Published in Image Processing On Line on YYYY–MM–DD.
Submitted on YYYY–MM–DD, accepted on YYYY–MM–DD.
ISSN 2105–1232 © YYYY IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://www.ipol.im/

A MATLAB SMO Implementation to Train a SVM Classifier: Application to Multi-Style License Plate Numbers Recognition

Pablo Negri[1][2]

[1] CONICET, Godoy Cruz 2290, Buenos Aires, Argentina
[2] Instituto de Ciencias de la Computacin (ICC), Departamento de Computacin, Universidad de Buenos Aires, Buenos Aires, Argentina (pnegri@uba.dc.ar)

PREPRINT January 11, 2018

Abstract

This paper implements the Support Vector Machine (SVM) training procedure proposed by John Platt denominated Sequential Minimimal Optimization (SMO). The application of this system involves a multi-style license plate characters recognition identifying numbers from “0” to “9”. In order to be robust against license plates with different character/background colors, the characters (numbers) visual information is encoded using Histograms of Oriented Gradient (HOG). A reliability measure to validate the system outputs is also proposed. Several tests are performed to evaluate the sensibility of the algorithm to different parameters and kernel functions.

Source Code

The source code is written in MATLAB. It implements the Platt’s SMO algorithm to train and test a SVM classifier on a multi-style license plate character dataset. Compilation and usage instruction are included in the `README.txt` file of the archive. The online demo allows to test the character (numbers) recognition system, from license plate images.

Supplementary Material

The supplementary files of the work includes a dataset of license plate numbers from four countries having different fonts, and character/background colors.

Keywords: Sequential Minimal Optimization, Support Vector Machine, Multi-Style License Plate Recognition, Histogram of Oriented Gradient

1 Introduction

Sequential Minimal Optimization (SMO) [14] can be considered as the simplest algorithm to train a Support Vector Machine (SVM) classifier. It uses the *divide and conquer* approach to solve analytically a large quadratic programming (QP) optimization problem. The reduction in complexity has several advantages: saving processing time and reducing memory consumption. It is also a very interesting implementation that can be employed for pedagogical purposes, because the variables of the iterative algorithm can be easily accessed and interpreted in the learning process.

This paper uses SMO to implement a multi-class SVM classifier to address the problem of recognizing multi-style license plate numbers. The One-Against-All approach is employed to classify the incoming characters from "0" to "9". In order to be robust to changing colors and backgrounds, the features describing the characters' shape consist of Histograms of Oriented Gradients [3], as proposed by Gómez et al. [7].

2 Histogram of Oriented Gradient Feature Space

The Histogram of Oriented Gradient (HOG) [3] is a widely used image descriptor to successfully recognize different kinds of object classes, such as pedestrians, vehicles, etc. It computes the gradient of the image $I(x, y)$, using, for example a 3x3 size Sobel filter, to get $(M(x, y), O(x, y))$, the matrices corresponding to the gradient magnitude and the gradient orientation, respectively.

The gradient directions are discretised into D bins, and the histograms are determined by accumulating the gradient magnitude of each pixel of the region by their gradient directions. It is important to notice the gradient directions, not the gradient orientations, are used: Directions do not discriminate between dark-to-light and light-to-dark transitions. Thus, the HOG feature set is independent of the character and background colors. We choose $D = 6$. Each HOG feature describing a region of the image is a histogram of D bins.

Each histogram j is defined as : $h_j(x_j, y_j, s_j, r_j)$, where r_j is the type of rectangle, s_j is the scale and (x_j, y_j) is its position in the window. The types of rectangles depend on the $(width, height)$ ratio which can be (s, s), $(s, s/2)$, $(s/2, s)$. We have a total of four scales : $s : \{4, 6, 8\}$.

Algorithm 1: HOG feature j

```
input  : $(M, O, x_j, y_j, s_j, r_j)$
output : $h_j$
$R \Leftarrow defineRectanglePositions(x_j, y_j, s_j, r_j)$
Initialize $h_j$ as a zero vector of length $D$
forall the pixels positions $(x, y) \in R$ do
    $o = mod(round(D \cdot O(x, y)/\pi), D) + 1$
    $h_j(o) = h_j(o) + M(x, y)$
$T = \sum_{k=0}^{D-1} h_j(k)$
for $k \leftarrow 0$ to $D - 1$ do
    $h_j(k) = h_j(k)/T$
```

Each histogram h_j is computed using algorithm 1. The function *defineRectanglePositions()* transforms r_j position and size from a normalized character pattern with a size of 16x12 pixels to their relative values in the input image shape. Thus, it is not necessary to resize the image to compute each histogram. Once all the pixels inside region R have been evaluated and accumulated on h_j, the *bin* values are normalized to sum 1.

Fig. 1 shows two samples from an Argentinean and an American license plate HOG features, with $D = 6$, computed on regions with two different rectangles. The green rectangle corresponds to a square feature and the red one is a vertical rectangle feature. As can be seen, both HOG features are very similar for the two different license plate numbers.

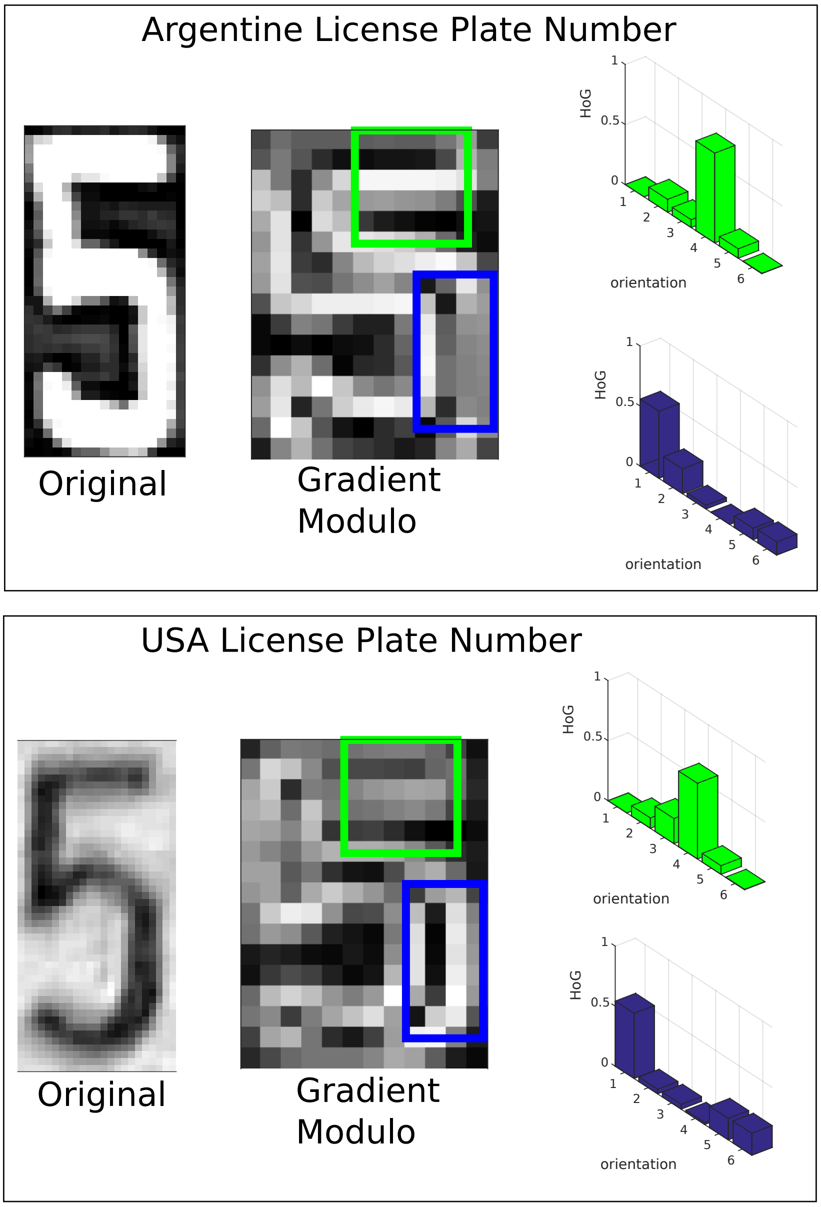


Figure 1: The figure shows the same HOG features computed for two samples from different datasets.

