Report on TimeSeries DeepLearning analyses on a weather dataset

by Giuseppe Insana, March 2022

The dataset

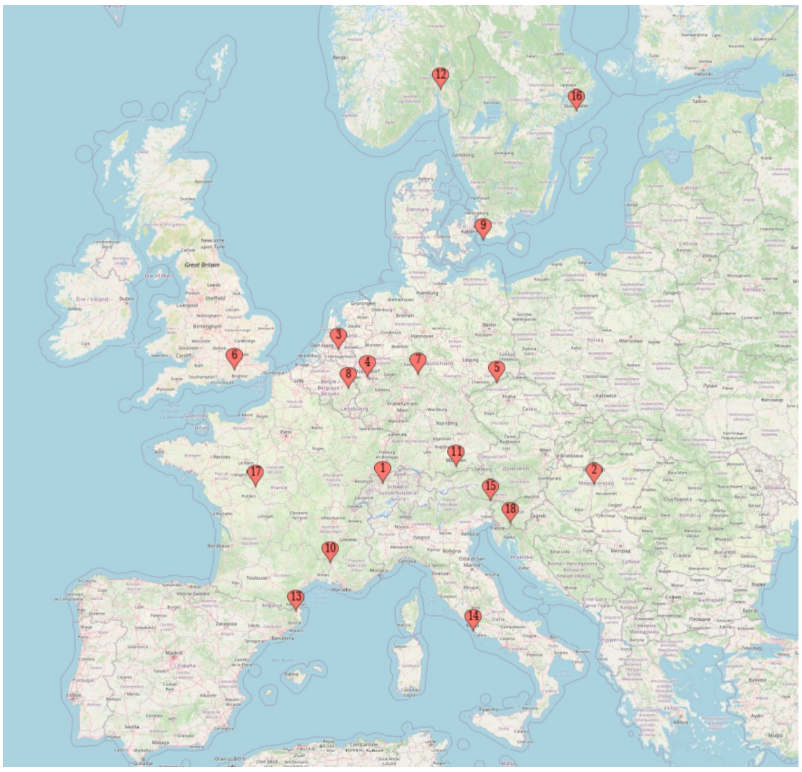
I decided to work on a weather dataset, chiefly because I never worked on this kind of data before (having mostly dealt with biological and linguistic analyses in the past).

The weather dataset analysed in this report has been created by Huber Florian from ECA&D data (see section on Data origin at the end for references).

It contains daily weather observations from 18 different European weather stations through the years 2000 to 2010.

The description of the data set says that the minimal set of variables 'mean temperature', 'max temperature' and 'min temperature' are available for all locations. An additional number of measured variables ('cloud_cover', 'wind_speed', 'wind_gust', 'humidity', 'pressure', 'global_radiation', 'precipitation', 'sunshine') are provided, but not for all the locations.

The following map shows the locations which are included in the dataset:



1	Basel (CH)
2	Budapest (HU)
3	De Bilt (NL)
4	Düsseldorf (DE)
5	Dresden (DE)
6	Heathrow (UK)
7	Kassel (DE)
8	Maastricht (NL)
9	Malmo (SE)
10	Montélimar (FR)
11	München (DE)
12	Oslo (NO)
13	Perpignan (FR)
14	Roma (IT)
15	Sonnblick (AT)
16	Stockholm (SE)
17	Tours (FR)
18	Ljubljana (SI)

Objective of the analysis

We will try to predict tomorrow's weather for one location based on today's weather measures across all locations. In this report we will present some time series specific EDA and then focus on trying to improve prediction accuracy using time series models. In particular we will try to predict whether tomorrow is going to be sunny in the city of Basel, with how many hours of sunshine, based on the observations from a certain consecutive sequence of days.

Data exploration

A comprehensive exploratory data analysis has been previously conducted and presented in another report (please check the separate document, file named **EDAcourseproject_report_Giuseppe_Insana.pdf**). We will here briefly summarise the findings of that report to give a description of the data we will process and to show the rationale behind the actions we will take. We will then perform some Time Series specific EDA.

We have analysed the types and amount of the available data, checking ranges and distributions of all observations.

Data attributes

The original data is loaded into a pandas dataframe which has 3654 rows (one per each day) and 165 columns:

- DATE, with integer values from 20000101 to 20100101, corresponding to interval from Jan 1st 2000 to Jan 1st 2010
- MONTH, integer 1 to 12
- and another 163 columns for the weather measurements at the different locations.

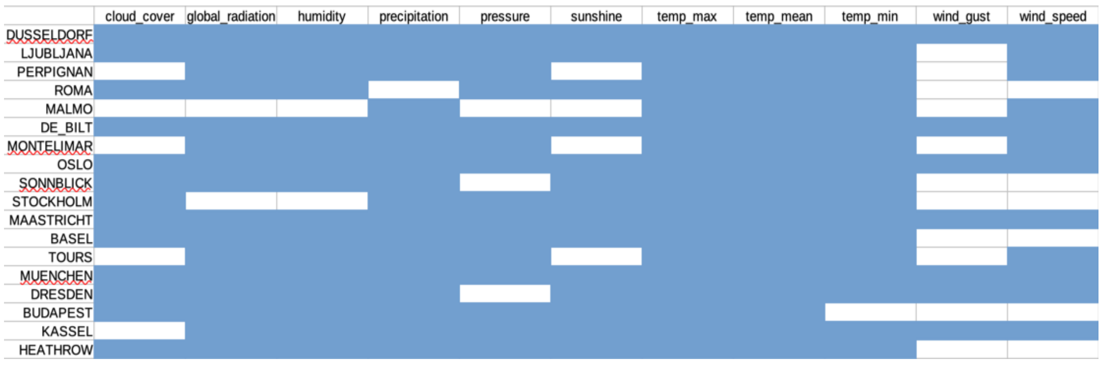
The measurements are labelled as LOCATION_*measure* (e.g. BASEL_cloud_cover, BASEL_pressure, OSLO_precipitation...) with the measured variables being: **cloud_cover, global_radiation, humidity, precipitation, pressure, sunshine, temp_max, temp_mean, temp_min, wind_gust, wind_speed**

All the measurements are loaded as floating point numbers, with the exception of cloud_cover, loaded as integer.

The **physical units** for the variables are described as follows:

cloud_cover in oktas; **wind_speed** and **wind_gust** in m/s; **humidity** in fraction of 100%; **pressure** in 1000 hPa, **global_radiation** in 100 W/m^2; **precipitation** in 10 mm; **sunshine** in 1 Hours; **mean max** and **min temperature** in Celsius degrees.

The following table shows which measures are available for which location:



	cloud_cover	global_radiation	humidity	precipitation	pressure	sunshine	temp_max	temp_mean	temp_min	wind_gust	wind_speed
DUSSELDORF	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
LJUBLJANA	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		🟦
PERPIGNAN		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
ROMA	🟦	🟦	🟦		🟦	🟦	🟦	🟦	🟦		
MALMO				🟦			🟦	🟦	🟦		🟦
DE_BILT	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
MONTELIMAR		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
OSLO	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
SONNBLICK	🟦	🟦	🟦	🟦		🟦	🟦	🟦	🟦		
STOCKHOLM	🟦			🟦	🟦	🟦	🟦	🟦	🟦		
MAASTRICHT	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
BASEL	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
TOURS		🟦	🟦	🟦	🟦		🟦	🟦	🟦		🟦
MUENCHEN	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
DRESDEN	🟦	🟦	🟦	🟦		🟦	🟦	🟦	🟦	🟦	🟦
BUDAPEST	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦			
KASSEL		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
HEATHROW	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		

This could obviously present problems of **unbalanced data**, in particular when extending the model to the prediction for different cities.

