ECE 236A: Final Project, Part I

A 1-Norm Support Vector Machine for Linear Classification

David Martinez & Peter Racioppo

Introduction/Problem Statement

We consider the problem of creating a multi-class classifier using techniques from linear programming. A straightforward approach to linear classification is to draw hyperplanes in the n-dimensional feature space of the data in order to partition the space into classes. Classification is performed on the "Pen-Based Recognition of Handwritten Digits Data Set." Each data point belongs to one of ten classes (corresponding to the digits 0 through 9) and contains 16 features, corresponding to the xy-coordinates of equidistantly sampled points from the drawings. We use 7,494 data points as a training set and 3,498 points for testing. We follow an approach based on 1-norm support vector machines. We then consider the problem of modifying the algorithm to be robust to random corruption of some of the features.

Background

A set of points $\{v_1, ..., v_N\}$ with binary labels from {+/- 1}, is said to be linearly separable if it is possible to find a hyperplane that strictly separates the two classes. Since the margin size is inversely proportional to the square of the w-vector, the general support-vector machine minimizes the objective function:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x}_i - b)\right)\right] + \lambda\|\vec{w}\|^2$$

Here, λ is a constant that determines the tradeoff between maximizing the margin size and minimizing the hinge-loss function. Optimization is typically performed using either quadratic programming, a sub-gradient method, or coordinate descent. We can achieve significant efficiency improvements by replacing the squared 2-norm in the objective function with a 1-norm, thus transforming the problem into a linear program. The 1-norm also tends to encourage sparsity in the training parameters, and is thus an effective means of regularization.

Classifier I: Training

In order to adapt the SVM algorithm to more than two classes, we separately train an SVM between every class pair. Thus, for all *i* and *j* from 1 to *m* (the number of classes), we train a hyperplane between all the points in classes *i* and *j*. Thus, our algorithm requires $N = m(m-1)/2$ hyperplanes for data with *m* classes (for $m=10$ classes, this comes to $N=45$ hyperplanes).
We performed *k*-fold cross-validation on the data, which helps to prevent overfitting. We removed 10% of the data from each of the 10 classes as a testing batch, which was not used for training. The remaining data was split into *n* separate batches (default $n=5$), each having an equal number of samples from each class, and a 45-hyperplane 1-norm binary SVM was fit for each batch. We averaged each possible combination of the *n* sets of hyperplane parameters W

and w0, a total of $2^n - 1$ sets, and selected the SVM which performed best on the testing batch. Though exponential in the number of batches, this step isn't cost prohibitive, since we can limit the number of batches to a small number. The value of λ was determined by a line search on this hyperparameter. The inclusion of the λ term in the optimization improves robustness by maximizing the size of the margin.

Classifier I: Testing

Testing employs a voting system. Each hyperplane computes on which side of the hyperplane a data point falls and then "votes" for the corresponding class, and the classifier chooses the class with the most votes. In the event of a tie, the classifier chooses the class containing the hyperplane with the largest margin (that is, the distance from the hyperplane). Since the margin is a measure of classification confidence, this amounts to choosing the vote of the hyperplane which is most "sure" of its vote. A slightly more sophisticated option to break ties would have been to use a weighted sum of the margins.

