Kernel Methods Challenge: DNA Classification

Team Name: MTM
Margaux Törnqvist, Théo Uscidda and Mohammad Fahes

Public Rank (resp. Private): 3 / 74 (14/74)
Public Score (resp. Private): 68.333% (67.066%)

March 26, 2021

1. Introduction

Transcription factors (TFs) are proteins that bind specific sequence motifs in the genome to activate or repress transcription of target genes. This data challenge consisted in predicting whether a DNA sequences is a binding site for a given TF or not. The classification into bound (label 1) or unbound (label 0) was done on 3 different datasets each composed of 2000 DNA sequences of length 101 nucleotides belonging to $\{A, C, G, T\}$. This challenge brought us to explore kernel methods for classification and therefore implement several kernels and typical classifiers detailed in the following sections.

2. Methods

After have explored different classification methods applied to the bag-of-words representation of the data-sets, we decided to move on building kernels adapted to strings.

2.1. Kernels for strings

Spectrum Kernel A classic kernel for DNA sequence classification is the spectrum kernel [2] which is defined on a input space of all finite length sequences built from an alphabet $\mathcal{A}$. Here $\mathcal{A} = \{A, C, G, T\}$. Let's recall, that the $k-$spectrum of a sequence is the set of all sub-sequences of length k, called k-sequence, that it contains. Our feature map Φ_k is indexed by all $k-$sequences a from alphabet $\mathcal{A}$ and denotes the number of occurrences of a in x without gaps:

$$\Phi_k(x) = (\Phi_a(x))_{a \in \mathcal{A}^k}$$

The $k-$spectrum is then :

$$K_k(x, y) = \langle\, \Phi_k(x), \Phi_k(y) \rangle$$

We implemented two versions of this kernel, one using dictionaries and one using arrays. The dictionary version had the advantage of storing only the sub-sequences present in sequences x and y to compute $K_k(x, y)$, and the array version to get access to each $\Phi_k(x)$.

Exponential Spectrum Kernel The exponential spectrum kernel is derived from the original spectrum kernel. What changes is the feature map Φ_k which denotes the Hadamard d_H distance between sequences a and x set therefore as the power of a parameter λ.

$$\Phi_a(x) = \lambda^{d_H(a,x)}$$

Somehow, we couldn't compute Gram matrices for our dataset due to the high complexity of our code which we didn't manage to optimize.

TF-IDF Spectrum Kernel Moreover, we got inspired by *NLP* to rethink the spectrum kernel. In particular we thought of using *TF-IDF*, which is a numerical statistic that is intended to reflect how important a word is to a document given a corpus. Thus, we though of computing the occurrences of sequences by weightening them by a factor giving information on how scarce or on the contrary how common this sequence is in our alphabet of $k-$sequences. However, we didn't really had time to explore how this method could improve our results in comparison with the classic spectrum.

Mismatch Kernel The feature map in (k, m)-mismatch kernel [3] goes from the space of all finite sequences from an alphabet $\mathcal{A}$ (as before $\mathcal{A}$ = {A,C,G,T} and $|\mathcal{A}|$=4) to the 4^k- dimensional vector space indexed by the set of k-length subsequences from $\mathcal{A}$ (k-mers). The idea is to allow some mismatches ($\leq m$) between the k-mers. For a fixed k-mer, we define the (k, m)-neighborhood generated by it ($N_{(k,m)}(\alpha)$), which is the set of all the sequences (of length k) whose characters are in $\mathcal{A}$ and that differ from our k-mer by at most m mismatches.
The feature map $\phi_{(k,m)}$ is defined as follows: $\phi_{(k,m)}(\alpha) = (\phi_\beta(\alpha))_{\beta \in \mathcal{A}^k}$ where $\phi_\beta(\alpha) = \begin{cases} 1 & \text{if } \beta \in N_{k,m}(\alpha) \\ 0 & \text{otherwise} \end{cases}$

So, for a sequence x,

$$\phi_{(k,m)}(x) = \sum_{k-mers\ \alpha\ in\ x} \phi_{(k,m)}(\alpha)$$

The (k, m)-mismatch kernel is then: $\mathrm{K}_{(k,m)}(x, y) = \langle\, \phi_{(k,m)}(x), \phi_{(k,m)}(y) \rangle$

2.2. Kernels combinations

Sum of Kernels As each k-kernel gives a representation of our sequences in their respective *RKHS*, it could be in-

teresting to combine them in order to have a even more detailed representation of our sequences. Concatenating feature vectors from different *RKHS* is equivalent to consider the feature vector of the sum of their respective associated representative kernels. More precisely if we consider the following sum of p kernels $K_S = \sum_i^p K_i$, the associated feature vector is :

$$\Phi_S(x) = (\Phi_1(x), ..., \Phi_M(x))^T$$

We tried to sum different types of kernels: Gaussian, spectrum and mismatch. It seemed that mixing the types of kernels wasn't beneficial. We then restricted ourselves to sum kernels from the same type. More precisely, the best results were obtained by summing Mismatch kernels, but we also report in section 3 results obtained with sums of Spectrum kernels. The combinations were found by cross-validation, as detailed in the section 2.3.

