Kernels in Machine Learning Project - Protein sequences classification

Charles AUGUSTE - Yonatan DELORO - Amine KHELDOUNI
Team name: OTP Kernel | Contact: firstname.lastname@eleves.enpc.fr

The goal of the "Kernels in Machine Learning" challenge was to gain practical experience on kernel-based learning algorithms suited for structured data. The challenge aimed at predicting whether or not a DNA sequence region is binding to a specific transcription factor (TF), for three datasets containing 2000 training sequences and 1000 test sequences. Our best solution to this classification task was based on Support Vector Machine (SVM)) algorithm using a sum of Spectrum, Mismatch, and Levenshtein-Exponential Kernels. First we describe the various kernels and algorithms we implemented. Then, we explicit our best solution and discuss our results. Finally, we suggest several ideas which could enhance the performances of the less successful methods we tried.

1 Designing Kernels for DNA sequences

1.1 Gaussian Kernel with Euclidean distance

First of all, we implemented the Gaussian kernel with Euclidean metric $K(s,s') = \exp(-\gamma||e(s) - e(s')||_2^2)$, where s and s' are two DNA sequences, and where $e(s)$ is the one-hot encoding matrix of the sentence s (of size $(n,4)$ with n the sentence length, mapping A,T,C,G to the $\mathbb{R}^4$ standard basis) which we then flattened to a vector.

1.2 Exponential Kernel with Levenshtein distance

Then, we implemented the exponential kernel $K(s,s') = \exp(-\gamma d(s,s')))$, using, as for the metric d, the Levenshtein edit distance, which computes the length of the shortest path of operations to transform s into s', using only insertion, removal and substitution of characters as the three allowed edit operations. The kernel is pd. as the Levenshtein metric satisfies the distance properties [4]. We computed the Levenshtein distance between s and s' using dynamic programming, but finally used the Python Levenstein library to speed-up computation (time complexity $O(|s|.|s'|)$).

1.3 Fisher Kernel - with Hidden Markov Model

The Fisher Kernel $K(s,s') = \phi_{\theta_0}(s)^T I(\theta_0)^{-1}\phi_{\theta_0}(s')$ with $\phi_{\theta_0}(s) = \nabla_\theta \log P_\theta(s)|_{\theta=\theta_0}$ and $I(\theta_0) = \mathbb{E}[\phi_{\theta_0}(s)\phi_{\theta_0}(s)^T]$, compares two sequences s and s' looking at how much each parameter of the probabilistic model we designed actually contributes to generating s and s'. Following [1] approach, we found it relevant to implement a Hidden Markov Model (HMM) as our probabilistic model. We computed the maximum likelihood estimator θ_0 using the EM algorithm and the forward-backward algorithm. We tried different underlying models with Markov assumptions of order $m = 1, 3, 6$ and different numbers of underlying states.

1.4 Spectrum and Mismatch Kernels

The Spectrum kernel compares two sequences looking at the number of common subsequences of a given length they contain : $K(s,s') = \sum_{u\in A_k} \psi_u(s)\psi_u(s')$, with A_k the set of all possible subsequences of length k and $\psi_u(x)$ the occurrences counter of u in s. The Mismatch kernel [2] takes into account subsequences similar up to a certain number of mismatches m: $\psi_u(s)$ becomes the number of occurrences of subsequences of length k at distance lower than m mismatches from u. To compute these kernels, we did not use the usual trie implementation but rather used sparse matrices that we filled using efficient mismatch computation based on memoization.

1.5 LA Kernel with Nystrom approximation

$K^{LA,\beta}(s,s')$ sums up the scores obtained from local alignments with gaps of the sequences s and s' [3], with scores computed given a substitution matrix $S(c,c')_{c,c'\in\{A,T,C,G\}}$ and a gap penalty function $g_{d,e}(n) = d + e(n-1)$ (β is an additional parameter). We implemented the dynamic programming described in [3] to compute it, then applied $\frac{1}{\beta}\log$. transformation to counter the fast decrease of the kernel value with the similarity, and finally substracted the smallest neigative eigenvalue from the train Gram matrix diagonal to make it p.d. as proposed in [3].

To compute the LA kernel, we had to make use of Nystrom approximation [6]: we selected randomly $p << 2000$ anchor sequences $(a_j)_{1\leq j\leq p}$ in a given training dataset and encoded each sequence s with a p-dimensional vector $\phi(s) = K_f^{-1/2}[K(s,a_1),...,K(s,a_p)]^T$ with K_f the Gram matrix for the anchors. Train and test Gram matrices were computed using kernel approximation $\phi(s)^T\phi(s')$. Choosing $p = 40$ reduced considerably the number of kernel evaluations, and the expected number of hours for training + prediction from $333,3$ to $6,7$.

1.6 Normalizing Kernels

We also considered normalizing the data to the unit norm in the feature space for the above kernels, building $K^{norm}(s,s') = \frac{K(s,s')}{\sqrt{K(s,s)K(s',s')}}$.

1.7 Combining Kernels with (Weighted) Sums

Finally, we combined some of the kernels mentioned above using weighted sums of kernels $\sum_i \eta_i K_i$, with $\eta \geq 0$. More precisely, we tried two options: (i) choosing any η_i equal to 1, which is equivalent to concatenating the features for the various K_i [6], and (ii) using Multiple Kernel Learning to optimize η (cf. 2.2), which can be thought as a risk minimization estimator with group lasso penalty [6], or as selecting features of the K_i .

