# Machine learning with kernel methods 2021 Kaggle Competition

Julien HAURET, Liam GONZALEZ, Sam PEROCHON
Master MVA
Google Brain team. Public score : 0.645, Private : 0.649

## Abstract

*The Kaggle competition proposes to predict whether a DNA sequence region is a binding site to a specific transcription factor. Transcription factors (TFs) are regulatory proteins that bind specific sequence motifs in the genome to activate or repress transcription of target genes. Genome-wide protein-DNA binding maps can be profiled using some experimental techniques and thus all genomics can be classified into two classes for a TF of interest: bound or unbound. In this challenge, we will work with three datasets corresponding to three different TFs.*[1]
*We first investigated to work with the provided feature matrices* `mat100`*, before moving towards the raw DNA sequences and some specific string kernels.*

## 1. Method.

Our approach to predict the nature of the test sequences can be summarized as follows :

- Designing specific feature space for DNA sequences.
- Building classification algorithms using those features.

### 1.1. Feature space for biological sequences

**Mat100 Features** This transformation of the raw data was provided, and was computed using bag-of-words representation. For each of the 3 data sets (indexed with 0,1 and 2), we transformed the data so that they have zero mean and unit variance, using the statistics of the training data sets. A visualization of the features is proposed on Figure 1.

**Spectrum kernel** We then turned out to start from the raw DNA sequences and build our own feature space. This first kernel consists in looking for the number of occurrences in the sequence of a specific substring. To build the representation $\phi$ of the data, one has to fix the length of the substring

[1]Each of the three training dataset is composed of 2000 sequences, with a test set to predict of 1000 sequences.

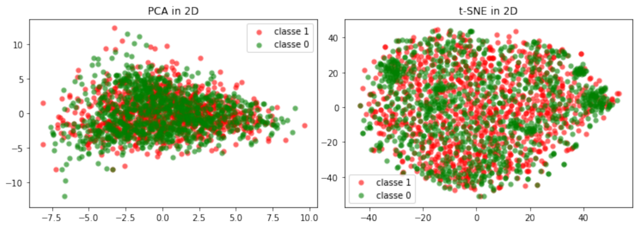

Figure 1. PCA and t-SNE 2D representations of Mat100 features

and look for every substrings of this length . This methodology was introduced in the paper of Leslie and al. 2002 [1]
.
An efficient implementation is required to be used in practice. A visualization of these features is proposed on Figure 2. In practice, we always used a substring size of length 8 which is the maximum that our RAM can handle. Again, we can see that the features do not seem to be clearly separated.

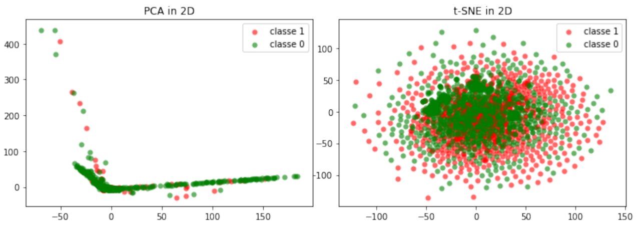

Figure 2. PCA and t-SNE 2D representations of Kernel spectrum features

**Mismatch and Substring kernels** The mismatch kernel from [2] and the substring kernel from [3] only have been implemented in a naive way and can not be used in practice due to computation time.

**Bio kernel** We have tried to use the function `globalxx` from the Bio.pairwise2 [4] module which computes global alignments between two sequences. We can not provide a feature representation for this kernel as the mapping $\phi$ is implicit.

## 1.2. Classification algorithms

**Ridge Regression.**

First baseline classifier consisted in implementing from scratch a Ridge Regression classifier. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_{\omega} \|y - X\omega\|_2^2 + \alpha\|\omega\|_2^2$$

In order to test our implementation of the algorithm, we split the raw training set into 90% for the training and 10% as validation set. We then compared our results with the SCIKIT-LEARN implementation, which validated our implementation [5].

