Course Project Report

Handwritten Digit Classification

1. Introduction

Handwritten digits are a common part of everyday life. One of the first uses that comes to mind is that of zip codes. A zip code consists of some digits and is one of the most important parts of a letter for it to be delivered to the correct location. Many years ago, the postman would read the zip code manually for delivery. However, this type of work is now automated by using optical character recognition (OCR). Other immediate applications of the digit recognition techniques include postal mail sorting, automatically address reading, bank check processing, etc. As a benchmark for testing classification algorithms, the MNIST dataset has been widely used to design novel handwritten digit recognition systems. There are a great amount of studies based on MNIST dataset reported in the literature, suggesting many different methods. In my mini-project, I will walk through a basic solution in C# .NET for accurately reading the handwritten digit dataset.

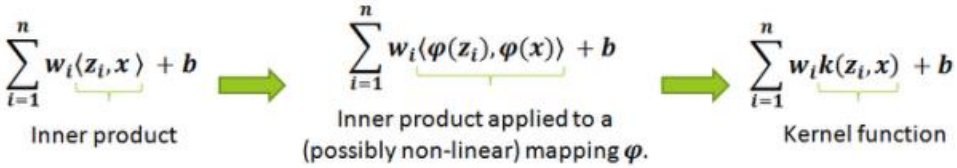
2. Data Description

The provided dataset is a subset of the MNIST digits. The dataset has 10 classes (digits 0 through 9) with 4000 images (400 images per class). Each feature vector is a vectorized image (784 dimensions), containing grayscale values [0,255]. The original image dimensions are 28×28. First of all, I will convert the dataset file to CSV file in order to use the CsvHelper library for parsing the CSV file of images. Due to the values [0-255] being too variable for the learning algorithm to process. I would adopt the method called "feature scaling" to normalize the data so that each value falls within 0 - 1. One more thing, I would like to split the training set in half. The first half will remain as the training data. The second half will serve as the cross validation data.

3. The Learning System

a. The target function

I will be using One-against-one Multi-class Kernel Support Vector Machine Classifier. One advantage of SVMs is that their solutions are sparse, they do not require the entire training set to be always available during evaluation. Only a (typically small) subset will be needed. This subset is what is commonly called "support vectors". Based on the above data description, it is perhaps a better method. However, the original SVM optimal hyperplane algorithm is a linear classifier. For addressing this problem, we can create non-linear classifier by applying the kernel trick to those maximum-margin hyperplanes. Here is the schematic diagram demonstrating how the kernel trick can be applied to the original SVM formulation:



I will be creating a solution in C# .NET, and use the Accord .NET library to implement the machine learning algorithm. Using this, Accord.MachineLearning.VectorMachines namespace, we can easily implement the algorithm. It contains both standard SVMs and the kernel extension given by KernelSupportVectorMachines. For multiple classes or categories, the framework offers MulticlassSupportVectorMachines and MultilabelSupportVectorMachines. In my solution, I use Multi-class machine. Multi-class machines can be used for cases where a single class should be picked up from a list of several class labels. And the multi-class machines also support two types of classification: the faster decision based on Decision Directed Acyclic Graphs, and the more traditional based on a Voting scheme. One big advantage of Accord .NET library is that, a huge variety of kernels functions is available in the Accord.Statistics namespace, and new kernels can be created easily using the IKernel interface. In my project, I apply a Gaussian kernel.

The Gaussian kernel is an example of radial basis function kernel, and is the preferred kernel when we don't know much about the data we are trying to model.

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

Alternatively, it could also be implemented using

$$k(x, y) = \exp\left(-\gamma\|x - y\|^2\right)$$

The adjustable parameter sigma plays a major role in the performance of the kernel, and should be carefully tuned to the problem at hand. If overestimated, the exponential will behave almost linearly and the higher-dimensional projection will start to lose its non-linear power. In the other hand, if underestimated, the function will lack regularization and the decision boundary will be highly sensitive to noise in training data.

b. Representation

Before running the SVM, we need to read the data from CSV file into memeory. Specially, we will read the data from each row into an array of doubles, with each label (0-9) being an integer. MLData type can be used to hold each row of data.

```
public class MLData
    {
        //Input
        public List<double> Data { get; set; }
        //Output
        public int Label { get; set; }
        public MLData()
        {
            Data = new List<double>();
        }
    }
```

