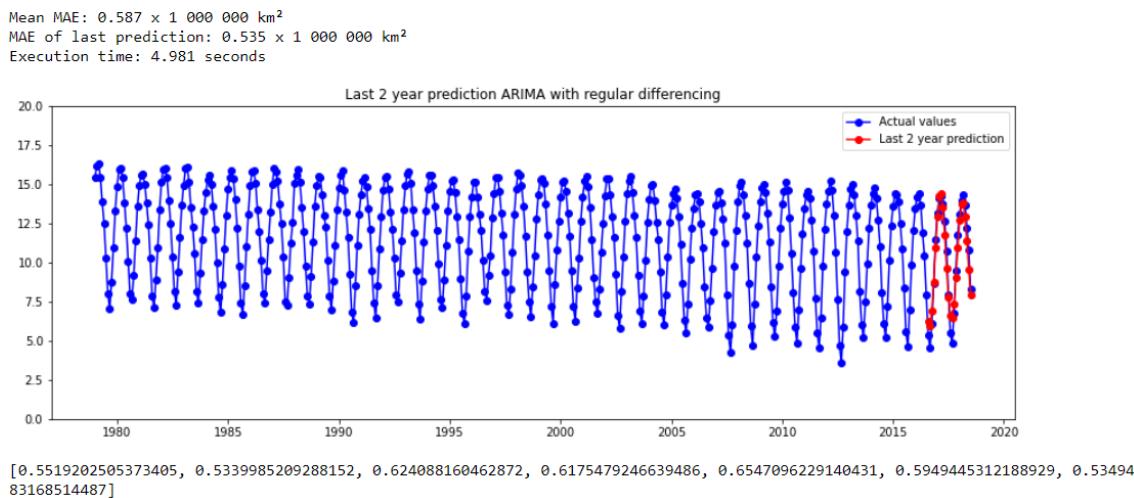
```

1 ts_diff = ts - ts.shift(1)
2 ts_diff = ts_diff.dropna()
3 test_stationarity(ts_diff)

```

Er dient ook nog vermeld te worden dat door 2 verschillende types differentiaties te testen, het aantal gebruikte testsets bij crossvalidation wel verschillen. Zo zullen er bij seizoendifferentiatie minder waarden beschikbaar zullen zijn. Concreet leidt dit ertoe dat er bij random walk differentiatie 7 trainingssets zijn die meer dan 300 waarden bevatten terwijl er bij seizoendifferentiatie 6 trainingssets zullen zijn die meer dan 300 waarden bevatten.

Figuur 3.20: Resultaat van ARIMA bij random walk differentiatie



3.3.3 ARIMA

Ook hier zal een ARIMA model opgesteld worden zowel voor de random walk gedifferenteerde data als de seizoensgedifferentieerde data.

Random walk differentiatie

Ook hier zal aan hyperparameterbepaling gedaan worden. Aangezien zo goed als alles identiek is als bij univariate non-seasonal differentiatie zal de code hiervan niet meer weergegeven worden alle code is echter terug te vinden in de bijlage.

Uit die hyperparameterbepaling zal blijken dat de combinatie (3,0,4) de beste prestatie zal leveren wanneer het bereik van de parameters p en q 5 zal zijn.

Wanneer deze hyperparameters ingevoerd worden wordt het resultaat dat afgebeeld staat op figuur 3.20 verkregen.

Seizoensdifferentiatie

Ook hier zullen eerst de hyperparameters bepaald worden. Met (3,0,3) als optimale waarden bij een range van 0 tot 5 voor de waarden p of q . Wanneer deze waarden uitgebreider getest worden resulteert dit in de uitvoer die waar te nemen valt op figuur 3.21.

3.3.4 SARIMA

Naast ARIMA zelf is er ook een variant op het ARIMA model die specifiek voor data met een seizoenseffect is ontworpen genaamd SARIMA. Deze variant zal in dit deel onder de loep genomen worden.

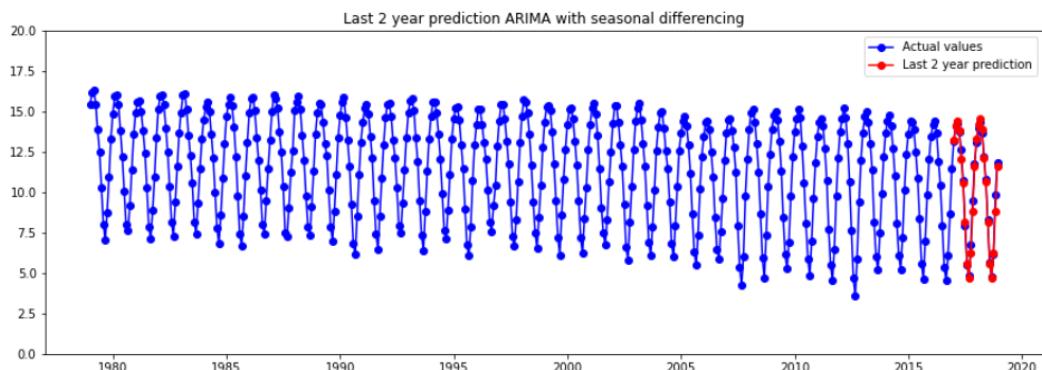
Random walk differentiatie

De hyperparameterbepaling zal er hier wel anders uitzien, zo dienen voor het seizoenseffect PDQ waarden bepaald te worden naast de pdq waarden voor de data zelf en de m waarde die het aantal tijdstappen zal weergeven van de data die binnen 1 sequentie van het seizoenseffect vallen.

De beste parametercombinatie van de parameters p, d, q, P, D, Q met een bereik van 3 zal

Figuur 3.21: Resultaat van ARIMA bij seizoendifferentiatie

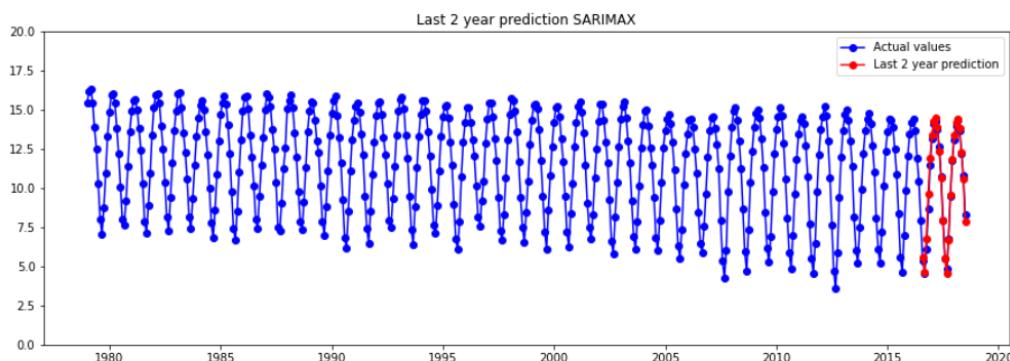
Mean MAE: $0.328 \times 1\ 000\ 000\ km^2$
 MAE of last prediction: $0.225 \times 1\ 000\ 000\ km^2$
 Execution time: 10.461 seconds



[0.48861485304926316, 0.26346070636629143, 0.35937111188428744, 0.3753477846215573, 0.2567851618849564, 0.2248188597690386]

Figuur 3.22: Resultaat SARIMAX bij random walk differentiatie

Mean MAE: $0.219 \times 1\ 000\ 000\ km^2$
 MAE of last prediction: $0.179 \times 1\ 000\ 000\ km^2$
 Execution time: 50.800 seconds



[0.19468134775270865, 0.2727208340446456, 0.2718027423914433, 0.17930539251035138, 0.22744152935594694, 0.20468858347158767, 0.17935643576480706]

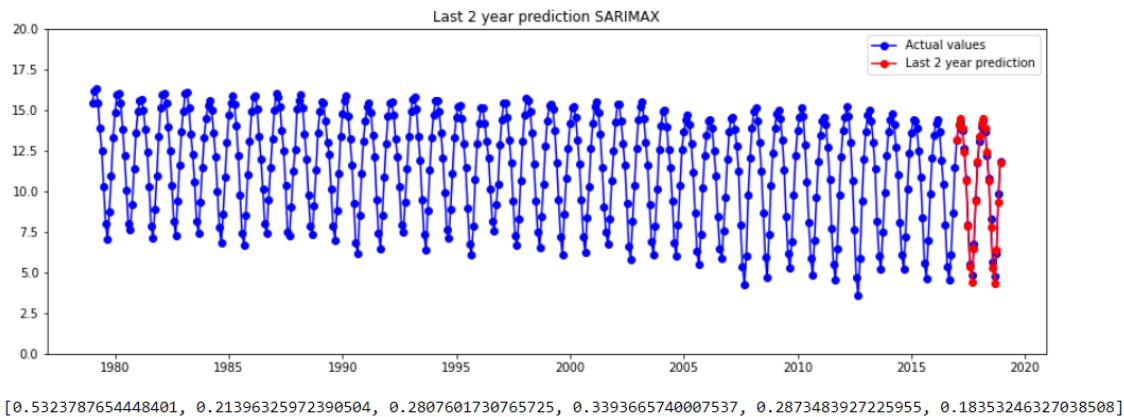
(1,0,2,0,1,2) zijn. Het uitgebreid resultaat van deze combinatie wordt weergegeven op figuur 3.22.

Seizoendifferentiatie

Deze testopstelling is identiek aan de vorige uitgezonderd van de invoerdata die hier seizoendifferentieerd zal zijn. Ook hier zullen de optimale parameters (1,0,2,0,1,2) zijn. Dit de uitvoer geven die afgebeeld staat op figuur 3.23.

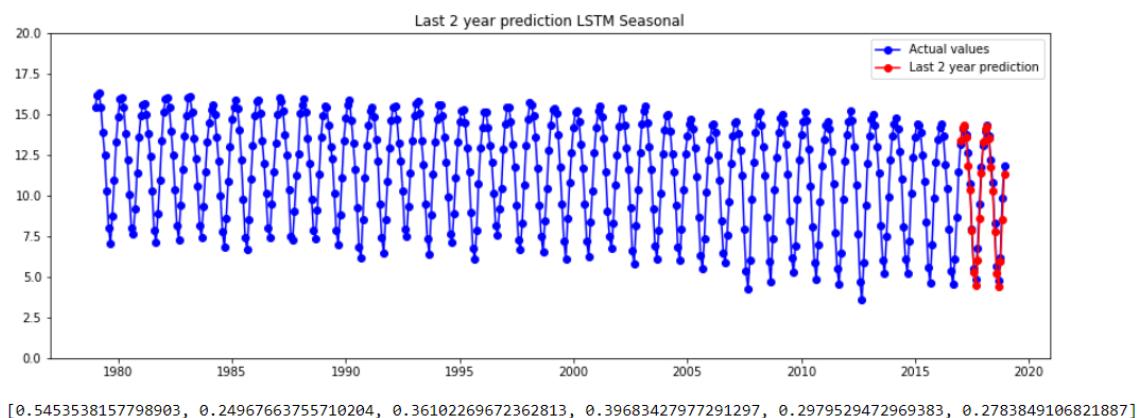
Figuur 3.23: Resultaat SARIMAX bij seizoendifferentiatie

Mean MAE: $0.306 \times 1\ 000\ 000 \text{ km}^2$
 MAE of last prediction: $0.184 \times 1\ 000\ 000 \text{ km}^2$
 Execution time: 50.648 seconds



Figuur 3.24: Resultaat LSTM bij seizoendifferentiatie

Mean MAE: $0.355 \times 1\ 000\ 000 \text{ km}^2$
 MAE of last prediction: $0.278 \times 1\ 000\ 000 \text{ km}^2$
 Execution time: 146.811 seconds

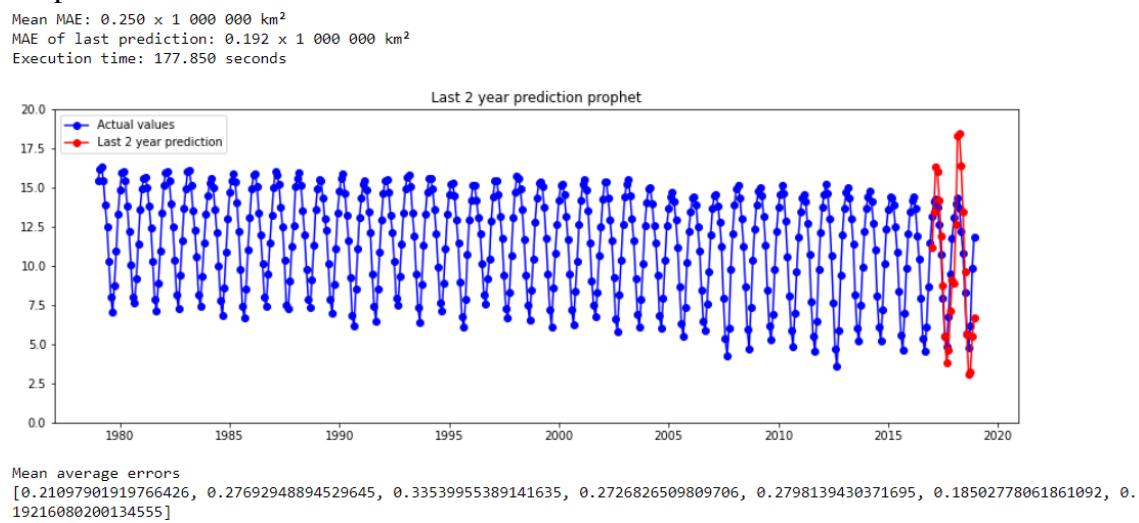


3.3.5 LSTM

Random walk differentiatie Seizoendifferentiatie

De opstelling zal identiek zijn als de bovenstaande hier zullen de optimale parameters bestaan uit 1 neuron een dropout rate van 0.99 en een batchgrootte van 8 met het eindresultaat dat zich op figuur 3.24 bevindt.

Figuur 3.25: Resultaten de voorspellingen voor univariate seisoensgebonden data met Prophet



3.3.6 Prophet

Ook hier verloopt alles bijna analoog met de niet seisoensgebonden variatie behalve dat er hier wel 2 hyperparameters getest worden, aangezien er hier wel een seizoenseffect aanwezig is kan de de variabele seasonality_prior_scale ook onderzocht worden. Het laatste resultaat van de reeks wordt weergegeven op Figuur 3.25.

Figuur 3.26: Resultaten van de univariate seisoensgebonden voorspellingen

	Mean MAE ($\times 1\ 000\ 000\ km^2$)	Execution time (s)	Last MAE ($\times 1\ 000\ 000\ km^2$)
ARIMA	0.587	7.508	0.535
ARIMA_seasonal_differencing	0.328	5.991	0.225
SARIMA	0.219	44.109	0.179
SARIMA_seasonal_differencing	0.306	57.663	0.184
LSTM	0.610	1036.232	0.336
LSTM_seasonal_differencing	0.355	146.811	0.278
Prophet	0.250	177.850	0.192

3.3.7 Evaluatie

Wanneer we de resultaten afgebeeld op Figuur 3.26 beschouwen kunnen we concluderen dat SARIMA met random walk differentiatie de beste voorspellingen zal treffen wanneer we de gemiddelde MAE beschouwen namelijk $0.219 (\times 1\ 000\ 000\ km^2)$. Ook de laatste voorspelling is de meest accurate van alle modellen. Ook de uitvoeringstijd van SARIMA met random walk differencing is relatief laag enkel de ARIMA modellen zijn sneller maar die zijn een stuk minder accuraat. Het SARIMA-model dat werkt met seisoensgedifferentieerde data presteert een stuk slechter met een gemiddelde MAE van $0.360 (\times 1\ 000\ 000\ km^2)$

Daarnaast presteert het Prophet-model ook goed met een gemiddelde MAE van $0.250 (\times 1\ 000\ 000\ km^2)$ al heeft het wel de op één na langste uitvoeringstijd. Het ARIMA-model met seisoendifferentiatie heeft een gemiddelde MAE van $0.328 (\times 1\ 000\ 000\ km^2)$ maar heeft wel de kortste uitvoeringstijd. De LSTM modellen scoren beide vrij slecht met hoge uitvoeringstijden dus deze vallen niet aan te raden met deze implementatie om seisoensgebonden tijdreeksen te voorspellen.

Bij seisoensgebonden data presteren de LSTM- en ARIMA-modellen beter wanneer er aan seisoendifferentiatie gedaan wordt, maar de implementatie met random walk differentiatie behaalt de beste score bij SARIMAX dus er kan niet gesteld worden dat de ene soort differentiatie altijd effectiever zal zijn dan de andere. De laatste voorspellingen van alle modeltypes worden weergegeven op Figuren 3.27 en 3.28.

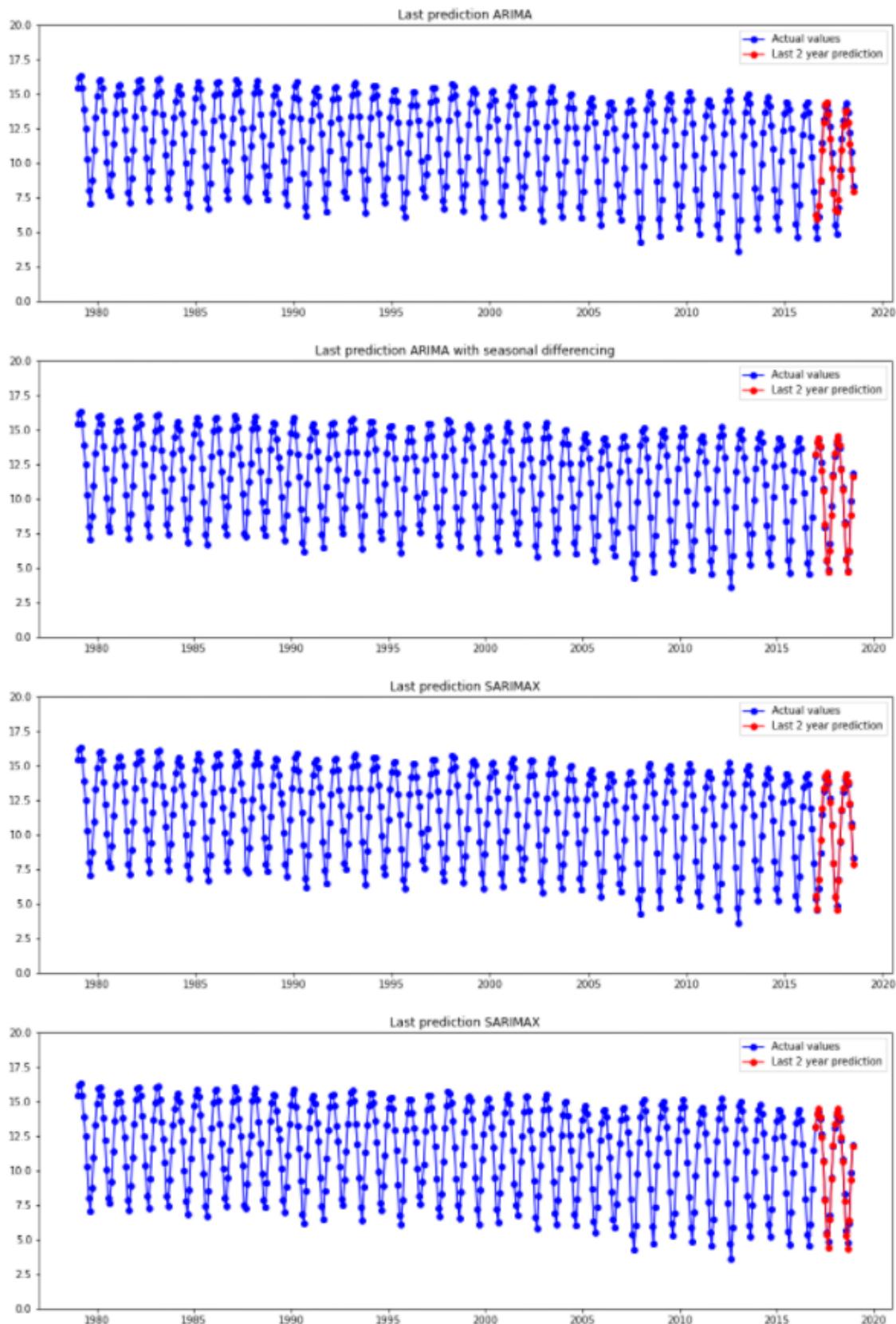
3.4 Multivariate niet-seizoensgebonden

In dit onderdeel van deze bachelorproef zal onderzocht worden welk van de 3 modellen de beste voorspelling maken voor meervoudige tijdreeksen. Dit houdt in dat het model een tijdreeks krijgt die meerdere features bevat per tijdstap en deze features in verband brengt om tot accuratere voorspellingen te komen. De testset zal dan ook altijd enkel blijven bestaan uit de tijdsstappen die voorspeld moeten worden.

