Advanced Machine Learning Miniproject 1 - Handwritten Digit Classification

Rãzvan-George Mihalyi

Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Type: Miniproject Report
Date: October 23, 2011

Abstract

The recognition of handwritten characters in general, or (arabic) digits, in a more specific sense, is defined as the ability to interpret the legible handwritten input from sources such as paper documents. One distinguishes between offline – the recognition is done after the handwriting act is complete and the algorithm input is an image or scan of the handwritten characters or digits – and online recognition, where the recognition is done at the time of writing and is most common in pen/stylus-based screen interfaces. In this project, the focus will be on the former type of handwritten digits recognition, namely offline recognition. The main challenges attributed to this problem are linked to the variety of handwriting styles, of transformations each individual applies to the representation of a perfect digit. Once these challenges are overcome, the applications of an algorithm that can correctly classify handwritten digits span a large range of practical domains – from scanning and transforming handwritten postal codes to digitizing handwritten manuscripts or archive documents. This document introduces two supervised learning algorithms used for the classification of handwritten digits and is structured as follows: Section 1 describes the project task, Section 2 presents a high-level overview of the employed methods and of the obtained results, Section 3 provides a deeper description of the methods used, while Section 4 documents the results. Lastly, Section 5 discusses these results and provides an account of future improvements.

Contents

List of Figures

List of Algorithms

1 Introduction

The problem tackled in the first Miniproject, part of the Advanced Machine Learning Fall 2011 session at Jacobs University, is that of supervised learning of offline handwritten digits classification. Put simply, the goal is to learn a model from a provided training dataset that allows for a correct labeling of new digit samples.

Since the problem is that of offline handwritten digits classification, the representation of digits is in static images. More specifically, there exist ten classes, corresponding to the digits, i.e.,s '1', '2', ... , '9', '0'. For convenience of use within `MATLAB`, the labels used throughout this project are 1, 2, ... , 9, 10, respectively for the above digits (please note that '0' $\rightarrow$ 10). Each digit, given as a vector of 256 pixels, can be represented in form of a 16×16 pixel matrix, where each pixel has a grayscale value ranging from 0 (white) to 255 (black). The training dataset (`trainData2.txt`), a sample of which is shown in Figure 1, consists of 2000 images (with 200 images for each of the ten classes) and their corresponding labels (`trainLabels2.txt`). It is worth mentioning that images in the training data are stored in a random order.

For visualization purposes, one can use the provided function `showImage2` for displaying a single image. The script `showSomeImages` can also be used to randomly draw and display 16 images from the training dataset.

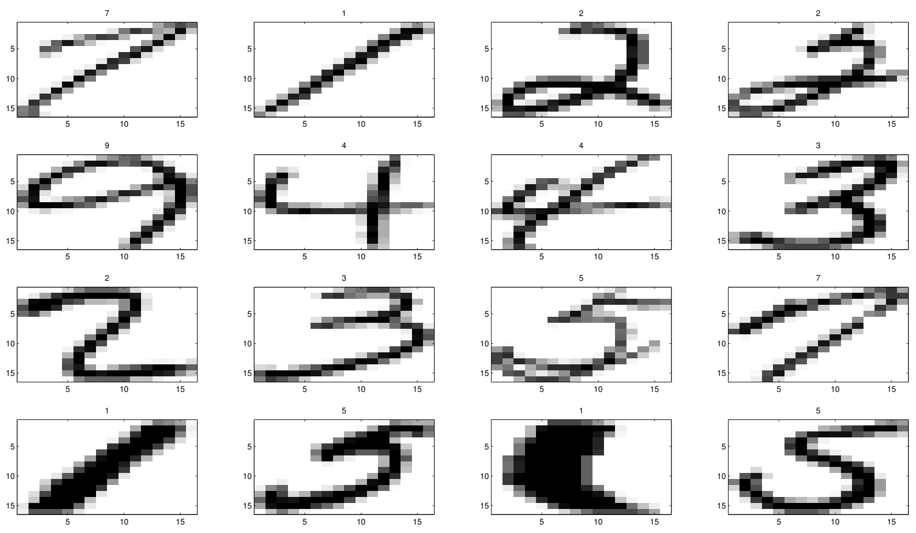

Figure 1: Randomly selected images from the training data set. Generated using the `showSomeImages` script.

