Kernel methods - Data Challenge : DNA sequence binary classification

Gauthier Avite, Julien Genzling
Master MVA | March 2025
gauthier.avite@polytechnique.edu, julien.genzling@polytechnique.edu

1. Introduction

The goal of the challenge is to implement from scratch, kernel-based binary classification methods, to predict whether a DNA sequence is a binding site to a specific transcription factor. We are given 3 datasets of 2000 labeled sequences of 101 nucleotides. For each dataset, we have to predict the labels of 1000 unlabeled sequences.
We describe the methods we used to solve this task and their implementation, and then we present our results. The code is available at `https://github.com/JulienGenzling/KMProt` and you can reproduce the results of our best submission by following the instructions given in the `README.md` file.

2. Methods and Implementation

Let $\mathcal{X}$ be the space of DNA sequences and $\mathcal{Y} = \{-1, 1\}$ be the binary labels. Given a training set $\mathcal{D}_{\text{train}} = (x_i, y_i)_{i=1}^n$, kernel-based methods implicitly map inputs to a feature space $\mathcal{F}$ via $\phi : \mathcal{X} \to \mathcal{F}$ and define a kernel function:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{F}}$$

The kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ has elements $\mathbf{K}_{ij} = K(x_i, x_j)$ and we normalize it so that:

$$\mathbf{K}_{\text{norm}}(i, j) = \frac{\mathbf{K}_{i,j}}{\sqrt{\mathbf{K}_{i,i}\mathbf{K}_{j,j}}}$$

We then fit a classifier to get $(\alpha_i)_{i=1}^n$ and b so that the prediction for a new sequence x is:

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i K(x_i, x) + b\right)$$

2.1. Kernels

We first conducted some rapid experiments using basic kernels operating on the embedding vectors given in the dataset. Given that the results weren't satisfactory, we didn't try to optimize them and focused on string kernels. We focused on 2 different string kernels: the spectrum kernel and the mismatch kernel.

For the k-spectrum kernel [1], we have:

$$K(x_i, x_j) = \sum_{u \in \mathcal{A}^k} \phi_u(x_i) \cdot \phi_u(x_j)$$

where $\mathcal{A}^k$ is the set of all 4^k possible k-mers and ϕ_u counts the number of occurences of u in x. In practice we don't store vectors of length 4^k in memory (can take up to a few GB of RAM for $k = 15$ and billions of operations) but dictionaries (roughly a few KB per sequence in RAM and $\mathcal{O}(87)$ in time complexity).

In our code, we implemented a multi-spectrum kernel which just sums k-spectrum kernels:

$$K(x_i, x_j) = \sum_{k=k_{\min}}^{k_{\max}} \sum_{u \in \mathcal{A}^k} \phi_u(x_i) \cdot \phi_u(x_j)$$

For the mismatch kernel [2], we have:

$$K(x_i, x_j) = \sum_{u \in \mathcal{B}_m^k} \phi_u(x_i) \cdot \phi_u(x_j)$$

where $\mathcal{B}_m^k$ is the set of all possible k-mers with up to m mismatches. For high values of m, the dictionaries can be very large (up to 10^6 for $k = 15$ and $m = 3$) and the dot product requires $\mathcal{O}(n^2)$ lookups and comparisons so we fasten the computation of the Gram matrix by organizing them in sparse matrices with `scipy.sparse`.

We also implemented a weighted sum kernel, defined as:

$$K(x_i, x_j) = \sum_{u=1}^{p} \beta_u K_u(x_i, x_j)$$

where the $(K_u)_{u=1}^p$ are the best p kernels and the $(\beta_u)_{i=u}^p$ are weights defined based on the performance of the associated kernel.