To accelerate the computation of the histograms, we use “Integral Histogram” [15] which is an intermediate representation of the input image inspired by Viola & Jones *Integral Image* [21]. The Integral Histogram is a three dimensional table (the third dimension corresponds to orientations) that makes it possible to accumulate the gradient magnitude for each orientation in a rectangular region with only four references to it. Thus, each HOG feature can be built with $4 \times D$ access in the Integral Histogram.

Based on the 16x12 pixels pattern size, and computing overlapped rectangles r_j with a displacement of one pixel, a set of 871 rectangles is obtained. The descriptor of one character results in a vector $\mathbf{x}$ of $G = 871$ concatenated histograms. This vector, of $G \times D$ values $\in \Re$, will be the input of the SVM multi-class classifiers.

3 SVM-SMO Implementation

This section gives a very brief introduction to SVM classification and does not intend to be a guide of SVM learning or convex optimization. Interested readers on those subjects can have a look at [1, 2, 17]. Here, the principal results for the non-linear case are presented and how the analytic solution is implemented. The idea is to deduce the equations which will be employed on the Platt's pseudo-code for the sake of completeness of the Sequential Minimal Optimization methodology. This description follows the presentation proposed in [14], section 1.1. Considering that Platts work gives a limited explanation of some equations, the procedure is completed based on [17], section 10.5.

3.1 SVM Introduction

Vapnik [20] introduces Support Vector Machines as an optimization algorithm seeking to find the hyperplane with the maximum margin discriminating two classes on a dataset, as shown on Fig. 2.

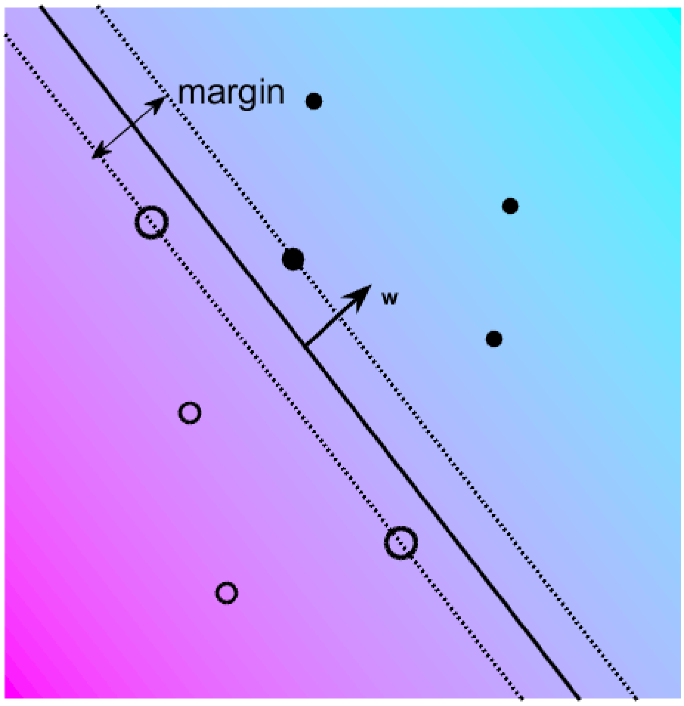


Figure 2: The figure shows a toy sample of the SVM margin and a hyperplane definition in a two-class problem. The figure was produced using the demo code from [17].

In its general form, the decision function that evaluates an input sample $\mathbf{x}_t$ is:

$$f(\mathbf{x}_t) = sign\left(\sum_{i=1}^{N} y_i \alpha_i K(\mathbf{x}_t, \mathbf{x}_i) - b\right) \tag{1}$$

where the couples $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,N}$ corresponds to the training samples. $\mathbf{x}_i \in \Re^d$ is the input vector from sample i with size d, and y_i is the associated label that takes two possible values: -1 and 1. $K(\mathbf{x}_t, \mathbf{x}_i)$ is a kernel function estimating the similarity between sample t and i on the feature space.

Examples of kernel functions will be presented at section 5.2. The constant b is the threshold of the function. The values of coefficients α correspond to the Lagrange multipliers of a quadratic programming (QP) problem. They are found by minimizing the following objective function:

$$\begin{aligned}
\underset{\alpha \in \Re^N}{\text{minimize}} \quad & \Psi(\alpha) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)\alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i \\
\text{subject to} \quad & 0 \leq \alpha_i \leq C, \forall i, \\
\text{and} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0.
\end{aligned} \tag{2}$$

where the constant C determines a compromise between margin maximization and training error minimization. A large value for C involves a high penalization on errors.

When the number of training samples is very small, the Support Vector optimization problem can be solved analytically. In that case, the solution of eq. 2 involves the inversion of a Hessian matrix of size $N \times N$ [1]. For large datasets, the manipulation of this matrix becomes intractable in memory and time requirements.

The efforts were then conducted to divide the QP problem into a series of smaller QP sub-problems. Vapnik [19] proposes a technique known as *chunking* which relies on the fact that training samples with $\alpha_i = 0$ are not involved with the solution of the QP problem. The QP sub-problems use a subset of the non-zero α_i and the M training samples that violates the most the Karush-Kuhn-Tucker conditions (KKT):

$$\begin{aligned}
\alpha_i = 0 &\Rightarrow y_i f(\mathbf{x}_i) \geq 1 \\
0 < \alpha_i < C &\Rightarrow y_i f(\mathbf{x}_i) = 1 \\
\alpha_i = C &\Rightarrow y_i f(\mathbf{x}_i) \leq 1
\end{aligned} \tag{3}$$

With the obtained solution, the valued of the α_i are updated and a new subset is selected. The algorithm finishes when the entire set of non-zero α_i is identified.

Osuna [13] also reduces the QP problem into QP sub-problems with a fixed size Hessian matrix, by removing samples and adding others which violate KKT conditions. Joachims [8] employs an heuristic to choose the samples to be used on the QP sub-problem.

3.2 Sequential Minimal Optimization: Two Variable Analytic Solution

SMO is a simple algorithm that push the *chunking* method to the smallest possible expression by using only two-Lagrange-multiplier at each iteration. It finds the optimal value for these multipliers, and updates the SVM framework, until the entire QP problem is solved. The advantage of SMO is that for two Lagrange multipliers, the optimization sub-problem can be solved analytically. This methodology is detailed below in this section.