2 Overview

This section provides an outline of the methods used in this miniproject and briefly presents the results. These topics will be considered in greater detail in the subsequent sections.

There are two methods for offline handwritten digit classification that are investigated in this miniproject, namely linear regression and least squares support vector machines (LSSVM [6]). The former, also provided as a sample solution in the miniproject handover, is used together with an algorithm that reduces the dimensionality of the solution space from 256 (the dimension of the pixel vector input space) to 14 (the dimension of the constructed feature vector). The latter, LSSVM is implemented as a `MATLAB` toolbox [7] and is used, for comparison purposes, both with and without the dimensionality-reduction algorithm.

The learning algorithms are run on the training data and labels – as described in the introductory section – and within a 5-fold cross-validation scheme. The results are as follows:

1. Linear Regression on a 14-dimensional feature vector (`MATLAB` folder `Mihalyi3`)
   Average training rate: **0.74575**
   Average testing rate: **0.725**
2. LSSVM on a 14-dimensional feature vector (`MATLAB` folder `Mihalyi2`)
   Average training rate: **0.96263**
   Average testing rate: **0.8865**
3. LSSVM on the 256-dimensional input (`MATLAB` folder `Mihalyi`)
   Average training rate: **0.99725**
   Average testing rate: **0.9525**

The solution with the highest average testing rate in the cross-validation scheme, i.e. LSSVM on the 256-dimensional input (`MATLAB` folder `Mihalyi`), gives a learning model, stored in `'Mihalyi/yourModel.mat'`, which should be the model used for subsequent testing.

3 Methods

3.1 Preliminaries

A preliminary step to implementing algorithms for digit classification was to establish a platform for assessing the performance of these, and of future algorithms. Following the procedure in Section 2.8 from [1], cross-validation was used as means of overcoming overfitting and of providing a metric of how well the algorithm would generalize to new data.

In order to do so, the k-fold cross-validation procedure was used. This method considers a splitting of the training set into equally sized subsets. The training is then done on $k-1$ subsets and the testing (validation) is performed on the remaining subset, e.g., $Training = \{X_1 \cup X_2 \ldots \cup X_{k-1}\}$, $Testing = X_k$. k such learning trials are carried out, where each trial uses a different validation subset (out of the initial k subsets), e.g., $Trial_1 = \{Training = \{X_1 \cup X_2 \ldots \cup X_{k-1}\}, Testing = X_k\}$, $Trial_2 = \{Training = \{X_1 \cup X_2 \ldots \cup X_{k-2} \cup X_k\}, Testing = X_{k-1}\}$, etc. One can average over the training and testing rate from these repetitions, thus obtaining an average performance of the learning model. This performance is deemed a more reliable metric for the performance of the algorithm, as it counteracts the overfitting of training data, of key importance for the ability of the algorithm to generalize well to new data, i.e., when the algorithm is presented with new, unavailable at the moment of training, data.

It is worth mentioning that throughout this project 5-fold cross-validation schemes were used. Moreover, cross-validation has also been a means of establishing a benchmark – the average performance of the sample solutions: linear regression and nearest neighbor – against which the learning methods were compared. The benchmark results are provided in the results Section 4.

3.2 Linear Regression

3.2.1 Formalism

Linear regression aims to linearly model the relationship between a scalar variable y and a vector $\mathbf{x}$. In the simplest case, linear regression is seen as the attempt to find a model that linearly fits a number of samples. This can be seen in Figure 2, where the model is a line that is fitted to a number of two-dimensional points [5].

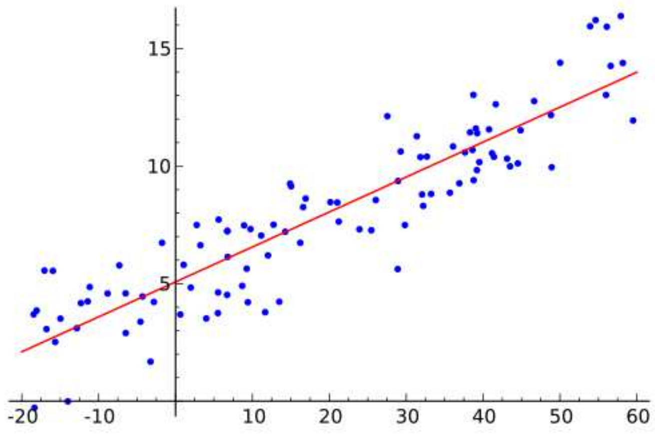

Figure 2: Example of simple univariate linear regression. Image from `http://en.wikipedia.org/wiki/Linear_regression`

.

