Support Vector Machines: Hand-Written Digit Recognition

Tyler McDonnell
tyler@cs.utexas.edu

1 Introduction

Support Vector Machines are powerful supervised learning models widely used in practice for classification and regression analysis. Given a set of N labeled examples, each defined by a feature vector $\vec{x}$, the SVM training algorithm builds a model that assigns new examples into one of the class labels from the training data. Formally, SVMs construct a separating hyperplane between the two classes of data points in $\mathbb{R}^D$, where D may be equal to $\|\vec{x}\|$ or much larger.

In these experiments, I will apply SVMs to the specific problem of handwritten digit recognition and compare their performance to that of the simple, but effective, K-Nearest Neighbors Algorithm.

2 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a very simple, but surprisingly powerful, classification algorithm. Intuitively, KNN takes an observation $\vec{x}$ and classifies it according to the simple majority of the K examples that are closest to $\vec{x}$ in some sense. lassification algorithm. Intuitively, K-Nearest Neighbors takes an observation $\mathbf{x}$ and classifies it according to examples in our training data that are similar or "close" to $\mathbf{x}$ in some sense. Since $\mathbf{x}$ is an input vector of dimensionality X, we can think of it as a point in $\mathbb{R}^X$. Thus, one simple formulation of KNN is to find the K examples from our training set that are closest to $\mathbf{x}$ in $\mathbb{R}^X$ in terms of Euclidean distance, and take the simple majority among the labels of these "neighbors".

The Euclidean distance in $\mathbb{R}^n$ is defined as:

$$d(p,q) = \sum_{i=1}^{n} (q_i - p_i)^2$$

3 Support Vector Machines

As previously mentioned, SVMs construct separating hyperplanes between classes of data points in some, often very high-dimensional, data space. The particular strength of SVMs is that they also try to maximize the so-called functional margin, or the distance between the separating hyperplane and the nearest training data point from any class. In practice, larger functional margins tend to reduce the generalization error of a classifier, so SVMs tend to be very powerful linear classifiers.

The classic formulation of SVMs treats the construction of the separating hyperplane as an optimization problem. If the training data are linearly separable, we can construct two parallel hyperplanes that separate the classes of data; the maximum-margin hyperplane that we seek is then the hyperplane halfway between them. Thus, we can formulate an optimization problem where we wish to minimize the normal vector of the hyperplane subject to the constraint that each data point in the training set must lie on the correct side of the margin (i.e., each data point is correctly classified by the separating hyperplane).

However, in many cases, the training data may not be linearly separable. Luckily, SVMs can handle this as well. The so-called *kernel trick* can handle non-linear classification by transforming the feature space into a higher dimension in which the data is linearly separable. Since the VC dimension of a linear classifier is $N+1$, we know that for any finite training set, there exists a dimensionality where we are guaranteed that a linear classifier can separate the data. These kernel functions are carefully formulated as inner products, making the feature space transformation computationally feasible.

3.1 Implementation

The SVM implementation found in Matlab's Statistics and Machine Learning Toolbox was used for the experiments in this paper. Recall that SVMs are binary linear classifiers, and here we are interested in classifying digits in the class set {0-9}. Fortunately, it is possible reduce multi-class problems into a series of binary classification problems, and the Machine Learning Toolbox also offers hooks for handling this transformation. For these experiments, I used the one-versus-one coding design paradigm, which learns $(K(K-1))/2$ binary classifiers, one for each pair of classes, in order to provide a multi-class label.

4 Experiments

The following sections describe several experiments in detail. The first two experiments compare SVM and KNN performance while varying the dimensionality of the input data and number of training examples. The final section takes a look at some of the important tune-able knobs when deploying SVMs in practice.

4.1 SVMs vs. KNN: Learning Curve

Figure 4.1 compares the performance of an SVM using a Gaussian Radial Basis Function kernel and KNN across numerous training data sizes for all 10,000 test images. Not surprisingly, SVMs offer considerable performance gains when large amounts of training data are available.

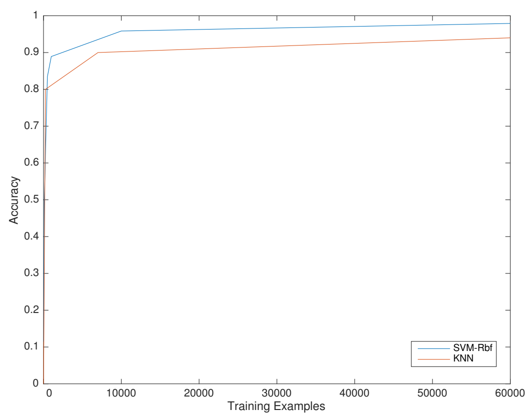

Figure 1: Learning Curve: SVM vs. KNN

Interestingly, however, KNN actually outperforms SVMs when the amount of available training data is very small in this particular problem. One hypothesis for this behavior is that the feature vectors in this case are very sparse, and the subset of features important for discerning handwritten digits is very small. For one, this is what makes handwritten digits such a powerful example for motivating PCA. But it also means that Euclidean Distance-based KNN is very effective: unimportant features are often zero. At any rate, for even modest data sizes, SVMs quickly begin to outperform KNN by significant margins.