2 Predicting TF-bindings with Kernels

2.1 Support Vector Machine

Predicting a TF-binding is a binary classification problem. The standard Support Vector Machine algorithm solves the penalized risk estimator minimization problem $\min_{f \in H} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i f(x_i)) + \lambda ||f||_H^2$ (P) using hinge loss $\phi(u) = \max(1-u, 0)$ (denoting (x_i, y_i) the training data, with $y_i \in \{-1, 1\}$, and H the RKHS). Thanks to the representer theorem, the solution can be expanded as $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$. For computation efficiency, we chose to solve the dual formation (D) of the convex problem (P), which is [6]: $\max_{\alpha \in \mathbb{R}^n} J(\alpha) = -\frac{1}{2}\alpha^T K \alpha + \alpha^T y$ subject to $0 \leq Diag(y)\alpha \leq \frac{1}{2\lambda n}$, where K is the training Gram matrix. We solved it thanks to the quadratic programming solver from CVXOPT library.

In order to find a good regularization parameter λ, whose optimal value is known to depend on the dataset, we first used a grid search and selected the value which led to the best score on a validation set. However, we observed that λ depended on the split so we decided to implement a K-fold validation. In practice, the latter took much more time for the kernels for which evaluation was expensive, so we often used a simple validation to select a reasonably good λ.

2.2 Multiple Kernel Learning (MKL)

To optimize η using the weighted sum of kernels $K_\eta = \sum_{1 \leq i \leq M} \eta_i K_i$, we attempted to implement the SimpleMKL algorithm designed by [5] which solves, using a reduced gradient algorithm: $\min_{\eta \in \Sigma_M} J(\alpha^*_{K_\eta})$ with $\alpha^*_{K_\eta}$ the solution of problem (D) for kernel K_η (Σ_M denotes the simplex in $\mathbb{R}^M$).

3 Results and discussion

Our best solution was achieved using a sum of normalized Spectrum and Mismatch Kernels ($k = \{4, 6, 8\}$, $m = \{0, 1, 2\}$) and Exponential Kernel with Levenshtein Distance ($\gamma = 0.5$). The feature space was of very high dimension (4^k per mismatch kernel of length k). This solution led to 71.07% accuracy in the public leaderboard ($22^{th}/76$) and to 67.53% with the last 50% ($37^{th}/76$). Despite our validation process, there is maybe still some overfitting with our quite high dimensional feature space.

Definitely, the Spectrum and the Mismatch kernels were those which performed the best. Normalizing them seemed to increase performances. Summing them (and with the Levenshtein one) enabled to further increase the performance. However, using MKL to optimize the η of the sum was not fruitful as it converged to a vector with zero weights for any kernel except for one, generally among the best, which was weighted 1. Despite several proof-readings, we did not manage to spot a mistake in the final code, and the target $J(K_\eta)$ kept decreasing as expected. Maybe we can hypothesize that, as many of the kernels we used in the sum were somehow similar, the MKL only selected the best one (as the Lasso regularization tends to discard many features).

The Fisher Kernel yielded around $60 - 65\%$ only on our validation sets. Simple HMM did not capture enough information and held bad results while it was hard to make more complex models (greater Markov order) converge. Therefore, all generative models we used yielded average results.

The LA Kernel with Nystrom approximation (using BLAST matrix $S(c, c') = 5.1_{c=c'} - 4.1_{c \neq c'}$ and $(e, d, \beta) = (1, 7, 0.5)$) led to at most 55% on our validation sets. The dimension of the projection space (40) was certainly too small to hope for something, given that we chose its basis at random among the 2000 training sequences. In addition, the many hours needed to assess such kernel made it impossible for us to do parameters tuning properly.

The relative success of Spectrum and Mismatch kernels with respect to Gaussian and LA Kernels might be interpreted by the fact that substructures presence in sequences are maybe more discriminative than specific DNA basis at given positions.

4 Ideas for potential improvements

To improve regularization and allow for better generalization, one idea could have consisted in using ensemble methods. Instead of choosing λ with a k-fold cross validation process, one idea is to predict the label with each of the k SVM trained on 80% of the data, and predict the final label with the sign of the mean of the labels given by each predictor.

For LA kernel, we would have certainly needed a faster implementation of kernel evaluation (using a transducer) to use a larger number of anchor points for its Nystrom approximation, as well as Kernel PCA algorithm to get more interesting projective dimensions.

Finally, one useful pre-processing could have consisted in grouping the DNA basis by 3 into amino acids, and computing kernels on the new sequences. As several triplets encode for the same acid, introducing such prior knowledge may have led to better performances, though we are not sure at which position to begin the transformation.

5 Acknowledgements

Thanks for this challenge as exciting as instructive! It is a bit disappointing to observe that the most classic kernels seemed to perform the best on this task, but we are happy to have implemented other kernels or algorithms discussed in class, even if this work was not quantitatively fruitful. Link to our Git repo.

References

[1] Tsuda, Kin, and Asai. "Marginalized kernels for biological sequences". In: *Bioinformatics* (2002).

[2] Leslie et al. "Mismatch string kernels for SVM protein classification". In: *NIPS*. 2004.

[3] Saigo et al. "Protein Homology Detection Using String Alignment Kernels". In: *Bioinformatics* (Aug. 2004).

[4] Xu and Zhang. "Kernels based on weighted Levenshtein distance". In: *IEEE International Joint Conference on Neural Networks* (2004).

[5] Rakotomamonjy et al. "SimpleMKL". In: *Journal of Machine Learning Research* (2008).

[6] Mairal and Vert. "Machine Learning with Kernel Methods". In: (2018).