Kernel Methods for Machine Learning: Predicting whether a DNA sequence region is binding site to a specific transcription factor.
Team Name: Kernel InShape

Pirashanth RATNAMOGAN
MVA ENS Paris-Saclay
pirashanth.ratnamogan@ens-paris-saclay.fr

Othmane SAYEM
MVA ENS Paris Saclay
othmane.sayem@ens-paris-saclay.fr

Abstract

In order to implement and gain practical experience with kernel methods and with general machine learning techniques, the Kernel Methods for Machine Learning Class is ended by a data challenge. This year, the challenge is a binary classification task: predicting whether a DNA sequence region is binding site to a specific transcription factor.

1. Introduction

Beyond reaching the best score possible on the Kaggle competition leaderboard, during this competition we have tried to reach several other goals: apply what we have seen in class, try to apply some state-of-the-art techniques linked to the subject. In this report, we will first see the first ideas that we have implemented, then we will go through details about what implementation produced the best final score.

2. Building Kernel for DNA sequences

In class, we have seen plenty of Kernels that can be applied to DNA sequences. During our research, the most adapted technique for the task that we are trying to achieve, seems to be based only on the spectrum kernel and its variant the mismatch kernel. Because we have little expertise in the task that we are trying to solve, we wondered why only this quite simple Kernel was applied and why other were not used. Hence, we have implemented and tested many kernels that can be applied to our DNA sequences. Most of the following Kernels that we have implemented are precisely described in the class slides. In all our implementations, we have normalized the gram matrix.

2.1. Spectrum Kernel and Mismatch Kernel

The spectrum kernel is a relatively simple kernel that compares two sequences by comparing the number of common sub-sequences of a given length they have.
The mismatch kernel is a variant of the spectrum kernel where it can be allowed to have one or more mismatch in the compared sub-sequences.

2.2. String Kernel

The string Kernel is also a kernel that compares subsequences of two sequences but this time gaps within the sequence are allowed. Our implementation was largely inspired by Tim Shenkao github's one. This method seems really good but it took too much time to compute kernels of sequences with a length of 101.

2.3. Edit distance Kernel

A Kernel used for string comparison is the Edit distance Kernel. It compares sequences in a RKHS were the distance is defined by the so called Edit distance. This Kernel doesn't seem to be relevant for our task where composition is more important than position.

2.4. Fisher Kernel (HMM)

HMM fisher kernels are known to be state of the art for DNA sequences classification. Thus, inspired by the Jaakkola's paper, we represented the sequences as a Hidden Markov Model, for which we computed the maximum likelihood (using forward-backward algorithm of multiple sequences), and then the fisher vectors corresponding to the emission probabilities. We tried also to add the fisher vectors with respect to the transition probabilities of the HMM, to which we added the emission fisher vectors.

2.5. Fisher Kernel (GMM)

The challenge proposed some optional numeric files computed as a bag of word representation. Using what was explained on the class about: Aggregation of visual words using (fisher vectors), we have implemented a variant of this baseline using the same patches but we have aggregated them using the fisher vectors instead of the simple mean.

2.6. Local Alignement Kernel

Local Alignement Kernel is a really interesting Kernel that tries to compare subsequences by finding the cost to align them. An easy way to implement dynamic programming scheme was given in class. The problem with this method was that in order to make it performs one need to compute a substitution matrix that gives the cost to transform one letter into another one. However BLOSUM 62 doesn't performs well on our task so we tried to compute our own substitution matrix using the same method

as BLOSUM. However because our sequences were not aligned it doesn't provide good outcomes.

2.7. Graph Kernel

Another idea we tried is representing the sequences as sort of graph of words then computing some graph kernels. To create these graphs, we used a graph-of-words approach, by taking ngrams with a window of 2 (or 3). The nodes of our graph are the ngrams of the sequences, and the edges are the proximity of these nodes. Once all the graphs created for the dataset, we computed the graph kernels. We have tested the Shortest Path graph kernel, but also the Weisfeiler-Lehman and the Graphlet kernels. However, the classification results were not great.