2.2. Classifiers

For the classifiers, we first implemented Kernel Ridge Regression and Kernel Logistic Regression but these methods had very poor results. Therefore we implemented the dual SVM which solves:

$$\max_{\substack{0\leq\alpha_i\leq C \\ \sum_{i=1}^{n}\alpha_i y_i=0}} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i,x_j)$$

We found that SVM largely outperforms KLR and KRR probably because transcription factors typically recognize short DNA motifs (6-12 nucleotides) within our 101-nucleotide sequences, creating low signal-to-noise ratio conditions where SVM's margin-based approach is well suited. Additionally, SVM handles the high-dimensional feature spaces generated by our k-mer representations more effectively, making it better suited for capturing sparse binding patterns.

2.3. Pipeline

During this challenge, we developed a comprehensive custom pipeline from scratch, motivated by the constraint against using existing libraries.
Our pipeline is built around a modular architecture where `Dataset`, `Fitter`, and `Kernel` objects are configured through a central `config.json` file. The `CrossValid` object executes k-fold cross-validation, measuring performance via mean accuracy across folds. For hyperparameter optimization, the `Gridsearch` object efficiently distributes cross-validation experiments across multiple cores to evaluate various kernel and classifier configurations. The `EnsembleSearch` object then leverages these results to create an optimized weighted kernel, combining the best-performing configurations based on their validated performance metrics.

3. Results

We conducted a comprehensive hyperparameter optimization for our SVM model, exploring the regularization parameter C, the multi-spectrum kernel parameters k_{min} and k_{max}, and the mismatch kernel parameters k and m. Our analysis determined that $C = 1$ yielded optimal performance.

Figure 1 shows our k_{min} and k_{max} parameter sweep results, revealing distinct binding patterns. The first transcription factor binds to variable-length sequences (from $k_{\text{min}} = 5$), indicating binding flexibility and explaining its lower prediction accuracy. The second shows clearer preferences: longer sequences for dataset 2 and mid-length sequences with narrower distribution for dataset 1, correlating with higher accuracy. These patterns suggest relative transcription factor lengths: dataset 2 (longest), dataset 1 (medium), and dataset 0 (shortest).

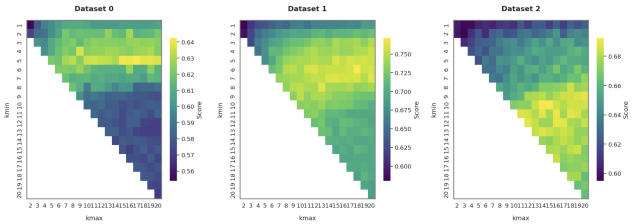

Figure 1: k_{min} and k_{max} parameter sweep for multi-spectrum kernel

We obtained our best submission with a weighted sum kernel defined earlier, of the following kernels (see `best_config.json` file). $M(k,m)$ stands for mismatch kernel and $S(k_{\text{min}}, k_{\text{max}})$ for the multi-spectrum kernel:

- Dataset 0 : $M(10,2), S(5,9), S(5,17), S(5,17), S(5,16)$
- Dataset 1 : $S(6,18), M(10,1), S(8,13), S(7,15), S(7,20)$
- Dataset 2 : $S(10,15), S(10,14), S(11,16), S(12,16), S(11,14)$

You can see the results in table 1.

Table 1: Our 2 best submissions. The second one is the ensemble and the first one was obtained by using only the first kernels cited in the above lists of kernels. We can see that the first one has better CV results but is less robust than the ensemble since it performs worse on the LB.

Dataset	Local CV	Public LB	Private LB
0	0.6450	–	–
1	0.7715	0.7087 (5/37)	0.7193 (2/37)
2	0.6925	–	–
0	0.6325	–	–
1	0.7655	0.7087 (5/37)	0.7227 (2/37)
2	0.6885	–	–

4. Conclusion

Building a pipeline and testing different kernels taught us a lot. Our second-place finish came from having a solid local validation setup and thorough hyperparameter search, made possible by good code organization.

References

[1] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: a string kernel for svm protein classification. *Pacific Symposium on Biocomputing*, pages 564–575, 2002. PMID: 11928508.

[2] Christina Leslie, Eleazar Eskin, and William Stafford Noble. Mismatch string kernels for svm protein classification. *Bioinformatics*, 20(4):467–476, 2004.