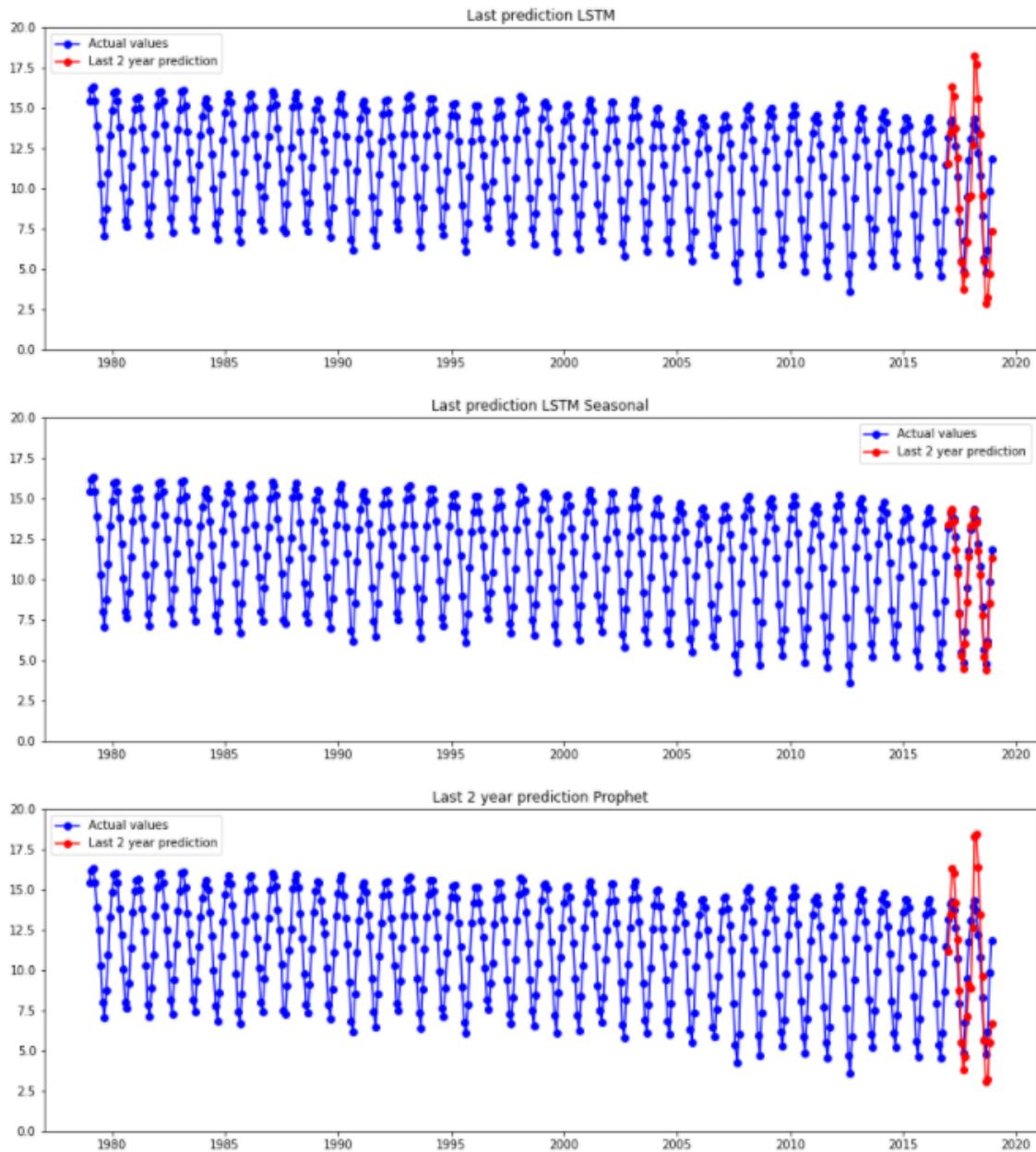
De data die gebruikt zal worden voor deze sectie wordt grafisch weergegeven op Figuur 3.29. Ook hier zal er gebruik gemaakt worden van cross validation.

Aangezien Prophet geen multivariate tijdreeksen als invoer ondersteunt zal dit niet geëva-

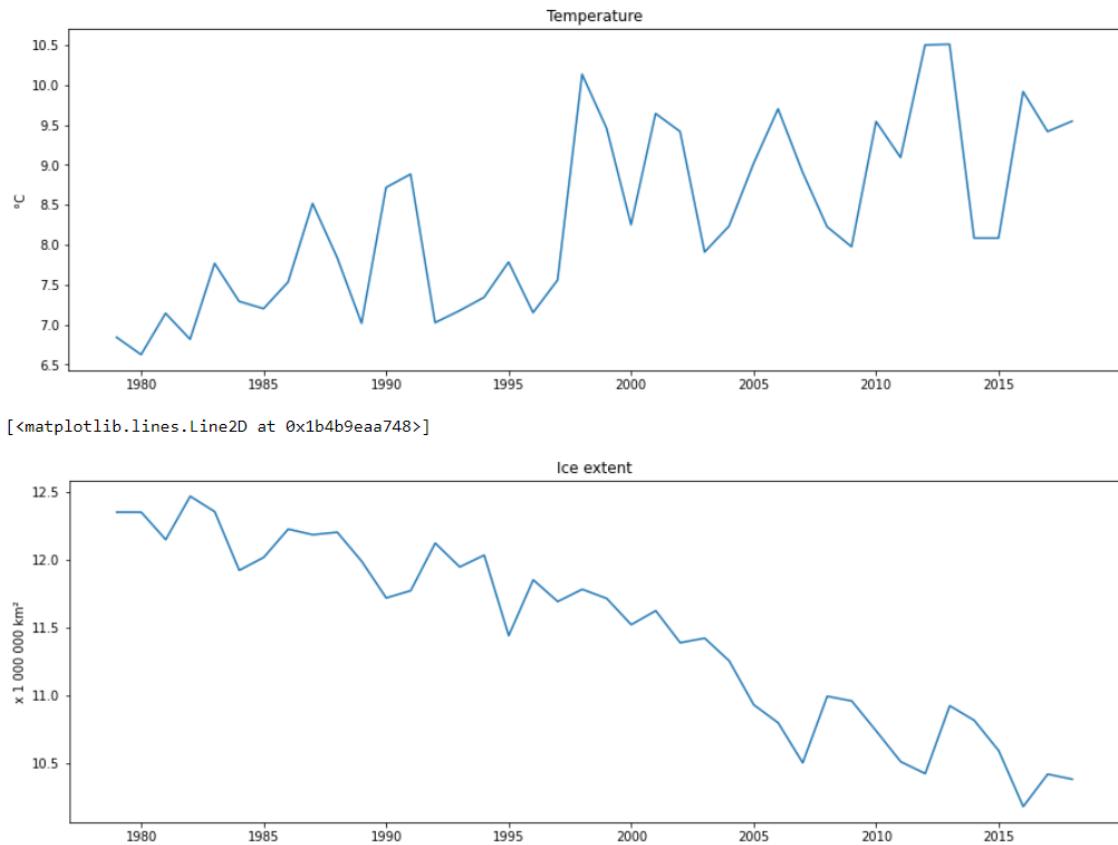
Figuur 3.27: Grafische weergaven van de laatste voorspellingen van univariate seizoensgebonden data (deel 1)



Figuur 3.28: Grafische weergaven van de laatste voorspellingen van univariate seizoensgebonden data (deel 2)



Figuur 3.29: Grafische weergave multivariate tijdreeks zonder seizoenseffect



lueerd kunnen worden.

3.4.1 Stationariteit

Ook hier zal de ijsdikte een dalende trend vertonen zoals weergegeven werd bij de univariate analyse en hier zal niets aan veranderen aangezien dit dezelfde data is. Ook deze data zal random walk gedifferentieerd worden maar dit keer niet enkel de ijsdikte maar ook de temperatuur. De resultaten van die differentiatie zijn zichtbaar op figuur 3.30. Daaruit kan afgeleid worden dat de data nu stationair is.

3.4.2 VARMAX

Aangezien een ARIMA model slechts in staat is om tijdreeksen met 1 variabele te voor-spellen moet er een variant gebruikt worden die wel in staat is om multivariate tijdreeksen te interpreteren genaamd VARMAX.

Ook hier wordt de gebruikelijke methode toegepast om de hyperparameters te bepalen en daar verandert niet veel aan. Behalve dat enkel de p en q variabelen verwacht worden, de d variabele is hier niet nodig.

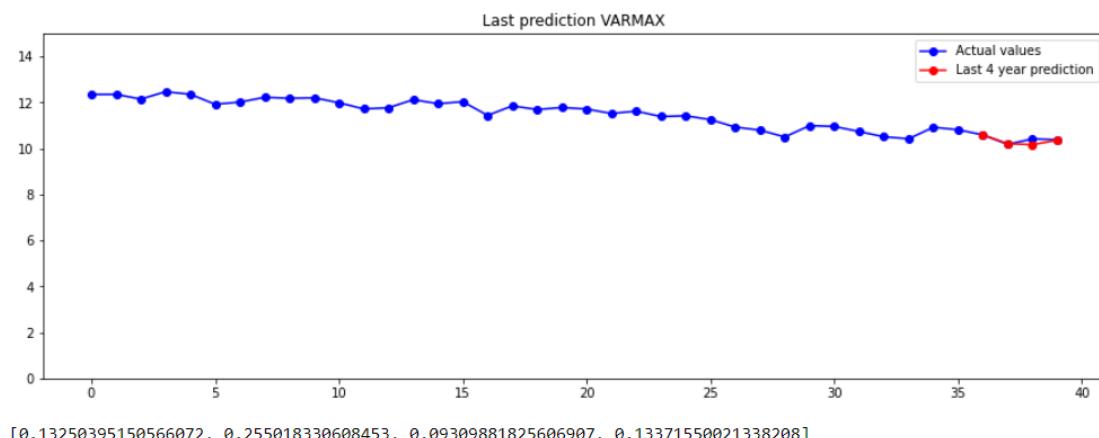
Bij een range van 0 tot 5 zijn ook hier de optimale waarden voor deze parameters (3,3). Wanneer we deze waarden invoeren in de uitgebreidere versie van de evaluatiemethode wordt het resultaat bekomen dat weergegeven wordt op Figuur 3.31.

Figuur 3.30: Grafische weergave gedifferentieerde multivariate tijdreeks zonder seizoens-effect



Figuur 3.31: Resultaat multivariate voorspelling VARMAX

Mean MAE: $0.154 \times 1\ 000\ 000 \text{ km}^2$
MAE of last prediction: $0.134 \times 1\ 000\ 000 \text{ km}^2$
Execution time: 63.963 seconds



3.4.3 LSTM

Wanneer er multivariate voorspellingen gemaakt moeten worden met LSTM ziet de structuur van het neurale netwerk er anders uit dan bij een univariate voorspelling. Zo zal er deze keer een grote functie gebruikt worden om het model op te stellen en de voorspelling te maken. Deze functie staat uitgeschreven bij listing 22.

De hyperparameters die hier bepaald zullen worden zijn het aantal neuronen en het aantal epochs om tot het beste resultaat te komen. Een enkel neuron blijkt alweer optimaal te zijn en met 200 epochs verkrijgt men het beste resultaat. De uitgebreide uitvoer van dit resultaat wordt weergegeven op Figuur 3.32.

Listing 22: Voorspellingsfunctie LSTM

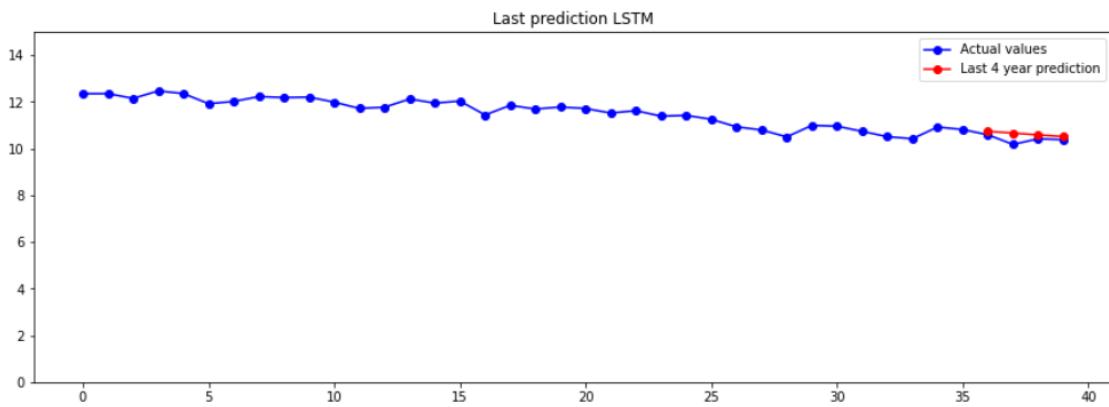
```

1 def predict_LSTM(train, test, n_neurons, n_epochs):
2     test['sum'] = test['mean_temp'] + test['ice_extent']
3
4
5     # define input sequence
6     in_seq1 = train.values[:,0]
7     in_seq2 = train.values[:,1]
8     out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
9
10    # convert to [rows, columns] structure
11    in_seq1 = in_seq1.reshape((len(in_seq1), 1))
12    in_seq2 = in_seq2.reshape((len(in_seq2), 1))
13    out_seq = out_seq.reshape((len(out_seq), 1))
14
15    # horizontally stack columns
16    dataset = hstack((in_seq1, in_seq2, out_seq))
17
18    # choose a number of time steps
19    n_steps_in, n_steps_out = 4, 4
20
21    # convert into input/output
22    X, y = split_sequences(dataset, n_steps_in, n_steps_out)
23
24    # the dataset knows the number of features, e.g. 2
25    n_features = X.shape[2]
26
27    # define model
28    model = Sequential()
29    model.add(LSTM(n_neurons, activation='relu', input_shape=(n_steps_in,
30        ↳ n_features)))
30    model.add(RepeatVector(n_steps_out))
31    model.add(LSTM(n_neurons, activation='relu', return_sequences=True))
32    model.add(TimeDistributed(Dense(n_features)))
33    model.compile(optimizer='adam', loss='mae')
34
35    # fit model
36    model.fit(X, y, epochs=n_epochs, verbose=0)
37
38    # demonstrate prediction
39    x_input = test.values
40    x_input = x_input.reshape((1, n_steps_in, n_features))

```

Figuur 3.32: Resultaat multivariate voorspelling LSTM

Mean MAE: $0.200 \times 1\ 000\ 000 \text{ km}^2$
MAE of last prediction: $0.210 \times 1\ 000\ 000 \text{ km}^2$
Execution time: 41.617 seconds



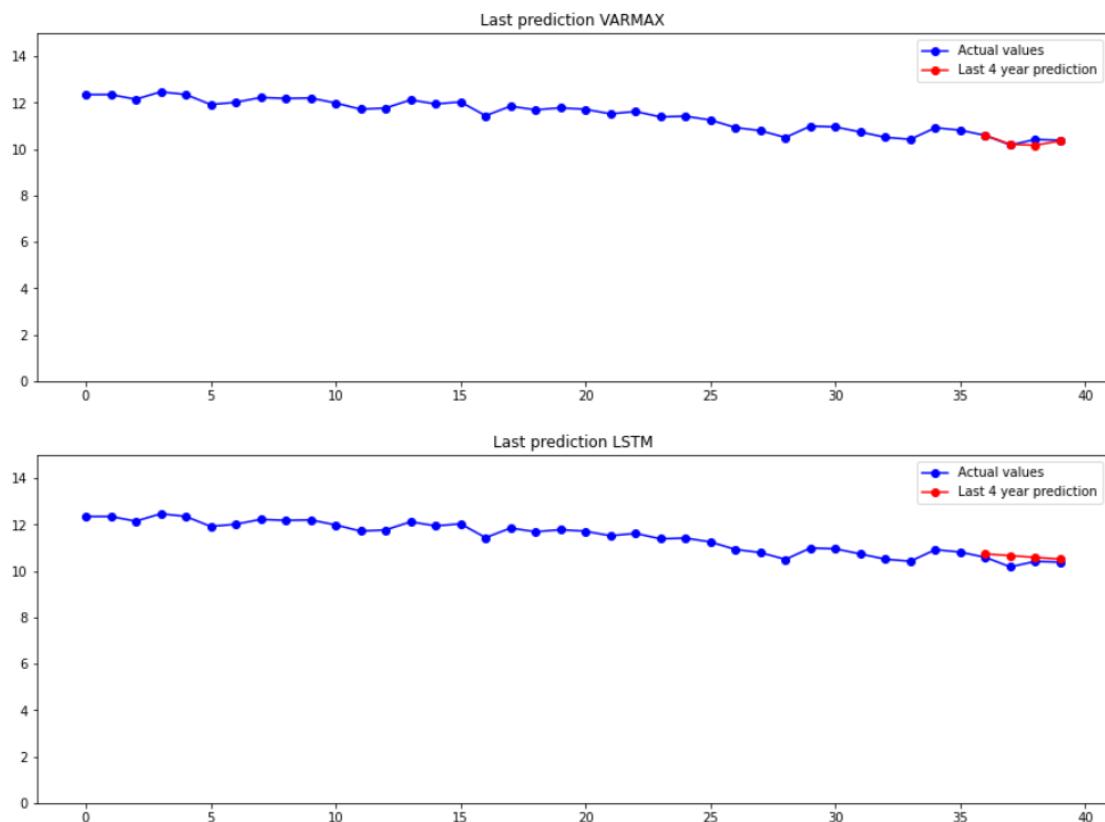
[0.14411208071264658, 0.24311231278801904, 0.20436435040655798, 0.20952587437320158]

```
41     yhat = model.predict(x_input, verbose=0)
42     return yhat
```

Figuur 3.33: Resultaten multivariate voorspelling

	Mean MAE ($\times 1\ 000\ 000 \text{ km}^2$)	Execution time (s)	Last MAE ($\times 1\ 000\ 000 \text{ km}^2$)
VARMAX	0.154	49.062	0.134
LSTM	0.198	26.708	0.210

Figuur 3.34: Grafische weergaven van de laatste voorspellingen van de VARMAX en LSTM modellen



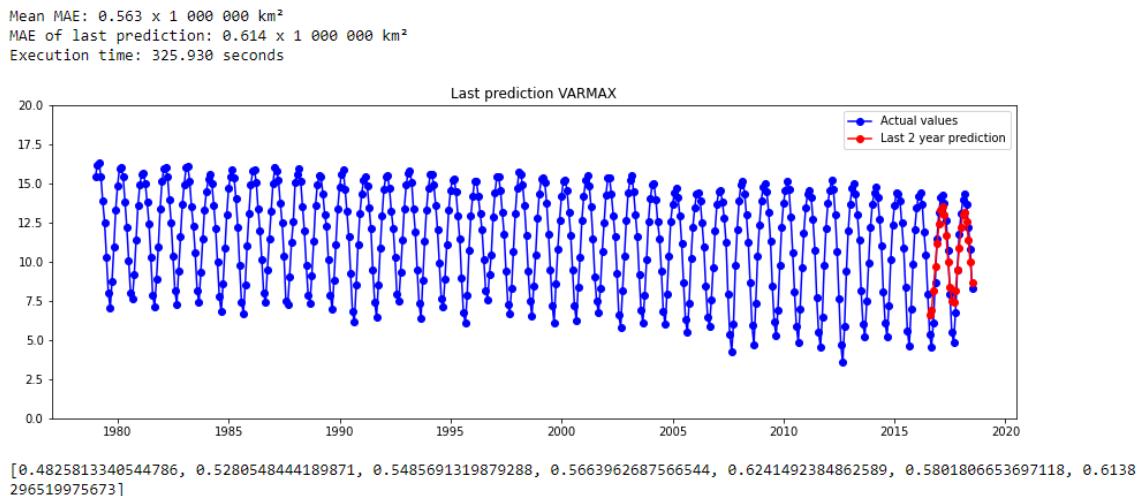
3.4.4 Evaluation

Wanneer we de resultaten van de multivariate niet-seizoensgebonden voorspellingen weer gegeven op Figuur 3.33 vergelijken kunnen we stellen dat VARMAX het best presteert volgens cross validation maar wel een hogere uitvoeringstijd heeft. Daarnaast zal de laatste voorspelling van het ijsoppervlak door het VARMAX model ook accurater zijn dan dat van het LSTM model. Deze laatste voorspelling wordt grafisch weergegeven op Figuur 3.34.

3.5 Multivariate seizoensgebonden

De laatste combinatie die getest zal worden is de dataset waar een seizoensverband en een extra variabele aanwezig is. Deze extra variabele zal andermaal de temperatuur zijn in een dataset die de maandelijkse temperatuur en ijsdikte zal bevatten. Hier zullen ook,

Figuur 3.35: Resultaat VARMAX bij random walk differentiatie



net als bij de niet seizoensgebonden data, enkel VARMAX en LSTM getest worden. Hier zal ook weer cross validation gebruikt worden. Er zal ook weer random walk en seasonal differentiatie gebruikt worden.

3.5.1 VARMAX

Random walk differentiatie

Wanneer we de hyperparameters bepalen bij dit VARMAX model blijken de parameters (4,1) de optimale waarden te zijn bij een range van 0 tot 5. Wanneer we deze invoeren om een uitgebreider resultaat te krijgen bekomen we de uitvoer die te beschouwen valt op Figuur 3.35.

Seizoensdifferentiatie

De optimale hyperparameters bij het VARMAX model dat opgebouwd wordt op basis van seizoensgedifferentieerde data zijn (0,1) wanneer de range van 0 tot 5 loopt. Het resultaat van dit model wordt afgebeeld op Figuur 3.36.

3.5.2 LSTM

Random walk differentiatie

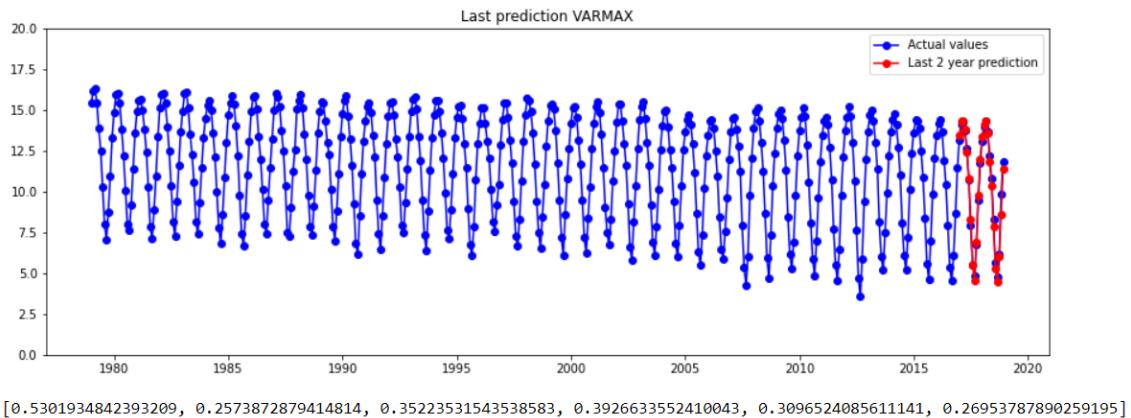
De optimale parameters bij Random walk differentiatie en bleken 30 neuronen te zijn en 300 epochs. Een hoger aantal neuronen en epochs zouden leiden tot een beter resultaat aangezien de MAE verlaagde naargelang deze parameters verhoogden maar er moest een grens getrokken worden om de uitvoeringstijd beperkt te houden. Wanneer we het model met de aangegeven parameters analyseren zullen we het resultaat bekomen dat op Figuur 3.37.

