ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Študijný odbor:
Informačné systémy

Bc. Daniel Václavik

Využitie SVM na klasifikáciu znakov získaných z evidenčných čísel vozidiel
Vedúci práce: Ing. Peter Tarábek, PhD.

Registračné číslo: 300/2013 Apríl 2014

ŽILINA, 2014

ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY DIPLOMOVEJ PRÁCE.

Študijný program : Informačné systémy

Zameranie: Aplikovaná informatika

Meno a priezvisko
Daniel Václavik

Osobné číslo
554274

Názov práce v slovenskom aj anglickom jazyku

Využitie SVM na klasifikáciu znakov získaných z evidenčných čísel vozidiel

Licence plate character classification using the SVM

Zadanie úlohy, ciele, pokyny pre vypracovanie (Ak je málo miesta, použite opačnú stranu)

Cieľ diplomovej práce:
Cieľom práce je preskúmať možnosti využitia SVM (support vector machines) pre klasifikáciu znakov získaných z evidenčných čísel vozidiel. Okrem presnosti má byť dôraz kladený aj na rýchlosť metódy tak, aby klasifikácia všetkých znakov EČV prebiehala v reálnom čase. Úlohou je tiež preskúmať vplyv vybraných metód predspracovania na kvalitu riešenia. Práca bude mať k dispozícii databázu vysegmentovaných a normalizovaných znakov.

Obsah:

1. Analýza súčasného stavu a popis SVM.
2. Výber vhodnej knižnice, ktorá implementuje SVM. Výber je potrebné zdôvodniť na základe vhodne zvolených kritérií.
3. Skúmanie vplyvu vybraných metód predspracovania na rýchlosť a kvalitu klasifikácie.
4. Popis a porovnanie rôznych typov jadier (kernels) pre SVM.
5. Experimentálne zhodnotenie a záver.

Témy z predmetov študijného zamerania 5SI030: 1, 2, 5
Meno a pracovisko vedúceho DP: Ing. Peter Tarábek, PhD., KDS, ŽU
Meno a pracovisko tútora DP:

vedúci DP	tútor	vedúci katedry	garant
29.10.2013		29.10.2013	30.10.2013
(dátum a podpis)	(dátum a podpis)	(dátum a podpis)	(dátum a podpis)

Zadanie zaregistrované dňa 29. 10. 2013 pod číslom 300/2013 podpis

DECLARATION OF ORIGINALITY

I hereby declare that this thesis contains no material that has been accepted for any other degree in any university. To the best of my knowledge and belief this thesis contains no material previously published or written by any other person. The work submitted in this thesis is the product of my own original research, except where I have duly acknowledged the work of others.

Žilina, 29.4. 2014 DANIEL VÁCLAVIK

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor Ing. Peter Tarábek, PhD. for his continuous patience and guidance in all the time of research and writing of the thesis. Also, I thank my parents for supporting me throughout all my studies at University.

ABSTRACT

VÁCLAVIK, Daniel: Licence plate character classification using the SVM. Faculty of Management Science and Informatics – Supervisor: Ing. Peter Tarábek, PhD. – Žilina 2014 – 53 pages.

The thesis' main aim was to perform an assessment of the support vector machine (SVM) and related methods, and develop a license plate character classification utilizing the SVM classification model. Such classifier is required to run under real-time conditions and perform the character recognition with a reasonable accuracy. The overview of object recognition and the SVM-related methods is also outlined.

The novel image description method (utilizing image histogram projections) and enhanced Directed Acyclic Graph SVM multi-class method are proposed. The grid-search technique is utilized for the SVM and kernel parameter identification. The experimental results and analysis are presented for a range of the SVM models.

Keywords: machine learning – optical character recognition – image classification – support vector machine – LIBSVM

ABSTRAKT

VÁCLAVIK, Daniel: Využitie SVM na klasifikáciu znakov získaných z evidenčných čísel vozidiel [diplomová práca] – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky – Vedúci práce: Ing. Peter Tarábek, PhD. – Žilina 2014 – 53 strán.

Hlavným cieľom práce je preskúmanie možností SVM pre klasikačné úlohy v oblasti inteligentného rozpoznávania obrazu. Navrhnutá metóda pre klasifikáciu znakov získaných z evidenčných čísel vozidiel je schopná pracovať v real-time podmienkach a vykazuje nadpriemerné výsledky. Práca obsahuje krátke zhrnutie teoretických poznatkov o SVM a rôznych metód rozpoznávania obrazu.

V práci je predstavená metóda deskripcie obrazu, ktorá využíva vertikálne a horizontálne projekcie histogramu a nový spôsob klasikácie využívajúci rozhodovací orientovaný acyklický graf. Výsledky experimentov pre rôzne nastavenie parametrov SVM a jadra (kernel) sú predstavené spolu s vyhodnotením získaných výsledkov.

Kľúčové slová: strojové učenie – optické rozpoznávanie znakov – klasifikácia obrazu – support vector machine – LIBSVM

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

GLOSSARY

ANPR	Automatic Number Plate Recognition
LP	License Plate
OCR	Optical Character Recognition
ML	Machine learning
PCA	Principal component analysis
SV	Support vector
DDAG	Decision Directed Acyclic graph
ANN	Artificial neural network
OSH	Optimal separating hyperplane
RBF	Radial basis function
XML	eXtensible Markup Language
CSV	Comma-separated values file
DLL	Dynamically linked library
OO	Object-oriented
SMO	Machine learning

1 INTRODUCTION

The natural evolving in the traffic control management has led to the vivid development in the field of Automatic Number (License) Plate Recognition systems (ANPR, ALPR) in the past decades. A surveillance camera placed near the traffic is now capable of capturing a vehicle moving in high speed. The retrieved video frames can be processed, analyzed and the read LP identification number can be sent to the officer or traffic engineer in a blink of an eye.

The part of an ANPR software system responsible for the correct LP characters' recognition usually employs some Optical character recognition (OCR) technique. Some machine learning method can also be a part of such solution. Somewhere at the beginning of the modern LP issuing era, there was obviously a motivation to construct a set of LP characters which could be very easily distinguished from each other. The strict rules, which apply for the license plate formats employed in the member countries of the European Union, is very popular for the ANPR software system producers mainly because of the character normalization.

The thesis' main aim was to perform an assessment of the support vector machine (SVM) and related methods, and develop a LP character classification utilizing the SVM classification model. Such classifier is required to run under real-time conditions and perform the character recognition with a reasonable accuracy. The SVM, in its conventional form a binary classifier, must be adjusted in order to classify all 36 alphanumerical characters of the English alphabet. Since most of the ANPR systems are deployed outdoors, the image acquisition conditions can be very diverse and the SVM must be capable of learning to perform the classification even under some less or more complicated terms.

The fundamentals of image processing, objects recognition methods and the Support vector machine in particular are the presented in *Section 2*. Upon reading this chapter, the reader shall be equipped with the general knowledge of the SVM fundamentals and the methods employed in the preprocessing phase of the LP classification method.

Section 3 focuses on the set of available SVM libraries and frameworks. A comparison of the libraries' features, shortcomings and overall performance is presented. The core functionalities of the application developed for the SVM training and LP classification method testing is outlined.

In the experiment-filled *Section 4*, the important SVM hyperparameters and the kernel configurations are identified, tested and evaluated. The image descriptor method employing the horizontal/vertical image histogram projections on the overlapping regions within the

image grid is proposed. A novel approach to the Directed acyclic graph SVM formulation is outlined and evaluated.

Section 5 features the final evaluation of the experimental results and the concluding assessment of the proposed method as well as the suggested ideas for the further research in the extensive Support vector machine field.

2 IMAGE PROCESSING AND OBJECT RECOGNITION

This chapter aims to provide the reader with a fast and comprehensive overview of the image processing methods encompassed throughout the Thesis, and Support Vector Machine (SVM) background in particular. Upon reading this chapter, the reader shall be equipped with the basic ideas which stand behind the SVM and other learning algorithms.

2.1 FUNDAMENTALS OF IMAGE PROCESSING

In this section, we would like to outline the basic principles of some of the most important image processing techniques used in computer vision. The basic image manipulation techniques, which are used in early stages of computer vision applications, are called **preprocessing** methods. These methods manipulate the image's pixels in order to enhance certain properties of the input image (color, sharpness, etc.). **Feature extraction** methods' purpose is to transform the input image into the set of features (called *feature vector*). By extracting these features from the input images, the input data redundancy can be suppressed so that only the relevant information is used in the later stages. These features are usually used as the input for the **object recognition** algorithms such as Support Vector Machine (SVM) or Artificial Neural Network (ANN), which aim is to detect and/or classify the objects within the images.

2.1.1 Preprocessing

In this section, some of the image processing techniques which are used mainly in the early stages of the learning algorithms are introduced. At first, the input images are preprocessed using a combination of the preprocessing methods such as resizing, thresholding or smoothing. Preprocessing methods like smoothing or thresholding play also an important part in the overall performance of the learning algorithms such as the SVM or the ANN.

2.1.1.1 Histogram

A **digital image** is usually two-dimensional representation of a reality. It contains objects and the background. Through the process of digitization, the image is transformed from its

natural, continuous form to the discrete form comprising of the finite set of points. Such representation enables the computer to process images very naturally as a 2D matrix of values. Each *pixel* is not only a point representation in the image but rather a rectangular region, the elementary cell of the grid. The number at each pixel (basic picture element) position represents a value of the brightness function *b(x,y),* also known as luminance or intensity.

The **histogram** is the graph which projects the distribution of the intensity levels (black or gray pixels) in an image. In a sense, it is also a plot of the discrete probability density function which defines the likelihood of a given pixel value occurring within the image [1]. The histogram can be formed by calculating the frequency of occurrence of each of the permitted pixel values (e.g. 256 possible pixel values for grey-scale 8-bit image). The plot of original (top left) and corresponding equalized (top right) histogram next to the Baboon images are shown in *Fig. 2.1* from [2].

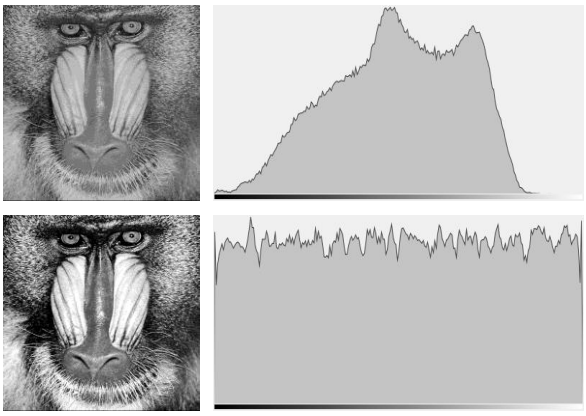


Fig. 2.1 Histogram equalization

2.1.1.2 Discrete convolution

Discrete convolution is a method that uses a window of a specific finite size (usually a square matrix of odd edge length) to scan across the image. The output pixel value is the weighted sum of the pixels from the input image within the window. The weights are the constants from a particular filter and are assigned to every pixel of the window itself. The window with its weights is called the ***convolution kernel***[3].

The convolution function formula is shown in (2.1). The indices i and j denote x and y coordinates of the input image, *g(i,j)* is the intensity value of the [i,j]-pixel in the output image. Kernel h (sometimes referred to as a filter) is the K *x* K (K is a positive odd value) matrix having the origin at the centre of the kernel *h(0,0)*.

$$g(i,j) = f * h = \sum_{m=-K}^{K} \sum_{n=-K}^{K} f(i-m, j-n) \cdot h(m,n) \qquad (2.1)$$

2.1.1.3 Noise reduction

The level of noise present in the input images that are to be processed may have a severe impact on the overall success rate of the machine learning methods. The methods used to suppress the low image quality due to the high noise level are called *smoothing (blurring) filters*. These methods utilize the use of a discrete convolution, and apply kernels of different sizes and array values to get a filtered output image.

The basic smoothing method is considered to be **mean** filtering. It assigns equal weight w_K to all the pixels in the pixel's neighborhood. A weight of $w_K = 1/(N \cdot M)$ is used for $N \times M$ neighborhood. This approach should be able to eliminate the significance of the unrepresentative pixels in the processed pixel's surroundings. Often a *3 x 3* square kernel is used, as shown in *Fig. 2.2*, although larger kernels (e.g., a *5 x 5* square) can be used for more severe smoothing [4].

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fig. 2.2 Filtering kernels
Mean filtering kernel (left)and discrete approximation to Gaussian function (σ = 1) (right).

Mean filter is a simple and easy way to implement smoothing method, but accompanies significant drawbacks. For example, a single pixel with a very unrepresentative value can severely affect the mean value of all the pixels in its neighborhood. The filtering does not preserve edges and may cause a problem if sharp edges are required in the output. **Median** filter usually yields better results compared to the mean filter as we can observe in *Fig. 2.3,* but it comes with the worse computational speed. Obviously, the computational complexity rises because of the need to order the array of values for each pixel's neighborhood to find the median statistics of the set.

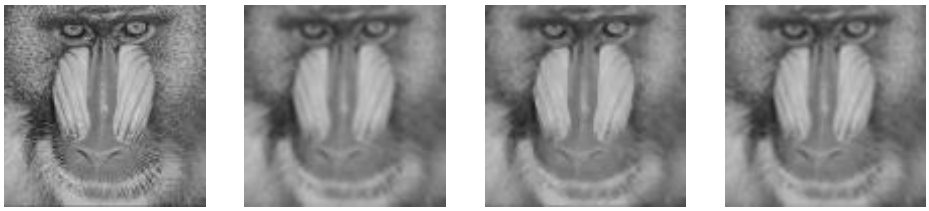


Fig. 2.3 Smoothing operations
From left to right: Original Baboon image, output image after 3 x 3 mean smoothing, output image after 3 x 3 median smoothing and output image after 3 x 3 Gaussian smoothing (σ = 1). Source:[2].

In some cases, **Gaussian** filtering might be more suitable solution because of its ability to preserve the neighborhood's natural weights, which are more significant towards the "centre" of the matrix. Because of this, Gaussian filtering provides better smoothing and edge-preservation results at the same time. *"Gaussian smoothing commonly forms the first stage of an edge-detection algorithm, where it is used as a means of noise suppression."* [1]

2.1.1.4 Thresholding

Thresholding (sometimes referred to as **binarization**) is considered to be the simplest method of *image segmentation*. The actual image segmentation is done by comparing the intensity values of each pixel with the respect to the threshold and assigning new value – usually black (representing the object) and white (background) to the corresponding pixel in the output image. This corresponds to Black (0) and white (255) respresenting an object and a background in 8 bit grey-scale output image in (2.2).

$$o(x,y) = \begin{cases} 0 & if\ i(x,y) > t \\ 255 & otherwise \end{cases} \tag{2.2}$$

Adaptive thresholding

The extremely challenging task of selecting the optimal global threshold value can be overcome by applying a different threshold value for each pixel in the input image. The **local threshold** is determined by the values of the pixels in the neighborhood of given pixel. This approach assumes that illumination may differ significantly over the whole image but is usually uniform within a local neighborhood. The local threshold value *t* is determined by the thresholding function which takes the local *N x N* pixel neighborhood into consideration only.

$$o(x,y) = \begin{cases} 0 & if\ i(x,y) > T(x,y) \\ 255 & otherwise \end{cases} \tag{2.3}$$

Some of the well known local binarization methods include *Niblack*, *Bernsen*, *Savakis* and *Savuola* methods which prove to be more than competitive alternatives to the global binarization methods such as *Mean*, *Median* or *Otsu* [5]. The main disadvantage is the computational complexity of such methods, which sometimes makes them not suitable for real-time applications.

2.1.2 Feature description (extraction)

Feature extraction methods are used to decrease the size of the input vector required to describe the input data with a sufficient accuracy. Edge detection descriptors (*Canny*, *Sobel*, *Prewitt*), corner detection descriptors (*Harris operator*, *Shi and Tomasi*) or *Histogram of gradients* (*HoG*) are just some of the most widely used techniques.

2.1.2.1 Vertical/Horizontal image projection plot analysis

Vertical projection plot analysis is the method which is widely used in object (character) segmentation. The potential occurrence of an object having certain shape or size can be detected processing a *vertical* or a *horizontal projection* histogram plot. *"Obtaining a binary image, the idea is to add up image columns or rows and obtain a vector (or projection), whose minimum values allow us to segment characters."* [6]

Generally the black pixels or gradient values of a plate are projected vertically. The local minimums called *valleys* are assumed to be the spaces between the characters. This approach does not work on such vertical projected histogram when there has been an overlap which could be because of bad thresholding or noise or even drop of rain [7].

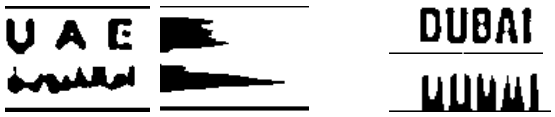


Fig. 2.4 Vertical and horizontal projection histogram plots

2.1.2.2 Histogram of oriented gradients

The basic idea behind the Histogram of oriented gradients (HoG) descriptor is that local object appearance and shape can often be characterized by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions [8]. First, the image is divided into small connected regions called *cells*. A histogram of gradient directions or edge orientations is calculated from the pixels within each cell. The most common method is to apply the one-dimensional filtering mask [–1,0,1] and $[-1,0,1]^T$ in horizontal and/or vertical direction for all pixels within the cell. The combination of these histograms then represents the descriptor.