*Both models reached 60.5% accuracy on the public test set with Mat100 features.*

**Kernel Ridge Regression.**

In order to move to a non-linear classifier, we implemented from scratch a Kernel Ridge Regression, with a Gaussian RBF kernel of the form:

$$K(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\|x-y\|_2^2}$$

By noting $K$ the Gramm matrix associated to our data, the problem becomes :

$$\arg\min_{\alpha\in\mathbb{R}^n} \frac{1}{n}(K\alpha - y)^T(K\alpha - y) + \lambda\alpha^T K\alpha$$

with analytical solution : $\hat{\alpha} = (K + \mathbb{I})^{-1}y$.

To find the optimal hyperparameters, $\lambda$ the regularization parameter and $\sigma$ the parameter of the RBF kernel, we used a coarse then fine grid-search strategy. Again here, we used 90% of the training data to train our model, and tested it on the remaining 10% of the validation set, and compared our performances with the SCIKIT-LEARN implementation.

*We used the same final optimal hyperparameters ($\lambda = 1$ and $\sigma = 1.233$), and obtained 58,1% accuracy for both implementation, on the public test set with Mat100 features. To explain this accuracy we deduced that using an inappropriate kernel can be worst than a simple linear kernel.*

**SVM.**

We look for the solutions for the following problem :

$$\arg\min_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \varphi(y_i f(x_i)) + \lambda\|f\|_{\mathcal{H}}^2$$

Which, by the representer theorem, is equivalent to :

$$\begin{aligned} &\arg\min_{\alpha\in\mathbb{R}^n} \frac{1}{n}\sum \varphi(y_i[K\alpha]_i)) + \lambda\alpha^T K\alpha \\ &= \arg\min_{\alpha} \sum \alpha_i - \frac{1}{2}\alpha^T H\alpha \\ &\quad s.t \quad 0 \le \alpha_i \le C = \frac{1}{2\lambda n} \\ &\quad \alpha^T y = 0 \end{aligned}$$

With $H = \text{Diag}(y).K.\text{Diag}(y)$. According to (7.18) in the textbook from Bishop [6], we can also add a bias so that the sign of $f$ tells us the predicted labels. We used : $b = \frac{1}{N_S}\sum_{n\in S}(y_n - \sum_{m\in S}\alpha_m y_m K_{n,m})$, with $S$ being the indices of the support vectors.
This optimization problem takes the form of a quadratic programming problem (QP), which we implemented using the Python package *cvxopt* [7].

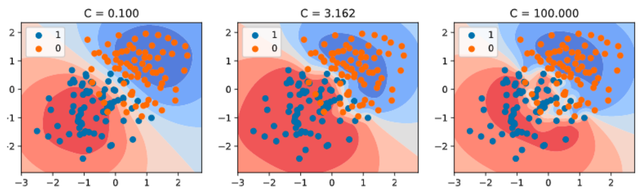

Figure 3. **SVM predictions for different values of $C$.** We simulated 2D data to test our implementation, and we observe that the lower $C$ is, the more the solution is regularized.

*We kept using this classification algorithm for the different designed kernels and finetuned C for each dataset.*

## 2. Results.

Results are represented in Table 2. The best result on public leaderboard have been obtained with the sum of the spectrum and bio kernels. Whereas for the private leaderboard, it is the spectrum kernel alone which performed the best. The classification algorithm was SVM in both cases. No overfitting was observed.

| | Public Leaderboard | Private Learderboard |
|---|---|---|
| Accuracy | 0.64533 | 0.64866 |
| Rank | 50/74 | 45/74 |

Table 1. Results on Kaggle

## 3. Conclusions.

Time constraints prevented us from implementing the efficient implementation of mismatch and substring kernels. However, the rather simple spectrum kernel is still effective when coupled with SVM to deal with biosequences. An ensemble algorithm could have been added to group all our classifiers. Our Code lies on Github [8].

## References

[1] 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

[2] Christina S Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004. 1

[3] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002. 1

[4] Peter Cock. biopython. `https://github.com/biopython/biopython/blob/master/README.rst`, 2021. 1

[5] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 2

[6] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. 2

[7] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014. 2

[8] Liam Gonzalez, Sam Perochon, and Julien Hauret. Kaggle kernelml 2021. `https://github.com/jhauret/Kaggle_KernelML_2021`, 2021. 2