Kernels methods - Data challenge report - 2018 / 2019

Team: Triple Kerneliet. **Members**: Dorian Baudry & Alexandre Filiot. **Rank (public/private)**: $24/38$

1. Code

The `zip` file comes from our Github https://github.com/afiliot/Kernel_methods_MVA. It fully provides our work for this data challenge. The `README.md` file explains the structure of the repo and **contains some references**. Finally, `run.py` reproduces our best submission on private/public leaderboard.

2. Introduction

The goal here is to predict whether a DNA sequence region is binding site to a specific transcription factor. We only used the raw DNA sequences. As a starting point, one may describe some of the general practises adopted for this challenge. First, we made sure that the different data sets were balanced (no need for over/under-sampling techniques like ROSE or SMOTE). Then, as soon as some parameters needed to be optimized, this was done through cross-validation in order to avoid overfitting. Finally, we paid attention to the clarity and compactness of the code. In particular, we used the classical scheme in ML, `fit, predict, score`.

3. Kernel

We built different kernels based on our data and specifically to the 3 TFs. Aronszjan's theorem states that all of the following kernels are p.d. as for all DNA sequences $\mathbf{x}, \mathbf{x}'$, $K(\mathbf{x}, \mathbf{x}') = \langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle_{\mathbb{R}^p}$ where Ψ is a mapping from the space of DNA sequences to $\mathbb{R}$ and p is the length of feature vectors, which varies according to the method. We will not always make this mapping explicit but rather write $K(\mathbf{x}, \mathbf{x}')$ for $\mathbf{x}, \mathbf{x}'$ 2 DNA sequences.

3.1. Spectrum Kernel

SP(k) relates to the Spectrum Kernel for k-mers. The term k-mer refers to all $L - k + 1$ possible substrings of length k that are contained in the sequence $\mathbf{x}$ of length L. Here, $\Psi(\mathbf{x})_t = \sum_{i=1}^{L-k+1} \mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x})$ where $\Psi(\mathbf{x})$ is a d-dimensional vector with d equal to the number of possible k-mers (4^k here), $L = 101$ and where $\mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x})$ is the string of length k starting at position i of $\mathbf{x}$.

3.2. Weight Degree Kernel

WD(d) refers to the Weight Degree Kernel of order d. It compares 2 sequences $\mathbf{x}'$ and $\mathbf{x}$ by summing all contributions of k-mer matches of lengths $k \in \{1, ..., d\}$ weighted by coefficients $\beta_k = 2(d - k + 1)/(d(d + 1))$. Hence, $K(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \beta_k \sum_{i=1}^{L-k+1} \mathbf{1}(\mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x})) = \mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x}'))$.

3.3. Weight Degree Kernel with Shifts

WDS(d, s) refers to the Weight Degree Kernel of degree d with Shifts of size s. It allows to tolerate some positional variation of the matching k-mers up to shifts of s characters. Namely, $K(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \beta_k \sum_{i=1}^{L-k+1} \sum_{s=0,\, s+i \leq L}^{S} \delta_s \mu_{k,i,s,\mathbf{x},\mathbf{x}'}$ with $\mu_{k,i,s,\mathbf{x},\mathbf{x}'} = \mathbf{1}\{\mathbf{u}_{\mathbf{k},\mathbf{i}+\mathbf{s}}(\mathbf{x})) = \mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x}')\} + \mathbf{1}\{\mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x})) = \mathbf{u}_{\mathbf{k},\mathbf{i}+\mathbf{s}}(\mathbf{x}')\}$, $\beta_k = 2(d - k + 1)/(d(d + 1))$ and $\delta_s = 1/(2(s + 1))$.

3.4. Mismatch Kernel

MM(k, m) refers to the Mismatch Kernel for k-mers with m mismatches. For $m = 0$, we retrieve the k-spectrum kernel. Namely, $K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{L-k+1} \mathbf{1}(\mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x}')) \in N_{k,m}(\mathbf{u}_{\mathbf{k},\mathbf{i}}(\mathbf{x}))$ where $N_{k,m}(\boldsymbol{\alpha})$ is the set of all k-mers that differ from $\boldsymbol{\alpha}$ by at most m mismatches.

3.5. Local Alignment Kernel

LA((e, d, β)) refers to the LA Kernel with affine gap penalty $g(0) = 0, g(n) = d + e(n - 1)$ for $n \geq 1$. LA kernel can be computed by dynamic programming. We also implemented Smith-Waterman LA which considers maxima instead of sums. To tackle the diagonal dominance issue, we normalize the kernel with $1/\beta$ term. In order to make the normalized kernel p.d., we added spectral translation or empirical kernel map rules[1].

3.6. Substring Kernel

SS(k) refers to the substring kernel of order k. It exactly corresponds to the one studied in class. Namely, to follow the course notations, we have that $K(\mathbf{x}, \mathbf{x}') =$

[1] See Vert et al., "Convolution and local alignment kernels" (2004), p.11 for more details.

$\sum_{\mathbf{u}\in\Sigma_k}\sum_{\mathbf{u_{k,i}(x)}=\mathbf{u}}\sum_{\mathbf{u_{k,i}(x')}}\lambda^{l(i)+l(i')}$ where $l(i)=i_k-i_1-1$.

