# P3

April 30, 2021

## 1 MA8701 Prject II:

https://github.com/metno/havvarsel-data-driven-pred/blob/main/dataset1.csv

God morgen! As promised here is our data set  I wanted to include one more covariate (a oceanographic one) but I failed during the weekend - I will try it after the lecture again. BUT as soon as you start with data analysis please let know since I dont want to interrupt ongoing work with changing the data set afterwards!

As discussed the data set consists of the following timeseries: - water_temp (the response) - measurements of 7 atmospheric values… the naming convention is "station_id"(where the measurement is done) + "value"(what got measured) + "number"(0=measurement in 2m height, 1=measurement in 10 height above ground)

We have - water_temp in degC - measured at 7,10,14, 16 every day during swimming season

- air_temperature in degC
- wind_speed in m/s
- cloud_area_fraction in 0,…,8 (0 no clouds, 8 fully cloudy)
- mean(solar irradiance) in w/m2 - a quantity for the intensity of the sun
- sum(duration of sunshine) in min - sunshine in the last hour
- mean(realtive humidity) in %
- mean(downwelling shortwave flux) in W/m2 - like irradiance but only for UV radiation (normally criterion for getting sun burned, but I thought that it could maybe get interesting)

I hope thats something we can work with :) In case of questions just ask me since I was fighting a lot with those weather data bases and know somehow what is going on with weather measurements in Norway (not forecasts though haha)

#### 1.0.1 Water temperature is a time series data, thus, it is shifted one day behind to take considerations of the influence from the previous days. But we still assume the independence between each samples.

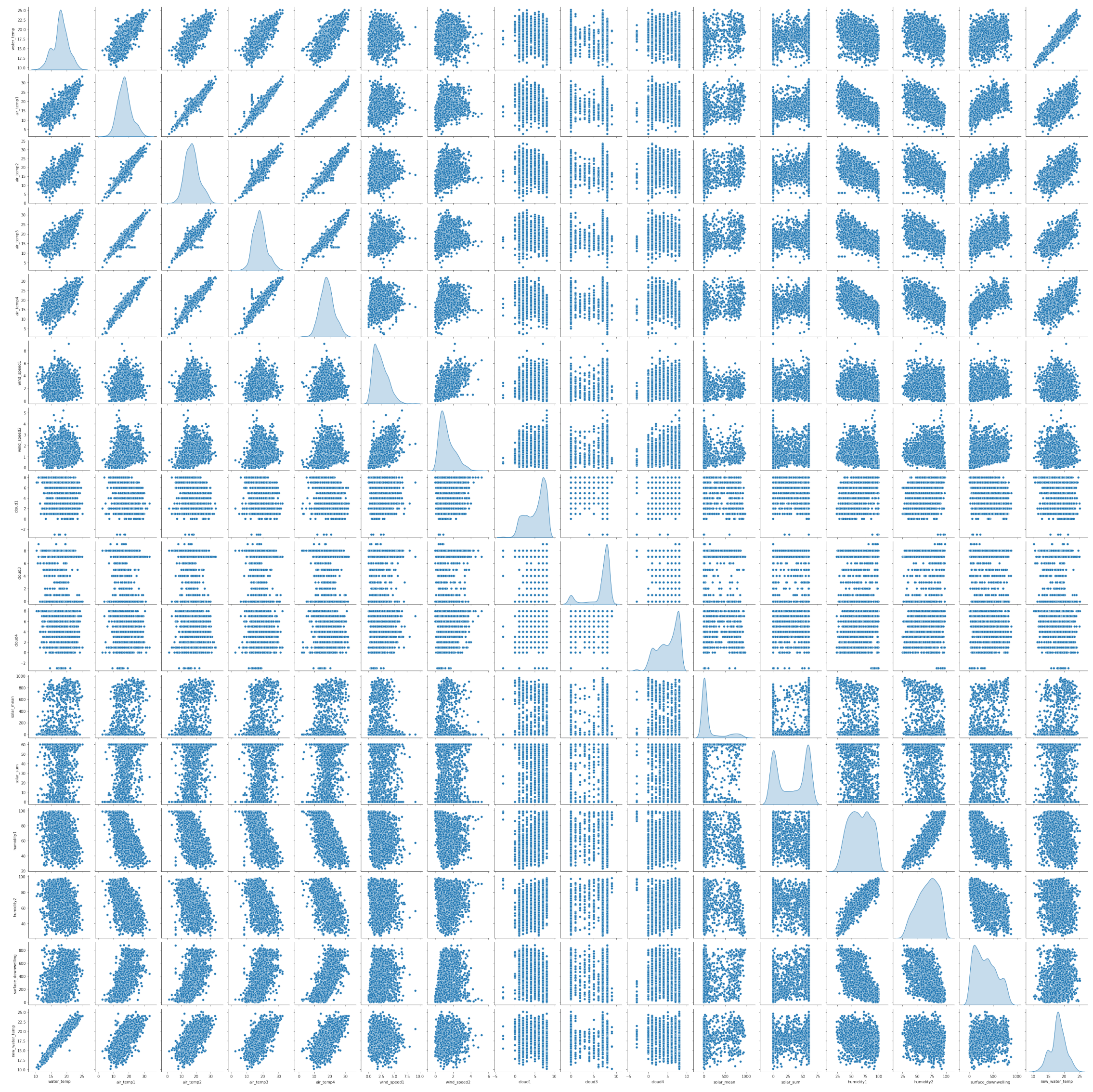
[138]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
# Make numpy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
import tensorflow as tf
```