The algorithm chooses two training samples ($\mathbf{x}_1$,y_1) and ($\mathbf{x}_2$,y_2). Their associated Lagrange multipliers are α_1 and α_2. For the sake of simplicity, the kernel function computation is expressed as $K(\mathbf{x}_1, \mathbf{x}_1) = K_{11}$, $K(\mathbf{x}_1, \mathbf{x}_2) = K(\mathbf{x}_2, \mathbf{x}_1) = K_{12}$, and $K(\mathbf{x}_2, \mathbf{x}_2) = K_{22}$. For two variables α_1 and α_2, the optimization of objective function $\Psi(\alpha)$ from equation 2 becomes [17]:

$$\begin{aligned}
\underset{\alpha_1, \alpha_2}{\text{minimize}} \quad & \Psi(\alpha) = \frac{1}{2}(\alpha_1^2 K_{11} + 2s\alpha_1\alpha_2 K_{12} + \alpha_2^2 K_{22}) - \alpha_1 - \alpha_2 \\
\text{subject to} \quad & 0 \leq \alpha_1 \leq C, \\
& 0 \leq \alpha_2 \leq C, \\
& s\alpha_2 + \alpha_1 = \gamma
\end{aligned} \tag{4}$$

where $s = y_1 \cdot y_2$, $\gamma \in \Re$.

To solve the QP problem of eq. 4, the objective function $\Psi(\alpha)$ and the constraints are simplified with the substitution: $\alpha_1 = \gamma - s\alpha_2$. Eq. 4 is then rewritten as:

$$\begin{aligned} \underset{\alpha_2}{\text{minimize}} \quad & \Psi(\alpha) = \frac{1}{2}\alpha_2^2(K_{11} + K_{22} - 2K_{12}) + \alpha_2(s\gamma K_{12} - s\gamma K_{11} + s - 1) + \frac{\gamma^2 K_{11}}{2} - \gamma \\ \text{subject to} \quad & 0 \leq \alpha_2 \leq C, \\ & \gamma - C \leq \alpha_2 \leq \gamma, \ \text{(if } s = 1) \\ & -\gamma \leq \alpha_2 \leq -\gamma + C, \ \text{(if } s = -1). \end{aligned} \tag{5}$$

where constraints on α_2 are related to constraints on α_1, from equation 4. The optimization problem will now be solved for α_2.

The quadratic objective function $\Psi(\alpha_2)$ on eq. 5 has the form:

$$\Psi(\alpha_2) = \frac{\chi}{2}\alpha_2^2 - \zeta\alpha_2 + \kappa \tag{6}$$

with

$$\zeta = s\gamma K_{11} - s\gamma K_{12} - s + 1 \tag{7}$$

$$\chi = K_{11} + K_{22} - 2K_{12} \tag{8}$$

$$\kappa = \frac{\gamma^2 K_{11}}{2} - \gamma \tag{9}$$

The minimal α_2 of 6 is the root of:

$$\frac{\partial\Psi(\alpha_2)}{\partial\alpha_2} = \chi\alpha_2 - \zeta = 0 \tag{10}$$

which is is placed at $\alpha_2 = \chi^{-1}\zeta$. This value, however, must to be clipped into the constrains in eq. 5. Both constrains can be combined on the interval $L \leq \alpha_2 \leq H$ using:

$$L = \begin{cases} max(0, s(\gamma - C)) & \text{if } s = 1 \\ max(0, s\gamma) & \text{otherwise} \end{cases} \tag{11}$$

$$H = \begin{cases} min(C, s\gamma) & \text{if } s = 1 \\ min(C, s(\gamma - C)) & \text{otherwise} \end{cases} \tag{12}$$

The value of α_2, solution of eq. 10, is computed considering two cases:
Case 1: $\chi = 0$

$$\alpha_2 = \begin{cases} H & \text{if } \zeta > 0 \\ L & \text{otherwise} \end{cases} \tag{13}$$

Case 2: $\chi > 0$

$$\alpha_2 = min(max(L, \chi^{-1}\zeta), H) \tag{14}$$

Case $\chi < 0$ occurs when two training samples have the same feature vector. To avoid this situation, a preliminary step eliminating duplicated inputs is then mandatory.

It can be proved that the new value of α_2 gets the minimal values along the direction of the constrains by using the following equation (see [17], Proposition 10.4):

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\chi} \tag{15}$$

where $E_i = f(\mathbf{x}_i) - y_i$ is the error evaluating the ith sample with eq. 1. This value of α_2^{new} has to be *clipped* to assure that the value will lie between the constraints:

$$\alpha_2^{new,clipped} = \begin{cases} H & if \quad \alpha_2^{new} \geq H \\ L & if \quad \alpha_2^{new} \leq L \\ \alpha_2^{new} & otherwise \end{cases} \tag{16}$$

Finally, the value of α_1^{new} is then obtained from $\alpha_2^{new,clipped}$ as follows:

$$\alpha_1^{new} = \alpha_1^{old} + s(\alpha_2^{old} - \alpha_2^{new,clipped}) \tag{17}$$

3.3 Implementation of the Sequential Minimal Optimization Algorithm

Platt's paper [14] proposes the pseudo-code of some routines to solve the SMO algorithm. Here, the pseudo-code is highly inspired from [14], and also follows the MATLAB implementation code that supports this article.

Algorithm 2 shows the *Main Routine* that initializes the SVM training algorithm for a two classification problem. The inputs of the procedure are the following:

- Training Dataset: composed of N pairs $(\mathbf{x}_i, y_i)$, where N is the length of the dataset, $\mathbf{x}_i$ is the feature vector of sample i, and y_i is the target of sample i corresponding to the following labels: $1, -1$, indicating to which it belongs.

- Kernel Matrix Function $\mathbf{K}$: this is a two dimensional NxN matrix. Each element (i, j) from the matrix consists of the output of the non-linear kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$.

- C parameter: the SVM training is very sensitive to this parameter. Generally, its value is computed by a k-fold cross validation approach, choosing the value which maximizes the results in a validation set.

- Initial b parameter: bias threshold parameter of the SVM hyperplane. It is initialized to 0.

- $\mathbf{E}$ error vector: this vector stores the errors of the training samples: $E_i = f(\mathbf{x}_i) - y_i$, where f_i is computed using equation 1. The initial values of the elements are: $E_i = -y_i, \forall i$. This vector will be updated every time that a Lagrange multiplier changes its value, because f_i also changes.

- Initial α vector: the values of the Lagrange multipliers (α_i), with $i = 1, ...N$, are initialized to zero.

Algo 2 is the main script that evaluates all the samples and their associated Lagrange multipliers α_i. This procedure calls the *examineExample()* routine which optimizes and updates the Lagrange multipliers. *numChanged* accumulates the number of multipliers updated at each iteration by the function *examineExample()*. The routine converges when the values of the multipliers have not changed in a whole iteration. The output of the procedure is the list of Lagrange multipliers α_i and bias b. In combination with training samples $\mathbf{x}_i$, and targets y_i, any incoming test sample can be evaluated using equation 1.

Algorithm 2: Main Routine

```
examineAll = 1
numChanged = 0
while examineAll OR numChanged > 0 do
    numChanged = 0
    if examineAll then
        forall the element i of the training set do
            numChanged = numChanged + examineExample(i)
        examineAll = 0
    else
        forall the element i of the training set where $0 < \alpha_i < C$ do
            numChanged = numChanged + examineExample(i)
        examineAll = 1
```

The *examineExample* routine (algorithm 3) iteratively chooses all the samples from the training set and searches another sample to update its Lagrange multipliers. At the end of the cycle, if the values of the multipliers have converged, the algorithm stops.