Seizoensdifferentiatie

De parameters die voor het beste resultaat zorgen bij het LSTM-model en seizoensdifferentieerde data zijn 1 voor het aantal neuronen en 200 voor het aantal epochs en in

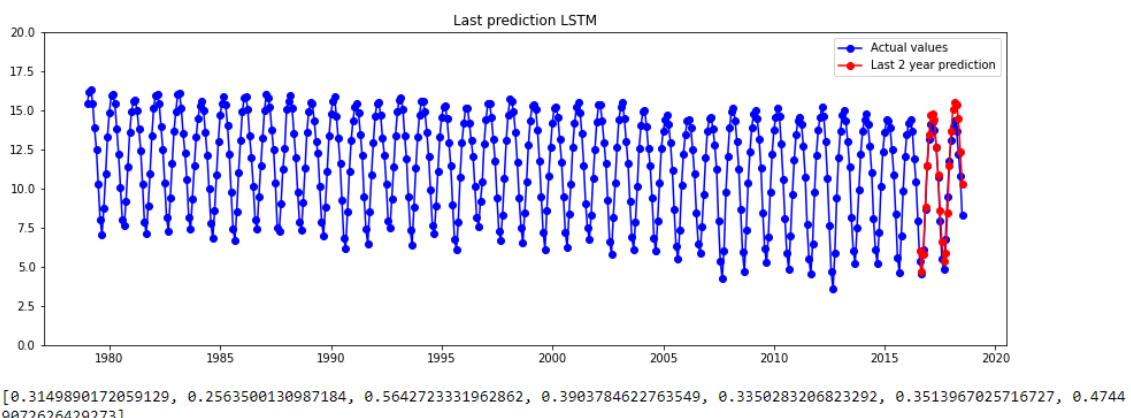
Figuur 3.36: Resultaat VARMAX bij seizoensorientatie

Mean MAE: $0.550 \times 1\ 000\ 000 \text{ km}^2$
 MAE of last prediction: $0.270 \times 1\ 000\ 000 \text{ km}^2$
 Execution time: 48.637 seconds



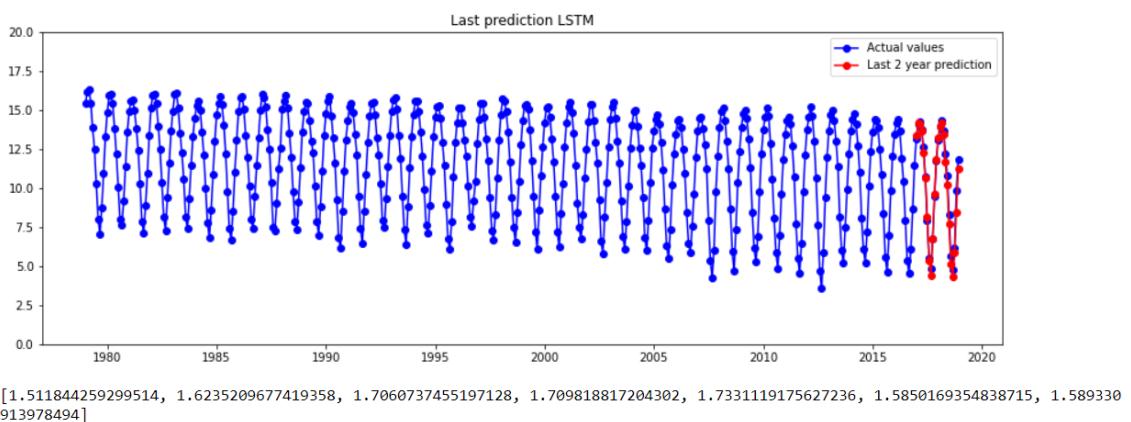
Figuur 3.37: Resultaat LSTM bij random walk differentiatie

Mean MAE: $0.384 \times 1\ 000\ 000 \text{ km}^2$
 MAE of last prediction: $0.474 \times 1\ 000\ 000 \text{ km}^2$
 Execution time: 313.415 seconds



Figuur 3.38: Resultaat LSTM seizoensdifferentiatie

Mean MAE: $0.550 \times 1\ 000\ 000 \text{ km}^2$
MAE of last prediction: $1.589 \times 1\ 000\ 000 \text{ km}^2$
Execution time: 153.886 seconds



tegenstelling tot de random walk differentiatie blijkt het verhogen van deze parameters niet te zorgen voor betere resultaten. Het resultaat van dit model wordt afgebeeld op Figuur 3.38.

Figuur 3.39: Resultaten multivariate seizoensgebonden voorspellingsmodellen

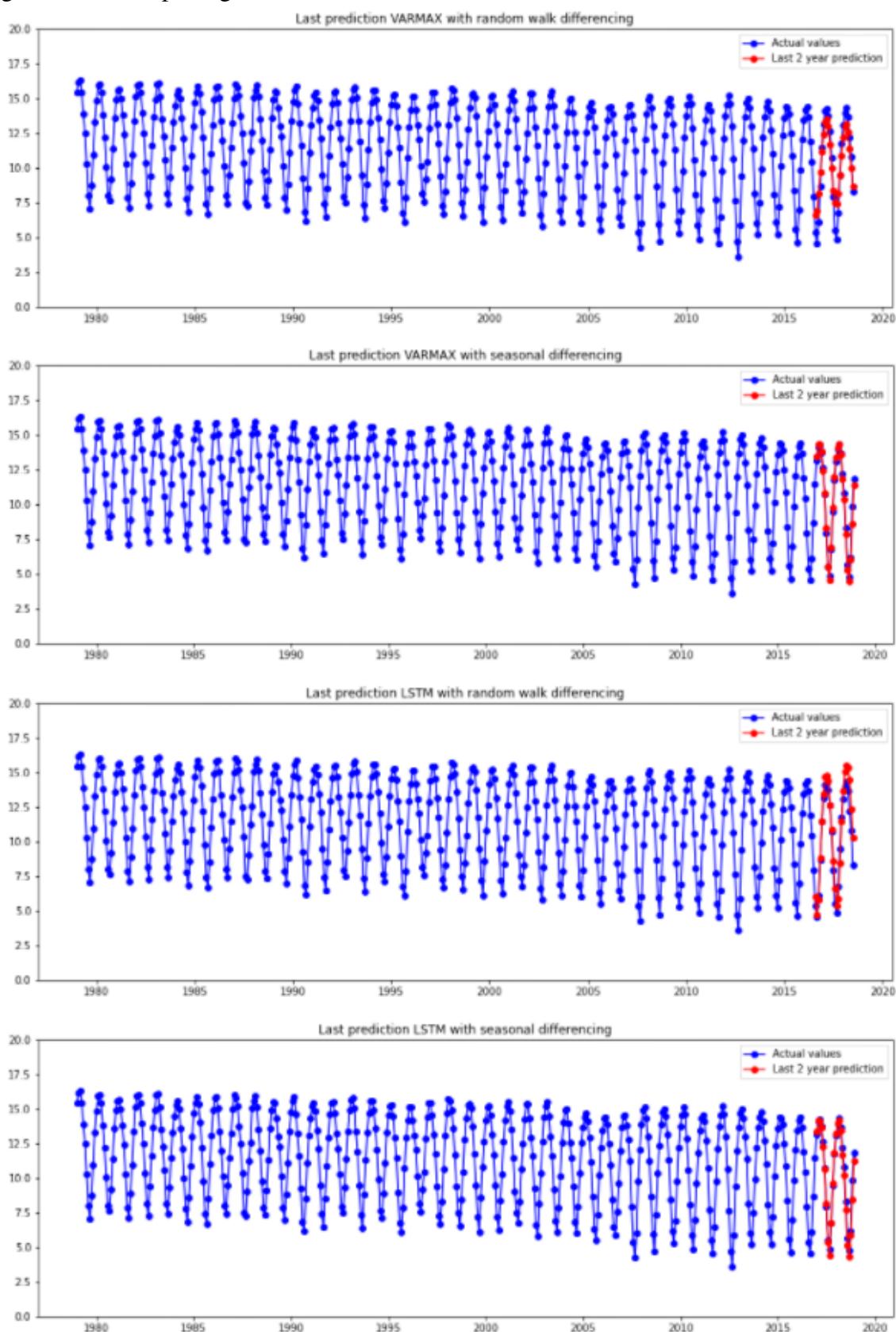
	Mean MAE ($\times 1\ 000\ 000 \text{ km}^2$)	Execution time (s)	Last MAE ($\times 1\ 000\ 000 \text{ km}^2$)
VARMAX	0.563	325.930	0.614
VARMAX_seasonal	0.550	48.637	0.270
LSTM	0.384	313.415	0.474
LSTM_seasonal	0.550	153.886	1.589

3.5.3 Evaluation

Wanneer we de resultaten op figuur 3.39 beschouwen kunnen we duidelijk aflezen dat LSTM met random walk differencing de laagste gemiddelde fout zal hebben en dus het best zal scoren wanneer we ons baseren op cross validation. Het scoort een stuk beter dan de andere modellen die allemaal een zeer gelijkaardige MAE hebben. Dit model zal echter wel de hoogste uitvoeringstijd hebben.

Het VARMAX model met seizoendifferentiatie zal de beste score behalen wanneer we naar de laatste voorspelling kijken en zal ook de laagste uitvoeringstijd hebben dus dit zou op vlak van uitvoeringssnelheid een goed alternatief zijn. De grafische weergaven van de laatste voorspellingen zijn zichtbaar op Figuur 3.40. De waarden tussen de modellen die random walk differentiatie gebruiken zullen ietwat verschillen van degene die gebruik maken van seizoendifferentiatie aangezien de data met 5 tijdstappen getransformeerd moest worden om tot evenwaardige testsetgroottes te komen. Dit was noodzakelijk om alle modellen met elkaar te vergelijken.

Figuur 3.40: Grafische weergaven van de laatste voorspellingen van multivariate seizoensgebonden voorspellingsmodellen



4. Conclusie

Wanneer er wordt gewerkt met cross validation en de Mean Average Error wordt gebruikt als foutmaat. Zullen de volgende modellen de beste voorspellingen kunnen maken van de evolutie van het ijsoppervlak op de polen:

- Bij de univariate niet-seizoensgebonden tijdreeks zal het ARIMA-model het beste resultaat behalen.
- Bij de univariate seizoensgebonden tijdreeks presteert het SARIMA-model met random walk differentiatie het best.
- Bij de multivariate niet-seizoensgebonden tijdreeks behaalt het VARMAX-model de beste resultaten.
- Bij de multivariate seizoensgebonden tijdreeks zal het LSTM model de beste prestatie leveren.

Mogelijke onderwerpen voor verder onderzoek zouden dieper in kunnen gaan op gevallen met externe regressoren waarbij bepaalde features in de tijdreeks ook in de testset meegegeven worden. Hiervoor zijn namelijk ingebouwde opties bij ARIMA- en Prophet-modellen. Daarnaast zou een uitgebreidere studie van LSTM-modellen bij deze data interessant kunnen zijn aangezien ze in deze bachelorproef slechts beperkt aan bod komen.

A. Onderzoeksvoorstel

A.1 Introductie

Artificiele intelligentie wordt steeds meer toegepast dus de optimale methodes bepalen om voorspellingen te maken is van vitaal belang. Verschillende datasets kunnen er volledig anders uitzien en deze kunnen dan ook op verschillende manieren ingedeeld worden. Voor dit onderzoek zal gefocust worden op data die tijdsgebonden is. Door het gebruik van dit type data zullen de modellen rekening moeten houden met de tijdsafhankelijkheid tussen de verschillende waarden.

A.2 Stand van zaken

Er zijn heel wat methoden die kunnen toegepast worden om een voorspelling te maken van tijdsgebonden data. Voor deze paper zullen enkel polynomiale vergelijkingen, ARIMA en LSTM getest worden.

De meest primitieve manier om een trend te voorspellen is het fitten van een polynomiale vergelijking op de trainingsdata en deze nadien toe te passen op de testdata. Daarnaast kan ook de ARIMA-methode (Brownlee, 2018a) gebruikt worden ofwel het Autoregressive Integrated Moving Average. Deze methode combineert autoregressie en voortschrijdend gemiddelde. Autoregressie modeert de volgende stap in een sequentie als een lineaire functie van de waarden uit voorgaande tijdsintervallen. De methode van het voortschrijdend gemiddelde modeert de volgende stap in de sequentie als een lineaire functie van de resterende fouten van een gemiddeld proces bij voorgaande tijdsintervallen. Er moet ook opgemerkt worden dat er een verschil is tussen een model met een voortschrijdend gemiddelde en het voortschrijdend gemiddelde van de dataset zelf.

Ook neurale netwerken kunnen toegepast worden bij het maken van voorspellingen van tijdsreeksen. LSTM (Long Short Term Memory) is een vaak gebruikt modeltype om tijd-

reeksen te voorspellen. Dit model zal het verloop van de volgende waarden voorspellen op basis van de ingevoerde waarden rekening houdend met de chronologie waarin ze voorkomen. Hierbij zal de invloed van oudere waarden minder relevant worden naargelang er meer waarden ingevoerd worden.

Op zowel de ARIMA als de LSTM modellen bestaan er varianten om multivariate tijdreeksen te voorspellen, bij ARIMA worden deze benoemd als VARMAX modellen. Ook bij polynomiale regressie kunnen multivariate times series voorspeld worden. Ook voor tijddreksdata waar een duidelijk seizoenseffect zichtbaar is bestaat er een variant op het ARIMA model genaamd SARIMA.

A.3 Methodologie

Om na te gaan welke methodes de beste resultaten behalen zullen zowel polynomiale regressie, ARIMA en LSTM toegepast worden op 2 datasets, 1 waarbij een duidelijk seizoenseffect zichtbaar is en 1 waar geen duidelijke seisoensgebonden invloed aanwezig is. Daarnaast zullen ook al deze methodes of gespecialiseerdere varianten van deze methodes toegepast worden op datasets met en zonder seizoeneffect waarbij meerdere invoerparameters gebruikt zullen worden.

Om deze methodes te scoren zullen de laatste waarden weggelaten en voorspeld worden waardoor uit de foutmarge tussen de voorspellingen en de werkelijke waarden afgeleid zal kunnen worden welke methode de meest accurate voorspelling zal kunnen maken. Om deze methodes te quoteren zullen de r^2 en de RMPSE (Root Mean Square Percentage Error) scoringsmethodes benut worden.

A.4 Verwachte resultaten

Er valt te verwachten dat polynomiale regressie het zwakste resultaat zal behalen aangezien polynomiale technieken, door de aard van een veelterm, doorgaans minder goed zijn voor extrapolatie waarvoor ze in deze context benut zullen worden. Ik verwacht dat LSTM best zal scoren aangezien dit type model specifiek voor tijddreksen is opgesteld gevuld door ARIMA.

A.5 Verwachte conclusies

Er valt te verwachten dat de voorgestelde technieken goede resultaten zullen behalen. Vooral voor LSTM liggen mijn verwachtingen vrij hoog omdat ik reeds een artikel (**Siami-Namini2018**) heb gelezen waarbij de voorspellingen voor LSTM accurater zijn. Ik heb een pak minder vertrouwen in polynomiale regressie aangezien deze techniek minder goed presteert bij extrapolatie en maar beter tot zijn recht komt bij interpolatie. ARIMA zal waarschijnlijk ook goede resultaten behalen.

B. Broncode

Hier staat alle code afgebeeld die gebruikt is voor het opstellen van deze bachelorproef. De broncode kan ook teruggevonden worden op dit adres: https://github.com/DeclercqEmiel/Notebooks_BP

B.1 Broncode datavoorbereiding

Listing 23: Broncode datavoorbereiding

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[2]:
5
6
7  import pandas as pd
8  import matplotlib.pyplot as plt
9  import numpy as np
10
11
12  # # Dataset exploration
13
14  # ## Dataset #1: seaice
15
16  # source: https://www.kaggle.com/nsidcorg/daily-sea-ice-extent-data
17
18  # In[3]:
19
20
21  ice = pd.read_csv('./data/seaice.csv')
```

```

22 ice.columns = ['Year', 'Month', 'Day', 'Extent', 'Missing', 'Source Data',
23 'hemisphere']
24 ice
25
26
27 # In[5]:
28
29
30 plt.plot(ice[['Extent']])
31
32
33 # In[6]:
34
35
36 ice.groupby('hemisphere').count()
37
38
39 # In[7]:
40
41
42 ice.groupby('Year').count()[['Extent']]
43
44
45 # In[8]:
46
47
48 ice.groupby('Year').mean()[['Extent']]
49
50
51 # In[9]:
52
53
54 plt.scatter(ice.groupby('Year').mean()[['Extent']][:-1].index,
55     ↳ ice.groupby('Year').mean()[['Extent']][:-1])
56
57 # In[10]:
58
59
60 ice.groupby(['Year', 'hemisphere']).count().tail(60)
61
62
63 # In[11]:
64
65
66 plt.xlabel('Years')
67 plt.ylabel('Extent')
68 plt.plot(ice[ice['hemisphere'] ==
69     ↳ 'north'].groupby('Year').mean()[['Extent']][:-1].index, ice[ice['hemisphere'] ==
70     ↳ 'north'].groupby('Year').mean()[['Extent']][:-1], label='Northern
71     ↳ hemisphere')
72 plt.plot(ice[ice['hemisphere'] ==
73     ↳ 'south'].groupby('Year').mean()[['Extent']][:-1].index, ice[ice['hemisphere'] ==
74     ↳ 'south'].groupby('Year').mean()[['Extent']][:-1], label='Southern
75     ↳ hemisphere')
76 plt.legend()

```

```
71
72
73 # In[12]:
74
75
76 plt.xlabel('Years')
77 plt.ylabel('Extent')
78 plt.scatter(ice.groupby('Year').mean()[['Extent']][:-1].index,
79             ice.groupby('Year').mean()[['Extent']][:-1])
80
81 # In[ ]:
82
83
84
85
86
87 # In[13]:
88
89
90 print('start : ' + str(ice['Year'][0]))
91 print('end : ' + str(ice['Year'].tail(1).iloc[0]))
92
93
94 # In[14]:
95
96
97 2019-1978
98
99
100 # ## Dataset #1: Toronto_temp
101
102 # In[8]:
103
104
105 tt
106
107
108 # In[4]:
109
110
111 # Source: https://www.kaggle.com/rainbowgirl/climate-data-toronto-19372018
112 tt = pd.read_csv('./data/Toronto_temp.csv')
113 tt = tt[tt['Day'] == 1]
114 tt['Year'] = tt['Year'].replace({'2,013':'2013',
115 '2,014':'2014',
116 '2,015':'2015',
117 '2,016':'2016',
118 '2,017':'2017',
119 '2,018':'2018'})
120 # tt.groupby('Year').count()
121 tt = tt[(tt['Year'] != '1937')]
122 ttt = tt.groupby('Year').count()
123 #ttt.head(50)
124 #ttt.groupby('Year').count().tail(50)
125 meantt = tt.groupby('Year').mean()['Mean Temp (C)']
```

```

126 meantt
127 #meantt.index
128 #meantt
129 meantt.sort_index(inplace=True)
130
131 plt.xlabel('Years')
132 plt.ylabel('Temperature (C)')
133 plt.xticks(np.array(range(0,meantt.size,10)))
134 plt.scatter(meantt.index, meantt)
135
136 print('start : ' + meantt.index[0])
137 print('end : ' + meantt.index[-1])
138
139 new_row = pd.Series({'Mean Temp (C)' : 0.555556, 'Year': '2018', 'Month':12})
140 tt = tt.append(new_row, ignore_index=True)
141 tt['Year'] = tt['Year'].astype(int)
142 mean_temp_monthly = tt[['Year', 'Month', 'Mean Temp
   (C)']].set_index(['Year', 'Month']).sort_index()
143 # mean_temp_monthly
144 mean_temp_monthly =
   mean_temp_monthly[mean_temp_monthly.index.get_level_values(0).astype(int) >=
   1979 ]
145 mean_temp_monthly
146
147
148 # In[6]:
149
150
151 tt
152
153
154 # ## Dataset #3: seaice2
155
156 # Completer version of dataset #1
157 #
158 # source: https://nsidc.org/arcticseaicenews/sea-ice-tools/
159
160 # In[13]:
161
162
163 ice2.mean()[1:-2]
164
165
166 # In[17]:
167
168
169 ice2_mean
170
171
172 # In[19]:
173
174
175 ice2.mean()[1:-2]
176
177
178 # In[24]:

```

```
179
180
181 ice2.mean()
182
183
184 # In[26]:
185
186
187 ice2
188
189
190 # In[27]:
191
192
193 ice2 = pd.read_csv('./data/seaice2.csv')
194 # ice2
195 ice2_mean = ice2.mean()[1:-2]
196 ice2_mean
197 ice2_mean.index = ice2_mean.index.values.astype(int)
198
199 plt.title('Yearly ice extent')
200 plt.scatter(ice2_mean.index,ice2_mean)
201 plt.xlabel('Years')
202 plt.ylabel('Extent')
203 plt.show()
204
205 # ice2['2018']
206 #
207 # → pd.concat([ice2['2016'],ice2['2017'],ice2['2018'],ice2['2019']]).reset_index()[0]
208 # ice2[['2018']].append(ice2[['2019']])
209 ice2.rename(columns={'Unnamed: 0' : 'Month', 'Unnamed: 1' : 'Day'}, inplace =
210 # → True)
211 ice2.drop([' ', '1981-2010', 'Day', '1978', '2020'],axis=1,inplace=True)
212 values = ice2.values
213 i = 0
214 for row in values :
215 if type(row[0]) != str :
216 values[i][0] = month
217 else:
218 month = row[0]
219 i = i +1
220 # ice2.columns.values
221 ice2_clean = pd.DataFrame(values)
222 ice2_clean.columns = ice2.columns.values
223 # ice2_clean.head(5)
224 ice2_monthly_mean =
225 # → ice2_clean.set_index('Month').astype(float).groupby('Month',sort=False).mean()
226 # ice2_monthly_mean
227 # ice2_monthly_mean.T.stack().index.get_level_values(0)
228 #
229 # → ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
230 ice2_monthly_mean_chron =
231 # → ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
232 # ice2.columns.size
233 plt.title('Monthly ice extent')
234 plt.plot(ice2_monthly_mean_chron.values)
```

