Advanced learning models 2020-2021
Data Challenge

Benkeroum El Mehdi

18/02/2021

***Abstract*— The purpose of this Data challenge is to predict whether a DNA sequence of the vocabulary $\{A, C, T, G\}$ is binding site to a specific transcription factor or not. It's requested to implement machine learning algorithms based on kernel methods. For this purpose, a set of kernels for sequence classification were implemented and tested, using kernelized learning algorithms such as support vector machine. The results on the training and test sets were encouraging, indeed, we had an accuracy and recall of 0.99% on training set and 0.71066% $\{9th\ rank\}$ on the public leaderboard and 0.69866% $\{9th\ rank\}$ on the private leaderboard. (please click on this link Leaderboard)**

I. Data preprocessing

A large part of machine learning project is dedicated to data processing, in order to make it usable for machine learning algorithms. In fields such as bioinformatics, we are faced with data that are not represented in a vector space of $\mathbf{R}^d$. In order to make this data interpretable for machine learning algorithm, we use Kernels, or similarities matrix, that provide a way to think in terms of similarities between data points. The following kernels are those which were used in this project:

The K-SPECTRUM KERNEL

Given a number $k > 0$, the k-spectrum kernel (SpectrumKernel.py) of a DNA-sequence is defined as:

$$K_k(seq_1, seq_2) = < \phi_k(seq_2), \phi_k(seq_2) >_{\mathbf{R}^{\sharp|\mathbf{alphabet}|^{\mathbf{k}}}} \quad (1)$$

Where: $\phi_k(seq) = (Count_{kmer}(seq))_{kmer \in \sharp|alphabet|^k}$, and kmer is a subsequence of length k contained in the set of all possible k-mers of the set of alphabet . In other words, ϕ_k represents the spectrum of all kmers of the given alphabet (for example:{A,C,T,G}), for a given sequence. The temporal complexity of this transformation is $O(\sharp|alphabet|^k(L - k + 1))$ [where $L = length(seq)$], indeed, it is necessary to calculate all the kmers in the given sequence which is done in $O((L - k + 1))$, and then calculate the frequencies of each kmer of the kmers in the sequence, which is done in $O(\sharp|alphabet|^k(L - k + 1))$. This implies a time complexity for calculating Gram Matrix of $O(\frac{N(N-1)}{2}\sharp|alphabet|^k(L - k + 1))$!. In order to remedy this, a variant of calculation of Gram Matrix was proposed in the project (SpectrumIndexedKernel.py), it has a time complexity of $O(N(L + \sharp|alphabet|^k))$. More precisely, this method transforms the vector of DNA-sequences, into index matrices according to the following rule: {A:0,C:1,G:2,T:3}, and then performs a multiplication between this matrix and a judicious matrix, in order to get $(\phi_k(seq))_{seq \in Data}$. In the case of our project, we have $L = 101$ and alphabet={A,C,T,G}, as soon as $k >= 5$, this complexity becomes $O(4^k N)$ (vs $O(4^k L N^2)$ for the naive method).

The MULTIPLE SPECTRUM KERNEL

This involves combining several k-spectrum kernel (MultipleSpectrumIndexedKernel.py), for different values of k, with a possible positive weight assigned to each of them. This kernel is useful to avoid an under-learning observed when using the previous kernel to train algorithms. Thanks to its low computational complexity: $O(N \sum_k 4^k)$ compared to other type of kernel like **(k-m)Mismatch kernel** and the capacity that gives to our models for their learning, this kernel was chosen to meet this classififcation problem.

The (k-m) MISMATCH KERNEL

The mismatch kernel (kmMismatchKernel.py) is similar to the spectrum kernel. The kernel parameter k has the same meaning in the mismatch kernel as in the spectrum kernel. The mismatch kernel also extracts substrings of fixed length k. The kernel parameter m defines the maximum number of mismatches that are allowed for each k-mer. This kernel remains powerful, but very expensive to calculate, it has a time complexity of $O(k^m \sharp|Alphabet|^m(L - k + 1)N^2)$!

II. kernelized SVM

In machine learning, support vector machines are a class of supervised learning models, used for both, classification and regression problems. They have a high generalization performance, ie, robust prediction. For the binary classification, they learn a linear decision boundary between the positive $(+)$ and negative $(-)$ classes. It often happens that the two classes are not linearly separable in the original space. For this reason, it's proposed to map the original space into a much higher dimensional space, called feature space **FS**, presumably making the separation linear. This new space is equipaed by the product scalar $\forall(x, y) \in \mathbf{FS}^2 :\ K(x, y) =< \phi(x), \phi(y) >$.

Computing the SVM classifier

Mathematically, the problem boils down to solving the following optimization problem:

$$\begin{cases} max_{\lambda \in \mathbf{R^N}} L_K(\lambda) = \sum_i^N \lambda_i - \frac{1}{2}\sum_{i,j=1}^N \lambda_i \lambda_j y_i y_j K(x_i, x_j) \\ Subject\ to: \sum_i^N \lambda_i y_i = 0\ and\ 0 \leq \alpha_i \leq \frac{1}{2N\alpha} \end{cases}$$

Where $\alpha > 0$ is the regularisation parameter. This algorithm was implemented in the file RegularizedSvm.py of the project.

III. Training and results

Three instances of the previous algorithm were used, each of them has been trained on one of the three training sets (for reasons related to training, specifically convergence and memory) with a multiple spectrum kernel of list of k equal to **[1,2,3,4,5,6,7,8,9,10]** (TuningKlist.py), this choice is justified in the following curve, in fact, for any fixed value of regularisation parameter $\alpha > 0$, the precision increases with the range of the list of k.