Null and Out-Of-Range values

The dataset does not contain missing values per se but there are several out-of-range values that can be considered as Null/Missing/Invalid observations:

cloud_cover measures should vary from 0 (sky completely clear) to 8 (sky completely clouded) oktas. But the data contains two data points (both for Stockholm, 20080724 and 20090625) with a value of -99 and one (again Stockholm, 20031108) with a value of 9

pressure: the data contains three entries (again for Stockholm, 20071008, 20000124, 20070603) with value of -0.099 and one entry (Tours, 20081230) with value of 0.0003. These out of range values can again be considered invalid and a decision should be taken for them akin to those mentioned before for cloud_cover.

sunshine: the data contains 29 negative values for hours of sunshine (again for the Stockholm location) which should be treated as invalid/null and dealt appropriately (as mentioned for cloud_cover and pressure). For the location of Oslo there are 24 measurements with more than 18 hours of sunshine, with 20 of them being 24h. While the northern latitude make very long daylight possible, this is for almost 18 hours in midsummer, while these huge values are from Nov-Dec 2006. We must hence treat these as wrong invalid data as well.

Distribution of values

After imputing the out of range values (shown below in the section on Data cleaning and feature engineering), the range, mean and standard deviation for all measures across all locations are the following:

measure	**mean**	**std**	**range**	
cloud_cover:	5.14	2.33	0.00 .. 8.00	*(okta)*
global_radiation:	1.37	0.95	0.01 .. 4.42	*(i.e. 1 to 442 W/m2)*
humidity:	0.75	0.14	0.10 .. 1.00	*(i.e. 1% to 100%)*
precipitation:	0.23	0.58	0.00 .. 16.04	*(i.e. 0 to 160.4 mm)*
pressure:	1.02	0.01	0.96 .. 1.05	*(i.e. 959 to 1016 hPa)*
sunshine:	5	4.41	0.00 .. 17.80	*(hours)*
temp_max:	14.5	9.58	-24.70 .. 41.10	*(°C)*
temp_mean:	10.39	8.41	-26.60 .. 33.10	*(°C)*
temp_min:	6.33	7.58	-30.30 .. 26.30	*(°C)*
wind_gust:	10.06	3.88	1.50 .. 41.00	*(m/s)*
wind_speed:	3.33	1.89	0.00 .. 16.30	*(m/s)*

The dataset was thoroughly explored visually by way of plots, to see the actual distribution of values and to gather insights, plotting the measured features as a whole or grouped spatially or temporally.

Major findings:

- strong dependence of weather measures by month and by location (obviously) for both ranges, mean and variance; for example pressure has smaller variance in summer months compared to winter months;
- the seasonal component appears to be responsible for multimodality in certain features;

- there are also year to year variations, with for example some years being on average warmer or colder; no overall trend was observed but this is probably due to the scale of the dataset (only ten years period);
- precipitation and wind_speed are the most skewed measures (more than 0.75 skew value) across all locations and for some locations humidity and cloud_cover as well. The most strongly skewed features are:

DRESDEN_precipitation	13.077
PERPIGNAN_precipitation	10.781
MONTELIMAR_precipitation	7.479
BUDAPEST_precipitation	5.662
MALMO_precipitation	5.337
MUENCHEN_precipitation	5.206
BASEL_precipitation	4.529
STOCKHOLM_precipitation	4.481
TOURS_precipitation	4.233
LJUBLJANA_precipitation	3.828

Correlations

The correlation between features was analysed, both for measures in a single location or across different locations.

The major insights were:

- related variables have (as expected) very high correlation:
    - temp_mean with temp_min and temp_max
    - wind speed and wind gust
- global radiation has high positive correlation with sunshine
- global radiation and sunshine correlate negatively with humidity
- precipitation appears to have extreme values concentrated where pressure has average values
- locations nearby have measures more highly correlated compared to locations geographically distant

For a complete analysis of correlated measures and how this was used to reconstruct the topology of the weather locations, please check the separate document, file named **unsupervised_courseproject_report_Giuseppe_Insana.pdf**).

Data cleaning and feature engineering

Out of range values

To clean the dataset we first dealt with the **out-of-range values** identified for the cloud_cover, pressure and sunshine measures.

The number of these invalid values appear in 1.64% of the total rows but as they only affect one location at a time they constitute only 0.01% of the total values. In the previous report we **recommended to not drop** the whole days' measurements but **instead to impute** the invalid values.

To do so we wrote code which gathers the values for the involved measure and location on the 5 days before and 5 days after the date of the out-of-range value. These values are then averaged and the average is used to impute the invalid value. When the invalid values appeared in succession the strategy is modified to use the window of 10 days (5 before and 5 after) but for the year before.

For example, the code identifies the above mentioned out-of-range values for pressure:

idx	DATE	STOCKHOLM_pressure	TOURS_pressure
23	20000124	-0.0990	1.0234
2710	20070603	-0.0990	1.0205
2837	20071008	-0.0990	1.0242
3286	20081230	1.0328	0.0003

and proceeded to impute them as following:

```
 ** Imputing 3 value(s) for column STOCKHOLM_pressure
 idx23: from -0.099 to 1.00449 using mean data from range 18-29
 idx2710: from -0.099 to 1.02079 using mean data from range 2705-2716
 idx2837: from -0.099 to 1.0203099999999998 using mean data from range 2832-2843

** Imputing 1 value(s) for column TOURS_pressure
idx3286: from 0.0003 to 1.02485 using mean data from range 3281-3292
```

Outliers

The **main outliers** identified in the EDA report were:

- the very low values for humidity, with majority of outliers from Sonnblick and (in much lower proportion) Perpignan
- the precipitation extremes
- the highest recorded wind_speed measures (which are almost all from Perpignan location)

These could represent extreme weather conditions, which may or may not hinder the prediction abilities of the ML models.

Similarly to what said for out-of-range values, the outliers account for a negligible portion of the dataset but non-negligible number of rows. Thus it would be best to impute the single values (using the same strategy as above, averaging over a window of days) rather than excluding the whole rows.

As previously observed during regression and classification work on this dataset, imputing these outliers made negligible difference to the results of the trained models.

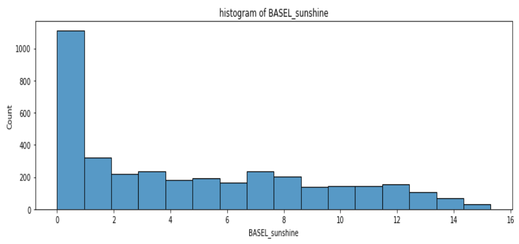
Feature engineering

The original **dataset does not contain categorical data** which would need to be converted (e.g. by one-hot encoding).

The DATE information is dropped from the dataset, leaving 164 columns (MONTH and weather measures across locations).

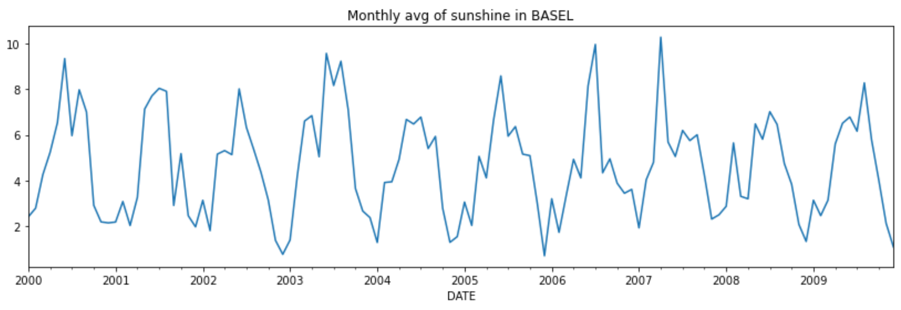
Time Series specific EDA

There are a series of analyses which are specific to time series and have not been shown on previous reports.