```

230 plt.xticks(np.array(range(0,500,75)))
231 plt.xlabel('Cumulative month')
232 plt.ylabel('Extent')
233 plt.show()
234
235 # np.unique(ice2_monthly_mean_chron.index.values).size*12
236 print('from ' + ice2_monthly_mean_chron.index.values[0] + ' until ' +
237   ice2_monthly_mean_chron.index.values[-1])
238 ice2_monthly_mean_chron =
239   ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
240 ice2_monthly_mean_chron.columns = ['ice_extent']
241 ice2_monthly_mean_chron
242
243
244 # # Dataset Combination
245
246
247 ice2_monthly_mean_chron_cut = ice2_monthly_mean_chron[:-12]
248 # ice2_monthly_mean_chron
249 # ice2_monthly_mean_chron_cut
250 # mean_temp_monthly
251 # ice2_monthly_mean_chron_cut
252 combined = mean_temp_monthly[mean_temp_monthly.index.get_level_values(0) >= 1979]
253 combined['ice_extent'] = ice2_monthly_mean_chron_cut.values
254 # combined
255 combined.rename(columns={'Mean Temp (C)': 'mean_temp'}, inplace=True)
256 dataframe_monthly = combined
257 # dataframe_monthly
258 # dataframe_monthly[['mean_temp']]
259 plt.plot(dataframe_monthly[['mean_temp']].values[-24:],label='temperature')
260 plt.plot(dataframe_monthly[['ice_extent']].values[-24:],label='ice extent')
261 plt.legend()
262 plt.show()
263 dataframe_yearly = combined.groupby('Year').mean()
264 # dataframe_yearly
265 # dataframe_monthly[['mean_temp']].values
266 plt.plot(dataframe_monthly[['mean_temp']].values,label='temperature')
267 plt.plot(dataframe_monthly[['ice_extent']].values,label='ice extent')
268 plt.legend()
269 dataframe_monthly.to_csv('./data/dataframe_monthly.csv')
270 dataframe_yearly.to_csv('./data/dataframe_yearly.csv')

```

B.2 Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij univariate, niet-seizoensgebonden tijdreeksen

Listing 24: code voor differentiatie

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Imports
5
6 # In[1]:
7
8
9 import pandas as pd
10 import numpy as np
11 import matplotlib.pyplot as plt
12 get_ipython().run_line_magic('matplotlib', 'inline')
13 from matplotlib.pyplot import rcParams
14 rcParams['figure.figsize'] = 15,5
15 from sklearn.model_selection import TimeSeriesSplit
16 from sklearn.metrics import mean_absolute_error
17 import warnings
18 from statsmodels.tsa.arima_model import ARIMA
19 import timeit
20 from numpy import array
21 from keras.models import Sequential
22 from keras.layers import LSTM
23 from keras.layers import Dense
24 import tensorflow as tf
25 from fbprophet import Prophet
26
27
28 # # Dataprep
29
30 # In[2]:
31
32
33 # read time series
34 ts = pd.read_csv('./data/dataframe_yearly.csv', index_col=0, usecols=[0,2])
35
36 # print out first values
37 ts.head()
38
39
40 # In[3]:
41
42
43 # plot time series
44 plt.plot(ts)
45
46
47 # ## Differentiatie
48
49 # In[4]:
50
51
52 # define method to visualise the stationarity of a time series
53 def test_stationarity(timeseries):
54     #Determining rolling statistics
```

```

56     rolmean = timeseries.rolling(12).mean()
57     rolstd = timeseries.rolling(12).std()
58
59     #Plot rolling statistics:
60     orig = plt.plot(timeseries, color='blue',label='Original')
61     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
62     std = plt.plot(rolstd, color='black', label = 'Rolling Std')
63     plt.legend(loc='best')
64     plt.title('Rolling Mean & Standard Deviation')
65     plt.show(block=False)
66
67 # check stationarity of time serie
68 test_stationarity(ts)
69
70
71 # In[5]:
72
73
74 # take the random walk difference of the time serie
75 ts_diff = ts - ts.shift(1)
76 ts_diff = ts_diff.dropna()
77
78 # display stationarity of the newly differenced time serie
79 test_stationarity(ts_diff)
80
81
82 # ## Cross validation setup
83
84 # In[6]:
85
86
87 # initialize TimeSeriesSplit object
88 tscv = TimeSeriesSplit(n_splits = 8)
89
90 # loop trough all split time series that have a trainingsset with more than 20
# → values
91 for train_index, test_index in tscv.split(ts_diff):
92     if train_index.size > 20:
93
94         # initialize cross validation train and test sets
95         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
96
97         # visualize cross_validation structure for reference
98         print("TRAIN:", train_index.size)
99         print("TEST:", test_index.size)
100        print()
101
102
103 # ## General methods
104
105 # In[7]:
106
107
108 # plot
109 plt.plot(ts, marker='o', color='blue',label='Actual values')
110 plt.ylim([0,15])

```

```
111 plt.legend()
112
113 plt.show()
114
115
116 # In[8]:
117
118
119 # define functions used throughout the notebook
120
121 # define function for plotting last prediction and the actual data
122 def full_graph(predicted_diff, title):
123
124     # format predictions by adding NaN values in front
125     predictionsArray = np.asarray(revert_diff(predicted_diff, ts))
126     zerosArray = np.zeros(ts.values.size-len(predictionsArray.flatten()))
127     cleanPrediction =
128         pd.Series(np.concatenate((zerosArray,predictionsArray))).replace(0,np.NaN)
129     cleanPrediction.index = ts.index.values
130
131     # plot
132     plt.title(title)
133     plt.plot(ts, marker='o', color='blue',label='Actual values')
134     plt.plot(cleanPrediction, marker='o', color='red',label='Last 4 year
135         prediction')
136     plt.ylim([0,15])
137     plt.legend()
138
139     plt.show()
140
141     # define function for reverting a differenced dataset
142     def revert_diff(predicted_diff, og_data):
143
144         # retrieve last value
145         last_value = og_data.iloc[-predicted_diff.size-1][0]
146
147         # initialize reverted array
148         predicted_actual = np.array([])
149
150         # add each value in the differenced array with the last actual value
151         for value_diff in predicted_diff:
152             actual_value = last_value + value_diff
153             predicted_actual = np.append(predicted_actual, actual_value)
154             last_value = actual_value
155
156
157     # # ARIMA
158
159     # In[9]:
160
161
162     # ARIMA
163     from statsmodels.tsa.arima_model import ARIMA
164     import itertools
```

```

165 import warnings
166 import sys
167 from sklearn.metrics import mean_absolute_error
168
169
170
171 # Define the p, d and q parameters to take any value between 0 and 2
172 p = q = range(0, 5)
173 d = range(0,3)
174
175 # Generate all different combinations of p, q and q triplets
176 pdq = list(itertools.product(p, d, q))
177
178 # initialize variables
179 best_pdq = pdq
180 best_mean_mae = np.inf
181
182 # specify to ignore warning messages to reduce visual clutter
183 warnings.filterwarnings("ignore")
184
185 # loop trough all possible parameter combinations of pdq
186 for param in pdq:
187     print(param)
188
189     # some parametercombinations might lead to crash, so catch exceptions and
190     # continue
191     try:
192
193         # initialize the array which will contain the mean average errors
194         maes = []
195
196         # loop trough all split time series that have a trainingsset with more
197         # than 20 values
198         for train_index, test_index in tscv.split(ts_diff):
199             if train_index.size > 20:
200
201                 # initialize cross validation train and test sets
202                 cv_train, cv_test = ts_diff.iloc[train_index],
203                 cv_test = ts_diff.iloc[test_index]
204
205                 # build model
206                 model = ARIMA(cv_train, order=(param))
207
208                 # fit model
209                 model_fit = model.fit()
210
211                 # make predictions
212                 predictions = model_fit.predict(start=len(cv_train),
213                 end=len(cv_train)+cv_test.size-1, dynamic=False)
214
215                 # renaming for clarity
216                 prediction_values = predictions.values
true_values = cv_test.values
217
218                 # error calculation this part of the cross validation
maes.append(mean_absolute_error(true_values, prediction_values))

```

```

217
218
219     # error calculation for this parameter combination
220     mean_mae = np.mean(maes)
221     print('MAE: ' + str(mean_mae))
222
223     # store parameters resulting in the lowest mean MAE
224     if mean_mae < best_mean_mae:
225         best_mean_mae = mean_mae
226         best_maes = maes
227         best_pdq = param
228         best_predictions = prediction_values
229
230     except Exception as e:
231         print(e)
232         continue
233
234     # logging
235     print()
236     print('Best MAE = ' + str(best_mean_mae))
237     print(best_pdq)
238
239     # best range(0,10):
240
241
242 # In[10]:
243
244
245 # # result
246 # best_pdq = (3, 0, 0)
247
248
249 # In[11]:
250
251 start_time = timeit.default_timer()
252
253 # specify to ignore warning messages
254 warnings.filterwarnings("ignore")
255
256 print("----")
257
258 # initialize the array which will contain the mean average errors
259 maes = []
260
261 # loop trough all split time series that have a trainingsset with more than 20
262 # → values
263 for train_index, test_index in tscv.split(ts_diff):
264     if train_index.size > 20:
265
266         # initialize cross validation train and test sets
267         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
268
269         # build model
270         arima = ARIMA(cv_train, best_pdq).fit(start_ar_lags=1, disp=False)
271

```

```

272     # make predictions
273     predictions = arima.forecast(steps=4)
274     prediction_values = predictions[0]
275     true_values = cv_test.values
276
277     # error calc
278     maes.append(mean_absolute_error(true_values, prediction_values))
279
280     # last actual prediction
281     last_prediction_ARIMA = prediction_values
282
283     print("I",end="")
284
285     # store results to variables
286     time_ARIMA = timeit.default_timer() - start_time
287     mae_mean = np.mean(maes)
288     MAE_ARIMA = mae_mean
289     last_MAE_ARIMA = maes[-1]
290
291     # logging
292     print()
293     print('Mean MAE: %.3f x 1 000 000 km' % MAE_ARIMA)
294     print('MAE of last prediction: %.3f x 1 000 000 km' % last_MAE_ARIMA)
295     print('Execution time: %.3f seconds' % time_ARIMA)
296     full_graph(last_prediction_ARIMA, 'Last prediction ARIMA')
297     print('Mean average errors:')
298     print(maes)
299
300
301     # # LSTM
302
303     #
304     # → https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/
305
306     # ## Functions
307
308     # In[12]:
309
310     from keras.layers import Dropout
311     # split a univariate sequence into samples
312     def split_sequence(sequence, n_steps_in, n_steps_out):
313         X, y = list(), list()
314         for i in range(len(sequence)):
315             # find the end of this pattern
316             end_ix = i + n_steps_in
317             out_end_ix = end_ix + n_steps_out
318
319             # check if we are beyond the sequence
320             if out_end_ix > len(sequence):
321                 break
322
323             # gather input and output parts of the pattern
324             seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
325             X.append(seq_x)
326             y.append(seq_y)

```

```
327     return array(X), array(y)
328
329 def build_model(raw_seq, n_steps_in, n_steps_out, n_features, n_neurons, dropout,
330   ↵ batch_s):
330
331     # split into samples
332     X, y = split_sequence(raw_seq.values.flatten(), n_steps_in, n_steps_out)
333
334     # reshape from [samples, timesteps] into [samples, timesteps, features]
335     X = X.reshape((X.shape[0], X.shape[1], n_features))
336
337     # define model
338     model = Sequential()
339     model.add(LSTM(n_neurons, activation='relu'))
340     model.add(Dropout(dropout))
341     model.add(Dense(n_steps_out))
342     model.compile(optimizer='adam', loss='mae')
343
344     # fit model
345     model.fit(X, y, batch_size=batch_s, epochs=100, verbose=0)
346
347     return model
348
349
350 def predict(x_input, model, n_features):
351     n_features = 1
352
353     # reshape data
354     x_input = x_input.reshape((1, n_steps_in, n_features))
355
356     # predict
357     yhat = model.predict(x_input, verbose=0)
358
359     return yhat
360
361
362 # ## Determine hyperparameters
363
364 # In[24]:
365
366
367 # Disabled tf warning because of visual clutter
368 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
369
370
371 # constant variables
372 n_steps_in = 4
373 n_steps_out = 4
374 n_features = 1
375 maes = []
376 global_maes = []
377
378 # optimizable variables
379 n_neurons_array = [1,10,20]
380 dropout_array = [0,0.5,0.99]
381 batch_size_array = [1,8]
```

```

382
383
384 # initialize values
385 best_MAE = 100
386 best_n_neurons = 0
387 best_activation = 'none'
388 best_dropout = 0
389 best_batch_size = 0
390
391 # loop over all possible parameter combinations
392 for n_neurons in n_neurons_array:
393     for dropout in dropout_array:
394         for batch_size in batch_size_array:
395
396             print("----")
397
398             # loop through all split time series that have a trainingsset with
399             # more than 20 values
400             for train_index, test_index in tscv.split(ts_diff):
401                 if train_index.size > 20:
402
403                     # initialize cross validation train and test sets
404                     y_train, y_test = ts_diff.iloc[train_index],
405                     # ts_diff.iloc[test_index]
406
407                     # build model
408                     lstm_model = build_model(y_train, n_steps_in, n_steps_out,
409                     # n_features, n_neurons, dropout, batch_size)
410
411                     # make predictions
412                     x_input = array(y_test)
413                     y_predicted = predict(x_input, lstm_model,
414                     # n_features).flatten()
415                     y_actual = y_test.values
416
417                     # error calculation this part of the cross validation
418                     maes.append(mean_absolute_error(y_actual, y_predicted))
419
420                     print("I",end="")
421
422                     # last actual prediction
423                     last_prediction_LSTM = y_predicted
424
425
426                     # error calculation for this parameter combination
427                     MAE_LSTM = np.mean(maes)
428                     last_MAE_LSTM = maes[-1]
429                     global_maes.append(MAE_LSTM)
430
431
432                     # store parameters resulting in the lowest mean MAE
433                     if best_MAE > MAE_LSTM:
434                         best_n_neurons = n_neurons
435                         best_dropout = dropout
436                         best_batch_size = batch_size
437                         best_MAE = MAE_LSTM
438
439                     # log values for parameter combination

```

```
434     print()
435     print(n_neurons)
436     print(dropout)
437     print(batch_size)
438     print(MAE_LSTM)
439     print()
440
441 # log parameter combination with best result
442 print('Best:')
443 print('N neurons')
444 print(best_n_neurons)
445 print('Dropout rate')
446 print(best_dropout)
447 print('Batch size')
448 print(best_batch_size)
449 print('MAE')
450 print(best_MAE)
451 plt.bar(range(0,len(global_maes)), global_maes)
452
453
454 # In[25]:
455
456
457 # Run #1 Best:
458 # 1
459 # 0
460 # 1
461 # 0.1919512481592649
462 # Wall time: 8min 36s
463
464 # Run #2 Best:
465 # 1
466 # 0.5
467 # 1
468 # 0.19969739902546374
469 # Wall time: 7min 16s
470
471 # Run #3 Best:
472 # 5
473 # 0.99
474 # 8
475 # 0.20111841616444215
476 # Wall time: 6min 47s
477
478
479 # In[26]:
480
481
482 best_n_neurons = 1
483 best_dropout = 0
484 best_batch_s = 8
485
486
487 # ## Final result LSTM
488
489 # In[27]:
```

```

490
491 start_time = timeit.default_timer()
492
493 # Disabled tf warning because of visual clutter
494 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
495
496
497 # constant variables
498 n_steps_in = 4
499 n_steps_out = 4
500 n_features = 1
501 maes = []
502
503
504 # optimizable variables
505 n_neurons = best_n_neurons
506 dropout = best_dropout
507 batch_s = best_batch_s
508
509 print("----")
510
511 # loop trough all split time series that have a trainingsset with more than 20
512 # values
513 for train_index, test_index in tscv.split(ts_diff):
514     if train_index.size > 20:
515         # initialize cross validation train and test sets
516         y_train, y_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
517
518         # build model
519         lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features,
520             n_neurons, dropout, batch_s)
521
522         # make predictions
523         x_input = array(y_test)
524         y_predicted = predict(x_input, lstm_model, n_features).flatten()
525         y_actual = y_test.values
526
527         # error calc
528         maes.append(mean_absolute_error(y_actual, y_predicted))
529
530         print("I", end="")
531
532         # last actual prediction
533         last_prediction_LSTM = y_predicted
534
535         # store variables
536         time_LSTM = timeit.default_timer() - start_time
537         MAE_LSTM = np.mean(maes)
538         last_MAE_LSTM = maes[-1]
539
540         # visualisation
541         print()
542         print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
543         print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
544         print('Execution time: %.3f seconds' % time_LSTM)

```

```
544 full_graph(last_prediction_LSTM, 'Last prediction LSTM')
545 print('Mean average errors')
546 print(maes)
547
548
549 # # Prophet
550
551 # In[28]:
552
553
554 # formatting dataframe
555 ts_formated_prophet = ts_diff.reset_index().rename(columns = {'Year' : 'ds',
556   ↳ 'ice_extent' : 'y'})
556 ts_formated_prophet['ds'] =
557   ↳ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds']).astype(str),
558     ↳ format='%Y'))
559
560
561 # In[38]:
562
563
564 # Python
565 import itertools
566 import numpy as np
567 import pandas as pd
568
569 # define dataframe
570 df = ts_formated_prophet
571
572 param_grid = {
573   'changepoint_prior_scale': [0.001, 0.01, 0.1, 1, 2, 5, 10, 15, 20, 25],
574 }
575
576 # Generate all combinations of parameters
577 all_params = [dict(zip(param_grid.keys(), v)) for v in
578   ↳ itertools.product(*param_grid.values())]
579
580 # initialize variables
581 maes = []
582 global_maes = []
583 best_MAE_prophet = np.inf
584
585 # Use cross validation to evaluate all parameters
586 for params in all_params:
587
588   # loop trough all split time series that have a trainingsset with more than
589   ↳ 20 values
590   for train_index, test_index in tscv.split(ts_formated_prophet):
591     if train_index.size > 20:
592
593       # initialize cross validation train and test sets
594       train = ts_formated_prophet.iloc[train_index]
595       y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
596       X_test = ts_formated_prophet.iloc[test_index][['ds']]
597
598       # Fit model with given params
```

```

595     model = Prophet(**params, weekly_seasonality=False,
596                      daily_seasonality=False)
597     model = model.fit(train)
598
599     # make predictions
600     forecast = model.predict(X_test)
601     y_pred = forecast['yhat'].values
602
603     # last actual prediction
604     last_prediction_prophet = y_pred
605
606     # error calculation this part of the cross validation
607     maes.append(mean_absolute_error(y_test, y_pred))
608
609     # error calculation for this parameter combination
610     MAE_prophet = np.mean(maes)
611     last_MAE_prophet = maes[-1]
612     global_maes.append(MAE_prophet)
613
614     # logging
615     print('changepoint_prior_scale: ' + str(params['changepoint_prior_scale']))
616
617     # store parameters resulting in the lowest mean MAE
618     if best_MAE_prophet > MAE_prophet:
619         best_params = params
620         best_MAE_prophet = MAE_prophet
621
622     # log optimal result
623     print('changepoint_prior_scale: ' + str(best_params['changepoint_prior_scale']))
624     print(best_MAE_prophet)
625
626
627 # In[39]:
628
629
630 # best_params = {'changepoint_prior_scale': 2}
631
632 # In[40]:
633
634
635
636 # Disabled tf warning because of clutter
637 warnings.filterwarnings("ignore") # specify to ignore warning messages
638
639 start_time = timeit.default_timer()
640
641 # initialize variables
642 maes = []
643
644 for train_index, test_index in tscv.split(ts_formated_prophet):
645     if train_index.size > 20:
646
647         # initialize cross validation train and test sets
648         train = ts_formated_prophet.iloc[train_index]
649         y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()

```

```
650     X_test = ts_formated_prophet.iloc[test_index][['ds']]
651
652     # build model
653     model = Prophet(**best_params, weekly_seasonality=False,
654                      daily_seasonality=False)
655     model.fit(train)
656
657     # make predictions
658     forecast = model.predict(X_test)
659     y_pred = forecast['yhat'].values
660
661     # error calc
662     maes.append(mean_absolute_error(y_test, y_pred))
663
664     # last actual prediction
665     last_prediction_prophet = y_pred
666
667     # store results
668     time_Prophet = timeit.default_timer() - start_time
669     MAE_Prophet = np.mean(maes)
670     last_MAE_Prophet = maes[-1]
671
672     # visualize results
673     print()
674     print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_Prophet)
675     print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_Prophet)
676     print('Execution time: %.3f seconds' % time_Prophet)
677     full_graph(last_prediction_prophet, "Last 4 year prediction prophet")
678     print('Mean average errors')
679     print(maes)
680
681
682     # # Evaluation
683
684     # In[33]:
685
686
687     # formatting
688     results =
689     [[MAE_ARIMA, time_ARIMA, last_MAE_ARIMA], [MAE_LSTM, time_LSTM, last_MAE_LSTM], [MAE_Prophet, time_Prophet]]
690
691     # display results
692     pd.DataFrame(results, columns=['Mean MAE (x 1 000 000 km\u00b2)', 'Execution time
693     (s)', 'Last MAE (x 1 000 000
694     km\u00b2)'], index=['ARIMA', 'LSTM', 'Prophet']).round(decimals=3)
695
696
697     # visualize results of last prediction
698     full_graph(last_prediction_ARIMA, "Last prediction ARIMA")
699     full_graph(last_prediction_LSTM, "Last prediction LSTM")
700     full_graph(last_prediction_prophet, "Last prediction Prophet")
```

B.3 Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij univariate, seizoensgebonden tijdreeksen

Listing 25: Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij univariate, seizoensgebonden tijdreeksen

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[40]:
5
6
7  # Voorspellen voor 1 jaar
8  # Of voorspellen op 3 jaar apart omdat voorspellen van 1 jaar misschien de
  → algemene trend minder volgt
9  # SARIMA proberen anders gwn ARIMA
10
11
12 # In[41]:
13
14
15 import pandas as pd
16 import numpy as np
17 # matplotlib is the Python library for drawing diagrams
18 import matplotlib.pyplot as plt
19 get_ipython().run_line_magic('matplotlib', 'inline')
20 # set the size of the diagrams
21 from matplotlib.pyplot import rcParams
22 rcParams['figure.figsize'] = 15,5
23 import timeit
24 import warnings
25 from sklearn.model_selection import TimeSeriesSplit
26
27
28 # # General functions
29
30 # In[42]:
31
32
33 def test_stationarity(timeseries):
34
35     #Determining rolling statistics
36     rolmean = timeseries.rolling(36).mean()
37     rolstd = timeseries.rolling(24).std()
38
39     #Plot rolling statistics:
40     orig = plt.plot(timeseries, color='blue',label='Original')
41     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
42     std = plt.plot(rolstd, color='black', label = 'Rolling Std')
43     plt.legend(loc='best')
44     plt.title('Rolling Mean & Standard Deviation')
```

```

45     plt.show(block=False)
46
47 def full_graph(predicted, og_dataset, title):
48     zerosArray = np.zeros(og_dataset.values.size-len(predicted.flatten()))
49     cleanPrediction =
50         pd.Series(np.concatenate((zerosArray,predicted))).replace(0,np.Nan)
51
52     # plot
53     plt.title(title)
54     plt.plot(og_dataset.index, og_dataset.values,marker='o',
55             color='blue',label='Actual values')
56     plt.plot(og_dataset.index, cleanPrediction,marker='o',
57             color='red',label='Last 2 year prediction')
58     plt.ylim([0,20])
59     plt.legend()
60
61     plt.show()
62
63
64 def revert_diff(predicted_diff, og_data):
65     last_value = og_data.iloc[-predicted_diff.size-1][0]
66     predicted_actual = np.array([])
67     for value_diff in predicted_diff:
68         actual_value = last_value + value_diff
69         predicted_actual = np.append(predicted_actual, actual_value)
70         last_value = actual_value
71     return predicted_actual
72
73
74 def revert_seasonal_diff_recursion(last_seasons_value, diff_value):
75     return last_seasons_value + diff_value
76
77
78 def revert_diff_seasonal(predicted_diff, og_data):
79     prediction_size = predicted_diff.size
80
81     history = ts[:-prediction_size].values.flatten()
82     for value_diff in predicted_diff[-prediction_size:]:
83         new_value = revert_seasonal_diff_recursion(history[-12], value_diff)
84         history = np.append(history,new_value)
85     return history[-prediction_size:]
86
87
88
89
90 # In[43]:
91
92
93 ts
94
95
96 # In[45]:

```

