Kernel Methods Challenge- Report
Team: The WAY

Wassim BOUATAY - Amrou CHOUCHENE - Yassine NAJI
github.com/WassimBouatay/-MVA-Kernel-Methods-Challenge

Public Score: 0.66666 - Private Score: 0.65533

Abstract

The goal of this challenge is to predict whether a DNA sequence region is a binding site to a specific transcription factor or not. We were given 3 datasets, each has a different transcription factor. In this report, we will detail our work and the different kernels that we used.

1. Kernels

DNA sequences are strings. We need to map these strings into a vector of $\mathbb{R}^d$ in order to classify and analyze them. This is exactly why the kernel method was designed for. There is a lot of works in the literature and a lot of kernels designed specifically for DNA sequences.

As we saw in the course, for every positive definite kernel K there exists a mapping ϕ that transforms the data from an initial space to another (in our case from a string to $\mathbb{R}^d$) such that $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$.

1.1. Spectrum Kernel

First, we started with this simple kernel. For every string s we create a vector of size 4^k which corresponds to all the possible substrings of size k using the 4 letters {'A', 'C', 'G', 'T'}.

The mapping $\phi(s) = (\phi_u(s))_{u \in U}$ where U is the set of all possible substrings of size k, and $\phi_u(s)$ is the occurence of u in the string s.

We added a normalization factor making the spectrum kernel as follows:

$$K(x, y) = \frac{1}{\|\phi(x)\| \, \|\phi(y)\|} \sum_{u \in U} \phi_u(x) \phi_u(y)$$

In the following table we show the results on training and validation for each dataset using $k = 8$.

	Dataset 1	Dataset 2	Dataset 3
Training accuracy	100.0%	100.0%	100.0%
Validation accuracy	62.5%	62.5%	75.5%

We changed a bit the idea of spectrum kernel. Instead of calculating the occurrences of substrings of a fixed size k, we calculated the occurrences of all substring of $k \leq d$. Using $d = 7$ we got the following results.

	Dataset 1	Dataset 2	Dataset 3
Training accuracy	87.3%	88.3%	91.1%
Validation accuracy	66.0%	66.5 %	71.5%

1.2. Mismatch Kernel

As we saw previously, the representation used in the Spectrum Kernel is to treat DNA sequence segments as combinations of k adjacent nucleotide residues. However, we thought that adding some flexibility to our Kernel representation could improve the results, especially when dealing with biological problems because using exact kernel matching in a biological sequence could lead to some errors. For this reason, we implemented the Mismatch Kernel. We implemented the concept of (k, m)-mismatch kernels, which considers k-mer counts with m inexact matching of k-mers as discussed in [1]. For the Mismatch Kernel, we have:

$$K(x, y) = \sum_{\gamma \in \Gamma_k} c_x^{k,m}(\gamma).c_y^{k,m}(\gamma)$$

$$c_x^{k,m}(\gamma) = \sum_{g \in S(\gamma, m, k)} c_x(g)$$

where γ is a k-mer, $c_x(\gamma)$ is the number of occurrences of k-mer γ in sequence x, Γ_k is the set of all possible k-mers and $S(\gamma, m, k)$ is the set of contiguous substrings of length k that differ from γ in at most m-characters.
However, we were sadly surprised by the huge computational time of computing this kernel, especially when computing the set $S(\gamma, m, k)$ and $c_x^{k,m}(\gamma)$. For instance, for m=1 and k=4, this algorithm takes 2 hours in order to compute the Kernel for 2000 input data points. We mention also that if m increases, the computational time increases a lot.
The best validation accuracy was obtained for m=0. But since running the Mismatch Kernel for m=0 is equivalent to running the Spectrum-Kernel, we used m=1 or m=2 which gave low validation accuracy. The following table summarizes the results for m=1 and k=4:

	Dataset 1	Dataset 2	Dataset 3
Validation accuracy	54.0%	55.5%	52.0%

1.3. Weighted Degree Kernel

This is also similar to the spectrum kernel. But in the Weighted Degree kernel, we only focus on the correspondence of substrings at the same position. We also take all the substrings of length $\leq d$. The kernel is defined as follows:

$$K(x, y) = \sum_{k=1}^{d} \beta_k \sum_{l=1}^{L-k} 1_{\{u_{k,l}(x) = u_{k,l}(y)\}}$$

where β_k is the weight of substrings of size k, L is the length of the string and $u_{k,l}(x)$ is the substring of x of size k and starting at index l. We take $\beta = \frac{2(d-k+1)}{d(d+1)}$ (we give longer matches less weights because they already imply many shorter weights).

In the following table we show the results on training and validation for each dataset. For the testset we got 60.4%. These values are given by taking $d = 6$.

	Dataset 1	Dataset 2	Dataset 3
Training accuracy	84.2%	84.1%	81.7%
Validation accuracy	64.5%	54.3%	68%

1.4. Smith-Waterman Kernel

In order to take into consideration the best alignment between 2 sequences, we implemented the Smith Waterman Kernel which is based on the SW algorithm explained in [2]. We defined the similarity score S matrices as $s_{i,j} = 3$ if $i = j$ and $s_{i,j} = -3$ elsewhere. We chose a gap penalty matrix W which is linear with a slop of 2. The value of the kernel between 2 sequences is the maximum score given by the matrix H, which represent the score of the best alignment:

$$K(x_u, x_v) = \max H^{uv}$$

Where :

$$H^{uv}_{i0} = H_{0j} = 0 \quad for \quad 0 \le i, j \le n$$

and

$$H^{uv}_{ij} = \max \begin{cases} H^{uv}_{i-1,j-1} + S(x_{u,i}, x_{v,j}), \\ \max_{k\ge1}\{H^{uv}_{i-k,j} - W_k\}, \\ \max_{l\ge1}\{H^{uv}_{i,j-l} - W_l\}, \\ 0 \end{cases}$$

This kernel is symetric but not definite positive.

