



Faculteit Bedrijf en Organisatie

Vergelijkende studie van voorspellingsmodellen voor 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:

Academiejaar: 2020-2021

Eerste examenperiode

Faculteit Bedrijf en Organisatie

Vergelijkende studie van voorspellingsmodellen voor 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:

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 vlotte begeleiding van deze bachelorproef en Stijn Lievens voor het opnemen van het co-promotorschap.

Mijn ouders wil ik ook hartelijk bedanken voor alle steun, niet alleen 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. Tenslotte verdient ook mijn zus, Edith Declercq 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.

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 voorspellingstechniek die onder de loep zal genomen is polynomiale regressie. De tweede techniek die zal toegepast worden is een ARIMA/VARMAX model. Tenslotte zal een recurrent neurale netwerk van het type LSTM (Long Term Short Memory) op dezelfde data toegepast worden. Voor elk van deze 4 types tijdreeksen zal dan bepaald worden welk van de 3 technieken het beste resultaat zal opleveren.

Inhoudsopgave

1	Methodologie	11
1.1	Vorbereiding	11
1.1.1	Datavorbereiding	11
1.2	Univariate niet-seizoensgebonden	20
1.2.1	Algemene methodes	20
1.2.2	Stationariteit	21
1.2.3	Cross validation	23
1.2.4	ARIMA	23
1.2.5	LSTM	27
1.2.6	Prophet	33
1.2.7	Evaluatie	36
1.3	Univariate seizoensgebonden	38
1.3.1	Algemene methodes	38

1.3.2	Stationariteit	39
1.3.3	ARIMA	40
1.3.4	SARIMA	41
1.3.5	LSTM	42
1.3.6	Prophet	42
1.3.7	Evaluatie	44
1.4	Multivariate niet-seizoensgebonden	45
1.4.1	Stationariteit	45
1.4.2	VARMAX	47
1.4.3	LSTM	47
1.4.4	Evaluation	48
1.5	Multivariate seizoensgebonden	48

Lijst van figuren

1.1	Ruwe invoerdata	13
1.2	Grafische weergave jaarlijkse temperatuur	13
1.3	Resultaat van data formatting	14
1.4	Ruwe data van de jaarlijkse ijsdiktes	16
1.5	Grafische weergave van de maandelijkse ijsdiktes	17
1.6	Data van de maandelijks ijsdiktes	17
1.7	Grafische weergave van de ijsdikte en de temperatuur van de laatste 2 jaar	19
1.8	Grafische weergave van de ijsdikte en de temperatuur	19
1.9	Resultaat test stationarity	22
1.10	Resultaat test stationarity na random walk differencing	22
1.11	Resultaat finale iteratie ARIMA	26
1.12	Grafische weergave verschil in MAE's bij verschillende hyperparameters met MAE op de y-as	30
1.13	Resultaat finale iteratie LSTM	32
1.14	Resultaat finale iteratie Prophet	35
1.15	Resultaat van de univariate non-seasonal analyse	36

1.16	Grafische weergave van de laatste voorspellingen van de univariate non-seasonal analyse	37
1.17	39
1.18	40
1.19	40
1.20	Resultaat van ARIMA bij random walk differentiatie	41
1.21	Resultaat van ARIMA bij seizoensdifferentiatie	41
1.22	Resultaat SARIMAX bij random walk differentiatie	42
1.23	Resultaat SARIMAX bij seizoensdifferentiatie	43
1.24	Resultaat LSTM bij seizoensdifferentiatie	43
1.25	Grafische weergave multivariate tijdreeks zonder seizoenseffect	45
1.26	Grafische weergave gedifferentieerde multivariate tijdreeks zonder seizoenseffect	46
1.27	Resultaat multivariate voorspelling VARMAX	47
1.28	Resultaat multivariate voorspelling LSTM	49
1.29	Resultaten multivariate voorspelling	49
1.30	Grafische weergaven van de laatste voorspellingen van de VARMAX en LSTM modellen	50

1. 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.

1.1 Voorbereiding

1.1.1 Datavoorbereiding

Dataset 1, temperatuur

Hier zal het 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 1: 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 1 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 1.2.

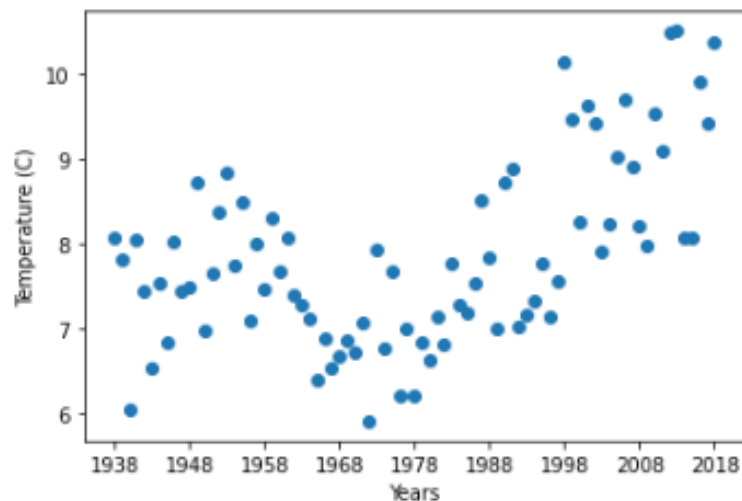
Op lijn 3 worden enkel de waarden van de eerste dag van de maand behouden in de dataset. Daarna worden van lijn 4 tot lijn 9 de jaartallen aangepast zodat deze overeenkomen met

Figuur 1.1: 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 1.2: 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 ijkings gedefiniëerd van de grafiek die de data zal weergeven en zichtbaar is op figuur 1.2 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 1.3: 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 dataformattering uitgevoerd zichtbaar op figuur 1.3.