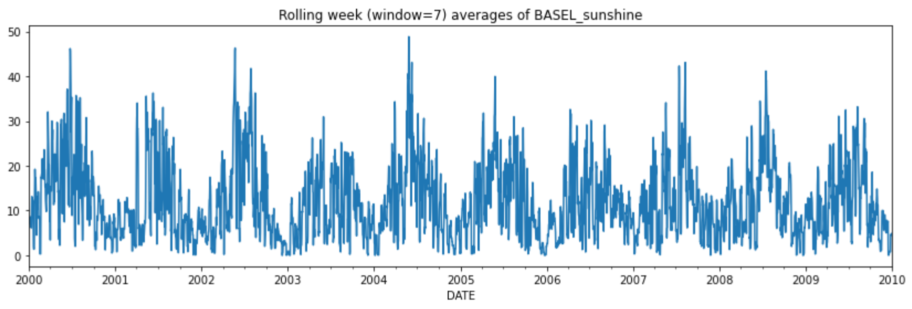


We already know that the distribution of the sunshine measure is very skewed, but we can better understand how it changes across the year and along the ten years of measures.

We can start by **aggregating** the target variable (for which we have daily measures) by week, month or year. For example we can plot the monthly averages across the ten years of data:

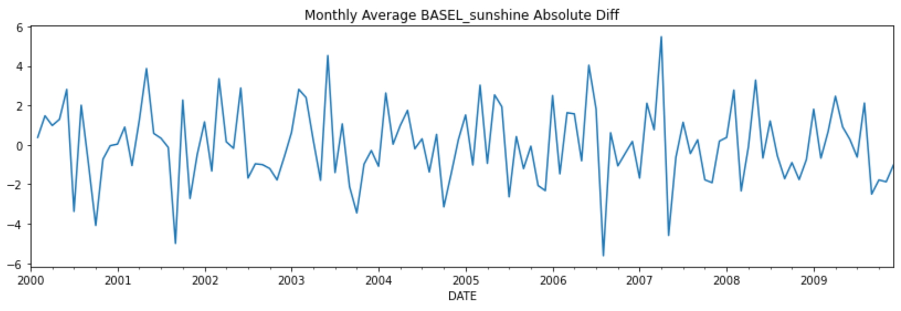


Alternatively we can use **rolling averages**, for example for a week of measures:

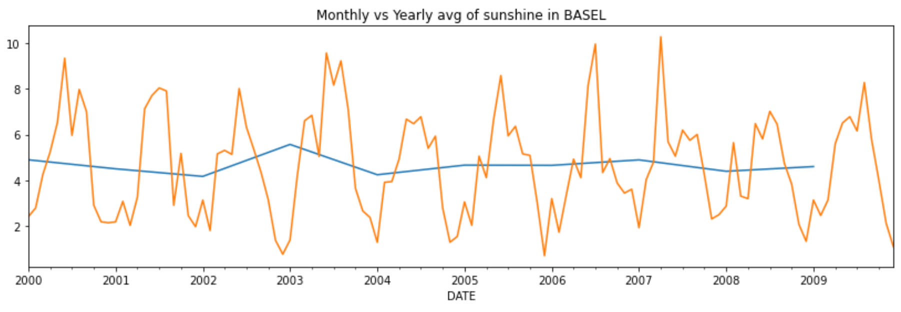


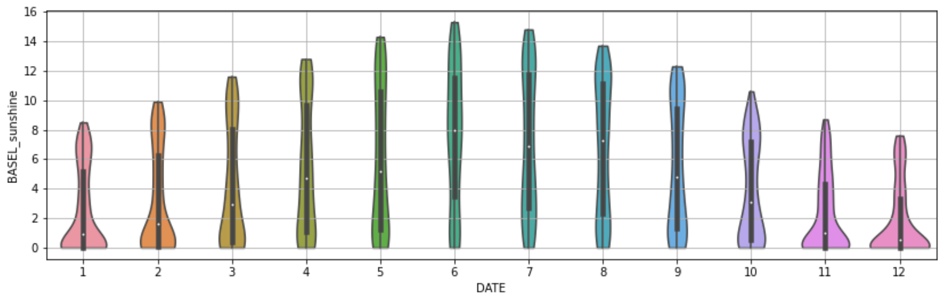
Both reveal at a glance the seasonality of the data.

The following plot shows the **differencing** between average of the target measure for each month:

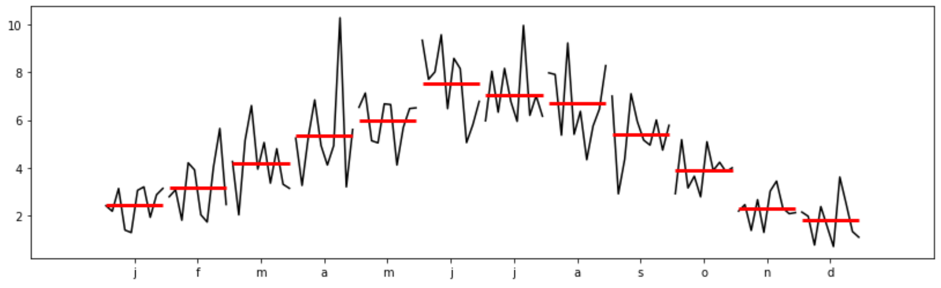


Aggregating by year shows that while there are sometime "sunnier years", the yearly average is quite constant:



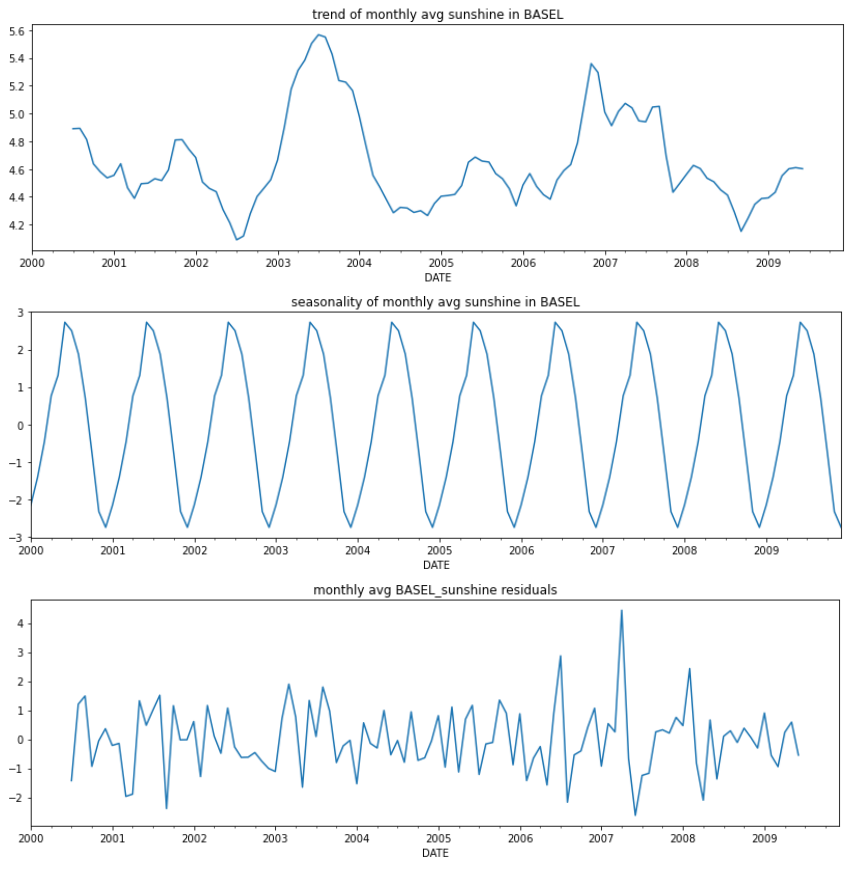


The variance is not constant across the months...