Algorithm 3: examineExample()

```
input  : i index sample
output : flagChanges
$r_i = E_i \cdot y_i$
if ($r_i <$ -tol AND $\alpha_i < C$) AND ($r_i >$ tol AND $\alpha_i > 0$) then
    if exists $j \neq i$ with $0 < \alpha_j < C$ then
        j = secondChoiceHeuristic(i)
        if takeStep(j,i) then
            return 1
        forall the remaining examples $j \neq i$ with $0 < \alpha_j < C$ do
            j = randomly picked example
            if takeStep(j,i) then
                return 1
    forall the possible indexes j on train dataset do
        if takeStep(j,i) then
            return 1
return 0
```

Each time that *examineExample()* picks one training sample i, it is necessary to choose another sample j to solve the optimization problem for two multipliers. Platt proposes a heuristic to identify sample j so as to maximize the numerator of eq. 15. This methodology, referred to as *Second Heuristic* seeks to find the pair of samples where the difference in the classification error is significant, and there is still place for improvement conditioned to the Lagrange multipliers.

Algorithm 5 is function *takeStep*, which updates the values of the Lagrange multipliers with indexes i and j. If the change between new and old values is considerable, bias threshold b and error list E are updated, using alg. 6 and alg. 7 respectively. If the change is not noticeable, the function outputs a zero value (false) indicating this situation. Section 2.3 of Platt's original work [14] details the methodology which update thresholds b.

The output of the SMO learning algorithm are the values of α_i, and bias b. In order to test a new input sample using eq. 1, these values are needed: feature vectors $\mathbf{x}_i$, and labels y_i of the training samples.

Algorithm 4: secondChoiceHeuristic()

```
input  : $E$,$i$
output : $j$
sE,sidx = sort values of error vector E
if $E_i > 0$ then
    choose the sample with lowest error to maximize the step size $|E_i - E_j|$
    if sidx[1] equals $i$ then
        $j$ = sidx[2]
    else
        $j$ = sidx[1]
else
    choose the sample with highest error to maximize the step size $|E_i - E_j|$
    if sidx[last] equals $i$ then
        $j$ = sidx[last-1]
    else
        $j$ = sidx[last]
return $j$
```

The next section implements this algorithm to train a pool of classifiers which will recognize license plate numbers.

4 Multi-class SVM Recognition Framework

The License Plate Number recognition is considered as an $M = 10$ multi-class problem. This problem is solved using the One-Against-All approach [16, 9].

4.1 SVM Multiclass Recognition Strategies

SVM basically consists on a binary classifiers, generating a hyperplane which separates to classes from a training dataset. Two of the most used strategies adapting SVM to multiclass tasks are: One-Against-One, and One-Against-All [11, 6].

Let be the training dataset composed of N samples: $\{x_1, y_1\}, ..., \{x_N, y_N\}$. Each $x_i \in \Re^d$ is the input vector of concatenated HOG features, and $y_i \in \{1, ..., M\}$ is the corresponding label associated with one of the M classes. The One-Against-All approach trains separately M binary SVM classifers, one class vs all the other classes: the machine corresponding to class i is trained setting label $y = 1$ to samples of the $i - th$ class, while the other samples get label $y = -1$.

Equation 1 then becomes:

$$f_i(\mathbf{x}_t) = \sum_{j=1}^{N} \mathbf{1}_{y_j=i} \alpha_{i,j} K(\mathbf{x}_t, \mathbf{x}_j) - b_i \tag{18}$$

where notation $\mathbf{1}_{y_j=i}$ is the label function of y_j, which takes value 1 when the sample has the same label as the training class, $y_j = i$, and -1; otherwise, variables $\{\alpha_{i,j}, b_i\}$ correspond to the output of the training algorithm taking the ith class as positive, as explained in section 3.3.

In the testing phase, a sample $\mathbf{x}_t$ is classified as in class i^* whose f_i^* produces the largest value of the SVM output function of equation 18:

$$i^* = argmax_{i=1,...,M} f_i(\mathbf{x}_t) \tag{19}$$

Algorithm 5: takeStep()

```
input   : $j$,$i$
output : $flagChanged$
$s = y_i y_j$
if $y_j$ equals $y_i$ (s=1) then
    L=max(0,$\alpha_i + \alpha_j - C$)
    H=min($C$,$\alpha_i + \alpha_j$)
else
    L=max(0,$\alpha_i - \alpha_j$)
    H=min($C$,$C + \alpha_i - \alpha_j$)
if the boundaries overlap (L=H) then
    return 0
$\chi = \mathbf{K}(j,j) + \mathbf{K}(i,i) - 2\mathbf{K}(j,i)$
if $\chi > 0$ then
    $a_2 = \alpha_i + y_i \cdot \frac{(E_j - E_i)}{\chi}$
    if $a_2 < L$ then
        $a_2$ = L
    else if $a_2 > H$ then
        $a_2$ = H
else
    if $(y_i \cdot (E_j - E_i)) < 0$ then
        $a_2$ = H
    else if $(y_i \cdot (E_j - E_i)) > 0$ then
        $a_2$ = L
    else
        $a_2 = \alpha_i$
if $|a_2 - \alpha_i| < eps \cdot (a_2 + \alpha_i + eps)$ then
    return 0
$a_1 = \alpha_1 + s \cdot (\alpha_i - a_2)$
updateThreshold($j$,$i$,$a_1$,$a_2$)
$\alpha_j = a_1$
$\alpha_i = a_2$
updateErrorList()
return 1
```

Algorithm 6: updateThreshold()

```
input   : $j$,$i$,$a_1$,$a_2$
output : $b$
$b_1 = E_j + y_j \cdot (a_1 - \alpha_j) \cdot \mathbf{K}(j,j) + y_i \cdot (a_2 - \alpha_i) \cdot \mathbf{K}(j,i) + b$
$b_2 = E_i + y_j \cdot (a_1 - \alpha_j) \cdot \mathbf{K}(j,i) + y_i \cdot (a_2 - \alpha_i) \cdot \mathbf{K}(i,i) + b$
$b = 0.5(b_1 + b_2)$
```

Algorithm 7: updateErrorList()

```
input   :
output : $E$
forall the element $i$ of the training set do
    $u_i = \sum_{j=1}^{N} y_j \alpha_j \mathbf{K}(i,j) - b$
    $E_i = u_i - y_i$
```

The One-Against-One strategy involves the contruction of a machine for each pair of classes, resulting in $M(M-1)/2$ binary classifiers. A classifier discriminating between classes i and j can be represented by the function f_{i-j}. When the methodology is applied to a test sample $\mathbf{x}_t$, it is classified as the class which gets most of the votes from the $M(M-1)/2$ classifiers.

In this paper, the One-Against-All approach is taken because it is better suited to construct the reliability measure, which is developed in the next section.

4.2 Reliability Measures for Multi-Class SVM

The outputs of the multi-class recognition framework are analyzed in order to incorporate a reliability measure to evaluate classification results.