Given a training set $(\mathbf{x}_i, y_i)_{i=1,\ldots,N}$, the equation of linear regression is:

$$\overrightarrow{y} = \overrightarrow{X} \cdot \overrightarrow{\omega} + \overrightarrow{\epsilon} \tag{1}$$

where $\overrightarrow{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$, $\overrightarrow{X} = \begin{pmatrix} \overrightarrow{\mathbf{x}_1} \\ \overrightarrow{\mathbf{x}_2} \\ \vdots \\ \overrightarrow{\mathbf{x}_N} \end{pmatrix}$ are given, the term $\overrightarrow{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix}$ is the intrinsic noise or disturbance and $\overrightarrow{\omega} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_N \end{pmatrix}$ is the vector of the linear model parameters, which need to be obtained (or estimate).

In order to do so, a popular method is that of normal equations [1], which reduces the minimization problem to:

$$\begin{aligned} \overrightarrow{X}^T \cdot \overrightarrow{y} &= \overrightarrow{X}^T \cdot \overrightarrow{X} \cdot \overrightarrow{\tilde{\omega}} \qquad (2) \\ \underbrace{(\overrightarrow{X}^T \cdot \overrightarrow{X})^{-1} \cdot \overrightarrow{X}^T}_{\text{pseudo-inverse of } \overrightarrow{X}} \cdot \overrightarrow{y} &= \overrightarrow{\tilde{\omega}} \qquad (3) \end{aligned}$$

In one of the miniproject sample solutions, the above method for linear regression was employed, with the additional construction of label indicators, which map the scalar value of label y_i to a vector representation $\tilde{y}_i$, e.g., $y_i = 10 \rightarrow \tilde{y}_i = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$.

Note: For the handwritten digit classification task, the above variables would map to the data, as follows: $\overrightarrow{y} \rightarrow$ digit labels – column vector with 2000 entries, $\overrightarrow{X} \rightarrow$ collection of training data – a 2000×256 matrix.

3.2.2 Reducing dimensionality

As mentioned in the introductory section, the training data consists of 2000 instantiations of images, each being an image vector of 256 pixels (such that it can also be represented as a 16×16 matrix). With 10 labels to distinguish from, the dimension of the training input vector space becomes 10^{256}. In order to avoid the curse of dimensionality [1], a technique to reduce the high dimensionality of the input space is required. The technique used in this case is feature extraction, where features, i.e. one-dimensional functions over the original input space are extracted, ideally without loss of information, from the high-dimensional input vectors. The learning algorithm is then performed on the feature space, instead of the input space.

Let the method used in the sample solution, i.e., linear regression using the input space high dimension, be called Method 0 and the method described in this report, i.e., linear regression using the feature space reduced dimension be called Method 1. Method 0 and Method 1 have similar performance to that on the entire space if and only if the feature extraction step does not lead to loss of information. Moreover, if overfitting occurs in Method 0, this will be detected by a higher average testing performance in Method 1.

The features used in Method 1, together with their descriptions, are listed below:

- **Correlation with the average digit representation** – starting from an idea described in [1], where a feature example was that of a maximum overlap with the representation of a perfect digit, under scaling, rotation and translation transformations. Here, under the assumption that the training data is representative of the entire "population", one can obtain the average representation of a digit (see Figure 4) in the training set and take the maximum overlap among all labels of the correlation between the image and the representations of the average digits. The maximum overlap and the corresponding label are included in the feature vector. **Note:** For correlation, the `MATLAB` `corr2` function [12] was used. This was preferred to a simple dot product between image vectors, since it was found experimentally that `corr2` yields less misclassifications than the dot product.

- **Correlation with the pixel contributions** – similar to the idea behind the previous feature, this feature first computes 10 matrices, corresponding to the 10 labels, each of which containing a weight (contribution) each pixel has towards the label. This is obtained by the following algorithm:

```
foreach label from 1 to 10 do
    foreach pixel from 1 to 256 in training set do
        pixelContribution(label) = $\sum_{imageLabel=label} pixelValue / \sum pixelValue$;
    end
end
```

Algorithm 1: Computation of Pixel Contributions

This allows for an image in the training data set to be compared to the matrices of pixel contributions. The maximum overlap (vector dot product) among the 10 labels then gives the label classification. As in the previous feature, the maximum overlap and the corresponding label are included in the feature vector.