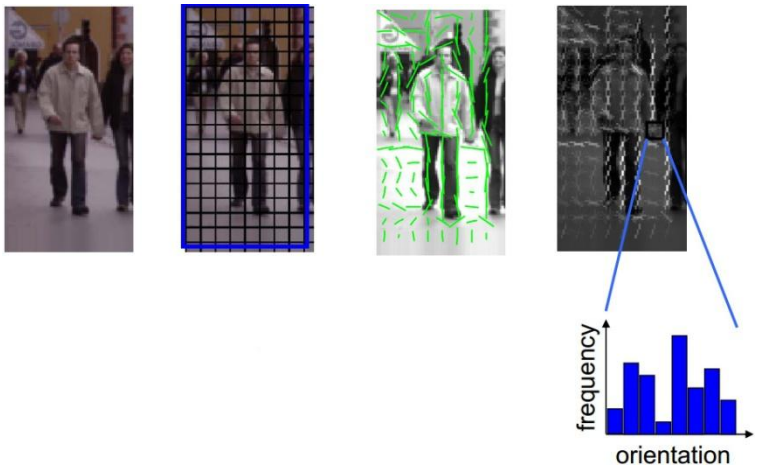


Fig. 2.5 Histogram of oriented gradients example

As for the vote weight, pixel contribution can either be the gradient magnitude itself, or some function of the magnitude. In the actual tests done by the researchers who first described the HoG descriptor in [8], the gradient magnitude itself generally produced the best results. We can see the original image, the image tilled into *8 x 8 pixel* cells, the image with the dominant directions and their magnitude and the HOG descriptor in *Fig. 2.5* from [9].

For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a *block*, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination or shadowing. The HOG descriptor maintains a few key advantages over other descriptor methods. Since the HOG descriptor operates on localized cells, the method upholds invariance to geometric and photometric transformations, except for object orientation.

2.1.2.3 Principal component analysis

The Principal Component Analysis (PCA) is a method used in data analysis. Its goal is to transform the input variables (the term "correlated" in statistics theory) to a set of new independent variables (thus "uncorrelated"). These new variables are called *principal components*, or axes [10]. Decreasing the number of principal components causes the problem to shrink in size. *"From a set of N images in a space of p descriptors, the goal of this method is to find a representation in a small space of q dimensions ($q \ll p$)"* [10] which preserves the "best description" of an object.

The method's input are the variables structured in a matrix M with N rows and P columns as shown in (2.4).

$$M = \begin{pmatrix} X_{11} & \cdots & X_{1P} \\ \vdots & \ddots & \vdots \\ X_{N1} & \cdots & X_{NP} \end{pmatrix} \tag{2.4}$$

The output are then the variables $C_1, C_2, \ldots, C_k, \ldots, C_q$, where $q < p$ and $C_i : \forall i \in 1..q$ are uncorrelated, of maximum variance and of decreasing importance. Each random variable $X_i = (X_{1i}, \ldots, X_{Ni})$ has *a mean* $\overline{X_i}$ and standard deviation σ_{x_i}. The method's aim is to determine the correlation matrix in order to measure the dependences among variables and subsequently create a variable list in a decreasing importance.

$$\widetilde{M} = \begin{pmatrix} \dfrac{X_{11} - \overline{X}_1}{\sigma(X_1)} & \cdots & \dfrac{X_{1P} - \overline{X}_P}{\sigma(X_P)} \\ \vdots & \ddots & \vdots \\ \dfrac{X_{N1} - \overline{X}_N}{\sigma(X_N)} & \cdots & \dfrac{X_{NP} - \overline{X}_P}{\sigma(X_P)} \end{pmatrix} \tag{2.5}$$

2.1.3 Object recognition

In image processing, object recognition (pattern recognition) methods are the algorithms devised to predict (classify) the class membership of an unknown patterns. These methods can be split into two main groups according to the nature of the specific task they intended to perform:

- *Detection methods* are algorithms dealing with detecting instances of certain objects (such as vehicles, roads on the satellite images, or human faces) in the input images or video streams. They usually yield binary decision – the instance of an object is either in the image, or it is not present at all.
- *Classification methods* perform various recognition routines in order to assign a certain class membership to the given object within the image. Obviously, any detection problem is a classification problem as well – only the number of possible class assignments is binary (an object is either present, or it is not in the image). The number of classes usually determines the complexity of the given classification task, and may vary from some tens (optical character recognition) to some hundreds (vehicle's make and model recognition) to possibly thousands of different classes (protein classification).

2.1.3.1 Machine learning

Machine learning (ML) is a field of study which is not limited not only to the object recognition. It is considered to be a branch of artificial intelligence, which focuses on the systems, which are capable of learning from the data. A machine learning system is usually trained on a dataset consisting of known and labeled patterns I order to learn to distinguish between the classes present. After successful learning, such system is capable of a new sample classification. There are three main approaches in ML training procedures [11]:

- *Supervised learning*, in which a "teacher" provides output targets for each input pattern, and corrects the network' s errors explicitly;
- *Semi-supervised* (or *reinforcement*) *learning*, in which a teacher merely indicates whether the network' s response to a training pattern is "good" or "bad";
- *Unsupervised learning*, in which there is no teacher, and the network must find regularities in the training data by itself.

A set of the ML methods include well known and widely used algorithms such Artificial neural networks (ANN), Support vector machine (SVM), Bayesian networks (BN) or Decision tree learning and many other novel approaches. We present an introduction to the

ANN and BN fundamentals in the following chapter. *Section 2.2* focuses entirely on the fundamentals of the SVM as it is also the main study subject of the Thesis.

Artificial neural network

An artificial neural network (ANN) is a system that is inspired by a system of interconnected neurons such as the human central nervous system, brain in particular. A system emulating this natural behavior is capable of completing tasks such as machine learning and pattern recognition.

A neural network contains a large number of very simple processing units, analogous to neurons in the brain. At each moment in time, each unit simply computes a scalar function of its local inputs, and broadcasts the result (called the *activation value*) to its neighboring units. The units in a network are typically divided into *input units*, which receive data from the environment; *hidden units*, which may internally transform the data representation; and/or *output units*, which represent decisions.

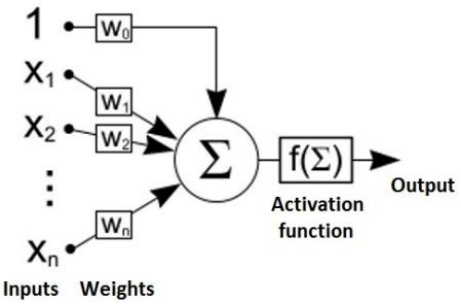


Fig. 2.6 Artificial neural network scheme

The units in a network are organized into a given topology by a set of *connections*, or *weights*, shown as lines in a diagram in *Fig. 2.6* from [12]. Each weight has a real value, typically ranging from $-\infty$ to $+\infty$, although sometimes the range is limited [11]. The value (or strength) of a weight describes how much influence a unit has on its neighbor; a positive weight causes one unit to excite another, while a negative weight causes one unit to inhibit another. Connectivity between two groups of units, such as two layers, is often complete (connecting all to all), but it may also be random (connecting only some to some), or local (connecting one neighborhood to another).

Computation always begins by presenting an input pattern to the network, or *clamping* a pattern of activation on the input units. Then the activations of all of the remaining units are computed. In unstructured networks, this process is called *spreading activation*; in layered networks, it is called *forward propagation*, as it progresses from the input layer to the output layer. A given unit is typically updated in two stages: first we compute the unit's *net input* (or internal activation), and then we compute its *output activation* as a function of the net input.

Net input x_j is usually computed as the weighted sum of its inputs,

$$\boldsymbol{x}_j = \sum_i y_i w_{ij} + \theta_j \tag{2.6}$$

where y_i is the output activation of an incoming unit, and w_{ij} is the weight from unit i to unit j. In general, the net input is offset by a variable bias term θ_j, but, in practice, this bias is usually treated as another weight w_{j0} connected to an invisible unit with activation $y_0 = 1$ [11].

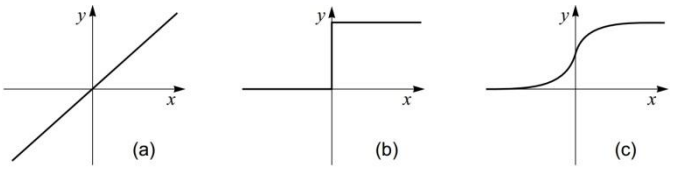


Fig. 2.7 Local, threshold and sigmoidal activation functions

Once we have computed the unit's net input x_j, we compute the output activation y_j as a function of x_j. This activation function (also called a *transfer function*) usually takes one of three forms – *linear*, *threshold*, or *sigmoidal* – as shown in *Fig. 2.7* from [11]. In the linear case, we have simply $y = x$. This is not used very often because it's not very powerful: multiple layers of linear units can be collapsed into a single layer with the same functionality. In order to construct nonlinear functions, a network requires nonlinear units. The simplest form of nonlinearity is provided by the threshold activation function illustrated in *Fig. 2.7(b).*

$$y = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.7}$$

The most common function, according to [13] is now the sigmoidal function illustrated in *Fig. 2.7(c).*

$$y = \frac{1}{1+exp(-x)} \quad \text{or} \quad y = tanh(x) \tag{2.8}$$

"Training a network, in the most general sense, means adapting its connections so that the network exhibits the desired computational behavior for all input patterns." [11] In general, networks are nonlinear and multilayered, and their weights can be trained only by an iterative procedure, such as gradient descent on a global performance measure [14]. This requires multiple passes of training on the entire training set; each pass is called *iteration* or *epoch*. Moreover, since the accumulated knowledge is distributed over all of the weights, the weights must be modified very gently so as not to destroy all the previous learning. A small constant called the *learning rate* (ε) is thus used to control the magnitude of weight modifications [11].

Bayesian network

Bayesian networks, also known as *belief networks*, belong to the family of probabilistic graphical models. These graphical structures are used to represent knowledge about an uncertain domain. [15] (In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables). These conditional dependencies in the graph are often estimated by using known statistical and computational methods.

Bayesian network is defined by:

- A directed acyclic graph (DAG) $G = (V, E)$, where V is a set of nodes of G, and E is a set of the edges of G;
- A finite probabilistic space (Ω, Z, p);
- A set of random variables associated with graph nodes and defined on (Ω, Z, p) as:

$$p(V_1, V_2, .., V_n) = \prod_{i=1}^{n} p\big(V_i | C(\mathrm{V_i})\big) \tag{2.9}$$

where $C(V_i)$ is a set of *causes* (*parents*) of V_i in graph G.

The BN learning problem is to, given training data and prior information (e.g., expert knowledge, casual relationships), estimate the graph topology (network structure) and the parameters of the joint probability distribution in the BN. Learning the BN structure is considered a harder problem than learning the BN parameters. "*It has been proven that [BN structure learning] is an NP-Hard problem, and therefore any learning algorithm that would be appropriate for use on such a large dataset such as microarray data would require some form of modification for it to be feasible."* [10] Therefore, some heuristic method, such as the Maximum-likehood estimation, is usually introduced, according to [15].

2.2 SUPPORT VECTOR MACHINE

Consider a typical classification problem. Some input vectors (**feature vectors**) and some **labels** (classes of objects) are given. "The objective of the classification problem is to predict the labels of new input vectors so that the error rate of the classification is minimal." [16] We have N training points, each input point $\boldsymbol{x}_i$ is a vector consisting of D feature values (*dimensionality* of D) and a single binary value y_i – so called *label* (+1 and -1 for example). The training data point can be expressed in the following form:

$$\{\boldsymbol{x}_i, y_i\}\ \ i = 1..N, y_i \in \{-1,1\}, \boldsymbol{x} \in \boldsymbol{R}^D \tag{2.10}$$

2.2.1 Linear SVM

Basic application of the Support Vector Machine (SVM) solves only a binary (two-class) classification problem. The aim of the classifier is to learn the similarities in the objects' feature vectors within the same class and generalize these similarities sufficiently enough so that the unseen data point from the same class could be predicted. *"Support Vector Machines can be thought of as a method for constructing a special kind of rule, called a* ***linear classifier****, in a way that produces classifiers with theoretical guarantees of good predictive performance (the quality of classification on unseen data)"* [17].

We can simplify the task, which usually consists of several tens, hundreds or even thousands dimensions, and consider only a two-dimensional input space (feature vectors with D=2). An example (*Fig. 2.8* from [16]) shows the case in which two-dimensional objects can be separated by a line and form class *<green>* and class *<red>* with no inter-class member swapping. We call such set of objects **linearly separable**.

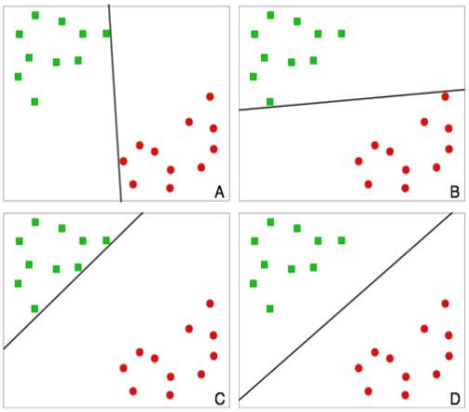


Fig. 2.8 Finding the optimal hyperplane for linearly separable data points

As we can see, there are many possible "lines" (or *hyperplanes* if $D > 2$, which separate the space into two half spaces) separating the objects of the two classes which can be drawn (in *Fig. 2.8*). Any given hyperplane can be described by:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \tag{2.11}$$

where $\boldsymbol{w}$ is a *norm* to the hyperplane and $\frac{b}{\|\mathbf{w}\|}$ is a perpendicular distance from the hyperplane to the origin.

"Among all hyperplanes separating the data, there exists a unique one, called the ***optimal hyperplane****, distinguished by the maximum margin of separation between any training point and the hyperplane."* [18] We can see that such an optimal separating line is the one on the bottom right picture in *Fig. 2.8*. *"A special characteristic of SVM is that the solution to a classification problem is represented by the support vectors that determine the* ***maximum margin hyperplane****."* [19]

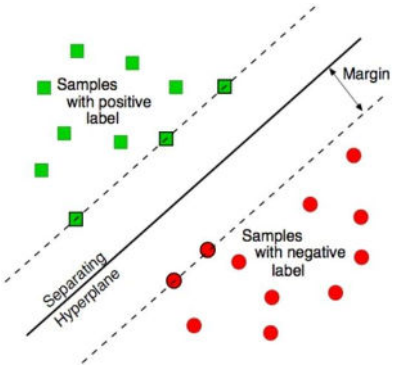


Fig. 2.9 Separating hyperplane, maximum margin and support vectors

2.2.1.1 Optimization problem behind the SVM

*"**Support vectors** are the examples closest to the separating hyperplane and the aim of the SVM is to orientate this hyperplane in such a way as to be as far as possible from the closest members of both classes."* [20] Once the separating hyperplane is chosen, only support vectors (data points lying on the margin – circled points in Fig. 2.9 from [16] and in *Fig. 2.10* from [20]) are considered for the classification task and other data points become irrelevant and not needed anymore. Support vectors can be geometrically described as:

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b = +1 \qquad \textit{for } H_1 \tag{2.12}$$

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b = -1 \qquad \textit{for } H_2 \tag{2.13}$$

The distance between the origin and the hyperplane H_1 is equal to $|+1 - b| / \|\boldsymbol{w}\|$ (d_1 in *Fig. 2.10*). Similarly, the distance between the origin and the hyperplane H_2 is equal to $|-1 - b| / \|\boldsymbol{w}\|$ (d_2). *"We can easily calculate that the margin of the linear classifier H (the distance between hyperplanes H_1 and H_2) equals* $\frac{2}{||\boldsymbol{w}||}$*."* [19]

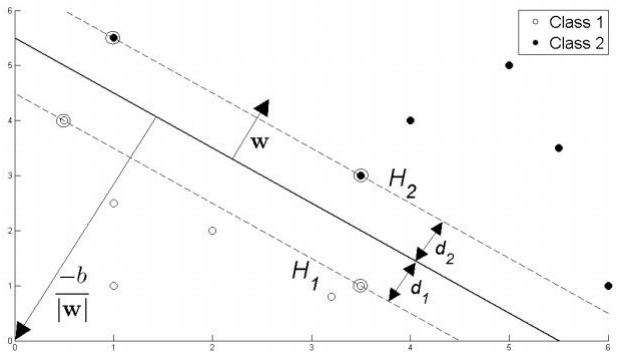


Fig. 2.10 Defining a separating hyperplane in two-dimensional feature space

Considering the above, the process of finding the optimal separating hyperplane can be formulated as solving a mathematical **optimization problem**, which can be expressed by the following model [9] p.13:

$$\max_{\boldsymbol{w}} \quad \frac{2}{\| \mathbf{w} \|} \tag{2.14}$$

$$subject\ to \quad \boldsymbol{w}^T \boldsymbol{x}_i + b \geq 1 \quad if\ y_i = +1 \quad for\ i = 1..N \tag{2.15}$$

$$\boldsymbol{w}^T \boldsymbol{x}_i + b \leq -1 \quad if\ y_i = -1 \quad for\ i = 1..N \tag{2.16}$$

Or equivalently:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2} \| \mathbf{w} \|^2 \tag{2.17}$$

$$subject\ to \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \quad for\ i = 1..N \tag{2.18}$$

This optimization problem represents the minimization of a quadratic function under the linear constraints, which can be solved via quadratic programming.

2.2.1.2 Soft Margin SVM

In the previous text, we introduced linear SVM model, which works only if the training set is linearly separable, which causes the separating hyperplane to correctly classify all patterns. Such classifiers are also known as *hard margin* classifiers. However, *"the linear separability of two classes of the patterns might not be a valid assumption for real-life applications."* [19] The **non-separability** of the input data might be caused by the noise in the input data, or there is simply a high degree of "similarity" between the training members of the two classes. We can see such example in *Fig. 2.11* from [9]. Of course, no linear (hard margin) classifier can be computed for this learning set, but *"several hyperplanes can be calculated in such a way as to minimize the number of classification errors."* [19] The majority of the data points lie on the correct "side" of the hyperplane, but there is a small number of misclassified objects.

