HO
GENT

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 neuraal 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

Lijst van figuren

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 #tt.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 de die 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.

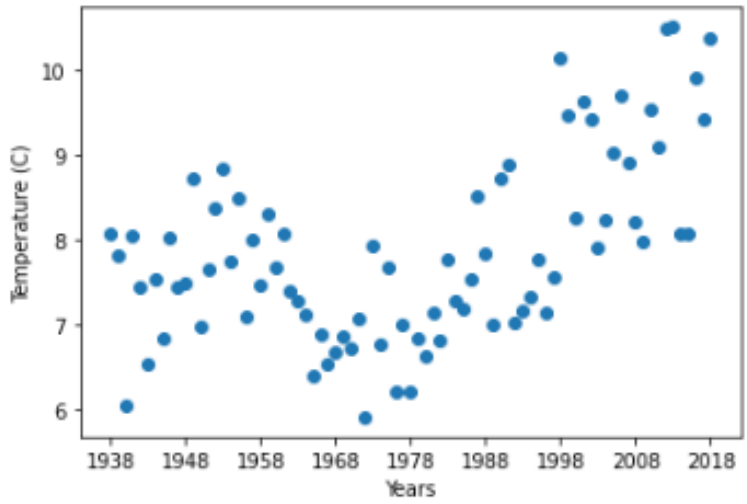
Zo wordt op lijn 2 het originele csv-bestand ingelezen deze ruwe data zal er uit zien als deze zichtbaar op figuur 1.2.
Op lijn 3 worden enkel de waarden van de eerste dag van de maand behouden in de dataset Daarna worden van lijn 4 tot lijn 9 de jaartallen aangepast zodat deze overeenkomen met

