Analysis of the MNIST Data Set with LIBSVM Software and K-Nearest Neighbors

Anna Ayzenshtat

December 6, 2012

Abstract

LIBSVM is a way to implement various SVM kernel classifications on data sets with relatively low start-up cost. This paper examines the use of LIBSVM to classify handwritten digits from the classic MNIST data set. While fairly accurate performance is achieved, both accuracy and run time can be improved with more advanced techniques not provided by LIBSVM. We compare SVM performance against K-nearest neighbors performance.

1 Introduction

1.1 Support Vector Machines (SVM) Overview

Given a training set of N feature vectors $x_i \in \mathbb{R}^d$ with corresponding labels $y_i \in \{-1, 1\}$, SVM produces a decision function which predicts labels of test data given the test data features. In particular, SVM solves the following optimization problem:

$$\begin{aligned} \min_{\vec{w},b,\vec{\xi}} \quad & \frac{\|\vec{w}\|_{\ell_2}^2}{2} + C\|\vec{\xi}\|_{\ell_1} \\ \text{subject to} \quad & y_i(\vec{w}^T\phi(\vec{x}_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \quad \forall i \in [N], \end{aligned} \tag{1}$$

where $\phi : \mathbb{R}^d \to \mathbb{R}^m$ for some $m \geq d$ is a function that maps the training vectors x_i from a d-dimensional space into some higher m-dimensional space. The purpose of the function ϕ is to classify data that are not linearly separable by SVM in d dimensions but are linearly separable by SVM in higher dimensions. The ξ_i's are called slack variables and give the method some stability with respect to outliers. $C > 0$ assigns how much penalty we impose for too much slack.

Figure 1: Sample images from the MNIST dataset

The function ϕ in turn defines the kernel function $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$. Different kernels work well for different data sets. The following kernels are available in LIBSVM:

- linear: $K(x_i, x_j) = x_i^T x_j$
- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^s, \quad \gamma > 0$
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2), \quad \gamma > 0$
- sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$,

where γ, r, and s are parameters.

2 Obtaining and Preprocessing the Data

2.1 MNIST Data Set

The data set was obtained from kaggle.com, a data analysis contest site. Although kaggle.com provides both a training set of 42000 handwritten digits and a testing set of 18000 handwritten digits, their testing set excludes true answers. Thus, we used the so-called training set provided for both training and validation by training on a random 3/4 of it, then validating on the remaining 1/4. Each row of the data set contains a 28^2-pixel encoding for a handwritten digit on a grayscale of values between 0 and 255. Each digit can be then viewed on a 28×28 pixel grid (see Figure 1) .

2.2 Preparing MNIST for SVM

The authors of the LIBSVM software, Chih-Chung Chang and Chi-Jen Lin, state that scaling before applying SVM is very important, citing that the main advantages of scaling are to avoid attributes in greater numerical ranges from dominating those in smaller

numeric ranges and to avoid computational problems for LIBSVM. For example, the issue of scaling arises while computing kernel values. In particular, the linear, polynomial, and sigmoid kernels depend on inner products of the feature vectors and large feature values might obscure subtleties of the data. In addition, large inner products can cause numerical problems for LIBSVM.

The authors thus recommend scaling each feature to the range $[-1, +1]$, to be determined by parameters of the training data. The testing data will then be scaled by the same parameters as the training data and will then not be necessarily restricted to the range $[-1, 1]$. The reason we cannot scale the testing data separately to also fall in the range $[-1, 1]$ is that all decision parameters need to be set by the training set before the decision-making processing begins. Thus, not only do the $\vec{w}, b, \vec{\xi}$ parameters found by SVM with respect to the training set belong to the set of decision parameters, but so also do the scaling constants of the training set. These parameters are then used to to classify all future data, which will be the testing data.

For our implementation, we scaled each fixed column of a training set as follows:

- We subtracted the column min value from each entry of the column.
- We divided the resulting column by the difference of the column max value minus the column min value to get each value between 0 and 1.
- We multiplied the entire result by 2, then subtracted 1 to get each value between -1 and 1.

For more, see [1].