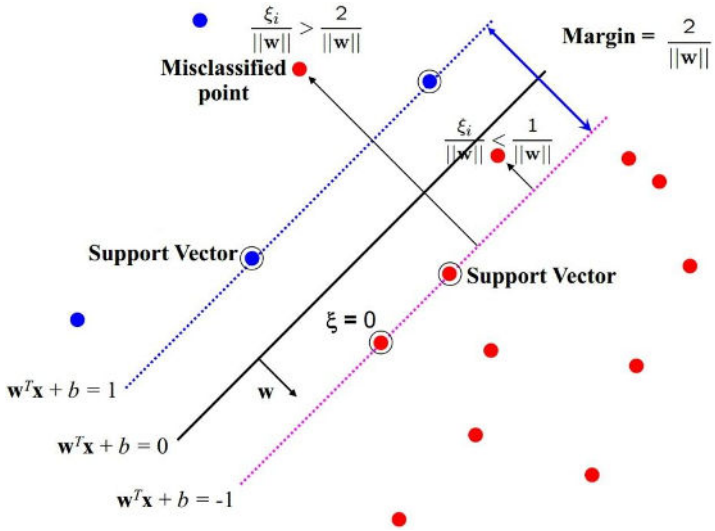


Fig. 2.11 Soft margin SVM

In order to adapt the SVM to handle the not fully linearly separable data, we can relax the constraints (2.18) by allowing for some misclassified points. Introducing a positive **slack variable** (ξ), we can add a penalty to the points lying on the "wrong" side of the hyperplane. The penalty for the patterns classified correctly will then equal to zero. We can add a penalty increasing with the point's distance from the "correct" hyperplane in the case of *misclassified* data points. We have an example of a misclassified data point, margin violating point and the correctly classified points in the *Fig. 2.11*. Support vector machines which allow the linear constraint relaxations are commonly referred to as *soft margin* classifiers.

As we are trying to minimize the number of misclassifications and the extent of the "impact" these points have on the overall classification error, we need to adapt the objective function (2.17) to minimize the new penalty term as well:

$$\min_{\boldsymbol{w},\forall\xi_i} \quad \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{\forall i} \xi_i \tag{2.19}$$

$$subject\ to \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i \quad for\ i = 1..N \tag{2.20}$$

$$\xi_i \geq 0 \quad for\ i = 1..N \tag{2.21}$$

Parameter C in (2.19) controls the trade-off between the slack variable penalty and the size of the margin [20]. $\boldsymbol{C}$ is also referred to as a **regularization parameter**. We can observe that every constraint can be satisfied if ξ_i is sufficiently large. It can be adjusted by the user, and can either increase or decrease the penalty for classification errors. *"A large C assigns a higher penalty to classification errors, thus minimizing the number of misclassified patterns. A small C maximizes the margin so that the OSH (optimal separating hyperplane) is less sensitive to the errors from the learning set."* [19]

2.2.2 Non-linear SVM

"SVM models were originally defined for the classification of linearly separable classes of objects," [19] but can be adapted to classify even the models where more complex relationships exist between input parameters and the pattern labels. To discriminate linearly non-separable data, we can fit the SVM model with nonlinear functions to provide an efficient classifier.

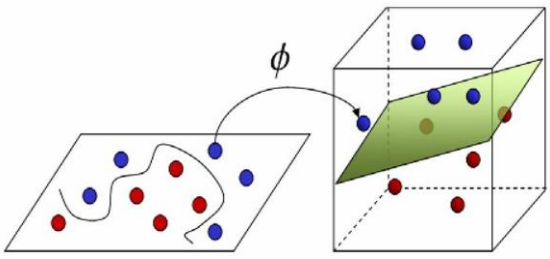


Fig. 2.12 Input space and high dimensional feature space

The main idea is to map the data into a different space, called **feature space**, and to construct a linear classifier in this space. *"It can also be seen as a way to construct non-linear classifiers in the original space."* [17] (See also *Fig. 2.12* from [21].) This process is sometimes referred to as a **kernel trick**.

The non-linear functions used to fit the non-linear models are all based on calculating the inner products of two vectors, *"which can be a significant problem if the feature space is too large and it is often not possible at all."* [16] The aim of the kernel trick is to avoid the explicit inner product calculation.

If we have a function ϕ (called **feature mapping function**),

$$\phi : R^h \rightarrow \omega \tag{2.22}$$

$$\boldsymbol{x} \rightarrow \phi(\boldsymbol{x}) \tag{2.23}$$

which maps the input space into the high-dimensional feature space ω, we can define (under certain conditions), an inner product in feature space which has an equivalent **kernel** in input space:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \tag{2.24}$$

To safely assume that the kernel K represents an inner product in a feature space, it has to, according to the *Mercer's theorem*, which says that kernel K must be a *symmetric positive definite function,* satisfy the following conditions:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{k}^{\infty} a_k \phi_k(\boldsymbol{x}_i) \phi_k(\boldsymbol{x}_j), a_k \geq 0 \tag{2.25}$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \iint K(\boldsymbol{x}_i, \boldsymbol{x}_j) g(\boldsymbol{x}_i) g(\boldsymbol{x}_j) d\boldsymbol{x}_i d\boldsymbol{x}_j > 0 \tag{2.26}$$

Using kernel to calculate the inner product of the feature vector in higher dimension the linear constraints (2.20) can be rewritten to (2.28). The following optimization problem is then solved.

$$\min_{\boldsymbol{w}, \forall \xi_i} \quad \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{\forall i} \xi_i \tag{2.27}$$

$$subject\ to \quad y_i(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad for\ i = 1..N \tag{2.28}$$

$$\xi_i \geq 0 \quad for\ i = 1..N \tag{2.29}$$

2.2.2.1 Kernels

The following are some of the most popular kernels used in the Support vector machine according to [22]. The schematic plots of the kernel are in *Fig. 2.13* from [19].

Linear (Dot) kernel

The basic – linear kernel *"should be used as a test of the nonlinearity in the training set, as well as a reference for the eventual classification improvement obtained with nonlinear kernels."* [19]

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j \tag{2.30}$$

Polynomial kernel

The polynomial kernel is a simple and efficient method for modeling nonlinear relationships but has the downside of possible overfitting that may appear when the degree (parameter *d*) increases.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (a + \boldsymbol{x}_i \cdot \boldsymbol{x}_j)^d \tag{2.31}$$

Radial Basis Function (RBF) kernel

The *Gaussian* form of the RBF kernel is commonly used, with parameter σ controlling the shape of the separating hyperplane.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left(-\frac{\| \boldsymbol{x}_i - \boldsymbol{x}_j \|^2}{2\sigma^2}\right) \tag{2.32}$$

Sigmoid kernel

Also known as a *neural* kernel. The hyperbolic tangent (*tanh*) function, with a sigmoid shape, is the most used transfer function for artificial neural networks, but can be uses a kernel function for SVMs as well.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = tanh(a\boldsymbol{x}_i \cdot \boldsymbol{x}_j + c) \tag{2.33}$$

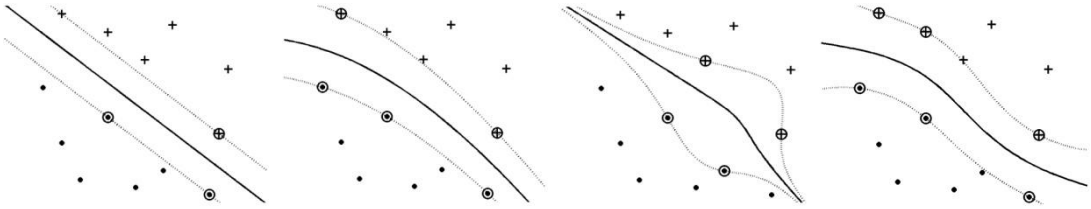


Fig. 2.13 Linear, polynomial (d = 2), polynomial (d = 10) and RBF kernel

2.2.3 Multi-class SVM

Until now, only binary (two-class) classification has been discussed. *"Many real-world problems, however, have more than two classes – an example being the widely studied optical character recognition (OCR) problem."* [18] Due to various problems evolving from the nature of the multi-class problems, a direct solution of multiclass problems using a single SVM formulation is usually avoided according to [23]. The usual approach is to train a combination of several binary SVMs to solve a given multiclass problem.

2.2.3.1 One versus the rest

One vs. Rest or *one-against-all* is probably the earliest multi-class method according to [22]. It constructs k SVM models considering the k-class classification problem. In the i-th SVM, one class represents the data points from the i-th class, the other class contains all example from the other classes (e.g. class <*K*> and class<A..J,L..Z,0..9> in OCR classification problem).

The main disadvantage of this approach lies in so called **winner-takes-all** strategy. *"The binary classifiers used are obtained by training on different binary classification problems, and thus it is unclear whether their real-valued outputs (before thresholding) are on comparable scales. This can be a problem, since situations often arise where several binary classifiers assign the pattern to their respective class (or where none does); in this case, one class must be chosen by comparing the real-valued outputs."* [18] On the other hand, the arguably important advantage is the small number of SVMs which is required for the classification, as opposed to the one-versus-one approach.

2.2.3.2 Pairwise classification

Pairwise or *one-against-one* method constructs one binary classifier for every pair of distinct classes. Therefore, together $K(K-1)/2$ binary classifiers are constructed. Each binary SVM is trained taking the examples from data points from class i as positive and the examples from j as negative. During the classification, so called **max-vote** strategy is applied. After each of the $K(K-1)/2$ binary classifiers makes its vote, max-vote strategy assigns x to the class with the largest number of votes.

For thirty six alphanumerical classes (letters A..Z,0..9), *one-against-one* approach results in total number of 36(36-1)/2 = *630* binary SVM classifiers, which is significantly higher than using one-against-all approach which needs only *36* classifiers. The difference of the computational cost of both methods is obvious. *"Although this suggests larger training times, the individual problems that we need to train on are significantly smaller, and if the training*

algorithm scales superlinearly with the training set size, it is actually possible to save time." [18] Individual classifiers, however, tend to result in the fewer support vectors than they would be in the one-against-all approach, which may potentially save some computational cost, considering the overall model reduction.

2.2.3.3 Directed acyclic graph SVM

A Directed Acyclic Graph (DAG) is a graph whose edges have an orientation and no cycles [24]. Multi-class SVM model utilizing some desired properties of a DAG is called (Decision) Directed acyclic graph (DAGSVM) and was first introduced in [24]. This method modifies one-against-one approach – the training phase of the both is the same – all $K(K\text{-}1)/2$ binary SVMs need to be trained, although not all binary SVMs are used during a new pattern's classification. A rooted binary directed acyclic graph is constructed with the internal nodes and the leaves representing the binary SVMs as shown in *Fig. 2.14* from [24].

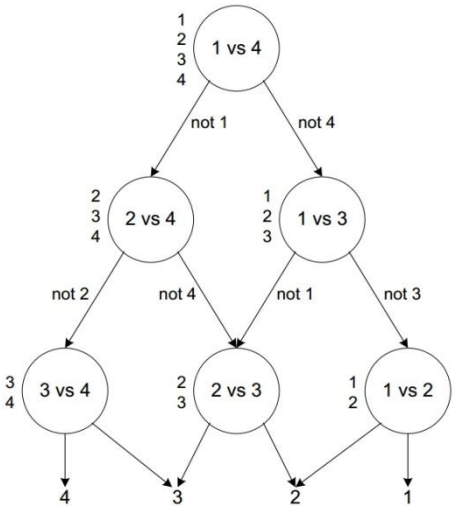


Fig. 2.14 Directed acyclic graph SVM

Given a test sample, starting at the root node, the binary decision function is evaluated. Then it moves to either left or right depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class. An advantage of using a DAG is that some analysis of generalization can be established [25]. The advantage of DAGSVM is that classification is faster than by conventional pairwise support vector [26] which is given by the expected height of the binary DAG being $K - 1$.

3 SVM LIBRARIES AND TESTING APPLICATION

In this section, the review of the popular SVM solutions is presented. A comparison of the libraries' features, shortcomings and overall performance is presented. The core functionalities of the application developed for the SVM training and testing purposes is outlined.

3.1 SUPPORT VECTOR MACHINE LIBRARIES

The quest to the labyrinth of the online SVM community was initially a search for the perfect SVM package that is around. Great expectations set for the tool's complexity and the ease of use were soon terminated. We have not found any library or framework incorporating all the required tasks at the same time, although some of the examined libraries are very professional and powerful tools indeed. The initial requirements set for the library implementing SVM are as follows:

- Certain level of complexity and robustness (and compactness at the same time),
- Written preferably in C/C++,
- Open Source code,
- Capable of solving large-scale problem – possibly tens of thousands of input data and input features
- Utilizing the SVM and kernel parameters and sparse model structure as described in *Section 3.1.3*

There are multiple relevant SVM library listings available online – [27], [28], [29] and [30] for example. These lists were put together independently by the researches in the field of computer vision and SVM in particular which were mostly evaluated or studied themselves. Basically all top ranked libraries in these lists were written in C/C++ which proves that the family of the popular managed languages such Java or C# is not a competitive player in this field.

The preferability of C/C++ to Java or C# is due to the computational complexity of the studied problem. In terms of computation speed, C/C++ can easily outperform these. All other libraries other than those written in C/C++ were taken out from the further search. As we can observe from the list of SVM software compiled from the Google search directory ranks in

[28], there are hardly any Java or C# libraries, and if, they are usually just a ported versions of some of the popular C/C++ libraries as the case of LIBSVM.

3.1.1 Toolboxes and frameworks

There is a number of possibilities in utilizing some of the existing solutions in the field of machine learning (ML). Some of the established machine learning frameworks also offer the SVM support. However, these frameworks (or toolboxes) tend to be of rather massive dimensions. It is simply a natural consequence of the ability to support the numerous algorithms and methods at the same time. Some of the most popular ML frameworks include:

- **SHOGUN** – general purpose machine learning (C++) package with focus on large scale learning and kernel methods (SVM) in particular; provides generic SVM object interfacing to several different SVM implementations (LIBSVM, SVMLight, liblinear, etc.) [31],
- **Shark** – a cross-platform feature-rich C++ machine learning library; provides methods for linear and non-linear optimization, kernel-based learning algorithms, artificial neural networks and other ML techniques [32],
- **dlib** – another ISO C++ portable ML library, which provides, on top of the numerous learning algorithms implemented, a very detailed documentation and useful debugging modes [33].

Other very popular and highly regarded tools include some MATLAB toolboxes (Spider [34]) and python-based packages (scikit-learn [35]).

3.1.2 Dedicated SVM libraries

The user popularity and the citation figures for the relevant dedicated SVM libraries are presented by Martin Sewel in [28]. We reckon the top three libraries from this list to be probably the most widely used tools utilized by the researchers in the field of ML and SVM. The following five libraries were ranked highest in this listing:

- **LIBSVM** – Definitely the most popular SVM dedicated library endorsed by both researchers and ordinary users, introduced by Chang and Lin in [36]; it provides different SVM formulations as well as efficient cross-validation model selection, multi-class classification support and other common SVM learning features; apart from its original C/C++ implementation, LIBSVM offers interface in other 15 different languages together with many extensions and domain-specialized plugins. It is open source and the source code and binaries can be downloaded from [37],

- **SVMLight** – C implementation of SVM classification method introduced by Joachims in [38] provides fast optimization algorithm as well as the ability to solve both classification and regression problems; other useful features include the error rate, the precision and the recall estimation calculations for the given SVM model, it is also capable of handling large-scale problem and many thousands of support vectors. This library is open source, however, the source code is reportedly compilable only on some of the Linux distribution operating systems. Linux, Windows and Mac OS binaries are available at [39],
- **SVMTorch** – originally a C library completely rewritten in C++ in object style introduced by Collobert in [40] features the common SVM framework methods and introduces the sparse vector and binary file format for the SVM model persistency and is tailored especially for large-scale problems. It runs (and compiles) under Linux and Windows and can be downloaded from [41],
- **mySVM** – a C++ library with a unique built-in support of the novel *anova* kernel; available for UNIX/Windows operating systems at [42],
- **TinySVM** – another C++ implementation of the support vector classification and support vector regression method; it utilizes the fast optimization algorithm stemming from [38]; the authors report that the optimization for handling the binary features is two times faster than in the SVMLight's implementation. It provides multi-platform source files and binaries [43] and ports to Perl, python and Java.

3.1.3 Selecting a suitable library

In order to select a relevant library, one might consider some dedicated SVM package to be a smarter solution compared to utilizing a bulky general machine learning framework. An undisputed advantage a compactness of such solution, which usually come in few source code files, is the fact that it can be easily recompiled without the need to deal with the possible build issues which tend to happen with more complex software. Another convenient offer of the dedicated SVM libraries is the fact that, in most cases, there is a university researcher with a certain level of expertise in the field of SVM, which guarantees the correctness of the particular SVM methods employed in the library.

Prior to any testing or development, we identified certain parameters which needed to be met for the selected library, which would serve as a supporting package for the research in the Thesis (see *Tab.* 3.1).

All of the "top five" SVM packages offer the same kernel selection and the sparse vector file format seems to be a common standard. The availability of an SVM multiclass algorithm played an important role in the decision making. Both LIBSVM and SVMTorch use well known and researched multiclass algorithms (see *Section 0* for details). The main deciding point, however, turned out to be the date of the most recent update of a particular package. All

libraries except LIBSVM were are a bit outdated versions mostly from a decade ago. The user's community for these libraries is either non-existent or one must cope with a rather austere "readme.txt" kind of documentation. All of the specified parameters were met by the LIBSVM package only.