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 2: Datavoorbereiding dataset temperatuur

```

1  #source: https://nsidc.org/arcticseaicenews/sea-ice-tools/
2  ice2 = pd.read_csv('./data/seaice2.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 1.4: 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 ' +
    ↳ ice2_monthly_mean_chron.index.values[-1])
46 ice2_monthly_mean_chron =
    ↳ 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 1.4
 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 geïnitieerd 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.

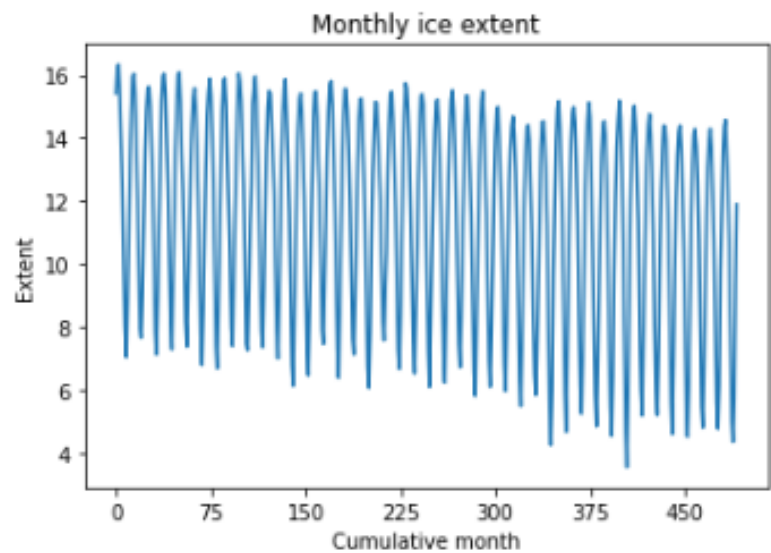
Het deel code van lijn 37 tot en met lijn 42 zal zorgen voor de grafische weergave van de maandelijks ijsdikte weergegeven op figuur 1.5.

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 figuur 1.6.

Figuur 1.5: Grafische weergave van de maandelijkse ijsdiktes



Figuur 1.6: 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 3: 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 geïntialiseerd 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 duidelijkere variabelenaam.

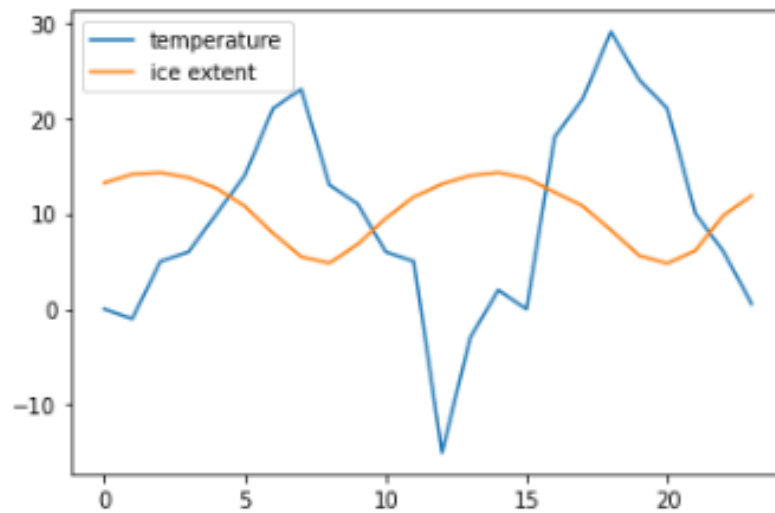
De code op lijn 13 tot 17 zorgt voor de grafische weergave van de gecombineerde dataset van de laatste 2 jaren weergegeven op figuur 1.7. Hier valt op te merken dat de ijsdikte zal vergroten wanneer de temperatuur laag is. Dit is logisch aangezien het ijs zal smelten bij hogere temperaturen.

De code op lijn 17 zal het jaarlijks gemiddelde nemen.

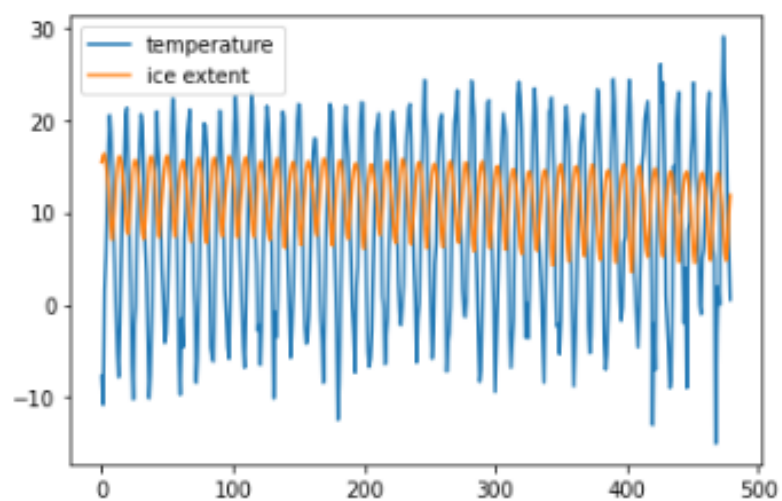
Op lijn 20 tot 23 wordt de code weergegeven die ervoor zal zorgen dat het volledige maandlijks gemiddelde weergegeven zal worden van 1979 tot 2018. Dit valt te beschouwen op figuur 1.7. Hierop kan vastgesteld worden dat de temperatuur lichtjes zal dalen terwijl de ijsdikte zal stijgen.

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

Figuur 1.7: Grafische weergave van de ijsdikte en de temperatuur van de laatste 2 jaar



Figuur 1.8: Grafische weergave van de ijsdikte en de temperatuur



1.2 Univariate niet-seizoensgebonden

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

1.2.1 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 grafiektitel.

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 4: 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
20  plt.show()
21
22  # define function for reverting a differenced dataset
23  def revert_diff(predicted_diff, og_data):
24
25      # retrieve last value
26      last_value = og_data.iloc[-predicted_diff.size-1][0]
27
28      # initialize reverted array
29      predicted_actual = np.array([])
30
31      # add each value in the differenced array with the last actual value
32      for value_diff in predicted_diff:
33          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

```

1.2.2 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 5: 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 1.9. 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 6: Test stationarity bij random walk differencing

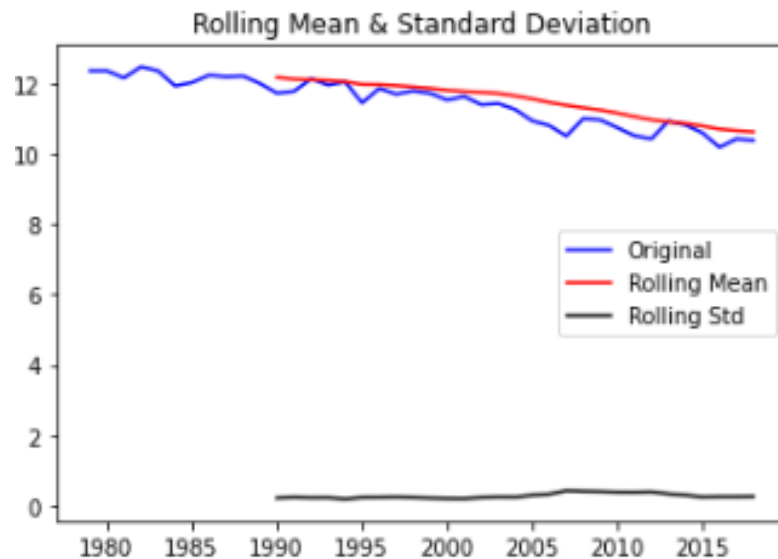
```

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)

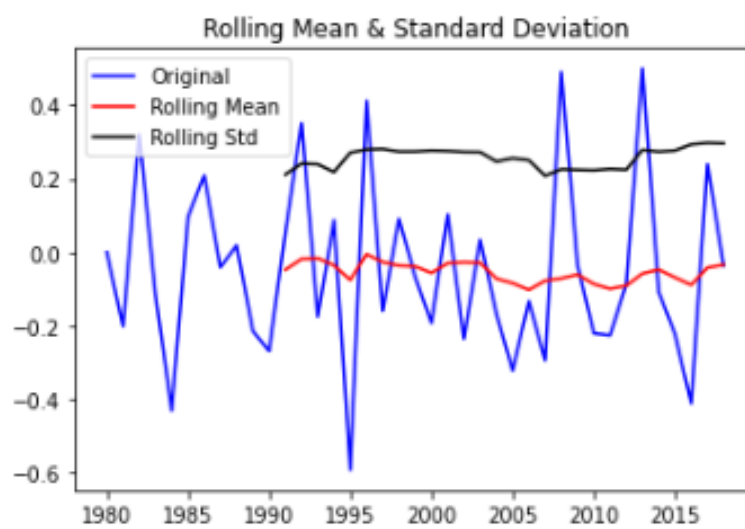
```