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 ijkingen 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 geindexeerd 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

```
#source: https://nsidc.org/arcticseaicenews/sea-ice-tools/
ice2 = pd.read_csv('./data/seaice2.csv')
# ice2
ice2_mean = ice2.mean()[1:-2]
# ice2_mean
ice2_mean.index = ice2_mean.index.values.astype(int)

plt.title('Yearly ice extent')
plt.scatter(ice2_mean.index,ice2_mean)
plt.xlabel('Years')
plt.ylabel('Extent')
plt.show()

# ice2['2018']
# pd.concat([ice2['2016'],ice2['2017'],ice2['2018'],ice2['2019']]).reset_index()[0]
# ice2[['2018']].append(ice2[['2019']])
ice2.rename(columns={'Unnamed: 0' : 'Month', 'Unnamed: 1' : 'Day'}, inplace = True)
ice2.drop([' ','1981-2010','Day','1978','2020'],axis=1,inplace=True)
values = ice2.values
i = 0
for row in values :
    if type(row[0]) != str :
        values[i][0] = month
    else:
        month = row[0]
    i = i +1
# ice2.columns.values
ice2_clean = pd.DataFrame(values)
ice2_clean.columns = ice2.columns.values
# ice2_clean.head(5)
ice2_monthly_mean = ice2_clean.set_index('Month').astype(float).groupby('Month',sort=False).mean()
# ice2_monthly_mean
# ice2_monthly_mean.T.stack().index.get_level_values(0)
# ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
ice2_monthly_mean_chron = ice2_monthly_mean.T.stack().reset_index(level=['Month']).drop(columns=['Month'])
# ice2.columns.size
plt.title('Monthly ice extent')
plt.plot(ice2_monthly_mean_chron.values)
plt.xticks(np.array(range(0,500,75)))
plt.xlabel('Cumulative month')
plt.ylabel('Extent')
plt.show()

# 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	N
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	N
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	N
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	N
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	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	N
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	N
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	N
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	N
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	N

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.

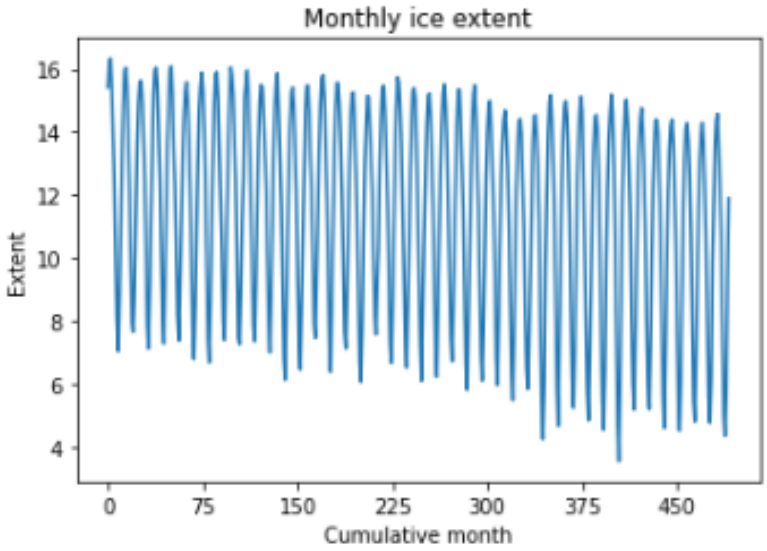
De code van lijn 8 tot lijn 12 zal ervoor zorgen dat de jaarlijkse data grafisch weergegeven wordt.
Op lijn 17 zullen de kolomnamen hernoemd worden en op lijn 18 worden de overbodige kolommen verwijderd.

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 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 figuur1.5.

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 figuur1.6.

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

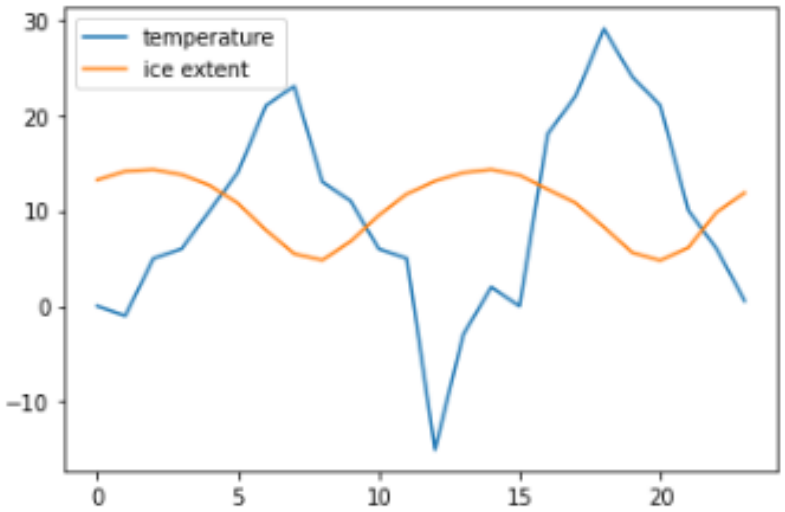
```
1  ice2_monthly_mean_chron_cut = ice2_monthly_mean_chron[:-12]
2  # ice2_monthly_mean_chron
3  # ice2_monthly_mean_chron_cut
4  # mean_temp_monthly
5  # ice2_monthly_mean_chron_cut
6  combined = mean_temp_monthly[mean_temp_monthly.index.get_level_values(0) >= 1979]
7  combined['ice_extent'] = ice2_monthly_mean_chron_cut.values
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 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 duidelijkere variabelenaam.

