# DNA Sequence Classification

Haingoharijao Faniriniaina Ramandiamanana
Jama Hussein Mohamud
(Kernel)

May 31, 2020

## 1 Introduction

As machine learning field grows day by day, people are now becoming more and more interested in studying structural data, for example *DNA* sequence. The purpose of this work is to predict whether a *DNA* sequence region is binding site to a specific transcription factor (TF). TFs are regulatory proteins that bind specific sequence motifs in the genome to activate or repress transcription of target genes. We have a data-set that consists of 2000 *DNA* sequences, and each *DNA* sequence is 101 in length, where the sequence is a combination of the four nucleotides $A, G, C, T$. The key part of this project is to have best encoding scheme to achieve better performance in classifying two classes (i.e. as bound (0) or unbound (1)).

## 2 Sequence embedding and kernels

Dealing with text data is always difficult in the sense that we have to be very careful not to lose information. It is even becoming difficult in this case because *DNA* sequence is very sensible and has a very particular way to keep information, namely we have to take into account the regions of the *DNA* sequence mostly contain information. A simple and natural way to encode the data is to assign each nucleotide $A, G, C, T$ to an integer value, for instance $0, 1, 2, 3$ respectively (i.e. ordinal encoding). But that seems too "simple" for us to extract any features from the data, precisely very biased. Then another approach we did was to split the *DNA* sequence into subsequences of length $k$ (called *Kmers*) and then encode each subsequence $u$ to get a vector representation of the sequence. Encoding each $u$ depends on some functions $\phi_u(x)$ where $x$ is the sequence that we want to encode. For instance, we have tried two $\phi_u(x)$ functions:

- $\phi_u(x)$ is the number of times $u$ appears as subsquence of $x$ (with length $k$ fixed)(bag of words encoding)
- $\phi_u(x)$ can also be represented as the expansion in base 4 of $u$, where each nucleotide is encoded as integers provided previously.

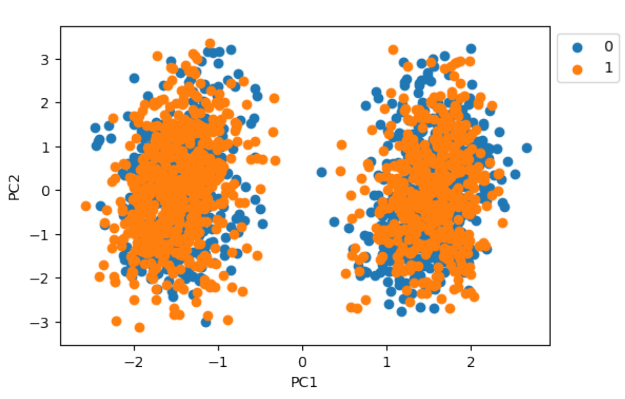
Once this is defined, we are now able to represent our sequence as $\Phi(x) = (\phi_u(x))_{u\in\mathcal{A}^k}$ where $\mathcal{A} = \{A, G, C, T\}$ and $k$ is the length of the *Kmers.* Moreover, this even allows us to define a kernel on the feature space, which is,

$$\begin{aligned} K(x, x') &= \Phi(x)^\top \Phi(x') \\ &= \sum_{u\in\mathcal{A}^k} \phi_u(x)\phi_u(x') \end{aligned}$$

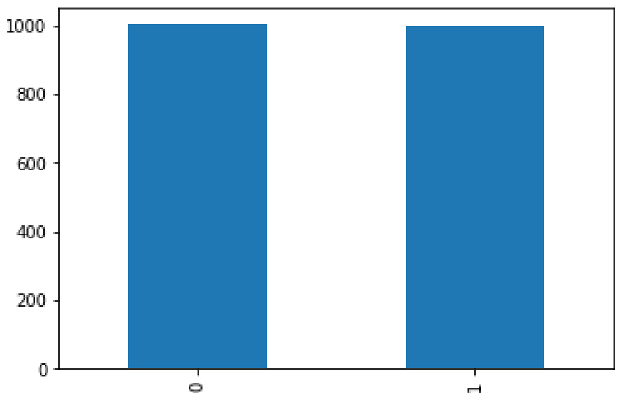
This is generally known as *spectrum kernel.*

Another well performing approach we tried was one-hot encoding (i.e assign $A$ to $(1, 0, 0, 0)$, $B$ to $(0, 1, 0, 0)$, $C$ to $(0, 0, 1, 0)$ and $D$ to $(0, 0, 0, 1)$), and then concatenate the binaries of each sequence which gave us 404 feature vectors. This method performed fairly well and it is so far the best performing representation of our data, even better than bag of words and spectrum kernel representation. We also tried concatenating one-hot features with additional features that was provided and encoded via bag of words, but this also did not improve our performance we achieved with only one-hot encoded features

## 3 Classifier model used



(a) *PCA on the data one-hot representation*



(b) *Distribution of the Dataset*

As can seen from the figures, the data seems to be perfectly balanced (see figure 1b), but unfortunately, the data is too mixed and it is very hard to design a perfect model that perfectly classifies the classes (see figure 1a). Another challenging part about the data was that it is too small and it introduces over-fitting most of the time. Therefore, to perfectly measure how our model is doing, we divided our data to train and validation sets at the proportion of $80\% - 20\%$.

In our work, we have tried two types of classifiers: *Kernelized Logistic Ridge Regression* (KLR) and *Kernelized Soft SVM* (KS-SVM). First, with one-hot encoding, we trained our *Soft SVM* using *RBF* kernel, and achieved an accuracy of around 65 on the validation set. We then concatenate the representation of the data given to us (*Xtr_100*) and the one-hot encoding, then train again with *Soft SVM* using *RBF* kernel, which gave us almost the same performance.

Similarly, we used *spectrum kernel* and this gave us same performance as the the one-hot encoding with *Soft SVM* and *RBF* kernel. In *spectrum kernel*, we have chosen $k$ randomly such that $k$ is not bigger than the length of the *DNA* sequence. We observed that the more $k$ is bigger, the more it performs well on the training set, which sometimes leads to overfitting. However, Due to computational expensiveness with large $k$ we couldn't go $k$ higher than 7. *Multiple spectrum kernel* was thereafter our next experiment where we did linear combinations of the *spectrum kernel* with different range of $k$ . We chose $k$ randomly and weighted each kernel with random coefficients that sum to 1. Finally, we have tried to repeat almost everything we have done but with KLR, which gave us small improvement over KS-SVM and gave us our highest score on leadership board.

On top of all, we optimized our implementation by storing all sparse vectors/matrix in a sparse format. Then any matrix multiplication were done in a very fast time. The *spectrum kernel* was also implemented in a matricized form to avoid the element-wise implementation which is known to be slow.

## 4 Conclusion

In conclusion, the data representation was very bad in such away that it is hard to classify. This is normal as it reflects most of the data that we deal with in real life. The classifiers we used are performing identically the same. So, the main important thing was building very good representation of the data and the choice of the kernels.