2.3 How LIBSVM Fits a Multi-Class Problem into a Binary Framework

It is important to note that SVM is a binary, not multi-class classifier. The way LIBSVM performs multi-class classification is by training $\binom{10}{2} = 45$ separate SVM classifiers. For example, one of the 45 SVM decision functions answers the question, "Is this a 2 or a 7?" Each testing point is then classified by each of these 45 SVM classifiers. The digit with majority vote out of the 45 is the final decision.

3 Using LIBSVM

3.1 Application of Linear SVM using LIBSVM to a Small Multi-Class Example Set

Before testing out the linear and RBF SVM kernels on the vast MNIST handwritten digit data set, we needed to acquaint ourselves with the LIBSVM built-in Python wrapper. We also needed to confirm that the simplest of the SVM kernels, the linear one, indeed performs as expected. To do so, we randomly generated points in $[-5, 5] \times [-5, 5]$, assigning points in the first and fourth quadrants to be red, points in the second quadrant to be green, and points in the third quadrant to be blue (see Figure 2). Using the Python wrapper, we ran LIBSVM with the linear kernel classifier on another set of 10,000 randomly generated points in $[-5, 5] \times [-5, 5]$. The classification performance can be assessed graphically in Figure 3. The linear classifier appears to do well, as expected since the training points generated are linearly separable, with possibly some errors on the boundaries of the quadrants.

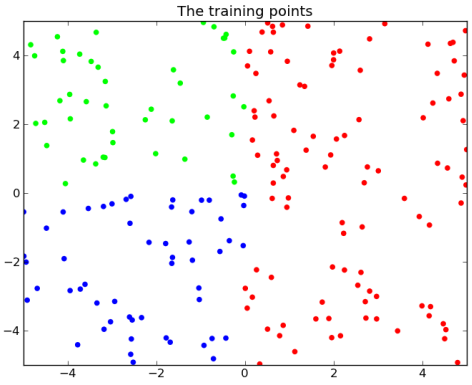

Figure 2: Sample data set: reds are generated in 1st and 4th quadrants, greens in the 2nd quadrant, blues in the 3rd quadrant.

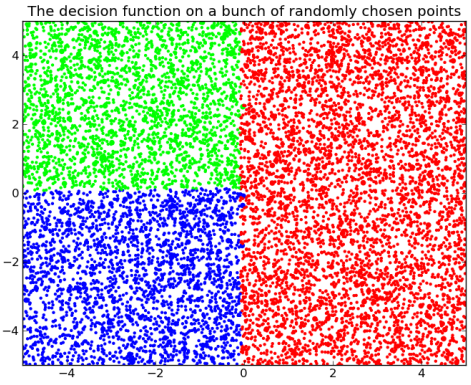

Figure 3: Performance of Linear SVM in classifying randomly generated points in the plane.

3.2 Performances of Various Kernels on the MNIST Data Set

Recall the linear kernel $K(x_i, x_j) = x_i^T x_j$. Recall also the parameter C of Equation 1. We ran LIBSVM using the linear kernel with various values of C in $\{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$. See Table 1 for some of the results.

Table 1: Performance of Linear Kernel

C	Accuracy (%)
0.001	92.6476
0.01	93.7762
0.02	93.8143
0.03	93.7571
0.1	93.1952
1.0	92.2000
10	91.4714
100	91.1667

Recall the polynomial kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^s$, where $\gamma > 0$ and s are parameters. We ran LIBSVM using the polynomial kernel with various values of C in $\{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$ and various values of s in $\{5, 6, 7, 8, 9\}$. We used the default LIBSVM value for $\gamma = \frac{1}{\text{number of features}} = \frac{1}{28^2} = \frac{1}{784}$. It appears that higher accuracies are achieved as C increases. Some of the highest accuracies witnessed for some of the parameter pairings mentioned above were $C = 20, s = 8$ and $C = 100, s = 9$ (see Table 2). To make sure the witnessed accuracies weren't an accident, we ran 3-fold cross-validation to get an average accuracy for each pairing. The results are listed in Table 3 below.