- **Object area** – this feature uses the `MATLAB` Image Processing Toolbox function `bwarea` to obtain the area of occupied by the digit.
- **Pixel percentages** – this feature uses a function that computes the percentage of pixels lying in all the four quadrants (8×8 matrices) of the image, i.e. upper right matrix, lower left matrix, etc. All the four percentages are included in the feature vector.
- **Number of holes**, i.e. closed circles – this feature uses the `MATLAB` Image Processing Toolbox function `bwboundaries` to detect the number of holes (of size greater than 2×2 pixels) – see Figure 3. This feature is thought to be useful in correctly classifying 8 and in distinguishing between e.g. 1 and 9 based on the number of holes. **Note:** It has been observed experimentally that the function often detects false positive holes of size 2×2, hence the threshold.

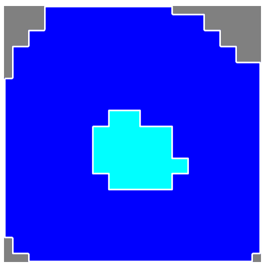

Figure 3: Hole detected by the `MATLAB` `bwboundaries` function. The outer circle is interpreted as the original digit and the inner circle as the hole.

- **Hole area** – based on the previous feature, the `polyarea` function was used to compute the area of the hole obtained as in the previous case.
- **Maximum Euclidean distance** – between pixels on the digit boundary area.

- **Maximum row-wise distance** – between pixels on the digit boundary area and on the same row. This feature is thought to be useful in distinguishing 1's from any other digit, because the maximum row-wise distance for a 1 is lower than the same distance for other digits. This column-wise distance is also computed and included in the feature vector.

It is worth mentioning that each feature was added to the feature vector if it improved the average performance of the learning model, when run in a 5-fold cross-validation scheme. In total, the feature vector has dimension 14.

Note: As mentioned in Section 2, the linear regression method on the 14-dimensional space, can be found in the deliverable folder `Mihalyi3`.

For a further discussion of feature selection, construction and extraction, please refer to Section 5.

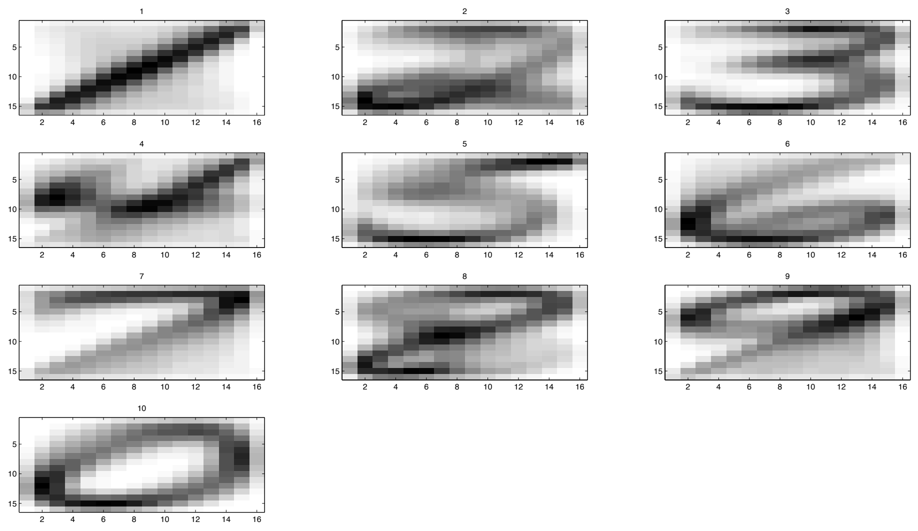

Figure 4: Representation of the average digits in the training dataset. Generated using the `showImage2` function.

3.3 LSSVM

3.3.1 Formalism

Support Vector Machines (SVMs) represent a supervised learning method to analyze data and they are mainly used in classification problems or regression analysis [2].The focus in this miniproject is going to be on classification. Thus, given a dataset and the corresponding labeling, the SVM acts as a non-probabilistic binary linear classifier and builds a linear model (or hyperplane) that best separates the two classes. Important concepts for SVMs are the separating hyperplane, the maximum-margin hyperplane (which states that the best hyperplane is the one that separates the two classes and lies at maximal distance from any samples), the soft-margin (which allows for some degree of misclassification) and the kernel function (which maps the samples to a higher dimension, with the goal of finding a separation between the data) [3].