	LIBSVM	SVMLight	SVMTorch	mySVM	TinySVM
Language	**C/C++**	C	C++	C++	C++
Multiclass algorithm	**1-vs-1**	proprietary	1-vs-rest	✘	✘
Cross-validation	✓	✓	✘	✓	✘
Sparse data format	✓	✘	✓	✓	✓
Kernel - linear	✓	✓	✓	✓	✓
Kernel - polynomial	✓	✓	✓	✓	✓
Kernel - RBF	✓	✓	✓	✓	✓
Kernel - Sigmoidal	✓	✓	✓	✓	✓
Kernel - User defined	✓	✓	✓	✓	✘
Recent update	**IV/2014**	VIII/2008	XI/2001	VI/2004	VIII/2002

Tab. 3.1 Dedicated SVM libraries overview

The authors of the **LIBSVM** have created a vivid community around the library, therefore its development is progressing. *"From 2000 to 2010, there were more than 250,000 downloads of the package. In this period, we answered more than 10,000 emails from users"*, the LIBSVM authors remark in [36]. The library is also successfully used in the fields of computer vision (LIBPMK), natural language processing (Maltparser), neuroimaging (PyMVPA), or Bioinformatics (BDVal).

The library's regular updates include both functionality improvements (employing a novel approach, supporting new SVM model formulations and so on) and bug fixes, which are usually submitted by its loyal users. The library features easy-to-understand source code and usable API. The library's complex FAQ list is also available online [37].

The LIBSVM consists of the four modules – the core module (now completely rewritten in OO style C++), training, testing and scaling modules, which usually serve as the user API. The SVM optimization method employs the Sequential Minimal Optimization (SMO) method introduced by Platt in [44] and offer additional functionalities such as shrinking (shortens the time needed for a SVM model training, kernel caching (the values of kernel calculations are retained in RAM for fast retrieval), or posterior prediction probability estimation.

3.2 SUPPORTING APPLICATION

From early on, there was a obvious need for a tool capable of performing automated SVM tests. The time and energy dedicated to the application development were rewarded later on in the final stages of the research, while conducting numerous experiments. We

reckon that such complex tasks would have not been accomplished in such a timely manner with this level of ease.

3.2.1 Libraries and tools

As discussed in the previous section, LIBSVM package was utilized for the testing purposes during the course of the thesis' research. However, not only its API functions alone were used. Some of the core LIBSVM functions were altered or completely rewritten (eg. in the case of the proposed Enhanced DDAGSVM multi-class method – see *Section 4.5.1*). For the image manipulation routines, computer vision and image/video processing library OpenCV, was employed.

3.2.1.1 OpenCV

OpenCV (Open Source Computer Vision) is a library consisting of the functions aimed mainly at performing *real time computer vision* tasks. OpenCV's application areas include *motion tracking*, *segmentation*, *gesture recognition*, *object identification*, *human-computer interaction* and many others.

This popular library offers a cross-platform solution (on-demand support for Windows, UNIX, Mac OS, Android and iOS). It was originally written in C, but completely rewritten in OO C++. This interface is the most widely used version nowadays, a variety of customized wrappers for other languages such as C# (*EmguCV*), Python, Java and Ruby is available. OpenCV incorporates Intel's *Integrated Performance Primitives* (IPP), a multi-threaded library optimized for data processing applications. Once the IPP library is detected on the OS environment, the IPP's set of optimized routines is employed to accelerate the performance of the OpenCV-dependent application. The general OpenCV package comes with the support of *Intel Threading Building Blocks* (TBB), library for easier multithread image processing handling.

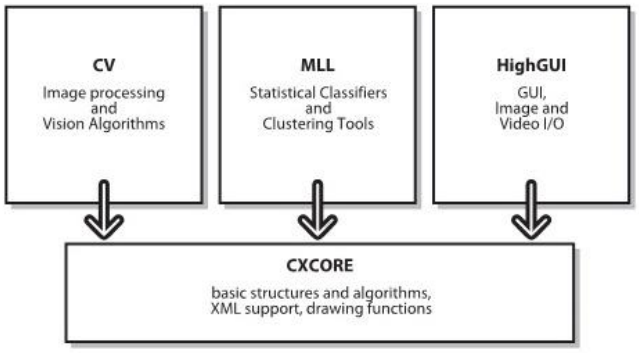


Fig. 3.1 OpenCV modules
Dependencies within OpenCV's modules which are grouped into four main components. Source: [45]

The SVM supporting application features OpenCV's *core*, *imgproc* and *highgui* modules. The OpenCV's functions employed for our purposes are not directly dealing with any machine learning methods. The main usage is during the preprocessing phase of the SVM classifier training and prior to the actual character classification procedure, however the ease of use and the familiar interface made OpenCV a great supporting utility for the main – classification task.

3.2.2 Application overview

At first, we must note that the application is not intended to serve as a general SVM testing tool. Some assertions are given, thus the image file format, image bit depth or a possible number of classes currently supported are limited. This tool performs three main tasks, which evolved in our research to be almost a regular indispensable routine:

- *SVM training and testing wizard* – this functionality enables the user to choose from the range of the supported parameters, kernels, preprocessing functions or multi-class algorithms to put together a suiting SVM model configuration, which is subsequently trained and tested on the given datasets,
- *Batch SVM configuration generator* – this tool makes the routine of batch SVM configuration generation an easy task – a user can select the parameters and a range of values, which are to be generated. These options are then combined and configuration files are generated in XML format ready to be processed by the engine. This "batch" approach saves both time and effort which would be otherwise needed to perform the task manually.

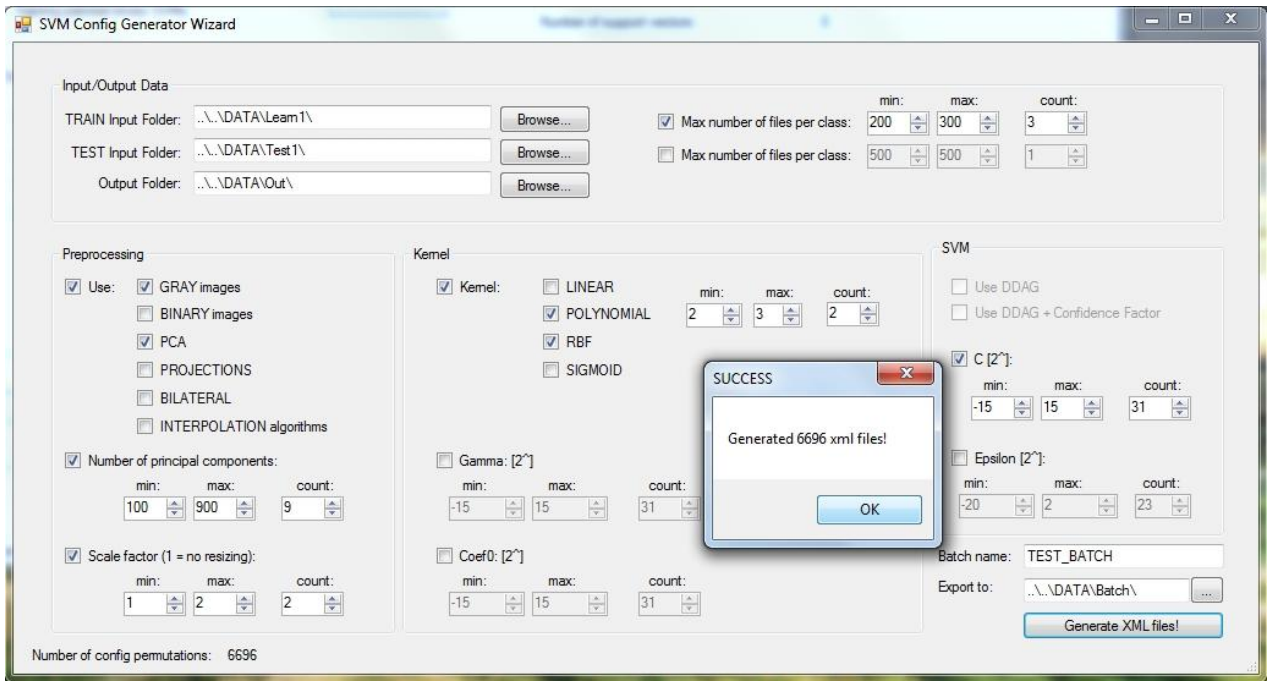


Fig. 3.2 Batch SVM configuration generator

- *Batch SVM classification launcher* – when using this mode, the input SVM configuration files are read and the SVM models are trained (if needed) and tested on the given datasets. The SVM training and testing can be a time consuming routine, which may require hours to complete. While using the batch launcher, all the procedures are done automatically, and after training & testing for a single SVM model is finished, the preliminary results are exported in a *CSV* file. A user can pause or stop a single SVM run task or terminate the whole batch.

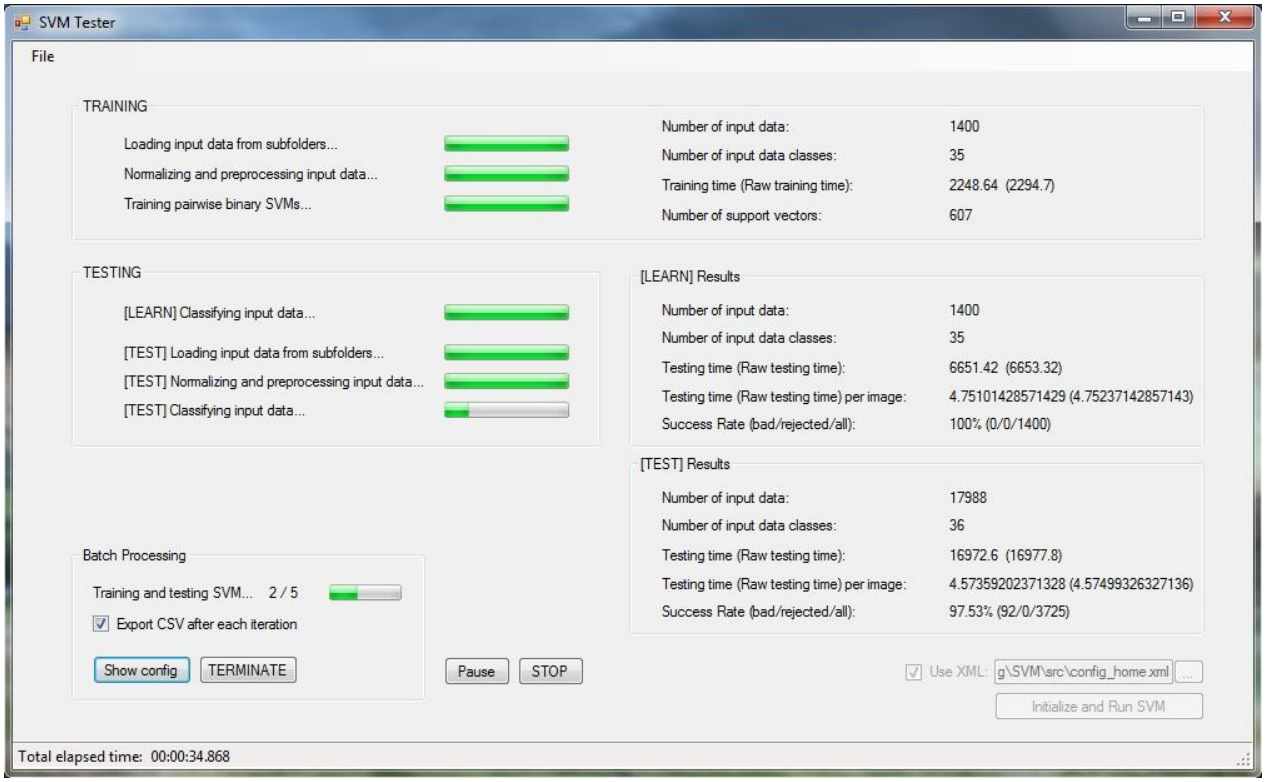


Fig. 3.3 Batch SVM classification launcher

The application utilizes a rather functional approach to the user interface and focuses on the application compactness and ease of use. The outputs are generated in user-friendly formats (*XML*, *CSV*). The optional settings also allow a user to generate the misclassified images, a list of most frequently misclassified class pairs with the total figures for each pair, or the list of the misclassified (or rejected) images per class. User is continuously given the latest updates from the SVM engine (current number of misclassified/rejected/positive images, average computational speed, etc.).

The core engine (SVM functionalities) is available as C++ DLL. The application logic and the user interface are implemented in C#. The application engine runs in a worker thread, which keeps the application responsive. C++/C# Interop (Marshalling) services are used when converting the C++ structures to its C# counterparts (and vice versa). A simple API exposing

the common SVM methods (train an SVM classifier with the given configuration, classify given sample) was created for the DLL.

4 SVM CLASSIFICATION METHOD

In this section, we present the experimental results of various SVM related techniques employed during the time of our research. The image descriptor method employing the horizontal/vertical image histogram projections on the overlapping regions within the image grid is proposed. A novel approach to the Directed acyclic graph SVM formulation is outlined and its ability to reduce an SVM model's error rate is evaluated.

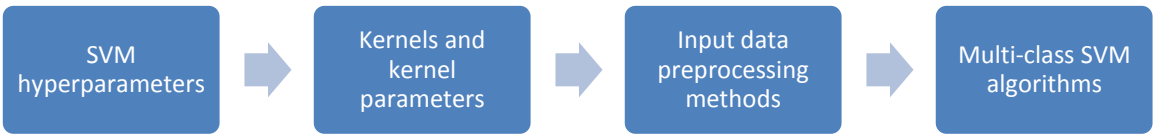


Fig. 4.1 Building up the SVM classification method

4.1 INPUT DATA AND METRICS

A dataset comprised of some hundreds of license plate character images was used while developing the character classification method. It consists of 10 digits and 25 letter of the English alphabet (letter 'Q' is omitted due to the lack of samples for this particular class). The set of images within each class was split into two subsets – 200 images, which were processed during the training phase, and 500 images used for testing purposes. Such a high number of samples in the training set than may yield better generalization performance indicator for the particular SVM classification model.

The input images were already normalized to 8-bit grayscale with the size of $48\ x\ 32$ pixels. The fairly large input image dimensions and the level of similarities within images from a particular class resulted in the very promising classification success rates early on. Somewhere at the beginning of the modern LP issuing era, there was obviously a motivation to construct a set of LP characters which could be very easily distinguished from each other. The strict rules, which apply for the license plate formats employed in the member countries of the European Union, is very popular for the ALPR software system producers mainly because of the character normalization.

The feature vector values ($0 - 255$ using the 8-bit images) were scaled to the [-1; +1] range. The process of the input feature scaling has an important influence on the overall SVM classifier performance according the LIBSVM author Lin as remarked in [22]: *"The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the*

calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems." The authors of [22] recommend linearly scaling each attribute to the range [1; +1] or [0; 1]. The same scaling technique than applies to both training and testing data.

We employ the metrics called *success rate*, which is used to express the particular's model generalization performance on the testing data. Given the size of the testing dataset N, and the number of successfully classified images using an SVM classifier N_p, the classifier's success rate S can be calculated as the following ratio:

$$S = \frac{N_p}{N} \tag{4.1}$$

The tests described throughout the Thesis were conducted on a regular Windows PC with 3.4 GHz Intel *i*7 processor and 16 GB of RAM.

4.1.1 Suitable parameter value identification

The authors in [22] identify so called "*grid-search*" as the viable way in the process of finding the optimal parameter values for the given classification problem: "*There are two motivations why we prefer the simple grid-search approach. One is that, psychologically, we may not feel safe to use methods which avoid doing an exhaustive parameter search by approximations or heuristics. The other reason is that the computational time required to find good parameters by grid-search is not much more than that by advanced methods since there are only [small number of] parameters."* The process of grid-searching can be easily parallelized because the coupled parameters in the grid are independent. "*Many of advanced methods are iterative processes, e.g. walking along a path, which can be hard to parallelize."*

Since conducting a complete grid-search may be time-consuming the authors in [22] recommend using a coarse grid first. A possible way is to construct the grid of exponentially growing parameter values, and conduct the initial search. We use the powers of two (2^n) throughout all the tests while searching for the optimal parameter values. This approach has proven to be both feasible (not very time consuming) and sufficient in terms of the results obtained while using this technique. After the coarse grid-search is done and a possible "optimal" value range is set, one can conduct a finer grid-search within this range to identify the "optimal" parameter value.

4.2 SVM MODEL HYPERPARAMETERS

Finding the fitting SVM model hyperparameters (meta-parameters) – regularization constant and stopping criterion – which contribute enormously to the overall classification success rate are reasonably considered to be the first step in any SVM model creation.

Although the optimal selection of these parameters is highly data-dependent, a certain generality can be deduced.

The authors in [22] consider the RBF kernel with SVM hyperparameter values $C = 1$ a $\gamma = 1/N$ (N is the number of classes in the classification problem) a reasonable starting point. The default value for a stopping criterion ε is usually to 0.001 (adopted by LIBSVM as well). This parameter is also tuned, after the initial "sufficiently good" value of C, which has a grave influence on the overall result, is found.

