ALM Data challenge report

Ulysse Enjalbert

February 16, 2020

1 Introduction

This data challenge was a sequence classification task. Our goal was to predict whether a DNA sequence region contain or not a specific transcription factor.

For that, we were given 3 training sets of each 2000 DNA sequences, labeled 0 or 1 for the presence of the specific transcription factor.

We also have 3 optional files containing numeric data representing each training set. They are not optimal representations of the sequences, but easier to deal with. We will use them in our first approach.

Finally, we have 3 test sets of each 1000 DNA sequences to predict, and we have to return a file containing their predicted labels.

2 First approach: the Perceptron algorithm

As said in the introduction, my first approach used the *_mat100.csv* files containing the numerical reprensentation of the sequences. I implemented the Perceptron algorithm, which is the simplest possible solution to our problem.

To do that, I used the libraries `pandas` and `numpy`.

There is not much to say about the code, that is really basic. However, I had a first issue with this method: the results were really bad, worse than what I expected. My prediction function was returning almost only 1 as label, which obviously had an accuracy of 50% because the training set was distributed 50/50 between 0 and 1.

My issue was that I didn't change the label 0 in -1 to compute the weights. When I changed that, the results were more consistent. To see the accuracy of my algorithm, I've splitted my training set in two, using 90% of it to train and 10% to validate.

I tried different learning rate, and the best results were with $\eta = 0.001$. I had an accuracy around 0.55 which is still not good. The biggest issue is that the data may not be linearly separable in the space provided. We need to use a kernel, preferably one specific to DNA sequences. That will be our second approach.

3 Second approach: Spectrum kernel

The kernel that I will use is the spectrum kernel. This kernel was imagined for DNA sequence, and seems to be really appropriate for our problem.

We index the feature space by fixed-length strings: $\Phi(\mathbf{x}) = (\Phi_u(\mathbf{x}))_{u \in \mathcal{A}^k}$, with $\Phi_u(x)$ being the number of occurrences of u in $\mathbf{x}$.

If we take for example $\mathbf{x} = TCTGCC$, the 3-spectrum of $\mathbf{x}$ is $\mathcal{A}^3$ =(TCT, CTG, TGC, GCC).

The k-spectrum kernel is:

$$K(\mathbf{x},\mathbf{x'}) = \sum_{u\in\mathcal{A}^k} \Phi_u(\mathbf{x})\Phi_u(\mathbf{x'})$$

In this approach, I will use the libraries **pandas**, **collections**, **cvxopt** and **numpy**.

3.1 Gram matrix:

I wrote a function $k\_n(\mathbf{x},\mathbf{x'},\mathbf{k\_spectrum})$ that compute $K(\mathbf{x},\mathbf{x'})$ for $k = \mathbf{k\_spectrum}$. I optimised the calculus using dictionaries with the function *Counter* of the library **collections**.

Then I simply used a double loop *for* to compute the different Gram matrices for each of the 3 training sets and a given **k_spectrum**. It is possible to optimize it using the fact that K is symmetric to compute only half of the term of each matrices, but I didn't do it as I had to compute these matrices only once, saving them locally after.

3.2 Dual problem

I will use the dual formulation of our SVM problem:

$$\max_{\alpha\in\mathbb{R}^n} 2\alpha^T\mathbf{y} - \alpha^T\mathbf{K}\alpha \qquad \text{s.t.} \qquad 0 \le y_i\alpha_i \le \frac{1}{2\lambda n}$$

To maximize this, we will use the library `cvxopt` with the function *solver.qp*. As this function can only minimize, we will minimize $\alpha^T\mathbf{K}\alpha - 2\alpha^T\mathbf{y}$.

3.3 Prediction function

Once we have α, we need to compute $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i,\mathbf{x})$ for every $\mathbf{x}$ we want to predict. If $f(\mathbf{x}) < 0$ then the label is 0, else if $\mathbf{x} \geq 0$ then the label is 1.

3.4 Optimization of the hyper-parameters

We need to optimize the choice of **k_spectrum** and λ using a validation set just like in the first approach. I chose **k_spectrum** $= 10$ because less gave less accuracy, and more could lead to overfitting. $\lambda = 0.1$ gave good results with an accuracy of 0.68, that was my first (and best) submission on Kaggle. $\lambda = 0.01$ seems to give better results on the validation set (0.69), but not in practice on Kaggle.

An amelioration would be to use cross-validation instead of randomly choosing a validation set among the training set. I didn't do it because of a lack of time, but that seems to be the best thing to do to optimize the hyper-parameters. To do that, we probably need to parallelize the code in Python, as each try takes a few minutes.

4 Conclusion

To conclude, I can see some optimizations to get better results. First, a cross-validation to optimize the parameters as said before. Second possibility is to use another kernel, maybe a substring kernel.

Finally, if I have to remember only a few things about this challenge, that would be:

- be careful about the data (for example the label being 0 instead of -1)
- choose wisely the kernel, as the results depend mostly on it
- find the best libraries for the problem, like `cvxopt` to solve the quadratic problem