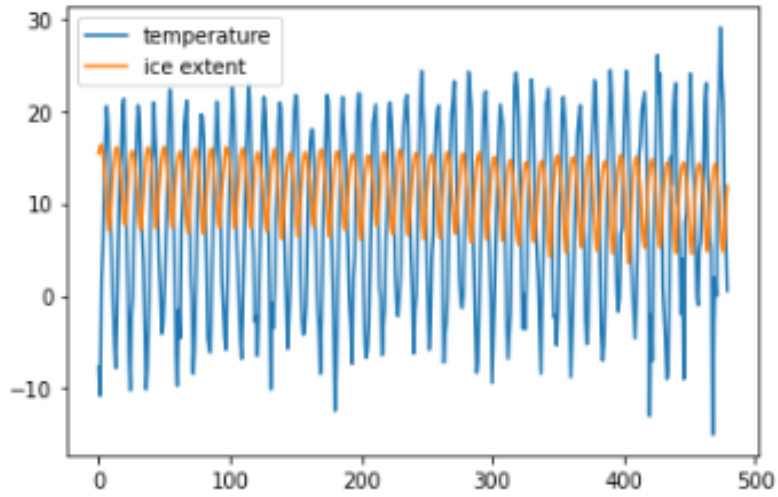
De code op lijn 13 tot 17 zorgt voor de grafische weergave van de gecombineerde dataset van de laatste 2 jaren weergegeven op figuur1.7. Hier valt op te merken dat de ijsdikte zal vergroten wanneer de temperatuur laag is. Dit is logisch aangezien het ijs zal smelten bij hogere temperaturen.

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 maandelijks gemiddelde weergegeven zal worden van 1979 tot 2018. Dit valt te beschouwen op figuur 1.7. Hierop kunnen we vaststellen 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

1.2.1 Voorbereiding

In dit deel zullen de voorspellingsmodellen voor univariate niet-seizoensgebonden data opgesteld worden. De modeltypes die bekeken zullen worden zijn ARIMA, LSTM en Prophet.
Eerst zal de data die in het vorige onderdeel voorbereid werd opgehaald worden.

Listing 4: Uitlezen van data

```
ts = pd.read_csv('./data/dataframe_yearly.csv', index_col=0, usecols=[0,2])
```

Daarna 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.

1.2.2 Stationariteit

Listing 5: Test stationarity

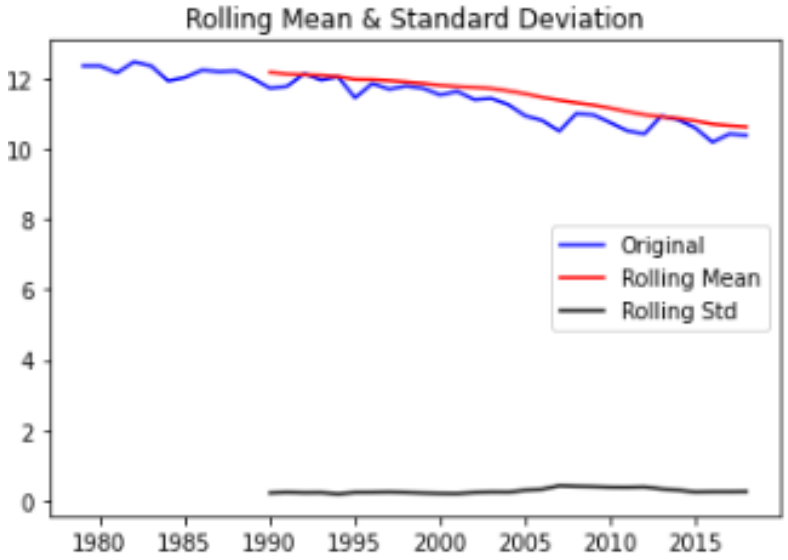
```
# define method to visualise the stationarity of a time series
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

# check stationarity of time serie
test_stationarity(ts)
```

Het resultaat van de test_stationarity methode wordt weergegeven op figuur 1.9. Daarop zien we dat de data niet stationair is maar er een dalende trend aanwezig is.
Om deze trend te neutraliseren en de data stationair te maken zullen we het random walk difference nemen en nogmaals de stationariteit testen.

Listing 6: Test stationarity bij random walk differencing

```
# take the random walk difference of the time serie
ts_diff = ts - ts.shift(1)
```

Figuur 1.9: Resultaat test stationarity




```
ts_diff = ts_diff.dropna()

# display stationarity of the newly differenced time serie
test_stationarity(ts_diff)
```

Door hiervan nogmaals de stationariteit te testen bekomen we figuur1.10. Hierop valt af te lezen dat de data nu wel stationair is.