4.2 SVMs vs. KNN: Dimensionality of Input

Figure 4.2 compares the performance of SVMs using a Gaussian Radial Basis Function kernel with KNN for various dimensionalities T, where the dimensionality reduction is accomplished via PCA. In this figure N in the model description is the number of training examples used. Just as in the case of KNN, PCA is a powerful tool for SVMs which actually provides even higher classification accuracy for low values of T than the original feature space. We also see that for most reduced dimensionalities, SVMs perform significantly better than KNN. Interestingly, the performance of SVMs appears to drop off faster than KNN when PCA is performed but a large number of eigenvectors are maintained.

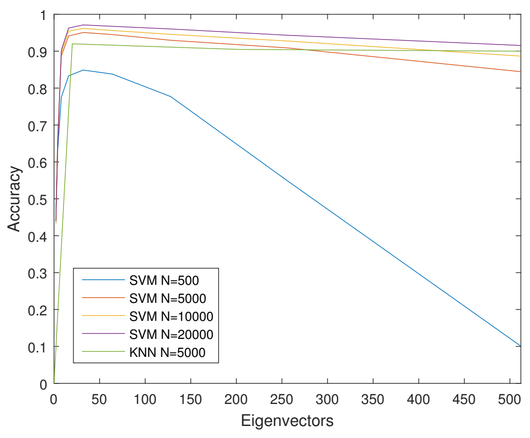

Figure 2: Learning Curve: SVM vs. KNN

4.3 SVMs: Run-Time

We have seen that SVMs almost unilaterally outperform KNN by significant margins for practical training sets and feature spaces. It is now important to consider what other trade-offs might beat work with SVMs. One convenient aspect of KNNs is that in their simplest formation, they require no training time: we are simply computing nearest neighbors according to some distance metric in the input data space at test time. We might perform some dimensionality reduction beforehand, but we will not consider that part of the KNN training time. Unfortunately, KNN pays for this non-existent train time during the classification phase, when it must compute the nearest neighbors for each test sample in order to classify it. However, slightly more sophisticated formulations of KNN can pay some upfront cost to carefully formulate the search space (e.g., by building a K-D tree) to speed classification.

Table 4.3 shows the training and test times for a few select training and test sizes. Again, these numbers are from a common desktop machine and are meant to give a general picture of time complexity, not precise benchmarks. As we can see, SVMs can incur significant training times, and generally, the training time will far outweigh classification time. In fact, the time complexity is to difficult to nail down, though a commonly cited approximation for SVMs using the rbf kernel is $O(N_{examples}^2 * N_{features})$. In other words, for this particular problem, the run-time is dominated by the number of training examples. More generally, dimensionality reduction is not a complete solution for driving down the training time of SVMs, though it is still an important and powerful tool.

Model	Training Size	Test Size	Train Time	Test Time	Total	Accuracy
SVM-RBF	5000	5000	6	13	19	0.89
SVM-RBF	5000	10000	6	27	33	0.89
SVM-RBF	60000	10000	104	53	157	0.95
SVM-RBF	60000	10000	415	143	558	0.95

Train, Test, and Total times are in seconds.

Table 1: SVM Run-Time Comparison

4.4 SVMs: Kernel Function Selection

Kernel functions affect both the accuracy and run-time for SVM training and classification. Table 4.4 shows the accuracy and execution time (train + test) of various common kernel functions for digit classifications using $N = 60,000$ training examples over all $10,000$ test cases. The execution time is in seconds on a standard desktop computer and meant to give a general, not a precise, picture of performance of the various kernel functions.

Kernel Function	Training Accuracy	Test Accuracy
Linear	0.9341	0.9364
Gaussian	0.9942	0.9791
RBF	0.9944	0.9796
Polynomial Order-1	0.9340	0.9366
Polynomial Order-2	0.9932	0.9814
Polynomial Order-3	0.9992	0.9841
Polynomial Order-4	0.9999	0.9830
Polynomial Order-5	1.0000	0.9823
Polynomial Order-6	1.0000	0.9814
Polynomial Order-7	1.0000	0.9799

Trained on N=60,000 examples.

Table 2: Kernel Function Comparison

As we can see, most kernel functions perform similarly and offer remarkable accuracy. Order-3 and Order-4 polynomials offered the highest accuracy on new test cases, and intuitively, higher order polynomials were able to perfectly classify the training data. The linear kernel function performed significantly worse than the others tested. We can also see that the performance of kernel functions varies much more for very, very small training sets. While this is interesting to note, these 50 examples is an impractically small training set for SVMs.

Previous literature offers recommendations for identifying appropriate kernel functions for a specific problem instance. Unfortunately, there is no simple rule for kernel selection, and automated methods can be tricky. The upside, as we can see here, is that many kernel functions perform remarkably well for most problems. In particular, the radial basis function is highly recommended as a general high-performing kernel function for problems that require non-linear classifiers.

5 Conclusion

Support Vector Machines are a powerful binary linear classifier which devise a separating hyperplane between classes of training points which maximizes the distance to the nearest training point from any particular class. For the particular problem of handwritten digit recognition, SVMS outperform KNN, a simple but effective classifier to this domain, by significant margins in all practical test cases. More generally, SVMs are considered among the best and most widely used linear classifiers. Unlike KNNs, SVMs can incur a large computational cost during the training phase, but through the so-called kernel trick, SVMs can be extended to provide non-linear classification by transforming the feature space into a higher dimensionality without suffering intractable computational complexity gains.