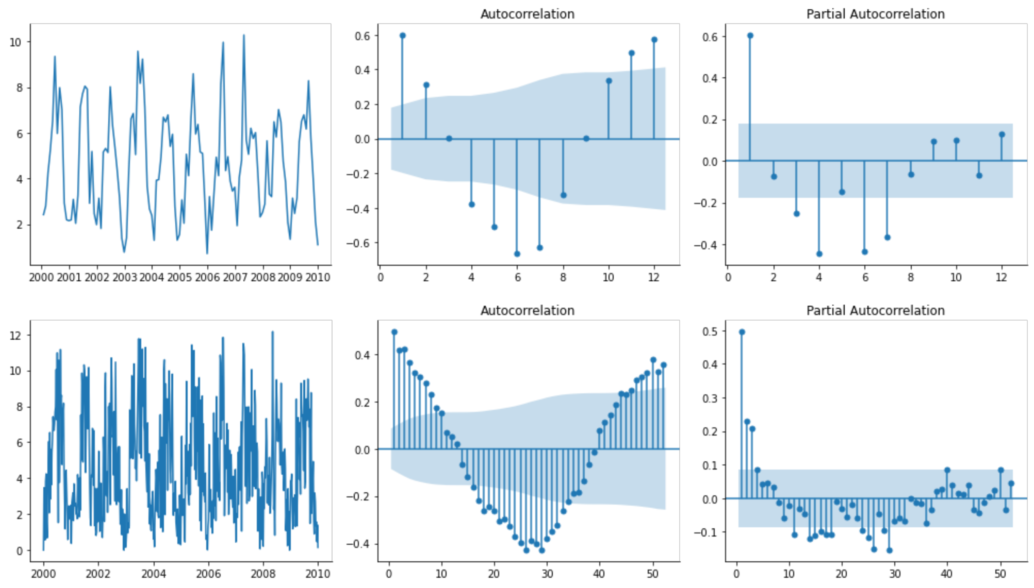


.. and monthly average sunshine is quite different year by year

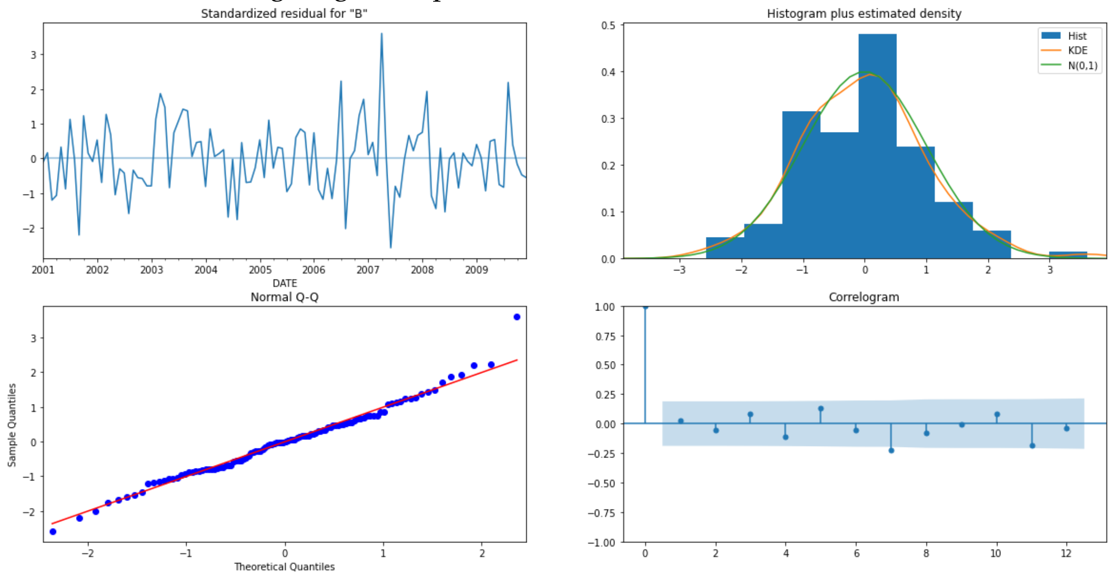
We can **decompose** the time series for the target measure into trend, seasonality and residuals, using *seasonal_decompose* from the *statsmodel* library:



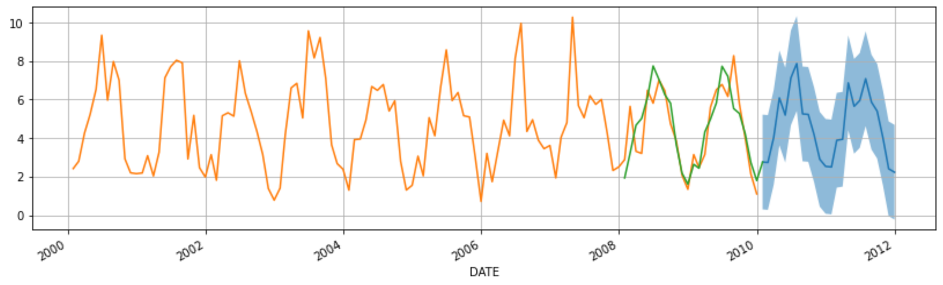
The data clearly has trends, seasonality and **autocorrelation**. As for the latter, these are the correlation plots for monthly and weekly averages:



Enlisting the help of *auto_arima* from the *pmdarima* library, the best parameters for a **Seasonal ARIMA** model were found as ARIMA(0,0,0)(4,1,0)[12], i.e. a SARIMAX(4, 1, 0, 12) model, with an AIC of 384. Testing the same parameters with *SARIMAX* from *statsmodel* library produced a similar AIC of 386 and the following diagnostic plot:



Using either of the two fitted models (from *pmdarima* or *statsmodel*) we could start making the first approximate forecasts:



orange: real data; green: SARIMA fit; blue: forecast for next two years with 95% confidence interval

Models training

Procedure

The input data has been scaled with *StandardScaler* and the most skewed features (with skew value more than 0.75) have been transformed using the transformer *log1p* (using *sklearn*'s *ColumnTransformer*). This has been shown in previous reports to always provide better performance to the models.

We set aside a *consecutive* sample of 20% of the entire dataset to be used as **test** for evaluation. The remaining 80% is the data on which we **train** each model on. We have repeated the evaluation twice. First by holding out the last 20% of the data (the last two years) as test set (training on the first eight years) and then by holding out the first 20% of the data (the first two years of data) as test and training on the last eight years of data. The scores presented are the average of the scores obtained with these two different splits, as cross validation.

In order to use historical data to forecast the future, input data was reshaped so as to present input and output pairs to the deep learning models. This was done using ***keras*** *TimeseriesGenerator* preprocessing utility. This allows to specify a number of sequential data vectors (subsequent days of measures, in our case) that are fed as input sequence to the deep learning models.

Different sequence_length amounts have been tried for different models, to explore the influence of this key parameter on the accuracy of predictions. For example a sequence_length of 7 means that the models was given 7 days of weather observations and was tasked with the prediction of the target measure for the 8th day.

The models were built using ***keras*** *Sequential* API.

The input layer was set to accept a matrix of shape (sequence_length, features_number) and the output layer was a single dense layer of size 1. Different model architectures were examined by training, comparing and presenting the results. Unless differently specified, the activation function for the layers was set as '*relu*' and no activation was added to the last output layer.

The maximum training was defined at 700 epochs, but often early stopping would terminate training much earlier, as will be explained below.

Batch size was set as 365: presenting one full year of data measures for the learning. '*adam*' was used as the optimizer and '*mae*' (mean absolute error) as loss function. The metrics of *mse* and *mae* were recorded for train and test sets during fitting. Models were all trained without shuffling the training data but for the most promising models we also tried re-training shuffling the order of the sequences presented (not, obviously, the order of the data in each sequence) to see whether it would prove helpful. In almost all cases, shuffling did not improve the final performance.

For **scoring**, two metrics have been used: the mean squared error (MSE) and the R^2 score (coefficient of determination).

With $\hat{y}_i$ being the predicted value of the ith sample, and y_i as the corresponding true value, then the mean squared error (MSE) estimated over n samples is defined as:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$$

while the R^2 score is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.

The train data metrics MSE and R^2 (indicated respectively as columns **trainMSE** and **trainR2** in the tables below) are computed over predictions from train data, while the test data scores (indicated as **testMSE** and **testR2**) are computed against the test data which has been set aside.

To test several different architectures and combinations of regularisation techniques, we wrote code which would parse a string with a standard model definition (explained in the next section) and create the model accordingly.