4.2.1 Regularization parameter

Regularization parameter introduced in the objective function term (2.20) is the penalty assigned to the classification points laying on the "wrong" side of the separating hyperplane. *"The penalty for classification errors increases when the capacity C increases, with the consequence that the number of erroneously classified patterns decreases when C increases. On the other hand, the margin decreases when C increases, making the classifier more sensitive to noise or errors in the training set."* [19]

In order to fulfill both of these requirements – reasonably small C for a large margin classifier and a large C for a small number of classification errors at same time, one must accept a certain tradeoff. Setting the appropriate C can balance the tradeoff between margin maximization and error minimization [46]. The grid-search for parameter C was done within the range of $[2^{-15}; 2^{15}]$.

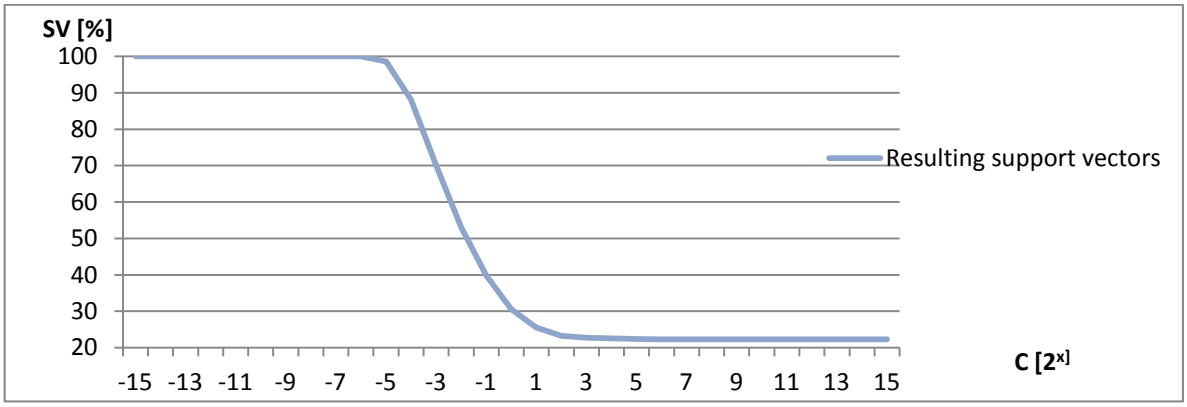


Fig. 4.2 Percentage of support vector selected for different C parameter values

As we can observe in *Fig. 4.2*, the increase in the value of C caused the classifier to continuously create larger margin, ie. the output number of model's support vectors was decreasing. There is also a continuous increase in the number of support vectors (up to the point when all training data points are treated as the support vectors) defining the model while the value of C decreases. This increase caused the SVM training process to take up more than

ten times more compared to the lower values of *C*, with the significantly worsened testing time per image.

The danger of model overfitting is clearly visible and can be avoided by choosing the value of parameter *C* near the peak in the curve of the success rate figures on the testing dataset as shown in *Fig. 4.3*. *C* value was set to 16, as this value lies within the curve's peak range and yields both reasonable classification and testing speed rates.

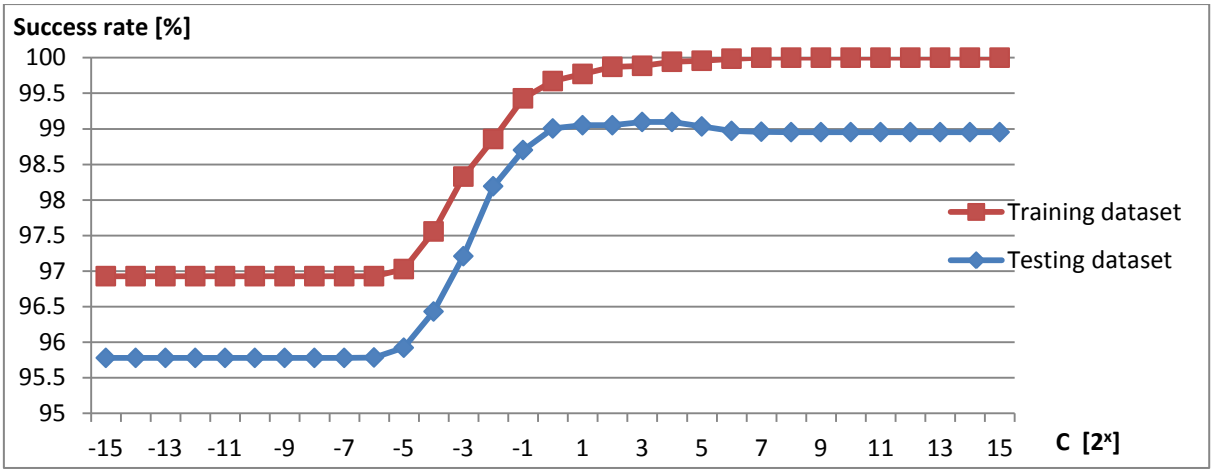


Fig. 4.3 Success rates for different C parameter values

4.2.2 Stopping criterion

The tolerance of stopping, or termination criterion, while optimizing (ie. minimizing a loss function) the SVM model during the training phase is set by parameter ε. This value can affect the number of output support vectors – the bigger ε, the fewer support vectors are selected [47]. *"An increase in ε means a reduction in requirements for the accuracy of approximation. It also decreases the number of SVs, leading to data compression."* [48]

We have assumed that any ε value from the relevant range of ε values can be chosen without losing the level of optimality of the output SVM model, because of the high dependency on the input data volume. Initially, a grid-search in the reasonable range $[2^{-14};2^{0}]$ was conducted for $C = 16$.

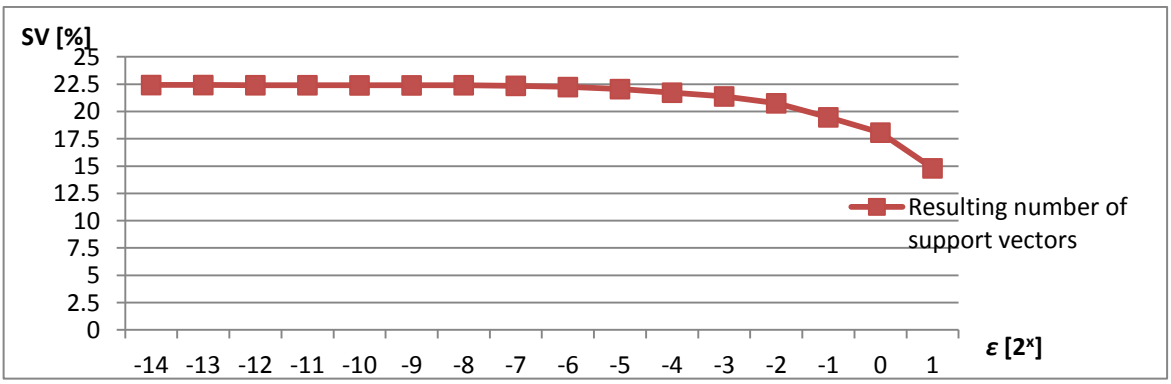


Fig. 4.4 Percentage of support vector selected for different ε parameter values

As shown in *Fig. 4.4*, the model complexity increases with the decreasing value of parameter ε. It is clearly a result of a more thorough optimization, as we can observe in the increasing number of iterations done to achieve approximately the same results as with the model with higher ε. Another negative product of the decrease in ε is the higher number of support vectors generated by the SVM, which causes an increase in the testing speed while maintaining the testing success rates.

Another, finer, grid-search was conducted using the full range (near the curve's peak as shown in *Fig. 4.3*) of the C values. As presumed, the results (as shown in *Fig. 4.5*) confirmed the suitability of both values, $C = 16$ and $\varepsilon = 2^{-1} = 0.5$.

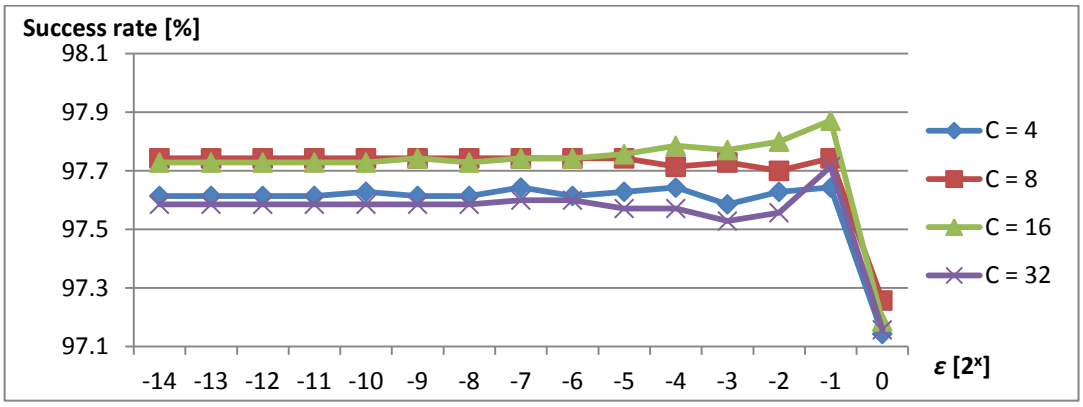


Fig. 4.5 Testing success rate for different C and ε parameter values

4.3 KERNELS AND KERNEL PARAMETERS

When trying to find the appropriate kernel and its parameter's values, one could consider the number of features and the number of instances (training and testing dataset). Hsu et al. in [22] suggest the three possible cases:

- Number of instances ≪ number of features
- Both numbers of instances and features are large
- Number of instances ≫ number of features

The authors in [22] suggest using linear kernel in the first case – *"Apparently, when the number of features is very large, one may not need to map the data [to the higher dimensional feature space]."* When both numbers of instances and features are sufficiently large, the linear and non-linear usually yield similar results. The last case should be treated the opposite way as the first one – one often maps data to higher dimensional spaces according to [22].

Our training dataset consisting of 7000 instances and around 1500 features would lay among the first category, but we decided to find the most suitable kernel setup by conducting grid-search. According to [47], *"selecting a particular kernel type and kernel function parameters is usually based on application-domain knowledge and also should reflect distribution of input (x) values of the training data."*

The **linear kernel**'s undisputed advantage is the fact that one may not need to map the data to the higher-dimensional feature space and only the dot product is calculated for each point while conducting the model optimization. This kernel does not have any additional parameters, therefore it can be used right "out of box".

The negative effect of dataset's linear non-separability can be further improved by adjusting the value of a regularization parameter *C*. The peak in the success rate curve, as plotted in *Fig. 4.6*, is achieved while fixing parameter *C* value to the range of $[2^{-7}; 2^{-4}]$.

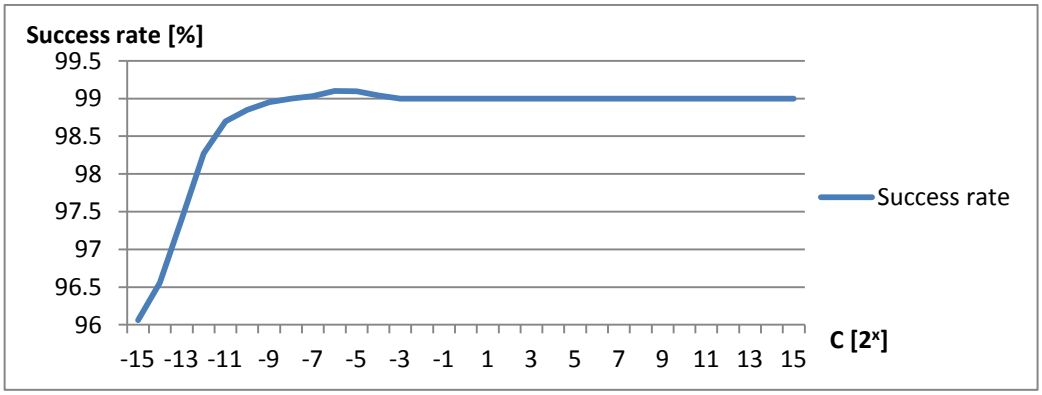


Fig. 4.6 Linear kernel performance using different regularization parameter values

4.3.1 RBF kernel

The Radial Basis Function's parameter σ controls the shape of the separating hyperplane. This parameter plays an important role in the proper model fitting for a particular classification problem. The RBF kernel's performance is highly dependent on this parameter, which can cause the output number of support vector to increase rapidly while choosing the value from an inappropriate range of values. It results in the overfitted model with poor generalization properties (low error rates for training dataset and high error rates for the testing set) and an increased computation time of the classification.

In order to find suitable value of this parameter, we conducted a grid-search for σ value from range $[2^{-25}; 2^{0}]$. The results seem to be a demonstration of Gaussian roots of this kernel as can be observed from the shape of the curve in *Fig. 4.7*.

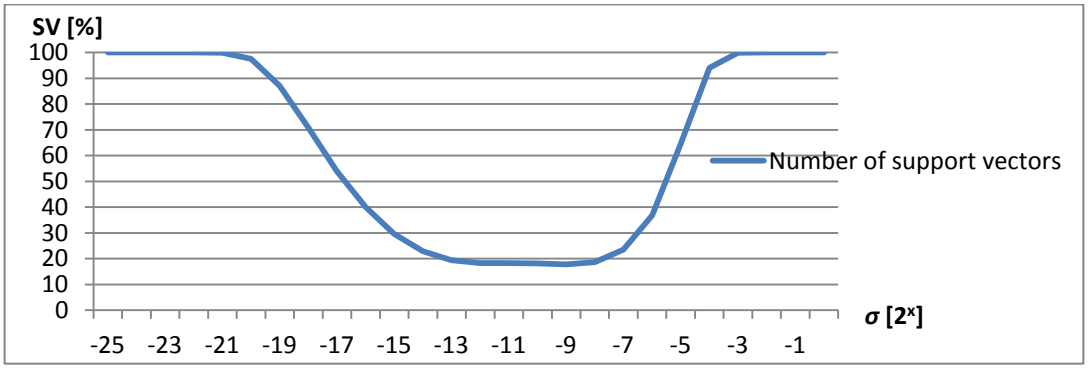


Fig. 4.7 Number of chosen support vectors for different σ values

An obvious model overfitting can be seen with σ value approaching 2^0. The training dataset' perfect error rate is in big contrast with its testing error rate. The SVM models treating whole training dataset (7000) as the support vectors are also trained with the decreasing σ values resulting in the rapidly increasing classification times, but the testing dataset classification remains stable below 4%. The best value for σ was proved to be value near the curve's saddle point $\mathbf{2^{-11}}$ in *Fig. 4.7*.

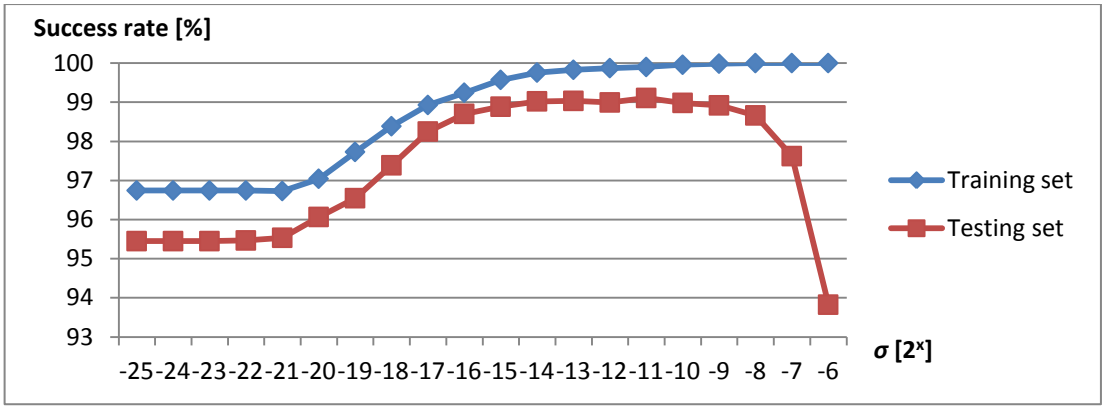


Fig. 4.8 Model's success rates for different σ values

4.3.2 Polynomial kernel

There is usually a new parameter γ introduced (4.2) to the basic polynomial kernel formula (2.31) in the popular SVM libraries polynomial kernel's formulations such as *LIBSVM* or *SVMLight*. This parameter can provide further improvement by scaling the inner product first prior to applying the kernel function.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (a + \gamma \boldsymbol{x}_i \cdot \boldsymbol{x}_j)^d \qquad (4.2)$$

The degree in the polynomial kernel formula determines the "shape" of the separating hyperplane. Obviously, the linear kernel is just a special case of the polynomial kernel having $d = 1$, $\gamma = 1$ and $a = 0$. *"The downside of using [this kernel] is the overfitting that might appear when the degree increases. As the degree of the polynomial increases, the classification surface becomes more complex."* [19]

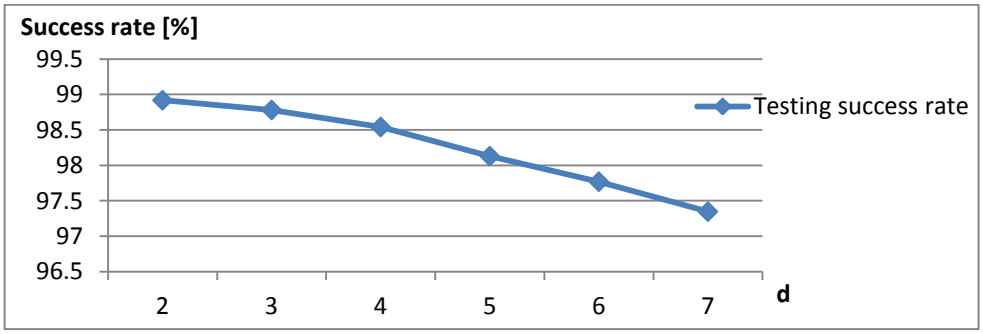


Fig. 4.9 Polynomial kernel's testing success rate for different polynomial degrees

The increase in the value of the degree caused the model to grow in complexity, but classifier's outcome tends to yield rather worse error rates (*Fig. 4.9*) and longer training time caused by the increase in the average number of iterations needed to find the fitting model while optimizing. The polynomial of the **second degree** was considered for the further stages of the polynomial kernel's parameter grid-search.