Door hiervan nogmaals de stationariteit te testen wordt figuur 1.10 bekomen. Hierop valt af te lezen dat de data nu wel stationair is.

Figuur 1.9: Resultaat test stationarity



Figuur 1.10: Resultaat test stationarity na random walk differencing



1.2.3 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 7: 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
   → 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         # visualiize cross_validation structure for reference
12         print("TRAIN:", train_index.size)
13         print("TEST:", test_index.size)
14         print()

```

1.2.4 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 8: 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 parameter combinations might lead to crash, so catch exceptions and
    ↪ continue
28     try:
29
30         # initialize the array which will contain the mean average errors
31         maes = []
32
33         # loop through all split time series that have a trainingsset with more
    ↪ than 20 values
34         for train_index, test_index in tscv.split(ts_diff):
35             if train_index.size > 20:
36
37                 # initialize cross validation train and test sets
38                 cv_train, cv_test = ts_diff.iloc[train_index],
    ↪ ts_diff.iloc[test_index]
39
40                 # build model
41                 model = ARIMA(cv_train, order=(param))
42
43                 # fit model
44                 model_fit = model.fit()
45
46                 # make predictions
47                 predictions = model_fit.predict(start=len(cv_train),
    ↪ end=len(cv_train)+cv_test.size-1, dynamic=False)
48
49                 # renaming for clarity
50                 prediction_values = predictions.values
51                 true_values = cv_test.values
52
53                 # error calculation this part of the cross validation
54                 maes.append(mean_absolute_error(true_values, prediction_values))
55
56
57         # error calculation for this parameter combination
58         mean_mae = np.mean(maes)
59         print('MAE: ' + str(mean_mae))
60
61         # store parameters resulting in the lowest mean MAE
62         if mean_mae < best_mean_mae:
63             best_mean_mae = mean_mae
64             best_maes = maes
65             best_pdq = param
66             best_predictions = prediction_values
67

```

```

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.

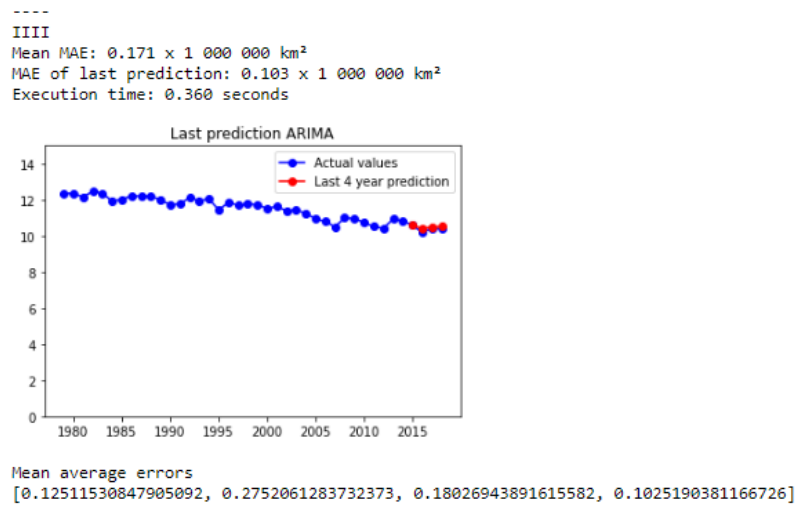
Listing 9: Finale iteratie ARIMA

```

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 through all split time series that have a trainingset with more than 20
   ↳ values
12 for train_index, test_index in tscv.split(ts_diff):
13     if train_index.size > 20:
14
15         # initialize cross validation train and test sets
16         cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]
17
18         # build model
19         arima = ARIMA(cv_train, best_pdq).fit(start_ar_lags=1, disp=False)
20
21         # make predictions
22         predictions = arima.forecast(steps=4)
23         prediction_values = predictions[0]
24         true_values = cv_test.values
25
26         # error calc
27         maes.append(mean_absolute_error(true_values, prediction_values))
28
29         # last actual prediction
30         last_prediction_ARIMA = prediction_values
31
32     print("I", end="")
33

```

Figuur 1.11: Resultaat finale iteratie ARIMA



```

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 figuur1.11. Hier wordt de gemiddelde MAE overheen 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.

1.2.5 LSTM

Het volgende model dat getest zal worden is een LSTM model. Aangezien dit een neurale 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 dienen 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 10: 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 11: 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:

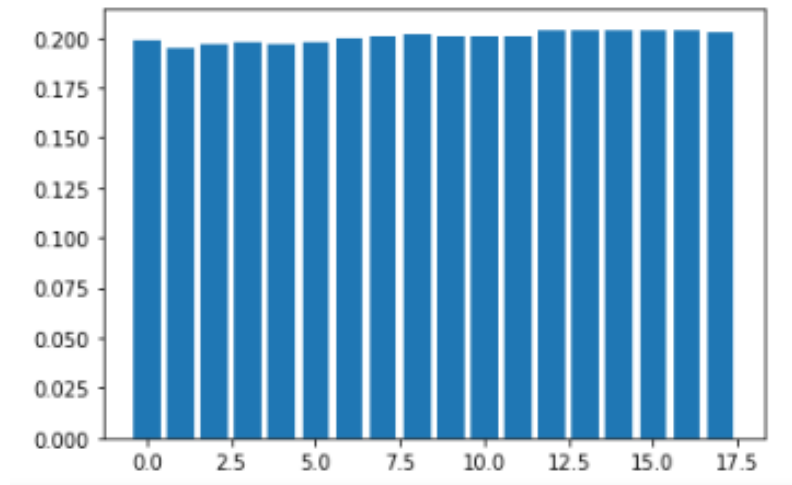
```

```

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 1.12: 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 1.12 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 12: 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

```

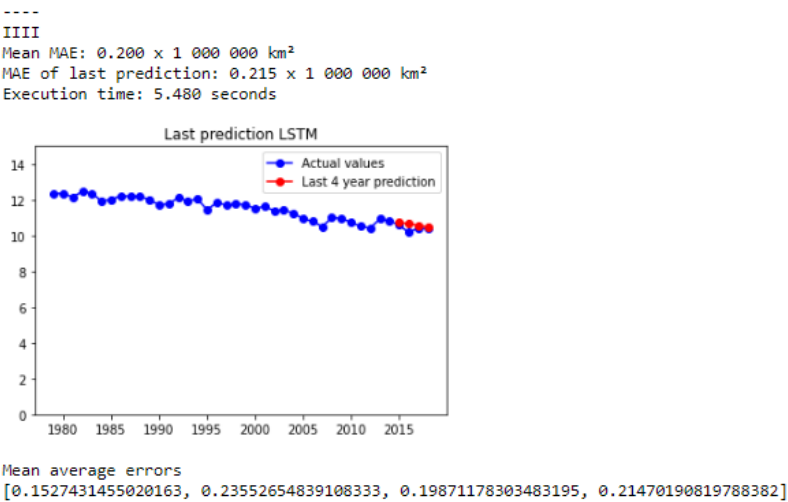
```

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
  ↪ 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,
  ↪ n_neurons, dropout, batch_s)
31
32         # make predictions
33         x_input = array(y_test)
34         y_predicted = predict(x_input, lstm_model, n_features).flatten()
35         y_actual = y_test.values
36
37         # error calc
38         maes.append(mean_absolute_error(y_actual, y_predicted))
39
40         print("I",end="")
41
42 # last actual prediction
43 last_prediction_LSTM = y_predicted
44
45 # store variables
46 time_LSTM = timeit.default_timer() - start_time
47 MAE_LSTM = np.mean(maes)
48 last_MAE_LSTM = maes[-1]
49
50 # visualisation
51 print()
52 print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
53 print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
54 print('Execution time: %.3f seconds' % time_LSTM)
55 full_graph(last_prediction_LSTM, 'Last prediction LSTM')
56 print('Mean average errors')
57 print(maes)

```

De bovenstaande code zal leiden tot de uitvoer die zichtbaar is op figuur??. Ook hier kan de gemiddelde MAE overheen de verschillende iteraties bij cross validation weergegeven worden 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. Net zoals bij ARIMA.

Figuur 1.13: Resultaat finale iteratie LSTM



1.2.6 Prophet

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

Listing 13: Code voor het formatteren van de data voor Prophet

```

1 # formatting dataframe
2 ts_formated_prophet = ts_diff.reset_index().rename(columns = {'Year' : 'ds',
   ↪ 'ice_extent' : 'y'})
3 ts_formated_prophet['ds'] =
   ↪ pd.DataFrame(pd.to_datetime(ts_formated_prophet['ds']).astype(str),
   ↪ 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 14: 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
   ↪ itertools.product(*param_grid.values())]
15
16 # initialize variables
17 maes = []
18 global_maes = []
19 best_MAE_prophet = np.inf
20
21 # Use cross validation to evaluate all parameters
22 for params in all_params:
23
24     # loop trough all split time series that have a trainingsset with more than
   ↪ 20 values
25     for train_index, test_index in tscv.split(ts_formated_prophet):
26         if train_index.size > 20:
27
28             # initialize cross validation train and test sets
29             train = ts_formated_prophet.iloc[train_index]
30             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 1.14 verkregen.

Listing 15: Finale iteratie Prophet

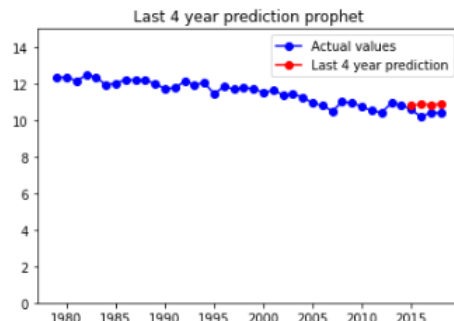
```

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 1.14: 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 1.15: 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

1.2.7 Evaluatie

Wanneer deze resultaten gecombineerd worden wordt de tabel die zichtbaar is op figuur 1.15 verkregen. Om dit resultaat grafisch te schetsen worden op figuur 1.16 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 16: Code voor het weergeven van de resultaten

```

1 # formatting
2 results =
  → [[MAE_ARIMA,time_ARIMA,last_MAE_ARIMA],[MAE_LSTM,time_LSTM,last_MAE_LSTM],[MAE_Prophet,time_Pr
3
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)

```

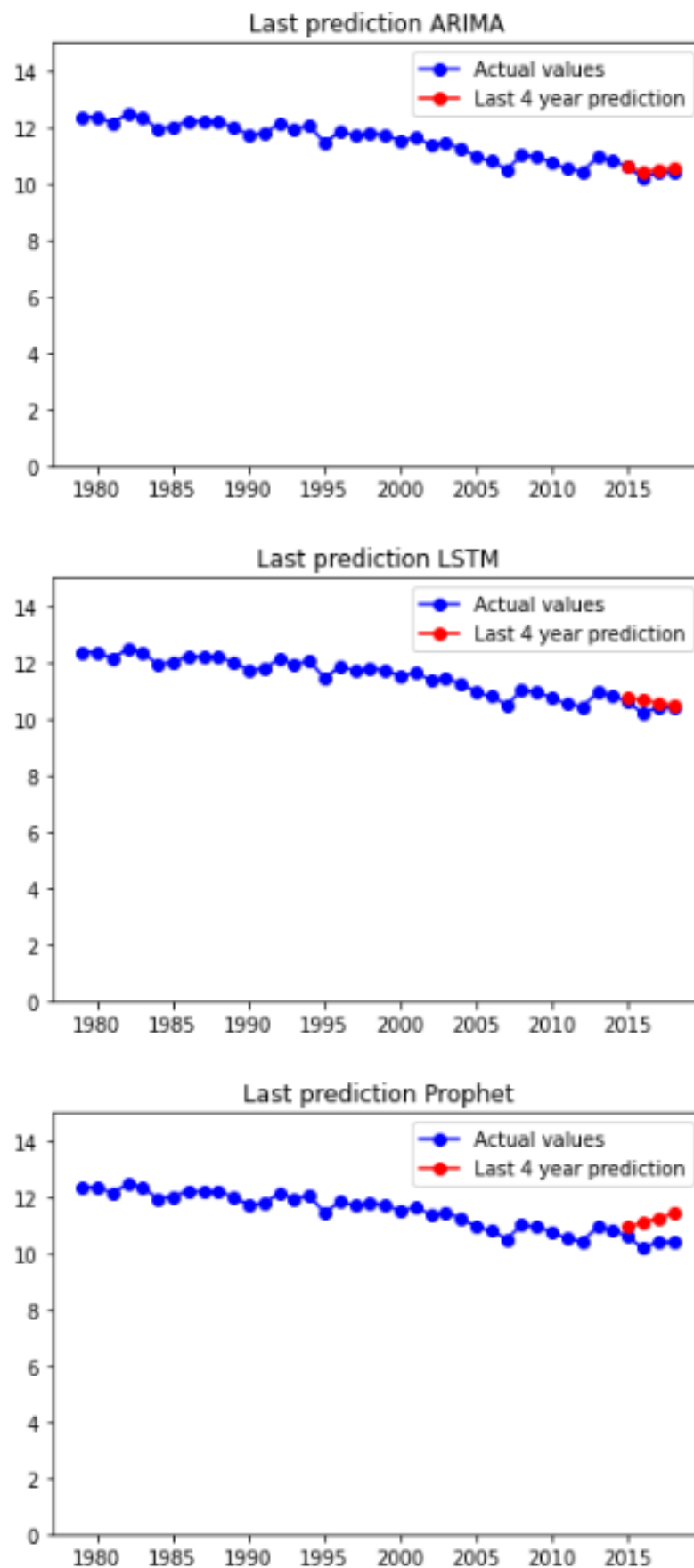
Listing 17: Code voor het grafisch weergeven van de resultaten