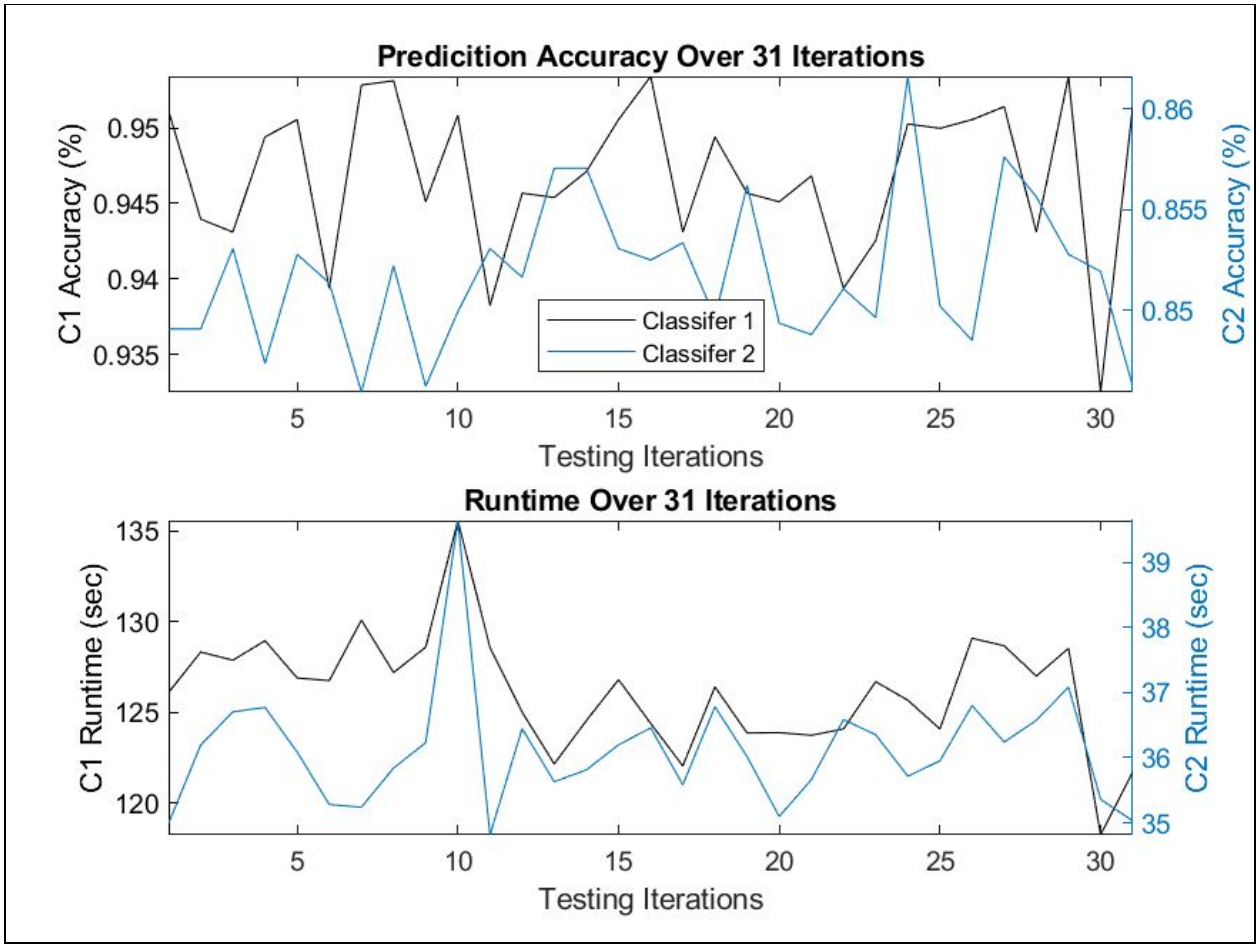
Classifier II

Clearly, much of the information encoded in Classifier 1's hyperplane/voting system is redundant. The goal of classification is not to distinguish between every class-pair but between the data included/excluded by one class. Thus, we alter the algorithm to draw only n hyperplanes: for each class, we draw a hyperplane between the set of points in that class and the set of points not in that class. The number of hyperplanes is thus reduced from $O(n^2)$ for Classifier 1 to $O(n)$ for Classifier 2. We employ the same batching/cross-validations methodology as in Classifier 1. The voting system is likewise unaltered, except that each hyperplane no longer chooses between two classes, but instead makes a binary decision on whether a point belongs to a class. Ties are again decided by the maximum margin.

Results and Discussion

Classifier 1 averaged 94.69% testing accuracy, with an average run time of 126 seconds.
Classifier 2 averaged 85.17% testing accuracy, with an average run time of 36 seconds.
Using same $\lambda = 10.778$

From Figure 1 and the averages, it can be clearly said that Classifier 1 is much more accurate than Classifier 2 on every iteration. Although, this accuracy comes at a very high trade off of time. To train and test the data, it takes approximately 4 times more time to use Classifier 1 compared to 2. For real-time applications, an algorithm like Classifier 2 would be much better for computational speed.



Conclusion

Introducing a 1-norm into the SVM cost function improves robustness and prevents overfitting, as compared to a vanilla linear classification algorithm, and improves efficiency and sparseness characteristics, as compared to a 2-norm SVM. Classifier 1's $O(n^2)$ hyperplane voting system proved capable of correctly classifying some 95% of testing data. Classifier 2's $O(n)$ voting system removed some of Classifier 1's redundancy, but at the cost of an almost 10% drop in testing accuracy. Future work should focus on the use of the kernel trick to map the data to higher dimensions during the training process.