We implemented a Dynamic Programming version of the algorithm on Python with some optimizations to reduce the complexity of computing such kernel, but it remains quite expensive $O(n^2)$ where n is the length of the sequence, it takes around 100ms to compute the kernel of a pair (thus around 55h for each dataset!). To further speed up our computation, we used the Numba JIT compiler which allows parallelization and fast compilation. The execution time dropped drastically to 5ms, so we needed only 3h to compute the full kernel matrix for each dataset.

The following table shows the results of a 10-folds cross-validation :

	Dataset 1	Dataset 2	Dataset 3
Training accuracy	76.5%	75.0%	80.2%
Validation accuracy	63.5%	59.1%	70.0%

1.5. Sum kernel

We have seen in the course that combining kernels by summing them may help improving performances, since it's equivalent to represent data in the corresponding RKHS by a concatenation of representations. Thus, we took our best 3 kernels : $Spectrum_{1\le k\le 7}$, $Spectrum_{k=8}$ and SW, and we normalized them by dividing by the maximum value of the kernel on the dataset (strictly positive in our case). We had been able to improve the validation scores for the datasets 1 and 3 as we can see in the following table :

	Dataset 1	Dataset 2	Dataset 3
Training accuracy	96.2%	99.3%	96.2%
Validation accuracy	66.6%	65.6%	72.3%

2. Classifiers

2.1. Ridge classifier

We implemented the Ridge classifier which minimizes the following objective:

$$\min_{\alpha\in\mathbb{R}^n} \quad \frac{1}{n}\sum_{i=1}^{n} \ell(y_i[K\alpha]_i) + \frac{\lambda}{2}\alpha^T K\alpha$$

In the case of least-squares-loss, we compute directly α using the closed formula :

$$\alpha = (K + n\lambda I)^{-1} y$$

2.2. SVM

We implemented the SVM which minimizes the following objective:

$$\min_{\alpha\in\mathbb{R}^n} \quad \frac{1}{2}\alpha^T K\alpha - \alpha^T y$$

such that for every i, $0 \le y_i\alpha_i \le C$

We used a QP solver from the CVXOPT library. SVM gave better results compared to the Ridge Logisitic classifier

We used SVM for all the kernels. Moreover, all the mentioned parameters of the kernels as well as the parameter C were fine tuned using k-fold cross validation.

3. Results

We tried to combine several kernels with different methods. For instance, We tried an ensemble kernel, where we make a voting for the majoritarian class among the predictions of the 5 kernels.

The results of all methods on the public test set are presented in the following table:

Method	Public Score	Private Score
Ensemble Kernel	**66.6%**	**65.5%**
Spectrum Kernel $k \le 7$	65.6%	65.2%
Sum of kernels	64.8%	62.8%
SW Kernel	64.1%	64.0%
Spectrum kernel $k = 8$	62.7%	62.8%
WD Kernel $d = 6$	60.4%	59.6%

4. Conclusion

In this challenge, we tried different approaches and different kernels in order to classify DNA sequences. The best result was achieved using the ensemble method on different kernels, which gave 0.65533 accuracy on the private leaderboard.

We learned a lot from this challenge. Despite the difficulty of the task as we were not allowed to use any machine learning tool, it was very interesting and we enjoyed implementing all the methods by ourselves from scratch.

References

[1] *Transfer String Kernel: Mismatch-Kernel section*. URL: https://arxiv.org/pdf/1609.03490.pdf.

[2] *Smith–Waterman algorithm*. URL: https://en.wikipedia.org/wiki/Smith%5C%E2%5C%80%5C%93Waterman_algorithm.

Appendix - Not used approach

Deep Kernel:

In order to learn an optimal a representation $\phi(x)$, we used a deep learning approach which consists of a auto-encoder, for which we consider the latent space as our RKHS where x is represented by $\phi(x)$. We introduced the decoder in the architecture in order to have a representation which characterise fully the data point x, so that we are able to recover x from it, and do not simply overfit on the few training samples.

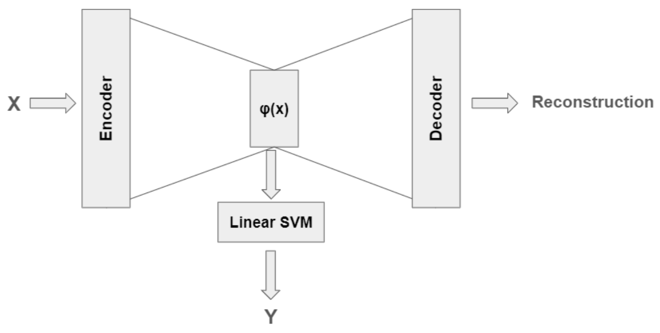

Figure 1: Auto-encoder

We encoded the sequences into matrix of size: $\mathbb{R}^{4,100}$, by performing 1 hot encoding to characters. The encoder architecture is composed of 5 layers of stridden 1D convolutions of kernel size 3 with maxpooling of size 2 and ReLU activation. The decoder is composed of 5 layers of strided transposed convolutions with ReLU activations. The linear SVM is simply a 2 layer perceptron with no activation.

The loss function we considered is simply the sum of the BCE loss of the SVM and the BCE loss of the auto-encoder.

We tested this approach on the first dataset with a random validation split of 20%, we had been able to reach a validation accuracy of 65% in dataset 1. However, **we didn't submit on Kaggle using this method** because we implemented it using the Pytorch library.