Machine learning with kernel methods
Protein family prediction challenge

GiveMeSumKernel

Mahdi Kallel & Arnaud Barral

Telecom Paris

firstname.lastname@telecom-paris.fr

Haythem Hassayoun

Paris Dauphine

haythemhassayoun@gmail.com

dataset	public	private
accuracy	0.65400	0.65466

Abstract

This report presents our work and results for the kaggle competition for the Machine learning with kernel methods class. The source code is available here https://github.com/kallel-mahdi/Kernel_Methods_Challenge2021/.

1. Introduction:

We are working with protein which means string sequence with four possibles letters 'A', 'C', 'T' and 'G'. Numerical representations of the data were also provided in this competition. We use these representation for real kernels and we use the string kernels for the string sequences of DNA.

2. Kernels:

In this section we present the kernels we implemented for the challenge.

2.1. Real kernels :

For the provided numeric data we experimented mainly with three kernels :

- RBF : $K(x,y) = \exp(\frac{-||x-y||^2}{2\sigma})$
- Laplacian : $Laplacian : K(x,y) = \exp(\frac{-||x-y||}{\delta})$
- Polynomial: $K(x,y) = (x^T y + c)^d$

2.2. String kernels:

We have the string representation of each DNA sequence, we studied a few string kernels to leverage structural similarity of sequences.

2.2.1 Spectrum kernel:

The spectrum kernel [2] is defined on the space $\mathcal{S}$ of all finite length sequences of characters from an alphabet A.

The feature map is indexed by all possible sub-sequences a of length k from alphabet $\mathcal{A}$. $\phi_k(x) = (\phi_a(x))_{a\in\mathcal{A}^k}$ where $\phi_a(x)$ denotes number of times the sub-sequence a occurs in the sequence x.

Thus the image of a sequence x under the feature map is a weighted representation of its k-spectrum. The k-spectrum kernel is then $K(x,y) = \langle\phi_k(x), \phi_k(y)\rangle$. The more $x\&y$ have commun sub-sequences the higher it's value.

For our implementation of this kernel we build a suffix tree for the collection of k-length sub-sequences of of all the sequences in the dataset, obtained by moving a k-length sliding window across each of sequence x. At each k-depth leaf node representing the k-length a of the suffix tree, we store the vector $\phi_a(\mathcal{X}) = (\phi_a(x_1), .., \phi_a(x_n))$ counting the occurrences of a in all the sequences of the dataset $\mathcal{X}$.

If we denote by $\Phi = (\phi_a(\mathcal{X}))_{a\in\mathcal{X}}$ then the kernel matrix over the whole dataset can be computed as $K = \Phi^T\Phi$.

2.2.2 Mismatch kernel:

We denote by $N_{(k,m)(a)}$the the set of all k-length sequences a' from $\mathcal{A}^k$ s.t a' differs from a by m mismatches.

The mismatch kernel [1] is a slight variant of the spectrum kernel where instead of counting only the occurrences of the k-length sub-sequences a as a feature map, we count the occurrences it's neighbours $N_{(k,m)}(a)$.

We use a recursive algorithm to efficiently construct the now denser suffix tree.

2.3. Learning with many kernels:

Working with kernels allows for a natural way to work with multiple representations. When using a weighted sum of multiple kernels $K_s(x,x') = \sum w_i K_i(x,x')$ the models learns using the concatenation of each kernel's feature map.

The "Sum Kernel" is the easiest way to leverage this trick, by setting all $w_i = 1$. For this to work all the kernels are normalized using $K_i^n(x, x') = \frac{K_i(x,x')}{\sqrt{K(x,x}\sqrt{K(x',x')}}$.

In the above formulation we give all the kernels equal weights in the learning procedure and thus the same importance, however in most cases a kernel can provide better representations than another and we would like this to be reflected in it's assigned weight.

"Multiple kernel learning" allows for learning simultaneously the predictor (SVM) and the kernel weights. We implemented our version following the guidelines from [3].