In several cases we present **plots of predicted vs true measure** for both train and test sets which often highlight overfitting problems. We also present **time series plots** of a year of observations and the models' predictions, to show how well a model was able to pickup and follow patterns (especially seasonality and weather extremes).

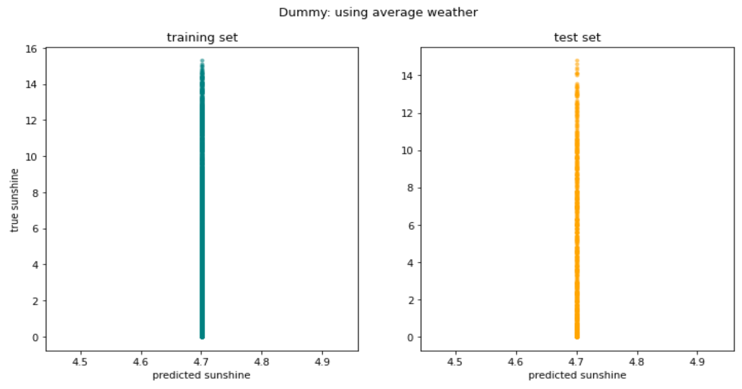
Naming convention

The model names, appearing in the tables below and as title of the plots, reflect the model architecture and important parameters. They encode the model definition which is then parsed to create the models. All model names start with L followed by a number (e.g. "L7" which indicate the sequence_length of observations used as input. After that the name contains the model layer type(s) followed by a number which indicates the number of units (e.g. R5 stands for RNN layer with 5 units). A number by itself indicates a Dense layer (e.g. a model like LSTM10-5 would imply a LSTM layer followed by a Dense layer of 5 units). Layers with dropout are indicated by a d followed by the dropout amount (e.g. d0.4 is 40%), whereas a capital D indicates input_dropout of RNN/LSTM. Finally, model names ending with "es" indicate Early Stopping was used in their training.

Baseline

To compare machine learning models it is often useful to have a baseline. The simplest approach is to use a dummy regressor, which always uses the mean for the measure as the prediction. For BASEL_sunshine the dummy regressor always predict 4.7 hours of sunshine, regardless. It obviously performs extremely bad:

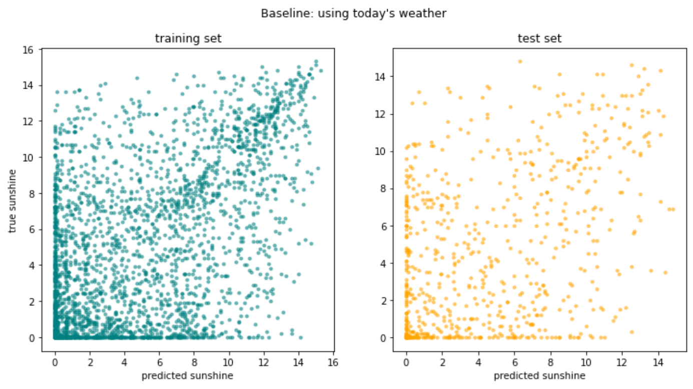
	trainMSE	trainR2	testMSE	testR2
dummy_mean	19.082	0.000	17.403	-0.002



Another baseline would be to use today's measured observation in order to predict tomorrow's weather. For example, if today the city's sunshine was measured in 5 hours, this baseline model would predict tomorrow's sunshine to also be 5 hours.

It performs better than the dummy regressor, but still poorly, showing that today's amount of sunshine is not a good indicator of the amount of sunshine we'll see tomorrow:

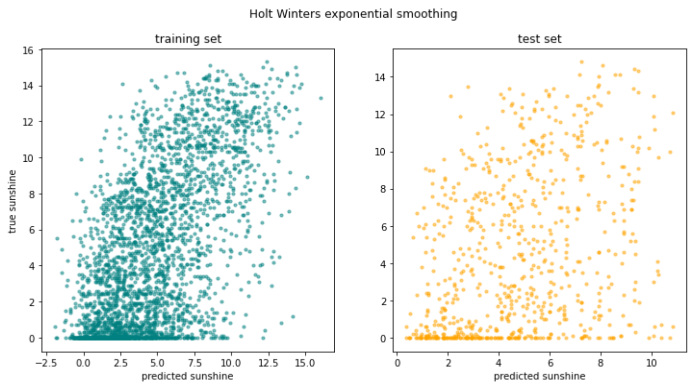
	trainMSE	trainR2	testMSE	testR2
same_as_today	17.404	0.088	19.300	-0.112



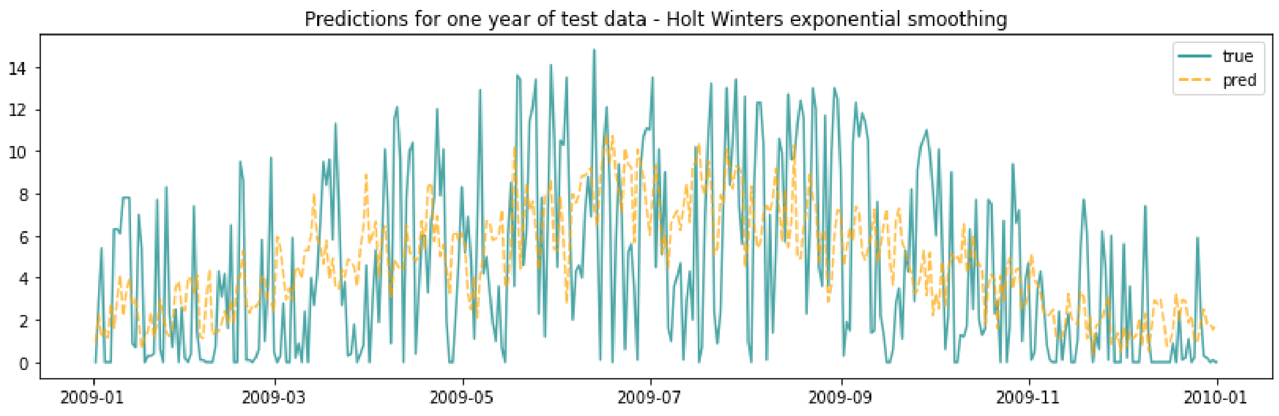
A better baseline is to fit Holt Winter's Exponential Smoothing (from ***statsmodels*** package) on 8 years (80% of the dataset) of the target measure time series and use it to forecast the last two years.

The fit on training data is much better, as shown by the lower MSE, but we are still far away from being able to obtain good predictions:

	trainMSE	trainR2	testMSE	testR2
exp_smoothing	12.637	0.000	16.926	-0.002



The following plot shows the forecast for the last year over-imposed to the real measures:

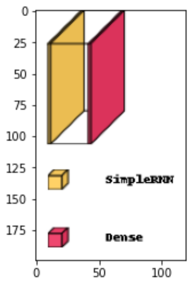


Shallow models

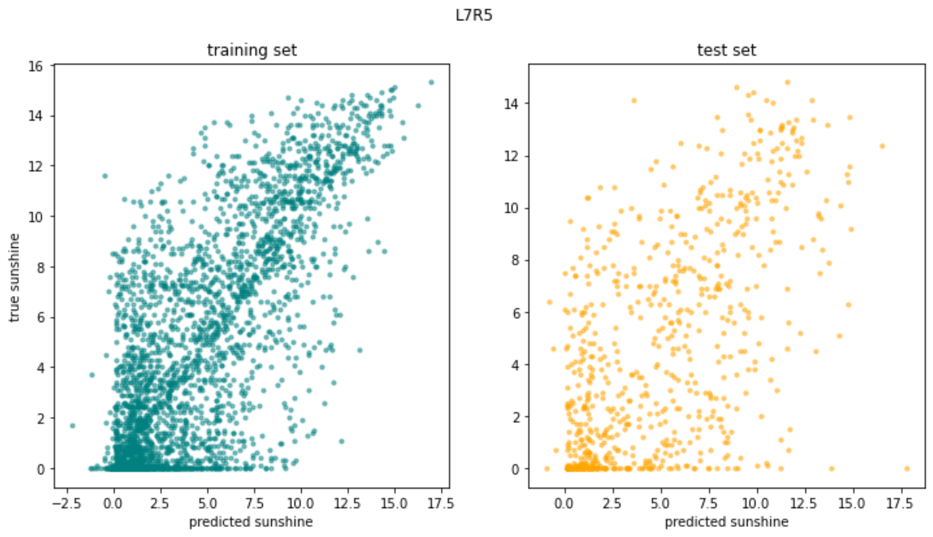
We started testing simple model architectures of a single SimpleRNN layer of 5 units.

