Kernel Methods Kaggle Challenge

Matteo Sammut and Paul Caucheteux

April 12, 2024

1 Introduction

We are addressing a multi-class classification problem with the objective of accurately predicting the classes of images belonging to the *CIFAR10* dataset. It has 10 classes. The training dataset contains 5000 pictures, each with dimensions 32x32, and our task is to predict the labels for 2000 test samples.

The challenge was to implement Kernel Methods to solve this problem, adhering to the crucial "Do It Yourself" rule, meaning no use of ML libraries. Details rules and results are available here and the code is available there.

2 Implemented algorithms

2.1 Problem formulation

The multi-class classification problem can be formulate in many ways, either as regression, a multi-class classification, or multiples binary classification problems. Each formulation has its own pros and cons in terms of performance and computational efficiency.

Regression formulation. Even though it seems not really adapted to a classification problem, we implemented the **Kernel Ridge Regression** with a rounding operator at the end to match the classes $(0-9)$. While the training is very fast compared to the other methods and it successfully learns the training data, it fails to generalize to testing data even with a high regularization.

Multi-class formulation. The literature does not provide many kernel methods directly adapted to multi-class classification. The main reason is computational: the dimensions of the matrices involves in such formulations grow exponentially with the number of classes. We attempted to implement the Kernel Logistic Regression for the multi-class setting presented in [KPS07] but without success. The idea of this paper is to solve the multi-class classification with maximum likehood and traditionally reformulate Newton's method as an iteratively reweighted least-square. To reduce the computational challenges, they propose a alternating descent version that consist of solving a iteratively reweighted least-square problem for each class at each iteration. This formulation can be easily embedded in a distributed computing environment.

Multiples binary formulations. One way to solve the computational challenges is to apply bi-class classification Kernel methods to a modified Dataset. Two principal methods emerge:

- The **OnevsOne method**: We split the training Dataset into ten, respecting the classes, and we compute a bi-class classifier to separate each class. We obtain $K(K-1)$ classifier. Then, for a new example, we predict with a 'majority voting' principle, i.e., we predict the class which has been the most predicted by the $K(K-1)$ classifiers.
- The **OnevsAll method**: For each class, we segment the dataset into two parts: one where the data points belong to the target class, and the other where the data points belong to any class but the target. We then label the first part as 1 (indicating presence of the class) and the second part as -1 (indicating absence of the class). We obtain K classifier. Then for a new example we predict the class corresponding to the classifier that outputs the higher score.

As the dataset is calibrated, these methods are really adapted to our problem. In fact the OnevsOne method will be the preferred choice because it reduces computational complexity by focusing on smaller

data segments—specifically, each classifier uses only 1/5 of the dataset for optimization. For the rest of this report, we will consider this *multiple binary formulation*.

2.2 Multiples bi-class models

The *multiples bi-class formulations* necessite to use a binary classification model. The most natural is the **Kernel Logistic Regression**. We first implemented this model with the *optimize* package, which worked but wasn't optimal. So, we implemented the iteratively reweighted least-square (IRLS) optimization method to resolve it.

We also implement the **Kernel SVM** which is supposed to have better accuracy for bi-classification problem. We first implemented this model with the *optimize* package, but due to the large dimensions, we gained a lot of precision by using the *cvxopt* package.

2.3 Kernels

First we implemented two kernels: polynomial and RBF. However, the computation of the RBF kernel on the full training set took too much time due to the large dimension. Thus, we implemented the **Nyström approximation** to compute the RBF kernel. Later we found a more efficient way to compute the RBF kernel and therefore we no longer use this approximation.

3 Experiments

We developed functions to extensively test various hyperparameters, enabling us to evaluate the accuracy for each combination. These parameters included the variance of the RBF kernel, the degree of the polynomial kernel, and the regularization parameter for both SVM and Logistic Regression. Additionally, we experimented with integrating PCA with SVM to simplify the SVM optimization by reducing the dimensionality of the images.

It revealed that SVM outperformed traditional logistic regression in terms of accuracy. However, the combination of PCA and SVM did not enhance the performance, suggesting that dimensionality reduction might not always be beneficial for this dataset. Importantly, the results highlighted the significant impact of hyperparameter tuning on the final accuracy, underscoring the need for careful selection and evaluation of these parameters to achieve optimal performance.

References

[KPS07] Peter Karsmakers, Kristiaan Pelckmans, and Johan A. K. Suykens. Multi-class kernel logistic regression: a fixed-size implementation. In *2007 International Joint Conference on Neural Networks*, pages 1756–1761, 2007.