Kernels Methods for Machine Learning Project Report

Lucas Morin
MSIAM M2 2020-2021

1 Project Overview

The goal of the project is to perform a classification task using kernel methods.
We want to predict whether a given DNA sequence is bounded or unbounded for a transcription factor of interest. Three data sets are given, corresponding to three different transcription factors. Each transcription factor describe a new classification task, then, three different predictive models should be implemented.

To build incrementally a complex model, one can start by working with a numerical embedding of DNA sequences. A first learning algorithm "Kernel Ridge Regression" (KRR) using kernels for numerical vectors is then implemented. Afterwards, one may implement "Support Vector Machines" (SVM) algorithm. Finally, new kernels working with DNA sequences (or strings) can be designed.

The code of the project is available at : `https://github.com/lucasmorin222/kernelmethods`.
It is structured as follows, "models.py" implements learning algorithms which are SVM and KRR. "kernels.py" implements kernels which are Gaussian Kernel, Polynomial Kernel and Mismatch Kernel (generalizing Spectrum Kernel). "tuning.py" is used for hyperparameters tuning. For this purpose, the annotated data is splitted into training and validation sets. Measuring performances on both sets allow us to detect underfitting and overfitting. "tools.py" implements functions for data reading and data augmentation. "main.py" implements the competition submission. **It achieves a score of 0.723 on the private leaderboard and 0.729 on the public leaderboard**.

To reproduce the submission, the script "start.sh" can be called from Python.

2 Learning Algorithms

The first learning algorithm implemented is **"Kernel Ridge Regression"**.The input labels are not modified, 0 for a unbounded sequence, 1 for a bounded sequence.

It is questionable to use this algorithm because we are not doing regression but classification. We still can predict a class by looking if the output of the algorithm is greater or less than 1/2. It could be interesting to implement "Kernel Logistic Regression" and compare their performances.

For KRR, the underlying optimization problem has a closed form solution.

The data used during training needs to be saved in the model. During the predict phase, we can calculate the similarity between new samples and training samples.

According to the representer theorem, the prediction for a given new sample is a combination of its similarities between training samples. The weights in this combination are learned during training.

The second learning algorithm implemented is **"Support Vector Regression"**. This time, the input labels need to be modified, -1 for an unbounded sequence and 1 otherwise.

The optimization problem is solved using an iterative method. The primal or the dual problems can be solved equivalently. We can consider 2 different optimization problems, regarding the loss function we may use. In the given implementation, the squared hinge loss optimization problem is solved using the primal, and the hinge loss optimization problem is solved using the dual.

For prediction, 2 functions are implemented. One of them only uses support vectors to speed up computation.

SVR and KRR involve an hyperparameter λ which controls regularization.

3 Kernels

To work with numerical data, 2 kernels are implemented : **Gaussian Kernel** and **Polynomial Kernel**. The degree of the polynomial kernel is an hyperparameter to be tuned. For the Gaussian Kernel, the scale parameter (gamma) was not tuned but calculated by replacing the variance (sigma) by the empirical variance of the data points.

The Mismatch Kernel was implemented to directly compute similarities between DNA sequences. The first step is to embed every data point x with a vector which number of dimensions is equal to the number of possible kmers of length k. k is an hyperparameter to be tuned. At a given index, this vector contains a great value if the corresponding kmer is found multiple times in the sequence x. Kmers almost matched (up to 1 or 2 mismatches), are also took into account, but they are given a lower weight. To decide which index of the embedding vector corresponds to a given kmer, we replace its nucleotides with 2 bits codes (00 for A, 01 for C, 10 for T, 11 for G). The index is the decimal representation of the sequence of bits. Finally, the Mismatch Kernel is just a dot product between vectors' embeddings. For runtime efficiency, the training points embeddings are saved.

4 Results summary

	Average public + private leaderboard	Hyperparameters
Kernel Rigde Regression + Gaussian Kernel	61,8%	Gamma : 65 Regularization : 1e-10
Support Vector Machines + Gaussian Kernel	63,9%	Gamma : 65 Regularization : 100
Support Vector Machines + Mismatch Kernel	72,1%	Length of kmers : 9 (and 8 for the last set) Regularization : 100 Number of mismatches : 2 Mismatches weights : [10, 5, 1]
Support Vector Machines + Mismatch Kernel + Data augmentation	72,2%	Length of kmers : 8 Regularization : 100 Number of mismatches : 2 Mismatches weights : [10, 5, 1]

5 To achieve better performances

To achieve better performances, **data augmentation** was performed. We can duplicate each training point by taking the reversed complementary DNA sequence. In practice, it did not improve significantly scores.

It could be interesting to evaluate more precisely each model, looking at other performance measures as the logloss and the confusion matrix. This could allow to combine models strengths using an **ensemble method**. Following the same idea, combinations of kernels could be used. For instance, we could consider a kernel which is the combination of several mismatch kernels with different kmers lengths.