Least Squares Support Vector Machines (LSSVMs) have been introduced in [6] with the goal of obtaining a learning model by solving a set of linear equations instead of more complex (quadratic programming) problems that need to be solved for the classic SVM problem.

The original SVM formulation states that given a training set $(\mathbf{x}_i, y_i)_{i=1,\dots,N}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and the binary labels $y_i \in \{-1, 1\}$, the SVM classifier satisfies the following conditions:

$$\begin{cases} w^T\phi(x_i) + b \geq 1, \text{if} \quad y_i = 1 \\ w^T\phi(x_i) + b \leq -1, \text{if} \quad y_i = -1 \end{cases} \tag{4}$$

This is equivalent to

$$y_i[w^T\phi(x_i) + b] \geq 1, \tag{5}$$

for $i = 1, \dots, N$, where $w^T\phi(x_i) + b$ is the equation of the separating hyperplane [4]. It can, however, be the case that the separating hyperplane does not exist. In order to obtain such a hyperplane, a violation (soft-margin) of Eq. (5) is introduced by the variable ξ_i:

$$\begin{cases} y_i\left[w^T\phi(x_i) + b\right] \geq 1 - \xi_i, & i = 1, \dots, N, \\ \xi_i \geq 0, & i = 1, \dots, N. \end{cases} \tag{6}$$

According to [6], the solution to the classifier is found by minimizing the following problem:

$$\min_{w,\xi_k} J_1(w, \xi_k) = \min_{w,\xi_k}(\frac{1}{2}w^Tw + c\sum_{k=1}^{N}\xi_k), \tag{7}$$

subject to the constraints in Eq. (6). This minimization problem is then transformed (after substituting Lagrange multipliers in the Lagrangian function) into the following quadratic programming problem:

$$\max_{\alpha_i}\ Q_1(\alpha_i, \phi(x_i)) \ = \max_{\alpha_i}(-\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i, x_j) + \sum_{i=1}^{N}\alpha_i), \tag{8}$$

where $K(x_i, x_j) = \langle\phi(x_i), \phi(x_j)\rangle$ is the kernel function and $\alpha_i \geq 0$ are Lagrange multipliers. The solution to this problem gives the hyperplane and hence the classifier that distinguishes between the two classes.

The least squares solution to the SVM classifier, as presented in [6], reformulates the classification problem:

$$\min_{w,b,e} J_3(w, b, e) = \min_{w,b,e}(\frac{1}{2}w^Tw + \frac{\gamma}{2}\sum_{i=1}^{N}e_i^2) \tag{9}$$

subject to the equality constraints: $y_i\left[w^T\phi(x_i) + b\right] = 1 - e_i, \quad i = 1, \dots, N.$
Using the Lagrangian and studying the conditions for optimality, it is shown that the linear solution to the optimization problem is given by:

$$\begin{bmatrix} 0 & -Y^T \\ Y & ZZ^T + \gamma^{-1}I_N \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{10}$$

where $Z = [\phi(x_1)^T y_1, \ldots, \phi(x_N)^T y_N]$, $Y = [y_1, \ldots, y_N]^T$, $1_N = [1, \ldots, 1]^T$, $\alpha = [\alpha_1, \ldots, \alpha_N]^T$ and I_N is an $N \times N$ identity matrix.
Hence, the classifier Eq. (4) is found by solving the linear system Eq. (10), instead of quadratic programming [6].

As introduced in [8], [7], the kernel function K can be:

- Linear kernel : $K(x, x_i) = x_i^T x$,
- Polynomial kernel of degree d: $K(x, x_i) = \left(1 + x_i^T x / c\right)^d$,
- Radial basis function RBF kernel : $K(x, x_i) = \exp\left(- \|x - x_i\|^2 / \sigma^2\right)$,

Given an efficient algorithm for binary classification, one is also interested in extending this to solve multiclass classification problems [3]. This can be done by so-called "one-versus-one" or "one-versus-all" methods, as seen below:

```
initialize wins[] = 0;
foreach $label_1$ from 1 to 10 do
    foreach $label_2$ from 1 to 10 do
        if $label_1 \neq label_2$ then
            $label$ = classify($label_1$, $label_2$);
            wins[$label$]++; // increase a count of wins for a specific label
        end
    end
end
return indexOf(max(wins)); // return the label which was classified most frequently
```