```
97
98
99 ts['date'] = pd.to_datetime(ts['Month'].astype(str) + ts['Year'].astype(str),
  ↵   format='%m%Y', errors='ignore')
100
101
102 # In[46]:
103
104
105 ts
106
107
108 # In[47]:
109
110
111 ts = ts[['date', 'ice_extent']]
112 ts.set_index('date', inplace=True)
113 plt.plot(ts[['ice_extent']])
114
115
116 # In[48]:
117
118
119 test_stationarity(ts)
120
121
122 # ### Differencing
123
124 # In[49]:
125
126
127 ts_diff_seasonal = ts - ts.shift(12)
128 ts_diff_seasonal = ts_diff_seasonal.dropna()
129 test_stationarity(ts_diff_seasonal)
130
131
132 # In[50]:
133
134
135 ts_diff = ts - ts.shift(1)
136 ts_diff = ts_diff.dropna()
137 test_stationarity(ts_diff)
138
139
140 # ### Cross validation setup
141
142 # In[51]:
143
144
145 def display_cross_validation(dataset, n_splits):
146     tscv = TimeSeriesSplit(n_splits = n_splits)
147
148     for train_index, test_index in tscv.split(dataset):
149         if train_index.size > 300:
150             # initialize cross validation train and test sets
```

```

151     cv_train, cv_test = dataset.iloc[train_index],
152         ↳ dataset.iloc[test_index]
153
153     print("TRAIN:", train_index.size) # visualize cross_validation
154         ↳ structure for reference
154     print("TEST:", test_index.size)
155     print()
156
157
158 # In[52]:
159
160
161 ts_diff = ts_diff[:-5] # need the -5 to get testsets for 24 months/2 years
162 display_cross_validation(ts_diff, 18)
163 tscv_diff = TimeSeriesSplit(n_splits = 18)
164
165
166 # In[53]:
167
168
169 display_cross_validation(ts_diff_seasonal, 18)
170 tscv_diff_seasonal = TimeSeriesSplit(n_splits = 18)
171
172
173 # # ARIMA
174
175 # ## random walk differencing
176
177 # ### Determine hyperparameters
178
179 # In[20]:
180
181
182 # ARIMA
183 from statsmodels.tsa.arima_model import ARIMA
184 import itertools
185 import warnings
186 import sys
187 from sklearn.metrics import mean_absolute_error
188
189
190
191 # Define the p, d and q parameters to take any value between 0 and 2
192 p = q = range(0, 5)
193 d = range(0,3)
194
195 # Generate all different combinations of p, q and q triplets
196 pdq = list(itertools.product(p, d, q))
197 best_pdq = pdq
198 best_mean_mae = np.inf
199 warnings.filterwarnings("ignore") # specify to ignore warning messages
200 for param in pdq:
201     print(param)
202     try: # some parametercombinations might lead to crash, so catch exceptions
203         ↳ and continue
203         maes = []

```

```

204     for train_index, test_index in tscv_diff.split(ts_diff):
205         if train_index.size > 300:
206             # initialize cross validation train and test sets
207             cv_train, cv_test = ts_diff.iloc[train_index],
208             ↳ ts_diff.iloc[test_index]
209
210             # build model
211             model = ARIMA(cv_train, order=(param))
212             model_fit = model.fit()
213
214             # make predictions
215             predictions = model_fit.predict(start=len(cv_train),
216             ↳ end=len(cv_train)+cv_test.size-1, dynamic=False)
217             prediction_values = predictions.values
218             true_values = cv_test.values
219             # error calc
220             #     print(true_values)
221             #     print(predictions.values)
222             maes.append(mean_absolute_error(true_values, prediction_values))
223
224
225             mean_mae = np.mean(maes)
226             print('MAE: ' + str(mean_mae))
227
228             if mean_mae < best_mean_mae:
229                 best_mean_mae = mean_mae
230                 best_maes = maes
231                 best_pdq = param
232                 best_predictions = prediction_values
233             except Exception as e:
234                 print(e)
235                 continue
236
237             print()
238             print('Best MAE = ' + str(best_mean_mae))
239             print(best_pdq)
240
241             # best range(0, 10)
242             # Best MAE = 0.23763938034311669
243             # (8, 0, 9)
244             # Wall time: 1h 32min 14s
245
246             # In[54]:
247
248
249             # best_pdq = (8, 0, 9) # range 10
250             best_pdq = (3, 0, 3)
251
252             # ### Final result
253
254             # In[55]:
255
256
257

```

```
258 from statsmodels.tsa.arima_model import ARIMA
259 from sklearn.metrics import mean_absolute_error
260
261
262 start_time = timeit.default_timer()
263
264 warnings.filterwarnings("ignore") # specify to ignore warning messages
265
266 print("-----")
267
268 maes = []
269
270 for train_index, test_index in tscv_diff.split(ts_diff):
271     if train_index.size > 300:
272         # initialize cross validation train and test sets
273         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
274
275         # build model
276         model = ARIMA(cv_train, order=(best_pdq))
277         model_fit = model.fit()
278
279         # make predictions
280         predictions = model_fit.predict(start=len(cv_train),
281                                         end=len(cv_train)+cv_test.size-1, dynamic=False)
282         prediction_values = predictions.values
283         true_values = cv_test.values
284
285         # error calc
286         #     print(true_values)
287         #     print(predictions.values)
288         maes.append(mean_absolute_error(true_values, prediction_values))
289
290         print("I",end="")
291
292 time_ARIMA = timeit.default_timer() - start_time
293 mae_mean = np.mean(maes)
294 MAE_ARIMA = mae_mean
295 last_MAE_ARIMA = maes[-1]
296 last_prediction_ARIMA = prediction_values
297
298 print()
299 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_ARIMA)
300 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_ARIMA)
301 print('Execution time: %.3f seconds' % time_ARIMA)
302
303 reverted_prediction_values = revert_diff(last_prediction_ARIMA, ts[:-5])
304 full_graph(reverted_prediction_values, ts[:-5], 'Last 2 year prediction ARIMA with
305             regular differencing')
306 print(maes)
307
308 # ## Seasonal differencing
309
310 # ### Determine hyperparameters
311 # In[22]:
```

```

312
313
314 # ARIMA
315 from statsmodels.tsa.arima_model import ARIMA
316 import itertools
317 import warnings
318 import sys
319 from sklearn.metrics import mean_absolute_error
320
321
322
323 # Define the p, d and q parameters to take any value between 0 and 2
324 p = q = range(0, 5)
325 d = range(0,3)
326
327 # Generate all different combinations of p, q and q triplets
328 pdq = list(itertools.product(p, d, q))
329 best_pdq = pdq
330 best_mean_mae = np.inf
331 warnings.filterwarnings("ignore") # specify to ignore warning messages
332 for param in pdq:
333     print(param)
334     try: # some parametercombinations might lead to crash, so catch exceptions
335         # and continue
336         maes = []
337         for train_index, test_index in
338             tscv_diff_seasonal.split(ts_diff_seasonal):
339             if train_index.size > 300:
340                 # initialize cross validation train and test sets
341                 cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
342                     ts_diff_seasonal.iloc[test_index]
343
344                 # build model
345                 model = ARIMA(cv_train, order=(param))
346                 model_fit = model.fit()
347
348                 # make predictions
349                 predictions = model_fit.predict(start=len(cv_train),
350                     end=len(cv_train)+cv_test.size-1, dynamic=False)
351                 prediction_values = predictions.values
352                 true_values = cv_test.values
353                 # error calc
354                 #     print(true_values)
355                 #     print(predictions.values)
356                 maes.append(mean_absolute_error(true_values, prediction_values))
357
358                 mean_mae = np.mean(maes)
359                 print('MAE: ' + str(mean_mae))
360
361                 if mean_mae < best_mean_mae:
362                     best_mean_mae = mean_mae
363                     best_maes = maes
364                     best_pdq = param
365                     best_predictions = prediction_values
366             except Exception as e:

```

```

364     print(e)
365     continue
366
367 print()
368 print('Best MAE = ' + str(best_mean_mae))
369 print(best_pdq)
370
371
372 # ### Final result
373
374 # In[56]:
375
376
377
378 best_pdq=(3,0,3) # pq range 4
379 # best_pdq=(8, 0, 6) # pq range 10
380
381
382 # In[57]:
383
384
385 from statsmodels.tsa.arima_model import ARIMA
386 from sklearn.metrics import mean_absolute_error
387
388 start_time = timeit.default_timer()
389
390 warnings.filterwarnings("ignore") # specify to ignore warning messages
391
392 print("-----")
393 maes = []
394
395 for train_index, test_index in tscv_diff_seasonal.split(ts_diff_seasonal):
396     if train_index.size > 300:
397         # initialize cross validation train and test sets
398         cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
399                             ts_diff_seasonal.iloc[test_index]
400
401         # build model
402         model = ARIMA(cv_train, order=(best_pdq))
403         model_fit = model.fit()
404
405         # make predictions
406         predictions = model_fit.predict(start=len(cv_train),
407                                         end=len(cv_train)+cv_test.size-1, dynamic=False)
408         prediction_values = predictions.values
409         true_values = cv_test.values
410
411         # error calc
412         #     print(true_values)
413         #     print(predictions.values)
414         maes.append(mean_absolute_error(true_values, prediction_values))
415
416     print("I",end="")
417

```

```

418 time_ARIMA_seasonal = timeit.default_timer() - start_time
419 mae_mean = np.mean(maes)
420 MAE_ARIMA_seasonal = mae_mean
421 last_MAE_ARIMA_seasonal = maes[-1]
422 last_prediction_ARIMA_seasonal = prediction_values
423
424 print()
425 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_ARIMA_seasonal)
426 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' %
427     ↪ last_MAE_ARIMA_seasonal)
428 print('Execution time: %.3f seconds' % time_ARIMA_seasonal)
429
430 reverted_prediction_values = revert_diff_seasonal(last_prediction_ARIMA_seasonal,
431     ↪ ts)
432 full_graph(reverted_prediction_values, ts, 'Last 2 year prediction ARIMA with
433     ↪ seasonal differencing')
434 print(maes)
435
436 # # SARIMAX
437
438 # ## Random walk differencing
439
440 # In[76]:
441
442
443 # SARIMAX
444
445 import itertools
446 import warnings
447 import sys
448 from statsmodels.tsa.statespace.sarimax import SARIMAX
449
450 # Define the p, d and q parameters to take any value between 0 and 2
451 p = q = P = Q = range(0, 3)
452 d = D = range(0, 2)
453
454 # Generate all different combinations of p, q and q triplets
455 pdqPDQ = list(itertools.product(p, d, q, P, D, Q))
456 best_pdqPDQ = pdqPDQ
457 best_mean_mae = np.inf
458 warnings.filterwarnings("ignore") # specify to ignore warning messages
459 for param in pdqPDQ:
460     print(param)
461     try: # some parametercombinations might lead to crash, so catch exceptions
462         ↪ and continue
463     maes = []
464     for train_index, test_index in tscv_diff.split(ts_diff):
465         if train_index.size > 300:
466             # initialize cross validation train and test sets
467             cv_train, cv_test = ts_diff.iloc[train_index],
468                 ↪ ts_diff.iloc[test_index]
469
470             # build model
471             model = SARIMAX(cv_train,

```

```

469         order=param[:3],
470         seasonal_order=(12,) + param[3:])
471     model_fit = model.fit()
472
473     # make predictions
474     # predictions = model_fit.predict(start=len(cv_train),
475     #                                     end=len(cv_train)+cv_test.size-1, dynamic=False)
476     predictions = model_fit.forecast(steps=24)
477     prediction_values = predictions.values
478     true_values = cv_test.values
479     # error calc
480     #     print(true_values)
481     #     print(predictions.values)
482     maes.append(mean_absolute_error(true_values, prediction_values))
483
484     mean_mae = np.mean(maes)
485     print('MAE: ' + str(mean_mae))
486
487     if mean_mae < best_mean_mae:
488         best_mean_mae = mean_mae
489         best_maes = maes
490         best_pdqPDQ = param
491         best_predictions = prediction_values
492     except Exception as e:
493         print(e)
494         continue
495
496     print(best_predictions.size)
497     print(data_test.index.size)
498
499
500     predictions_df = pd.DataFrame(best_predictions).set_index(keys=data_test.index)
501
502     # plot
503     print()
504     print('Best MAE = ' + str(best_mean_mae))
505     print(best_pdqPDQ)
506     plt.plot(data_test,color='blue')
507     plt.plot(predictions_df, color='red')
508     plt.show()
509
510     # best range(0,2):
511     # Best MAE = 0.2524024604742092
512     # (1, 0, 1, 1, 1, 1)
513     # Wall time: 14min 50s
514
515     # best range(0,2):
516
517     # Best MAE = 0.22780663319275937
518     # (1, 0, 2, 0, 1, 2)
519     # Wall time: 2h 6min 26s
520
521
522     # In[58]:
523

```

```

524
525 best_pdqPDQ = (1, 0, 2, 0, 1, 2)
526
527
528 # In[59]:
529
530
531 from sklearn.metrics import mean_absolute_error
532 from statsmodels.tsa.statespace.sarimax import SARIMAX
533
534
535 start_time = timeit.default_timer()
536
537 warnings.filterwarnings("ignore") # specify to ignore warning messages
538
539 print("-----")
540
541 maes = []
542
543 for train_index, test_index in tscv_diff.split(ts_diff):
544     if train_index.size > 300:
545
546         # initialize cross validation train and test sets
547         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
548
549         # build model
550         model = SARIMAX(cv_train,
551                         order=best_pdqPDQ[:3],
552                         seasonal_order=(best_pdqPDQ[3:]+(12,)))
553         model_fit = model.fit()
554
555         # make predictions
556         predictions = model_fit.predict(start=len(cv_train),
557                                         end=len(cv_train)+cv_test.size-1, dynamic=False)
558         prediction_values = predictions.values
559         true_values = cv_test.values
560
561         # error calc
562         # print(true_values)
563         # print(predictions.values)
564         maes.append(mean_absolute_error(true_values, prediction_values))
565         print("I",end="")
566
567
568 time_SARIMA = timeit.default_timer() - start_time
569 mae_mean = np.mean(maes)
570 MAE_SARIMA = mae_mean
571 last_MAE_SARIMA = maes[-1]
572 last_prediction_SARIMA = prediction_values
573
574
575 print()
576 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_SARIMA)
577 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_SARIMA)
578 print('Execution time: %.3f seconds' % time_SARIMA)

```

```

579
580 reverted_prediction_values = revert_diff(last_prediction_SARIMA, ts[:-5])
581 full_graph(reverted_prediction_values, ts[:-5], 'Last 2 year prediction SARIMAX')
582 print(maes)
583
584 # ## Seasonal differencing
585
586 # ### Determine hyperparameters
587
588 # In[79]:
589
590
591 # SARIMAX
592
593
594 import itertools
595 import warnings
596 import sys
597 from statsmodels.tsa.statespace.sarimax import SARIMAX
598
599
600 # Define the p, d and q parameters to take any value between 0 and 2
601 p = q = P = D = Q = range(0, 3)
602 d = D = range(0, 2)
603
604 # Generate all different combinations of p, q and q triplets
605 pdqPDQ = list(itertools.product(p, d, q, P, D, Q))
606 best_pdqPDQ = pdqPDQ
607 best_mean_mae = np.inf
608 warnings.filterwarnings("ignore") # specify to ignore warning messages
609 for param in pdqPDQ:
610     print(param)
611     try: # some parametercombinations might lead to crash, so catch exceptions
612         # and continue
613         maes = []
614         for train_index, test_index in
615             tscv_diff_seasonal.split(ts_diff_seasonal):
616             if train_index.size > 300:
617                 # initialize cross validation train and test sets
618                 cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
619                 ts_diff_seasonal.iloc[test_index]
620
621                 # build model
622                 model = SARIMAX(cv_train,
623                                 order=param[:3],
624                                 seasonal_order=(12,)+param[3:])
625                 model_fit = model.fit()
626
627                 # make predictions
628                 predictions = model_fit.predict(start=len(cv_train),
629                     end=len(cv_train)+cv_test.size-1, dynamic=False)
630                 predictions = model_fit.forecast(steps=24)
631                 prediction_values = predictions.values
632                 true_values = cv_test.values
633                 # error calc
634                 #     print(true_values)

```

```

631     #     print(predictions.values)
632     maes.append(mean_absolute_error(true_values, prediction_values))
633
634
635     mean_mae = np.mean(maes)
636     print('MAE: ' + str(mean_mae))
637
638     if mean_mae < best_mean_mae:
639         best_mean_mae = mean_mae
640         best_maes = maes
641         best_pdqPDQ = param
642         best_predictions = prediction_values
643     except Exception as e:
644         print(e)
645         continue
646
647 print(best_predictions.size)
648 print(data_test.index.size)
649
650
651 predictions_df = pd.DataFrame(best_predictions).set_index(keys=data_test.index)
652
653 # plot
654 print()
655 print('Best MAE = ' + str(best_mean_mae))
656 print(best_pdqPDQ)
657 plt.plot(data_test,color='blue')
658 plt.plot(predictions_df, color='red')
659 plt.show()
660
661 # best range(0,2):
662 # Best MAE = 0.2524024604742092
663 # (1, 0, 1, 1, 1, 1)
664 # Wall time: 14min 50s
665
666 # best range(0,2):
667
668 # Best MAE = 0.22780663319275937
669 # (1, 0, 2, 0, 1, 2)
670 # Wall time: 2h 6min 26s
671
672
673 # In[60]:
674
675
676 best_pdqPDQ = (1, 0, 2, 0, 1, 2)
677
678
679 # In[61]:
680
681
682 from sklearn.metrics import mean_absolute_error
683 from statsmodels.tsa.statespace.sarimax import SARIMAX
684
685
686 start_time = timeit.default_timer()

```

```

687
688 warnings.filterwarnings("ignore") # specify to ignore warning messages
689
690 print("-----")
691
692 maes = []
693
694 for train_index, test_index in tscv_diff_seasonal.split(ts_diff_seasonal):
695     if train_index.size > 300:
696
697         # initialize cross validation train and test sets
698         cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
699                         ↳ ts_diff_seasonal.iloc[test_index]
700
701         # build model
702         model = SARIMAX(cv_train,
703                         order=best_pdqPDQ[:3],
704                         seasonal_order=(best_pdqPDQ[3:]+(12,)))
705         model_fit = model.fit()
706
707         # make predictions
708         predictions = model_fit.predict(start=len(cv_train),
709                                         ↳ end=len(cv_train)+cv_test.size-1, dynamic=False)
710         prediction_values = predictions.values
711         true_values = cv_test.values
712
713         # error calc
714         #     print(true_values)
715         #     print(predictions.values)
716         maes.append(mean_absolute_error(true_values, prediction_values))
717         print("I",end="")
718
719 time_SARIMA_seasonal = timeit.default_timer() - start_time
720 mae_mean = np.mean(maes)
721 MAE_SARIMA_seasonal = mae_mean
722 last_MAE_SARIMA_seasonal = maes[-1]
723 last_prediction_SARIMA_seasonal = prediction_values
724
725
726 print()
727 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_SARIMA_seasonal)
728 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' %
729       ↳ last_MAE_SARIMA_seasonal)
730 print('Execution time: %.3f seconds' % time_SARIMA_seasonal)
731
732 reverted_prediction_values =
733     ↳ revert_diff_seasonal(last_prediction_SARIMA_seasonal, ts)
734 full_graph(reverted_prediction_values, ts, 'Last 2 year prediction SARIMAX')
735 print(maes)
736
737 # # LSTM
738 # In[62]:

```

```

739
740
741 from keras.layers import Dropout
742 from numpy import array
743 from keras.models import Sequential
744 from keras.layers import LSTM
745 from keras.layers import Dense
746
747
748 # split a univariate sequence into samples
749 def split_sequence(sequence, n_steps_in, n_steps_out):
750     X, y = list(), list()
751     for i in range(len(sequence)):
752         # find the end of this pattern
753         end_ix = i + n_steps_in
754         out_end_ix = end_ix + n_steps_out
755
756         # check if we are beyond the sequence
757         if out_end_ix > len(sequence):
758             break
759
760         # gather input and output parts of the pattern
761         seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
762         X.append(seq_x)
763         y.append(seq_y)
764     return array(X), array(y)
765
766 def build_model(raw_seq, n_steps_in, n_steps_out, n_features, n_neurons, dropout,
767 ← batch_s):
768
769     # split into samples
770     X, y = split_sequence(raw_seq.values.flatten(), n_steps_in, n_steps_out)
771
772     # reshape from [samples, timesteps] into [samples, timesteps, features]
773     X = X.reshape((X.shape[0], X.shape[1], n_features))
774
775     # define model
776     model = Sequential()
777     model.add(LSTM(n_neurons, activation='relu'))
778     model.add(Dropout(dropout))
779     model.add(Dense(n_steps_out))
780     model.compile(optimizer='adam', loss='mae')
781
782     # fit model
783     model.fit(X, y, batch_size=batch_s, epochs=100, verbose=0)
784
785     return model
786
787 def predict(x_input, model, n_features):
788     n_features = 1
789     x_input = x_input.reshape((1, n_steps_in, n_features))
790     yhat = model.predict(x_input, verbose=0)
791     return yhat
792
793

```

```

794 # ## Regular differencing
795
796 # In[32]:
797
798
799 import timeit
800 import tensorflow as tf
801
802
803 start_time = timeit.default_timer()
804
805 # Disabled tf warning because of visual clutter
806 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
807
808
809 # constant variables
810 n_steps_in = 24
811 n_steps_out = 24
812 n_features = 1
813 maes = []
814 global_maes = []
815
816 # optimizable variables
817 n_neurons_array = [1,10,20]
818 dropout_array = [0,0.99]
819 batch_size_array = [1,2,8]
820
821 # n_neurons_array = [1,20]
822 # dropout_array = [0]
823 # batch_size_array = [1,8]
824
825
826
827 # initialize values
828 best_MAE = 100
829 best_n_neurons = 0
830 best_activation = 'none'
831 best_dropout = 0
832 best_batch_size = 0
833
834 for n_neurons in n_neurons_array:
835     for dropout in dropout_array:
836         for batch_size in batch_size_array:
837             print("----")
838             tscv = TimeSeriesSplit(n_splits = 17)
839             for train_index, test_index in tscv_diff.split(ts_diff):
840                 if train_index.size > 300:
841                     # initialize cross validation train and test sets
842                     y_train, y_test = ts_diff.iloc[train_index],
843                                     ts_diff.iloc[test_index]
844
845                     # build model
846                     lstm_model = build_model(y_train, n_steps_in, n_steps_out,
847                                     n_features, n_neurons, dropout, batch_size)
848
849                     # make predictions

```