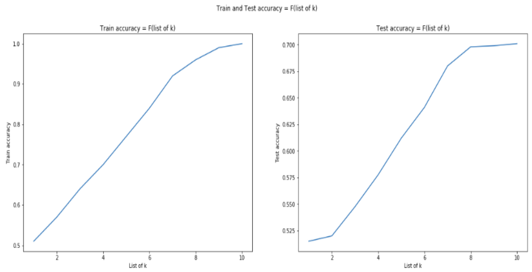


Fig. 1. The x-axis represents the list of k considered in training, for example at x = 5, the list of k is [1,...,x=5]. The curve on the test set, shows us that from x=8, ie, [1,...,8], the test accuracy does not change almost any more, and reaches a value of $\sim 70\%$

After this choice of the list of k (ie, [1,...,10]), as parameter for the multiple spectrum kernel used in training step, the alpha parameter was then tuned (TuningAlpha.py), in order to avoid overfitting but at the same time to have the best possible accuracy and recall.

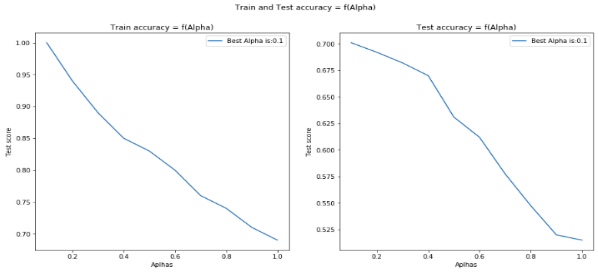
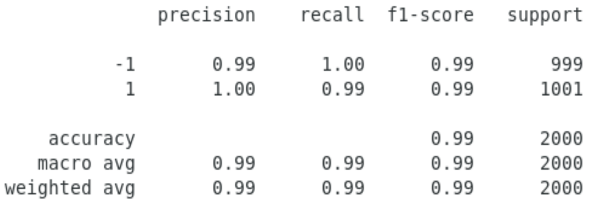


Fig. 2. The precision on the three sets decreases sharply for large values of alpha, taking this evolution into account, the choice of $\alpha = 0.1$ was made.

With these choices of hyperparameters, the three instances have been trained, on the three **balanced** training sets. Their classification report on training is as follows:

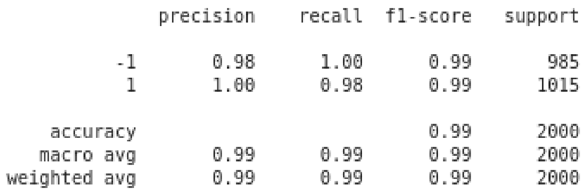


```
              precision    recall  f1-score   support

          -1       0.99      1.00      0.99       999
           1       1.00      0.99      0.99      1001

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       0.99      0.99      0.99      2000
```

Fig. 3. Classification report of svm n°0 on the training set n°0

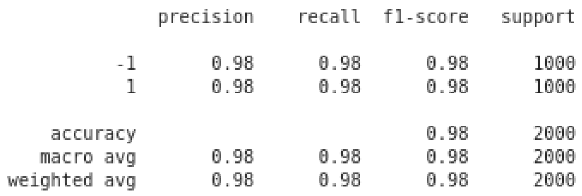


```
              precision    recall  f1-score   support

          -1       0.98      1.00      0.99       985
           1       1.00      0.98      0.99      1015

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       0.99      0.99      0.99      2000
```

Fig. 4. Classification report of svm n°1 on the training set n°1



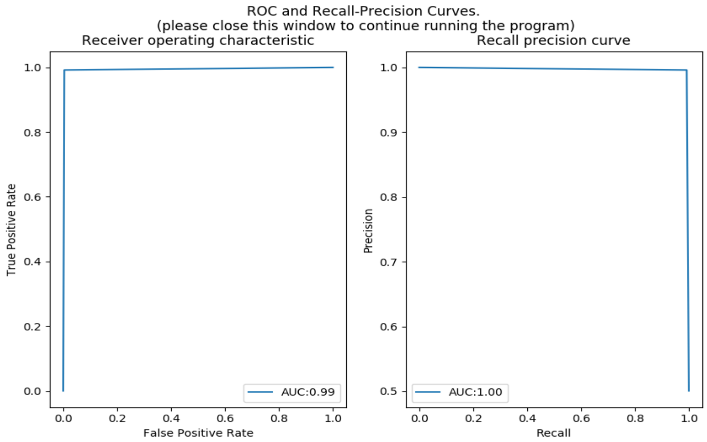
```
              precision    recall  f1-score   support

          -1       0.98      0.98      0.98      1000
           1       0.98      0.98      0.98      1000

    accuracy                           0.98      2000
   macro avg       0.98      0.98      0.98      2000
weighted avg       0.98      0.98      0.98      2000
```

Fig. 5. Classification report of svm n°2 on the training set n°2

The scores below and the following figure show that our instances are both precise and sensitive, with an accuracy and recall of 99%. The predictions of these models on all the test sets gave a accuracy of 71.066% (**9th rank** on the public Leaderboard)



IV. Conclusion

Thanks to our implementation of the svm algorithm as well as to the different kernels to preprocess our data, we were able to answer the requested classification problem with a relatively high score of 71.066% on the public Leader board. It is clear that this result can be improved by implementing other types of kernels, however, it requires computer resources large enough to be able to train the models for a long time without worrying about memory or blocking.