The methodology, proposed by Thome et al. [18] defines two confidence variables: c_d and c_r. c_r is a measure of performance, i.e. how well the character is identified. The other variable c_d is a measure of discrimination performance, i.e. how discriminant a classifier output is with respect to its $M-1$ competitors.

The IPOL implementation code follows the guidelines in the paper by Gomez et al. [7]. The main difference with [18] is that the individual output values of the SVM classification functions are transformed to enhance the discriminating response of the One-Against-All strategy.

Let $\mu_{\mathbf{x}_t}$ be the mean value of the M $f_i(\mathbf{x}_t)$ SVM classification outputs, and $\sigma_{\mathbf{x}_t}$ their standard deviation. A vector $\mathbf{v}$ with M elements is defined, where each ith element is obtained as follows:

$$\mathbf{v}(i) = \frac{(f_i(\mathbf{x}_t) - \mu_{\mathbf{x}_t})^2}{\sigma^2_{\mathbf{x}_t}}$$

Now, if the i^* index defines the largest value of the SVM classification outputs (see eq. 19), c_d and c_r confidence scores for sample $\mathbf{x}_t$ are defined as follows:

$$\begin{aligned} c_r(\mathbf{x}_t) &= \mathbf{v}(i^*) \\ c_d(\mathbf{x}_t) &= \frac{\mathbf{v}(i^*)}{\sum_{i=1,i\neq i^*}^{M} \mathbf{v}(i)} \end{aligned}$$

Two constant values, T_{CR} and T_{CD}, were estimated using the confidence scores computed for training samples. They get the lowest values of the scores corresponding to correct classifications:

$$\begin{aligned} T_{CR} &= \min_{t'=1,\ldots,N} \{c_r(\mathbf{x}_{t'}) | \operatorname*{argmax}_{i=1,\ldots,M}(f_i(\mathbf{x}_{t'})) = y_{t'}\} \\ T_{CD} &= \min_{t'=1,\ldots,N} \{c_d(\mathbf{x}_{t'}) | \operatorname*{argmax}_{i=1,\ldots,M}(f_i(\mathbf{x}_{t'})) = y_{t'}\} \end{aligned}$$

The output of the Multi-class Recognition Framework receives a reliability measure for sample $\mathbf{x}_t$ is computed as:

$$r(\mathbf{x}_t) = \frac{c_r(\mathbf{x}_t)}{T_{CR}} \cdot \frac{c_d(\mathbf{x}_t)}{T_{CD}} \tag{20}$$

Fig. 3 compares two character recognition results. One example is a valid "3" character number, which obtains a reliability measure $r = 3.28$, based on constants $T_{CR} = 2.23$ and $T_{CD} = 6.21$ fixed during the learning phase. As can be seen, the highest SVM output corresponds to binary classifier $f_{i='3'}(\mathbf{x}_t)$. The other SVM outputs have relatively stable values around -1. The example of character "E" obtains $r = 0.47$. The distribution of their SVM binary classifier outputs results in a variance that is 30% greater than the “3” character. This shows that the classifiers did not obtain a set of values where the correct character class is strongly discriminated. Both scores, c_r and c_d obtain values lower than the constants.

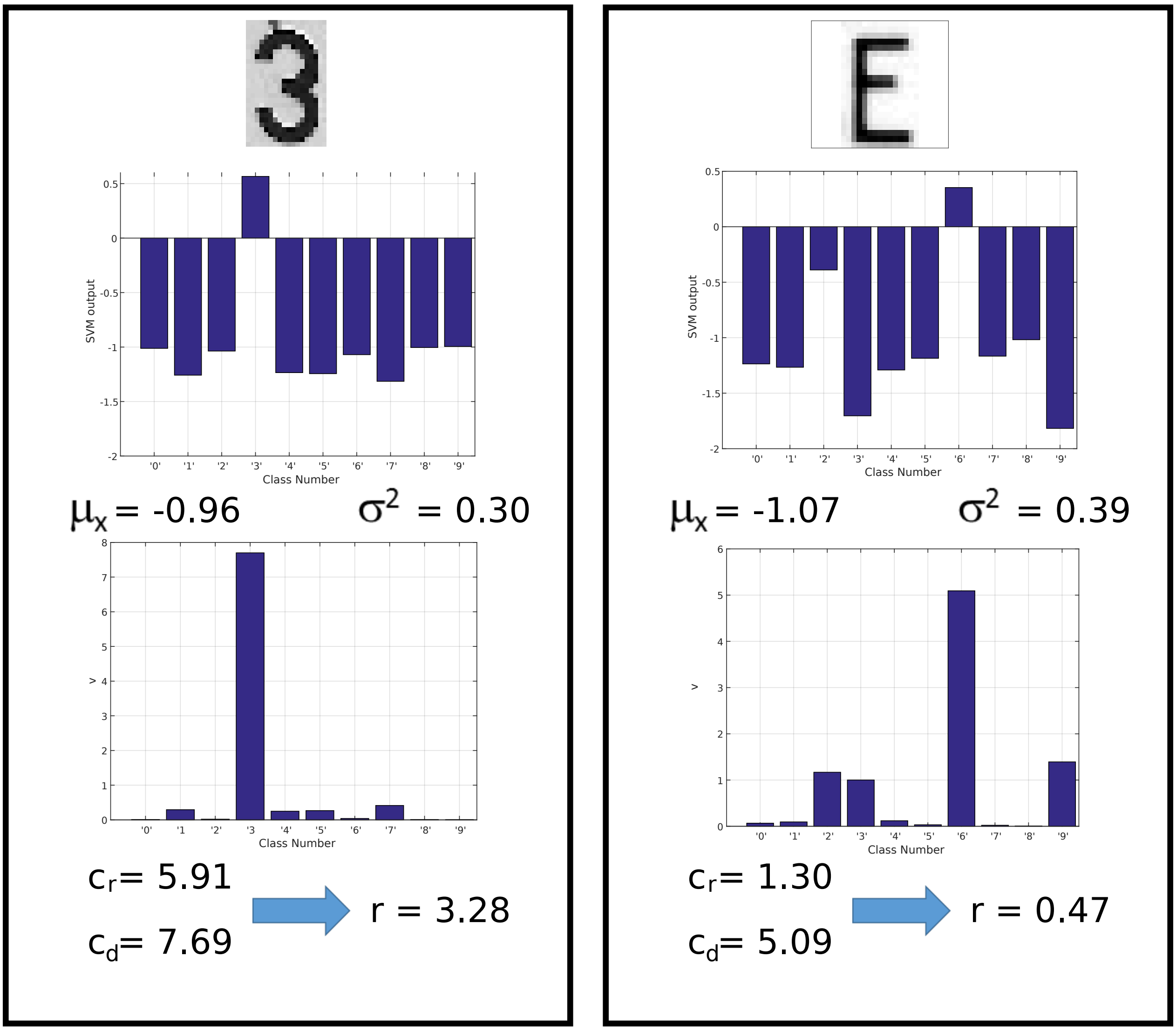


Figure 3: The figure shows two examples of the computation of the reliability measure.

Reliability measure $r < 1.0$ could then correspond to a number with a style that is not present in the learning dataset, a wrongly defined bounding box, or an image that is not matched to a number, which is the present case. If $r > 1.0$ the output corresponds to a character with a high discrimination ratio, and can be trusted as a real number.