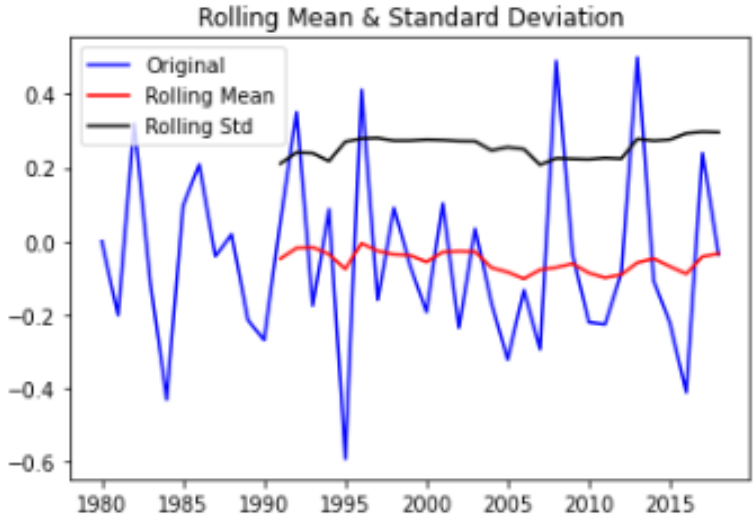
1.2.3 Cross validation

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 nemen we telkens een testgrootte van 4. 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

```
# initialize TimeSeriesSplit object
tscv = TimeSeriesSplit(n_splits = 8)

# loop trough all split time series that have a trainingsset with more than 20
↪ values
for train_index, test_index in tscv.split(ts_diff):
    if train_index.size > 20:

        # initialize cross validation train and test sets
        cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]

        # visiualize cross_validation structure for reference
```

Figuur 1.10: Resultaat test stationarity na random walk differencing




```
        print("TRAIN:", train_index.size)
        print("TEST:", test_index.size)
        print()
```

1.2.4 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 gediferentieerde voorspellingen terug omgezet worden naar voorspellingen in miljoenen vierkante kilometers. Hiervoor zijn de gedifferentieerde voorspellingen en de originele dataset nodig als invoerparameters.

Listing 8: Opstellen van algemene methodes

```
# define functions used troughout the notebook

# define function for plotting last prediction and the actual data
def full_graph(predicted_diff, title):

    # format predictions by adding NaN values in front
    predictionsArray = np.asarray(revert_diff(predicted_diff, ts))
    zerosArray = np.zeros(ts.values.size-len(predictionsArray.flatten()))
    cleanPrediction =
    ↪ pd.Series(np.concatenate((zerosArray,predictionsArray))).replace(0,np.NaN)
    cleanPrediction.index = ts.index.values

    # plot
    plt.title(title)
    plt.plot(ts, marker='o', color='blue',label='Actual values')
    plt.plot(cleanPrediction, marker='o', color='red',label='Last 4 year
    ↪ prediction')
```

```
    plt.ylim([0,15])
    plt.legend()

    plt.show()

# define function for reverting a differenced dataset
def revert_diff(predicted_diff, og_data):

    # retrieve last value
    last_value = og_data.iloc[-predicted_diff.size-1][0]

    # initialize reverted array
    predicted_actual = np.array([])

    # add each value in the differenced array with the last actual value
    for value_diff in predicted_diff:
        actual_value = last_value + value_diff
        predicted_actual = np.append(predicted_actual, actual_value)
        last_value = actual_value

    return predicted_actual
```

1.2.5 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

```
%%time
# ARIMA
from statsmodels.tsa.arima_model import ARIMA
import itertools
import warnings
import sys
from sklearn.metrics import mean_absolute_error

# Define the p, d and q parameters to take any value between 0 and 2
p = q = range(0, 5)
d = range(0,3)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# initialize variables
best_pdq = pdq
best_mean_mae = np.inf

# specify to ignore warning messages to reduce visual clutter
warnings.filterwarnings("ignore")

