ENS Lyon
2023/2024

Titouan Vayer

TD n° 1 : Kernels for ML

- Exercise 1: Kernel ridge regression for temperature trends forecasting -

The aim of this third exercise is to model the evolution of the temperature of certain countries over time, using the tools of kernel theory. In particular, we will see:

- Why in ML you can't do just anything.
- Why assumptions about the data and the model are important.
- Why managing hyperparameters is tricky.

To do this, we'll be collecting data from Berkeley Earth (http://berkeleyearth.lbl.gov). To avoid long and tedious work, you can download the data at https://github.com/leouieda/global-temperature-data/tree/main/data. Once this is done, you can use the following code to load the data:

```
# We will need the following libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
cmap = plt.cm.get_cmap('tab10') # nice cmap for figures
from os import listdir
from os.path import isfile, join
from sklearn.kernel_ridge import KernelRidge

data_path = './data/temperature' # where data is stored
all_countries = []
for f in listdir(data_path):
    if isfile(join(data_path, f)) and f.endswith('.csv'):
        all_countries.append(f.replace('.csv',''))
print(all_countries)

country_name = 'peru' # choose a country
assert country_name in all_countries

# Load the data
temp = pd.read_csv(data_path+'/'+country_name+'.csv', names=['date','temp'])
temp.head()

X = np.array(range(temp.date.shape[0])).reshape(-1,1)
y = np.array(temp.temp)
```

The time series in the dataset are pairs $(y_i, x_i) \in \mathbb{R}^2$ where y_i is the temperature value at time x_i. The aim of this exercise is to find a function $f : \mathbb{R} \to \mathbb{R}$ that models the evolution of temperature over time.

(i) Show a time series from a country of your choice. How many points does it contain? What can you tell at first glance about the appearance of this series?

(ii) In the dataset, what is the variable to be predicted? What is the input variable?

(iii) Divide the dataset into two sets, train and validation (temporally coherent), with a parameter that adjusts the relative size of these two sets (called, for example, *ratio*).

Initially, we'll just try to predict the trend in these temperatures. We'll be looking for a polynomial regression of degree 2, i.e. functions of the type $f(x) = a_0 + a_1 x + a_2 x^2$.

(iv) This polynomial regression corresponds to a ridge regression with a particular kernel: which one?

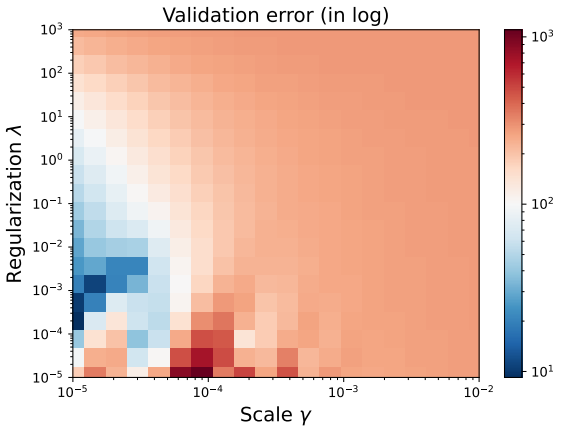
(v) Using `KernelRidge` train a polynomial regression model of degree 2 taking a validation ratio = 50% train and then with a validation ratio = 20% train. What can you conclude?

We will now use a Kernel Ridge Regression (KRR) model with a Gaussian kernel. $\kappa(t, t') = \exp(-|t - t'|^2/(2\gamma^2))$.

(vi) What is, a priori, the advantage of such a model?

(vii) Calculate and display the Gram matrix associated with this core for multiple values of γ. How do you interpret this hyparameter? What other hyperparameter should be taken into account in a KRR model?

(viii) With a train/validation ratio of 0.8, calculate the validation performance of each of the associated models on a given parameter grid. You can use `from sklearn.model_selection import ParameterGrid`. The idea is to obtain a figure resembling the following:



(ix) Choose the hyperparameter pair that gives the smallest error. What would it take to make this choice properly? (We'll do it at the end).

(x) Project the predictions to a 2050 horizon. What can you conclude? What should be taken into account in the model to improve it?

(xi) (Bonus question on this part) Do it all again without the help of scikit-learn.

This time we will repeat the same analysis with regressors of the form

$$f(x) = \sum_{k=1}^{K} a_k \cos(2\pi k f_0 x) + a_{K+1} x + a_{K+2} x^2 \,,$$

where $a_1, \cdots, a_{K+2}$ are to be optimized.

(xii) What are the hyperparameters of this model? Rewrite the problem of estimating $(a_k)_{k \in [\![K]\!]}$ as a linear regression problem (penalizing with a ℓ_2^2 norm).

(xiii) Follow the same procedure as above, implementing this function. You may find inspiration in the following class.

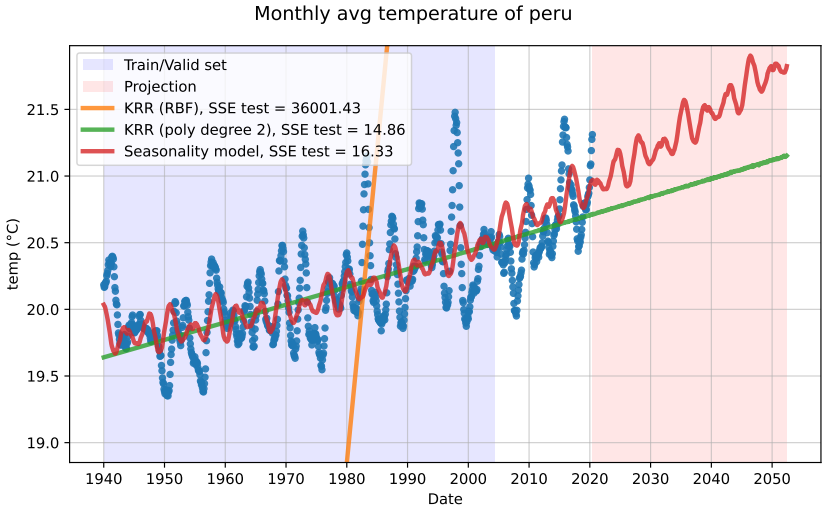


Figure 1: One possible result.

```python
from sklearn.linear_model import Ridge

class SeasonalityModel():
    def __init__(self, K=3, w0=12, alpha=1.0):
        self.K = K
        self.w0 = w0
        self.alpha = alpha
        self.model = Ridge(alpha=self.alpha, fit_intercept=False)

    def Phi(self, X):
        Y = np.zeros((X.shape[0], 1+2+self.K))
        Y[:,0] = np.ones(X.shape[0])
        Y[:,1] = X.ravel()
        Y[:,2] = X.ravel()**2
        for k in range(1, self.K+1):
                Y[:,2+k] = np.cos((2*k*np.pi*X.ravel()*self.w0))
        return Y

    def fit(self, X, y):
        Xtransfo = self.Phi(X)
        self.model.fit(Xtransfo, y)

    def predict(self,X):
        Xtransfo = self.Phi(X)
        return self.model.predict(Xtransfo)
```

(xiv) Conduct a cross-validation approach for selecting hyperparameters. A possible final result is shown in Figure 1. How could the model be improved? You can use the code below.

```
def frozendict(d: dict):
    keys = sorted(d.keys())
    return tuple((k, d[k]) for k in keys)

def do_cv(param_grid, X_in, y_in, model, n_splits=3):
        tscv = TimeSeriesSplit(n_splits=n_splits)
        results = {}
        for j, dic_param in enumerate(param_grid):

                instantiated_model = model(**dic_param)
                cv_errors = []
                for i, (train_index, valid_index) in enumerate(tscv.split(X_in)):
                        X_train, y_train = X_in[train_index], y_in[train_index]
                        X_valid, y_valid = X_in[valid_index], y_in[valid_index]

                        instantiated_model.fit(X_train, y_train)
                        ypred = instantiated_model.predict(X_valid)
                        err_ = mean_squared_error(y_valid, ypred)
                        cv_errors.append(err_)

                results[frozendict(dic_param)] = np.mean(cv_errors)
                print('Parameter {} done ..'.format(dic_param))
                print('--- {0:.0%} finished ---'.format(j/len(list(param_grid))))
        return results
```