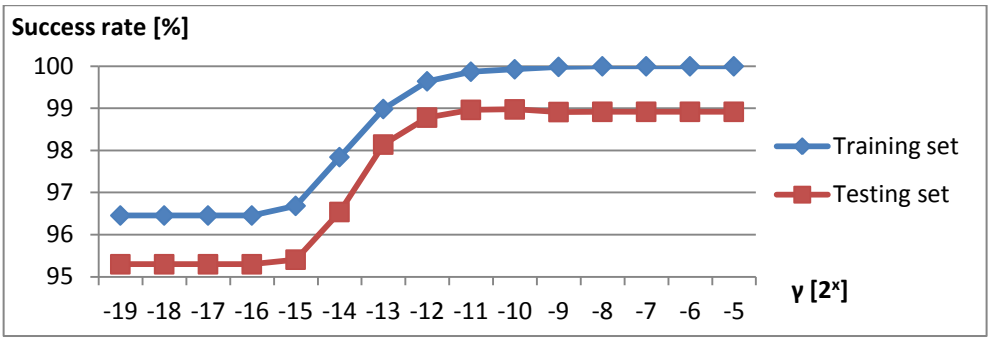


Fig. 4.10 Polynomial kernel's success rates for different γ values

Prior to finding the suitable value of parameter a, scaling parameter γ needed to be fixed. Grid-search was conducted testing the values of γ in range $[2^{-25}; 2^{25}]$ and a fixed value of $a = 0$. The γ value (2^{-10}) with the highest success rate was submitted to the parameter a grid-search to further improve its performance. As shown in *Fig. 4.11* the polynomial kernel's performance can be boosted by adjusting parameter a.

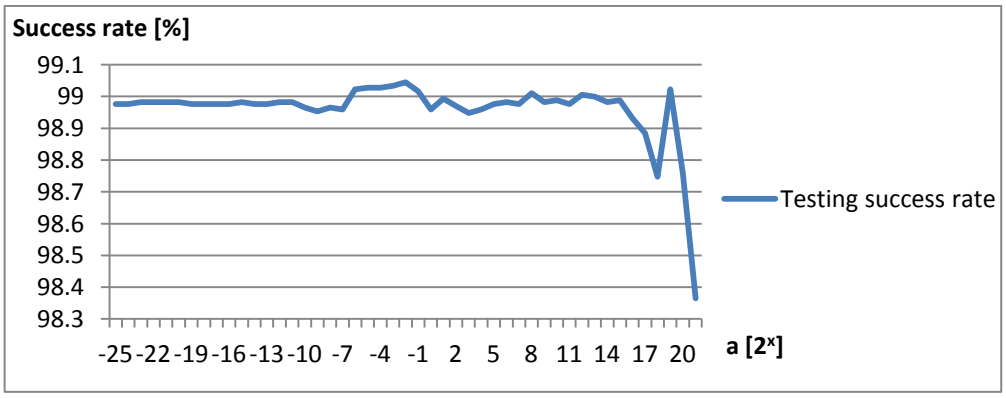


Fig. 4.11 Polynomial kernel's success rates for different a values

4.3.3 Sigmoidal kernel

The hyperbolic tangent (tanh) function, with a sigmoid shape, which is the most used transfer function for artificial neural networks [19], can also be used as the kernel in SVM models. However, not all sigmoidal kernels are valid. It was was proven by Lin et al. in [49], that sigmoid kernel does not yield the inner product of two vectors under some parameters.

The parameters a and c as in (2.33), which have a similar purpose as the parameters γ and a for polynomial kernel were searched for using parameter a grid-search $[2^{-17}; 2^{-6}]$ and parameter c grid-search $[2^{-17}; 2^{5}]$.

As shown in *Fig. 4.12*, the percentage of the output support vector (high number of support vectors chosen for a model usually creates classifiers with weaker generalization capacities) changes significantly over the searched range. The best success rate / SV count ratio happens to be for the model having $a = 2^{-11}$.

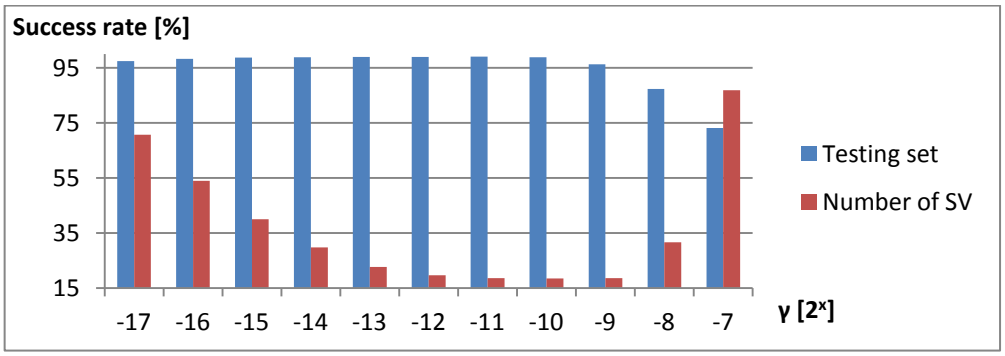


Fig. 4.12 Success rate and percentage of output SVs for different γ values

4.3.4 Conclusion

Even though many possible setups were compared, one may not be able to generalize a particular's kernel performance under all possible conditions. Any kernel's performance is highly domain- and data-dependent. The performance differences obtained while using the best kernel parameter values for each kernel (see previous chapters) were rather small.

We decided to use the **RBF kernel** ($\gamma = 2^{-11}$) in the further stages of the LP character classification method. The lowest error rate obtained by this model is only one of the advantages when using RBF kernel. This kernel setup (together with SVM hyper-parameters $C = 16$ and $\varepsilon = 0.5$) leaves promising opportunities in terms of input data preprocessing and training dataset boosting – the number of support vectors, which is slightly worse (higher) than in the rest of the kernels, could be reduced by the PCA analysis. Many researchers also report the RBF as a very competitive alternative to the linear kernel, which, on the other hand, usually outperforms the RBF kernel in high-scale problems [49].

4.4 INPUT DATA PREPROCESSING

The aim of the input data preprocessing prior to the SVM training is to reduce the number of features. The total number of features for the images being used for the experiments we conducted up to this point is 1536 (48 pixels height *x* 32 pixels width). Such a high number of features can cause the SVM to classify new images in much longer time, which can be crucial downfall when deploying SVM-based system in real-time demanding application.

This chapter is focused on the possible solution to the model complexity issues such as enormous classification time demands described. The aim is to find a viable way to decrease the number of features (and classification time subsequently) and to maintain certain level of success rate. We propose a new image description method, which utilizes the image histogram projections; examine the possible benefit of image scaling and image thresholding and experiment with the possible feature reduction via the Principal Component Analysis (PCA).

4.4.1 Proposed image feature descriptor

The motivation behind the proposed feature descriptor comes from the author's positive experience utilizing the histogram projection plot analysis in the license plate (LP) template classification in [2]. In this thesis, a histogram projection plot analysis system was constructed to decide on the possible object's occurrence in specified region within the LP.

Classifying a character image can be, in a way, reformulated as searching for a set of "objects" (or features) of certain shapes and sizes, which form a particular character in an input image. If this set of features can be extracted for each character using a suiting feature descriptor, the SVM model can be trained to predict the unseen image's character membership.

At first, the proposed feature descriptor splits the image's area into *8 x 8* square regions. The overlapping *8 x 8* regions are also created for vertical as well as for horizontal axis of the image. The values of the feature descriptor vector are then obtained applying the vertical and horizontal image histogram plot analysis within each small region to acquire the valuable, local, information.

In the next stage, the redundant histogram plot values, which are naturally created in each region overlap, are removed. These redundancies arise in the vertical projections of a horizontally overlapping cell pairs and in the horizontal projections of a vertically overlapping cell pairs. A region intensity value is also calculated for each cell as a ratio of the sum of all pixels' intensities to the sum of the maximum possible pixels' intensities in the particular region.

Given the image dimensions *H x W* and the desired region cell width *r*, we get $a = \frac{H}{r}$ cell rows and $b = \frac{W}{r}$ cell columns in each row. After adding the overlapping regions, the total number of cell rows almost doubles and is equal to $a + a - 1$, while the total number of cell columns becomes $b + b - 1$. The number of features per region cell is $r + r + 1$, a summation of a number of vertical and horizontal histogram plot values and the region intensity number. Each r histogram projection values is calculated as the sum of pixel intensities in the particular column (or row) of the histogram. The output number of the image projection plot and intensity value features can be calculated using via the following formula:

$$N' = (a + a - 1)(b + b - 1)(r + r + 1)$$
$$= (2a - 1)(2b - 1)(2r + 1) \quad (4.3)$$

The term (4.3) must be subtracted by the number of the redundant features – the vertical projection values of all horizontally overlapping cells and the horizontal projection values of all vertically overlapping cells. After the subtraction, we get the total number of features for the proposed feature descriptor as follows:

$$N = N' - 8[(a - 1)(2b - 1) + (b - 1)(2a - 1)] \quad (4.4)$$

Given an image's dimensions ($H = 48$, $W = 32$) and a region cell edge length $r = 8$, we get $a = H/r = 48/8 = 6$ cell rows and $b = W/r = 32/8 = 4$ cell columns per row. Using formula (4.4) end up with the following number of features per image:

$$N = (2 \cdot 6 - 1)(2 \cdot 4 - 1)(2 \cdot 8 + 1) - 8[(6 - 1)(2 \cdot 4 - 1) + (4 - 1)(2 \cdot 6 - 1)]$$
$$= 11 \cdot 7 \cdot 17 - 8(5 \cdot 7 + 3 \cdot 11)$$
$$= 1309 - 544 = \mathbf{765}$$

4.4.1.1 Feature descriptor variants

During the development, several variants of the proposed description method were tested. The aim was to try to identify the features which may not have a significant affect on the overall classification success rate and might be leaved out in order to further reduce the descriptor's size. These variants include:

- The basic descriptor as introduced above (variant *A*),
- An alternation of the basic descriptor – the number of the horizontal/vertical histogram projection values were reduced to a half – instead of 8 values per each projection within one region, only 4 were submitted to the descriptor by grouping every two neighboring histogram projection values together (variant *B*),
- A variant omitting all region intensity values (variant *C*),
- A variant omitting all overlapping cells leaving only the regions of the "original image grid" (variant *D*),
- A rather experimental model, which does not take all four corner cells into account, assuming the lack of important character features in these marginal areas (variant *E*).

The results obtained while using the approaches *A* – *E* (shown in *Tab. 4.1*) prove the proposed feature descriptor to be a viable alternative to the SVM models using the raw image pixels. Even though the success rates are a bit worse, the computational time needed for

classification per image are highly in favor of the proposed method (less than a half of the time needed compared to the best RBF kernel configuration using the raw image pixels as the input features – see *Section 4.3.4*).

	A	B	C	D	E	F
No. of features	765	421	**688**	408	697	344
No. of support vectors	1274	1419	**1285**	1465	1334	1479
Testing success rate	98.891%	98.851%	**98.908%**	98.828%	98.816%	98.816%
Testing time per image	1.383 ms	0.939 ms	**1.286 ms**	0.935 ms	1.346 ms	0.817 ms

Tab. 4.1 Performance of the proposed image descriptor variants

One additional descriptor (variant *F*) model was tested, which combined the best variant considering the classification success rate and the one with the best computation speed. The feature descriptor omitting the region intensity value and reducing the number of histogram projection values at the same time (variant C and variant B) resulted in only 344 features in total. Although the promises arising from combination of a feature reduction and success rate sustainability seemed to have a rather positive outlook, the results showed no improvements in neither the success rate, nor the model complexity (the highest number of support vectors generated by this model).

4.4.2 Image resizing

The image scaling (resizing) as the way of reducing an amount of the input features is obviously the easiest approach that might come to one's mind. When using a suitable interpolation algorithm (which yields a somewhat accurate but scaled-down copy of the original image), image resizing technique can be a powerful preprocessing tool. The following interpolation algorithms were tested (*OpenCV*'s implementation):

- Nearest-neighbor interpolation (*NEAR*),
- Bilinear interpolation (*LIN*),
- Resampling method using pixel area relation (*AREA*),
- Bicubic interpolation over 4x4 pixel neighborhood (*CUB*),
- Lanczos interpolation over 8x8 pixel neighborhood (*LANC*).

Scaling an image ($H\ x\ W$ dimensions) by a given scale factor f yields a new image of $H/f\ x\ W/f$ dimensions. The tests using the scale factors (for each interpolation algorithm) up to $f = 8$ (the resulting scaled-down image in such case has only a mere $6\ x\ 4 = 24$ pixels!) were conducted.

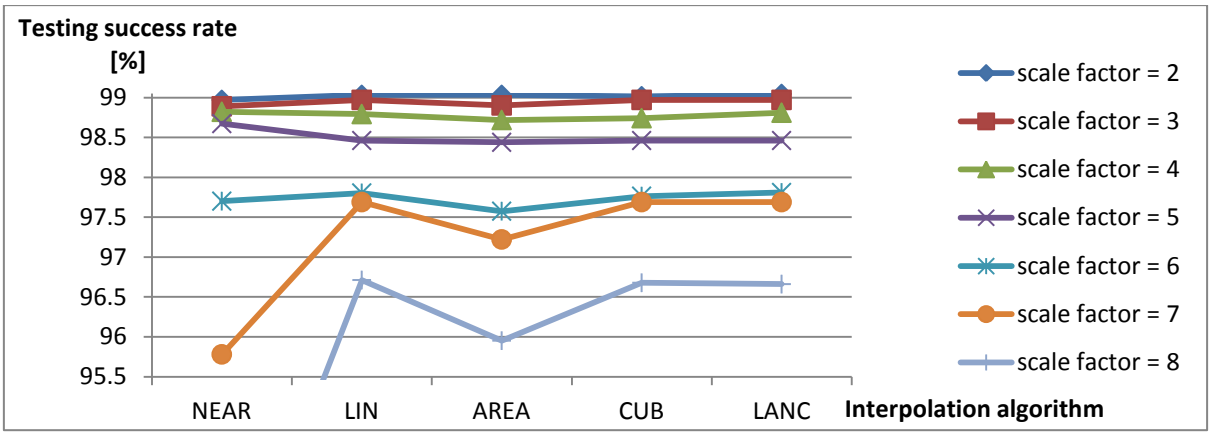


Fig. 4.13 The testing success rates obtained for different scale factors and interpolation algorithms

As shown in *Fig. 4.13*, both scale factor $f = 2$ and scale factor $f = 3$ yield very similar results – around 99%. The real difference in the interpolation algorithms performance can be observed in *Fig. 4.14* which shows the amount of time needed for the image classification. *LANC* dominated this criterion and comes out of our tests as the superior over other mentioned methods. If one considers a certain amount of trade-off between the method's success rate and the computational cost, the resizing the image using (*LANC*, $f = 3$) seems to be a reasonable choice. The classification speed is decreased to little less than a half of the speed obtained by (*LANC*, $f = 2$), while maintaining a relatively good success rate figures – *98.971 %* (*LANC*, $f = 3$) compared to *99.045%* (*LANC*, $f = 2$).

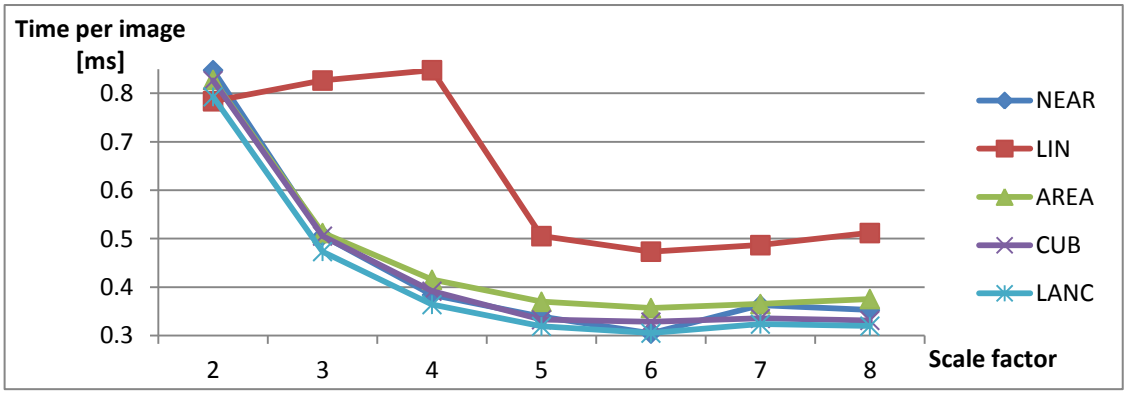


Fig. 4.14 The classification time for different scale factors and interpolation algorithms

4.4.3 Image thresholding

Image thresholding (binarization) is another image preprocessing method which has the potential for the input feature vector improvement. The binary image obtained after applying a thresholding method might result in a more generalized "character template", which would leave out a not necessarily needed "gray" shades of a 8 bit input image. However, it might have just the opposite influence resulting in the loss of the important local pixel neighborhood features. The grid-search using the basic thresholding method and a local (adaptive)

thresholding method was conducted. Threshold value $T(x, y)$ is a mean of $[block\ size \times block\ size]$ neighborhood of (x, y) using MEAN method and a weighted sum applying the Gaussian filter of $[block\ size \times block\ size]$ neighborhood.