Algorithm 2: one-versus-one multiclass classification

```
foreach $label$ from 1 to 10 do
    [$score$, $label$] = classify($label$, $\neg label$) // assign a score to the classified label
    output[$label$] = $score$;
end
[$maxScore, maxLabel$] = max(output); // use a "winner-takes-all" strategy
return $maxLabel$;
```

Algorithm 3: one-versus-all multiclass classification

Multiclass classification in the "one-versus-one" case is done by a maximum votes strategy. Each binary classification consists of a vote towards the classified label. The final output label is determined by the maximum number of votes, stored in the `wins` array in Algorithm 2. In contrast, the classification in the "one-versus-all" case is done by assigning a score to each classified label, the highest score thereof determining the final label.

The two algorithms also differ in terms of complexities (measured in number of binary classifications). More specifically, Algorithm 2 performs $\binom{10}{2} = 45$ binary classifications, whereas Algorithm 3 performs 10 binary classifications. However, Algorithm 2 was preferred, despite the additional computational cost, because it has experimentally exhibited higher testing rate accuracy.

3.3.2 Implementation

Given the LSSVM `MATLAB` toolbox [7], the implementation of the handwritten digit classifier did not pose any challenges. However, an account of the the important properties of this toolbox is given below.

It is important to mention that, other than the type of kernel function, there are no other parameters that require tuning, since these parameters are found by using a cascade of two optimization algorithms, namely Coupled Simulated Annealing (CSA) and one of three optimization algorithms, i.e. a standard simplex method [9], gridsearch [10] or linsearch. These algorithms find optimal values for the regularization parameter `gam` and the kernel parameter, in this case a smoothing factor, `sig2`. From [7]: *"The complete tuning process goes as follows: First, for every kernel, first Coupled Simulated Annealing (CSA) determines suitable starting points for every method. The search limits of the CSA method are set to [exp(10), exp(10)]. Second, these starting points are then given to one of the three optimization routines above [simplex, gridsearch or linsearch]."* Following the selection of tuning parameters, the LSSVM toolbox performs an internal cross-validation scheme on the model built with these parameters so as to estimate the performance of the model.

The toolbox implementation also gives the possibility of choosing the type of kernel function, between linear, polynomial or radial basis function (RBF) kernels. In this project, the RBF kernel was used, because it exhibited higher classification testing rates in the experiments. For more details, please see Section 4.

The `MATLAB` implementation of the LSSVM train method is shown below:

```
model = initlssvm(trainData, trainLabels, 'c', [], [], 'RBF_kernel');
model = tunelssvm(model, 'simplex', 'crossvalidatelssvm', {10, 'misclass'}, 'code_OneVsOne');
model = trainlssvm(model);
```

where `'c'` stands for type – can be either function estimation (`'f'`) or classifier (`'c'`), initially `gam` and `sig2` are set to null (`gam = []` and `sig2 = []`), the kernel type is RBF, `simplex` is the optimization algorithm (because `gridsearch` and `linsearch` may pose restrictions on the kernel type [7]) used together with CSA, `crossvalidatelssvm` estimates the performance, given the tuning parameters, of the learning model with an internal 10-fold cross-validation, `10` is the number of classes, `'misclass'` is the cost measure of the residual error and is defined by the normalized number of misclassifications, i.e., $C(e) = \frac{\sum_{i=1}^{N} |y_i \neq \hat{y}_i|}{N}$, where N is the number of samples in the testing set, y_i is the ground truth label and $\hat{y}_i$ is the output of the classifier. Lastly, `'code_OneVsOne'` is the type of multiclass classification.

Finally, the choice for this particular toolbox [7] is motivated by the linear solution to the SVM classifier, which yielded a more computationally efficient learning method. Other toolboxes for SVM classification, e.g. [11], have also been investigated, but were dropped due to high computational cost.

4 Results

The results section first introduces the benchmark, i.e. the performance of the sample solutions, followed by the performance of each of the learning methods detailed above. All the results reproduced below are obtained from a 5-fold cross-validation scheme.