2.8. Comparison of all the kernels

Our conclusion in our wide tests that actually the Spectrum Kernel and the Mismatch kernel were outperforming most of the other kernels for our task. What we understood was that composition of the DNA sequence is the key discriminative point for our task (much more than positional features). In addition to their really good performance the original implementation of the Spectrum and the Mismatch Kernel have the advantage to have a low computation complexity.

3. Building classifiers for our task

We have implemented various classifiers for this class. Indeed, some generative models (Fisher Kernel) needs to compute various models to be computed (HMM, GMM). In order to answer the binary classification task we have build two Kernel Classifiers.

3.1. Kernel Logistic Regression

We have implemented it using the algorithm given in the slides. Using it and the 6 spectrum Kernel we have obtained 73,466% accuracy on the public leaderboard.

3.2. Support Vector Machine

Using the library `convexopt` we have implemented a Support Vector Machine class that applies on gram matrices. Using the 6 spectrum kernel and SVM we have obtained 77,333% accuracy on the public leaderboard.

4. Our final model

As described in the previous sections the mismatch Kernel combined with SVM are performing the best on our test. Because the outcome was very sensitive we used various cross validation in order to find the rightful hyperparameters to make this models works at their best.

4.1. Choosing the kernel hyperparameters

In our sense there are mainly three parameters to set carefully: the length of the subsequences considered, the number of mismatches allowed and the gain associated to a subsequence that is found, up to a certain mismatch. Using cross validation and what we observe through our experiences with the data was the best parameters were to set them respectively to: 7, 1 and 0.1 (for all the datasets it seems to work good).

4.2. Choosing the classifier hyperparameters

First we did a big cross validation (grid search) in order to find the best C parameters for our SVM. However, what we have observed during our cross validation was that it was actually difficult to choose a good C parameter because the outcome depended a lot of the split between training and validation data. We have decided to set the C parameters to 0.5 that appears to be a good compromise (for all the datasets it seems to work well with the final regularization we used (next part)).

4.3. Using the ensemble methods to improve our algorithms regularization

Using the Kernel that we are using we are working in a vector space of size 16384 that is so huge in comparison with our number of samples (2000 to each dataset) leads to overfitting (not well dealed with the C parameter). However, the validation loss keeps being quite good in comparison with the other Kernels that we have tried. From those observations we have decided to use the so-called **bagging method** to improve our model generalization: for each dataset we have trained 10 models using 90% of the training samples sampled randomly, the average of those models gives us sort of probability to be in class 0 or 1. Finally, our final outcome is given by setting the outcome to 0 and if the obtained probability is lower to 0.5 and else 1. This methods has allowed to observe an incredible improvement in our finale outcome on our validations and on the public leaderboard.

4.4. Other ideas to improve the Mismatch Kernel

We have tried several other ideas in order to improve how our mismatch kernel was performing: weight all the dimensions of the embedding with their idf score through the datasets, combining various kernels of various size, considering a specific dataset we tried to use the positive samples of the two other datasets as negative samples for the current dataset, using classical kernels (rbf, polynomial...) in the embedding given by the mismatch kernel instead of using the basic linear kernel ... Unfortunately, none of these ideas allowed to obtain a better outcome but there exists certainly some basic tricks to improve the final outcome!

5. Conclusion

During this challenge we have tried to implement and test various Kernels that could be applied to DNA sequences. We have also used a lot of regularization and ensemble methods in order to get a good score. We have been pretty disappointed during the end of this challenge because we have been downgraded from the 2nd place in the public leaderboard (with almost 81% accuracy) to the 28th place (78% accuracy). We don't think that is due to some sort of overfitting because from what we have seen our final algorithm was working fairly well during our validation. We think that this is essentially due to the fact that our bagging deals fairly with some ambiguous samples in some cases (especially in the public leaderboard) but there are cases when it doesn't help a lot, and unfortunately it didn't help us in the private leaderboard. We don't think that this final outcome in the private leaderboard fairly represents the time that we spent on this project.. However, the challenge was pretty exciting and interesting, it allowed us to learn a lot!