Table 2: Performance of Polynomial Kernel

C	s	Accuracy (%)
0.005	4	68.409524
0.005	6	41.4857
0.01	2	86.285714
0.01	7	62.4190
0.02	2	89.380952
0.2	3	94.7429
1	3	96.542857
2	6	97.552381
5	9	97.990476
20	4	98.066667
20	7	98.361905
20	8	98.3524
100	9	98.2603
100	9	98.1048
100	9	98.4286

Table 3: Cross-Validated Performance of Polynomial Kernel

C	s	Average Accuracy
20	8	98.0762%
100	9	98.2603%

Now recall the RBF kernel $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$, where $\gamma > 0$ is a parameter. We ran LIBSVM using the RBF kernel with various values of C in $\{2, 5, 10, 20, 50, 100, 200, 500, 1000\}$ and γ in $(0, \frac{1}{784}]$ (see Table 4). Some of the highest accuracies were achieved for $C = 10, \gamma = 0.007$ and $C = 100, \gamma = 0.007$. Again, to make sure these readings weren't an accident, we ran 3-fold cross-validation to get an average accuracy for each pairing. The results are listed in Table 5 below.

Table 4: Performance of Radial Basis Function Kernel

C	γ	Accuracy (%)
0.1	0.02	74.152381
0.1	0.005	94.7810
1	0.02	95.7905
10	0.004	97.8000
100	0.007	97.885714
100	0.007	98.2476
500	0.004	97.8095
500	0.006	97.8381
500	0.01	97.828571
1000	0.007	98.057143

Table 5: Cross-Validated Performance of Radial Basis Function Kernel

C	γ	Average Accuracy
10	0.007	98.1429%
100	0.007	98.0667%

The best results were achieved by the polynomial and RBF kernels.

4 Performance of K-Nearest Neighbors

We ran K-nearest neighbors for a variety of K values, again training on a random 3/4 of the data and testing on the remainder. We preprocessed the data in the same manner as for SVM. Using 6-fold cross-validation, we obtained the results listed in Table 6.

Table 6: K-Nearest Neighbors Accuracies Using ℓ_2 distance

K	Average Accuracy (%)
1	96.6
3	96.7
5	96.6
7	96.5
9	96.3
11	96.1
13	96.0
15	95.8
17	95.7
19	95.6
21	95.4
23	95.3
25	95.2
27	95.1

It appears that $K = 1, 3, 5$ perform the best, with monotonic decline in accuracy thereafter. Average accuracy for K-nearest neighbors is lower than that obtained by SVM with the polynomial and RBF kernels.

5 Summary and Comments

SVM as implemented by LIBSVM performs reasonably well on the MNIST handwritten digits. Run time time, however, is a problem. Training 45 classifiers, then testing them out on each of 10500 validation points takes a long time. Each 3-fold cross-validation takes about 30 minutes on a Core i5 Macbook Air. There are possibly faster approaches. Run times can also probably be improved by using some form of dimensionality reduction, ilke PCA, in the preprocessing step.

Accuracy would most likely increase if more of the MNIST data set is used. For example, Corinna Cortes and Yann LeCun, who have worked extensively on this data set, trained

their various algorithms, including SVM, on 60000 MNIST digits and tested on 10000 (see [2]). The data set obtained from kaggle.com contains 60000 digits in total. Only 42000 are labeled, so we used this subset for both training and validation. The data set from kaggle.com was used in the interest of time; it is provided in a nice .csv format whereas the data set provided on Corinna Cortes and Yann LeCun's website requires a lot more preprocessing.

A technique to explore might be Virtual SVM (see [2] and [3]), which reduces computational cost and increases classification accuracy. LIBSVM does not exercise this technique as an option.

K-nearest neighbors not only has lower accuracy than SVM with the polynomial and RBF kernels, but it also has a longer run time. It took 6 hours to perform 6-fold cross-validation. Most of that computation time is taken up by calculating distances between validation points and training points.

References

[1] Chih-Chung Chang, Chih-Wei Hsu, and Chih-Jen Lin. A practical guide to support vector classification, April 2010.

[2] Corinna Cortes and Yann LeCun. The mnist database of handwritten digits.

[3] Dennis Decoste and Bernhard Scholkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002.