```

```
# loop trough all possible parameter combinations of pdq
for param in pdq:
    print(param)

    # some parametercombinations might lead to crash, so catch exceptions and continue
    try:

        # initialize the array which will contain the mean average errors
        maes = []

        # loop trough all split time series that have a trainingsset with more than 20 values
        for train_index, test_index in tscv.split(ts_diff):
            if train_index.size > 20:

                # initialize cross validation train and test sets
                cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]

                # build model
                model = ARIMA(cv_train, order=(param))

                # fit model
                model_fit = model.fit()

                # make predictions
                predictions =  model_fit.predict(start=len(cv_train), end=len(cv_train)+cv_test.size-1, dynamic=False)

                # renaming for clarity
                prediction_values = predictions.values
                true_values = cv_test.values

                # error calculation this part of the cross validation
                maes.append(mean_absolute_error(true_values, prediction_values))


        # error calculation for this parameter combination
        mean_mae = np.mean(maes)
        print('MAE: ' + str(mean_mae))

        # store parameters resulting in the lowest mean MAE
        if mean_mae < best_mean_mae:
            best_mean_mae = mean_mae
            best_maes = maes
            best_pdq = param
            best_predictions = prediction_values

    except Exception as e:
    print(e)
    continue

# logging
print()
print('Best MAE = ' + str(best_mean_mae))
```

```
print(best_pdq)
```

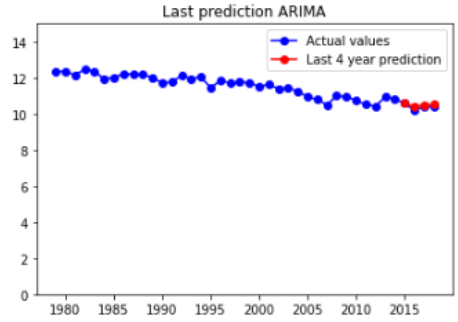
Hieruit blijkt dat de beste parametercombinatie voor een bereik van 0 tot 5 voor de p en q waarden en 0 tot 3 voor de d waarden (3,0,0) zijn. Dit zijn dan ook de parameterwaarden die gebruikt zullen worden voor de finale iteratie van ARIMA. Een hoger bereik zou tot een beter resultaat kunnen leiden maar 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

```
start_time = timeit.default_timer()

# specify to ignore warning messages
warnings.filterwarnings("ignore")

print("----")

# initialize the array which will contain the mean average errors
maes = []

# loop trough all split time series that have a trainingsset with more than 20
↪  values
for train_index, test_index in tscv.split(ts_diff):
    if train_index.size > 20:

        # initialize cross validation train and test sets
        cv_train, cv_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]

        # build model
        arima = ARIMA(cv_train, best_pdq).fit(start_ar_lags=1,disp=False)

        # make predictions
        predictions = arima.forecast(steps=4)
        prediction_values = predictions[0]
        true_values = cv_test.values

        # error calc
        maes.append(mean_absolute_error(true_values, prediction_values))

        # last actual prediction
        last_prediction_ARIMA = prediction_values

        print("I",end="")

# store results to variables
time_ARIMA = timeit.default_timer() - start_time
mae_mean = np.mean(maes)
MAE_ARIMA = mae_mean
last_MAE_ARIMA = maes[-1]

# logging
print()
print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_ARIMA)
```

Figuur 1.11: Resultaat finale iteratie ARIMA

```
----
IIII
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]
```

```
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. Hieruit kunnen we de gemiddelde MAE overheen de verschillende iteraties bij cross validation weergeven 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.

1.2.6 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 trainingsset, 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

```
from keras.layers import Dropout
# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out

        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
        break

        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

def build_model(raw_seq, n_steps_in, n_steps_out, n_features, n_neurons, dropout,
 ↪  batch_s):

    # split into samples
    X, y = split_sequence(raw_seq.values.flatten(), n_steps_in, n_steps_out)

    # reshape from [samples, timesteps] into [samples, timesteps, features]
    X = X.reshape((X.shape[0], X.shape[1], n_features))