```

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



1.3 Univariate seizoensgebonden

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

1.3.1 Algemene methodes

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

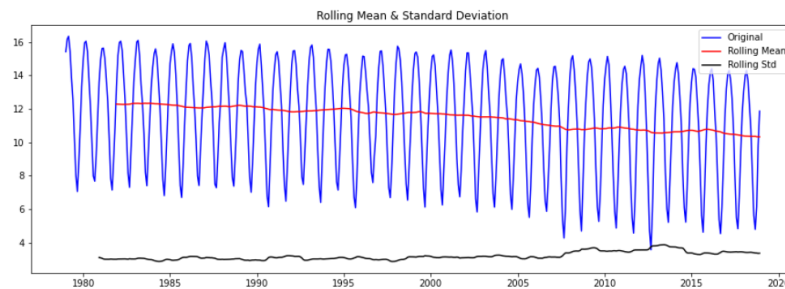
Zo herkennen kunnen de `test_stationarity`, `full_graph` en `revert_diff` methodes van de voorgaande sectie herkend worden. Naast deze methodes wordt hier ook nog de `revert_seasonal_diff_recursion` methode gedefinieerd. Deze wordt benut bij de seizoensgebonden variant van de `revert_diff` methode namelijk `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 18: 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 1.17



```

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:]

```

1.3.2 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 1.17. Deze loopt gelijk met de trend van de niet seizoengebonden data aangezien die data het jaarlijks gemiddelde is en dit maandelijkse waarden zijn.

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

Listing 19: Code voor seizoensdifferentiatie

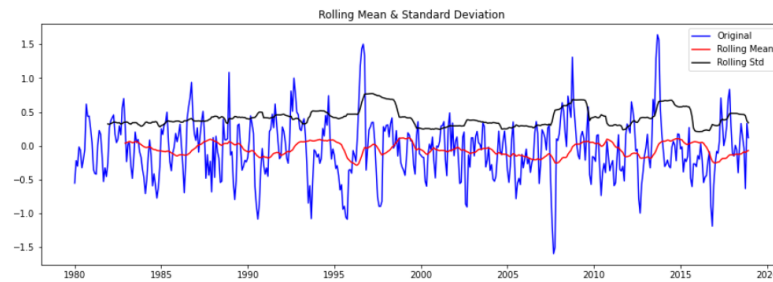
```

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

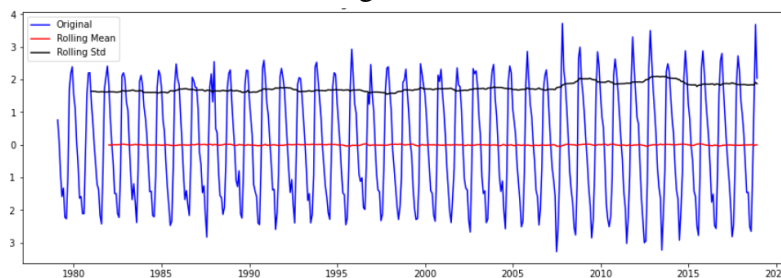
```

Wanneer seizoensdifferentiatie toegepast wordt zal de stationariteit voorgesteld kunnen worden zoals zichtbaar op figuur 1.18. Dit is wel stationair maar fluctueert wel behoorlijk naar het einde toe.

Figuur 1.18



Figuur 1.19



Daarnaast kan ook nog gewone random walk differentiatie toegepast worden op de data. De stationariteit van deze gedifferentieerde data wordt afgebeeld op figuur 1.19. Hier wordt vastgesteld dat het gemiddelde een horizontale lijn is. Dit houdt in dat de data zeer stationair is.

Listing 20: 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 seizoensdifferentiatie 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 seizoensdifferentiatie 6 trainingssets zullen zijn die meer dan 300 waarden bevatten.

1.3.3 ARIMA

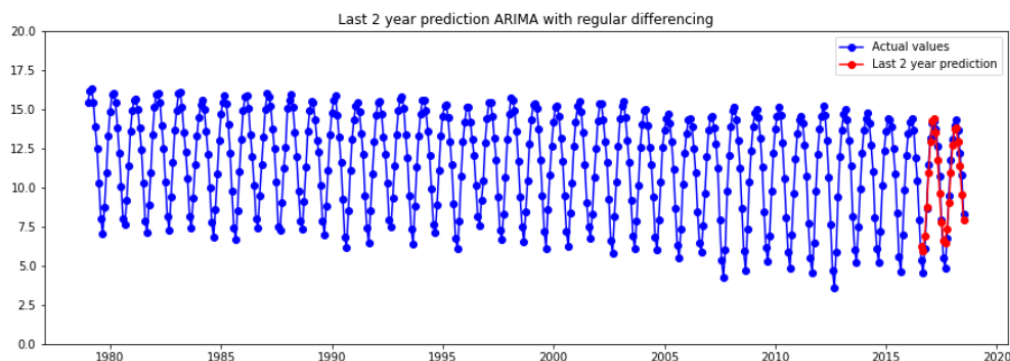
Ook hier zal een ARIMA model opgesteld worden zowel voor de random walk gedifferentieerde data als de seizoensgedifferentieerde data.

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.