5 Experiments and Results

5.1 License Plate Number Datasets

The multi-style license plate dataset is composed of character numbers extracted from the Dlavnekov dataset (USA) [5, 4], *COMVIS_cardataset_v1* (Pakistan) [12], Medialab’s dataset (Greece/Europe) [10], and an Argentinean license plate dataset. As can be seen on Fig 4, the license plate numbers of those datasets have opposite colors for the characters and the background.

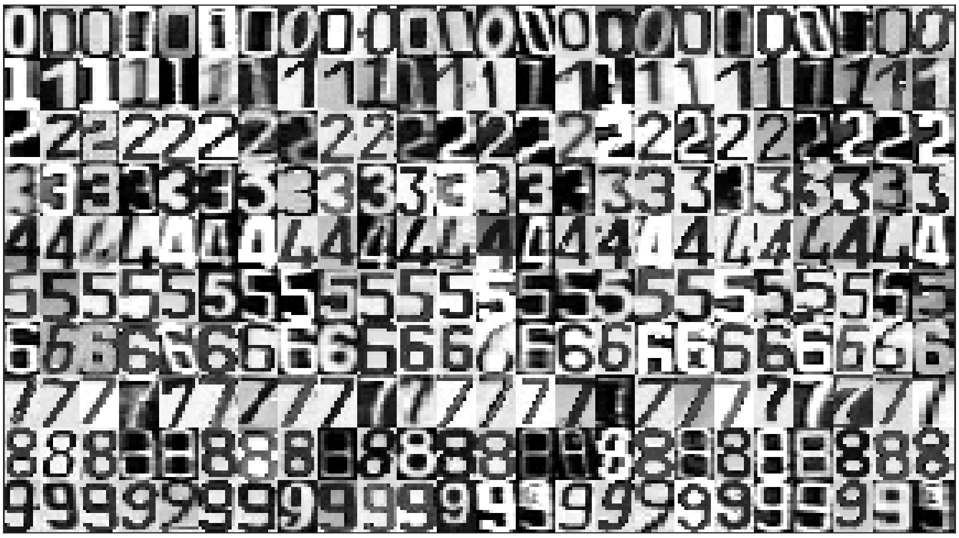


Figure 4: The figure shows some training and testing samples from the Multi-Style License Plate Numbers dataset, used in the recognition system.

Table 1 details the number of samples for each number class, discriminating between the American, Argentinean, Greek, and Pakistani datasets.

Number	Argentine	USA	Greece	Pakistan	Total
’0’	33	14	49	19	115
’1’	33	22	29	15	99
’2’	42	21	85	30	178
’3’	32	36	74	37	179
’4’	36	61	69	37	203
’5’	32	47	95	21	195
’6’	26	25	75	20	146
’7’	26	25	74	17	142
’8’	30	22	62	35	149
’9’	28	23	74	14	139
Total	335	310	686	245	1545

Table 1: This table shows the composition of the License Plate Number Dataset.

5.2 Evaluation of Parameters

This section evaluates the performance of the multiclass SVM classifiers, applying three different kernel functions, increasing the size of the training dataset, and for different values of constant C on the SMO algorithm.

The overall performance of each multi-class classifier is obtained by constructing a confusion matrix of the ten classes. The Equal Error Rate (EER) is computed as the average of the percentage of samples correctly classified, which are placed at the diagonal of the confusion matrix.

The training is carried out by randomly picking 20 or 40 samples per class. The total number of training samples is then $N = 200$ or $N = 400$. The test dataset is built using 50 remaining samples of each class, and resulting on a total set of 500 samples. This operation is repeated three times using different sets of samples. The performance of each choice is then computed as the mean value of the three EERs.

Linear, Polynomial and Radial Basis Function (RBF) kernels are used below to implement the SVM multi-classification.

Linear Kernel Function

The Linear Kernel function is the simplest similarity measure and is defined by the following equation:

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i' \cdot \mathbf{x}_j \tag{21}$$

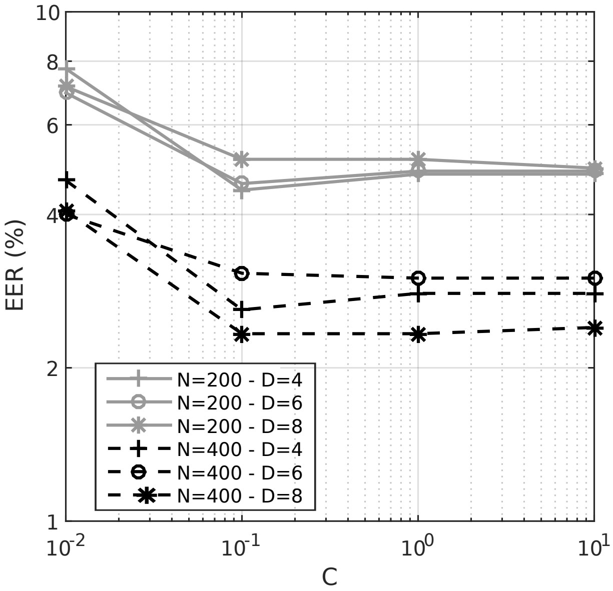


Figure 5: The figure shows the EER of the multi-class license plate number performance using a linear kernel function.

Fig. 5 shows the performance of the framework using a linear kernel function and changing the number of HOG directions D, the C parameter, and the number of training samples (N). The lowest EER value is equal to 2.3 %. It is obtained using the largest training dataset with $N = 400$, and the more detailed description of the shape with $D = 8$.

Polynomial Kernel Function

The Polynomial Kernel function is calculated as:

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = (c + \mathbf{x}_i' \cdot \mathbf{x}_j)^d \tag{22}$$

where d is the degree of the polynomial, and c a constant bias which was fixed to 1 for the tests.

Several tests were conducted using this kernel function and changing parameter d in the polynomial computation, HOG directions values D, training dataset size N, and SVM constant C. The reliability of the results strongly depend on the degree d of the polynomial function, as can be seen on table 2. With $d = 2$ (quadratic kernel), the classifiers generalize much better than greater values, obtaining reliable outputs from the SVM classifier.

Fig. 6 shows the performance of the classifiers fixing $d = 2$ for different values of C, N and D. The EER values decrease to $C = 10^{-1}$ and stabilize for greater values. For the smaller dataset

polynomial degree	Reliable outputs (%)
$d = 2$	97.6
$d = 3$	80.2
$d = 4$	74.7

Table 2: The table shows the percentage of test samples with valid reliability measure $r > 1$.

($N = 200$), the best accuracy is obtained with $D = 4$ HOG directions, with $EER = 3.8\%$. On the other hand, the lowest EER is 2.2% for the larger dataset ($N = 400$) and $D = 8$ HOG directions.

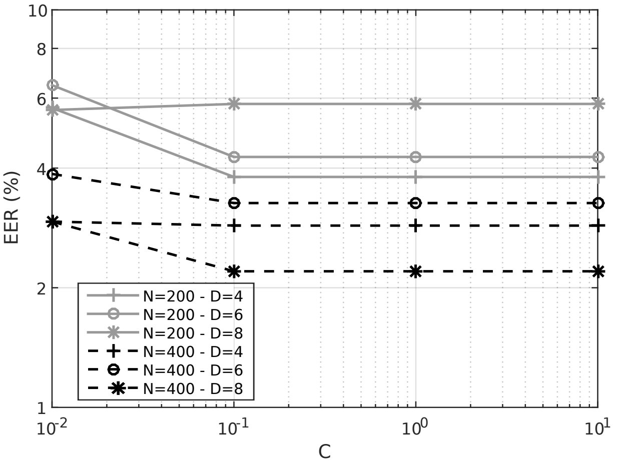


Figure 6: The figure presents the EER of the multi-class license plate number performance using polynomial kernel function, with $d = 2$.