    # define model
    model = Sequential()
    model.add(LSTM(n_neurons, activation='relu'))
    model.add(Dropout(dropout))
    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
```

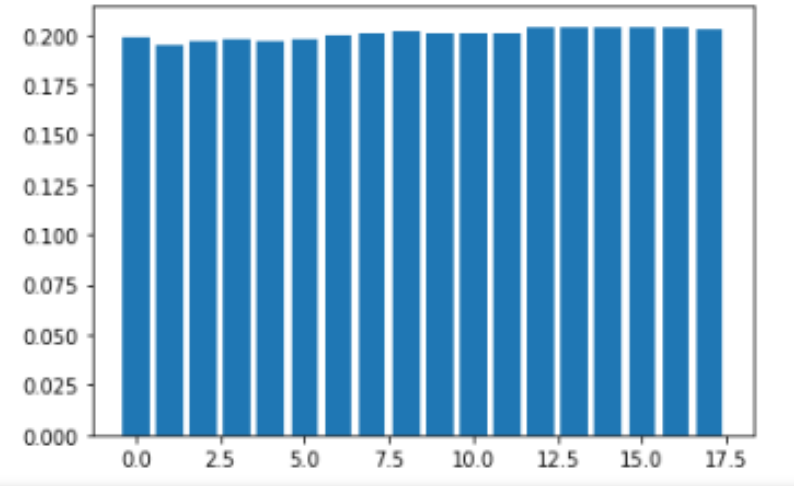
Ook bij LSTM dienen de hyperparameters geoptimaliseerd te worden. Dit gebeurt door middel van onderstaande code. Waarbij een reeks mogelijk waarden gedefinieerd wordt op lijn 15 tot 18 waarvan alle mogelijk combinaties getest worden en waarvan de best presterende combinatie bijgehouden wordt.

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:
```

```
    for dropout in dropout_array:
        for batch_size in batch_size_array:

            print("----")

            # loop trough all split time series that have a trainingsset with more than 20 values
            for train_index, test_index in tscv.split(ts_diff):
                if train_index.size > 20:

                    # initialize cross validation train and test sets
                    y_train, y_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]

                    # build model
                    lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features, n_neurons, dropout, batch_size)

                    # make predictions
                    x_input = array(y_test)
                    y_predicted = predict(x_input, lstm_model, n_features).flatten()
                    y_actual = y_test.values

                    # error calculation this part of the cross validation
                    maes.append(mean_absolute_error(y_actual, y_predicted))

                    print("I",end="")

                    # last actual prediction
                    last_prediction_LSTM = y_predicted

            # error calculation for this parameter combination
            MAE_LSTM = np.mean(maes)
            last_MAE_LSTM = maes[-1]
            global_maes.append(MAE_LSTM)

            # store parameters resulting in the lowest mean MAE
            if best_MAE > MAE_LSTM:
                best_n_neurons = n_neurons
                best_dropout = dropout
                best_batch_size = batch_size
                best_MAE = MAE_LSTM

            # log values for parameter combination
            print()
            print(n_neurons)
            print(dropout)
            print(batch_size)
            print(MAE_LSTM)
            print()

# log parameter combination with best result
print('Best:')
print('N neurons')
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 figuur1.12 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 we de waarden 1 voor het aantal neuronen gebruiken, 0 voor de beste dropout rate en 8 voor de batch size.

Listing 13: Finale iteratie LSTM

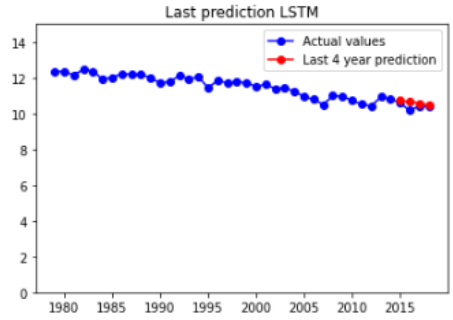
```
1  %%time
2
3  start_time = timeit.default_timer()
4
5  # Disabled tf warning because of visual clutter
6  tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
7
8
9  # constant variables
10 n_steps_in = 4
11 n_steps_out = 4
12 n_features  = 1
13 maes = []
14
15
16 # optimizable variables
```

