Methodology of Creating SVM Kernels from Scratch Using Python and NumPy

Austin Lackey and Tomy Sabalo Farias

DSCI 320, Colorado State University

December 13, 2023

Abstract

This paper presents a methodology for creating Support Vector Machine (SVM) kernels from scratch using Python and NumPy. We discuss the implementation of linear, sigmoid, polynomial, and radial basis function (RBF) kernels in a binary and multiclass SVM. These kernels are tested on E. coli data and compared to the results of the scikit-learn SVM implementation. The results show that the implemented kernels often yield better accuracy than the scikit-learn implementation; at the cost of increased training time. Kernel training times are ran multiple times and averaged to provide a more accurate metric since training times are low and have a high variance.

1 Introduction

In this section, provide an introduction to SVMs, their applications, and the importance of kernels in SVMs.

2 Methodology

In this section, describe the methodology used to create the SVM kernels from scratch.

2.1 Kernel Functions

We implemented four different kernel functions which are listed below.

- Linear Kernel
$$K(X, Y) = X^T Y \tag{1}$$
- Sigmoid Kernel
$$K(X, Y) = \tanh(\gamma X^T Y + r) \tag{2}$$

- Polynomial Kernel

$$K(X, Y) = (\gamma X^T Y + r)^d, \gamma > 0 \tag{3}$$

- Radial Basis Function (RBF) Kernel

$$K(X, Y) = \exp(-\gamma ||X - Y||^2), \gamma > 0 \tag{4}$$

2.2 Binary SVM

Discuss the implementation of the Binary SVM class, including the implementation of the different kernels.

2.3 Multiclass SVM

Discuss the implementation of the Multiclass SVM class, which uses the Binary SVM class.

3 Results and Discussion

In this section, present and discuss the results obtained using the implemented SVM kernels.

Table 1: Classifier Performance

Classifier	**Implementation**	**Kernel**	**Avg Accuracy**	**Avg Runtime**
Binary	sklearn	linear	1.00000	0.00052
Binary	custom	linear	0.99318	0.00795
Binary	sklearn	sigmoid	0.68180	0.00066
Binary	custom	sigmoid	0.99546	0.00757
Binary	sklearn	rbf	1.00000	0.00054
Binary	custom	rbf	0.99546	0.00977
Binary	sklearn	poly	1.00000	0.00040
Binary	custom	poly	0.97498	0.00929
Multi	sklearn	linear	0.77940	0.00096
Multi	custom	linear	0.81468	0.15420
Multi	sklearn	sigmoid	0.47060	0.00183
Multi	custom	sigmoid	0.82497	0.15641
Multi	sklearn	rbf	0.75000	0.00156
Multi	custom	rbf	0.82350	0.19177
Multi	sklearn	poly	0.76470	0.00111
Multi	custom	poly	0.83967	0.36326

Note: These times were calculated with 10 runs for each kernel and averaged.

4 Conclusion

In this section, provide a conclusion summarizing the work done and its implications.