Kernel methods - Data challenge

Team: Aldescu Marian & Mirzaee Saeed

February 19, 2021

The code can be found at: `https://github.com/saedmirzaeee/kernel-method`
Score: public/private - 0.685/0.698

1 Introduction

The aim of this work is to apply different kernel methods techniques on a binary classification task. More precisely, given a data set composed of DNA sequences annotated with 0(unbound TF)/1(bound TF), we want to train a classifier which is able to generalize on unseen data.

1.1 Data

As data comes in raw and numerical format, we want to exploit both of them. The advantage of the first is the simplicity of using, but we suspect some information comprised initially in the raw format may have been lost during pre-processing. The raw sequences could be more convenient to use, in terms of performance, but only if used together with a measure which conserves as much information as possible from the original sequences.

2 Kernels

To apply the kernel methods seen in the course, we want first to employ a similarity measure K, called *kernel*, between any two points $x, x' \in \mathcal{X}$, where $\mathcal{X}$ is the training set. An essential condition which needs to be satisfied by K is the positive definiteness, in other words, according to Aronszajn Theorem(1950), if there exists a Hilbert space $\mathcal{H}$ and a function $\phi : \mathcal{X} \to \mathcal{H}$, it can be expressed as a dot product $K(x, x') = \langle \phi(x), \phi(x') \rangle, \forall x, x' \in \mathcal{X}$.

2.1 Gaussian kernel

This is defined as:

$$K(x, x') = exp(-\frac{\|x - x'\|^2}{2\sigma})$$

where $\sigma > 0$ behaves in a similar manner as the degree d of the polynomial kernel. Small values of σ are associated to large values of d, hence allowing to fit very well the training data, but with the risk to overfit it. As σ increases, the kernel tends to become a constant function, causing difficulties to learn complex models.

2.2 Spectrum kernel

A natural way to compare 2 strings is to count how many substrings of a certain length k they have in common. For this we use the spectrum of order k of a sequence, which is the histogram of frequencies of all the contiguous substrings of length k. Formally, considering the alphabet Σ, and Σ^k - the set of all possible substrings of length k over Σ, and $\phi_u(x)$ the number of occurrences of u in x, the k-spectrum kernel of 2 strings x and x' is:

$$K(x, x') = \sum_{u \in \Sigma^k} \phi_u(x)\phi_u(x')$$

The spectrum kernel is p.d., as $K(x, x')$ can be easily seen as a dot product $K(x, x') = \langle \phi(x), \phi(x') \rangle$.

In our case, the alphabet is: $\Sigma = \{A, C, G, T\}$ and for Σ^k we did not generate all the substrings of length k(k-mers), but instead selected them from the training data set, hence obtaining Σ^k'$\subseteq \Sigma^k$. We used Σ^k' as a dictionary with values 0 for all the contained k-mers, such a dictionary being associated with each sequence $x \in \mathcal{X}$. For every sequence x, we populated its associated dictionary in $\mathcal{O}(|x|)$ time, by incrementing the values for each substring of length k in x. Considering the pre-indexation of the sequences, the kernel between any $x, x' \in \mathcal{X}$ is computed in $\mathcal{O}(|x| + |x'|)$. The optimal value of k needs to be tuned and we will see this in the Validation section.

3 Kernel methods

3.1 Kernel ridge regression(KRR)

Considering a RKHS $\mathcal{H}$ and a p.d kernel applied on a set $\mathcal{X}$, we can compute the Gram matrix($\mathbf{K}$). KRR aims to approximate a prediction function $\hat{f}$ over the training set $\mathcal{X}$, by minimizing the following:

$$\hat{f} = argmin_{f \in \mathcal{H}} \sum_{i=1}^{n} (y_i - f(x_i)) + \lambda \|f\|^2_{\mathcal{H}} \qquad (1)$$

By the representer theorem, new predictions can be expressed as:

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

where $\lambda > 0$, $n = |X|$ and $\alpha \in \mathbb{R}^n$ needs to be approximated. Eq.(1) is finally reduced to solve the linear system of equations:

$$\alpha(K + \lambda I_n) = y \tag{2}$$

To solve Eq.(2), we used *numpy* python library, more precisely *linalg.solve* function.

3.2 C-SVM

We will not insist on the the theoretical details, as we have already seen in the course, but it is important to recall that SVM problem can be reduced to solving a quadratic program under linear constraints. We also consider a variant of SVM, C-SVM, in which the problem can be formulated w.r.t. a regularization hyperparameter C. We used *cvxopt* python package to solve the equation.

4 Validation

Each of the previously presented kernel methods requires the tuning of some hyperparameters: σ in the case of the gaussian kernel, λ and C regularization parameters in the case of KRR and C-SVM respectively. Also, in the case of spectrum kernel, a proper value k needs to be approximated. To achieve this, and because the provided dataset lacks a validation set, we employ the k-fold cross validation.

4.1 K-fold cross validation

We chose to split the training set in k=5 folds, therefore, for 5 times, the models are trained on 80% of the data, w.r.t. a certain hyperparameter, and the other 20% part is reserved to measure the performance. The metric that we used to quantify the performance is the accuracy. The chosen parameter is the one with the best average accuracy over the 5 folds.

4.2 Other data treatments

Data augmentation - The fact that data comes in the form of sequences of base pairs and also knowing that the DNA is organized as a double helix of nucleotides, each base pair having a complementary, e.g. A-T, C-T, we got the idea to generate the associated sequences according to the complement rule and merge them in the train set. This proved in the end to be ineffective, as the computation time increased and no additional performance was achieved.

Data normalization - We also tried to normalize the kernel matrix asn in Eq.(3), thinking that it may bring a performance improvements, but finally, it didn't change the final accuracy.

$$K[x_i, x_j] \leftarrow \frac{K[x_i, x_j]}{\sqrt{K[x_i, x_i]}\sqrt{K[x_j, x_j]}} \tag{3}$$

5 Results

After running the cross validation to approximate the hyperparameters, we selected the values that can be seen in Table 1. In Table 2 we see the overall performance of our algorithms. KRR together with gaussian kernel achieved the lowest score of 0.63. We can see that spectrum kernel has a better result for both KRR and C-SVM, but surprisingly the highest accuracy is obtained by KRR with spectrum. We were expecting C-SVM to have the best result.

P	Alg.+kernel	Tried values	Best
λ_1	KRR+gaussian	$10^{-4}, 10^{-3}, 10^{-2}$	10^{-3}
σ	KRR+gaussian	0.01, 0.1, 0.5	0.1
λ_2	KRR+spectrum	0.01, 0.1, 0.3	0.1
k	KRR/C-SVM+spec.	6,7,8,9	7

Table 1: Selected hyperparameters. Actually we tested more values, but enumerated just some of them

Algorithm	Kernel	Accuracy
KRR	gaussian	0.63
KRR	spectrum	0.685
C-SVM	spectrum	0.66

Table 2: Results for the public leaderboard

On the public leaderboard, we achieved the accuracy of **0.685**(ranking 16/32), and on the private one we went up 6 places, with an accuracy of **0.698**(ranking 10/32).

6 Conclusion

At the beginning, we set out to work on both preprocessed(numerical) and raw data, and to see how big it is the difference of performance between them. Finally, using the appropriate kernels on raw data(spectrum) could lead to very good results. Also, we noticed that we moved up in the private leaderboard by 6 places, meaning that our model did not overfit and was able to generalize well on new data.

References

[1] John Shawe Taylor & Nello Cristianini. "Kernel methods for pattern analysis." Cambridge University Press (2011)

[2] Slides of the course