Data Challenge - Kernel Methods

Benjamin Deporte
benjamin.deporte@ens-paris-saclay.fr

Lilian Say
lilian.say@ens-paris-saclay.fr

Abstract

This report presents our results regarding the Data Challenge - Kernel methods of MVA class (2024-2025)

1 Code

Our code is available at `https://github.com/BenjaminDeporte/MVA_Kernel_Project`. It provides our work for the challenge, with a self-explanatory structure. The script **start.py** allows to reproduce our best submission.

2 Introduction

The goal of the project is to predict whether a DNA sequence will bind to a specific transmission factor. We chose to use only the raw DNA sequences for the project, and not the one-hot numeric encodings that were provided on the Kaggle site.

Section 3 presents the different kernels we coded and used for the project. Section 4 presents the algorithms we coded and used. Section 5 is a summary of the results of the different experiments. Last, section 6 provides some thoughts, feedback and discussion after the project.

3 Kernel

This section describes the two kernels we used for the project : **spectrum** and **mismatch**.

3.1 Spectrum

The kernel Spectrum [1] was presented in class, and we coded the exact algorithm. *k-mer* refers to a possible substring of length k in a DNA sequence x of length L, reading sequentially : for a given string, there are $L-k+1$ of those k-mers. The mapping of a sequence x if formally a vector $\Phi(x) = (\Phi_u(x))_{u\in\mathcal{A}^k}$ of dimension d equal to the number of all possible k-mers in the alphabet (here, $|\mathcal{A}|^k = 4^k$). $\Phi_u(x)$ is the number of occurences of the substring u in x.

The Python implementation relies heavily on the Collections library, which allows a pre-indexation of the substrings, and a -relatively- fast runtime.

3.2 Mismatch

The kernel Mismatch [2] was also presented in class. The idea is to allow possible mismatches, up to m characters, in each the substrings u being considered. This allows for taking into account genetic mutations on a given nucleïde. For $m=0$, we recover the Spectrum kernel.

We used the kernel mismatch with $m = 1$, mainly for reasons of computation time.

4 Kernel methods

This section is a presentation of the two methods we used : **KernelSVC** and **KernelLR**

4.1 KernelSVC

KernelSVC is a classifier based on the C-formulation of the SVM algorithm, presented in class. With the usual notations, the optimization problem boils down to:

$$\min_{\alpha\in\mathbb{R}^n} \frac{1}{2}\alpha^T K\alpha - \alpha^T y$$
$$\text{s.t } 0 \le \alpha_i y_i \le C,\ \forall i = 1, ..., n$$

The code used is an evolution of the code used for the homework 2.

4.2 KernelLR

We employ the kernel iteratively reweighted least squares (IRLS) method to solve the standard linear logistic regression discussed in class. Specifically, we iteratively solve a weighted kernel ridge regression until convergence to address the kernel logistic regression problem.

5 Results

The use of the KernelSVC classifier, along with the kernel Spectrum, allowed to produce fair results. The hyperparameters of the setup were the regularization C of the KernelSVC, and the length k of the substring considered in the kernel Spectrum. We ran a grid search to find the best pair of parameters, keeping 20% for validation purposes. As a result, we ran the final submission with $C = 2.0$ and $k = 7$

k	C	average validation accuracy
7	0.5	64.2%
7	1.0	64.5%
7	1.5	64.8%
7	2.0	65.0%
7	2.5	65.0%
7	3.0	64.8%

Table 1: Final validation accuracy on different hyperparameters of C on SVM

We applied kernel logistic regression with the spectrum kernel, achieving reliable results. The main hyperparameters were the substring length k in the spectrum kernel and the regularization parameter λ, though we observed that λ had little impact on the results. The final grid search, led us to select $\lambda = 0.1$ and $k = 7$ for the final model.

k	λ	average validation accuracy
7	0.01	62.0%
7	0.1	62.0%
7	1.0	62.0%
7	10	62.0%
5	0.1	57.6%
6	0.1	57.9%

Table 2: Final validation accuracy on different hyperparameters of k and λ on KLR

6 Feedback and discussion

Overall, we achieved a reasonable 66.4% accuracy on the public submission and 68.8% on the private submission with Kernel Logistic Regression and 64.9% accuracy and 66.3% on the private submission with Kernel SVM, which is a fair demonstration of the power of the kernel methods.

However, we feel we could have done better ! Testing other kernels (for example substring) or combinations of kernels. We spent a lot of time getting to grip with the following items:

- **scalability** - as the computation of the Gram matrix scales in n^2 and does not benefit from the use of a GPU, the computation times become too important beyond $N = 1000$ samples to run extensive optimizations.

- **importance of the optimization solver** - we regularly performed back to back testing of our kernel methods with the scikit implementation. Scikit runs on the libsvm library, which was not allowed for the challenge. Our code uses scipy.optimize.minimize which can use several solvers : SLSQP, BFGS, etc. We found differences in the outcomes of the computations of our codes (in terms of number of support vectors for example), and finally settled on SLSQP.

References

[1] C. S. Leslie, E. Eskin, and W. Noble, "The spectrum kernel: A string kernel for svm protein classification," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, vol. 7, pp. 564–75, Feb. 2002.

[2] C. S. Leslie, E. Eskin, A. Cohen, J. Weston, and W. S. Noble, "Mismatch string kernels for discriminative protein classification," *Bioinformatics*, vol. 20, no. 4, pp. 467–476, Mar. 2004.