Figuur 1.20: Resultaat van ARIMA bij random walk differentiatie

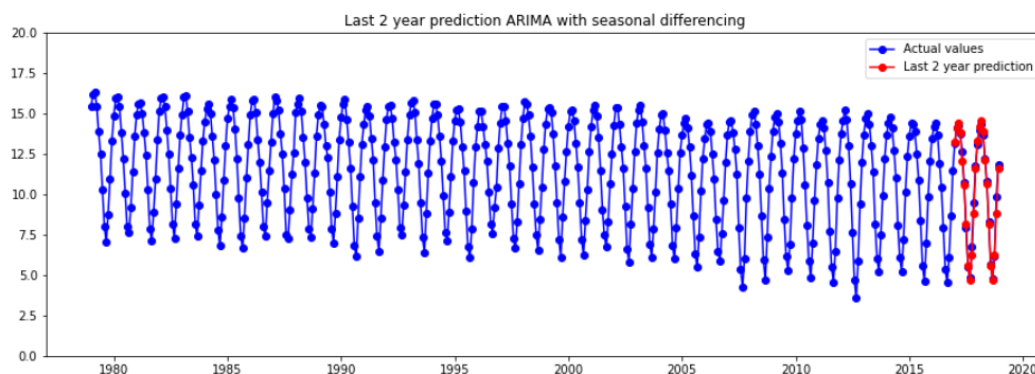
Mean MAE: $0.587 \times 1\,000\,000\text{ km}^2$
 MAE of last prediction: $0.535 \times 1\,000\,000\text{ km}^2$
 Execution time: 4.981 seconds



[0.5519202505373405, 0.5339985209288152, 0.624088160462872, 0.6175479246639486, 0.6547096229140431, 0.5949445312188929, 0.5349483168514487]

Figuur 1.21: Resultaat van ARIMA bij seizoensdifferentiatie

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



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

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 1.20 verkregen.

Seizoensdifferentiatie

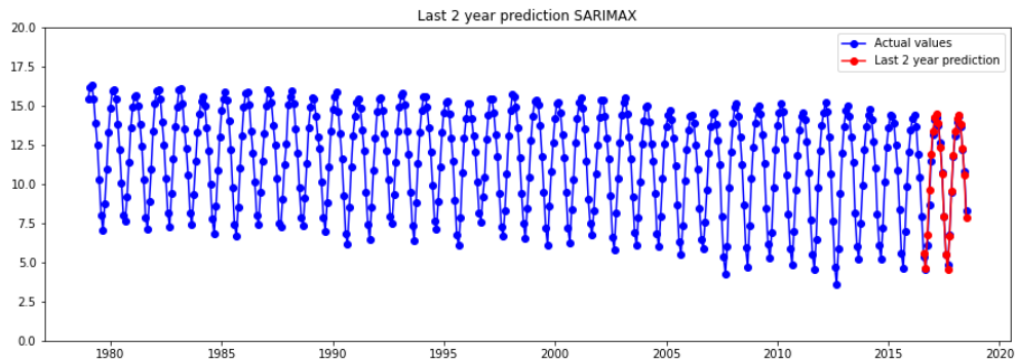
Ook hier zullen eerst de hyperparameters bepaald worden. Met (3,0,3) als optimale waarden bij een range van 0 tot 5 voor de waarden p of q . Wanneer deze waarden uitgebreider getest worden resulteert dit in de uitvoer die waar te nemen valt op figuur 1.21.

1.3.4 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

Figuur 1.22: Resultaat SARIMAX bij random walk differentiatie

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



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

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 (1,0,2,0,1,2) zijn. Het uitgebreid resultaat van deze combinatie wordt weergegeven op figuur 1.22.

Seizoensdifferentiatie

Deze testopstelling is identiek aan de vorige uitgezonderd van de invoerdata die hier seizoensgedifferentieerd zal zijn. Ook hier zullen de optimale parameters (1,0,2,0,1,2) zijn. Dit de uitvoer geven die afgebeeld staat op figuur 1.23.

1.3.5 LSTM

Random walk differentiatie

Seizoensdifferentiatie

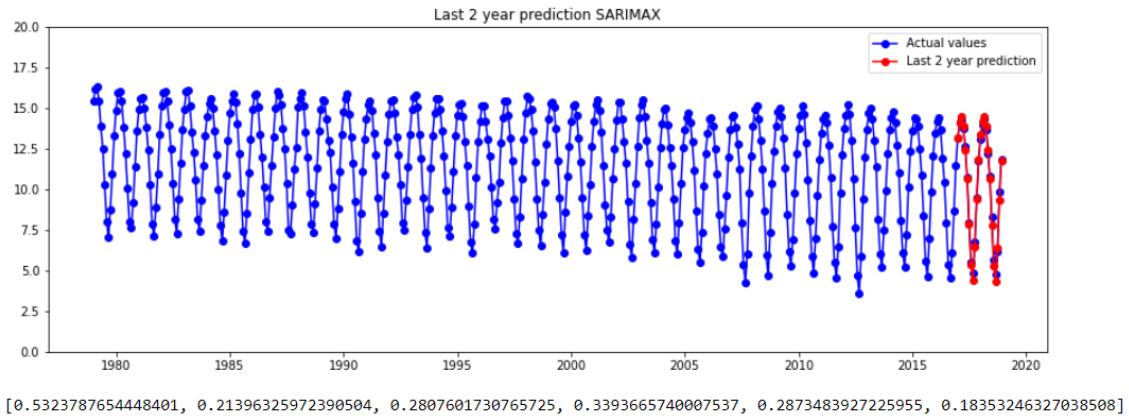
De opstelling zal identiek zijn als de bovenstaande hier zullen de optimale parameters bestaan uit 1 neuron een dropout rate van 0.99 en een batchgrootte van 8 met het eindresultaat dat zich op figuur 1.24 bevindt.

1.3.6 Prophet

Ook hier verloopt alles bijna analoog met de niet seizoensgebonden variatie worden er hier wel 2 hyperparameters getest

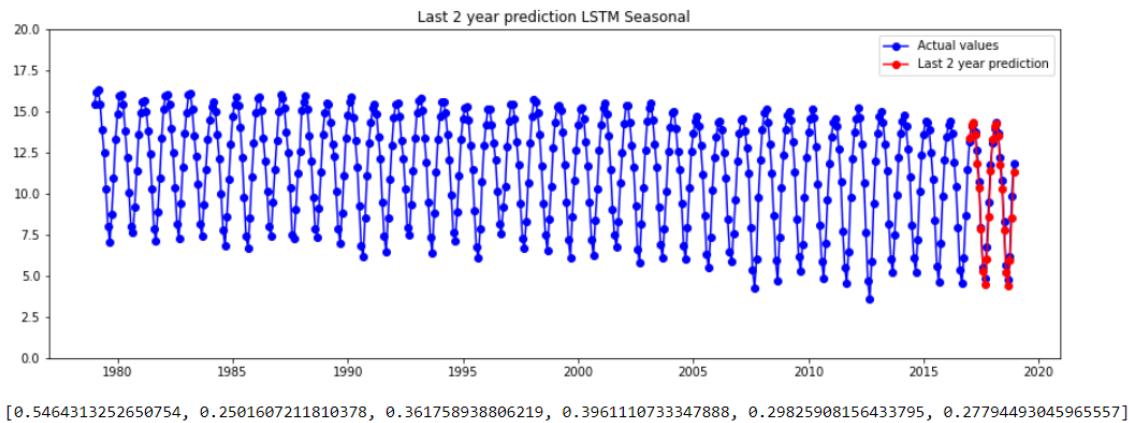
Figuur 1.23: Resultaat SARIMAX bij seizoensdifferentiatie