```python
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
# print(tf.__version__)
```

[139]:
```python
column_names = ['water_temp', 'air_temp1', 'air_temp2', 'air_temp3',␣
 ↪'air_temp4',
                'wind_speed1', 'wind_speed2', 'wind_speed3', 'cloud1', 'cloud2',
                'cloud3', 'cloud4', 'solar_mean', 'solar_sum', 'humidity1',␣
 ↪'humidity2',
                'surface_downwelling']
# raw_dataset = pd.read_csv('dataset1.csv', names = column_names, header =␣
 ↪None, index_col = 'time')
raw_dataset = pd.read_csv('dataset1.csv', header = None, names = column_names)
raw_dataset = raw_dataset.iloc[1:, :]
shifted_water_temp = raw_dataset['water_temp']
shifted_water_temp = shifted_water_temp.shift(1)
new_dataset = raw_dataset
new_dataset['new_water_temp'] = shifted_water_temp
new_dataset = new_dataset.iloc[1:, :]
```

[140]:
```python
dataset = new_dataset.copy()
# print(dataset.isna().sum())
del dataset["wind_speed3"] # remove empty data
del dataset["cloud2"]
dataset.tail()
for i in range(len(dataset.columns)):
    dataset.iloc[:,i] = pd.to_numeric(dataset.iloc[:,i], downcast = "float")
# Split the data set
train_dataset = dataset.iloc[:1856, :] # split the data in 2016-2019 into␣
 ↪training data
test_dataset = dataset.drop(train_dataset.index) # use the data in 2020 as the␣
 ↪test data
```

[141]:
```python
sns.pairplot(train_dataset, vars = train_dataset.columns[:], diag_kind='kde')
plt.show()
```



## 1.1 ### From the correlation plot, one can tell that the air temperature and water temperature tend to have very high correlations regardless of where air temperatures are collected, which makes sense since the air temperature will mostly likely be pretty similar when they are in a relatively close regions.

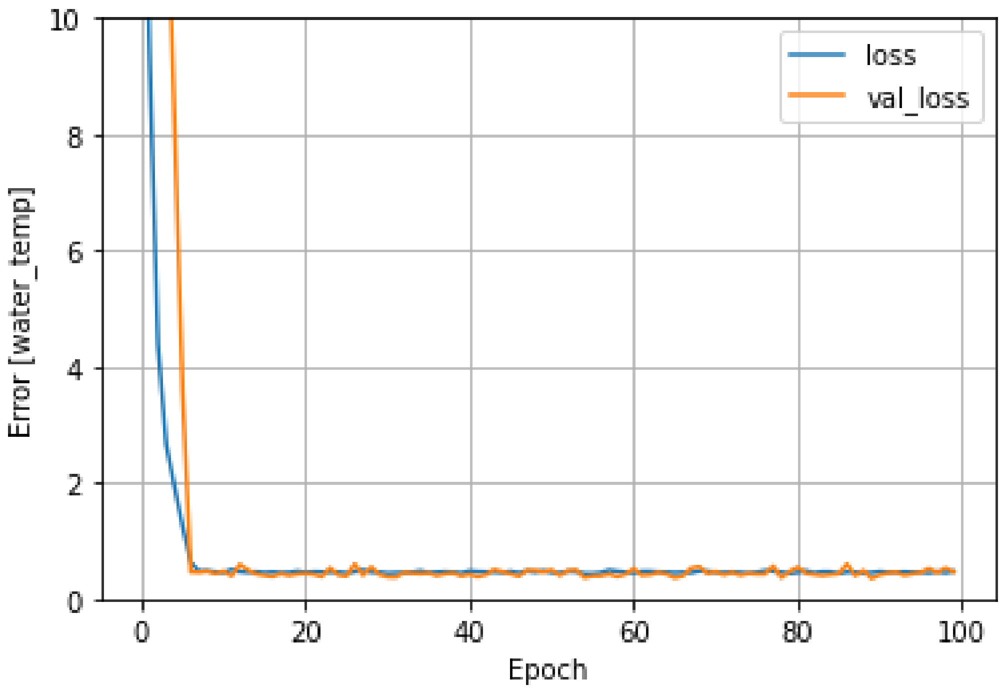
### 1.1.1 For the exploration purpose, only single neural network is going to be trained and tested.

[142]:
```
train_features = train_dataset.copy()
test_features = test_dataset.copy()

train_labels = train_features.pop('new_water_temp')
```

```
test_labels = test_features.pop('new_water_temp')
```

[143]:
```
normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(train_features))