Polynomial combinations of Kernels It then seems interesting to learn non-linear combinations of kernels, as in [1]. For a degree d and a weight vector $\mu \in \mathbb{R}^p$, we are interested in learning an optimal combination $K_\mu = (\sum_{i=1}^p \mu_i K_i)^d$. For instance, using KRR and adding a constraint on the weights vector μ, it can be formulated:

$$\min_{\mu \in \mathcal{M}} \max_{\alpha \in \mathbb{R}^n} -\alpha^\top (K_\mu + I)\alpha + 2\alpha^\top y \qquad (1)$$

To solve (1), we used the proposed *projection-based gradient descent* algorithm, setting $\mathcal{M} = \{\mu \geq 0, \|\mu\|_2 \leq 1\}$. For $d = 1$ we recovered an algorithm similar to `SimpleMKL` [4] that allows us to perform a kind of kernel selection. However, in both cases $d = 1$ or $d > 1$, it performed the same - or even worse - than simply summing the kernels. We then realized that summing the kernels and thus concatenating the kernel embeddings, although it seems naive, is efficient in practice, especially when it is not clear that we want to perform a kernel selection. Therefore, we restricted ourselves to the sum while combining kernels.

2.3. Classifiers and hyper-parameter tuning

Kernel Logistic Regression We implemented this classifier solving the induced primal problem. However, it did not improve KRR or KSMV results, although being slower (either solving it with `scipy.optimize` library or with a sequential weighted KRR). Therefore, we restricted ourselves to KRR and KSVM classifiers.

Kernel Ridge Regression We implemented this classifier solving the induced primal problem, which can be done in closed-form.

Kernel Support Vector Machine We implemented this classifier solving the induced dual problem - which is a quadratic program - with the *QP* solver of `cvxopt` library.

Hyper-parameter tunning All hyper-parameters were tuned using a systematic grid-search on a 5-fold cross-validation. When we used sums of kernels, we considered all possible combinations. For instance, when we were interested in summing Spectrum Kernels, we restricted ourselves to the k-mer sizes $k \in \{4, ..., 12\}$ and we tried all the sums of kernels $K_{\text{cand}} = \sum_{k=1}^q K_{i_k}$ for $\{i_1, ..., i_k\} \subset \{4, ..., 12\}$. This was done efficiently by calculating only once the Gram on the whole training set, then recovering for each fold the induced Gram matrix manipulating the indices of its elements. Moreover, each Gram matrix was calculated only once and saved in `.txt` format.

3. Results

Sum of Kernels		Xtr0	Xtr1	Xtr2
Spectrums	k	4,5,6	5,9,10	8,9,10
	λ	0.0001	0.1	1.0
	$\text{acc}_{val}(\%)$	62.1	63.7	70.3
	$\text{acc}_{test}(\%)$		62.06 *(64.20)*	
Mismatchs	k	8,9,10	4,8,10,11	4,6,9,11
	λ	2.9	2.3	1.6
	$\text{acc}_{val}(\%)$	66.25	67.2	77.45
	$\text{acc}_{test}(\%)$		66.666 *(68.333)*	

Note: The best score we obtained on the private leaderboard (67.066%) is due to the same model (sum of mismatch kernels) as above, but using the sum for k=7 and k=8 for Xtr0 instead of 8,9 and 10, and keeping the same kernels for Xtr1 and Xtr2.

4. Ideas for improvement

There are several potential ways to improve our results. One way is to try to use the mismatch kernels for $K \geq 12$. Indeed, we remarked that each time we computed mismatch kernels with higher values of k, we found new sums of those giving higher accuracies. Unfortunately, the method is memory consuming (when computing the (k,m)-neighborhood generated by a k-mer, where we have vectors of size 4^k), and requires a high computation power increasing with k when computing the inner product. One way to solve this is a mismatch tree data structure.
Another potential improvement could be in using ensemble methods such as majority voting, where each model makes a prediction for each validation instance and the final prediction is the one that receives more than half of the votes. This could lead to more "robust" prediction on the test set, and could improve the accuracy on the private leaderboard.

5. Conclusion

To conclude, this data challenge brought us to explore the world of kernels for sequences, to implement classic methods for classification such as KRR and KSVM and to learn how to optimize parameters of models. We were aware of the size of the dataset which could lead to some overfitting and tried to thus build as robust as possible models. If more complex kernels could be designed, we believe it's also important to consider the objets we manipulate, here DNA sequences, and thus use kernels which are adapted to our problem. Discussing with a biologist could be interesting in order to identify what is to him essential to look at in a DNA sequence to deduce whether a region is a binding site for a TF or not.

References

[1] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning non-linear combinations of kernels. 2009. 2

[2] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pages 564–575. World Scientific, 2001. 1

[3] Christina Leslie, Eleazar Eskin, Jason Weston, and William Stafford Noble. Mismatch string kernels for svm protein classification. *Advances in neural information processing systems*, pages 1441–1448, 2003. 1

[4] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. Simplemkl. *Journal of Machine Learning Research*, 9:2491–2521, 2008. 2