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



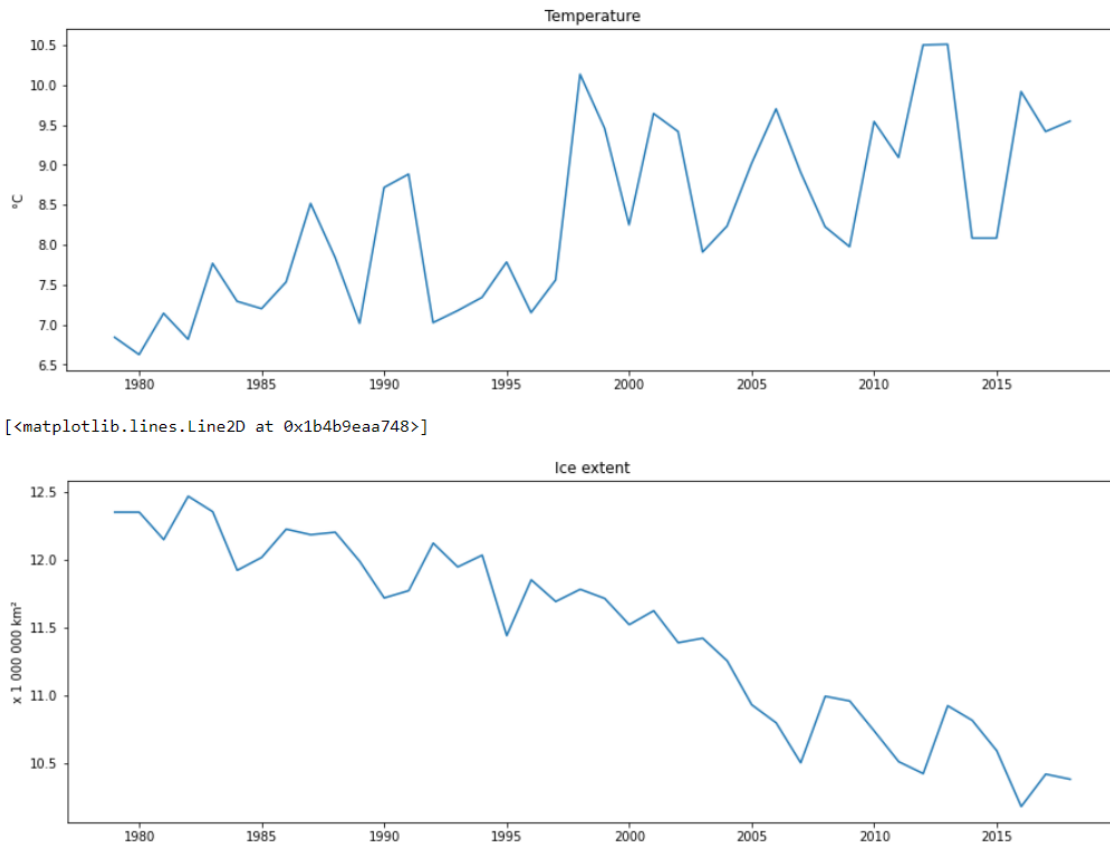
Figuur 1.24: Resultaat LSTM bij seizoensdifferentiatie

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: 140.942 seconds



1.3.7 Evaluatie

Figuur 1.25: Grafische weergave multivariate tijdreeks zonder seizoenseffect



1.4 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 tijdstappen die voorspeld moeten worden.

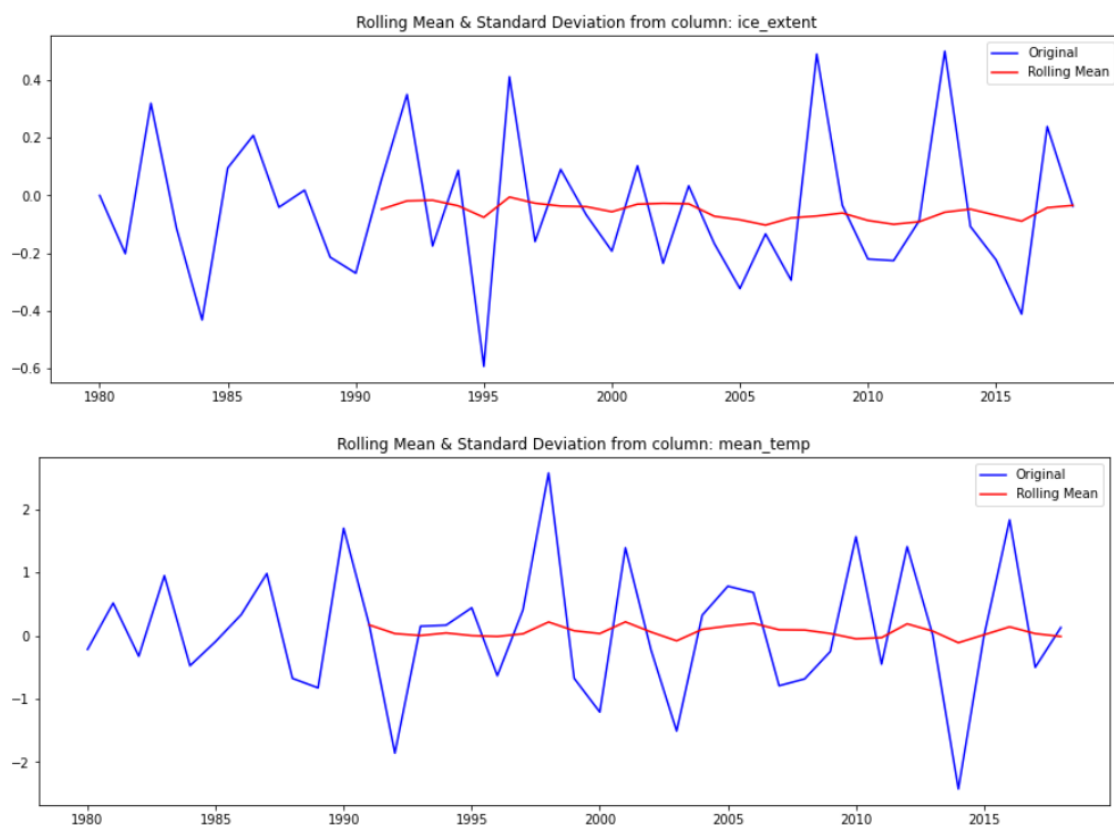
De data die gebruikt zal worden voor deze sectie wordt grafisch weergegeven op Figuur 1.25. Ook hier zal er gebruik gemaakt worden van cross validation.

Aangezien Prophet geen multivariate tijdreeksen als invoer ondersteunt zal dit niet geëvalueerd kunnen worden.