The above class can easily be converted into a multi-dimensional array of doubles and an array of integers for all rows in the data set. We parse the CSV file to obtain a list of MLData objects:

```
List<MLData> rows = Utility.ShowProgressFor<List<MLData>>(() =>
parser.Parse(path, count), "Reading Data");
inputs = rows.Select(t => t.Data.ToArray()).ToArray();
outputs = rows.Select(t => t.Label).ToArray();
```

Parsing the actual CSV file itself can be made easier by using the CsvReader C# .NET class. One more thing, I use own IRowParser interface to do the actual processing. This gives that flexibility with reading CSV files of differing formats - label at the front for training, no label for test data. Here is the rough code:

```
public interface IRowParser
{
    //return labels (0,1,...,9); output
    int ReadLabel(CsvReader reader);

    //return data (784 columns); intput
    List<double> ReadData(CsvReader reader);
}

//column 0 contains the label
//remaining columns contain the data, i.e.,pixels
public class ExistLabelParser:BaseParser
{
    public override int ReadLabel(CsvReader reader)
    {
        return Int32.Parse(reader[0]);
    }

    public override List<double> ReadData(CsvReader reader)
    {
        //read the data from the first column
        return ReadData(reader, 1);
    }
}

 //no label
//for test
public class NonLabelParser:BaseParser
{
    public override int ReadLabel(CsvReader reader)
    {
        return 0;
    }

    public override List<double> ReadData(CsvReader reader)
    {
        return ReadData(reader, 0);
    }
 }
```

If we try running the SVM against the raw data, we are likely to get poor results upon cross validation, with regard to accuracy. This is due to the values [0-255] being too variable for the learning algorithm to process. We can solve this by normalizing the data so that each value falls within [0-1]. This is also called feature scaling. The general formula is given as:

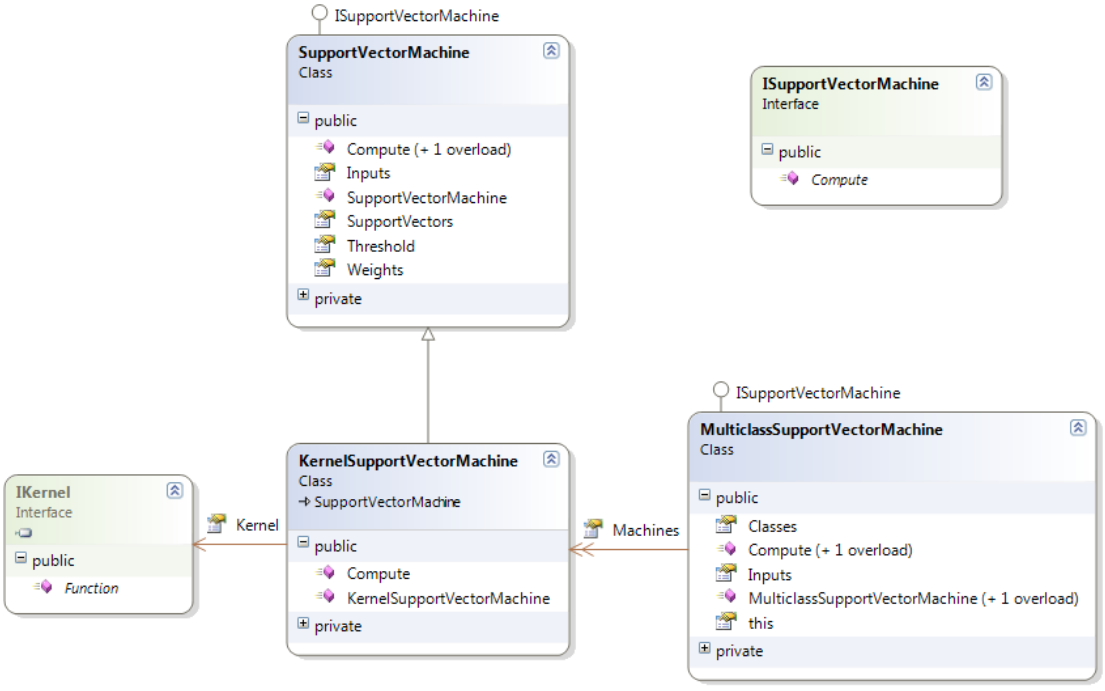
$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x is an original value, x' is the normalized value. Here is the code:

```
protected double Normalize(double value)
{
    //values fall within 0-1
    return value / 255d;
}
```

c. System structure

The classes structure for the machines included in the source code are shown below:



We can see it is very simple in terms of standard class organization. The Kernel Support Vector Machine class extends SupportVectorMachine with Kernels. MulticlassSupportVectorMachine employs a collection of KernelSupportVectorMachines working in a One-against-one voting scheme to perform multi-class classification.

d. The learning algorithm

SVM learning algorithms involved the use of quadratic programming solvers. Some of them used chunking to split the problem in smaller parts which could be solved more efficiently. Platt's Sequential Minimal Optimization (SMO) algorithm puts chunking to the extreme by breaking the problem down into 2-dimensional sub-problems that can be solved analytically, eliminating the need for a numerical optimization algorithm.

