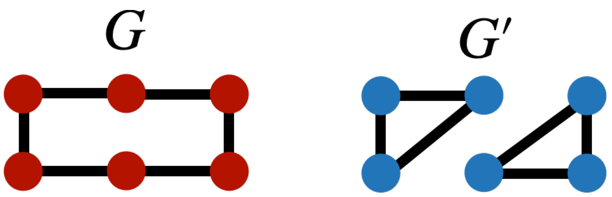

Figure 1: Two graphs

ENS Lyon
2023/2024

Titouan Vayer

TD : Kernels for ML

- Exercise 1: A proof for the positive definiteness of the Gaussian kernel -

The purpose of this exercise is to show that the Gaussian kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|_2^2/2\sigma^2)$ is a PD kernel for any $\sigma > 0$. In the following $\kappa_1, \kappa_2, \cdots$ are fixed PD kernels.

(i) Show that $\gamma\kappa_1$ for any $\gamma > 0$ is a PD kernel.

(ii) Show that $\kappa_1 + \kappa_2$ is a PD kernel.

(iii) Suppose that $\kappa(\mathbf{x}, \mathbf{y}) := \lim_{m \to +\infty} \kappa_m(\mathbf{x}, \mathbf{y})$ exists for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Show that it defines a PD kernel.

(iv) Consider two $n \times n$ PSD matrices $\mathbf{K}_1, \mathbf{K}_2$ and the matrix $\mathbf{K}_1 \odot \mathbf{K}_2$ defined by $\forall (i,j) \in [\![n]\!]^2$, $[\mathbf{K}_1 \odot \mathbf{K}_2]_{ij} = [\mathbf{K}_1]_{ij}[\mathbf{K}_2]_{ij}$ (this is known as the *Hadamard product* of two matrices). Show that $\mathbf{K}_1 \odot \mathbf{K}_2$ is a PSD matrix (this result is known as the Schur product theorem).

(v) Deduce that $\kappa(\mathbf{x}, \mathbf{y}) := \kappa_1(\mathbf{x}, \mathbf{y})\kappa_2(\mathbf{x}, \mathbf{y})$ is a PD kernel.

(vi) Consider $f : \mathcal{X} \to \mathbb{R}$ then show that $\kappa(\mathbf{x}, \mathbf{y}) := f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{y})f(\mathbf{y})$ is a PD kernel.

(vii) From the previous answers prove that $\kappa(\mathbf{x}, \mathbf{y}) := \exp(-\|\mathbf{x}-\mathbf{y}\|_2^2/2\sigma^2)$ is a PD kernel.

(viii) Consider a $n \times n$ PSD matrix $\mathbf{A}$ and a $m \times m$ PSD matrix $\mathbf{B}$. We define the tensor $\mathbf{A} \otimes \mathbf{B}$ as the $nm \times nm$ matrix defined by $\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} A_{11}\mathbf{B} & \cdots & A_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ A_{n1}\mathbf{B} & \cdots & A_{nn}\mathbf{B} \end{bmatrix}$. Show that $\mathbf{A} \otimes \mathbf{B}$ is a PSD matrix. Deduce that if $\kappa_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel on $\mathcal{X}$ and $\kappa_2 : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a PSD kernel on $\mathcal{Y}$ then $\kappa((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) := \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{y}, \mathbf{y}')$ is a PSD kernel on $\mathcal{X} \times \mathcal{Y}$.

- Exercise 2: WL test of isomorphism -

Show that the Weisfeiler-Lehman test of isomorphism cannot distinguish the two graphs in Figure 1.

- Exercise 3: 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()
```

The time series in the dataset are pairs $(y_i, t_i) \in \mathbb{R}^2$ where y_i is the temperature value at time t_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 $f(t) = a_0 + a_1 t + a_2 t^2$.

(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 the left plot of Figure 2.

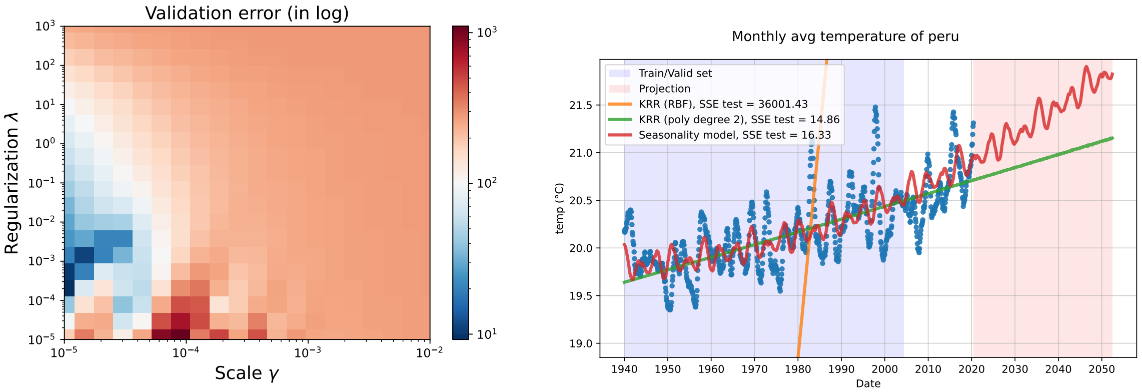

Figure 2: (left) CV errors w.r.t. parameters of the model (right) temperature predictions for different models.

(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) For this choice, calculate the prediction on all the data. What can you conclude?

(xi) Repeat the same procedure, playing with the finesse of the hyperparameter grid. What can you conclude?

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

(xiii) (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

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

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

(xiv) 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).

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