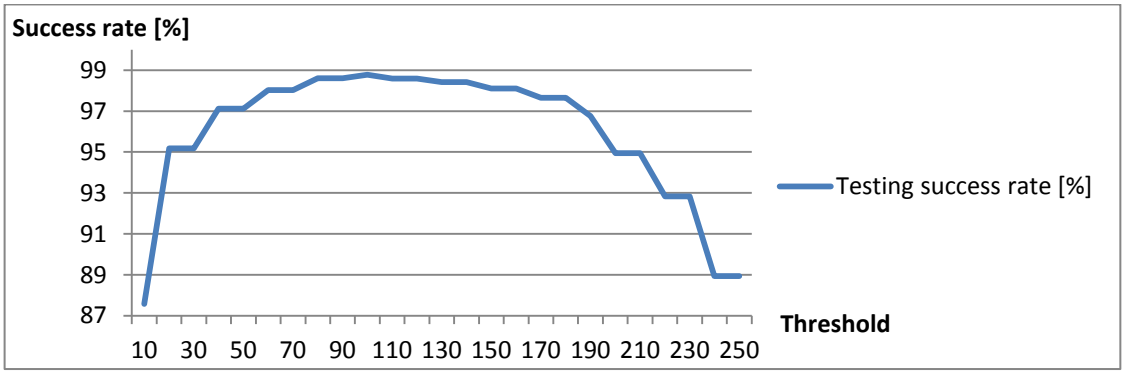


Fig. 4.15 Testing success rate of global thresholding methods using different threshold values

As shown in *Fig. 4.15*, the global thresholding method's success rates appear to be rather inferior to a basic image's raw pixel representation descriptor. Another problem could be caused by the fact that a threshold value is determined during or prior to the SVM training, which could cause discrepancy while classifying the images obtained from another source having different illumination properties.

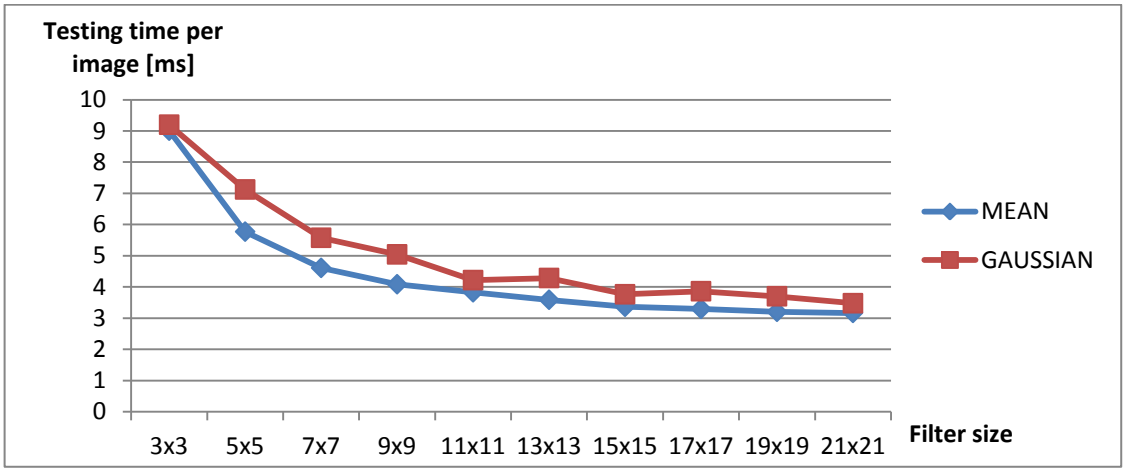


Fig. 4.16 Testing time per image [ms] of local thresholding methods using different filter sizes

On the other hand, the SVM models utilizing the adaptive thresholding, and especially in the case of the middle sized filter blocks, yield competitive success rate figures. The continuously decreasing computational speed while increasing the size of a filter, as shown in *Fig. 4.16*, is caused by the decrease in the number of the support vectors generated for these models, which should guarantee the model's improved generalization capability. It does even outweigh a somewhat higher computational demand for the thresholding method's increased filter size.

4.4.4 Principal Component Analysis method

The PCA is used to find a special basis for a set of input feature vectors. It is comprised of a set of the eigenvectors of the covariance matrix computed from the input set of vectors. After PCA is performed, vectors can be transformed from the original high-dimensional space to the subspace formed by a few most "important" eigenvectors (called the *principal components*), corresponding to the largest eigenvalues of the covariation matrix. This causes the dimensionality of an input vector and the correlation between the coordinates to shrink.

Since the actual PCA is done during the training phase, the overall computational cost required for the classification of the unseen samples was not heavily affected (only the input feature vector's projection to the eigenvector subspace needs to be performed). The grid-search was conducted using the following principal component numbers obtained by the PCA (the output number of principal components is the method's input parameter): 10, 25, 100, 250, 500, 750, 1000, 1250 and 1500. Both original grayscale and resized images (scale factors 2, 3, and 4 using LANC interpolation algorithm) were tested.

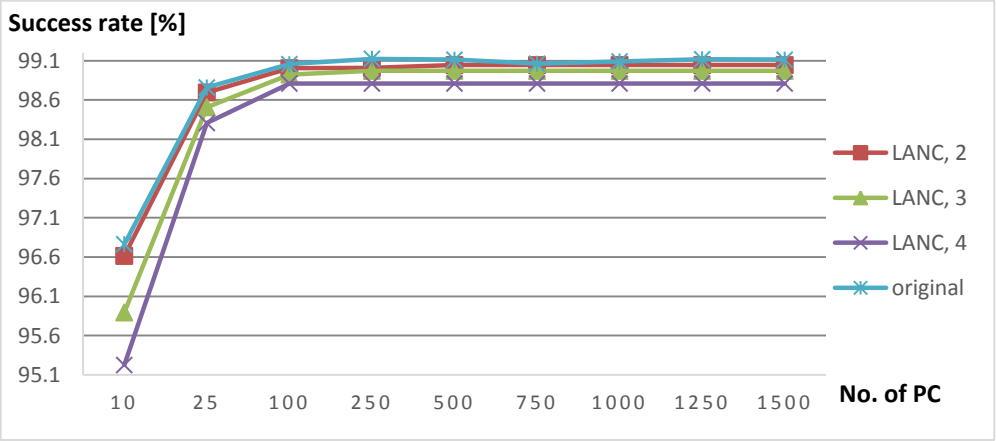


Fig. 4.17 Testing PCA success rates using different scaling options and principal component numbers

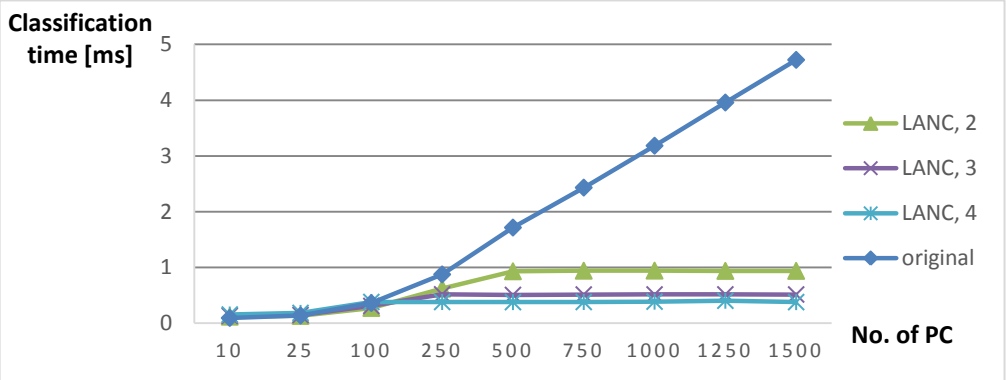


Fig. 4.18 Testing times per image [ms] of PCA using different scaling options

As shown in *Fig. 4.17* and *Fig. 4.18*, the model trained using 100 principal components seem to be a turning point. While the computation time increases linearly with the number of

principal component features selected, the success rate stabilizes and maintains suitably high level. The classification success rate using (*LANC*; 2) yields very similar figures to the Conclusion

4.4.5 Conclusion

The aim of any image preprocessing routine prior to the SVM training and testing is to reduce the number of the input features and thus decrease the computational time while maintaining a certain level of success rate. Since there are usually as many as six or seven characters on a standard license plate, the SVM character classification method should only require as little time as possible.

The number and the "quality" of the input features directly determined the model's complexity. However, there is no "golden rule"; even a small number of feature (but of very poor generalization properties) can guarantee a much higher output number of support vectors.

4.5 MULTI-CLASS ALGORITHM

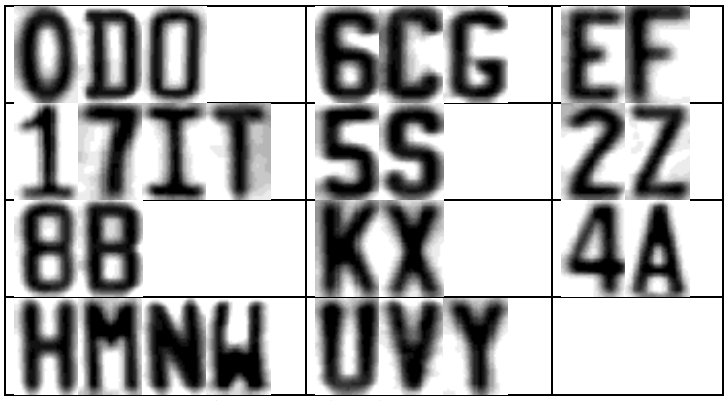
LIBSVM library utilized throughout the Thesis for testing purposes employs the pairwise coupling multi-class. Thus, each of the $K(K-1)/2$ pairwise binary SVMs must be evaluated for each new sample in the classification process. The results are then put into vote and a class with highest number of votes is assigned to the sample. However, there are some shortcoming of such approach. Even the authors of LIBSVM themselves admit the method's incompleteness in [36]: *"In case that two classes have identical votes, though it may not be a good strategy, now we simply choose the class appearing first in the array of storing class names."*

4.5.1 Proposed Enhanced DDAG character classification method

Basic fundamentals of Decision Directed Acyclic Graph SVM method (DDAG) together with other commonly used multi-class algorithm can be found in *Section* 2.2.3. We propose an enhanced DDAG multi-class algorithm which can further improve both the DDAG's reliability and decrease its error rate. The DDAGSVM's unreliability arises from the nature of the progressing through the binary SVMs in a directed graph. Once a particular binary SVM (node in a graph) determines the preliminary datapoint's class membership the wrong way (a binary SVM (O-9) returns '9' as the class membership for the 'O' character), there is just no mechanism to bring back the possibility of the datapoint to be interpreted as 'O' again, going forward down the "opposite" side of the "tree" (see *Fig. 2.14*).

In order to reduce the possibility of the datapoint's incorrect classification, we introduce an additional phase, which takes place after the path (a sequence of $K-1$ binary SVMs being evaluated while finally reaching a leaf) in a graph is found. A preliminary result of a classification is indicated by the class assigned to a sample upon entering a leaf node. This result is usually a correct prediction of the datapoint's class membership.

However, as we have discovered, after the some training and testing experiments we conducted on various SVM models, the false result rate could be further decreased by adding some additional binary SVM evaluations, which would either confirm or reject the assigned prediction value. We identified the possible similar character groups as shown in *Tab. 4.2*.



Tab. 4.2 Similar character groups

The proposed method algorithm is outlined in Fig. 4.19. The training phase is the same as in the standard one-vs-one multi-class approach. A directed acyclic graph is constructed using $K(K-1)/2$ binary SVMs as the nodes (*Fig. 2.14*). The leaf nodes indicate the predicted class. Once a preliminary result is assigned to a sample in a particular leaf node, it progresses forward to either the specific *character group test* or directly to the *general character test*.

Each character group test consists of maximum of six binary SVMs (potential maximum number of pairwise combinations of 4 different characters within the same group). These SVMs are evaluated and votes are added to each character in the group. The preliminary result is rejected if any of the binary SVMs assigns a negative vote to it. Additionally, it can be rejected if any other character from this group outperforms the preliminary result assignment (see the following chapter).

The general character test consists of ten additional SVM evaluations. Ten different opponents are randomly chosen for the preliminary result. It has to win all ten "battles" in order to be accepted.

4.5.1.1 Improving the SVM error rate

The conventional SVM models do not provide any confidence factor or output probability estimate. Although some novel approaches designed to deal with probability estimation and

the SVM's reject option were proposed in [50], [51] and [52], we decided to implement a simple (thus adding almost no additional computational requirements) and yet powerful method. It is based on evaluation of the distance between a datapoint and the separating hyperplane. We must note that our approach is of a rather experimental nature and has not yet been validated on other classification problems. However, it has proved to be a reliable method, as the results obtained during our tests suggest.

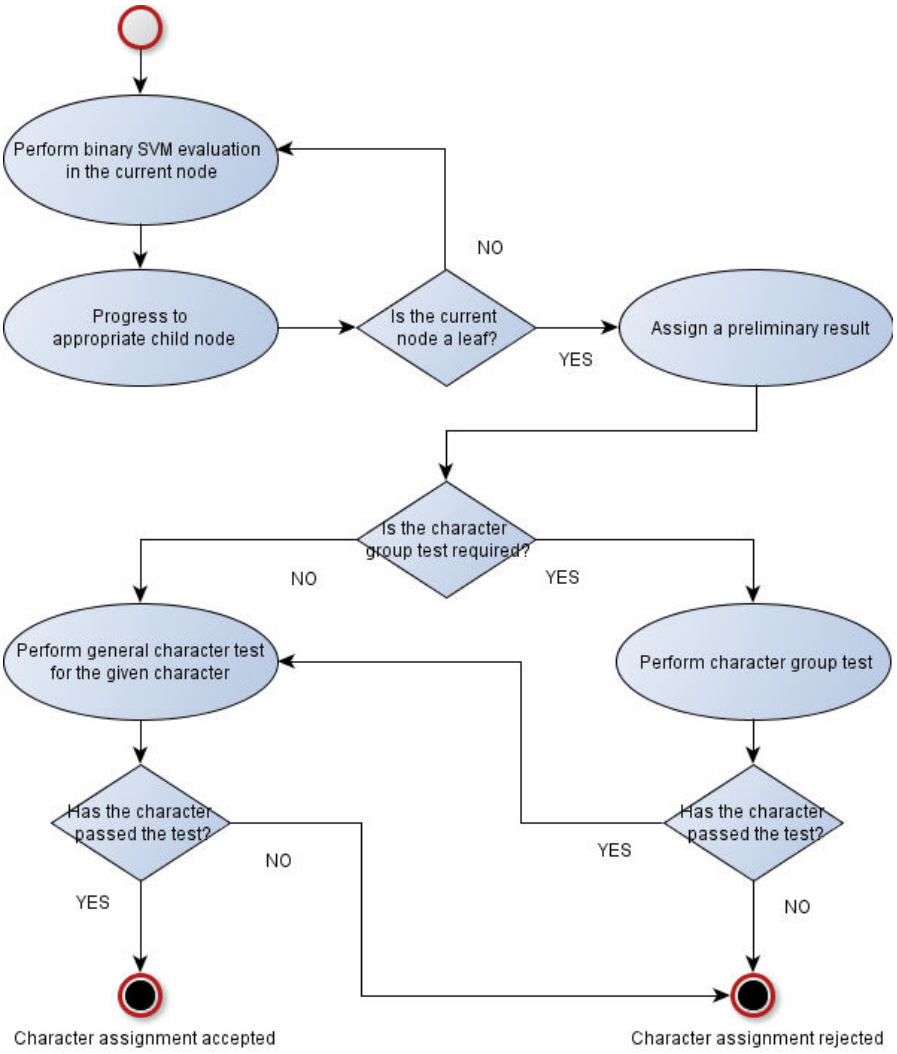


Fig. 4.19 Proposed enhanced DDAGSVM multi-class method

The *character rejection* denotes a possible classification output scenario, in which no class was assigned to the testing sample. It is due to the uncertainty of the pattern's possible membership, which is determined during the *character group test* or the *general character test*. The error rate of any conventional (multi-class) SVM can be computed as a complement to its success rate. Thus, 99.1% success rate figure is 0.9% error rate de facto (sum of all false positives). The aim of our method is to decrease the error rate of the SVM model while maintaining certain level of success rate. The number of the rejected samples cannot be disproportionately higher causing the decrease in the overall success rate.

The undisputed advantage of the proposed method is in its ability to decrease the overall error rate of the SVM while keeping the success rate fairly high (see *Fig. 4.22* and *Fig. 4.23*). The rejection rate shows a tradeoff between the number of misclassified and positively classified images. Another advantage of the proposed method is the reduction of the binary SVMs which are evaluated when compared to common one-vs-one algorithms. The maximum number of binary SVM classifications for the proposed method is $K - 1 + c$, where c denotes the highest possible number of additional tests (*character group test* or the *general character test*). Maximum number of the binary SVM classifications for the dataset consisting of the 35 characters (10 digits and 25 letters) is then $35 - 1 + (6 + 10) = \mathbf{50}$. It is more than eleven-fold decrease in the number of binary classifications needed compared to regular one-vs-one approach, which is $K(K-1)/2 = 35(35-1)/2 = \mathbf{595}$.

4.6 TRAINING AND TESTING DATASET

An additional test was conducted utilizing the proposed DDAGSVM method. A set consisting of 500 **non-character images** was added to the testing dataset. It included some trimmed characters, blobs of various shapes and sizes (retrieved from actual license plates), or other non-character patterns (see *Fig. 4.20*).