```
n_neurons = best_n_neurons
dropout = best_dropout
batch_s = best_batch_s

print("----")

# loop trough all split time series that have a trainingsset with more than 20
↪ values
for train_index, test_index in tscv.split(ts_diff):
    if train_index.size > 20:
        # initialize cross validation train and test sets
        y_train, y_test = ts_diff.iloc[train_index], ts_diff.iloc[test_index]

        # build model
        lstm_model = build_model(y_train, n_steps_in, n_steps_out, n_features,
        ↪ n_neurons, dropout, batch_s)

        # make predictions
        x_input = array(y_test)
        y_predicted = predict(x_input, lstm_model, n_features).flatten()
        y_actual = y_test.values

        # error calc
        maes.append(mean_absolute_error(y_actual, y_predicted))

        print("I",end="")

# last actual prediction
last_prediction_LSTM = y_predicted

# store variables
time_LSTM = timeit.default_timer() - start_time
MAE_LSTM = np.mean(maes)
last_MAE_LSTM = maes[-1]

# visualisation
print()
print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_LSTM)
print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_LSTM)
print('Execution time: %.3f seconds' % time_LSTM)
full_graph(last_prediction_LSTM, 'Last prediction LSTM')
print('Mean average errors')
print(maes)
```

De bovenstaande code zal leiden tot de uitvoer die zichtbaar is op figuur**??**. En ook hier kunnen we de gemiddelde MAE overheen de verschillende iteraties bij cross validation weergeven 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

```
----
IIII
Mean MAE: 0.200 x 1 000 000 km²
MAE of last prediction: 0.215 x 1 000 000 km²
Execution time: 5.480 seconds
```




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

1.2.7 Prophet

Als laatste dient ook Prophet nog bekeken 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

```
# formatting dataframe
ts_formated_prophet = ts_diff.reset_index().rename(columns = {'Year' : 'ds',
↪  'ice_extent' : 'y'})
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 15: Code voor het bepalen van de hyperparameters

```
# Python
import itertools
import numpy as np
import pandas as pd

# define dataframe
df = ts_formated_prophet

param_grid = {
'changepoint_prior_scale': [0.001, 0.01, 0.1, 1, 2, 5, 10, 15, 20, 25],
}

# Generate all combinations of parameters
all_params = [dict(zip(param_grid.keys(), v)) for v in
↪  itertools.product(*param_grid.values())]

# initialize variables
maes = []
global_maes = []
best_MAE_prophet = np.inf

# Use cross validation to evaluate all parameters
for params in all_params:

    # loop trough all split time series that have a trainingsset with more than
    ↪  20 values
    for train_index, test_index in tscv.split(ts_formated_prophet):
        if train_index.size > 20:

            # initialize cross validation train and test sets
            train  = ts_formated_prophet.iloc[train_index]
            y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
```

```
            X_test = ts_formated_prophet.iloc[test_index][['ds']]

            # Fit model with given params
            model = Prophet(**params, weekly_seasonality=False,
            ↪  daily_seasonality=False)
            model = model.fit(train)

            # make predictions
            forecast = model.predict(X_test)
            y_pred = forecast['yhat'].values

            # last actual prediction
            last_prediction_prophet = y_pred

            # error calculation this part of the cross validation
            maes.append(mean_absolute_error(y_test, y_pred))

    # error calculation for this parameter combination
    MAE_prophet = np.mean(maes)
    last_MAE_prophet = maes[-1]
    global_maes.append(MAE_prophet)

    # logging
    print('changepoint_prior_scale: ' + str(params['changepoint_prior_scale']))

    # store parameters resulting in the lowest mean MAE
    if best_MAE_prophet > MAE_prophet:
    best_params = params
    best_MAE_prophet = MAE_prophet

# log optimal result
print('changepoint_prior_scale: ' + str(best_params['changepoint_prior_scale']))
print(best_MAE_prophet)
```