The algorithm makes use of Lagrange multipliers to compute the optimization problem. Platt's algorithm is composed of three main procedures or parts:

- *run*, which iterates over all points until convergence to a tolerance threshold;
- *examineExample*, which finds two points to jointly optimize;
- *takeStep*, which solves the 2-dimensional optimization problem analytically.

After the algorithm ends, a new Support Vector Machine can be created using only the points whose Lagrange multipliers are higher than zero. The expected outputs y_i can be individually multiplied by their corresponding Lagrange multipliers α_i to form a single weight vector w.

$$F(x) = \sum_{i=0}^{N} \{\alpha_i y\, k(z_i, x)\} + b = \sum_{i=0}^{N} \{w_i\, k(z_i, x)\} + b$$

SMO has to maintain sum over examples of example weight times example label. Therefore, when SMO adjusts weight of one example, it must also adjust weight of another.

```
//initial training
if (machine ==null)
{
     MulticlassSupportVectorLearning teacher = null;

     machine = new MulticlassSupportVectorMachine(_pixelCount, new
Gaussian(_sigma), _classCount);

//create the Multi-class learning algorithm for the machine
     teacher = new MulticlassSupportVectorLearning(machine, inputs, outputs);

//configure the learning algorithm to use SMO
//to train the underlying SVMs in each of the binary class subproblems.
     teacher.Algorithm = (svm, classInputs, classOutputs, i, j)
                    => new SequentialMinimalOptimization(svm, classInputs,
classOutputs){ CacheSize = 0 };

     Utility.ShowProgressFor(() => teacher.Run(), "Training");
}
```

4. Experimental Results

The final result, using an SVM with a Gaussian Kernel, produced an accuracy of reading the handwritten digits as follows:

trail	*1*	*2*	*Mean*
Gaussian Kernel SVM (sigma=4)	0.9500	0.9465	0.94825
Gaussian Kernel SVM (sigma=3)	0.9390	0.9380	0.93850
Gaussian Kernel SVM (sigma=2)	0.7110	0.7545	0.73275
Gaussian Kernel SVM (sigma=1)	0.1705	0.1540	0.16225
Gaussian Kernel SVM (sigma=5)	0.9490	0.9405	0.94475

According to the above diagram, we know that using different sigma values, we can get totally different accuracy. SVM sigma training value, smoothes the gaussian i.e., fuzziness factor, which means that, lower values = lower bias and higher variance and maybe overfitting, higher values = higher bias and lower variance. However, the Standard Deviations are slightly different, which means the algorithm is stable as dealing with different datasets using same sigma value.

5. Conclusion

In this mini-project, I detailed and explored how (Kernel) Support Vector Machines could be applied in the problem of handwritten digit recognition with satisfying results. The SVM solutions are sparse, meaning only a generally small subset of the training set will be needed during model evaluation. This also means the complexity during the evaluation phase will be greatly reduced since it will depend only on the number of vectors retained during training.

One of the disadvantages of Support Vector Machines, however, is the multitude of methods available to perform multi-class classification since they cannot be applied directly to such problems. Nevertheless, here I employed a one-against-one strategy in order to produce accurate results. Another problem that raises from Kernel methods is the proper choice of the Kernel function (and the tuning of its parameters). This problem is often tractable with grid search and cross-validation, which are by themselves very expensive operations, both in terms of processing power and training data available. Nonetheless, those methods can be parallelized easily, and a parallel implementation of grid search is also available in the Accord.NET Framework.

References

[1] John C. Platt, Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Microsoft Research, 1998.

[2] Wikipedia contributors, "Sequential Minimal Optimization," *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/Sequential_Minimal_Optimization.

[3] Souza, César R. "Kernel Functions for Machine Learning Applications." 17 Mar. 2010. Web. <http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html>.

[4] Wikipedia contributors, "Support Vector Machine", *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/Support_vector_machine

[5] Wikipedia contributors, "Feature scaling", *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/Feature_scaling

[6] Accord.NET Framework, http://accord-framework.net/docs/html/N_Accord_MachineLearning_VectorMachines.htm