Fig. 4.20 Examples of non-character input images

The proposed method was able to reject approximately 35 % of all non-character samples (0.97 % of "non-characters rejected" as in *Fig. 4.21*). Thus, the success rate did not decrease significantly even after a new (unseen prior to the actual classification) class was added to the testing dataset. This leaves a rather positive outlook for the method's proper (false positives) rejection ability.

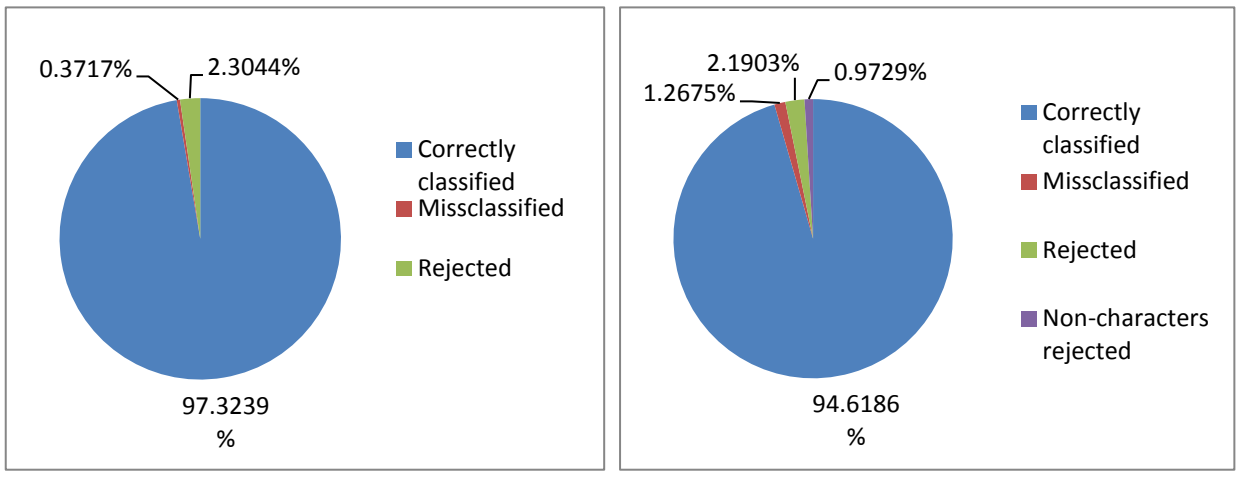


Fig. 4.21 DDAGSVM's performance prior to and after a non-character group was added

An additional test adjusting the **training dataset size** was also performed. The dataset (used throughout the Thesis) was split into two separate groups once more. This time, the training dataset's size and testing dataset's were switched. Thus, we ended up with a fixed dataset consisting of the 200 images (testing) and 500 images (training) per class. Starting at only 50 input images used for model training, additional images were attached up to the whole set of 500 images was used.

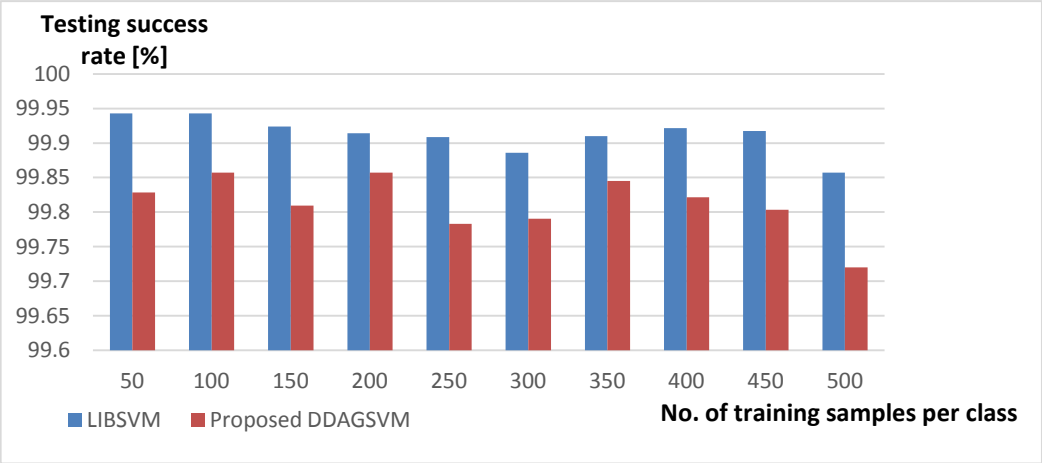


Fig. 4.22 The comparison of the success rates: LIBSVM and proposed enhanced DDAGSVM

As we can observe in *Fig. 4.22*, the classification success rates alone are in favor of the conventional LIBSVM's one-vs-one multi-class method's implementation. However, the error rate figures (*Fig. 4.23*) proves the error rate minimization qualities of the proposed DDAG multi-class method. The models trained in this experiment yielded up to a two-fold decrease in the error rate.

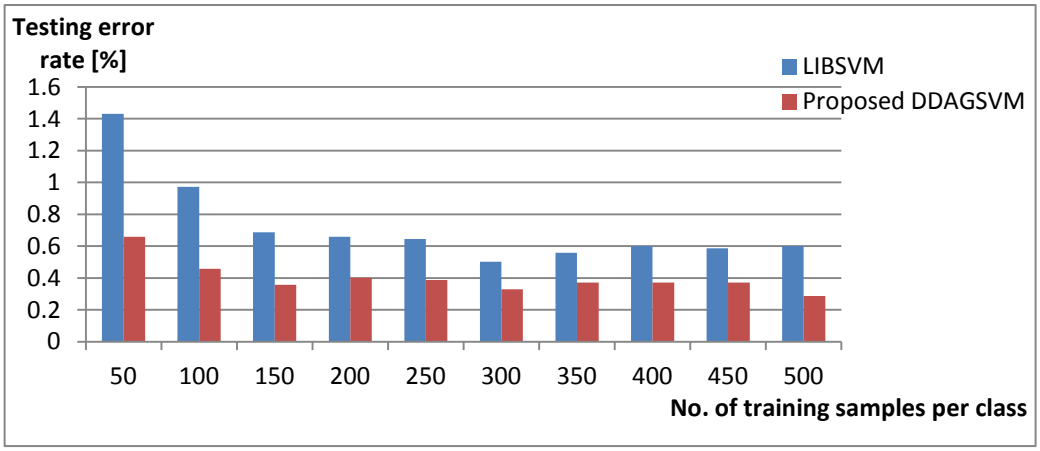


Fig. 4.23 The comparison of the error rates: LIBSVM and proposed enhanced DDAGSVM

5 CONCLUSION

The aim of the Thesis was to make an assessment of the Support vector machine method and SVM-related algorithms for pattern classification problems. We propose the license plate character classification method utilizing the SVM method and outline a novel multi-class SVM approach. The classification method was tested using the dataset comprised of 17,500 LP character images.

The SVM hyperparameters and especially the regularization parameter *C* has got a profound effect on the overall SVM method success rate, however, as outlined in *Section 4.2*, both SVM and kernel parameters are highly domain-dependent as well. Four kernels (linear, polynomial, sigmoidal, and radial basis function) were tested using grid-search technique for suitable parameter value identification. The tests proved the viability of the both linear and RBF kernel for large-scale classification problems (thousands of the input data with hundreds of input feature vectors).

The choice of the image preprocessing method can improve the classification time consumption to a great extent. Since the LP character classification method is required to run under real-time conditions, employing a suiting preprocessing method is extremely important. We outline a novel approach in the image feature description. The proposed image feature descriptor utilizes the vertical/horizontal image histogram projections on the overlapping regions within the image grid. Our approach together with the image scaling and Principal component analysis proved to be competitive image preprocessing (description) techniques. By using one these techniques, a trained SVM is capable of classifying the LP images in less than one sixth of time needed while utilizing the basic approach (input image's pixel intensities as the feature vector values).

The overall SVM model's complexity can be reduced by selecting a proper training set of images. After a proper image preprocessing and a feature description method is performed and the SVM is trained, one must choose a somewhat optimal model from a set of the output SVM models. As suggested multiple times in *Section 4*, the "optimal" SVM models are highly domain- and data-dependent. The models with less output support vectors proved to yield better results both in terms of testing success rates and classification time consumption.

Since the original Support vector machine algorithm is a binary (two-class) classification method in its nature, one must solve an additional multi-class problem. Throughout the thesis, we employed a conventional one-vs-one SVM multiclass approach. In *Section 4.5.1*, we proposed the Enhanced Directed acyclic graph SVM multi-class method. The main advantages of our approach are the rejection ability of the SVM and the classification time

reduction. The SVM commonly does not support a reject option, making the decision making based on the SVM output very unreliable (95 % success rate is 5 % error rate de facto). We propose algorithm which utilizes the character similarities analysis. One can assign a certain confidence factor to the given character by evaluating the performance of the pairwise binary SVMs within the similar character group. The rejection option caused the classification error rate to decrease up to one half (for some models) compared to the conventional one-vs-one approach (see *Section 4.6*).

5.1 OUTLOOK FOR THE FUTURE

The author's future work in the field of the Support vector machine will be certainly focused on the SVM multi-class method improvements. Since most of the classification problems require the multi-class approach, the SVM multi-class method's should not only be "present" as another extra post-processing phase, but rather shall be treated as an integral part of the SVM method. Decreasing the SVM's classification error rate is another important task worth studying. It becomes a major issue especially when deploying the SVM in a real-world software system.

REFERENCES

1. **SOLOMON C., BRECKON, T.** *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab.* Oxford : John Wiley & Sons, 2011. 978-0-470-84472-4.

2. **VÁCLAVIK, D.** *United Arab Emirate license type classification, Bachelor thesis.* Žilina : University of Žilina, 2012. .

3. **YOUNG, I. T., GERBRANDS, J. J., VLIET VAN, L. J.** *Fundamentals of Image Processing, Delft University of Technology.* Delft : Delft University of Technology, 1998. 907-5-691-0-17.

4. **SHIH, F. Y.** *Image processing and pattern recognition: fundamentals and techniques.* Hoboken : IEEE, 2010. 978-0-470-40461-4.

5. **SŇAHNIČAN, O.** *Segmentation and classification of moving objects in scenes with a fixed camera, Diploma thesis.* Žilina : University of Žilina, 2011. [Slovak only].

6. **ANAGNOSTOPOULOS, C. E., ANAGNOSTOPOULOS, I. E., ANAGNOSTOPOULOS, I. D., LOUMOS, V., KAYAFAS, E.** *License plate recognition from still images and video sequences: A survey.* IEEE Transactions on Intelligent Transportation System : vol. 9, no. 3, pp. 377-391, 2008.

7. **DASHTBAN, M. H., DASHTBAN, Z., BEVRANI, H.** *A Novel Approach for Vehicle License Plate Localization and Recognition.* International Journal of Computer Applications : vol. 26, no. 11, pp. 22-30, 2011.

8. **DALAL, N., TRIGGS, B.** *Histograms of Oriented Gradients for Human Detection.* French National Institute for Research in Computer Science and Control : Montbonnot, 2005.

9. **ZISSERMANN, A.** *The SVM classifier.* University of Oxford : The course on machine learning, 2014.

10. **MAHJOUB, A. M., GHANMY, N., JAYECH, K., MILED, I.** *Multiple models of Bayesian networks applied to offline recognition of Arabic handwritten city names.* Sousse : Research Unit SAGE.

11. **TEBELSKIS, J.** *Speech Recognition using Neural Networks, Diploma thesis.* Pittsburgh : Carnegie Mellon University, 1995.

12. **SIBI, P., ALLWYN JONES, S., SIDDARTH, P.** *Analysis of Different Activation Functions Using Back Propagation Neural Networks.* Journal of Theoretical and Applied Information Technology : vol. 47, no. 3, pp. 1264-1268, 2013. ISSN: 1992-8645.

13. **GERSHENSON, C.** *Artificial Neural Networks for Beginners.* Brighton : University of Sussex, 2003.

14. **HINTON, G. E.** *Connectionist Learning Procedures.* Toronto : Elsevier Science Publishers, 1989.

15. **BEN-GAL, I.** *Encyclopedia of Statistics in Quality & Reliability.* : Wiley & Sons, 2007.

16. **GUGGENBERGER, A.** *Another Introduction to Support Vector Machines.*
17. **FRADKIN, D, MUCHNIK, I.** *Support Vector Machines for Classification.* : DIMACS Series in Discrete Mathematics and Theoretical Computer Science, .
18. **SCHOLKOPF, B., SMOLA, A. J.** *Learning with Kernels.* Cambridge, Mass. : MIT Press, 2000. 978-0262194754.
19. **IVANCIUC, O.** *Applications of Support Vector Machines in Chemistry.* Galveston, Tex. : University of Texas, 2007.
20. **FLETCHER, T.** *Support Vector Machines Explained.* London : London's Global University, 2009.
21. **BHARADWAJ, A.** *Support Vector Machines.* New Delhi : Indian Agricultural Statistics Research Institute.
22. **HSU, C-W., CHANG, C.-C., LIN, C.-J.** *A Practical Guide to Support Vector Classification.* Taipei : National Taiwan University, 2010.
23. **DUAN, K.-B., KEERTHI, S. S.** *Which Is the Best Empirical Multiclass SVM Method? An Empirical Study.* Berlin : Springer-Verlag, 2005.
24. **PLATT, J. C., CHRISTIANINI N., SHAWE-TAYLOR, J.** *Large Margin DAGs for Multiclass Classification.* : MIT Press, 2000.
25. **HSU, C.-W., LIN, C.-J.** *A Comparison of Methods for Multi-class Support Vector Machines.* Taipei : National Taiwan University.
26. **ABE, S.** *Support Vector Machines for Pattern Recognition.* London : Springer, 2010. 978-1-84996-097-7.
27. **IVANCIUC, O.** SVM - Support Vector Machines. [Online] [Cited: 04 28, 2014.] http://www.support-vector-machines.org/SVM_soft.html.
28. **SEWELL, M.** SVM Software. [Online] [Cited: 04 28, 2014.] http://www.svms.org/software.html.
29. **CHRISTIANINI, N., SHAWE-TAYLOR, J.** Pointers to Support Vector Machine and Gaussian Processes Software. [Online] [Cited: 04 28, 2014.] http://www.support-vector.net/software.html.
30. **SCHOLKOPF, B.** Kernel Machines Software. [Online] [Cited: 04 28, 2014.] http://www.kernel-machines.org/software.
31. The Shogun Machine Learning Toolbox. [Online] [Cited: 04 28, 2014.] http://www.shogun-toolbox.org/page/features/.
32. Shark machine learning library. [Online] [Cited: 04 28, 2014.] http://image.diku.dk/shark/sphinx_pages/build/html/index.html.
33. dlib C++ Library. [Online] [Cited: 04 28, 2014.] http://dlib.net/.
34. Spider. [Online] [Cited: 04 28, 2014.] http://people.kyb.tuebingen.mpg.de/spider/.
35. scikit-learn Homepage. [Online] [Cited: 04 28, 2014.] http://scikit-learn.org/stable/.
36. **CHANG, C.-C., LIN C.-J.** *LIBSVM: A Library for Support Vector Machines.* Taipei : National Taiwan University, 2013.

37. **CHANG, C.-C., LIN, C.-J.** LIBSVM -- A Library for Support Vector Machines. [Online] [Cited: 04 28, 2014.] http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
38. **JOACHIMS, T.** *Making Large-Scale SVM Learning Practical.* Advances in Kernel Methods - Support Vector Learning : MIT Press, 1999.
39. —. SVM-Light Support Vector Machines. [Online] [Cited: 04 28, 2014.] http://svmlight.joachims.org/.
40. **COLLOBERT, R., BENGIO, S.** *SVMTorch: Support vector Machines for Large-Scale Regression Problems.* vol 1, pp. 143-160 : 2001, Journal of Machine Learning Research.
41. —. A Support Vector Machine for Large-Scale Regression and Classification Problems. [Online] [Cited: 04 28, 2014.] http://bengio.abracadoudou.com/SVMTorch.html.
42. **RUPING, S.** mySVM - a support vector machine. [Online] [Cited: 04 28, 2014.] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html.
43. TinySVM: Support Vector Machines. [Online] [Cited: 04 28, 2014.] http://chasen.org/~taku/software/TinySVM/.
44. **PLATT, J. C.** *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines.* : Microsoft Research, 1998.
45. OpenCV 2.4.0 Documentation. [Online] [Cited: 04 28, 2014.] http://opencv.itseez.com/index.html.
46. **CHAPELLE. O., VAPNIK, V., BOUSQUETS, O., MUKHERJEE, S.** *Choosing Multiple Parameters for Support Vector Machines.* Machine Learning : is. 46, pp 131-159, 2002.
47. **CHERKASSKY, V., MA, Y.** *Practical Selection of SVM Parameters and Noise Estimation for SVM Regression.* Minneapolis : University of Minnesota.
48. **KECMAN, V.** *Learning and Soft Computing.* Cambridge, Mass. : MIT Press, 2001.
49. **LIN, H.-T. , LIN, C.-J.** *A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods.* Taipei : National Taiwan University.
50. **WEGKAMP, M., YUAN, M.** *Support Vector Machines with a Reject Option.* Ithaca : Cornell University, 2012.
51. **LI, M., SETHI, I. K.** *SVM-Based Classifier Design with Controlled Confidence.* Rochester : oakland University.
52. **MA, C., RANDOPLH, M. A., DRISH, J.** *A Support Vector Machines-based Rejection Technique For Speech Recognition.* Schaumburg : Human Interface Laboratory Motorola Labs.