```

848         x_input = array(y_test)
849         y_predicted = predict(x_input, lstm_model,
850             ↪ n_features).flatten()
851         y_actual = y_test.values
852
853         # error calc
854         maes.append(mean_absolute_error(y_actual, y_predicted))
855
856         print("I",end="")
857
858         # last actual prediction
859         last_prediction_LSTM = y_predicted
860
861         time_LSTM = timeit.default_timer() - start_time
862         MAE_LSTM = np.mean(maes)
863         last_MAE_LSTM = maes[-1]
864         global_maes.append(MAE_LSTM)
865
866         if best_MAE > MAE_LSTM:
867             best_n_neurons = n_neurons
868             best_dropout = dropout
869             best_batch_size = batch_size
870             best_MAE = MAE_LSTM
871
872             print()
873             print(n_neurons)
874             print(dropout)
875             print(batch_size)
876             print(MAE_LSTM)
877             print()
878
879             print('Best:')
880             print('N neurons')
881             print(best_n_neurons)
882             print('Dropout rate')
883             print(best_dropout)
884             print('Batch size')
885             print(best_batch_size)
886             print('MAE')
887             print(best_MAE)
888             plt.bar(range(0,len(global_maes)), global_maes)
889
890         # In[63]:
891
892
893         best_n_neurons, best_dropout, best_batch_size = 1, 0, 1 # actual best params
894
895
896         # In[64]:
897
898
899         import timeit
900         import tensorflow as tf
901
902

```

```

903 start_time = timeit.default_timer()
904
905 # Disabled tf warning because of visual clutter
906 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
907
908
909 # constant variables
910 n_steps_in = 24
911 n_steps_out = 24
912 n_features = 1
913 maes = []
914 global_maes = []
915
916 print("-----")
917 tscv = TimeSeriesSplit(n_splits = 18)
918 for train_index, test_index in tscv.split(ts_diff):
919     if train_index.size > 300:
920         # initialize cross validation train and test sets
921         y_train, y_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
922
923         # build model
924         lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features,
925             ↪ best_n_neurons, best_dropout, best_batch_size)
926
927         # make predictions
928         x_input = array(y_test)
929         y_predicted = predict(x_input, lstm_model, n_features).flatten()
930         y_actual = y_test.values
931
932         # error calc
933         maes.append(mean_absolute_error(y_actual, y_predicted))
934
935         print("I",end="")
936
937 time_LSTM = timeit.default_timer() - start_time
938 MAE_LSTM = np.mean(maes)
939 last_MAE_LSTM = maes[-1]
940 global_maes.append(MAE_LSTM)
941 last_prediction_LSTM = y_predicted
942
943 # print('Best:')
944 # print('N neurons')
945 # print(best_n_neurons)
946 # print('Dropout rate')
947 # print(best_dropout)
948 # print('Batch size')
949 # print(best_batch_size)
950 # print('MAE')
951 # print(best_MAE)
952 # plt.bar(range(0, len(global_maes)), global_maes)
953
954 # time_ARIMA = timeit.default_timer() - start_time
955 # mae_mean = np.mean(maes)
956 # MAE_ARIMA = mae_mean
957 # last_MAE_ARIMA = maes[-1]

```

```

958 print()
959 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
960 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
961 print('Execution time: %.3f seconds' % time_LSTM)
962
963 reverted_prediction_values = revert_diff_seasonal(last_prediction_LSTM, ts)
964 full_graph(reverted_prediction_values, ts, 'Last 2 year prediction LSTM random
  ↪ walk differencing')
965
966 print(maes)
967
968
969 # ## Seasonal differencing
970
971 # In[99]:
972
973
974 # initialize values
975 best_n_neurons = 1
976 best_dropout = 0
977 best_batch_size = 1
978
979
980 # In[100]:
981
982
983 import timeit
984 import tensorflow as tf
985
986
987 start_time = timeit.default_timer()
988
989 # Disabled tf warning because of visual clutter
990 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
991
992
993 # constant variables
994 n_steps_in = 24
995 n_steps_out = 24
996 n_features = 1
997 maes = []
998 global_maes = []
999
1000 # optimizable variables
1001 n_neurons_array = [1,5,10,20]
1002 dropout_array = [0,0.5,0.99]
1003 batch_size_array = [1,2,4,8]
1004 ## set#2
1005 # n_neurons_array = [1,20]2
1006 # dropout_array = [0]
1007 # batch_size_array = [1,8]
1008
1009
1010
1011 # initialize values
1012 best_MAE = 100

```

```

1013 best_n_neurons = 0
1014 best_activation = 'none'
1015 best_dropout = 0
1016 best_batch_size = 0
1017
1018 for n_neurons in n_neurons_array:
1019     for dropout in dropout_array:
1020         for batch_size in batch_size_array:
1021             print("-----")
1022             # tscv = TimeSeriesSplit(n_splits = 17)
1023             for train_index, test_index in tscv_diff.split(ts_diff_seasonal):
1024                 if train_index.size > 300:
1025                     # initialize cross validation train and test sets
1026                     y_train, y_test = ts_diff_seasonal.iloc[train_index],
1027                                     ↪ ts_diff_seasonal.iloc[test_index]
1028
1029                     # build model
1030                     lstm_model = build_model(y_train, n_steps_in, n_steps_out,
1031                                     ↪ n_features, n_neurons, dropout, batch_size)
1032
1033                     # make predictions
1034                     x_input = array(y_test)
1035                     y_predicted = predict(x_input, lstm_model,
1036                                     ↪ n_features).flatten()
1037                     y_actual = y_test.values
1038
1039                     # error calc
1040                     maes.append(mean_absolute_error(y_actual, y_predicted))
1041
1042                     print("I",end="")
1043
1044                     # last actual prediction
1045                     last_prediction_LSTM = y_predicted
1046
1047                     time_LSTM = timeit.default_timer() - start_time
1048                     MAE_LSTM = np.mean(maes)
1049                     last_MAE_LSTM = maes[-1]
1050                     global_maes.append(MAE_LSTM)
1051
1052                     if best_MAE > MAE_LSTM:
1053                         best_n_neurons = n_neurons
1054                         best_dropout = dropout
1055                         best_batch_size = batch_size
1056                         best_MAE = MAE_LSTM
1057
1058                         print()
1059                         print(n_neurons)
1060                         print(dropout)
1061                         print(batch_size)
1062                         print(MAE_LSTM)
1063                         print()
1064
1065             print('Best:')
1066             print('N neurons')
1067             print(best_n_neurons)
1068             print('Dropout rate')

```

```

1066 print(best_dropout)
1067 print('Batch size')
1068 print(best_batch_size)
1069 print('MAE')
1070 print(best_MAE)
1071 plt.bar(range(0,len(global_maes)), global_maes)
1072
1073
1074 # In[65]:
1075
1076
1077 best_n_neurons, best_dropout, best_batch_size = 1, 0.99, 8 # ran for 5 hours
1078
1079
1080 # In[66]:
1081
1082 import timeit
1083 import tensorflow as tf
1084
1085
1086 start_time = timeit.default_timer()
1087
1088 # Disabled tf warning because of visual clutter
1089 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
1090
1091
1092
1093 # constant variables
1094 n_steps_in = 24
1095 n_steps_out = 24
1096 n_features = 1
1097 maes = []
1098 global_maes = []
1099
1100 print("-----")
1101 tscv = TimeSeriesSplit(n_splits = 18)
1102 for train_index, test_index in tscv.split(ts_diff_seasonal):
1103     if train_index.size > 300:
1104         # initialize cross validation train and test sets
1105         y_train, y_test = ts_diff_seasonal.iloc[train_index],
1106                         ts_diff_seasonal.iloc[test_index]
1107
1108         # build model
1109         lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features,
1110                                   best_n_neurons, best_dropout, best_batch_size)
1111
1112         # make predictions
1113         x_input = array(y_test)
1114         y_predicted = predict(x_input, lstm_model, n_features).flatten()
1115         y_actual = y_test.values
1116
1117         # error calc
1118         maes.append(mean_absolute_error(y_actual, y_predicted))
1119
1120         print("I",end="")

```

```

1120
1121
1122 time_LSTM_seasonal = timeit.default_timer() - start_time
1123 MAE_LSTM_seasonal = np.mean(maes)
1124 last_MAE_LSTM_seasonal = maes[-1]
1125 global_maes.append(MAE_LSTM_seasonal)
1126 last_prediction_LSTM_seasonal = y_predicted
1127 # print('Best:')
1128 # print('N neurons')
1129 # print(best_n_neurons)
1130 # print('Dropout rate')
1131 # print(best_dropout)
1132 # print('Batch size')
1133 # print(best_batch_size)
1134 # print('MAE')
1135 # print(best_MAE)
1136 # plt.bar(range(0, len(global_maes)), global_maes)
1137
1138 # time_ARIMA = timeit.default_timer() - start_time
1139 # mae_mean = np.mean(maes)
1140 # MAE_ARIMA = mae_mean
1141 # last_MAE_ARIMA = maes[-1]
1142
1143 print()
1144 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM_seasonal)
1145 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' %
    ↪ last_MAE_LSTM_seasonal)
1146 print('Execution time: %.3f seconds' % time_LSTM_seasonal)
1147
1148 reverted_prediction_values = revert_diff_seasonal(last_prediction_LSTM_seasonal,
    ↪ ts)
1149 full_graph(reverted_prediction_values, ts, 'Last 2 year prediction LSTM Seasonal')
1150
1151 print(maes)
1152
1153
1154 # # Prophet
1155
1156 # In[67]:
1157
1158
1159 ts_diff.reset_index()
1160
1161
1162 # In[68]:
1163
1164
1165 # formatting dataframe
1166 ts_formated_prophet = ts_diff.reset_index().rename(columns = {'date' : 'ds',
    ↪ 'ice_extent' : 'y'})
1167 ts_formated_prophet['ds'] =
    ↪ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds'].astype(str),
        ↪ format='%Y-%m-%d'))
1168
1169
1170 # In[69]:

```

```
1171
1172
1173 # initialize TimeSeriesSplit object
1174 tscv_prophet = TimeSeriesSplit(n_splits = 18)
1175
1176 # loop trough all split time series that have a trainingsset with more than 20
1177 # values
1178 for train_index, test_index in tscv_prophet.split(ts_formated_prophet):
1179     if train_index.size > 300:
1180
1181         # initialize cross validation train and test sets
1182         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
1183
1184         # visualize cross_validation structure for reference
1185         print("TRAIN:", train_index.size)
1186         print("TEST:", test_index.size)
1187         print()
1188
1189 # In[39]:
1190
1191
1192 # Python
1193 import itertools
1194 import numpy as np
1195 import pandas as pd
1196 from fbprophet import Prophet
1197 from sklearn.metrics import mean_absolute_error
1198
1199
1200 # define dataframe
1201 df = ts_formated_prophet
1202
1203 param_grid = {
1204     'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5, 0.75, 1],
1205     'seasonality_prior_scale': [0.001, 0.01, 0.1, 1, 2, 5, 10],
1206 }
1207
1208 # Generate all combinations of parameters
1209 all_params = [dict(zip(param_grid.keys(), v)) for v in
1210     itertools.product(*param_grid.values())]
1211
1212 # initialize variables
1213 maes = []
1214 global_maes = []
1215 best_MAE_prophet = np.inf
1216
1217 # Use cross validation to evaluate all parameters
1218 for params in all_params:
1219
1220     # loop trough all split time series that have a trainingsset with more than
1221     # 20 values
1222     for train_index, test_index in tscv_prophet.split(ts_formated_prophet):
1223         if train_index.size > 300:
1224
1225             # initialize cross validation train and test sets
```

```

1224     train = ts_formated_prophet.iloc[train_index]
1225     y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
1226     X_test = ts_formated_prophet.iloc[test_index][['ds']]
1227
1228     # Fit model with given params
1229     model = Prophet(**params, weekly_seasonality=False,
1230                      daily_seasonality=True)
1231     model = model.fit(train)
1232
1233     # make predictions
1234     forecast = model.predict(X_test)
1235     y_pred = forecast['yhat'].values
1236
1237     # last actual prediction
1238     last_prediction_prophet = y_pred
1239
1240     # error calculation this part of the cross validation
1241     maes.append(mean_absolute_error(y_test, y_pred))
1242
1243     # error calculation for this parameter combination
1244     MAE_prophet = np.mean(maes)
1245     last_MAE_prophet = maes[-1]
1246     global_maes.append(MAE_prophet)
1247
1248     # logging
1249     print('changepoint_prior_scale: ' + str(params['changepoint_prior_scale']))
1250     print('seasonality_prior_scale: ' + str(params['seasonality_prior_scale']))
1251     print(MAE_prophet)
1252
1253     # store parameters resulting in the lowest mean MAE
1254     if best_MAE_prophet > MAE_prophet:
1255         best_params = params
1256         best_MAE_prophet = MAE_prophet
1257
1258     # log optimal result
1259     print('changepoint_prior_scale: ' + str(best_params['changepoint_prior_scale']))
1260     print('seasonality_prior_scale: ' + str(best_params['seasonality_prior_scale']))
1261     print(best_MAE_prophet)
1262
1263
1264 # In[77]:
1265
1266 best_params = {'changepoint_prior_scale': 0.001, 'seasonality_prior_scale': 10}
1267
1268
1269 # In[78]:
1270
1271
1272 from fbprophet import Prophet
1273 # Disabled tf warning because of clutter
1274 warnings.filterwarnings("ignore") # specify to ignore warning messages
1275
1276 start_time = timeit.default_timer()
1277

```

```

1279 # initialize variables
1280 maes = []
1281
1282 for train_index, test_index in tscv_prophet.split(ts_formated_prophet):
1283     if train_index.size > 300:
1284
1285         # initialize cross validation train and test sets
1286         train = ts_formated_prophet.iloc[train_index]
1287         y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
1288         X_test = ts_formated_prophet.iloc[test_index][['ds']]
1289
1290         # build model
1291         model = Prophet(**best_params, weekly_seasonality=False,
1292                         daily_seasonality=False)
1293         model.fit(train)
1294
1295         # make predictions
1296         forecast = model.predict(X_test)
1297         y_pred = forecast['yhat'].values
1298
1299         # error calc
1300         maes.append(mean_absolute_error(y_test, y_pred))
1301
1302
1303
1304 # store results
1305 time_Prophet = timeit.default_timer() - start_time
1306 MAE_Prophet = np.mean(maes)
1307 last_MAE_Prophet = maes[-1]
1308 last_prediction_prophet = y_pred
1309
1310 # visualize results
1311 print()
1312 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_Prophet)
1313 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_Prophet)
1314 print('Execution time: %.3f seconds' % time_Prophet)
1315
1316 reverted_prediction_values = revert_diff_seasonal(last_prediction_prophet, ts)
1317 full_graph(reverted_prediction_values, ts, "Last 2 year prediction prophet")
1318 print('Mean average errors')
1319 print(maes)
1320
1321
1322 # In[ ]:
1323
1324
1325 Mean MAE: 0.250 x 1 000 000 km2
1326
1327
1328 # # Evaluation
1329
1330 # In[79]:
1331
1332
1333 # formatting

```

```

1334 results = [[MAE_ARIMA, time_ARIMA, last_MAE_ARIMA],
1335             [MAE_ARIMA_seasonal, time_ARIMA_seasonal, last_MAE_ARIMA_seasonal],
1336             [MAE_SARIMA, time_SARIMA, last_MAE_SARIMA],
1337             [MAE_SARIMA_seasonal, time_SARIMA_seasonal, last_MAE_SARIMA_seasonal],
1338             [MAE_LSTM, time_LSTM, last_MAE_LSTM],
1339             [MAE_LSTM_seasonal, time_LSTM_seasonal, last_MAE_LSTM_seasonal],
1340             [MAE_Prophet, time_Prophet, last_MAE_Prophet]]
1341
1342 # display results
1343 results = pd.DataFrame(results, columns=['Mean MAE (x 1 000 000
1344     ↵ km\u00b2)', 'Execution time (s)', 'Last MAE (x 1 000 000 km\u00b2)']
1345
1346
1347
1348 # In[84]:
1349
1350
1351 reverted_prediction_values = revert_diff(last_prediction_ARIMA, ts[:-5])
1352 full_graph(reverted_prediction_values, ts[:-5], 'Last prediction ARIMA')
1353
1354 reverted_prediction_values = revert_diff_seasonal(last_prediction_ARIMA_seasonal,
1355     ↵ ts)
1356 full_graph(reverted_prediction_values, ts, 'Last prediction ARIMA with seasonal
1357     ↵ differencing')
1358
1359 reverted_prediction_values = revert_diff(last_prediction_SARIMA, ts[:-5])
1360 full_graph(reverted_prediction_values, ts[:-5], 'Last prediction SARIMAX')
1361
1362
1363
1364 # In[83]:
1365
1366
1367
1368 reverted_prediction_values = revert_diff_seasonal(last_prediction_LSTM, ts)
1369 full_graph(reverted_prediction_values, ts, 'Last prediction LSTM')
1370
1371 reverted_prediction_values = revert_diff_seasonal(last_prediction_LSTM_seasonal,
1372     ↵ ts)
1373 full_graph(reverted_prediction_values, ts, 'Last prediction LSTM Seasonal')
1374
1375 reverted_prediction_values = revert_diff_seasonal(last_prediction_prophet, ts)
1376 full_graph(reverted_prediction_values, ts, "Last 2 year prediction Prophet")

```

B.4 Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, niet-seizoensgebonden tijdreeksen

Listing 26: Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, niet-seizoensgebonden tijdreeksen

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # Imports
5
6  # In[1]:
7
8
9  import numpy as np
10 import pandas as pd
11 # matplotlib is the Python library for drawing diagrams
12 import matplotlib.pyplot as plt
13 get_ipython().run_line_magic('matplotlib', 'inline')
14 # set the size of the diagrams
15 from matplotlib.pyplot import rcParams
16 rcParams['figure.figsize'] = 15,5
17 from math import sqrt
18 from sklearn.metrics import mean_squared_error
19 from sklearn.model_selection import TimeSeriesSplit
20
21
22 # In[2]:
23
24
25 def test_stationarity(timeseries, title):
26
27     #Determining rolling statistics
28     rolmean = timeseries.rolling(12).mean()
29     #      rolstd = timeseries.rolling(24).std()
30
31     #Plot rolling statistics:
32     orig = plt.plot(timeseries, color='blue',label='Original')
33     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
34     #      std = plt.plot(rolstd, color='black', label = 'Rolling Std')
35     plt.legend(loc='best')
36     plt.title('Rolling Mean & Standard Deviation from column: ' + title)
37     plt.show(block=False)
38
39 def full_graph(predicted_diff, title):
40     predictionsArray = np.asarray(revert_diff(predicted_diff, ts_ie))
41     zerosArray = np.zeros(ts_ie.values.size-len(predictionsArray.flatten()))
42     cleanPrediction =
43         pd.Series(np.concatenate((zerosArray,predictionsArray))).replace(0,np.NaN)
44
45     # plot
```

```

45     plt.title(title)
46     plt.plot(ts_ie.values,marker='o', color='blue',label='Actual values')
47     plt.plot(cleanPrediction,marker='o', color='red',label='Last 4 year
48     ↪ prediction')
49     plt.ylim([0,15])
50     plt.legend()
51
52     plt.show()
53
53 def revert_diff(predicted_diff, og_data):
54     last_value = og_data.iloc[-predicted_diff.size-1][0]
55     predicted_actual = np.array([])
56     for value_diff in predicted_diff:
57         actual_value = last_value + value_diff
58         predicted_actual = np.append(predicted_actual, actual_value)
59         last_value = actual_value
60     return predicted_actual
61
62
63 # ## Dataprep
64
65 # In[3]:
66
67
68 ts = pd.read_csv('./data/dataframe_yearly.csv', index_col=0).reset_index()
69 ts.rename(columns={'Year':'year'}, inplace=True)
70 ts.set_index('year', inplace=True)
71
72
73 # In[4]:
74
75
76 ts_ie = ts[['ice_extent']]
77
78
79 # In[5]:
80
81
82 ts
83
84
85 # In[6]:
86
87
88 plt.title('Temperature')
89 plt.plot(ts.iloc[:,0], label='temperature')
90 plt.ylabel('°C')
91 plt.show()
92 plt.title('Ice extent')
93 plt.ylabel('x 1 000 000 km\u00b2')
94 plt.plot(ts.iloc[:,1], label='ice extent')
95
96
97 # In[7]:
98
99

```

```

100 test_stationarity(ts[['ice_extent']], 'ice_extent')
101
102
103 # ### Differencing
104
105 # In[8]:
106
107
108 ts_diff = ts - ts.shift(1)
109 ts_diff = ts_diff.dropna()
110 test_stationarity(ts_diff[['ice_extent']], 'ice_extent')
111 test_stationarity(ts_diff[['mean_temp']], 'mean_temp')
112
113
114 # Datasets are stationary now
115
116 # # Cross validation setup
117
118 # In[9]:
119
120
121 tscv = TimeSeriesSplit(n_splits = 8)
122 dataset = ts_diff
123
124 for train_index, test_index in tscv.split(dataset):
125     if train_index.size > 20:
126
127         # initialize cross validation train and test sets
128         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
129
130         print("TRAIN:", train_index.size) # visualize cross_validation structure
131             # for reference
132         print("TEST:", test_index.size)
133         print()
134
135 # # VARMAX
136
137 # In[10]:
138
139
140 from statsmodels.tsa.statespace.varmax import VARMAX
141 import itertools
142 import warnings
143 import sys
144 from sklearn.metrics import mean_absolute_error
145
146
147
148 # Define the p, d and q parameters to take any value between 0 and 2
149 p = q = range(0, 5)
150
151 # Generate all different combinations of p, q and q triplets
152 pq = list(itertools.product(p, q))
153 best_pq = pq
154 best_mean_mae = np.inf