3.7. Gappy Kernel

GP(g,k) refers to the Gappy Kernel for k-mers and g defines the gap. More precisely, for a g-mer $\boldsymbol{\alpha}$, the gapped match set $G_{g,k}(\boldsymbol{\alpha})$ is the set of all k-mers t that occur in $\boldsymbol{\alpha}$ with $g-k$ gaps. Hence, $K(\mathbf{x},\mathbf{x}')=\sum_{i=1}^{L-k+1}\mathbf{1}(\mathbf{u_{k,i}}(\mathbf{x}'))\in G_{g,k}(\mathbf{u_{k,i}}(\mathbf{x}))$.

3.8. Normalization

To balance the weights of the kernels used in combining kernels methods, we always normalized the kernels. Namely, we applied the transformation $K_n(\mathbf{x},\mathbf{x}')=K(\mathbf{x},\mathbf{x}')/\sqrt{K(\mathbf{x},\mathbf{x}')K(\mathbf{x},\mathbf{x}')}$ which results in a p.d. kernel.

4. Kernel methods

This part is dedicated to the models used during the competition. Despite cross-validation, using only one different kernel for each data set was not sufficient. We thus used a two-stage (ALIGNF) algorithm first, then a one-stage (NLCK) one. Both techniques consist in combining kernels. This combination is rather learned (1-st.) or not (2-st.) during the training made by some kernel-based algorithms.

4.1. KLR, KRR, C-SVM

As an mandatory basis, we implemented Kernel Logistic Regression, Kernel Ridge Regression and C-SVM. All of these algorithms are inspired from the lecture notes (chapter "Supervised Learning"). We rapidly restricted our modelling to using only SVMs. To implement the latter, we used `cvxopt` (faster) or `scipy.Minimize` (with L-BFGS method).

4.2. ALIGNF

The first two-stage algorithm we used is `alignf`. It determines the mixture weights μ_k jointly by seeking to maximize the alignment between the convex combination kernel $K_\mu=\sum_{k=1}^p\mu_k K_k$ and the target kernel $K_Y=yy^\top$ for μ such that $\|\mu\|_2=1$. (Cortes et al. 2010) show that, for a positive convex combination, `alignf` can be solved by finding a solution to $\min_{v\geq 0}v^\top Mv-2v^\top a$ where $a=\left(\langle K_{1_c},K_Y\rangle_F,...,\langle K_{p_c},K_Y\rangle_F\right)^\top$, $M_{k\ell}=\langle K_{k_c},K_{\ell_c}\rangle_F$ with K_{j_c} the centered kernel of K_j, i.e. $K_{j_c}=U_nK_jU_n$ with $U_n=I-11^\top/n$. Despite its simple implementation, `alignf` only brought a slightly improvement to our previous performance. That's why we decided to consider Non-Linear Combination of Kernels through `NLCK` algorithm (Cortes et al. 2009).

4.3. NLCK

The advantage of `NLCK` is that the weight vector μ is learned during the learning. We adapted the original algorithm to include SVM. The algorithm written in pseudo-code is the following:

Algorithm 1 Projection-based Gradient Descent Algorithm

Input: $\mu_{init}\in\{\mu|\mu\geq 0\wedge\|\mu-\mu_0\|_2\leq\Lambda\},\eta\in[0,1],\epsilon>0,d>0,C_{nlck},\left(K_j\right)_{j=1}^p$
$\mu'\leftarrow\mu_{init}$
repeat
  $\mu\leftarrow\mu'$, $K_\mu=\sum_{k_1+...k_p=d}\mu_1^{k_1}\cdots\mu_p^{k_p}K_1^{k_1}\cdots K_p^{k_p}$
  $\alpha\leftarrow$ `SVM_step`(C_{nlck}) or `KRR_step`(C_{nlck})
  $\mu'\leftarrow-\eta\alpha^\top K_\mu\alpha+\mu$
  $\forall\, k\,,\mu'_k\leftarrow\max(0,\mu'_k)$
  normalize μ' s.t. $\|\mu'-\mu_0\|=\Lambda$
until $\|\mu'-\mu\|<\epsilon$

5. Final results

Our best submission has been obtained using the `NLCK` algorithm along with 9 kernels: MM$(4,1)$, MM$(5,1)$, MM$(6,1)$, SP(4), SP(5), SP(6), WD(4), WD(5), WD(10). Table 1 shows the results with hyperparameters tuning using 5-fold cross-validation.

Table 1. Top results on the three different validation sets (25% of train set) obtained with NLCK combined with SVM.

	NLCK			SVM	ACCURACIES	
DATA	Λ	d	C_{nlck}	C_{svm}	TRAIN	VAL
TF 1	5	3	1	1.9	0.99	0.655
TF 2	5	4	10	2.1	0.99	0.779
TF 3	1	3	0.01	3	0.99	0.684

At last, our accuracy on the private leaderboard was 0.70733 $(24/76)$ and fell down to 0.67466 $(38/76)$.

6. Feedback & Discussion

As you might see, we lost 14 places in the ranking, which is huge. Overfitting has something to do with it. In fact, we under-estimated the impact of our quasi-perfect prediction scores on the training sets. We were quite lucky on the restricted 20% of the testing set. We are nevertheless pleased to see how we managed this data challenge. Indeed, it allowed us to be more familiar with ML applied to DNA sequences classification tasks, and also be aware of some useful combining techniques. Nevertheless, along with overfitting, one second major drawback to our approach is the computational time. Indeed, we did not used tries to make kernels construction faster, and this was a pity as we did not fully exploit the different kernel methods available.