Report on ML Regression on a weather dataset

by Giuseppe Insana, December 2021

The dataset

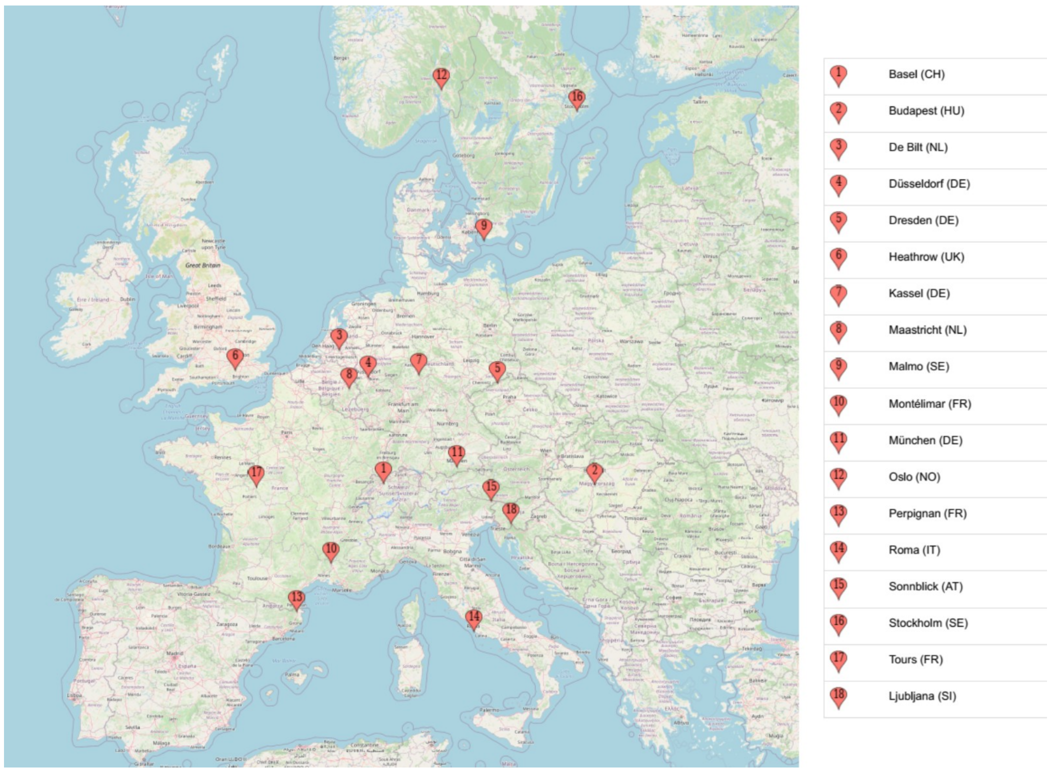
I chose to work on weather data, chiefly because I never worked on this kind of data before (having mostly dealt with biological and linguistic data in the past).

The weather data 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:



Objective of the analysis

We will try to predict tomorrow's weather for one location based on today's weather measures across all locations. We will focus chiefly on trying to achieve prediction accuracy but we will also try where possible to gain insights from the models, to help us in understanding the major factors that guide the predictions.

Data exploration

A comprehensive exploratory data analysis has been previously conducted on the model and presented in a previous report. 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 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.

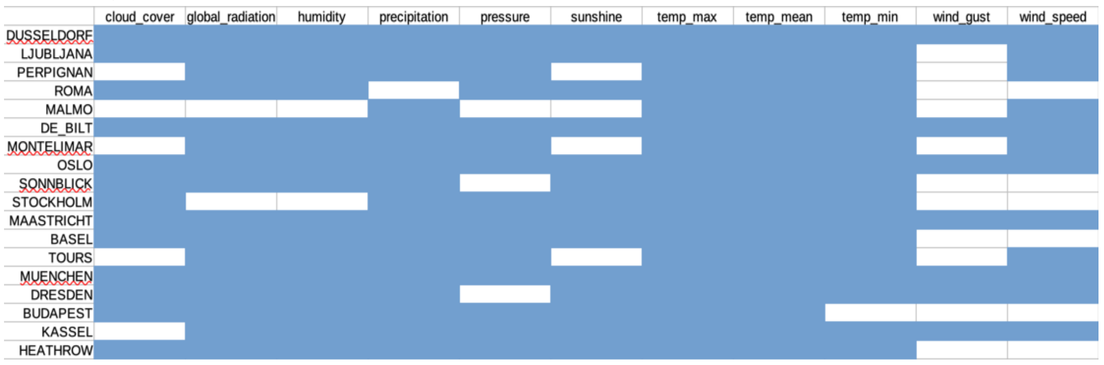
The measurements are labelled as LOCATION_*measure* (e.g. BASEL_cloud_cover, BASEL_pressure, OSLO_precipitation...) with the measured variables being: **cloud_cover, global_radiation, humidity, precipitation, pressure, sunshine, temp_max, temp_mean, temp_min, wind_gust, wind_speed**

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 can obviously present problems of **unbalanced data**, in particular when extending the model to the prediction for different cities. We will address this issue in the discussion.

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

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.

To see the effect of the outliers on the machine learning models, we will be training and comparing the regression models both before and after imputing the outliers.

Feature engineering

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

A simple feature engineering is the **addition of a DAY feature** (day of the month), which together with MONTH would give a time grounding for the predictions. We will be **dropping the DATE feature** from the training data and we will check **how important** the MONTH and the DAY informations are for weather prediction.

In the models we are going to train, we will be trying both **transformations** (transforming the skewed variables outlined above) and **scaling** of the features.

We will also be trying the addition of **interaction terms** using PolynomialFeatures.

The **biggest decision** to take is about the features to use as input for the regression models:

1. whether we want to keep all the features and all the locations

2. whether we'd prefer to have a balanced dataset which holds only a subset of features and a subset of locations so that we would have the same features for each location

We will be training models using both of these strategies so we can understand the impact of such a decision.

We believe that for the prediction of a certain variable at a certain location, knowledge of the weather variables at other locations, even if incomplete (some locations missing some features) could still be important, as long as the model will be able to deal with it.

Models training

Procedure

Each model tested (a part from the baseline models described below) will be setup using a *scikit-learn* **Pipeline**, grouping together the list of steps to be applied to the data, with the final one being the estimator and the preceding ones being transforms.

One of the main advantages of this setup is to be able to assemble a series of steps that can be cross-validated together over a series of different parameters.

It also lowers the risk of making mistakes with transformations and leaking test information into the train set.

At the beginning of the analysis we set aside a random 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.

For **cross validation**, we have used *KFold* with **5 splits**. This means that the training data is split in 5 subsets and each fold is then be used once as a validation while the remaining folds form the training set for that iteration.

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 averaging over the 5 folds according to the cross validation explained above (using *cross_val_score* function). The test data scores (indicated as **testMSE** and **testR2**) are computed after training the model on the entire train dataset and scoring against the test data which has been set aside.

The train scores are hence the averages of 5 fold cross validation and thus it can often be the case that the train scores are lower than the test scores.

The cross validation ensures a good comparison of the models for the training, and the final scoring on completely unseen test data is best practice to evaluate the prediction ability of the trained models.

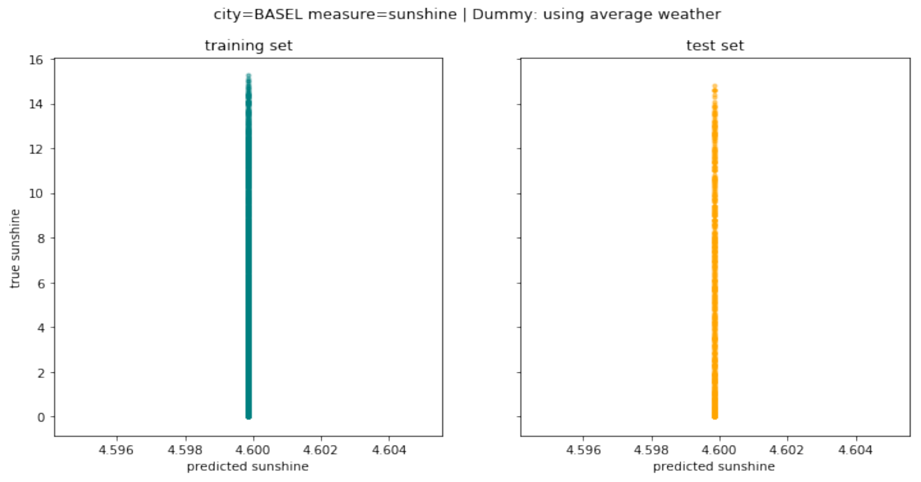
Note that to better illustrate the overfitting problems which in some case penalise the train scoring, the included **plots of predicted vs true measure** for train and test sets use the predictions based on the model trained on the whole train set (rather than plotting sklearn's *cross_val_predict* based predictions).

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 4.6 hours of sunshine, regardless.

It obviously performs extremely bad:

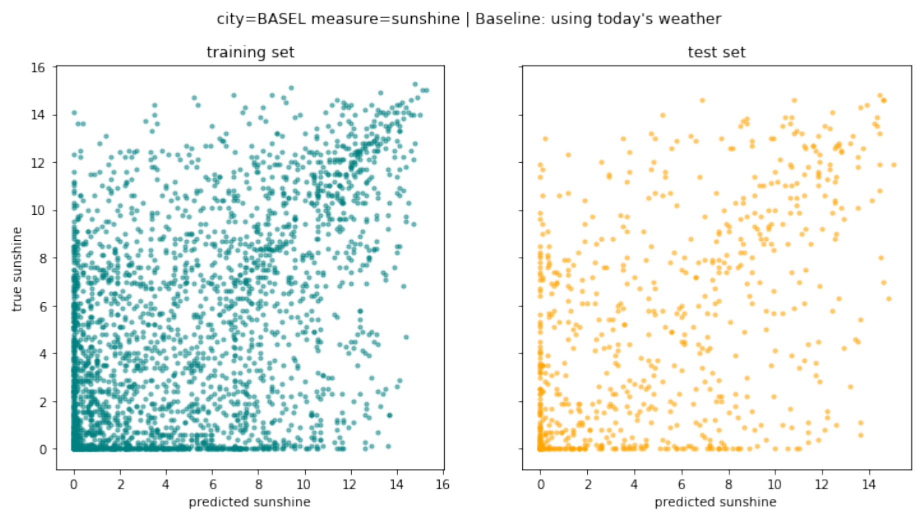
	trainMSE	trainR2	testMSE	testR2
dummy_mean	19.672	-0.004	19.597	-0.005



A better baseline we can devise is 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:

	trainMSE	trainR2	testMSE	testR2
same_as_today	17.494	0.056	18.917	0.030

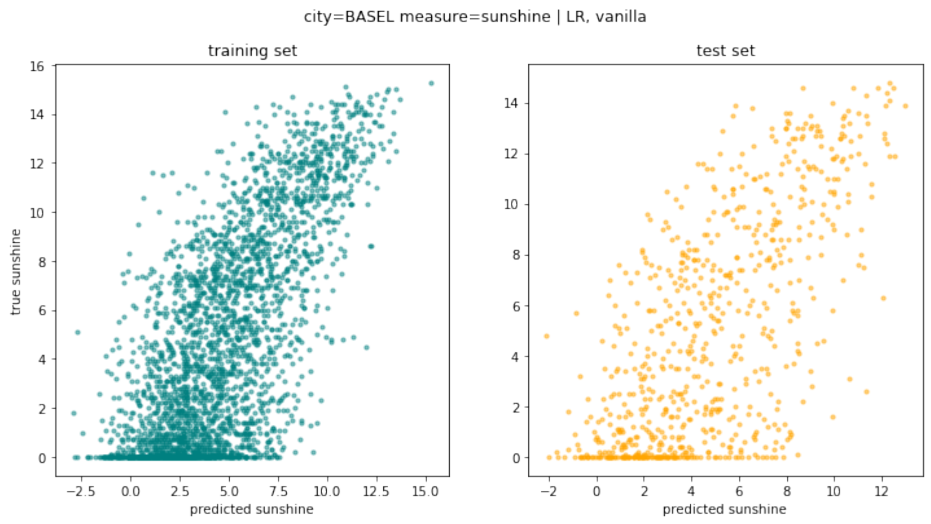


Simple Linear Regression

We can begin with simple linear regression, without any particular setup a part from the cross validation *KFold* explained above.

The results are encouraging. We have a much better predictive power than the baseline models, but we can probably do better.

	trainMSE	trainR2	testMSE	testR2
LR	10.432	0.435	10.779	0.447



Adding Column Transformations

We tested adding the transformer *log1p* only to specific columns (using sklearn's *ColumnTransformer*), first to all columns for the most skewed measures of precipitation, wind_gust and wind_speed and then only to the columns effectively more skewed (with skew value more than 0.75). This second strategy proved more successful in terms of R^2 score over both train and test sets.

It slightly improved the model, with a 1% increase in the R^2 metric for the test and 2% increase over train set:

	trainMSE	trainR2	testMSE	testR2
CT_LR	10.287	0.443	10.706	0.451

Adding Scaling

We then added scaling using sklearn's *StandardScaler*. As expected, there was no change in the performance of the basic LR model, but it is important for the other models.

	trainMSE	trainR2	testMSE	testR2
CTSS_LR	10.287	0.443	10.706	0.451

Putting all features on the same scale allows also to better interpret which features have more influence in predicting our target variable. For example we can check which features correspond to the fitted coefficients having a highest absolute value:

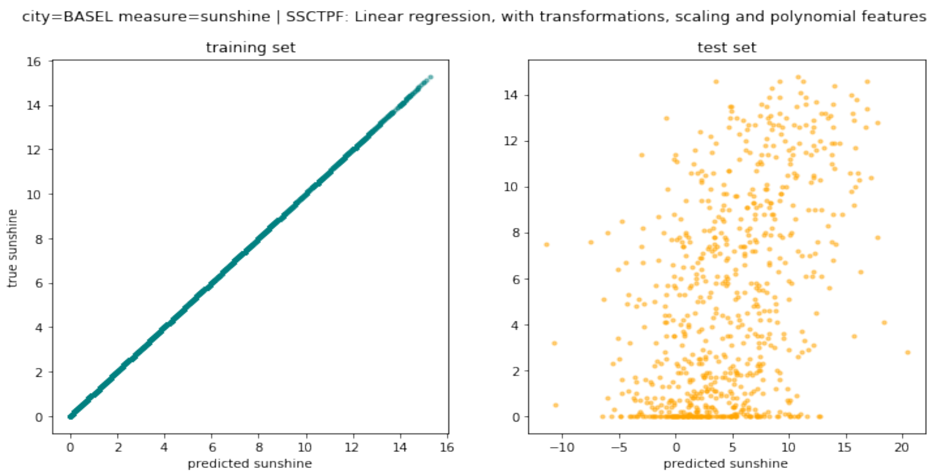
```
          Sum of coefficients: 124.62
     Coefficients set to zero: 0/164 (0.00%)
        Highest (absolute) coefficients:
MUENCHEN_temp_max          18.700439
OSLO_wind_speed           -11.531808
LJUBLJANA_pressure         -9.47122
OSLO_cloud_cover           -7.878451
MONTELIMAR_temp_mean        7.850594
TOURS_temp_mean             5.235179
TOURS_temp_max             -3.439027
DUSSELDORF_temp_mean        2.103888
OSLO_sunshine              -2.076062
HEATHROW_temp_min           2.011425
```

Adding PolynomialFeatures

The addition of polynomial features of second degree brings the number of features (and hence of parameters to train) from 164 to 13694 (the original measures, the measures squared and all the combinations of feature pairs).

The results are extremely bad with the model overfitting dramatically. The model trained on the whole set makes very good predictions on the training set (as can be seen in the left plot) but very bad on the test set. And even on the train set, the cross-validation folds have extremely poor performance as shown by the scoring metrics:

	trainMSE	trainR2	testMSE	testR2
CTSSPF_LR	19.540	-0.058	24.377	-0.250



Adding regularisation: Lasso

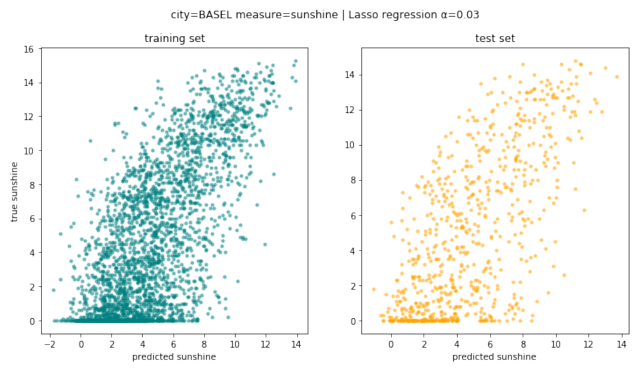
As shown by the above metrics, the addition of PolynomialFeatures creates drastic overfitting. To cope with that, regularisation techniques need to be added, introducing a cost component to the coefficients.

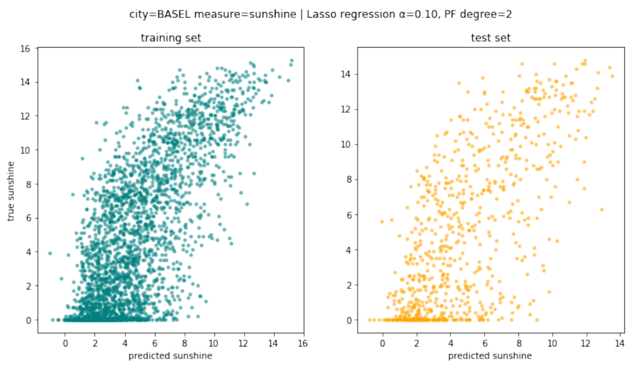
We used both Lasso and Ridge, with and without PolynomialFeatures of degree two while keeping the previously introduced transforms (*log1p* to skewed features and *StandardScaler*).

To tune hyper-parameters for regularisation, sklearn's *GridSearchCV* was used.

With polynomial features and an alpha of 0.1, Lasso achieves a cross-validated R^2 score of 0.493 on the training set and 0.468 on the test set:

	trainMSE	trainR2	testMSE	testR2
Lasso	9.992	0.459	10.700	0.451
	trainMSE	trainR2	testMSE	testR2
LassoPF	9.365	0.493	10.376	0.468





Lasso also has the advantage of setting coefficients to zero, so selectively removing the least important features from the model. Only 60 features out of 164 were kept for Lasso (zeroing 63.41% of the coefficients) and only 166 features out of 13694 for LassoPF (with polynomial features):

```
            Lasso, α=0.03
        Sum of coefficients: 8.21
Coefficients set to zero: 104/164 (63.41%)

           LassoPF, α=0.10
        Sum of coefficients: 9.21
Coefficients set to zero: 13528/13694 (98.79%)
```

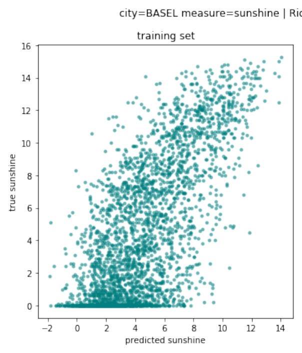
We will discuss below the most important features retained by Lasso.

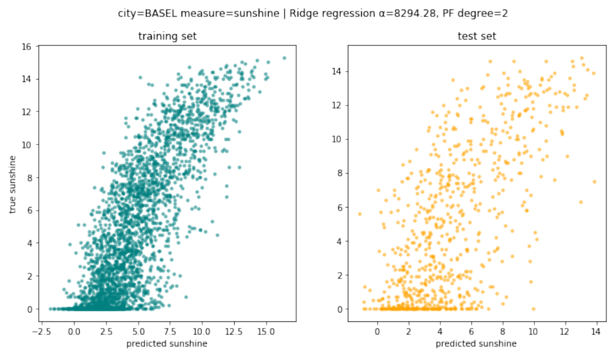
Adding regularisation: Ridge

Another regularisation technique, using a different formula to penalise high coefficients is used by the Ridge model.

The results were not better than Lasso:

	trainMSE	trainR2	testMSE	testR2
Ridge	10.050	0.456	10.967	0.438
RidgePF	10.077	0.455	10.515	0.461





These plots and those above for Lasso also make it quite clear that the models with polynomial features suffer more for overfitting, with the training predictions much better (when not cross-validated), but with same results on the final evaluation.

Furthermore, although the training time needed is lower, the fact that Ridge does not zero any of the coefficients makes it less attractive than Lasso for interpretation:

```
          Ridge, α=317.48
     Sum of coefficients: 15.20
Coefficients set to zero: 0/164 (0.00%)

         RidgePF, α=8294.28
     Sum of coefficients: 59.20
Coefficients set to zero: 0/13694 (0.00%)
```

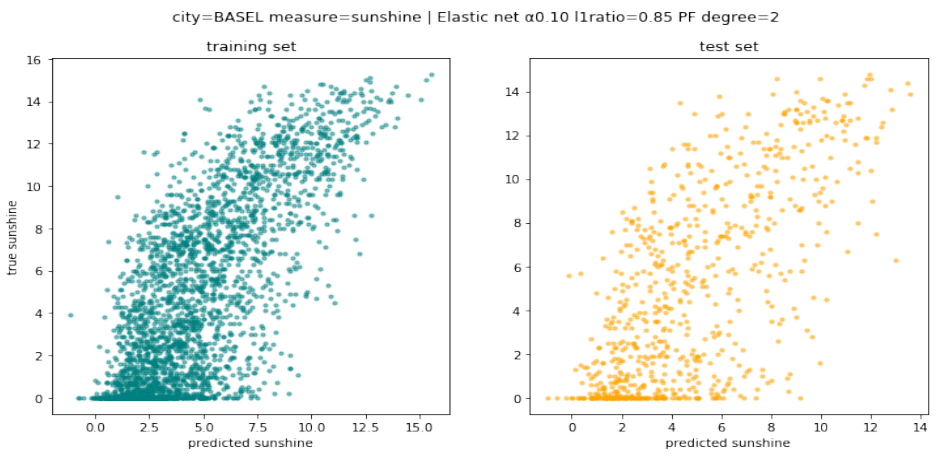
Combining regularisation techniques: Elastic Net

Ridge utilizes an L2 penalty and lasso uses an L1 penalty. Elastic net uses both the L2 and the L1 penalties.

Elastic-net is also considered useful when there are multiple features correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

The results are as good or slightly better than LassoPF:

	trainMSE	trainR2	testMSE	testR2
ElasticNet	9.348	0.494	10.326	0.470



The best hyper-parameters for ElasticNet were an alpha of 0.1 and an L1 ratio of 0.85. It kept 242 features with a non-zero coefficient:

```
ElasticNet, PF=2, α=0.10 l1_ratio': 0.85
        Sum of coefficients: 11.89
 Coefficients set to zero: 13452/13694 (98.23%)
```

This can be compared with the number of features LassoPF kept (166) to confirm the understanding of how ElasticNet works.

Feature importance

The top features identified by both Lasso and ElasticNet model (with the PolynomialFeatures included) are the same, with minor variations in the order. The top 20, having absolute coefficients above 0.1, are the following:

LassoPF	ElasticNET
TOURS_precipitation	TOURS_precipitation
MAASTRICHT_global_radiation	MAASTRICHT_global_radiation
BASEL_temp_mean	BASEL_temp_mean
ROMA_humidity	ROMA_humidity
DUSSELDORF_global_radiation	MUENCHEN_temp_min
BASEL_temp_min	BASEL_temp_min
DE_BILT_pressure	DE_BILT_pressure
MUENCHEN_temp_min	DUSSELDORF_sunshine
BUDAPEST_cloud_cover	DUSSELDORF_global_radiation
DUSSELDORF_sunshine	BUDAPEST_cloud_cover
DRESDEN_temp_max	DRESDEN_temp_max
BUDAPEST_cloud_cover DUSSELDORF_sunshine	BUDAPEST_cloud_cover DUSSELDORF_sunshine
DRESDEN_sunshine MUENCHEN_temp_min	DUSSELDORF_humidity
DUSSELDORF_sunshine^2	DRESDEN_sunshine MUENCHEN_temp_min
DRESDEN_sunshine	DUSSELDORF_sunshine^2
DUSSELDORF_humidity	DRESDEN_sunshine
MUENCHEN_temp_min ROMA_humidity	MONTELIMAR_temp_max
BASEL_temp_max	DUSSELDORF_wind_speed ROMA_sunshine
LJUBLJANA_cloud_cover TOURS_global_radiation	BUDAPEST_global_radiation DUSSELDORF_pressure
DE_BILT_pressure TOURS_precipitation	MUENCHEN_temp_min ROMA_humidity

Interestingly BASEL_sunshine (alone or in combination with other features) appears only near the end for both models, revealing even more (like above seen with the baseline model) how today's sunshine hours is not a very good indication of tomorrow's sunshine.

Similarly interesting is the fact that the MONTH feature appears after about 40 other features in the feature importance analysis, and only in combination with other features. The naive hypothesis was that MONTH would be a good predictor for the weather, due to the seasonal relationships previously observed.

Alternative strategies

These are alternative strategies we have tried and compared to what presented above:

1. Pre-selecting only a subset of features and cities (to create a balanced set), i.e. using only the measures 'cloud_cover', 'global_radiation', 'precipitation', 'pressure', 'sunshine' for the cities 'BASEL', 'BUDAPEST', 'DE_BILT', 'DUSSELDORF', 'HEATHROW', 'LJUBLJANA', 'MAASTRICHT', 'MUENCHEN', 'OSLO';
2. Pre-selecting the features most correlated to the target feature, using a table of correlation;

3. Removing the MONTH feature, as it did not seem to be have high coefficients fit to it by the models trained;
4. Adding a DAY feature, with the idea that maybe having both MONTH & DAY would help the prediction power;
5. Imputing the biggest outliers.

Results and comparison of the alternative strategies

For (1), using the balanced set resulted overall in a worse performance across all models tested. Interestingly, the models with polynomial features performed worse than the ones without. For example:

	trainMSE	trainR2	testMSE	testR2
Lasso	10.719	0.420	11.027	0.434
LassoPF	10.272	0.444	11.347	0.418

For (2), the columns with an absolute correlation value to the target feature of more than 0.5, sorted by absolute correlation are: 'TOURS_global_radiation', 'BASEL_global_radiation', 'MONTELIMAR_global_radiation', 'MAASTRICHT_global_radiation', 'BASEL_sunshine', 'TOURS_humidity'

Using only those as input, and running all previously described models, resulted in much worse performance, with the best score being:

	trainMSE	trainR2	testMSE	testR2
ElasticNet	11.987	0.394	10.315	0.442

On the other hand using as input only the features most correlated to *today's* BASEL_sunshine to predict tomorrow's measure as input resulted in comparable but slightly worse performance, with the best score being:

	trainMSE	trainR2	testMSE	testR2
ElasticNet	12.022	0.392	10.174	0.450

The only advantage of this strategy of pre-selecting the most correlated columns would be to speed up training, but we would not recommend it, preferring to instead let the model identify the most important features.

For (3), removing MONTH information resulted in only slightly worse predictions, with the best scoring model achieving:

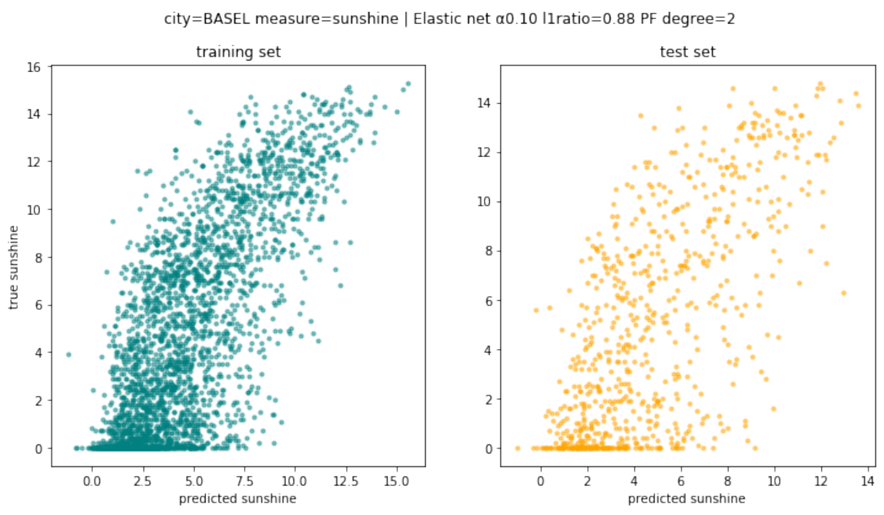
	trainMSE	trainR2	testMSE	testR2
ElasticNet	9.358	0.494	10.377	0.468

This confirmed the observation made above that MONTH's trained coefficients were not very high, pointing to this feature not appearing to be very important, but still worth keeping.

For (4), keeping MONTH and also adding a DAY feature (day of the month) resulted in equal or slightly better predictions:

	trainMSE	trainR2	testMSE	testR2
LR	10.436	0.435	10.761	0.448

CT_LR	10.290	0.443	10.693	0.452
CTSS_LR	10.290	0.443	10.693	0.452
Lasso	9.993	0.459	10.698	0.451
LassoPF	9.383	0.492	10.378	0.468
Ridge	9.940	0.462	10.834	0.444
RidgePF	10.103	0.453	10.533	0.460
ElasticNet	9.368	0.493	10.327	0.470



Finally, for (5) we also tried imputing the most extreme outliers for precipitation (369 values with more than 30 mm of daily precipitation), humidity (29 values lower than 25%) and wind_speed (30 with more than 12.5 m/s). This resulted in yet another slight improvement across most models, in particular offering a +1% improvement in R^2 test scores for RidgePF but with a slight decrease in the best scoring ElasticNet. The best hyper-parameters found for ElasticNet were the same as in the case without imputing outliers.

	trainMSE	trainR2	testMSE	testR2
LR	10.444	0.435	10.697	0.451
CT_LR	10.321	0.441	10.688	0.452
CTSS_LR	10.321	0.441	10.688	0.452
Lasso	9.996	0.459	10.707	0.451
LassoPF	9.401	0.491	10.370	0.468
Ridge	9.958	0.461	10.811	0.446
RidgePF	10.109	0.453	10.441	0.465
ElasticNet	9.398	0.491	10.354	0.469

Final model recommendation

After testing extensively the models described above (with and without Transforms, PolynomialFeatures, Regularisation) for all the strategies we outlined, we recommend using the **ElasticNet** model, with standard **scaling** and **log1p** transformation of the most skewed features, addition of **PolynomialFeatures** of 2nd degree and keeping both **MONTH** and **DAY** information, **without** necessarily having to impute the most extreme outliers (i.e. strategy (4) outlined above).

The best R^2 score for the final ElasticNet model was 0.470, with hyper-parameters: `PF=2, α=0.10 l1_ratio': 0.88`

The sum of coefficients was 10.49, with 13641 out of 13860 coefficients set to zero (98.42%).

As our objective was to aim for the most predictive power, we think this is the recommended model. If we were more interested in interpretation, then a Lasso model without PolynomialFeatures would be easier to interpret.

Key findings and insights

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

It was also made very clear that adding PolynomialFeatures without regularisation penalties creates useless models due with too much overfitting.

Scaling and transformation of skewed distributions help in achieving better predictions even in normal Linear Regression.

Feature importance analyses showed that about two hundred features out of the possible fourteen thousand are necessary and sufficient for the model to achieve its best predictive power.

It also showed how today's measure was very low in the rank of feature importance and that the month or day informations, although giving some marginal improvement, were also not fundamental.

Interestingly, the features with the highest coefficients from the same city we were trying to predict the weather of were limited to the temperature ones, while most of the top coefficients are for measures from locations surrounding it, like Tours, Maastricht, Roma and Dusseldorf.

The ElasticNet model proved the most performant and offered both the interpretability of Lasso and the performance of Ridge.

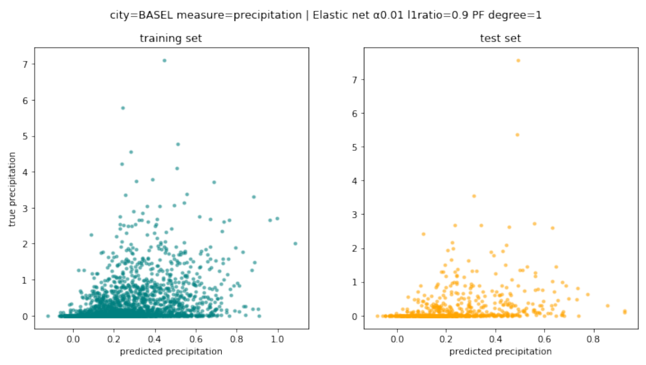
Imputing the most extreme outliers offered some improvement, in particular for the RidgePF model, but did not achieve the same best score as the basic strategy or the one adding the DAY information. Being an extra step which could be debatable (in choosing the threshold of what is considered outlier) and as it does not offer significant improvement, we didn't recommend it.

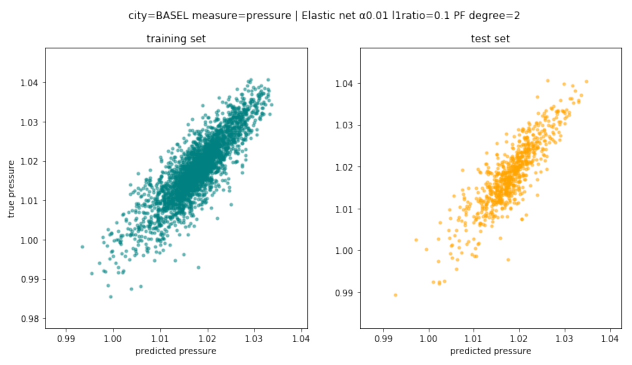
From the plots of predicted vs true values, it appears all models make very bad predictions when the true value is 0 where they instead predict a certain number of sunshine hours.

Next steps

It would be both straightforward and interesting to retrain all models changing the target measure. In particular for two main reasons: firstly, to see which ones are easier to predict and which ones are harder; secondly, to see whether the fact that the dataset is originally unbalanced has a big impact in predicting weather measures for some cities.

Preliminary results show that for example precipitation levels are much harder to predict, while pressure is much easier (apparently more consistent from day to day):





Another important extension would be to train a model that would simultaneously predict a variety of measures, rather than a single measure. The y_data would hence be a matrix, and not a vector. For example selecting as target not a single measure for a single city, like we did, but the same measure across several cities (e.g. sunshine across the locations) or conversely all the possible measures for a single city.

It would also merit attention the possibility of turning this into a classification problem, like "will it rain tomorrow?" and also repeating the analysis using different and possibly more advanced Machine Learning techniques.

Another interesting extension would be to work with a time frame of observations to predict tomorrow's weather. Instead of using only today's measure across all locations, we could use a series of measures for subsequent days (e.g. the weather of the past *week*) to predict tomorrow's weather.

Finally, it is worth noting that although the dataset's metadata mentions wind_direction data and informs us that it is measured in degrees, this feature was not present in the dataset. We think it would be very important to add this variable to the data set as it could prove important for weather prediction, with the hypothesis that the weather at one location could influence the weather at another location situated in the direction towards which the wind was blowing.

We could contact the data providers and see whether this information can be added, repeating then the analysis without.

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 is better to instead drop a combination of some features and some locations in order to minimise the quantity of data left. Nevertheless the models tested appeared to be quite able to cope with the unbalanced information and limiting to a subset actually worsened the predictions.

We confirmed how the addition of polynomial features must always be coupled with appropriate regularisation techniques to avoid heavily over-fitting models.

Although the main objective was prediction, we also managed to gather important interpretation insights by looking at feature importance and also by testing alternative strategies for choosing input data.

Finally we can say that weather prediction is definitely not an easy problem.

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

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