```

```

155 warnings.filterwarnings("ignore") # specify to ignore warning messages
156 for param in pq:
157     print(param)
158     try: # some parametercombinations might lead to crash, so catch exceptions
159         # and continue
160         maes = []
161         for train_index, test_index in tscv.split(dataset):
162             if train_index.size > 20:
163                 # initialize cross validation train and test sets
164                 cv_train, cv_test = dataset.iloc[train_index],
165                 # dataset.iloc[test_index]
166
167                 # build model
168                 model = VARMAX(cv_train, order=(param))
169                 model_fit = model.fit()
170
171                 # make predictions
172                 predictions = model_fit.forecast(steps=4, dynamic=False)
173                 prediction_values = predictions[['ice_extent']].values
174                 true_values = cv_test[['ice_extent']].values
175                 # error calc
176                 maes.append(mean_absolute_error(true_values, prediction_values))
177
178                 mean_mae = np.mean(maes)
179                 print('MAE: ' + str(mean_mae))
180
181             if mean_mae < best_mean_mae:
182                 best_mean_mae = mean_mae
183                 best_maes = maes
184                 best_pq = param
185                 best_predictions = prediction_values
186             except Exception as e:
187                 print(e)
188                 continue
189
190             # plot
191             print()
192             print('Best MAE = ' + str(best_mean_mae))
193             print(best_pq)
194
195             # # best range(0,5)
196             # Best MAE = 0.15358394799986985
197             # (3, 3)
198             # Wall time: 9min 27s
199
200             # In[57]:
201
202             best_pq = (3,3)
203
204
205             # In[58]:
206
207
208

```

```

209 from statsmodels.tsa.statespace.varmax import VARMAX
210 from sklearn.metrics import mean_absolute_error
211 import timeit
212 import warnings
213
214
215 start_time = timeit.default_timer()
216
217 warnings.filterwarnings("ignore") # specify to ignore warning messages
218
219 print("----")
220
221 maes = []
222
223 for train_index, test_index in tscv.split(dataset):
224     if train_index.size > 20:
225         # initialize cross validation train and test sets
226         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
227
228         # build model
229         model = VARMAX(cv_train, order=(best_pq))
230         model_fit = model.fit()
231
232         # make predictions
233         predictions = model_fit.forecast(steps=4, dynamic=False)
234         prediction_values = predictions[['ice_extent']].values
235         true_values = cv_test[['ice_extent']].values
236         # error calc
237         maes.append(mean_absolute_error(true_values, prediction_values))
238
239         print("I",end="")
240
241
242 time_VARMAX = timeit.default_timer() - start_time
243 mae_mean = np.mean(maes)
244 MAE_VARMAX = mae_mean
245 last_MAE_VARMAX = maes[-1]
246 last_predictions_VARMAX = prediction_values
247
248 print()
249 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_VARMAX)
250 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_VARMAX)
251 print('Execution time: %.3f seconds' % time_VARMAX)
252 full_graph(last_predictions_VARMAX, 'Last prediction VARMAX')
253 print(maes)
254
255
256 # # LSTM
257
258 # In[39]:
259
260
261 # multivariate multi-step encoder-decoder lstm example
262 from numpy import array
263 from numpy import hstack
264 from keras.models import Sequential

```

```

265 from keras.layers import LSTM
266 from keras.layers import Dense
267 from keras.layers import RepeatVector
268 from keras.layers import TimeDistributed
269 import warnings
270
271 warnings.filterwarnings("ignore") # specify to ignore warning messages
272
273
274 # split a multivariate sequence into samples
275 def split_sequences(sequences, n_steps_in, n_steps_out):
276     X, y = list(), list()
277     for i in range(len(sequences)):
278         # find the end of this pattern
279         end_ix = i + n_steps_in
280         out_end_ix = end_ix + n_steps_out
281         # check if we are beyond the dataset
282         if out_end_ix > len(sequences):
283             break
284         # gather input and output parts of the pattern
285         seq_x, seq_y = sequences[i:end_ix, :],
286         → sequences[end_ix:out_end_ix, :]
287         X.append(seq_x)
288         y.append(seq_y)
289     return array(X), array(y)
290
291 def predict_LSTM(train, test, n_neurons, n_epochs):
292     test['sum'] = test['mean_temp'] + test['ice_extent']
293
294     # define input sequence
295     in_seq1 = train.values[:,0]
296     in_seq2 = train.values[:,1]
297     out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
298
299     # convert to [rows, columns] structure
300     in_seq1 = in_seq1.reshape((len(in_seq1), 1))
301     in_seq2 = in_seq2.reshape((len(in_seq2), 1))
302     out_seq = out_seq.reshape((len(out_seq), 1))
303
304     # horizontally stack columns
305     dataset = hstack((in_seq1, in_seq2, out_seq))
306
307     # choose a number of time steps
308     n_steps_in, n_steps_out = 4, 4
309
310     # covert into input/output
311     X, y = split_sequences(dataset, n_steps_in, n_steps_out)
312
313     # the dataset knows the number of features, e.g. 2
314     n_features = X.shape[2]
315
316     # define model
317     model = Sequential()
318     model.add(LSTM(n_neurons, activation='relu', input_shape=(n_steps_in,
→ n_features)))

```

```

319     model.add(RepeatVector(n_steps_out))
320     model.add(LSTM(n_neurons, activation='relu', return_sequences=True))
321     model.add(TimeDistributed(Dense(n_features)))
322     model.compile(optimizer='adam', loss='mae')
323
324     # fit model
325     model.fit(X, y, epochs=n_epochs, verbose=0)
326
327     # demonstrate prediction
328     x_input = test.values
329     x_input = x_input.reshape((1, n_steps_in, n_features))
330     yhat = model.predict(x_input, verbose=0)
331     return yhat
332
333
334
335 # In[109]:
336
337
338 from statsmodels.tsa.statespace.varmax import VARMAX
339 from sklearn.metrics import mean_absolute_error
340 import timeit
341 import tensorflow as tf
342
343 start_time = timeit.default_timer()
344
345 # warnings.filterwarnings("ignore") # specify to ignore warning messages
346 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
347
348 maes = []
349 global_maes = []
350 best_MAE = np.inf
351
352
353 n_neurons_array = [1,5,10,20]
354 n_epochs_array = [100,200,300]
355
356 print("----")
357
358 maes = []
359 for n_neurons in n_neurons_array:
360     for n_epochs in n_epochs_array:
361         for train_index, test_index in tscv.split(dataset):
362             if train_index.size > 20:
363                 # initialize cross validation train and test sets
364                 cv_train, cv_test = dataset.iloc[train_index],
365                               dataset.iloc[test_index]
366
367                 yhat = predict_LSTM(cv_train, cv_test, n_neurons, n_epochs)
368
369                 prediction_values = yhat[0][:,1]
370                 true_values = cv_test[['ice_extent']].values
371
372                 # error calc
373                 maes.append(mean_absolute_error(true_values, prediction_values))

```

```
374             print("I",end="")
375             time_LSTM = timeit.default_timer() - start_time
376             MAE_LSTM = np.mean(maes)
377             last_MAE_LSTM = maes[-1]
378             global_maes.append(MAE_LSTM)
379
380             if best_MAE > MAE_LSTM:
381                 best_n_neurons = n_neurons
382                 best_n_epochs = n_epochs
383                 best_MAE = MAE_LSTM
384
385             print()
386             print(n_neurons)
387             print(n_epochs)
388             print(MAE_LSTM)
389             print()
390
391             print('Best:')
392             print('N neurons')
393             print(best_n_neurons)
394             print('Epochs size')
395             print(best_n_epochs)
396             print('MAE')
397             print(best_MAE)
398
399
400 # In[41]:
401
402
403
404 best_n_neurons, best_n_epochs = 1, 200
405
406
407 # In[42]:
408
409
410 from sklearn.metrics import mean_absolute_error
411 import timeit
412 import tensorflow as tf
413
414 start_time = timeit.default_timer()
415
416 # warnings.filterwarnings("ignore") # specify to ignore warning messages
417 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
418
419
420 print("----")
421
422 maes = []
423
424 for train_index, test_index in tscv.split(ts_diff):
425     if train_index.size > 20:
426
427         # initialize cross validation train and test sets
428         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
```

```

430     yhat = predict_LSTM(cv_train, cv_test, best_n_neurons, best_n_epochs)
431
432
433     prediction_values = yhat[0][:,1]
434     true_values = cv_test[['ice_extent']].values
435
436     # error calc
437     maes.append(mean_absolute_error(true_values, prediction_values))
438
439     print("I",end="")
440
441
442     time_LSTM = timeit.default_timer() - start_time
443     mae_mean = np.mean(maes)
444     MAE_LSTM = mae_mean
445     last_MAE_LSTM = maes[-1]
446     last_predictions_LSTM = prediction_values
447
448     print()
449     print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
450     print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
451     print('Execution time: %.3f seconds' % time_LSTM)
452     full_graph(last_predictions_LSTM, 'Last prediction LSTM')
453     print(maes)
454
455
456     # # Prophet
457
458     # In[ ]:
459
460
461     # formatting dataframe
462     ts_formated_prophet = ts_diff.reset_index().rename(columns = {'year' : 'ds',
463     ↪ 'ice_extent' : 'y'})
463     ts_formated_prophet['ds'] =
464     ↪ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds'].astype(str),
465     ↪ format='%Y'))
466
467
468     # In[ ]:
469
470     from fbprophet import Prophet
471     m = Prophet()
472     m.add_regressor('mean_temp')
473     m.fit(df_train)
474
475     # In[ ]:
476
477
478     # Python
479     import itertools
480     import numpy as np
481     import pandas as pd
482

```

```

483 warnings.filterwarnings("ignore") # specify to ignore warning messages
484
485 # define dataframe
486 df = ts_formated_prophet
487
488 param_grid = {
489     'changepoint_prior_scale': [0.001, 0.01, 0.1, 1, 2, 5, 10, 15, 20, 25],
490 }
491
492 # Generate all combinations of parameters
493 all_params = [dict(zip(param_grid.keys(), v)) for v in
494     itertools.product(*param_grid.values())]
495
496 # initialize variables
497 maes = []
498 global_maes = []
499 best_MAE_prophet = np.inf
500
501 # Use cross validation to evaluate all parameters
502 for params in all_params:
503
504     # loop trough all split time series that have a trainingsset with more than
505     # 20 values
506     for train_index, test_index in tscv.split(ts_formated_prophet):
507         if train_index.size > 20:
508
509             # initialize cross validation train and test sets
510             train = ts_formated_prophet.iloc[train_index]
511             y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
512             X_test = ts_formated_prophet.iloc[test_index][['ds', 'mean_temp']]
513
514             # Fit model with given params
515             model = Prophet(weekly_seasonality=False, daily_seasonality=False)
516             model.add_regressor('mean_temp')
517             model.fit(train)
518
519             # make predictions
520             forecast = model.predict(X_test)
521             y_pred = forecast['yhat'].values
522
523             # last actual prediction
524             last_prediction_prophet = y_pred
525
526             # error calculation this part of the cross validation
527             maes.append(mean_absolute_error(y_test, y_pred))
528
529             # error calculation for this parameter combination
530             MAE_prophet = np.mean(maes)
531             last_MAE_prophet = maes[-1]
532             global_maes.append(MAE_prophet)
533
534             # logging
535             print('changepoint_prior_scale: ' + str(params['changepoint_prior_scale']))
536
537             # store parameters resulting in the lowest mean MAE
538             if best_MAE_prophet > MAE_prophet:

```

```

537     best_params = params
538     best_MAE_prophet = MAE_prophet
539
540 # log optimal result
541 print('changepoint_prior_scale: ' + str(best_params['changepoint_prior_scale']))
542 print(best_MAE_prophet)
543
544
545 # In[ ]:
546
547
548
549
550
551 # In[57]:
552
553
554 maes = []
555 global_maes = []
556
557 from sklearn.metrics import mean_absolute_error
558
559 # loop trough all split time series that have a trainingsset with more than 20
560 # values
561 for train_index, test_index in tscv.split(ts_formated_prophet):
562     if train_index.size > 20:
563
564         # initialize cross validation train and test sets
565         train = ts_formated_prophet.iloc[train_index]
566         y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
567         X_test = ts_formated_prophet.iloc[test_index][['ds', 'mean_temp']]
568
569         # Fit model with given params
570         model = Prophet(weekly_seasonality=False, daily_seasonality=False)
571         model.add_regressor('mean_temp')
572         model.fit(train)
573
574         # make predictions
575         forecast = model.predict(X_test)
576         y_pred = forecast['yhat'].values
577
578         # last actual prediction
579         last_prediction_prophet = y_pred
580
581         # error calculation this part of the cross validation
582         maes.append(mean_absolute_error(y_test, y_pred))
583
584         # error calculation for this parameter combination
585         MAE_prophet = np.mean(maes)
586         last_MAE_prophet = maes[-1]
587         global_maes.append(MAE_prophet)
588         print(np.mean(maes))
589
590 # In[34]:
591

```

```

592
593 # formatting dataframe
594 ts_formated_prophet = ts_diff.reset_index().rename(columns = {'year' : 'ds',
595   ↪ 'ice_extent' : 'y'})
595 ts_formated_prophet['ds'] =
596   ↪ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds']).astype(str),
597     ↪ format='%Y'))
597
598 # In[39]:
599
600
601 df_train = ts_formated_prophet.iloc[:-4]
602 df_test = ts_formated_prophet.iloc[-4:]
603
604
605 # In[40]:
606
607
608 df_train.head()
609
610
611 # In[41]:
612
613
614 from fbprophet import Prophet
615 m = Prophet()
616 m.add_regressor('mean_temp')
617 m.fit(df_train)
618
619
620 # In[43]:
621
622
623 forecast = m.predict(df_test.drop(columns="y"))
624 forecast.set_index('ds')[['yhat']]
625
626
627 # In[ ]:
628
629
630 # example
631
632
633 # In[2]:
634
635
636 import pandas as pd
637 df = pd.DataFrame(pd.date_range(start="2019-09-01", end="2019-09-30", freq='D',
638   ↪ name='ds'))
638 df["y"] = range(1,31)
639 df["add1"] = range(101,131)
640 df["add2"] = range(201,231)
641 df.head()
642
643

```

```

644 # In[3]:
645
646 df_train = df.loc[df["ds"]<"2019-09-21"]
647 df_test = df.loc[df["ds"]>="2019-09-21"]
648
649
650
651 # In[4]:
652
653
654 from fbprophet import Prophet
655 m = Prophet()
656 m.add_regressor('add1')
657 m.add_regressor('add2')
658 m.fit(df_train)
659
660
661 # In[10]:
662
663
664 forecast = m.predict(df_test.drop(columns="y"))
665 forecast.set_index('ds')[['yhat']]
666
667
668 # ### Evaluation
669
670 # In[112]:
671
672
673 # formatting
674 results = [[MAE_VARMAX, time_VARMAX, last_MAE_VARMAX],
675 [MAE_LSTM, time_LSTM, last_MAE_LSTM]]
676
677 # display results
678 results = pd.DataFrame(results, columns=['Mean MAE (x 1 000 000
679 → km\u00b2)', 'Execution time (s)', 'Last MAE (x 1 000 000 km\u00b2)']
680 ,index=['VARMAX', 'LSTM']).round(decimals=3)
681 results
682
683 # In[65]:
684
685
686 full_graph(last_predictions_VARMAX, 'Last prediction VARMAX')
687 full_graph(last_predictions_LSTM, 'Last prediction LSTM')

```

B.5 Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, seizoensgebonden tijdreeksen

Listing 27: Broncode voor het vergelijken van ARIMA, LSTM en Prophet bij multivariate, seizoensgebonden tijdreeksen

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # In[54]:
5
6
7 import pandas as pd
8 import numpy as np
9 # matplotlib is the Python library for drawing diagrams
10 import matplotlib.pyplot as plt
11 get_ipython().run_line_magic('matplotlib', 'inline')
12 # set the size of the diagrams
13 from matplotlib.pyplot import rcParams
14 rcParams['figure.figsize'] = 15,5
15 from sklearn.model_selection import TimeSeriesSplit
16
17
18 # # General functions
19
20 # In[100]:
21
22
23 reverted_prediction_values
24
25
26 # In[104]:
27
28
29 ts[['ice_extent']].max()
30
31
32 # In[119]:
33
34
35
36
37
38 # In[112]:
39
40
41 plt.plot(last_predictions_VARMAX_seasonal)
42
43
44 # In[162]:
45
46
47
48 # reverted_prediction_values =
49 #     revert_diff_seasonal(last_predictions_VARMAX_seasonal, ts[['ice_extent']])
50 full_graph_seasonal(last_predictions_VARMAX_seasonal, 'Last 2 year prediction
51 #     ARIMA with seasonal differencing')
```

```

52 # In[171]:
53
54
55 def test_stationarity(timeseries, title):
56
57     #Determining rolling statistics
58     rolmean = timeseries.rolling(12).mean()
59     #     rolstd = timeseries.rolling(24).std()
60
61     #Plot rolling statistics:
62     orig = plt.plot(timeseries, color='blue',label='Original')
63     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
64     #     std = plt.plot(rolstd, color='black', label = 'Rolling Std')
65     plt.legend(loc='best')
66     plt.title('Rolling Mean & Standard Deviation from column: ' + title)
67     plt.show(block=False)
68
69 def full_graph(predicted_diff, title):
70     dataset_ie = ts[['ice_extent']][-5]
71     predictionsArray = np.asarray(revert_diff(predicted_diff, dataset_ie))
72     zerosArray = np.zeros(dataset_ie.values.size-len(predictionsArray.flatten()))
73     cleanPrediction =
74         pd.Series(np.concatenate((zerosArray,predictionsArray))).replace(0,np.NaN)
75
76     # plot
77     plt.title(title)
78     plt.plot(dataset_ie.index, dataset_ie.values,marker='o',
79             color='blue',label='Actual values')
80     plt.plot(dataset_ie.index, cleanPrediction,marker='o',
81             color='red',label='Last 2 year prediction')
82     plt.ylim([0,20])
83     plt.legend()
84
85     plt.show()
86
87 def full_graph_seasonal(predicted, title):
88     predicted = predicted.flatten() +
89     ts[['ice_extent']].values.flatten()[-48:-24]
90     zerosArray =
91     np.zeros(ts[['ice_extent']].values.size-len(predicted.flatten()))
92     cleanPrediction =
93         pd.Series(np.concatenate((zerosArray,predicted.flatten()))).replace(0,np.NaN)
94
95     # plot
96     plt.title(title)
97     plt.plot(ts.index, ts[['ice_extent']].values,marker='o',
98             color='blue',label='Actual values')
99     plt.plot(ts.index, cleanPrediction,marker='o', color='red',label='Last 2 year
      prediction')
100    plt.ylim([0,20])
101    plt.legend()
102
103    plt.show()
104
105 def revert_diff(predicted_diff, og_data):
106     last_value = og_data.iloc[-predicted_diff.size-1][0]

```

```

100     predicted_actual = np.array([])
101     for value_diff in predicted_diff:
102         actual_value = last_value + value_diff
103         predicted_actual = np.append(predicted_actual, actual_value)
104         last_value = actual_value
105     return predicted_actual
106
107
108 # # Dataprep
109
110 # In[56]:
111
112 ts = pd.read_csv('./data/dataframe_monthly.csv', index_col=0).reset_index()
113
114
115 # In[57]:
116
117
118 ts
119
120
121
122 # In[58]:
123
124
125 ts['date'] = pd.to_datetime(ts['Month'].astype(str) + ts['Year'].astype(str),
126    format='%m%Y', errors='ignore')
127
128 # In[59]:
129
130
131 ts
132
133
134 # In[60]:
135
136
137 ts = ts[['date', 'ice_extent', 'mean_temp']]
138 ts.set_index('date', inplace=True)
139 plt.plot(ts[['ice_extent']])
140 plt.title('ice_extent')
141 plt.show()
142 plt.plot(ts[['mean_temp']])
143 plt.title('mean_temp')
144
145
146 # In[61]:
147
148
149 test_stationarity(ts[['ice_extent']], 'ice_extent')
150 test_stationarity(ts[['mean_temp']], 'mean_temp')
151
152
153 # # Differencing
154

```

```

155 # In[62]:
156
157
158 ts_diff = ts - ts.shift(1)
159 ts_diff = ts_diff.dropna()
160 test_stationarity(ts_diff[['ice_extent']], 'ice_extent')
161 test_stationarity(ts_diff[['mean_temp']], 'mean_temp')
162
163
164 # In[63]:
165
166
167 ts_diff_seasonal = ts - ts.shift(12)
168 ts_diff_seasonal = ts_diff_seasonal.dropna()
169 test_stationarity(ts_diff_seasonal[['ice_extent']], 'ice_extent')
170 test_stationarity(ts_diff_seasonal[['mean_temp']], 'mean_temp')
171
172
173 # In[64]:
174
175
176 tscv = TimeSeriesSplit(n_splits = 18)
177 dataset = ts_diff[:-5] # need the -5 to get testsets for 24 months/2 years
178
179 for train_index, test_index in tscv.split(dataset):
180     if train_index.size > 300:
181
182         # initialize cross validation train and test sets
183         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
184
185         print("TRAIN:", train_index.size) # visualize cross_validation structure
186             # for reference
187         print("TEST:", test_index.size)
188         print()
189
190
191 # In[65]:
192
193
194 tscv = TimeSeriesSplit(n_splits = 18)
195
196 for train_index, test_index in tscv.split(ts_diff_seasonal):
197     if train_index.size > 300:
198
199         # initialize cross validation train and test sets
200         cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
201             ts_diff_seasonal.iloc[test_index]
202
203         print("TRAIN:", train_index.size) # visualize cross_validation structure
204             # for reference
205         print("TEST:", test_index.size)
206         print()
207
208
209 # In[66]:
210
211

```

```

208
209 # og_data = dataset[:-6]
210 # seasonal_entries = 12
211 # n_sets = int(og_data.shape[0]/seasonal_entries)
212 # split_sets = np.array_split(dataset[:-6], n_sets)
213 # ts_diff = pd.DataFrame(columns = og_data.columns)
214
215 # i = 0
216
217 # for year in split_sets[:-1]:
218 #     ## swap 0 and 1 around
219 #     # take difference
220 #     diff = split_sets[i+1].reset_index() - split_sets[i].reset_index()
221 #     diff = diff.iloc[:,1:]
222
223 #     # append to dataframe
224 #     ts_diff = ts_diff.append(diff[['ice_extent', 'mean_temp']],
225 #                               ignore_index=True)
225 #     i = i+1
226
227
228 # In[67]:
229
230
231 # wait
232 ts_seasonal_diff = (dataset - dataset.shift(12)).dropna()
233
234
235 # ## VARMAX
236
237 # ## Random walk differencing
238
239 # In[53]:
240
241
242 from statsmodels.tsa.statespace.varmax import VARMAX
243 import itertools
244 import warnings
245 import sys
246 from sklearn.metrics import mean_absolute_error
247
248
249
250 # Define the p, d and q parameters to take any value between 0 and 2
251 p = q = range(0, 5)
252
253 # Generate all different combinations of p, q and q triplets
254 pq = list(itertools.product(p, q))
255 best_pq = pq
256 best_mean_mae = np.inf
257 warnings.filterwarnings("ignore") # specify to ignore warning messages
258 for param in pq:
259     print(param)
260     try: # some parametercombinations might lead to crash, so catch exceptions
261         and continue
262         maes = []

```