We trained models using different sequence_length ("look-back") values, i.e. presenting to the models a sequence of 1, 7, 14, 30 or 60 days of weather measures to predict the target measure on the day after the sequence.

	trainMSE	testMSE	trainR2	testR2
L1R5	8.999	10.786	0.528	0.379
L7R5	8.517	13.414	0.554	0.228
L14R5	8.584	12.604	0.551	0.277
L30R5	8.539	11.326	0.554	0.357
L60R5	8.626	10.880	0.550	0.386



Without any regularisation, models with sequence_length 7 and 14 got the worst result on test data, but we believe this to be a side effect of overfitting.



In fact, as previously noted in the report on DeepLearning models, overfitting is one the main issues to tackle before exploring model architectures.

To address this, we applied a series of regularisation strategies:

1) EarlyStopping: to interrupt training when validation scores have stopped improving. An *EarlyStopping* callback during training, monitoring validation MSE with a patience of 15 (number of epochs with no improvement after which training will be stopped).

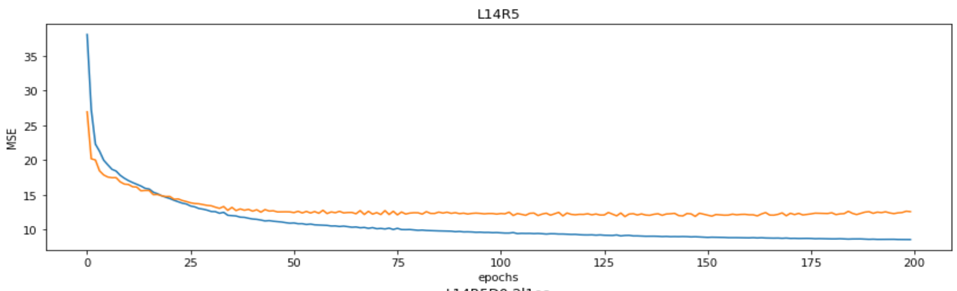
2) L1 regularisation: kernel weight regulariser was added, with the default L1 regularisation factor of 0.01

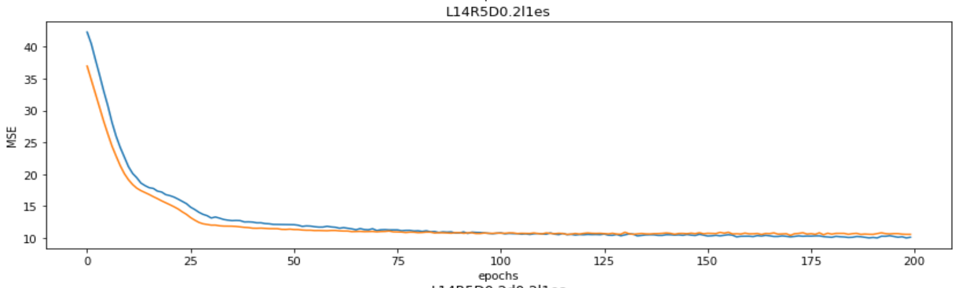
3) Dropout: dropping a fraction of the units for the linear transformation of the inputs at each step during training time. In case of RNN and LSTM layers we also tried *recurrent dropout:* dropping a fraction of the units for the linear transformation of the recurrent state.

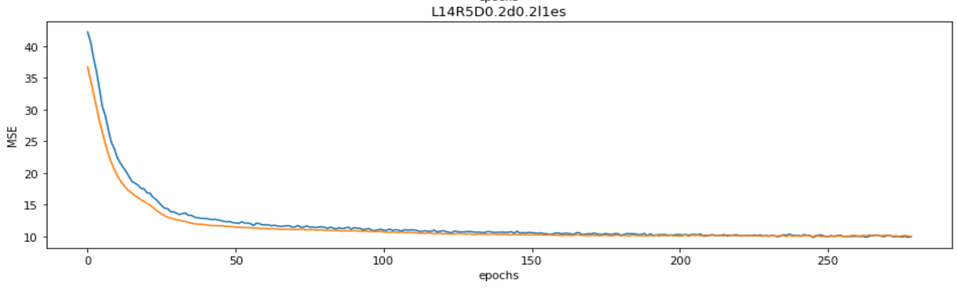
We tried all combinations of the above techniques.

The following plots – which show MSE vs Epoch during the training of the models – and the table with the metrics clearly reveal how the un-regularised model gets better and better on the train set but performance on validation data deteriorates. Not-regularised models achieve the lowest MSE on train data but highest (hence worst) on test data.

	trainMSE	testMSE	trainR2	testR2
L14R5	8.584	12.604	0.551	0.277
L14R5D0.2l1es	9.728	10.585	0.491	0.393
L14R5D0.2d0.2l1es	9.580	9.955	0.499	0.429







Blue: train data, Orange: test data (validation)

Comparing the same architecture and regularisation parameters for different sequence lengths now paints a very different picture, with actually a slightly better performance on sequence_length 14:

	trainMSE	testMSE	trainR2	testR2
L1R5D0.2d0.2l1es	9.423	10.105	0.506	0.418
L7R5D0.2d0.2l1es	9.318	10.033	0.512	0.422
L14R5D0.2d0.2l1es	9.580	9.955	0.499	0.429
L30R5D0.2d0.2l1es	9.377	10.078	0.510	0.428
L60R5D0.2d0.2l1es	9.498	10.200	0.504	0.424

As for LSTM, they take much longer to train but they should provide better results than RNN models. In fact, at least with this same model architecture, they performed worse:

	trainMSE	testMSE	trainR2	testR2
L1LSTM5D0.2d0.2l1es	9.773	10.576	0.488	0.391
L7LSTM5D0.2d0.2l1es	8.891	10.305	0.535	0.407
L14LSTM5D0.2d0.2l1es	8.978	10.196	0.530	0.415
L30LSTM5D0.2d0.2l1es	9.107	10.546	0.524	0.401
L60LSTM5D0.2d0.2l1es	9.230	10.507	0.518	0.407

This is probably due to different hyper-parameters needed to best tune the LSTM models.

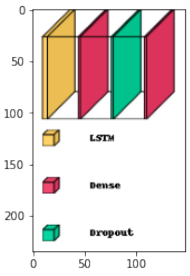
Going wider and deeper

We then proceeded to add units ("going wider") and layers ("going deeper") to the model.

As previously mentioned in the report on DeepLearning analyses, the number of possible model architectures and the number of hyper-parameters to try (e.g. all the combination of regularisation techniques and the values for those, like the proportion of dropout or the alpha parameter for L1 regularisation) quickly creates a runaway abundance of models to be tested.

We hence turned to tuning the model architecture using ***Keras Tuner***.

For each sequence_length we launched a search which explored the space of possible models created according to a series of parameters, comparing their performance after training to choose the ones with lowest MSE on test data.