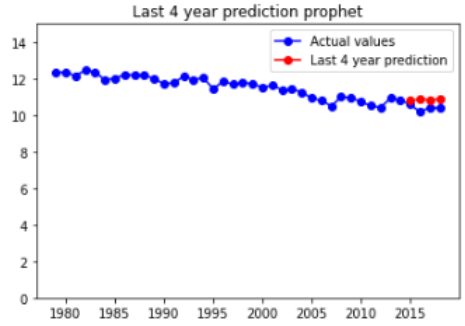
Hieruit zal blijken dat de optimale waarde voor de hyperparameter 2 is. Waneer dit ingevoegd wordt bij het uitvoeren van de onderstaande code krijgen we het resultaat dat zichtbaar is op figuur 1.14.

Listing 16: Finale iteratie Prophet

```
%%time

# Disabled tf warning because of clutter
warnings.filterwarnings("ignore") # specify to ignore warning messages

start_time = timeit.default_timer()

# initialize variables
maes = []

for train_index, test_index in tscv.split(ts_formated_prophet):
    if train_index.size > 20:

        # 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]
```

```
        train  = ts_formated_prophet.iloc[train_index]
        y_test = ts_formated_prophet.iloc[test_index][['y']].values.flatten()
        X_test = ts_formated_prophet.iloc[test_index][['ds']]

        # build model
        model = Prophet(**best_params, weekly_seasonality=False,
        ↪  daily_seasonality=False)
        model.fit(train)

        # make predictions
        forecast = model.predict(X_test)
        y_pred = forecast['yhat'].values

        # error calc
        maes.append(mean_absolute_error(y_test, y_pred))

        # last actual prediction
        last_prediction_prophet = y_pred


# store results
time_Prophet = timeit.default_timer() - start_time
MAE_Prophet = np.mean(maes)
last_MAE_Prophet = maes[-1]

# visualize results
print()
print('Mean MAE: %.3f x 1 000 000 km\u00b2' % MAE_Prophet)
print('MAE of last prediction: %.3f x 1 000 000 km\u00b2' % last_MAE_Prophet)
print('Execution time: %.3f seconds' % time_Prophet)
full_graph(last_prediction_prophet, "Last 4 year prediction prophet")
print('Mean average errors')
print(maes)
```

Figuur 1.15: Resultaat van de univariate non-seasonal analyse

	Mean MAE (x 1 000 000 km²)	Execution time (s)	Last MAE (x 1 000 000 km²)
ARIMA	0.171	0.360	0.103
LSTM	0.200	5.480	0.215
Prophet	0.252	8.891	0.308

1.2.8 Evaluatie

Wanneer we deze resultaten combineren komen we tot de tabel die zichtbaar is op figuur 1.15. Om dit resultaat grafisch te schetsen worden op figuur1.16 de laatste voorspellingen voor elk modeltype weergegeven.
Hieruit kunnen we dus stellen dat ARIMA het beste gemiddelde resultaat zal halen bij cross validation aangezien het hier een fout van 0.171 x 1 000 000 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 x 1 000 000 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 x 1 000 000 km^2 en een uitvoeringstijd van 8.891 seconden.

Listing 17: Code voor het weergeven van de resultaten

```
# formatting
results = [[MAE_ARIMA,time_ARIMA,last_MAE_ARIMA],[MAE_LSTM,time_LSTM,last_MAE_LSTM],[MAE_Prophet,time_Pr

# display results
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

```
# visualize results of last prediction
full_graph(last_prediction_ARIMA, "Last prediction ARIMA")
full_graph(last_prediction_LSTM, "Last prediction LSTM")
full_graph(last_prediction_prophet, "Last prediction Prophet")
```

1.3 Univariate seizoensgebonden

1.4 Multivariate niet-seizoensgebonden

1.5 Multivariate seizoensgebonden

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