Project 2: The SMO Algorithm

Sam Beckmann

April 28, 2017

1 Introduction

Project 2 was about programming a simplified version of the SMO algorithm for generating support vectors, and testing the algorithm with several datasets found online. I wrote my implementation in Python, which had pros and cons that I will later discuss. I chose to test SMO against the following five datasets, in addition to Nate's two spirals dataset: banknote authentication,[1] SPECTF heart,[2] breast cancer Wisconsin (Diagnostic),[3] Pima Indians diabetes,[4] Connectionist Bench (Sonar, Mines vs. Rocks).[5]

2 Implementation

I implemented the SMO algorithm in Python. Although Python made it quick and easy to catch errors and learn what the algorithm is doing through prototyping, if I were implementing SMO in any sort of production environment, I would want to write it in a faster, compiled language, such as C, as I did not realize how computationally intensive the algorithm would be. In fact, event the simple trials mentioned in this report were run overnight in order to be completed. In addition, I would look into ways to multi-thread the implementation to take advantage of computer hardware. The SMO algorithm focuses on optimizations of many pairs of data points, which would easily lend itself to multi-threading.

For SMO parameters, I used C = 1e10, tol = 1e-7 and $maxpasses$ = 10.

3 Testing

For testing purposes, I used two kernels, a simple dot product, $K(x, z) = xz$, and a Gaussian kernel, $K(x, z) = e^{-\frac{\|x-z\|^2}{2\sigma^2}}$ with a σ value of 1. Each data set

[1] https://archive.ics.uci.edu/ml/datasets/banknote+authentication
[2] https://archive.ics.uci.edu/ml/datasets/SPECTF+Heart
[3] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[4] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
[5] https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29

	Gaussian		Dot Product	
	μ	σ	μ	σ
Bank Notes	65.8	1.1	—	—
Diabetes	65.4	2.2	—	—
Diagnostic	64.5	3.9	—	—
Mines	63.5	4.2	—	—
SPECTF	91.8	2.7	93.6	3.2
Two-Spirals	59.7	6.4	—	—

Table 1: Results

was run with 5 trials on each kernel. The data sets were divided up to be 70% training and 30% testing. The trials were run overnight, and any trials not finished by the next morning were considered to not have converged.

4 Results

The mean and standard deviations of the results are presented in Table 1. Note that a dash indicates the algorithm did not converge in my overnight test. All reported values are in percentages.

5 Discussion & Conclusion

My initial perception of the SMO algorithm was that the accuracy of the result would be largely dependent on the choice of kernel. To my surprise, I found the question wasn't how accurate the results would be, but if the algorithm would converge at all in a reasonable time frame. Only one trial of the dot product matrix actually yielded results. Although the results it did yield were approximately the same as those created by the Gaussian kernel, the Gaussian kernel reached its result much quicker. Since the quality is comparable and the Gaussian kernel was much faster in all trials, it appears that the Gaussian kernel is better suited to nearly all data sets than a simple dot product.

I also was struck by how much the data set affected convergence time for SMO. The SPECTF data set would routinely accurately converge in less than 30 seconds, while the bank notes dataset could take upwards of a half hour, and not converge with nearly as high of accuracy. From this, I conclude the SMO is best suited to smaller datasets that need high accuracy than huge datasets. As the algorithm seeks out all pairwise optimizations, it scales exponentially withe dataset size. With larger datasets, I believe a smaller training set may be needed to converge within a reasonable time frame.