```
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)
```

(xvi) Conduct a cross-validation approach for selecting hyperparameters. A possible final result is shown in Figure 2. How could the model be improved?

- Exercise 4: Graph kernels for graphs classification -

In this exercise we will have access to a dataset of graphs $(G_1, y_1), \cdots, (G_n, y_n)$ where y_i are classes and we want to learn a function that takes a graph as input and output a label/class. This a problem of classification on the space of graphs. We will use the framework of graph kernels and the GraKel library. You can find the data here https://chrsmrrs.github.io/datasets/docs/datasets/ and a description of it here https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets.

(i) First download the MUTAG dataset in here https://chrsmrrs.github.io/datasets/docs/datasets/ in .zip format. You can load the data afterwards using the following code.

```
import numpy as np #we will need the following libraries
from grakel.datasets import fetch_dataset
from grakel.datasets.base import read_data
from grakel.kernels import WeisfeilerLehman, VertexHistogram
import zipfile

name = "MUTAG"
verbose = True
path = './data' # where data is stored
with zipfile.ZipFile(path+"/" + str(name) + '.zip', "r") as zip_ref:
    if verbose:
        print("Extracting dataset ", str(name) + "..")
    zip_ref.extractall()

if verbose:
    print("Parsing dataset ", str(name) + "..")

dataset = read_data(name,
                    with_classes=True,
                    prefer_attr_nodes=False,
                    prefer_attr_edges=False,
                    produce_labels_nodes=False,
                    is_symmetric=False,
                    as_graphs=True
                    )
G = dataset.data
y = dataset.target
```

(ii) Do a quick inspection of the dataset and labels: what do the data represent? how many classes? what can you conclude? What do you need to consider?

(iii) Using the following code based on the `networkx` library plot different training points

```
import networkx as nx
import matplotlib.pyplot as plt
m = 12
nx_g = nx.from_numpy_matrix(G[m].get_adjacency_matrix())
nx_g.add_edges_from(G[m].get_edges())
i=0
for _,v in G[m].node_labels.items():
    nx.set_node_attributes(nx_g, {i : {'label':v}})
    i+=1
cols = {0:'r', 1: 'b', 2:'g'}
pos=nx.layout.kamada_kawai_layout(nx_g)
nx.draw_networkx(nx_g,
               with_labels=True,
               labels=nx.get_node_attributes(nx_g,'label'),
               node_color=[cols[v] for k,v in
               nx.get_node_attributes(nx_g, 'label').items()]
               )
```

(iv) With `scikit-learn` splits the dataset into a training and a test set (you can use `from sklearn.model_selection import train_test_split`).

(v) The following code consider a Weisfeiler-Lehman graph kernel and train/test the model on the splits defined before. What is the kernel considered here? What do the parameters correspond to? What is the accuracy on the test set? Compare it to a simple naive baseline (you may use `from sklearn.dummy import DummyClassifier`).

```
wl_kernel = WeisfeilerLehman(n_iter=5, normalize=True,
                          base_graph_kernel=VertexHistogram)
K_train = wl_kernel.fit_transform(G_train)
K_test = wl_kernel.transform(G_test)
clf = SVC(kernel='precomputed')
clf.fit(K_train, y_train)
y_pred = clf.predict(K_test)
```

(vi) Implement a cross validation (CV) procedure with `StratifiedKFold` and compute the average CV score. Compare with the previous error. Is this a good measure of the generalization error of the complete model?

(vii) Implement a nested CV procedure (explanations on the board). You can use the following class.

```
class GK_classifier():
    def __init__(self, C=1, n_iter=5, normalize=True):
        self.C=C
        self.n_iter=n_iter
        self.normalize=normalize
        wl_kernel = WeisfeilerLehman(n_iter=self.n_iter,
                                  normalize=self.normalize,
                                  base_graph_kernel=VertexHistogram)
        self.graphkernel = wl_kernel
        self.svc=SVC(kernel='precomputed', C=self.C)

    def fit(self, X, y=None):
        K = self.graphkernel.fit_transform(X)
        self.svc.fit(K, y)
    def predict(self,X):
        K=self.graphkernel.transform(X)
        return self.svc.predict(K)
```