1.4.1 Stationariteit

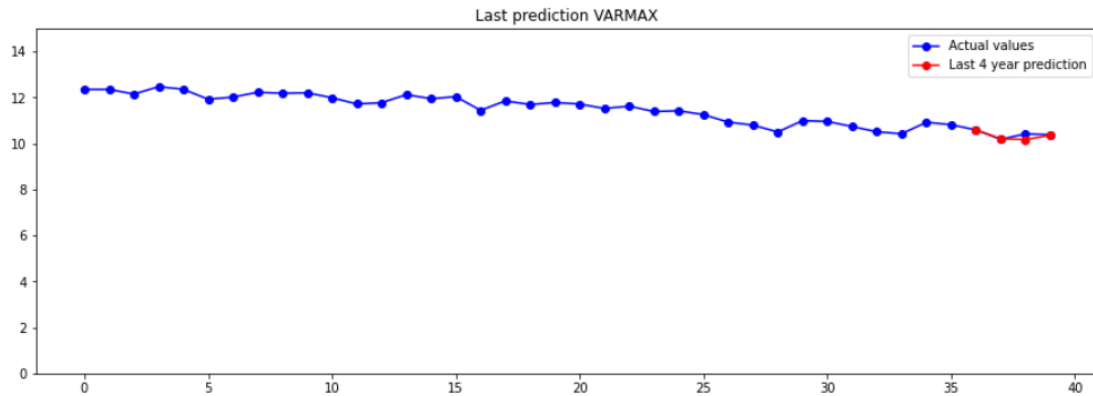
Ook hier zal de ijsdikte een dalende trend vertonen zoals weergegeven werd bij de univariate analyse en hier zal niks aan veranderen aangezien dit dezelfde data is. Ook deze data zal random walk gedifferentieerd worden maar dit keer niet enkel de ijsdikte maar ook de temperatuur. De resultaten van die differentiatie zijn zichtbaar op figuur 1.26. Daaruit kan afgeleid worden dat de data nu stationair is.

Figuur 1.26: Grafische weergave gedifferentieerde multivariate tijdreeks zonder seizoens-effect



Figuur 1.27: 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



[0.13250395150566072, 0.255018330608453, 0.09309881825606907, 0.13371550021338208]

1.4.2 VARMAX

Aangezien een ARIMA model slechts in staat is om tijdreeksen met 1 variabele te voorspellen 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 1.27.

1.4.3 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 21.

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 1.28.

Listing 21: 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    # covert 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)))
31    model.add(RepeatVector(n_steps_out))
32    model.add(LSTM(n_neurons, activation='relu', return_sequences=True))
33    model.add(TimeDistributed(Dense(n_features)))
34    model.compile(optimizer='adam', loss='mae')
35
36    # fit model
37    model.fit(X, y, epochs=n_epochs, verbose=0)
38
39    # demonstrate prediction
40    x_input = test.values
41    x_input = x_input.reshape((1, n_steps_in, n_features))
42    yhat = model.predict(x_input, verbose=0)
43    return yhat

```

1.4.4 Evaluation

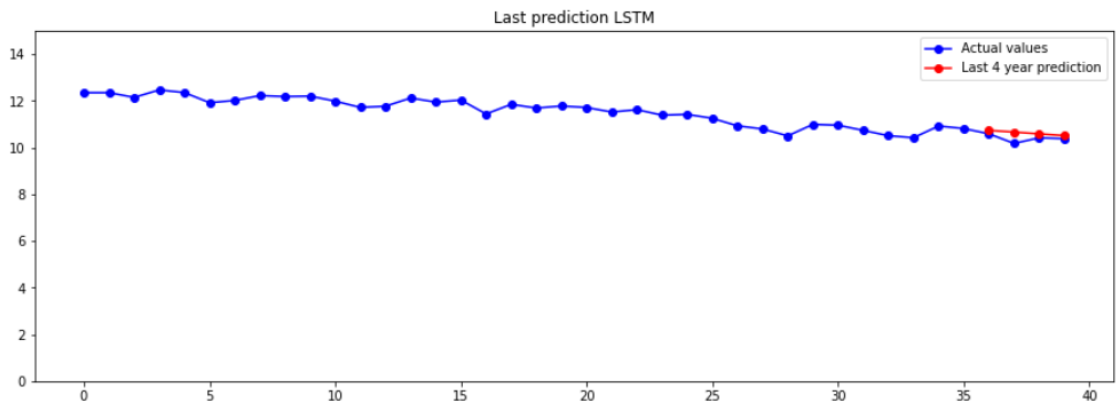
Wanneer we de resultaten van de multivariate niet-seizoensgebonden voorspellingen weergegeven op Figuur 1.29 vergelijken kunnen we stellen dat LSTM het best presteert volgens cross validation maar wel een hogere uitvoeringstijd heeft. Daarnaast zal de laatste voorspelling van het ijsoppervlak door het VARMAX model accurater zijn dan dat van het LSTM model. Deze laatste voorspelling wordt grafisch weergegeven op Figuur 1.30

1.5 Multivariate seizoensgebonden

De laatste combinatie die getest zal worden is de dataset waar een seizoensverband en een extra variabele aanwezig is.

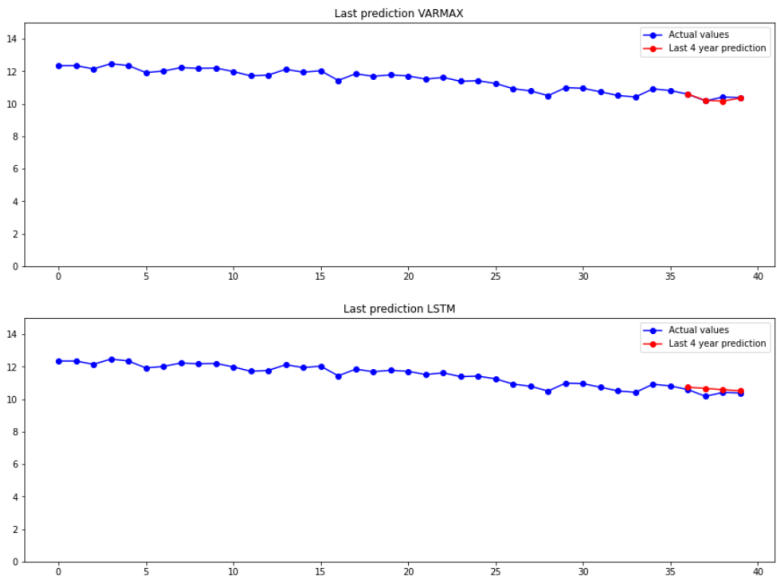
Figuur 1.28: Resultaat multivariate voorspelling LSTM

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]

Figuur 1.29: Resultaten multivariate voorspelling



Figuur 1.30: Grafische weergaven van de laatste voorspellingen van de VARMAX en LSTM modellen