For our experiments simpleMKL ended up assigned a.e.q weighs to all kernels despite requiring much more time to run. This can be explained by the fact that most kernels are able to perfectly fit on the training data and thus the choice of weights has no real effect on the objective.

With this in mind, we chose to use the "Sum Kernel" for the rest of our experiments.

3. Classifiers:

3.1. Support Vector Machine:

Support vector machine (SVM) is an algorithm that looks for the hyperplane with that is able to separate the data with the highest margin.

Kernel SVM is a variant that leverages the kernel trick in order to look for the separating hyperplane in the RKHS which is of higher dimension.

In our implementation we solve the dual problem of the C-SVM :

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}^d} 2 \sum_{i=1}^{n} \alpha_i y_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) \\ \text{s.t.} \leq y_i \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{1}$$

We solve this quadratic program using the CVXPY library for optimization.

Note that when tuning the "C" parameter we use the warmup property to use the previous solution α^* as an initial guess and thus solve the problem faster.

3.2. Kernel ridge regression:

Kernel ridge regression extends ridge regression to the nonlinear case by learning a prediction function on the RKHS. Thanks to the representer theorem the kernel ridge regression consists of solving the following problem:

$$\arg \min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|_2^2 + \lambda \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}$$

Which admits $\alpha = (K + \lambda n I_n)^{-1} y$ as a closed form solution.W we use the Numpy library to solve this linear system.

4. Training procedure

There seemed to be a correlation between each "training" file and the corresponding test files. Usually allowing for a 5% validation accuracy gain.

We trained our models and finetuned our parameters on each dataset separately and then concatenate the predicted results.

For the fine tunning procedure we use the optuna library [4] for bayesian search of the optimal parameters.
Each dataset was split into 80 % / 20% train validation .

We had to choose the learning algorithm (SVM / KRR) each had a single parameter. And we had to tune the parameters for the kernels.

KRR and SVM yielded very similar results on the validation data with KRR being 10x faster for learning.

At the beginning we used only one kernel per model and tuned both the learner and the kernel. We then plung the sum of these kernels into a learner and then we tune the learner's one last time.

Note that for the mismatch kernel m parameter had to bee small due to the exponential factor in complexity. For the k_{size} parameter, the best results were obtained for fairly large values, between 8 and 12 for the spectrum kernel and between 6 and 9 for the mismatch kernel and $m = 2$.

5. Results:

The first observation was that, as expected, the string kernels made for biological sequences, the spectrum and mismatch kernels, returned better accuracy on the validation dataset than the real kernels used with the provided representation which is not guaranteed to be optimal. The mismatch kernel was the best.

As expected the sum kernel returned better results than single kernels. We tried with various combinations.

The best combination found was one gaussian kernel, one mismatch kernel, two spectrum kernels (with differents parameters). When combining several times the same kernel, having really different parameters works best, intuitively choosing different range of values let the kernels learn different and complementary information and reduces the predictors variance.

With this strategy, we obtained rather good results on the public and on the private set. The accuracies on the public and on the private are almost the same, which seems to indicate that our method of combining several kernels, is able to generalise well.

With more time we would've experimented with ensembling strategies but implementing the Mismatch kernel and SimpleMKL consumed much of our time.

References

[1] Mismatch String Kernels for SVM Protein Classification: [*https://proceedings.neurips.cc/paper/2002/file/12b1e42dc0746f22cf361267de07073f-Paper.pdf* / Leslie et al.

[2] THE SPECTRUM KERNEL: A STRING KERNEL FOR SVM PROTEIN CLASSIFICATION: *https://www.ics.uci.edu/ welling/teatimetalks/kernelclub04/spectrum.pdf* / Leslie et al. 1

[3] SimpleMKL: *https://www.jmlr.org/papers/volume9/rakotomamonjy08a/rakotomamonjy08a.pdf* / Rakotomamonjy et al.

[4] : Optuna : https://optuna.readthedocs.io/en/stable/index.html 1