Kernel Methods - Data Challenge

Iskander Gaba, Hadi Dayekh

February 19, 2021

Note: Code for this project is on https://mega.nz/folder/KYgCxBxA#xd9gKD_3AUKmcE2qNZqUEA

1 Introduction

The challenge of this project was to apply kernel methods to classify DNA sequences into bound or unbound sequences. In our work, we implement kernel ridge regression and investigate the performance of basic kernels such as linear, polynomial, and Gaussian kernels on simple numerically transferred dataset. We also apply string-specific kernels like k-spectrum and mismatch kernels on the raw version of dataset. Our results show that string kernels vastly outperform other approaches. On Kaggle's data challenge, our team ranked **13th out of 32** on the **public leaderboard** with an accuracy of **69.266%**, and **8th out of 32** on the **private leaderboard** with an accuracy of **70.133%**. The performance gap with the top performers was about 2% only, proving a comparable performance for kernel ridge regression and that our classification is consistent and well generalizable.

2 Main Steps

2.1 Kernel Ridge Regression (KRR)

The first step was to make a generic implementation of KRR. Our KRR model is wrapped in a `KernelRidgeRegression` class that is initialized with a kernel function reference and a list of kernel arguments to be passed to that kernel. Once a model is initialized, it can call three functions:

- `fit`: This function is used to fit training data. It computes the kernel gram matrix `K_train`, and the α values matrix from training data and caches them for later use. It then calls `predict_vals` function (which we will discuss shortly) to return predicted values of and predicted classes of the training data points.
- `predict_vals`: This function computes the predicted values `Y_vals` and the predicted classes `Y_pred` that are defined to be the sign of `Y_vals`. We return both `Y_pred` and `Y_vals` to the user at the end.
- `predict`: This function is a wrapper around `predict_vals` that only returns `Y_pred`.

We also improved the performance of our implementation by caching `K_train` as well as $\Phi(X_{train})$ in the cases of mismatch and k-spectrum kernels to avoid redundant computations.

2.2 Basic Kernels

Next, we used our KRR model with basic kernels like the linear, polynomial and Gaussian kernels. We first worked with `mat100` files. The best performance we could obtain using these files was $\approx 63\%$ from using Gaussian KRR. label-encoding raw data did not exhibit much of learning capacity so we soon discarded this approach.

2.3 String Kernels

2.3.1 *k*-Spectrum

We initially implemented a parallelized version of HASKER algorithm described in [1]. We then changed our implementation to use mismatch kernel, which we will discuss in the next section, with $m = 0$ since that implementation proved to be much faster in practice.

A coarse grid search on different values of λ and k narrowed our choice of λ to be between 0 and 1, and suggested that $k = 5$ and $k = 7$ were the best candidates. We then performed a finer grid search on 50 values of λ to improve our score, with the best validation scores summarized in the table below. Note that to be able to cross-compare, the training and validation sets were fixed to be respectively the first 80% and last 20% of the data for each dataset.

Dataset	k	λ	Validation Accuracy
0	7	0.05	66.25%
1	5	0.75	70.25%
2	7	0.47	78.00%

Table 1: k-Spectrum KRR.

2.3.2 Linear Combination of Kernels

Our next step was to use a combination of the $k = 5$ and $k = 7$ spectrum kernels, still working with KRR. Doing so, we were able to slightly improve the validation score for datasets 0 and 2. The table below shows the results, with w_1 being the weight associated to the $k = 5$ kernel and consequently $(1 - w_1)$ the weight of the $k = 7$ kernel. Our submission with the combination kernel improved our testing score.

Dataset	w_1	λ	Validation Accuracy
0	0.33	1.0	66.75%
1	0.78	0.78	70.25%
2	0.11	0.23	78.25%

Table 2: Linear combination of $k = 5$ and $k = 7$ spectrum KRR with w_1 being the weight of $k = 5$.

2.4 Mismatch Kernel

2.4.1 Implementation of the Mismatch Kernel

The idea to use the mismatch kernel was inspired by the nature of mutations in DNA sequences by modeling the number of occurrences of a subsequence u of length k up to m mismatches. We achieve that by computing all data mappings $\Phi(X)$ using `scipy` sparse matrices, and applying sparse matrix dot product between $\Phi(X)$ and $\Phi(X')$ to obtain the kernel gram matrix.
With a single mismatch ($m = 1$), we were able to improve our validation score and consequently test score for both datasets 0 and 1, whereas the combination of the spectrum kernels performed better for dataset 2. Hence, we submitted predictions of the first two datasets using the $m = 1$ mismatch kernel and kept the linear combination discussed above for the last datasets.

Dataset	k	λ	Validation Accuracy
0	8	0.3	67.75%
1	8	0.6	71.00%
2	7	0.9	76.75%

Table 3: Mismatch KRR ($m = 1$).

3 Conclusion

In this project, we implemented a kernel ridge regression model to classify DNA sequences into bound or unbound sequences. We investigated multiple kernel options and found out that Mismatch and linear combinations of k-Spectrum kernels produce the best performance. The best performance was obtained using the parameters shown in Table 4 below.

Dataset	Parameters	Val. Accuracy
0	$k = 8, m = 1, \lambda = 0.3$	67.75%
1	$k = 8, m = 1, \lambda = 0.6$	71.00%
2	k_1=5, k_2=7, w_1=0.11, λ=0.23	78.25%

Table 4: Optimal kernels.

On Kaggle's data challenge, we ranked **13th out of 32** on the **public leaderboard** with an accuracy of **69.266%**, and **8th out of 32** on the **private leaderboard** with an accuracy of **70.133%** and a performance gap of about 2% from top performers. The reader can train and produce a classification using the optimal parameters and kernels in Table 4 above by running `python start.py` from our code. A file named `Yte.csv` should be created under `data_processed` directory which contains the predictions for the test dataset.

References

[1] M. Popescu, C. Grozea, and R. Tudor Ionescu. Hasker: An efficient algorithm for string kernels. application to polarity classification in various languages. *Procedia Computer Science*, 112:1755–1763, 2017. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France.