Benchmark:

- `sampleSolution` – Linear Regression
  Average training rate: **0.93287**
  Average testing rate: **0.8105**
- `sampleSolution2` – Nearest Neighbor
  Average training rate: **1**
  Average testing rate: **0.901**

Implemented Methods:

1. Linear Regression on a 14-dimensional feature vector (`MATLAB` folder `Mihalyi3`)
   Average training rate: **0.74575**
   Average testing rate: **0.725**
2. LSSVM on a 14-dimensional feature vector (`MATLAB` folder `Mihalyi2`)
   Average training rate: **0.962963**
   Average testing rate: **0.8865**
3. LSSVM on the 256-dimensional input (`MATLAB` folder `Mihalyi`)
   Average training rate: **0.99725**
   Average testing rate: **0.9525**

Other Results:

- As mentioned previously, the above Method 3 result comes from a "one-versus-one" multiclass classification scheme – Algorithm 2. The result with a "one-versus-all" multiclass classification scheme – Algorithm 3 is:
  Average training rate: **0.99988**
  Average testing rate: **0.9**
- Using a linear kernel instead of an RBF kernel, the result from Method 3 is:
  Average training rate: **0.98113**
  Average testing rate: **0.9255**
- Using a `'gridsearch'` [10] optimization algorithm instead of `'simplex'` [9] the result from Method 3 becomes:
  Average training rate: **0.99825**
  Average testing rate: **0.9505**

Note: For replicating the results, please use the `crossvalidate2` function or the `run.m` script, which were used to generate the above results. Also note that due to numerical optimization algorithms within the LSSVM toolbox, results may exhibit a variance of $< 1\%$.

Please note further that for comparison and discussion purposes all three learning methods are presented in this report and provided in the miniproject deliverable. However, only the highest performing method, namely Method 3 (LSSVM on the 256-dimensional input) should be considered for further testing and experimentation.

5 Discussion

This section draws conclusions from the results presented in the previous section and gives an account of future improvements that can be brought to the learning algorithms.

Firstly, by comparing the result of Method 1 (Linear Regression with the manually constructed features) with that of the benchmark, one can infer that these features do not capture enough information from the 256-dimensional space. Indeed, one cannot hope to overcome the performance of a learning algorithm that uses information from a higher dimensional space, but one can achieve very little loss of information and at least obtain comparable results. However, the results in Section 4 do not reflect this.

Secondly, given the unsatisfying result of using the same learning algorithm on reduced dimensions, one may attempt to employ a different learning algorithm. This can be seen in Method 2, where the algorithm used was LSSVM and the feature space was identical to that of Method 1. The rise in performance (by comparison of the average testing rates) is given by the more powerful ([1], [2]) SVM machinery. The performance of the algorithm in Method 2 is higher than that of the first benchmark, but not higher than that of the second benchmark.

In order to overcome both benchmark performances, Method 3 is used, where the manually constructed feature space is dropped and the entire input space (256 dimensions) is used. Comparing the result of Method 2 with that of Method 3, it can be seen that indeed the 14 handcrafted features do not capture enough information from the input space. Hence, the LSSVM algorithm is run on the entire 256-dimensional space. Given the efficient implementation of the LSSVM toolbox [7], the algorithm can learn a classification model in acceptable time – see statistics below:

- Method 1 processing time: $\approx$ 33 seconds to train and $\approx$ 33 seconds to classify
- Method 2 processing time: $\approx$ 470 seconds to train and $\approx$ 75 seconds to classify
- Method 3 processing time: $\approx$ 390 seconds to train and $\approx$ 13 seconds to classify

Note: These processing times were recorded on a Quad-Core 2 GHz i7 Processor with 8GB RAM, in the `MATLAB R2011a` release.

The subsection on other results, displays the performance of the "one-versus-all" multiclass classification scheme, which is lower than that of "one-versus-one". It is due to this higher performance that "one-versus-one" was preferred, despite the computational cost incurred from $\binom{10}{2} = 45$ binary classifications, as compared to 10 binary classifications in a "one-versus-all" scheme. Additionally, the learning performance of Method 3, altered to a linear kernel instead of the RBF kernel, is also presented. As expected [1], [7], the performance is lower for a simple linear kernel. Lastly, Method 3 was altered in terms of the optimization algorithm. Namely, gridsearch was used instead of simplex and the result shows a similar performance. However, the deviation between the result shown in the original Method 3 and this alteration falls beneath 1% and is thus deemed insignificant. In fact, in order to obtain optimization-invariant results, one can average the results from a number (here five) of different runs of the 5-fold cross-validation schemes on Method 3, thus obtaining:
Average training rate: **0.9973**
Average testing rate: **0.9522**

The above results and considerations indicate that Method 3 (LSSVM on the 256-dimensional input – `MATLAB` folder `Mihalyi`) is most likely to generalize well to new digit samples, thus yielding a better-than-average classification rate on subsequent testing sets.