```

262     for train_index, test_index in tscv.split(dataset):
263         if train_index.size > 300:
264             # initialize cross validation train and test sets
265             cv_train, cv_test = dataset.iloc[train_index],
266             ↵ dataset.iloc[test_index]
267
268             # build model
269             model = VARMAX(cv_train, order=(param))
270             model_fit = model.fit()
271
272             # make predictions
273             predictions = model_fit.forecast(steps=24, dynamic=False)
274             prediction_values = predictions[['ice_extent']].values
275             true_values = cv_test[['ice_extent']].values
276             # error calc
277             maes.append(mean_absolute_error(true_values, prediction_values))
278
279             mean_mae = np.mean(maes)
280             print('MAE: ' + str(mean_mae))
281
282             if mean_mae < best_mean_mae:
283                 best_mean_mae = mean_mae
284                 best_maes = maes
285                 best_pq = param
286                 best_predictions = prediction_values
287             except Exception as e:
288                 print(e)
289                 continue
290
291             # plot
292             print()
293             print('Best MAE = ' + str(best_mean_mae))
294             print(best_pq)
295
296             # best range(0,5)
297             # Best MAE = 0.1772618201196872
298             # (1, 1)
299             # Wall time: 7min 53s
300
301
302             # In[175]:
303
304
305             best_pq = (4,1)
306
307
308             # In[176]:
309
310
311             from statsmodels.tsa.statespace.varmax import VARMAX
312             from sklearn.metrics import mean_absolute_error
313             import timeit
314
315
316             start_time = timeit.default_timer()

```

```

317
318 warnings.filterwarnings("ignore") # specify to ignore warning messages
319
320 print("-----")
321
322 maes = []
323
324 for train_index, test_index in tscv.split(dataset):
325     if train_index.size > 300:
326         # initialize cross validation train and test sets
327         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
328
329         # build model
330         model = VARMAX(cv_train, order=(best_pq))
331         model_fit = model.fit()
332
333         # make predictions
334         predictions = model_fit.forecast(steps=24, dynamic=False)
335         prediction_values = predictions[['ice_extent']].values
336         true_values = cv_test[['ice_extent']].values
337         # error calc
338         maes.append(mean_absolute_error(true_values, prediction_values))
339
340         print("I",end="")
341
342
343 time_VARMAX = timeit.default_timer() - start_time
344 mae_mean = np.mean(maes)
345 MAE_VARMAX = mae_mean
346 last_MAE_VARMAX = maes[-1]
347 last_predictions_VARMAX = prediction_values
348
349 print()
350 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_VARMAX)
351 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_VARMAX)
352 print('Execution time: %.3f seconds' % time_VARMAX)
353 full_graph(prediction_values, 'Last prediction VARMAX')
354 print(maes)
355
356
357 # ## Seasonal differencing
358
359 # In[13]:
360
361
362 from statsmodels.tsa.statespace.varmax import VARMAX
363 import itertools
364 import warnings
365 import sys
366 from sklearn.metrics import mean_absolute_error
367
368
369
370 # Define the p, d and q parameters to take any value between 0 and 2
371 p = q = range(0, 5)
372

```

```

373 # Generate all different combinations of p, q and q triplets
374 pq = list(itertools.product(p, q))
375 best_pq = pq
376 best_mean_mae = np.inf
377 warnings.filterwarnings("ignore") # specify to ignore warning messages
378 for param in pq:
379     print(param)
380     try: # some parametercombinations might lead to crash, so catch exceptions
381         ↵ and continue
382         maes = []
383         for train_index, test_index in tscv.split(ts_diff_seasonal):
384             if train_index.size > 300:
385                 # initialize cross validation train and test sets
386                 cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
387                 ↵ ts_diff_seasonal.iloc[test_index]
388
389                 # build model
390                 model = VARMAX(cv_train, order=(param))
391                 model_fit = model.fit()
392
393                 # make predictions
394                 predictions = model_fit.forecast(steps=24, dynamic=False)
395                 prediction_values = predictions[['ice_extent']].values
396                 true_values = cv_test[['ice_extent']].values
397                 # error calc
398                 maes.append(mean_absolute_error(true_values, prediction_values))
399
400             mean_mae = np.mean(maes)
401             print('MAE: ' + str(mean_mae))
402
403             if mean_mae < best_mean_mae:
404                 best_mean_mae = mean_mae
405                 best_maes = maes
406                 best_pq = param
407                 best_predictions = prediction_values
408             except Exception as e:
409                 print(e)
410                 continue
411
412             # plot
413             print()
414             print('Best MAE = ' + str(best_mean_mae))
415             print(best_pq)
416
417             # best range(0,5) = (0,1)
418
419             # In[70]:
420
421
422             best_pq = (0,1)
423
424
425             # In[163]:
426

```

```
427
428 from statsmodels.tsa.statespace.varmax import VARMAX
429 from sklearn.metrics import mean_absolute_error
430 import timeit
431
432
433 start_time = timeit.default_timer()
434
435 warnings.filterwarnings("ignore") # specify to ignore warning messages
436
437 print("-----")
438
439 maes = []
440
441 for train_index, test_index in tscv.split(ts_diff_seasonal):
442     if train_index.size > 300:
443         # initialize cross validation train and test sets
444         cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
445                         → ts_diff_seasonal.iloc[test_index]
446
447         # build model
448         model = VARMAX(cv_train, order=(best_pq))
449         model_fit = model.fit()
450
451         # make predictions
452         predictions = model_fit.forecast(steps=24, dynamic=False)
453         prediction_values = predictions[['ice_extent']].values
454         true_values = cv_test[['ice_extent']].values
455         # error calc
456         maes.append(mean_absolute_error(true_values, prediction_values))
457
458         print("I",end="")
459
460 time_VARMAX_seasonal = timeit.default_timer() - start_time
461 mae_mean_seasonal = np.mean(maes)
462 MAE_VARMAX_seasonal = mae_mean
463 last_MAE_VARMAX_seasonal = maes[-1]
464 last_predictions_VARMAX_seasonal = prediction_values
465
466 print()
467 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_VARMAX_seasonal)
468 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' %
469       → last_MAE_VARMAX_seasonal)
470 print('Execution time: %.3f seconds' % time_VARMAX_seasonal)
471 full_graph_seasonal(last_predictions_VARMAX_seasonal, 'Last prediction VARMAX')
472 print(maes)
473
474 # # SARIMAX
475
476 # In[171]:
477
478
479 # setting up values for displaying prediction of the last 2 years
480 data = dataset
```

```

481 test_size = 24
482 data_train = data[:-test_size]
483 data_test = data[-test_size:]
484
485
486 # In[194]:
487
488
489 # singular test
490
491 import itertools
492 import warnings
493 import sys
494 from statsmodels.tsa.statespace.sarimax import SARIMAX
495 from statsmodels.tsa.arima_model import ARIMA
496
497 from sklearn.metrics import mean_absolute_error
498
499 warnings.filterwarnings("ignore") # specify to ignore warning messages
500
501 # Variables
502 endog = data_train[['ice_extent']]
503 exog = sm.add_constant(data_train[['mean_temp']])
504 exog_test = sm.add_constant(data_test[['mean_temp']])
505
506 # Fit the model
507 mod = sm.tsa.statespace.SARIMAX(endog, exog)
508
509 # fit model
510 model_fit = model.fit()
511
512 yhat = model_fit.forecast(steps = 24, exog=exog_test)
513
514 mae = mean_absolute_error(yhat, data_test[['ice_extent']])
515
516 # plot
517 print(mae)
518 plt.plot(data_test[['ice_extent']], color='blue')
519 plt.plot(yhat, color='red')
520 plt.show()
521
522
523 # Adding exogenous variables with SARIMAX requires to give the mean temperatures
      ↵ from the testset for forecasting, which is not in line with the other
      ↵ usecases, therefore it won't be further explored
524 # (extra research!)
525
526 # # LSTM
527
528 # ## Random walk diff
529
530 # In[72]:
531
532
533 ts
534

```

```

535
536 # In[73]:
537
538
539 # multivariate multi-step encoder-decoder lstm example
540 from numpy import array
541 from numpy import hstack
542 from keras.models import Sequential
543 from keras.layers import LSTM
544 from keras.layers import Dense
545 from keras.layers import RepeatVector
546 from keras.layers import TimeDistributed
547 import warnings
548
549 warnings.filterwarnings("ignore") # specify to ignore warning messages
550
551
552 # split a multivariate sequence into samples
553 def split_sequences(sequences, n_steps_in, n_steps_out):
554     X, y = list(), list()
555     for i in range(len(sequences)):
556         # find the end of this pattern
557         end_ix = i + n_steps_in
558         out_end_ix = end_ix + n_steps_out
559         # check if we are beyond the dataset
560         if out_end_ix > len(sequences):
561             break
562         # gather input and output parts of the pattern
563         seq_x, seq_y = sequences[i:end_ix, :],
564             sequences[end_ix:out_end_ix, :]
565         X.append(seq_x)
566         y.append(seq_y)
567     return array(X), array(y)
568
569 def predict_LSTM(train, test, n_neurons, n_epochs):
570     test['sum'] = test['mean_temp'] + test['ice_extent']
571
572     # define input sequence
573     in_seq1 = train.values[:,0]
574     in_seq2 = train.values[:,1]
575     out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
576
577     # convert to [rows, columns] structure
578     in_seq1 = in_seq1.reshape((len(in_seq1), 1))
579     in_seq2 = in_seq2.reshape((len(in_seq2), 1))
580     out_seq = out_seq.reshape((len(out_seq), 1))
581
582     # horizontally stack columns
583     dataset = hstack((in_seq1, in_seq2, out_seq))
584
585     # choose a number of time steps
586     n_steps_in, n_steps_out = 24, 24
587
588     # convert into input/output
589     X, y = split_sequences(dataset, n_steps_in, n_steps_out)

```

```

590
591     # the dataset knows the number of features, e.g. 2
592     n_features = X.shape[2]
593
594     # define model
595     model = Sequential()
596     model.add(LSTM(n_neurons, activation='relu', input_shape=(n_steps_in,
597                   ↴ n_features)))
597     model.add(RepeatVector(n_steps_out))
598     model.add(LSTM(n_neurons, activation='relu', return_sequences=True))
599     model.add(TimeDistributed(Dense(n_features)))
600     model.compile(optimizer='adam', loss='mae')
601
602     # fit model
603     model.fit(X, y, epochs=n_epochs, verbose=0)
604
605     # demonstrate prediction
606     x_input = test.values
607     x_input = x_input.reshape((1, n_steps_in, n_features))
608     yhat = model.predict(x_input, verbose=0)
609     return yhat
610
611
612
613 # In[18]:
614
615
616 from statsmodels.tsa.statespace.varmax import VARMAX
617 from sklearn.metrics import mean_absolute_error
618 import timeit
619 import tensorflow as tf
620
621 start_time = timeit.default_timer()
622
623 # warnings.filterwarnings("ignore") # specify to ignore warning messages
624 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
625
626 maes = []
627 global_maes = []
628 best_MAE = np.inf
629
630
631 # n_neurons_array = [1,5,10,20]
632 # n_epochs_array = [100,200,300]
633
634 n_neurons_array = [1, 5, 10, 20, 30]
635 n_epochs_array = [100,200,300]
636
637
638 print("-----")
639
640 maes = []
641 for n_neurons in n_neurons_array:
642     for n_epochs in n_epochs_array:
643         for train_index, test_index in tscv.split(dataset):
644             if train_index.size > 300:

```

```

645     # initialize cross validation train and test sets
646     cv_train, cv_test = dataset.iloc[train_index],
647     ↵   dataset.iloc[test_index]
648
649
650
651     prediction_values = yhat[0][:,0]
652     true_values = cv_test[['ice_extent']].values
653
654     # error calc
655     maes.append(mean_absolute_error(true_values, prediction_values))
656
657     print("I",end="")
658     time_LSTM = timeit.default_timer() - start_time
659     MAE_LSTM = np.mean(maes)
660     last_MAE_LSTM = maes[-1]
661     global_maes.append(MAE_LSTM)
662
663     if best_MAE > MAE_LSTM:
664         best_n_neurons = n_neurons
665         best_n_epochs = n_epochs
666         best_MAE = MAE_LSTM
667
668     print()
669     print(n_neurons)
670     print(n_epochs)
671     print(MAE_LSTM)
672     print()
673
674     print('Best:')
675     print('N neurons')
676     print(best_n_neurons)
677     print('Epochs size')
678     print(best_n_epochs)
679     print('MAE')
680     print(best_MAE)
681     plt.bar(range(0,len(global_maes)), global_maes)
682
683
684 # In[177]:
685
686
687 best_n_neurons, best_n_epochs = 30, 300
688
689
690 # In[178]:
691
692
693 from sklearn.metrics import mean_absolute_error
694 import timeit
695 import tensorflow as tf
696
697 start_time = timeit.default_timer()
698
699 # warnings.filterwarnings("ignore") # specify to ignore warning messages

```

```

700 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
701
702
703 print("-----")
704
705 maes = []
706
707 for train_index, test_index in tscv.split(dataset):
708     if train_index.size > 300:
709         # initialize cross validation train and test sets
710         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
711         yhat = predict_LSTM(cv_train, cv_test, best_n_neurons, best_n_epochs)
712
713         prediction_values = yhat[0][:,0]
714         true_values = cv_test[['ice_extent']].values
715
716         # error calc
717         maes.append(mean_absolute_error(true_values, prediction_values))
718
719         print("I",end="")
720
721
722 time_LSTM = timeit.default_timer() - start_time
723 mae_mean = np.mean(maes)
724 MAE_LSTM = mae_mean
725 last_MAE_LSTM = maes[-1]
726 last_predictions_LSTM = prediction_values
727
728 print()
729 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
730 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
731 print('Execution time: %.3f seconds' % time_LSTM)
732 full_graph(last_predictions_LSTM, 'Last prediction LSTM')
733 print(maes)
734
735
736 # ## Seasonal differencing
737
738 # In[76]:
739
740
741
742 from statsmodels.tsa.statespace.varmax import VARMAX
743 from sklearn.metrics import mean_absolute_error
744 import timeit
745 import tensorflow as tf
746
747 start_time = timeit.default_timer()
748
749 # warnings.filterwarnings("ignore") # specify to ignore warning messages
750 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
751
752 maes = []
753 global_maes = []
754 best_MAE = np.inf
755

```

```

756
757 # n_neurons_array = [1, 5, 10, 20]
758 # n_epochs_array = [100, 200, 300]
759
760 n_neurons_array = [1, 5, 10, 20, 30]
761 n_epochs_array = [100, 200, 300]
762
763
764 print("-----")
765
766 maes = []
767 for n_neurons in n_neurons_array:
768     for n_epochs in n_epochs_array:
769         for train_index, test_index in tscv.split(ts_diff_seasonal):
770             if train_index.size > 300:
771                 # initialize cross validation train and test sets
772                 cv_train, cv_test = ts_diff_seasonal.iloc[train_index],
773                               ts_diff_seasonal.iloc[test_index]
774
775                 yhat = predict_LSTM(cv_train, cv_test, n_neurons, n_epochs)
776
777                 prediction_values = yhat[0][:,0]
778                 true_values = cv_test[['ice_extent']].values
779
780                 # error calc
781                 maes.append(mean_absolute_error(true_values, prediction_values))
782
783                 print("I",end="")
784                 time_LSTM = timeit.default_timer() - start_time
785                 MAE_LSTM = np.mean(maes)
786                 last_MAE_LSTM = maes[-1]
787                 global_maes.append(MAE_LSTM)
788
789                 if best_MAE > MAE_LSTM:
790                     best_n_neurons = n_neurons
791                     best_n_epochs = n_epochs
792                     best_MAE = MAE_LSTM
793
794                 print()
795                 print(n_neurons)
796                 print(n_epochs)
797                 print(MAE_LSTM)
798                 print()
799
800 print('Best:')
801 print('N neurons')
802 print(best_n_neurons)
803 print('Epochs size')
804 print(best_n_epochs)
805 print('MAE')
806 print(best_MAE)
807 plt.bar(range(0,len(global_maes)), global_maes)
808
809
810 # In[164]:
```

```

811
812
813 best_n_neurons, best_n_epochs = 1, 200
814
815
816 # In[165]:
817
818
819 from sklearn.metrics import mean_absolute_error
820 import timeit
821 import tensorflow as tf
822
823 start_time = timeit.default_timer()
824
825 # warnings.filterwarnings("ignore") # specify to ignore warning messages
826 tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
827
828
829 print("-----")
830
831 maes = []
832
833 for train_index, test_index in tscv.split(dataset):
834     if train_index.size > 300:
835         # initialize cross validation train and test sets
836         cv_train, cv_test = dataset.iloc[train_index], dataset.iloc[test_index]
837         yhat = predict_LSTM(cv_train, cv_test, best_n_neurons, best_n_epochs)
838
839
840         prediction_values = yhat[0][:,0]
841         true_values = cv_test[['ice_extent']].values
842
843         # error calc
844         maes.append(mean_absolute_error(true_values, prediction_values))
845
846         print("I",end="")
847
848
849 time_LSTM_seasonal = timeit.default_timer() - start_time
850 mae_mean_seasonal = np.mean(maes)
851 MAE_LSTM_seasonal = mae_mean
852 last_MAE_LSTM_seasonal = maes[-1]
853 last_predictions_LSTM_seasonal = prediction_values
854
855 print()
856 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM_seasonal)
857 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' %
858      last_MAE_LSTM_seasonal)
859 print('Execution time: %.3f seconds' % time_LSTM_seasonal)
860 full_graph_seasonal(last_predictions_LSTM_seasonal, 'Last prediction LSTM')
861 print(maes)
862
863 # # Evaluation
864
865 # In[181]:

```

```
866
867
868 # formatting
869 results = [[MAE_VARMAX, time_VARMAX, last_MAE_VARMAX],
870             [MAE_VARMAX_seasonal, time_VARMAX_seasonal, last_MAE_VARMAX_seasonal],
871             [MAE_LSTM, time_LSTM, last_MAE_LSTM],
872             [MAE_LSTM_seasonal, time_LSTM_seasonal, last_MAE_LSTM_seasonal]]
873
874 # display results
875 results = pd.DataFrame(results, columns=['Mean MAE (x 1 000 000
876     ↵ km\u00b2)', 'Execution time (s)', 'Last MAE (x 1 000 000 km\u00b2)']
877             ↵ , index=['VARMAX', 'VARMAX_seasonal', 'LSTM', 'LSTM_seasonal']).round(decimals=3)
878
879
880 # In[182]:
881
882
883 full_graph(last_predictions_VARMAX, 'Last prediction VARMAX with random walk
884     ↵ differencing')
885 full_graph_seasonal(last_predictions_VARMAX_seasonal, 'Last prediction VARMAX
886     ↵ with seasonal differencing')
887 full_graph(last_predictions_LSTM, 'Last prediction LSTM with random walk
888     ↵ differencing')
889 full_graph_seasonal(last_predictions_LSTM_seasonal, 'Last prediction LSTM with
890     ↵ seasonal differencing')
```

Bibliografie

- Brown, S. (2020). Least Squares — the Gory Details. Verkregen van <https://brownmath.com/stat/leastsq.htm>
- Brownlee, J. (2017a). Encoder-Decoder Long Short-Term Memory Networks. Verkregen 24 november 2020, van <https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/>
- Brownlee, J. (2017b). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Verkregen 24 november 2020, van <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
- Brownlee, J. (2017c). Stacked Long Short-Term Memory Networks. Verkregen van <https://machinelearningmastery.com/stacked-long-short-term-memory-networks/#:~:text=A%20Stacked%20LSTM%20architecture%20can,for%20all%20input%20time%20steps>
- Brownlee, J. (2018a). 11 Classical Time Series Forecasting Methods in Python (Cheat Sheet). Verkregen 24 november 2020, van <https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/>
- Brownlee, J. (2018b). A Gentle Introduction to SARIMA for Time Series Forecasting in Python. Verkregen 24 november 2020, van <https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/>
- Brownlee, J. (2018c). How to Develop LSTM Models for Time Series Forecasting. Verkregen 20 mei 2020, van <https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/>
- Forecasting at scale. (2020). Verkregen van <https://facebook.github.io/prophet/>
- Johan Decorte, S. D. V. (2019). Machine learning with PYTHON.
- Kampakis, S. (2020). Performance measures: RMSE and MAE. Verkregen van <https://thedataScientist.com/performance-measures-rmse-mae/>
- Kenton, W. (2020). Multiple Linear Regression (MLR). Verkregen 24 november 2020, van <https://www.investopedia.com/terms/m/mlr.asp>

- Lau, R. (2020). Introduction to ARIMA: nonseasonal models. Verkregen van <https://people.duke.edu/~rnau/411arim.htm>
- Lievens, S. (2018). Artificiële Intelligentie, lesnota's HOGENT, "147–149".
- Liu, D. (2020). A Practical Guide to ReLU. Verkregen 24 november 2020, van <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>
- Olah, C. (2015). Understanding LSTM Networks. Verkregen 26 mei 2020, van <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Rob J Hyndman, G. A. (2018). Forecasting: Principles and Practice. Verkregen van <https://otexts.com/fpp2/>
- Shrivastava, S. (2020). Cross Validation in Time Series. Verkregen van <https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4>
- Sinha, P. (2019). LINEAR REGRESSION,LEAST-SQUARES REGRESSION,OUTLIERS AND INFLUENTIAL OBSERVATIONS,RESIDUALS,LURKING VARIABLES,EXTRAPOLATION. Verkregen van <https://cafepharmablog.wordpress.com/2019/02/20/linear-regressionleast-squares - regressionoutliers - and - influential - observationsresidualslurking - variablesextrapolation/>
- Srinivasan, A. V. (2019). Stochastic Gradient Descent — Clearly Explained !! Verkregen 24 november 2020, van <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31>
- Starmer, J. (2019). Gradient Descent, Step-by-Step. Verkregen van https://www.youtube.com/watch?v=sDv4f4s2SB8&ab_channel=StatQuestwithJoshStarmer
- Swaminathan, S. (2018). Logistic Regression — Detailed Overview. Verkregen van <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>
- Syed Ashiqur Rahman, D. A. A. (2019). Deep Learning using Convolutional LSTM estimates Biological Age from Physical Activity. Verkregen van <https://www.nature.com/articles/s41598-019-46850-0>