def plot_loss(history):
  plt.plot(history.history['loss'], label='loss')
  plt.plot(history.history['val_loss'], label='val_loss')
  plt.ylim([0, 10])
  plt.xlabel('Epoch')
  plt.ylabel('Error [water_temp]')
  plt.legend()
  plt.grid(True)
```

[144]:
```
linear_model = tf.keras.Sequential([
    normalizer,
    layers.Dense(units=1)
])

linear_model.predict(train_features)
linear_model.compile(
    optimizer=tf.optimizers.Adam(learning_rate=0.1),
    loss='mean_absolute_error')

history = linear_model.fit(
    train_features, train_labels,
    epochs=100,
    # suppress logging
    verbose=0,
    # Calculate validation results on 20% of the training data
    validation_split = 0.2)

plot_loss(history)
```

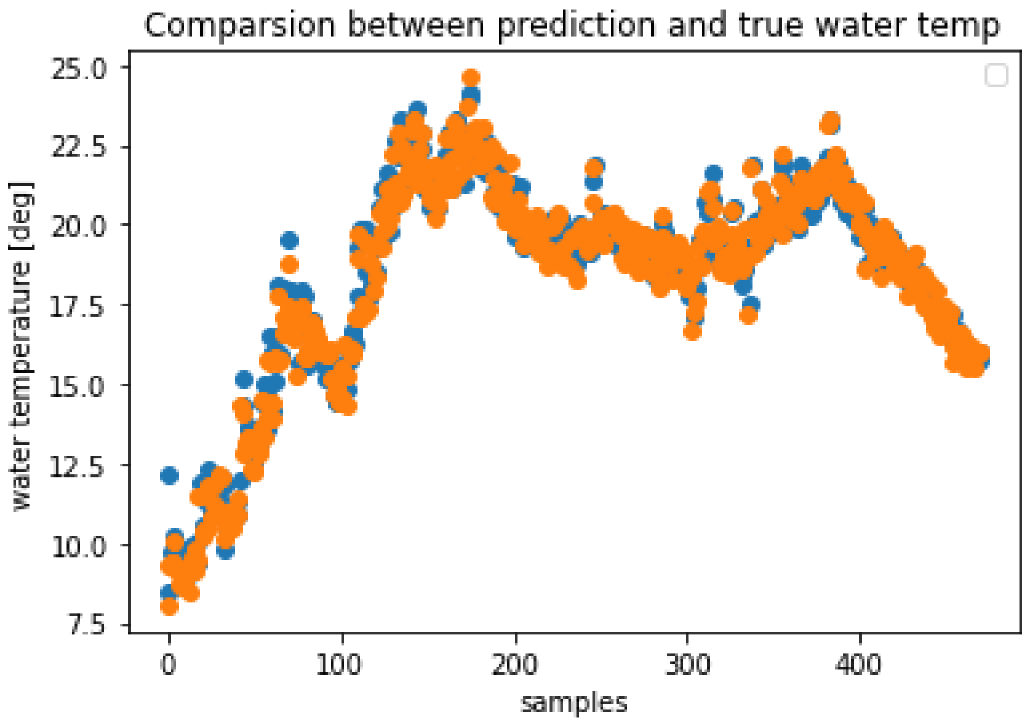


### 1.1.2 From the training history plot, one can tell that the training is pretty fast to converge to the required criterion. This is one the of the benefit for using the simple model.

[145]:
```
test_results = linear_model.evaluate(
    test_features, test_labels, verbose=0)
test_predictions = linear_model.predict(test_features).flatten()
# plot the estimation
x = np.arange(len(test_labels))
plt.scatter(x, test_labels)
plt.scatter(x, test_predictions)
plt.title("Comparsion between prediction and true water temp")
plt.legend('true', 'prediction')
plt.ylabel("water temperature [deg]")
plt.xlabel("samples")
plt.show()
```

```
<ipython-input-145-af33c22b0af1>:9: UserWarning: Legend does not support 't'
instances.
A proxy artist may be used instead.
See: https://matplotlib.org/users/legend_guide.html#creating-artists-
specifically-for-adding-to-the-legend-aka-proxy-artists
  plt.legend('true', 'prediction')
<ipython-input-145-af33c22b0af1>:9: UserWarning: Legend does not support 'r'
```

```
instances.
A proxy artist may be used instead.
See: https://matplotlib.org/users/legend_guide.html#creating-artists-
specifically-for-adding-to-the-legend-aka-proxy-artists
  plt.legend('true', 'prediction')
<ipython-input-145-af33c22b0af1>:9: UserWarning: Legend does not support 'u'
instances.
A proxy artist may be used instead.
See: https://matplotlib.org/users/legend_guide.html#creating-artists-
specifically-for-adding-to-the-legend-aka-proxy-artists
  plt.legend('true', 'prediction')
<ipython-input-145-af33c22b0af1>:9: UserWarning: Legend does not support 'e'
instances.
A proxy artist may be used instead.
See: https://matplotlib.org/users/legend_guide.html#creating-artists-
specifically-for-adding-to-the-legend-aka-proxy-artists
  plt.legend('true', 'prediction')
```



## 2 Conclusions from Deep learning part

Air temperature measurements seem having high correlations, so do the humidity. By using single layer and multiple layers neural network, it predicts the water temperature with relatively low error. Also, simple model converges fast when it comes to training.