As future improvements, there are a number of directions one could take. For instance:

- employing an automatic feature selection algorithm in cooperation with a cross-validation scheme, to remove features (essentially pixel dimensions), which contribute very little to the digit representation. One could use a heuristic similar to that detailed in Section 3, Algorithm 1 and iteratively remove pixels with low contributions, at the same time checking whether the overall learning performance in the cross-validation scheme drops dramatically. In order to further reduce the dimension of the feature space, one could also think of using Principal Component Analysis (PCA) [1], thus extracting the first few components which capture the highest variance in the dataset.
- introducing external (human) knowledge to feature design by implementing more accurate features. In addition to the feature that is able to detect circles (holes) in a digit, as outlined in Section 3, one could also think of a designing a feature that accurately detects the number of strokes in an image. This would help improve the current misclassification rate between '1's, '9's, '4's and '6's (most often misclassified by the linear regression learning algorithm). **Note:** This might require additional image processing knowledge, as it is important to have a robust (font- and style(italics, boldface, etc.)-invariant) detector.
- employing a less expensive computational machinery than the Support Vector Machines by implementing a Generalized Linear Discrimination network (RBF network) and optimizing the Radial Basis filters for the offline handwritten digits classification problem, i.e., possibly employ K-means clustering to find the cluster means (optimal positions of the Radial Basis filters) in the data cloud [1].

References

[1] H. Jaeger. Advanced Machine Learning Fall 2011 Lecture Notes. http://minds.jacobs-university.de/sites/default/files/uploads/teaching/lectureNotes/LN_ML_Fall11.pdf Retrieved October 23, 2011.

[2] M. Hearst et. al. "Support Vector Machines", IEEE Intelligent Systems, Trends & Controversies feature, 13 (4), July-August 1998.

[3] W. S. Noble. "What is a support vector machine?" Nature Biotechnology. 24(12):1565-1567, 2006.

[4] A. Ng. "Machine Learning (CS 229 Fall 2008)" Video Lecture on SVMs. Stanford Computer Science department. http://youtu.be/qyyJKd-zXRE. Retrieved October 23, 2011.

[5] A. Ng. Stanford "Online Machine Learning Class" Lecture Notes. http://ml-class.org. Stanford Computer Science department. Retrieved October 23, 2011.

[6] J. A. K. Suykens, J. Vandewalle. "Least squares support vector machine classifiers", Neural Processing Letters, vol. 9, no. 3, Jun. 1999, pp. 293-300., Lirias number: 218716.

[7] K. De Brabanter, P. Karsmakers, F. Ojeda, C. Alzate, J. De Brabanter, K. Pelckmans, B. De Moor, J. Vandewalle, J.A.K. Suykens. "LS-SVMlab Toolbox User's Guide version 1.8", Internal Report 10-146, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2010. http://www.esat.kuleuven.be/sista/lssvmlab/downloads/tutorialv1_8.pdf. Retrieved October 23, 2011.

[8] V. Jakkula. Tutorial on Support Vector Machines (SVM). http://www.slideshare.net/butest/svm-tutorial-3859042. Retrieved October 23, 2011.

[9] S.Reveliotis. "An Introduction to Linear Programming and the Simplex Algorithm." 1997. http://www2.isye.gatech.edu/~spyros/LP/LP.html. Retrieved October 23, 2011.

[10] sickit-learn Documentation. http://scikit-learn.sourceforge.net/modules/grid_search.html. Retrieved October 23, 2011.

[11] S. Canu, Y. Grandvalet, V. Guigue and A. Rakotomamonjy. "SVM and Kernel Methods Matlab Toolbox". Perception Systmes et Information, INSA de Rouen, Rouen, France, 2005. http://asi.insa-rouen.fr/enseignants/~arakotom/toolbox/index.html. Retrieved October 23, 2011.

[12] MATLAB R2011b Documentation. Image Processing Toolbox. http://www.mathworks.de/help/toolbox/images/ref/corr2.html. Retrieved October 23, 2011.