Radial Basis Function Kernel

The Radial Basis Function (RBF) is widely used in numerous applications with excellent results. It is considered that this kernel function projects the input feature vector to classification space with infinite dimensions. The RBF is defined as:

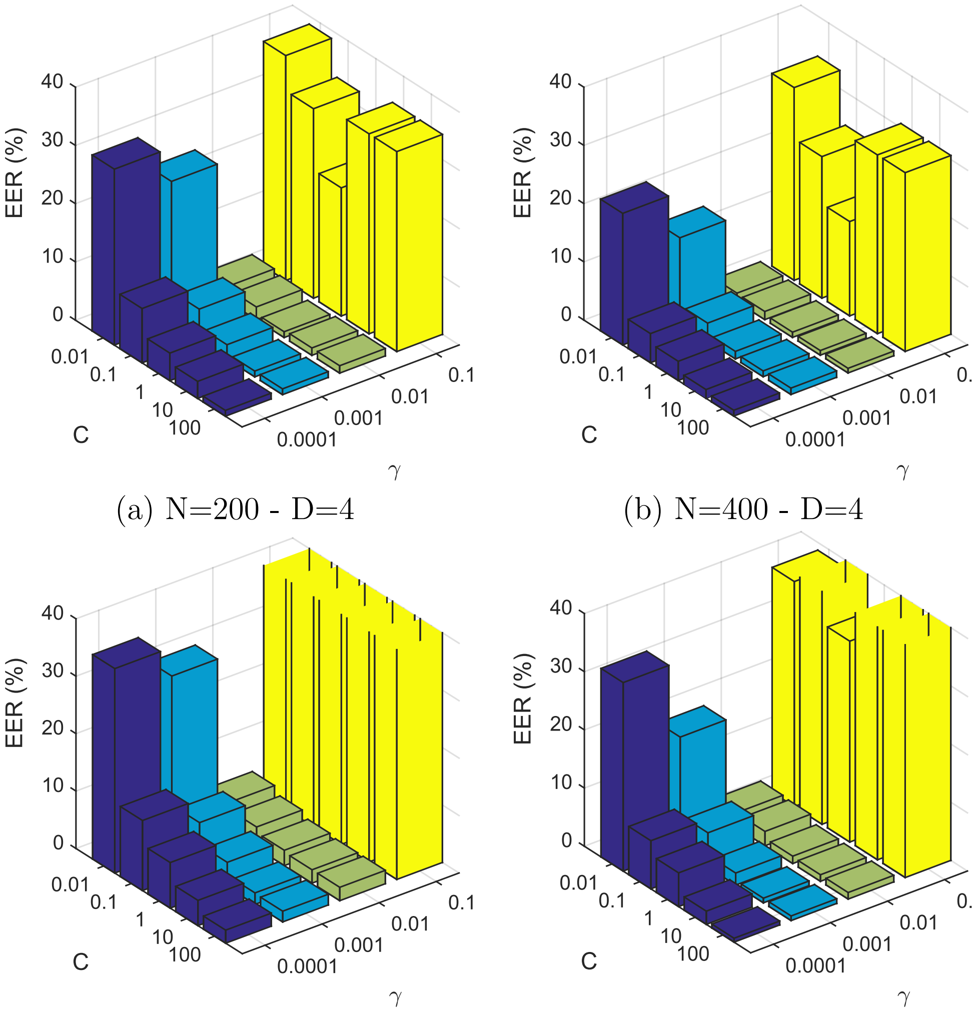
$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2} \tag{23}$$

where γ is a positive real number. In general, SVM classifiers using a RBF kernel produce reliable outputs using r measure from equation 20, as can be seen on table 3. The greatest reliability values are obtained for $\gamma = 0.01$.

γ N	0.0001	0.001	0.01	0.1
200	94.5	95.4	99.9	69.4
400	98.1	98.6	100.0	82.2

Table 3: The table shows the percentage of reliable SVM outputs for an RBF kernel and different parameter values using a reliability measure.

Fig. 7 compares the EER performance for different values on parameters C, N and D. The lowest $EER = 0.8\%$ is obtained for the larger dataset ($N = 400$), for values ($\gamma = 0.01$,$D = 4$,$C = 10$), and set ($\gamma = 0.001$,$D = 8$,$C = 10$). Using fewer training samples ($N = 200$), the performance is similar, obtaining a value of $EER = 1.0\%$ for ($\gamma = 0.001$,$D = 4$,$C = 10$) and ($\gamma = 0.01$,$D = 4$,$C = 1$).



(a) N=200 - D=4

(b) N=400 - D=4

(c) N=200 - D=8

(d) N=400 - D=8

Figure 7: The figure compares the sensibility of the RBF kernel to parameter C for different datasets and input features.

5.3 Operational Complexity and Time Considerations

Both the operational complexity and processing time depend on the different parameters. Table 4 presents the average processing time, in seconds, to evaluate an input test number in the multi-class recognition number system defined which is composed of different kernel functions and N and D parameters. The code is entirely written in MATLAB and corresponds to the same that supports this article on IPOL page. The scripts were executed on an Intel(R) Core(TM) i5 CPU 2.67 GHz, with 4 Gbytes of RAM.

The computational complexity of the linear kernel can be estimated by the number of directions D, the number of training samples N, the number of HOG features $G = 871$, and the number of classes 10. One linear kernel function call performs $D \times G$ sums and $D \times G$ products. The SVM output, as shown in eq. 1, adds $2 \times N$ products, corresponding to labels y_i and alphas α_i, and N sums. Thus, it takes $N \times D \times G + N$ sums, and $N \times D \times G + 2 \times N$ multiplications to obtain

Linear Kernel

D N	4	6	8
200	0.090	0.099	0.116
400	0.173	0.200	0.226

Optimized Linear Kernel

D N	4	6	8
200	$3.3\dot{1}0^{-4}$	$4.3\dot{1}0^{-4}$	$5.5\dot{1}0^{-4}$
400	$3.7\dot{1}0^{-4}$	$4.5\dot{1}0^{-4}$	$6.2\dot{1}0^{-4}$

Polynomial Kernel ($d = 2$)

D N	4	6	8
200	0.109	0.122	0.136
400	0.221	0.246	0.269

RBF Kernel

D N	4	6	8
200	0.130	0.152	0.170
400	0.257	0.308	0.331

Table 4: The table shows the average processing time in seconds that it takes to classify a testing sample in the multi-class recognition framework.

an individual output. To classify one testing sample, the framework will perform this operation 10 times. The complexity has an order of $10 \times N \times D \times G$ sums and $10 \times N \times D \times G$ multiplications.