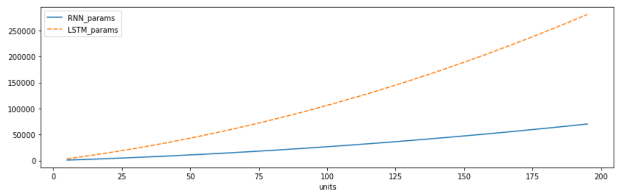
We tried both RNN and LSTM layers, with units ranging from 5 to 45 (in steps of 10), with combinations of L1 regularisation (none, 0.01 or 0.02) and dropout amounts (recurrent dropout, input dropout for RNN and LSTM, of either none, 0.2 or 0.4) and with an optional additional hidden dense layer (of 5 10 or 15 units) with optional dropout layer (0, 0.2 or 0.4 amount) and L1 kernel regularisation (none, 0.01 or 0.02). This meant a total search space of 7560 model architectures, and adding two choices for the *adam* optimizer's learning rate (0.001 and 0.01) it implied a total number of 15120 combinations. Using a *BayesianOptimization* strategy, the tuner tries to converge to the best model in an intelligent way, using the information from combinations already tried, without having to train and test all possible models.

We note that both longer sequence lengths and higher number of units increase considerably the training time.

A higher number of units mean a much higher number of weights that need to be trained. Given an LSTM layer of units u trained on f features, the number of weights is:

$$4\cdot(u + f\cdot u + u^2)$$

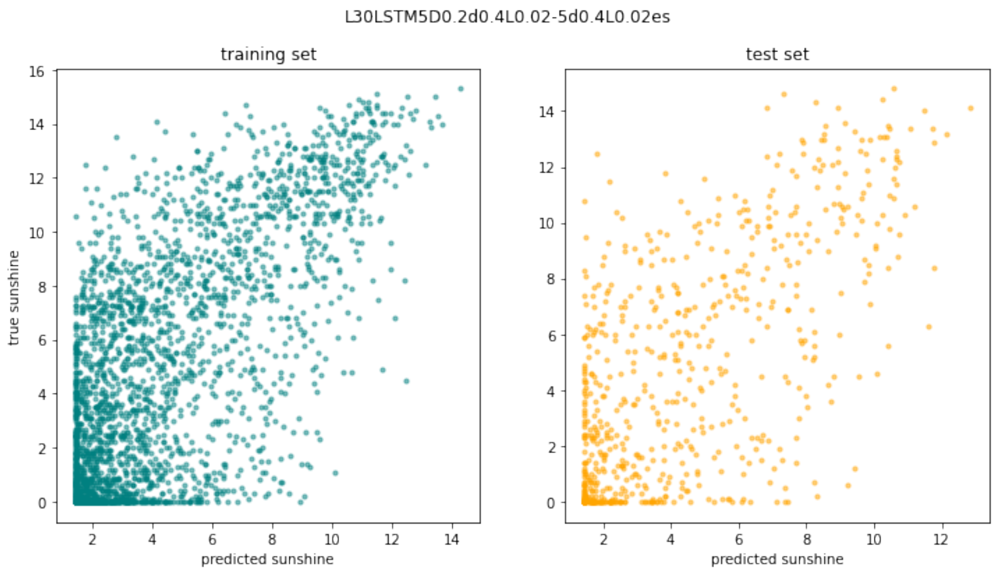
So for an LSTM of 5 units there would be 3400 parameters, 37800 parameters for 45 units and 292000 for 200 units. Conversely for RNN: for 5 units there would be 850 parameters, 9450 for 45 units and 73000 for 200 units.



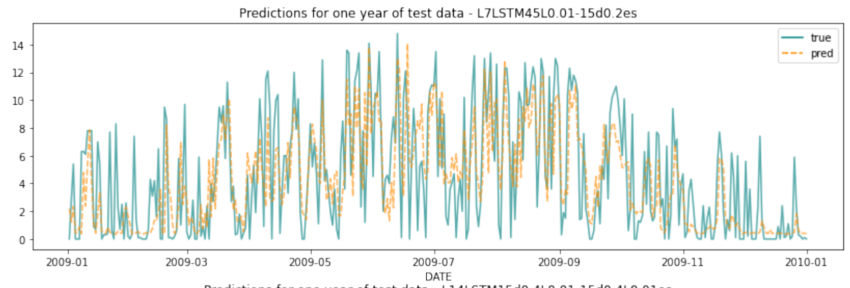
trainable parameters as function of recurrent units

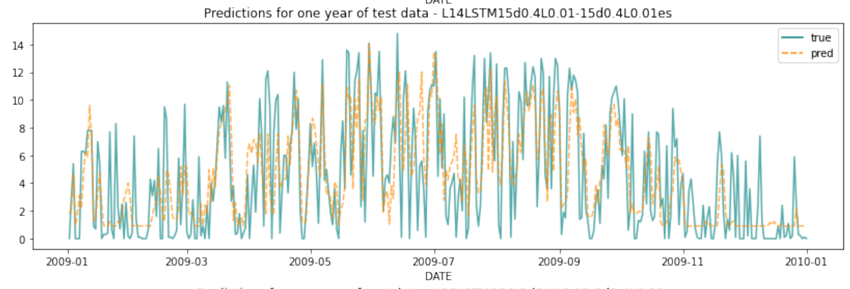
The best models found by the tuner, for different sequence lengths, were the following. They are all LSTM models, with much higher accuracy than what seen before (average of 0.46 R^2 on test data):

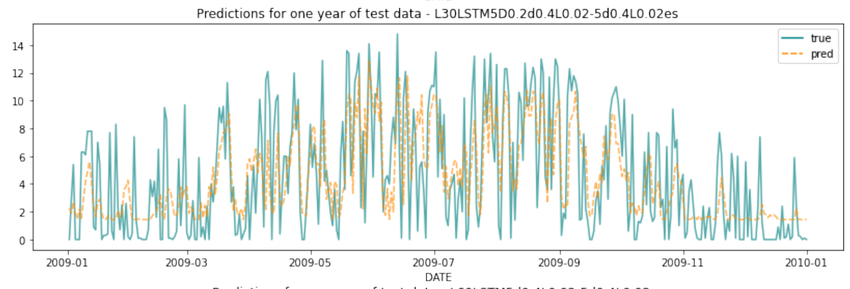
	trainMSE	testMSE	trainR2	testR2
L7LSTM45L0.01-15d0.2es	9.288	9.536	0.514	0.451
L14LSTM15d0.4L0.01-15d0.4L0.01es	8.872	9.329	0.536	0.465
L30LSTM5D0.2d0.4L0.02-5d0.4L0.02es	9.681	9.607	0.494	0.455
L60LSTM5d0.4L0.02-5d0.4L0.02es	9.539	9.609	0.502	0.457

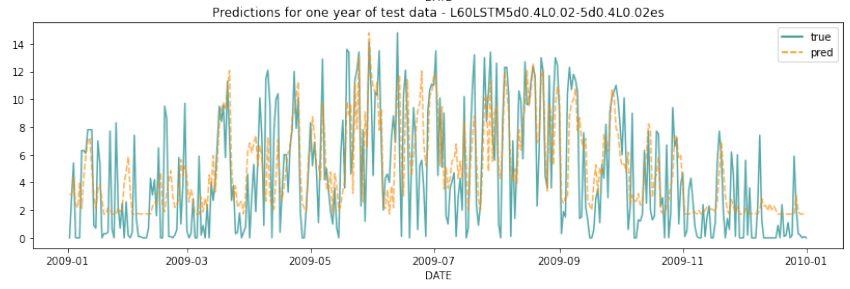


Interestingly, by observing the plots of predictions for the last year of test data we can see that the longer the sequence_length, the more the models err on "optimism": predicting more sunshine hours for days when there actually were none or very little:









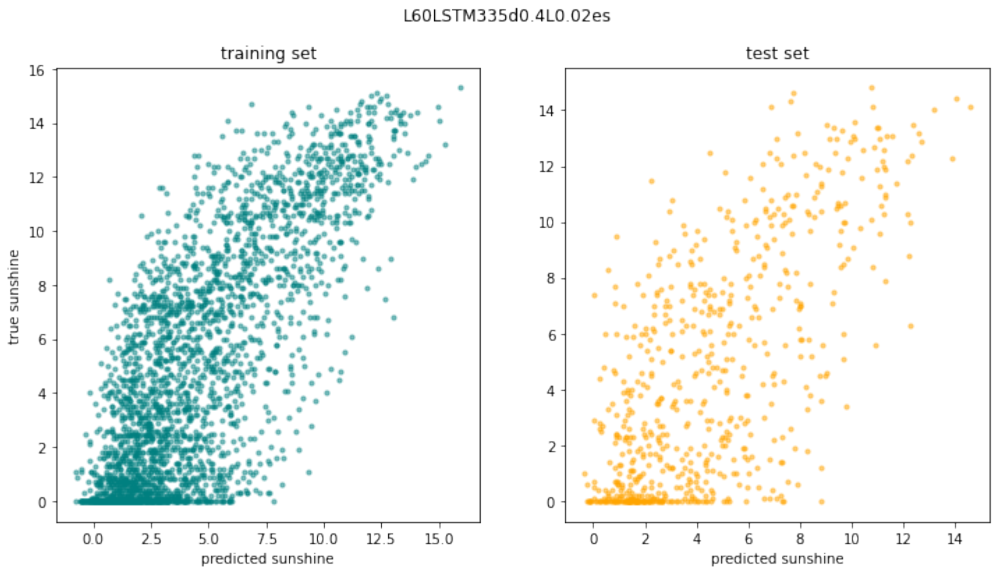
It may be that being given longer input sequences, the LSTM becomes more biased towards an average trend rather than short term variations. Nevertheless all plots show a very good correspondence to the real data, not only picking up the seasonality but also the different good and bad weather periods (and auto-correlations) inside each month.

Going even wider

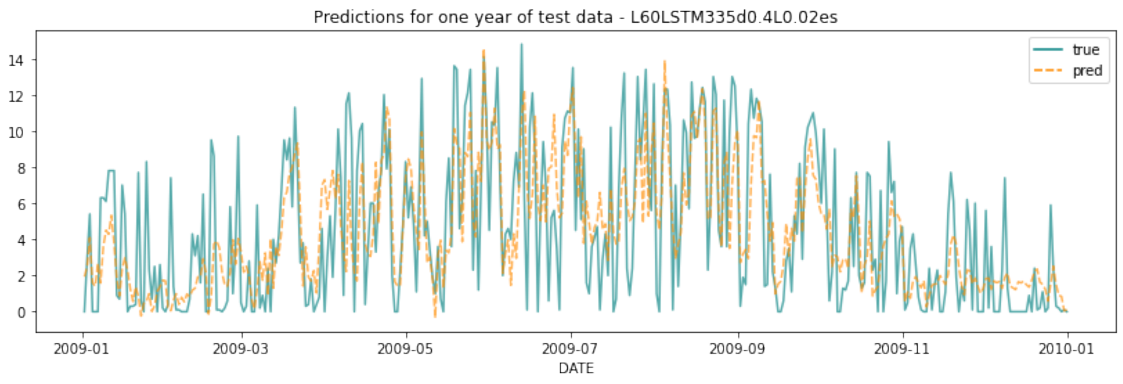
The number of input features is quite high, but as shown in previous reports, very correlated. That's why we started with a small number of units for the RNN or LSTM initial layers.

Nevertheless it is important to try and expand the search space to wider layers (higher number of recurrent units). This obviously took much longer, but yielded even higher accuracy:

	trainMSE	testMSE	trainR2	testR2
L7LSTM65D0.4d0.2L0.02-15d0.4L0.01es	7.918	9.318	0.586	0.464
L14LSTM335d0.4L0.02-15L0.01es	8.308	9.308	0.565	0.466
L30LSTM215d0.2L0.02-15d0.4L0.01es	8.159	9.368	0.573	0.468
L60LSTM335d0.4L0.02es	7.780	9.303	0.594	0.475



The predictions now follow even more closely the real data, with much less "optimism", although still erring a little more on completely overcast days.



Final model recommendation

After testing extensively the models described above, we recommend using the L60LSTM335d0.4L0.02es model (an LSTM layer of 335 units with recurrent dropout of 40% and L1 regularisation with alpha of 0.02, early stopping) trained over all the available columns scaled with standard scaling and transforming with log1p the most skewed features (>0.75 skew value), for sequences of 60 subsequent days to predict the following day.

As our objective was to aim for the most predictive power, that is the model with higher accuracy, although it is slow to train, with 670,336 parameters.

The test R^2 score for that model was 0.475. This is better, although slightly, than what obtained with the best DeepLearning model trained on a single day of measures (as shown in a previous report, we had obtained a test R^2 of 0.464 for a shallow model with a single Dense layer of 300 units with L2 regularisation followed by a 0.4 dropout layer).

Key findings and insights

We determined that although it is a very difficult problem to predict tomorrow's weather, we can do much better with appropriate models compared to baseline assumptions.

As neural networks are very prone to overfitting, adequate regularisation strategies are fundamental.

Creating deep learning models is not a trivial task. In particular, the number of possibilities for model architecture is so vast and this, coupled with the tuning of so many hyper-parameters, quickly creates a runaway abundance of potential models to be tested. This is even more relevant for time-series analyses as there is now also the additional parameter of length of input sequences. The use of Keras Tuner proved invaluable to cope with the combinatorics of possible models.

We also found that to deal with longer input sequences, wider LSTM layers are apparently needed, while for shorter input sequences, more narrow layers performed better.

It is still quite hard to achieve very high accuracy performance for the sunshine variable. Deep learning models are known to benefit from a great abundance of training data; it is entirely possible that having access to more years of weather observations would improve the results.

Next steps and limitations

It would be both straightforward and interesting to retrain all models changing the target measure.

Preliminary results show that for example pressure should be much easier to predict, as it is more consistent from day to day. In a previous report we showed how the baseline "same as today" model in fact already achieves 0.65 test R^2: and a previously tested shallow DeepLearning model reached 0.76.

Another interesting extension would be to try the simultaneous **prediction of multiple measures**. For example predicting the whole set of weather measures of a certain location, rather than only one. This can be achieved by simply changing the y_data to be multidimensional, without further modifications of the analysis pipeline.

Another extension, would be **multi step forecast**: feeding back the prediction for day n+1 and continue to predict the target measure for more days (n+2, n+3…).

For example the model could be using the last month of data to predict a full week into the future.

Also, more complex models could be tried, like stacked LSTM (two subsequent LSTM layers) or a hybrid model combining CNN and LSTM.

Conclusions

The dataset looked like a very promising one to use for machine learning with limited amount of data cleaning necessary.

The major initial drawback was that only two features are available at all locations (`temp_mean` and `temp_max`), but it is too restrictive to only consider these two. To achieve a balanced dataset it would be needed to drop a combination of some features and some locations in order to maximise the quantity of data left.

We also confirmed how training of neural networks must always be coupled with appropriate regularisation techniques to avoid heavily over-fitting models and showed the relevant contributions of different strategies.

When dealing with neural networks, not only it is very important to setup the analysis properly (like for other machine learning analyses) but it is also paramount to have a good way to organise the model creation and testing pipelines, due to the great abundance of possible combinations.

We addressed this firstly by using parsable definitions for models and code that would create a model from the definition; secondly, by using a tuner for intelligently exploring the model search space.

Methods and Materials

Data manipulation and analysis was performed on a MacOS laptop using own written code in python language, working in a jupyter notebook and taking advantage of the following python libraries: pandas, seaborn, scikit-learn, tensorflow / keras, visualkeras, statsmodels, pmdarima, keras tuner

Data origin acknowledgement

Dataset compiled by Huber, Florian (2021); Zenodo. https://doi.org/10.5281/zenodo.4770937

ORIGINAL DATA TAKEN FROM:

EUROPEAN CLIMATE ASSESSMENT & DATASET (ECA&D), file created on 22-04-2021 THESE DATA CAN BE USED FREELY PROVIDED THAT THE FOLLOWING SOURCE IS ACKNOWLEDGED:

Klein Tank, A.M.G. and Coauthors, 2002. Daily dataset of 20th-century surface air temperature and precipitation series for the European Climate Assessment. Int. J. of Climatol., 22, 1441-1453. Data and metadata available at http://www.ecad.eu