There are some options to reduce the computational complexity. Firstly, eq. 1 can be solved by only using support vectors: those training samples with $\alpha_i \neq 0$. Depending on the problem, this can consist of a 10 % reduction in the number of operations. Secondly, Platt proposes an optimization for linear kernels. Using eq. 21, evaluation function 1 becomes:

$$f(\mathbf{x}_t) = sign\left(\sum_{i=1}^{N} y_i \alpha_i \mathbf{x}_t \cdot \mathbf{x}_i - b\right) \tag{24}$$

Eq. 24 is linear and can be solved using a single weight vector $\mathbf{w}$:

$$f(\mathbf{x}_t) = sign\left(\mathbf{w} \cdot \mathbf{x}_t - b\right) \tag{25}$$

This drastically reduces the number of operations to $10 \times D \times G$ sums and $10 \times D \times G$ products.

The processing time to evaluate one example using an RBF kernel is by far the longest, considering the linear kernel optimization. The use of this kernel function represents a compromise between good performance and fast response. If the application does not need real time outputs, the RBF kernel represents a very good choice.

5.4 Results

This section presents the results obtained by running the Multi-Style License Number Recognition Framework IPOL code supporting this article.

Feature	Classification Method	Accuracy (%)
Raw data	Knn (K=5)	79.1
HOG	Knn (K=5)	96.8
Raw data	SVM	91.7
HOG	SVM	**99.0**

Table 5: Comparing classification results using HOG features and intensity raw values, and SVM against KNN classifiers.

Table 5 compares HOG features results against intensity raw values using two classification methods. Because the images of the character numbers have different sizes, they were resized to 16×12 pixels, to get an intensity feature vector with the same length. SVM is compared with a simple K-Nearest Neighbors (KNN) classification using $K = 5$. For HOG features, histogram distances are calculated using vector correlation, and for intensity features, the euclidean distance of the normalized pixels values. The overall performance of each system is evaluated by computing the classification rate of each number, then the mean of the diagonal of the confusion matrix. Table 6 shows the results on the corresponding confusion matrix using SVM classifier and HOG features, which yields the highest accuracy ratio.

The default kernel function for the SVM classifier is the RBF, with $\gamma = 0.01$, constant $C = 1$, and the number of HOG directions is $D = 4$. The training dataset is composed of $N = 200$ randomly chosen samples, consisting of 20 elements per class number. The remaining samples will make up the test set.

Results are very good, even if the number of training samples is very limited. This can be explained because LPR numbers should have standard shapes following the country regulations. It simplifies the generalization of the classes shape using HOG features, compared with intensity values. Finally, it is not then necessary to have a large training set, as the shapes of the same numbers have very similar features.

	’0’	’1’	’2’	’3’	’4’	’5’	’6’	’7’	’8’	’9’
’0’	**100.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
’1’	0.0	**97.3**	0.0	0.0	2.7	0.0	0.0	0.0	0.0	0.0
’2’	0.0	0.0	**99.4**	0.0	0.0	0.0	0.0	0.6	0.0	0.0
’3’	0.0	0.0	0.0	**98.7**	0.0	0.7	0.0	0.7	0.0	0.0
’4’	1.1	0.0	0.6	0.0	**98.3**	0.0	0.6	0.0	0.0	0.0
’5’	0.0	0.0	0.0	0.0	0.0	**100.0**	0.0	0.0	0.0	0.0
’6’	0.0	0.0	0.0	0.0	0.0	0.9	**99.1**	0.0	0.0	0.0
’7’	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	0.0	0.0
’8’	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	**99.2**	0.0
’9’	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.9	**98.1**

Table 6: Confusion matrix of License Plate Number recognition in the test samples from the four datasets.

The reliability measure, using eq. 20 for the SVM classifier using HOG features, is computed in each test sample. The percentage of samples with $r > 1.0$ is 93 %.

Image Credits

The multi-style license plate dataset *COMVIS_cardataset_v1* and the Medialab’s dataset are provided without license. The Argentinean dataset is also distributed without license. Dlavnekov’s dataset can be freely employed for research purposes.

References

[1] J.C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, 2 (1998), pp. 121–167. http://dx.doi.org/10.1023/A:1009715923555.

[2] C. Cortes and V. Vapnik, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297. http://dx.doi.org/10.1023/A:1022627411411.

[3] N. Dalal and B.Triggs, *Histograms of oriented gradients for human detection*, in IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, June 2005, pp. 886–893. http://dx.doi.org/10.1109/CVPR.2005.177.

[4] L. Dlagnekov and S. Belongie, *Recognizing cars*, Tech. Report CS2005-083, UCSD CSE, 2005.

[5] L. Dlagnekov and S. Belongie, *Ucsd/calit2 car license plate, make and model database*, 2005. http://vision.ucsd.edu/belongie-grp/research/carRec/car_rec.html.

[6] A. Gidudu, G. Hulley, and T. Marwala, *Image Classification Using SVMs: One-against-One Vs One-against-All*, in Asian Conference on Remote Sensing, 2007. arXiv:0711.2914.

[7] F. Gómez Fernández, P. Negri, M. Mejail, and J. Jacobo, *A multi-style license plate recognition system based on tree of shapes for character segmentation*, in Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, vol. 7042 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 443–450. http://dx.doi.org/10.1007/978-3-642-25085-9_52.

[8] T. Joachims, *Making large-scale svm learning practical*, in Advances in Kernel Methods ? Support Vector Learning, MIT Press, 1998, pp. 169–184. ISBN 0-262-19416-3.

[9] Y. Liu and Y.F. Zheng, *One-against-all multi-class svm classification using reliability measures*, in International Joint Conference on Neural Networks, vol. 2, 2005, pp. 849–854.

[10] National Technical University of Athenas Medialab. http://www.medialab.ntua.gr/research/LPRdatabase.html, last accesed on 2012.

[11] J. Milgram, M. Cheriet, and R. Sabourin, *?One Against One? or ?One Against All?: Which One is Better for Handwriting Recognition with SVMs?*, in International Workshop on Frontiers in Handwriting Recognition, La Baule (France), October 2006. inria-00103955.

[12] COMSATS Institute of Information Technology. http://comvis.ciitlahore.edu.pk/downloads/comvis_cardataset_v1.0.html.

[13] E. Osuna, R. Freund, and F. Girosi, *An improved training algorithm for support vector machines*, in IEEE Workshop Neural Networks for Signal Processing, Sep 1997, pp. 276–285. http://dx.doi.org/10.1109/NNSP.1997.622408.

[14] J. Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*, Tech. Report MSR-TR-98-14, Microsoft Research, 1998.

[15] F. Porikli, *Integral histogram: A fast way to extract histograms in cartesian spaces*, in IEEE Conference on Computer Vision and Pattern Recognition, 2005, pp. 829–836. http://dx.doi.org/10.1109/CVPR.2005.188.

[16] R. Rifkin and A. Klautau, *In defense of one-vs-all classification*, Journal of Machine Learning Research, 5 (2004), pp. 101–141.

[17] B. Schölkopf and A. Smola, *Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, 2002. ISBN 0262194759.

[18] N. Thome, A. Vacavant, L. Robinault, and S. Miguet, *A cognitive and video-based approach for multinational license plate recognition*, Machine Vision and Applications, 22 (2010), pp. 389–407. http://dx.doi.org/10.1007/s00138-010-0246-3.

[19] Vladimir Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer-Verlag New York, Inc., 1982. ISBN 0387907335.

[20] ——, *The nature of Statistical Learning Theory*, Springer, 1995. ISBN 9780387987804.

[21] P. Viola and M. Jones, *Robust real-time face detection*, International Journal of Computer Vision, 57 (2004), pp. 137–154. http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb.