Self-Tuning One-Class Support Vector Machines for Data Classification

by

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

Jun 2014

St. John's Newfoundland

Abstract

Support Vector Machine (SVM) based classifiers are most popular models for data classification in machine learning. To obtain high classification accuracy, parameter tuning methods such as cross-validation are often applied, which is however time-consuming. To address this problem, a simple, efficient and parameter-free algorithm is presented in this thesis. The algorithm is especially useful when dealing with datasets in the presence of label noise. Grown out of one-class SVM, the presented algorithm enjoys several distinct features: First, its decision boundary is learned based on both positive and negative examples, whereas the original one-class SVM training is only based on positive examples; Second, the internal parameters are self-tuned, which makes the algorithm handy to use even for first-time users. Compared with the benchmark method LIBSVM, the presented algorithm achieves comparable accuracy, while consuming only a fraction of the processing time. Applications in computer vision are presented to demonstrate the effectiveness of the algorithm.

Acknowledgements

First, I feel extremely lucky to have had the opportunity to study with my advisor Prof. Minglun Gong. It has been a great pleasure for me to work with him on several projects, including this thesis. I would like to thank him for his guidance, inspiration and support during my graduate studies. Prof. Gong strengthened my love for computer vision and machine learning, and shared his research experience with me. I have learned a lot about research methodology from him, as well as basic principles of being a better researcher. I would also appreciate his financial support for my life at Memorial.

Second, I would like to thank Dr. Yung-Yu Chuang, Dr. Antonio Criminisi, Dr. Stan Sclaroff, Dr. Yaser A. Sheikh, Dr. Seth Teller, Dr. Xue Bai, Dr. Jue Wang, Dr. Yu-Wing Tai, Mr. Inchang Choi, and Mr. Eduardo S. L. Gastal for sharing their datasets or experiment results used in this thesis with us or online.

Third, I would like to thank School of Graduate Studies and Department of Computer Science. Thanks to all staffs who helped me in the two years. I am also grateful to my friends at St. John's - Shibai Yin, Hao Yuan, Rose Reid and Yunhai Wang - for making my master study enjoyable. Without them, life at Memorial would be a lot more boring.

Last, I would like to thank my family for their love and support, especially my parents Chunyong Cai and Ping Qian, and my wife Huihong Niu. This thesis is dedicated to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Data classification is a fundamental problem in machine learning and has been extensively studied (e.g. [38], [53]). The problem is to identify which of a set of categories a new observed example belongs to, based on a training set of data containing examples whose category membership is known. A data example is represented using a feature vector which depicts the quantifiable properties of the example. Different from testing data, a training example contains an extra class label. An algorithm for data classification is known as a classifier, which maps testing data to a class.

Data classification can be thought of different problems across different terminologies. In the term of the class number, classification can be divided into two separate problems - binary classification and multiclass classification. In binary classification, only two classes are involved, whereas multiclass classification contains more than two classes. Many efficient classification methods have been proposed for binary and multiclass classification (e.g. [15], [20], [49], [40], [16], [52], [25]). In some methods, multiclass classification is solved by combining multiple binary classifiers (e.g. [49],

[40]).

In the term of machine learning, classification can be considered as an instance of supervised learning, i.e. analyzing the completely labeled training data and producing an inferred classifier, which can be used for mapping testing examples. An opposite situation is the problem of unsupervised learning, where the examples given to the learner are unlabeled and the learner is trying to find hidden structure in unlabeled data. It is also known as clustering in machine learning. Semi-supervised learning falls between supervised learning and unsupervised learning, where labeled data and unlabeled data are both used for training - typically a small amount of labeled data with a large amount of unlabeled data. The labeling process for generating training data is often expensive and time-consuming, requiring skilled human agents or physical experiments, whereas obtaining unlabeled data is relatively inexpensive and convenient. In such situations, semi-supervised learning is very useful and practical and researchers in machine learning also found that unlabled data can greatly help improve learning accuracy in some cases. Data classification may be solved in a semi-supervised learning manner [10], where the classifier is trained with both labeled and unlabeled data.

The algorithm presented in this thesis touch both supervised learning and semi-supervised learning. First, the algorithm is applied to solve supervised learning based multiclass classification in Chapter 4, where training examples are completely labeled. Second, a computer vision application is developed in Chapter 5, which follows semi-supervised learning. The algorithm is used for foreground segmentation and boundary matting for live videos. Initially, sparse strokes are provided by users to indicate the two classes: foreground and background. Then, the classifiers trained based on the

strokes are used to label unlabeled neighboring pixels. These unlabeled pixels become labeled ones, which is used for training with the labeled pixels in the next iterations. Hence, the final labeling uses information of both labeled and unlabeled pixels.

The most widely used classifiers for data classification includes: support vector machines, naive bayes classifier, multi-layer neural network, random decision trees, etc. This thesis is based on support vector machine (SVM) classifier. SVM was first introduced by Cortes and Vapnik [15] and has been a very popular and powerful method for both regression and classification. One of the important reasons that SVMs can become a standard tool for data classification is that it has high generalization performance without requiring priori knowledge in the particular fields. SVMs have delivered state-of-art performance in real-world applications such as text recognition, image classification, bioinformatics, etc [18].

Data classification problems exit in many areas, such as speech recognition, internet search engines, computer vision, hand-written character recognition, etc. A simple instance in computer vision is face detection, which is a binary classification problem. The classifier is trained using labeled face and non-face images and then is used to predict whether a newly observed image contains human faces.

1.1 One-class Support Vector Machine for Classification

One-Class Support Vector Machine (often referred as 1SVM or OSVM or OCSVM) is firstly introduced by Schölkopf et al. through extending the SVM methodology

[54], and is applied to novelty detection [55]. Suppose we are given one class of data, we would like to predict whether a newly observed data is a member of the class. To cope with the problem, One-class SVM is used to model the distributions of data in the class. If a testing data is too different, according to some measurement, from the 1SVM model, it is labeled as novelty. The problem is also known as one-class classification problem in machine learning (The training data is from one class). 1SVM has been successfully applied in many areas such as document classification [45], data density estimation [46], recommendation tasks [62], etc. More background about 1SVM can be found in Section 2.2 and 2.3.

Besides handling one-class classification problems, 1SVM model achieves high performance on binary/multiclass classification problems. The key idea is to train and maintain multiple 1SVMs which model training data distributions from different classes. Each 1SVM may label a testing example as inlier or outlier, and then these 1SVM models competitively determine the label of the example. This thesis applies the same idea to solve binary and multiclass classification problems. Compared with the SVM model specially designed for data classification, modeling different classes separately using multiple 1SVMs produces decison boundaries that enclose training examples more tightly in some cases. One example is the presented computer vision application, as illustrated in Figure 5.1.

Similar to other methods designed for data classification, 1SVM algorithm includes many tuning parameters, such as parameters in kernel functions, that are application-dependent and are often non-intuitive to machine learning practitioners. This issue is particularly pronounced with large-scale applications [26], where even moderate amount of tuning parameters might be computationally too expensive. Besides, hu-

man annotations tend to be error-prone especially when working with problems of large-scale and many classes [23]. This motivate us to consider a novel data classification approach that is parameter-free, efficient, and capable of dealing with noisy data. To achieve this, a Self-Tuning One-Class SVM algorithm (or STOCS) is developed, which should be able to automatically determine training parameters and perform well on noisy training data. The key insight of our approach is to train each 1SVM model using both positive (data from the same class with the 1SVM) and negative (data from other classes) examples. This allows us to adaptively choose the optimal parameter settings for different datasets. In contrast, the conventional methods train each 1SVM using positive examples only.

1.2 Contributions

In this thesis, we present a novel 1SVM model to solve data classification problems, which is applicable to binary and multiclass classification. We also demonstrate the ability of handling different level label noise and working on large-scale and dynamic data processing. In summary, the presented algorithm bears the following characteristics:

Parameter self-tuning: Our approach is parameter-free, which is handy to use even for first-time users. Based on the conventional online 1SVM learning framework introduced in Chapter 2, all parameters in our training algorithm are successfully removed or adaptively tuned step by step in Chapter 3. In addition, we use positive and negative examples for self-tuning parameters, resulting in that the decision boundary of our approach is learned from both positive and negative examples. In

comparison, the conventional 1SVM training is only based on positive examples. By enjoying these distinct features, our approach is shown to performs almost as well as optimal parameter settings tuned for individual datasets of the benchmark method, while consuming only a fraction of training time.

Robustness to label noise: Most real-world datasets used in supervised machine learning application are labeled by human annotations or physical experiments, where some labels may be corrupted by some reasons. Training with noisy labels may produce many potential negative consequences [23]: the accuracy of predictions may decrease, the complexity of training models and the training time may increase. Hence, designing data classification systems that can handle datasets with noisy labels is a problem of practical importance. In Section 3.4, by allowing different support vectors with different cut-off values, our approach adaptively limits the effects of noisy data, which can both simplify the final model (decrease support vector numbers) and reduce the effects on prediction accuracy. In contrast, the conventional 1SVM training cannot balance the two objectives well by applying a constant cut-off value for all support vectors. A lower cut-off value may over-limit the effects of other examples, producing more support vectors, whereas a larger cut-off cannot limit the effects of outliers, decreasing classification accuracy. Our approach can not only improve classification accuracy, but also makes the performance of classier more stable with increasing label noises.

Real-time computer vision application: Different from the original SVM and 1SVM learning which both use batch learning, our approach utilizes online learning, which allow us process dynamic and large-scale datasets. In Chapter 5, we demonstrate this feature with the developed application on dynamic video data processing.

The algorithm is used for developing a real-time computer vision application: integrated foreground segmentation and boundary matting for live videos. By utilizing the proposed STOCS training model, the thesis develops a unified framework for foreground segmentation and boundary matting for live videos. Foreground segmentation is a binary classification problem: classify a pixel as foreground or background, and boundary matting aims to get the opacity along foreground boundary. The ability of our proposed online 1SVM to train separate classifiers for foreground and background not only allows more robust labeling of pixels, but also facilitates the matting operation along object boundaries. This leads to an integrated solution for both foreground segmentation and boundary matting problems. Compared to state-of-art methods, our approach performs competitively under a variety of challenge scenarios such as fuzzy object boundaries, camera motion, topology changes and low fore/background color contrast as shown in Figure 5.4. This is usually achieved with minimal user interactions: users are only asked to annotate foreground and background of the first frame with few key strokes. The sparse strokes and unlabeled pixels are both used in our final training model under a semi-supervised learning manner. Furthermore, by designing independent execution at individual pixel locations, our implementation utilizes the graphics processing unit (GPU) for parallel computing, achieving real-time processing speed for VGA-sized videos.

In summary, the thesis is solving the problem of data classification, including both binary and multiclass cases, using a simple, efficient, and parameter-free online one-class SVM approach. The rest of the thesis is organized as follows. Background and related works are reviewed in Chapter 2. Chapter 3 discusses the details of our self-tuning one-class SVM training algorithm. Evaluations on multiclass classification and

computer vision applications are provided in Chapter 4 and 5, respectively. Finally, Chapter 6 concludes the thesis and suggests future research directions.

Chapter 2

Background and Related Work

Before presenting the main content of this thesis, we provide some background on SVM training and some related work. This chapter begins with a review of conventional SVM and 1SVM training framework. The online learning model used in this thesis is then described, followed by an introduction to batch and online learning. Finally, the chapter gives some related work on parameter tuning methods and label noise handling.

2.1 Support Vector Machine

This section introduces the basic idea and mathematical model of SVM training, by taking binary classification as an example. As shown in Figure 2.1 [1], suppose we have a set of data $\Omega = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ is the i_{th} $d-$dimensional training data point ($d = 2$ in Figure 2.1), $y_i \in \{-1, 1\}$ is the label of x_i. To split the two classes, we may have many available linear classifiers shown

[1] http://www.autonlab.org/tutorials/svm15.pdf

in Figure 2.1(a). SVM selects the one with the maximum margin in Figure 2.1(b), where the margin of a linear classifier is defined as the width that the boundary could be increased by before hitting a data point. The classifier boundary is known as hyperplane. In Figure 2.1(b), we have two hyperplanes (the boundary of the yellow part), and they are mathematically represented as: plus-plane= $\{w \cdot x + b = +1\}$, minus-plane= $\{w \cdot x + b = -1\}$, with $w \in \mathbb{R}^2$ in Figure 2.1 and $b \in R$. Then we can get the margin of the linear classifier as $M = 2/\sqrt{w \cdot w}$. Since SVM is looking for the classifier with the maximal margin, the problem may be converted to the following optimization problem:

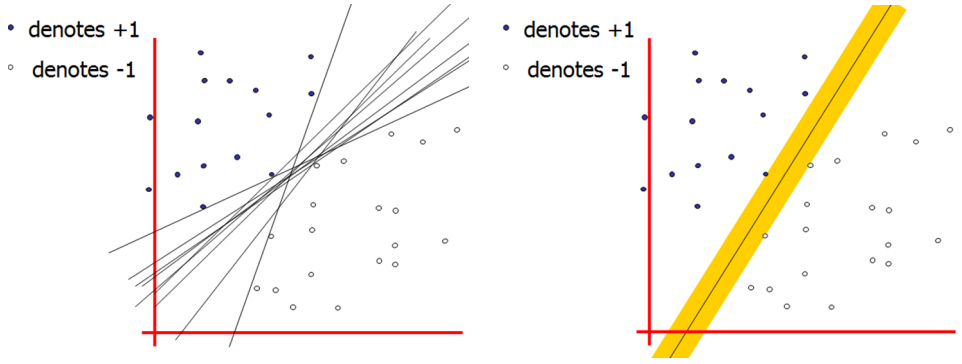
$$\begin{aligned} &\min \qquad \frac{||w||^2}{2} \\ &\text{subject to} \quad y_i(w \cdot x_i + b) \geq 1, i = 1, \cdots n. \end{aligned} \tag{2.1}$$

To better handling noisy data in SVM training, Cortes and Vapnik [15] create the soft margin method if there exists no hyperplane that can split the two classes. The key idea is that it tries to select hyperplanes that splits data points as cleanly as possible, while minimizing distance of error points to their correct class. To achieve this, the new method introduces slack variables ξ_i, which measure the degree of misclassification of x_i. The new objective function becomes:

$$\begin{aligned} &\min \qquad \frac{||w||^2}{2} + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \cdots, n. \end{aligned} \tag{2.2}$$

SVM can also create a nonlinear decision boundary when data points are not linearly separable as shown in Figure 2.2 [2]. By projecting data points through a nonlinear operator φ from their original space $\mathbb{R}^2$ to a new feature space F, data

[2] http://cdn.intechopen.com/pdfs-wm/11772.pdf



(a) binary dataset example (b) SVM solution

Figure 2.1: Given two classes of data points shown in (a), there are many possible linear classifiers. SVM training tries to find the one with the maximum margin (the one with the maximum width of the yellow part in (b)).

points which cannot be split by a liner line in $\mathbb{R}^2$ are separable by hyperplanes in F. Then, the objective function for nonlinear case can be written as:

$$\begin{aligned} \min \quad & \frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w^T\varphi(x_i)+b) \geq 1-\xi_i, \xi_i \geq 0, i = 1, \cdots, n. \end{aligned} \tag{2.3}$$

Equation 2.3 can be solved by Lagrange multipliers, and the output decision function rule for a testing data point x becomes:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n}\omega_i y_i \varphi(x)^T\varphi(x_i) + b\right), \tag{2.4}$$

where $\omega_i \geq 0$ are the Lagrange multipliers. The decision function is *supported* by the data points with $\omega_i > 0$. We call a data point with $\omega_i > 0$ *support vector*, hence, we have the name *Support Vector Machine*. In Figure 2.1(b), the points hitting the hyperplanes are support vectors. Notice that the number of support vectors is much smaller than the number of data points. According to Equation 2.4, the results of

the decision function only replies on the dot product of x and x_i. Hence, it is not necessary to define such a φ to explicitly project data points to the space F, as long as we can have a function $k(x, x_i) = \varphi(x)^T \varphi(x_i)$. The method of defining such a K is known as kernel trick, and k is known as kernel functions. Some common kernels include:

- Linear kernel: $k(x_i, x_j) = x_i \cdot x_j$
- Polynomial kernel: $k(x_i, x_j) = (x_i \cdot x_j + c)^d, c > 0$
- Gaussian radial basis function: $k(x_i, x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2), \gamma > 0$

Finally, the decision function of SVM becomes:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \omega_i y_i k(x, x_i) + b \right). \tag{2.5}$$

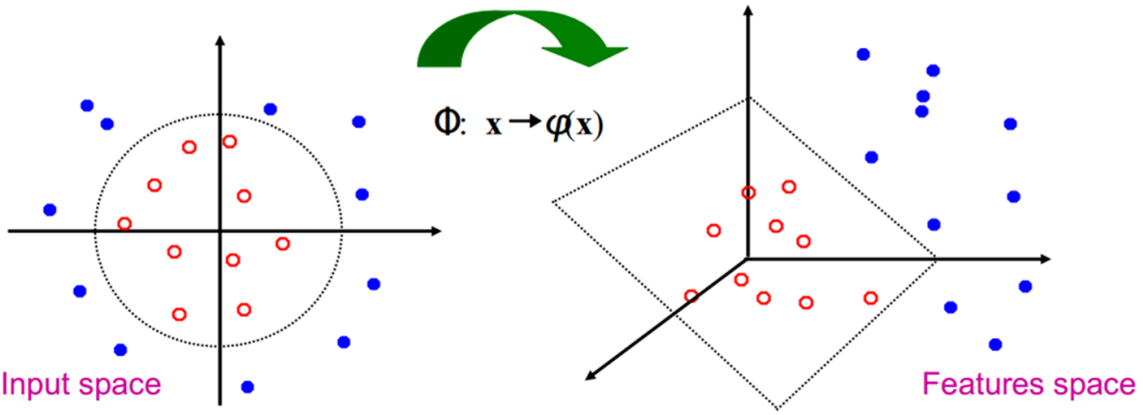


Figure 2.2: After applying a nonlinear function φ, data points which is not linearly inseparable in their original input space can be split by a linear hyperplane in feature space.

2.2 One-Class SVM

Given one class of data points, Schölkopf et al. [55] extend the above SVM algorithm to solve 1SVM classification problem. The key idea is that they assume the origin is another one class (The class contains only one point). The modified algorithm is trying to separate all data points from the origin in the feature space F and maximizes the distance from the hyperplane to the origin. The objective function is similar to the one of SVM algorithm, and details about it are omitted here since this thesis is not based on the 1SVM model in [55].

Our training method is similar to the one proposed by Tax and Duin [59]. They propose to use a spherical, instead of planar, to surround the one-class data points. Points inside the hypersphere are inliers and points outside the hypersphere are outliers. Hence, we need to minimize the volume of the hypersphere while at the same time include as many inliers as possible. Denote the radius and the center of the hypersphere as r and a, respectively. The minimization function can be changed from Equation 2.2 to:

$$\begin{aligned} \min_{r,a} \quad & \frac{||r||^2}{2} + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & \|x_i - a\|^2 \leq r^2 + \xi_i, \xi_i \geq 0, i = 1, \cdots, n, \end{aligned} \tag{2.6}$$

where C is the penalty parameter. After solving it by using Lagrange multipliers ω_i, we can compute the distance of a testing data point x to the center as:

$$\|x - a\|^2 = (x \cdot x) - 2\sum_{i}^{n}\omega_i(x \cdot x_i) + \sum_{i,j}^{n}\omega_i\omega_j(x_i \cdot x_j). \tag{2.7}$$

A testing data x is considered in-class when $\|x - a\|^2 \leq r^2$. By applying the idea of kernel trick (replacing dot products with kernel functions), we often define a score

function as:

$$f(x) = \sum_{i}^{n} \omega_i k(x, x_i) \geq \frac{1}{2}\left(k(x,x) + \sum_{i,j}^{n} \omega_i \omega_j k(x_i, x_j) - r^2\right), \tag{2.8}$$

to determine whether x is an inlier. When Gaussian kernel is used, the score function becomes $f(x) = \sum_{i=1}^{n} \omega_i \exp(-\gamma \left\|x - x_i\right\|^2) \geq C_r - r^2/2$, where C_r depends only on support vectors x_i and not on testing point x. Notice that the thesis is solving binary/multiclass classification problems, so our classification criteria is that a testing data belongs to the class with the maximum score.

2.3 Online One-Class SVM

Training a SVM or 1SVM model using a set of examples is a classical batch learning problem, the solution of which is obtained through minimizing a objective function such as Equation 2.3 and 2.6. Before introducing the online 1SVM model used in this thesis, we review the concept of batch and online learning.

2.3.1 Scaling up: Batch vs. Online Learning

In batch learning, examples for training are inputted to batch learner at the same time, whereas in online learning, examples are observed by the learner one by one in a time sequence. When an new example is observed, the online learner is updated according to its current model and the new example. The conventional SVM and 1SVM models are both based on batch learning, because all training examples are shown to a quadratic objective functions at the same time, where the solutions of which are found through minimizing the objective functions. Previous studies [5]

have shown that a similar or even better generalization performance can be achieved using online learning with a much less computational cost, by showing all examples repetitively to an online learner, when comparing to that of batch learning. The presented self-tuning algorithm is based an online one-class SVM model [30], which is presented in the next subsection.

2.3.2 Online Learning Model

The online learner we use follows the one proposed by [12] and [11]. Let $f_t(\cdot)$ be a score function of examples at time t, $k(\cdot,\cdot)$ be a kernel function, and ω_t be a non-negative weight of example of time t. When a new example x_t arrives, the score function becomes:

$$f_t(x_t) = \sum_{i=1}^{t-1} \omega_i k(x_i, x_t), \tag{2.9}$$

which is similar to Equation 2.8, and the update rule for weights is:

$$\begin{aligned} \omega_t &= \text{clamp}\left(\tfrac{\gamma-(1-\tau)f_t(x_t)}{k(x_t,x_t)}, 0, (1-\tau)\chi\right), \\ \omega_i &\leftarrow (1-\tau)\omega_i \qquad \forall i = 1,\ldots,t-1, \end{aligned} \tag{2.10}$$

where $\gamma := 1$ is the margin, $\tau \in (0,1)$ the decay parameter, and $\chi > 0$ the cut-off value (cut-off is used to handle noisy training data, which is similar to ξ_i in Equation 2.2 and 2.6). $\text{clamp}(\cdot, A, B)$ is an identical function of the first argument bounded by A and B. Intuitively, the underlying idea is that when a new example is observed, we first compute its score function using the current support vectors from the same class. Then we use the score function to update weights of support vectors. If the score is large enough, it means that the new example can be predicted well using the current model and it is not necessary to incorporate the new example as a support vector,

otherwise the new example should be added into the corresponding support vector list with a certain weight to make the learner support the new example. For each class, we train and maintain one online 1SVM model, i.e. store one support vector list. Training examples are repetitively shown to the learner in a time sequence, and training process converges until there are no changes for all support vector lists. Notice that a support vector is the example whose weight is greater than zero, so it can be represented as (x_i, ω_i).

Compared to the conventional SVM and 1SVM algorithms, the online 1SVM that we follow can work on large-scale data, where we only need to compute score functions and update weights at each iteration. The operations during training are very simple and it does not involve any complex data structures. Moreover, the online learner can handle dynamic data such as video data. When a new example (or a new frame in video data) is observed, the objective functions in the conventional SVM and 1SVM are both changed, and they should be solved again to obtain the new solution. If we are processing a large-scale dataset, solving a minimization problem is quite time-consuming. However, in online 1SVM model, we only need to compute the score function of the new example and update weights, which is very simple and efficient. Cheng et al. [11] have shown the online 1SVM works well with large-scale and dynamic data.

So far, the thesis has introduced the online learning model that is used in our approach for data classification. Before presenting our efficient and parameter-free approach, we review some related work.

2.4 Related Work

The thesis is solving data classification using an online 1SVM model. Related work on online learning is reviewed. We are mainly improving the training model on parameter tuning and the ability of working on noisy training data. Hence, previous research done on these two topics are also presented in this section.

2.4.1 Batch vs. Online Learning

Batch learning has been the standard methods for data classification such as [20], [52], [16], [25], etc. When employing batch learning for large-scale datasets, one often has to fight with a number of bottlenecks such as memory issue and computational costs. One notable exception is [4], where a scalable batch learning methods is proposed. Nevertheless, it requires complex speeding-up techniques such as disk swapping and chunking, which unfortunately introduce quite a few tuning parameters. This is in sharp contrast to online learning methods, such as [42] and [11], that are usually very simple and efficient.

2.4.2 Parameter Tuning

Parameter tuning is also known as estimation of internal parameters or adaptive bandwidth. Cross-validation [17], [32], [57], is probably the most widely used method for estimating the internal parameters. [37] empirically compares cross-validation with bootstrap [32] and finds that the latter one tends to introduce extremely large bias sometimes, while the former often performs significantly better. A typical cross-validation strategy is to perform coarse grid search over the space of internal pa-

rameters, which is however often time-consuming. [34] proposes efficient algebraic methods aiming at its speedup. In [43], the initial cross-validated parameter values are further refined by coordinate descent on induced objective functions. Despite of its popularity and usefulness, cross-validation also possesses several major issues and limitations, as discussed in [61] and [50]. Empirical results in [50] show that the final models for large-scale problems selected by cross-validation may easily become over-fitting and the performance on testing data may would be worse than expected. Cross-validation on large-scale problems is also time-consuming. Further, when training data contains label noises, noisy data would be used to validate models, causing the selected models being overfit to the noisy data and leading to poor classification accuracy. In contrast, by utilizing online learning and adaptive cut-off parameter setting, our approach can handle both large-scale training issue and label noises well. More details on our approach can be found in the following chapters.

Meanwhile other methods have also been studies: From the view of Bayesian evidence maximization, [28] instead considers a hybrid Monte Carlo approach based on the evidence gradients. [65] presents a adaptive Lasso method to adjust coefficient shrinkage for individual variables for regression related problems. The method of [58] is based on variable selection stability and is dedicated to problems involving penalized regression models.

The idea of adaptive or variable bandwidth has also been studied for density estimation [36], regression [33], and classification [19] problems. Existing works usually focus on being adaptive in term of *only* locality and is agnostic to different class labels, while the variable bandwidth considered in our approach is sensitive to its location, as well as adversary classes from its spatial vicinities. Furthermore, unlike existing

approaches that try to pick the same set of parameters for all support vectors, we allow different support vectors having different parameters. Our parameter tuning method is based on the following two observations: 1) allowing support vectors that are far away from the decision boundaries having larger influence area can effectively reduce the number of support vectors needed; and 2) assigning small influence areas to support vectors that are close to the decision boundaries help to reduce the level of confusion. Details on that are discussed in Chapter 3.

2.4.3 Label Noise

In real-life applications it is of great importance to make reliable predictions even in the present of noisy labels (known as Label Noise in machine learning). The problem has been studied by numerous research efforts, starting from the early works [1], [64] with theoretical analysis [7]. A kernel based Linear Discriminant Analysis (LDA) method is considered in [39], while [21] focuses on exploiting the positive instances. Following that of [7], various noise-resistant variants of the perceptron method are also proposed [35], [8]. Very recently, there are several attempts (e.g. [56], [47]) to provide a more rigorous account of the theoretical understanding and analysis of this problem. Interested readers may refer to review articles [48], [23] for further details.

Our approach tries to address the label noise problem through adaptively selecting cut-off parameters for individual support vectors. We notice that outliers are often sparsely distributed and surrounded by correctly labeled examples. By assigning low cut-off values to the support vectors that correspond to these outliers, we can effectively limit their impact to the final decision. Please note that our sparse

distribution assumption on outliers is different from the optimization based sparse learning strategy such as the L_1-norm methods in [44] and [63] that are designed for dealing with label noises. First, our approach utilize online learning, which does not involve optimization; Second, the sparse learning strategy such as the one in [44] utilizes the sparsity in the weight vector of the final decision induced by the L_1-norm optimization, which is different from our spare distribution assumptions of outliers. By converting a data classification problem to a L_1-norm optimization problem, the sparse learning approaches result in a sparse solution which is not sensitive to label noise and are thus able to deal with noises to some extent. More details about our label noise handling method can be found in Section 3.4.

In summary, we have reviewed the basic ideas in data classification using SVM and one-class SVM. Then, we present online 1SVM model. Compared to batch SVM and 1SVM model, online 1SVM model has more power to work on large-scale and dynamic data, which is the key reason that we choose online 1SVM as our training model. In the next section, we are aiming to improve online 1SVM model and make it more handy to use for common users.

Chapter 3

Self-Tuning One-Class SVM

One important challenge in one-class SVM training is parameter tuning. It is impossible to generalize a good parameter setting for all datasets from different sources. Over the years there have been significant amount of efforts on this topic. However, those parameter tuning algorithms are generally time-consuming and often sensitive to noise. To cope with the problem, we propose a novel Self-Tuning One-Class SVM (STOCS) for data classification.

Unlike the conventional one-class SVM, which uses only positive examples from a given class to train a model, STOCS makes use of both positive and negative examples, which helps STOCS capture the structure of training data and then improve classification accuracy. According to the updating rules of online 1SVM model in Equation 2.10

$$\begin{aligned} \omega_t &= \text{clamp}\left(\tfrac{\gamma-(1-\tau)f_t(x_t)}{k(x_t,x_t)}, 0, (1-\tau)\chi\right), \\ \omega_i &\leftarrow (1-\tau)\omega_i \qquad \forall i = 1,\ldots,t-1, \end{aligned} \tag{3.1}$$

three parameters are involved in the training process: decay parameter τ, kernel

bandwidth σ, and cut-off value χ. The decay parameter is used to limit the effects of support vectors obtained in the early training stage, which should be gradually reduced throughout the iteration process. Kernel bandwidth controls the value of kernel function, and cut-off is used to limit the effects of outliers. Considering σ and χ are associated with each support vector, STOCS adaptively sets kernel bandwidth σ and cut-off value χ for each support vector *individually*. Notice that original 1SVM algorithm sets them as the same constants for all support vectors. Hence, at each support vector, we not only store its weight ω, but also the corresponding parameters, forming a quadruple $(x, \omega, \sigma, \chi)$. In the following section, before presenting the parameter self-tuning method of STOCS, we discuss the requirement of kernel functions used in STOCS.

3.1 Adjustable Kernel Functions

In this section, we provide guidelines and requirements for selecting kernels for STOCS.

The kernel function $k(u, v)$ used in Equation 2.9 computes the dot product of two high-dimensional vectors to which examples u and v are mapped. It determines how much one example, if chosen as support vector, will provide support to the other example. A good kernel function should output high values when applied to two similar examples, and low values for dissimilar ones. Here we call a kernel function a *normalized* kernel if it satisfies the following property:

$$\begin{cases} k(x_t, x_t) = 1, & \forall x_t \\ 0 \leq k(x_t, x_s) \leq 1, & \forall x_t, x_s. \end{cases}$$

By definition, the Gaussian kernel, $k(u,v) = \exp(-\|u-v\|^2/\sigma^2)$, is a normalized kernel. When applied to normalized histogram vectors, the histogram intersection kernel, $k(u,v) = \sum_{i=1}^{n} \min(u_i, v_i)$, is also normalized. The linear kernel, $k(u,v) = u \cdot v$, is not normalized in general, but it becomes normalized if the input vectors are both nonnegative and normalized, i.e., $u_i \geq 0$ and and $\|u_i\| = 1$, $\forall u_i$.

We further call a *normalized* kernel $k(u,v,\sigma)$ with parameter σ *adjustable*, if and only if it possesses the following two properties:

i) $\exists\sigma$, we can always satisfy $k(x_t, x_s, \sigma) \leq T, T \in (0,1)$ for all $x_t, x_s, x_t \neq x_s$;

ii) if $k(x_t, x_s) \geq k(x_t, x_l)$, then $k(x_t, x_s, \sigma) \geq k(x_t, x_l, \sigma)$ holds regardless σ value.

Gaussian kernel is adjustable since setting $\sigma = \sqrt{-\|x_t - x_s\|^2 / \log T}$ would satisfy the first requirement and adjusting σ does not alter the overall shape of the Gaussian either. For any normalized kernel function $k(u,v)$ that is not inherently adjustable, we can define an adjustable version as:

$$k(u,v,\sigma) = \max\left(1 - (1-T)\frac{1-k(u,v)}{1-\sigma}, 0\right) \tag{3.2}$$

As shown in Figure 3.1, for a given σ, $k(u,v,\sigma)$ is a monotonically-increasing piecewise-linear function with respect to $k(u,v)$. By definition, setting $\sigma = k(x_t, x_s)$ allows $k(x_t, x_s, \sigma) \leq T$, regardless how the original kernel function $k(u,v)$ is defined. Furthermore, since $k(u,v,\sigma)$ is a monotonic function, it does not change the score ordering among examples defined by $k(u,v,\sigma)$.

In conclusion, STOCS requires kernel functions normalized. Making a kernel adjustable also allows us to control the support of an example can provide on others. We evaluate both Gaussian and linear kernels in our experiments. Notice that linear kernel doe not have an explicit σ, but we can use Equation 3.2 to make it adjustable.

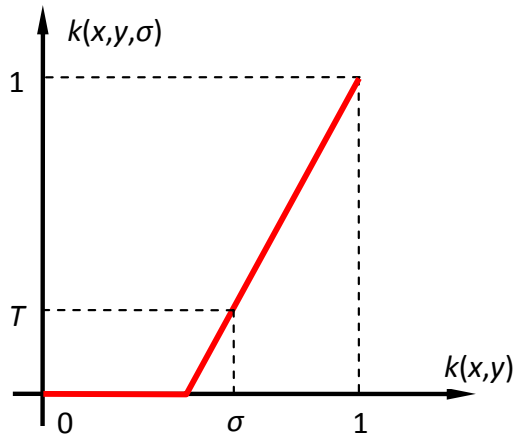


Figure 3.1: Define an adjustable kernel $k(u, v, \sigma)$ based on a normalized kernel $k(u, v)$.

3.2 Removal of Decay Parameter τ

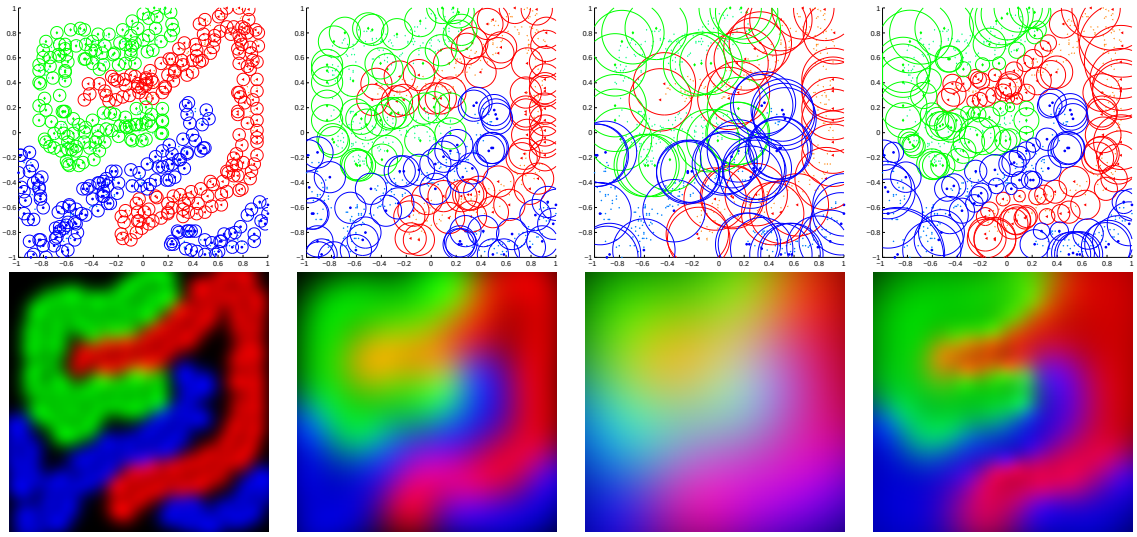
When using the online learning model presented in Section 2.3.2, the parameter τ needs to be carefully adjusted throughout the iterative process. This is because at the beginning of the training, the active set is empty and the score function of any input data will be zero. Consequently, the first group of support vectors added into the active set tend to have large ω values, which needs to be lowered to their proper values in the later iteration. As shown in Equation 2.10, once a support vector (x_t, ω_t) is added to the active set, over the time its weight ω_t is only affected by the decay parameter τ. Hence, the initial value for τ needs to be large to effectively reduce the weights of existing support vectors. On the other hand, the iterative training process cannot converge unless $\tau \approx 0$. As a result, τ needs to be gradually reduced throughout the iterative process. Conventionally, the decay parameter is set based on an exponential function: $\tau = \exp(-\frac{t}{\xi})$, where parameter ξ controls how fast the decay parameter τ decreases.

In STOCS, τ is eliminated through an explicitly *reweighting* scheme. That is, if a training example x_t arrives and it turns out identical to the example in an existing support vector (x_i, ω_i), this support vector is taken out before computing the score function and then replaced with (x_t, ω_t) that carries the newly obtained weight. Also considering that we always have $k(x_t, x_t) = 1$ for normalized kernels, the new score function and the update rule become:

$$f_t(x_t) = \sum_{i=1}^{t-1} \omega_i \delta(x_i \neq x_t) k(x_i, x_t, \sigma_i),$$
$$\omega_t = \text{clamp}\left(\gamma - f_t(x_t), 0, \chi_t\right), \tag{3.3}$$

where $\delta(\cdot)$ is an indicator function with $\delta(true) = 1$ and $\delta(false) = 0$.

Intuitively, this modified online learning method resets the weight component of a particular support vector (x_t, ω_t), based on how well the separating hyperplane defined by the remaining support vectors is able to classify example x_t. If the score function $f_t(x_t)$ computed based on the remaining support vectors is large, it means that x_t can be supported well by the remaining support vectors. As a result, a small value should be assigned to ω_t, and vice versa. Hence, this reweighting process can either increase or decrease the weight ω_t according to how much the current model can support x_t. Namely, the weights of all support vectors can be automatically and adaptively adjusted during the training process. Hence, decay is not necessary any more.



(a) $\sigma = 0.1$, $\chi = 0.5$, # of SVs: 443

(b) $\sigma = 0.25$, $\chi = 0.5$, # of SVs: 144

(c) $\sigma = 0.5$, $\chi = 0.5$, # of SVs: 91

(d) adaptive σ, $\chi = 0.5$, # of SVs: 180

Figure 3.2: Visualization of the classification results for the "threeclass" dataset. Top row shows the selected support vectors with the corresponding σ values illustrated using circles. Bottom row shows the decision maps, which encode the output scores from the three one-class SVMs using red, green, and blue channels respectively. Both black color in (a-b) and orange/cyan/magenta colors in (b-c) indicate regions with ambiguities.

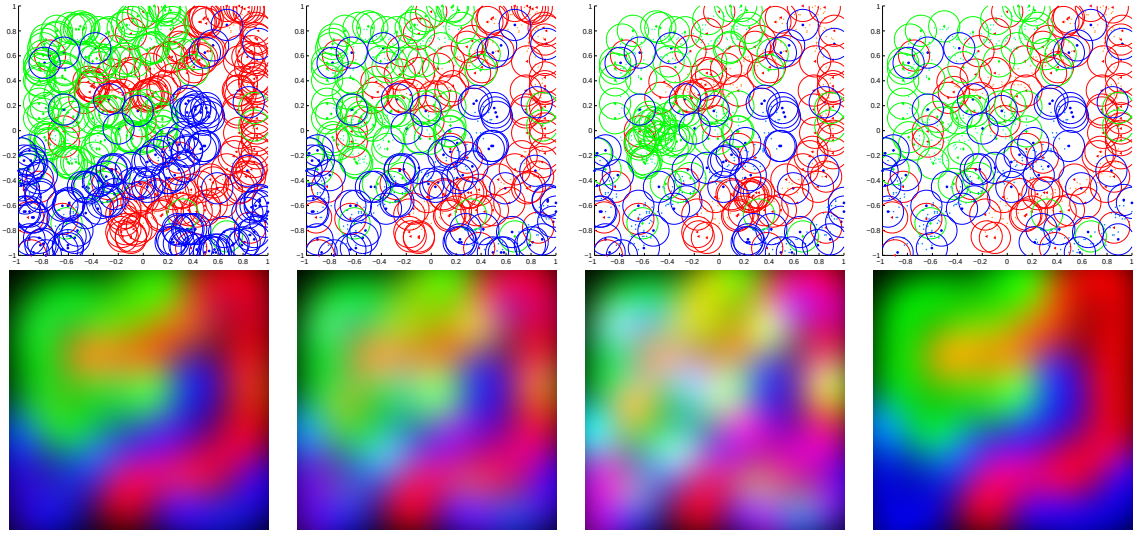
3.3 Adaptive Kernel Bandwidth σ

As mentioned above, the adjustable kernel function used in STOCS contains a parameter σ. We now discuss how to tune σ_i automatically and individually for each support vector $(x_i, \omega_i, \sigma_i, \chi_i)$. The core idea here is that a support vector should not provide strong supports to negative examples, i.e., given a support vector x_i, we require $k(x_i, x_n, \sigma_i) \leq T$ for any negative example x_n, where T is a constant threshold. Using this constraint, we can easily compute σ_i based on the definition

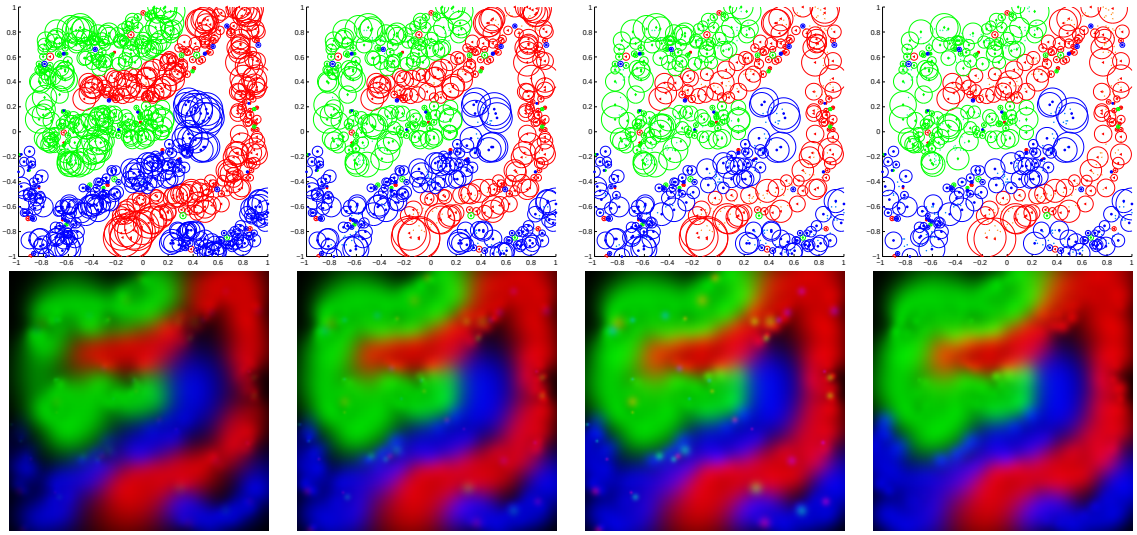
of the adjustable kernel function. That is, as soon as we meet a negative example x_n that yields $k(x_i, x_n, \sigma_i) > T$ during the online learning, we update σ_i to ensure $k(x_i, x_n, \sigma_i) \leq T$. Since adjusting σ does not affect the score ordering among examples, the supports to previously seen negative examples can only decrease, which ensures the training process converges. It is also worth noting that for static datasets, σ_i can be precomputed for each example x_i using the negative example x_n that yields the largest $k(x_i, x_n, \sigma_i)$ value.

Without losing generality, here we use the Gaussian kernel to illustrate the idea. For Gaussian kernel, the kernel bandwidth σ is the standard deviation of the Gaussian function and controls the radius of the influence area of each support vector. As shown in Figure 3.2(a), when σ is too small, the obtained classifier can only recognize data that are close to one of the provided training examples, resulting over-fitting. On the other hand, large σ value causes fuzzy decision boundaries among different classes; see Figure 3.2(c). Hence, to ensure a proper σ value is used, previous approaches often rely on cross validation.

As shown in Figure 3.2(d), through automatically selecting different Gaussian support for different support vectors, the classifier obtained provides a sharp decision boundaries between the three training classes. which also output relatively few support vectors. Demonstration of the effectiveness of self-tuning σ will be presented in Section 4.3.



(a) $\sigma = 0.25$, $\chi = 0.1$, # of SVs: 398 (b) $\sigma = 0.25$, $\chi = 0.25$, # of SVs: 236 (c) $\sigma = 0.25$, $\chi = 0.5$, # of SVs: 205 (d) $\sigma = 0.25$, adaptive χ, # of SVs: 168



(e) adaptive σ, $\chi = 0.1$, # of SVs: 748 (f) adaptive σ, $\chi = 0.25$, # of SVs: 521 (g) adaptive σ, $\chi = 0.5$, # of SVs: 422 (h) adaptive σ, adaptive χ, # of SVs: 385

Figure 3.3: Visualization of the classification results when the dataset is corrupted with label noise, i.e., 5% of the random data have their labels changed. Under both fixed (a-d) and adaptive (e-h) Gaussian support situations, using adaptive cut-off values helps to limit the impacts of outliers.

3.4 Adaptive Cut-off Value χ

The last standing parameter is the cut-off value χ, which is used to limit the effects of label noises (or outliers). When a training example (assuming it is mistakenly labeled) is observed, it may become a support vector with a large weight, yielding strong effect on final decision, resulting in that testing examples close to the outlier would be mislabeled. The cut-off χ is introduced to clamp weights of outliers and then limit their impact to classification. A large cut-off value cannot limit the effects of label noises, however, a small one may influence other correctly labeled support vectors. In STOCS, we adaptively select a cut-off value for each individual support vector.

Following the idea of tuning parameters for individual support vectors, here we also adaptively determine a proper χ value for each support vector. The key idea is that a support vector should have a higher χ if there are many positive examples within its support region and a smaller χ if there are many negative examples surrounding it. Hence we set:

$$\chi_t = 0.5 + \frac{h^+(x_t) - h^-(x_t)}{2(h^+(x_t) + h^-(x_t))} \tag{3.4}$$

where $h^+(x_t)$ and $h^-(x_t)$ are the numbers of positive and negative examples that satisfy $k(x_t, x_i, \sigma_t) \geq 0.6T$, respectively. When $h^+(x_t) > h^-(x_t)$, x_t is surrounded by more positive examples, thus x_t is not an outlier and should have a large cut-off value. When $h^+(x_t) < h^-(x_t)$, x_t is surrounded by more negative examples, thus x_t is possibly an outlier and can have a small cut-off value, which limits the effects of x_t on the decision function. Note that $h^-(x_t) \geq 1$ due to the way σ_t is calculated, ensuring the denominator being a non-zero value. Comparison between Figure 3.2(b)

and Figure 3.3(c), which are generated using the same set of parameters, suggests that the presents of label noises can severely distort the decision boundaries. Using a smaller χ values helps to reduce the impact of outliers, but at the expense of using more support vectors; see Figure 3.3(a-b). The use of adaptive σ also helps to limit the impact of outliers to their own neighborhoods; see Figure 3.3(e-g). Nevertheless, it does not fully address the problem.

As shown in Figure 3.3(d & h), using adaptive χ values can effectively limit the impacts of outliers. Note that, similar to σ, the χ values for different examples can also be precomputed when the dataset is static.

In summary, by assuming each support vector may have individual parameter settings, the three parameters, τ, σ, and χ, are successfully removed or adaptively tuned in online 1SVM training. Different from the original 1SVM training, our online 1SVM model of one class is trained using both positive and negative examples. We also achieve sharper decision boundary when training data contains label noise. In the next chapter, evaluations on our parameter self-tuning method are presented and comparisons with a benchmark method is also given.

Chapter 4

Experiments on Multiclass Classification

Evaluations and experiments on multiclass classification are presented in this chapter to demonstrate the effectiveness of STOCS. We first evaluate the effectiveness of reweighting and parameter self-tuning by comparing STOCS with the conventional online 1SVM training technique [11]. Comparisons with the benchmark approach, the LIBSVM [9], for multiclass classification in terms of classification accuracy, support vector number, and training time, are then reported.

4.1 Datasets

Table 4.1 summarizes information of datasets used in this chapter. All datasets can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. Here we give some details of several popular datasets. The "letter" dataset is created with gray images displaying the 26 capital letters in English alphabet, which is used

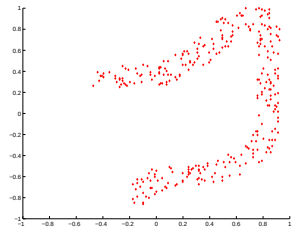
Dataset	Classes	Training Examples	Testing Examples	Features
letter	26	15000	5000	16
usps	10	7291	2007	256
satimage	6	4435	2000	36
pendigits	10	7494	3498	16
vowel	11	528	462	10
svmguide4	6	300	312	10
shuttle	7	43500	14500	9

Table 4.1: Datasets used for multiclass classification

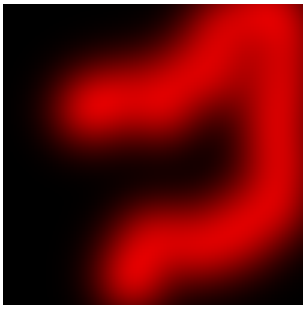
for letter recognition in machine learning. The "usps" and "pendigits" datasets are both created for handwritten digit recognition, which contain 10 numeric characters (from 0 to 9). The "shuttle" dataset contains 7 classes, in which approximately 80% of the data belongs to the first class. So the "shuttle" dataset can be used to demonstrate the effectiveness of classification algorithms when the distribution of training examples is distorted.

4.2 Effectiveness of Reweighting

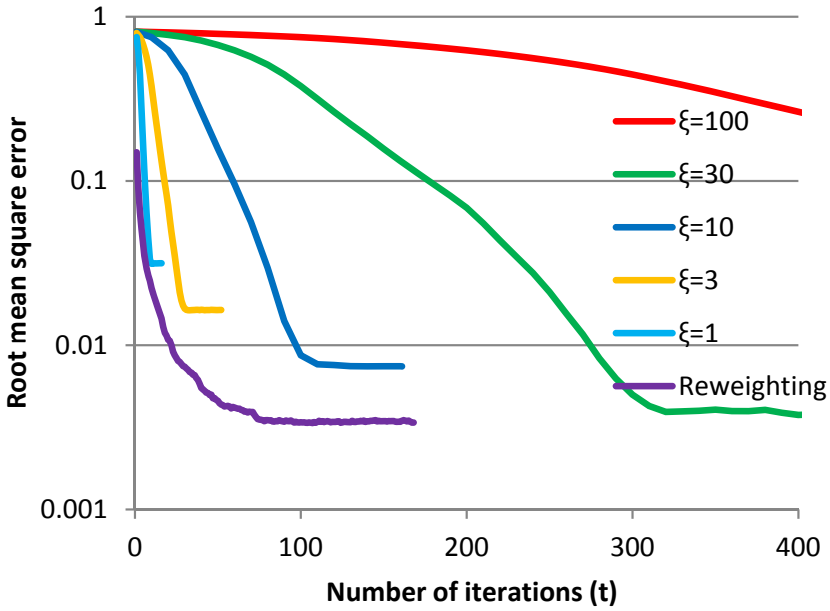
Figure 4.1 compares the convergence speed between the reweighting scheme (decay parameter is removed) and the conventional training approach under different decay rate settings. For the input dataset shown in Figure 4.1(a), we first generate a one-class SVM classifier using Equation (2.9) and (2.10) under fixed parameters ($\sigma = 0.25$; $\chi = 0.5$) and a very slow decay change rate ($\xi = 100$). This classifier is then used



(a) training data



(b) decision map



(c) convergence

Figure 4.1: Convergence of the conventional approach and the reweighting scheme. For a given set of examples (a), the decision map (b) obtained using the conventional approach under a very small decay rate is used as the ground truth. Comparing the decision maps generated after different iterations with the ground truth shows the convergence speeds of different approaches (c).

to generate a decision map (see Figure 4.1(b)), which is used as the ground truth. The decision maps obtained after different iterations and under different settings are compared with the ground truth. The results shown in Figure 4.1(c) suggest that the conventional approach generally converges after 10ξ iterations, i.e., $\tau < \exp(-10)$. Furthermore, when a faster decay change rate is used, the final decision map defers from the ground truth, indicating a premature convergence. The classifier obtained using reweighting with the constant σ and χ setting yields almost identical decision map as Figure 4.1(b), but it converges much faster than the conventional approach

under $\xi = 100$.

4.3 Effectiveness of Parameter Self-tuning

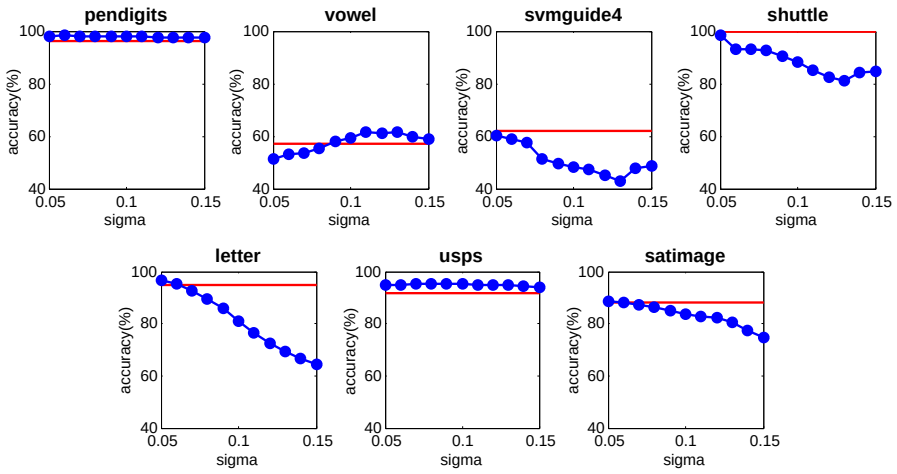


Figure 4.2: The effectiveness of self-tuning σ. Classification accuracy comparisons between self-tuned σ (red) and one-class SVM (blue) trained under different σ values are plotted.

We now evaluate the presented self-tuning scheme by comparing it with the conventional online one-class SVMs trained using fixed parameters. Datasets listed in Table 4.1 are used.

First we test the effectiveness of self-tuning σ using a Gaussian kernel and under a fixed cut-off value ($\chi = 0.5$). Both STOCS and the conventional one-class SVM are trained using the training data and the classification accuracy on test data are measured. To determine the proper σ range for these datasets when using the traditional one-class SVM, we preform a coarse-to-fine grid search. Figure 4.2 plots the classi-

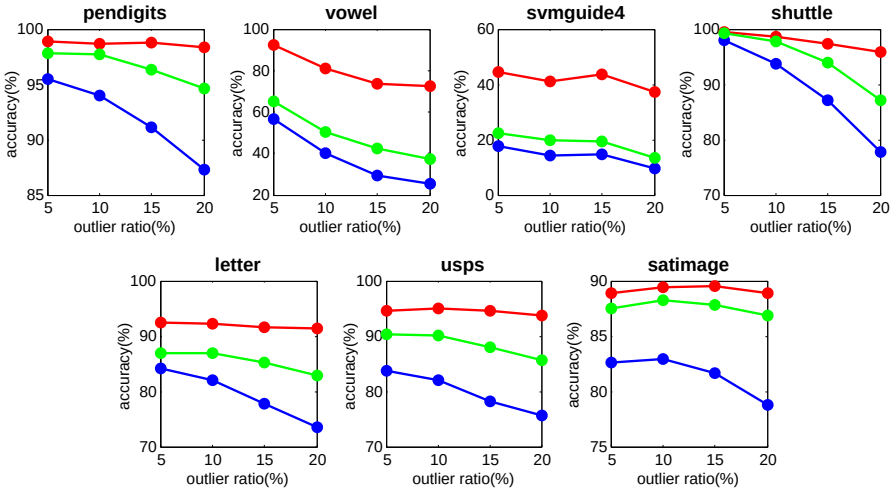


Figure 4.3: The effectiveness of self-tuning χ. Classification accuracy of self-tuned χ (red), fixed $\chi = 0.1$ (green), and fixed $\chi = 0.5$ (blue) under increasing label noise is compared.

fication accuracy of the traditional one-class SVMs when σ varies within the proper range found. As expected, the performance highly depends on the σ value and there is no single σ value that works well for all datasets. STOCS adaptively selects σ for each support vector based on a fixed threshold value $T = 0.25$. Its performance is close to the traditional one-class SVM under optimal settings in all cases, and even outperforms the latter in the "svmguide4" and "shuttle" datasets. We attribute such performance gain to adjusting kernel bandwidth for support vectors individual, rather than using the same tuned parameters for all.

Next, we evaluated the effectiveness of self-tuning χ on datasets corrupted with label noises. For each dataset, we randomly and incrementally select training examples as outliers and alter their labels. These outlier examples with their original labels are then used as testing data to evaluate whether the classifiers can correct

labeling errors. The test is repeated 10 times for each dataset to compute the average accuracy. The results of self-tuning χ and fixed χ are plotted in Figure 4.3, where σ is self-tuned in all tests. The comparison clearly shows that the performance of conventional one-class SVM drops as the more label noise is introduced. Using self-tuned χ values not only improves the accuracy, but also makes the performance of classifier more stable.

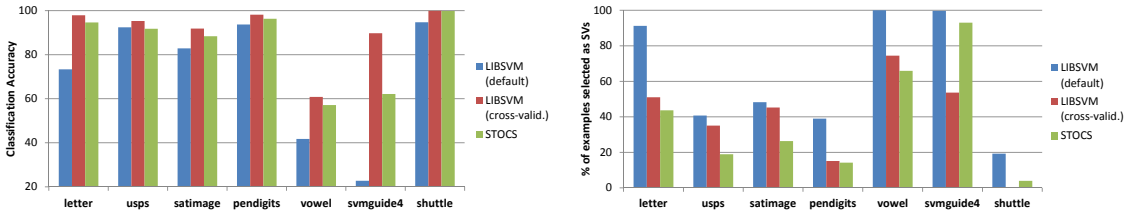
4.4 Comparisons with LIBSVM

Finally we compare the performances of STOCS with a benchmark approach, the LIBSVM [9] using the one-vs-all approach, under both Gaussian and Linear kernels. We note in the passing that it is widely accepted (e.g. [52]) that for multiclass classification, one-vs-all is one of the simplest methods that almost always delivers the best performance. It is thus preferable to more complex methods including output coding schemes [20], [25] or single machine schemes [16]. This observation motivate us to concentrate on the following comparisons with the LIBSVM implementation of the on-vs-all approach, which is arguably the most widely used multiclass classification method in practice.

4.4.1 Gaussian Kernel

We start with comparing the performances of STOCS and LIBSVM on the aforementioned seven datasets under Gaussian kernel. For LIBSVM, both default and cross-validated parameter settings are tested. Note that here LIBSVM is tuned for each dataset individually, resulting different parameter settings for different datasets.

STOCS, on the other hand, uses the same self-tuning procedure with the threshold $T = 0.25$ for all datasets.



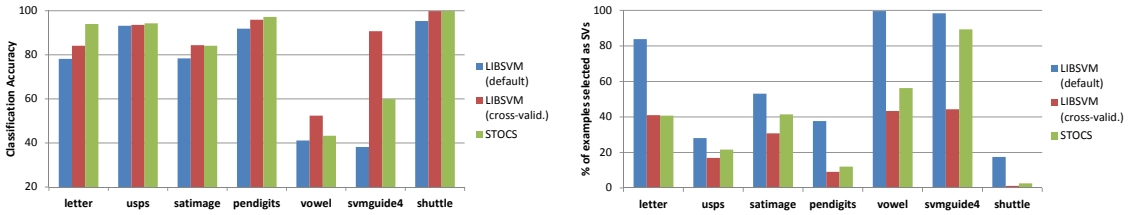
Method	*Process*	letter	usps	satimage	pendigits	vowel	svmguide4	shuttle
LIBSVM(default)	*Training*	17.38	12.97	1.06	1.31	0.09	0.05	27.9
LIBSVM(cross-valid.)	*Tuning*	11390.21	17130.30	1112.91	2101.55	25.42	15.54	29074.35
	Training	11.59	18.61	1.64	1.10	0.08	0.13	**3.97**
STOCS	*Training*	**2.50**	**3.26**	**0.72**	**0.52**	**0.02**	**0.02**	16.02

Figure 4.4: Comparison between STOCS and LIBSVM under Gaussian kernel. Top left shows the classification accuracy, top right shows the number of support vectors used, and bottom row gives the processing time needed.

Figure 4.4 shows that STOCS is more accurate than the LIBSVM under default settings in six out of the seven datasets. While LIBSVM with parameters tuned through cross-validation outperforms STOCS, the performance difference is less than 4% in six datasets. To achieve this performance gain, LIBSVM requires much longer processing time for parameter tuning. Furthermore, for most datasets, STOCS uses fewer support vectors than LIBSVM with both default and tuned parameters, allowing faster labeling of incoming examples.

4.4.2 Linear Kernel

To evaluate whether the presented approach works well for kernel functions that are not inherently adjustable, here we also compare STOCS with LIBSVM under linear



Method	*Process*	letter	usps	satimage	pendigits	vowel	svmguide4	shuttle
LIBSVM(default)	*Training*	9.72	8.13	1.11	1.07	0.13	0.06	16.5
LIBSVM(cross-valid.)	*Tuning*	966.02	379.90	111.15	80.83	1.72	2.87	14081.85
	Training	14.39	**5.51**	**0.92**	0.69	0.18	1.31	**4.62**
STOCS	*Training*	**2.09**	6.65	1.17	**0.41**	**0.02**	**0.01**	9.28

Figure 4.5: Comparison between STOCS and LIBSVM under linear kernel.

kernel. To satisfy the nonnegative and normalized conditions that makes linear kernel normalized, we first map all examples feature-wise to range [0,1] and then normalize individual example vector to unit length. The same normalized datasets are used to train both LIBSVM and STOCS. However, LIBSVM uses the original linear kernel, whereas STOCS uses its adjustable version; see Equation 3.2. The same threshold value $T = 0.25$ is used here for STOCS.

From Figure 4.5 we observe that, not only STOCS outperforms LIBSVM under the default parameter setting in all cases, it also outperforms LIBSVM with tuned parameters for two datasets and performs on par for two other datasets. This is remarkable since LIBSVM uses the best parameters specifically tuned for the given datasets.

4.4.3 Label Noise Handling

When the training data contains label noises, parameter tuning through cross-validation may cause the parameters being overfit to the corrupted data, leading to poor classification accuracy. This is evidenced by Figure 4.6, where the LIBSVM with tuned

parameters is outperformed by the default parameters for "usps" and "satimage" datasets. Our approach, on the other hand, is robust against label noises and performs consistently. It outperforms LIBSVM with tuned parameters in four of the seven datasets.

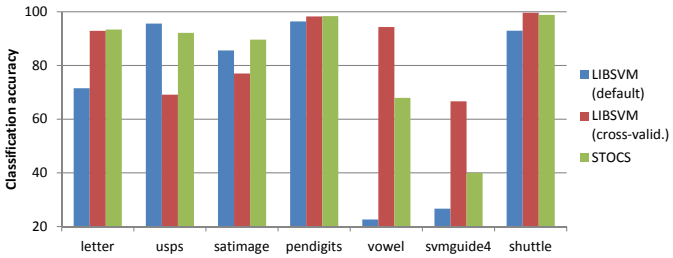


Figure 4.6: Comparison with LIBSVM on training data corrupted by 10% label noises.

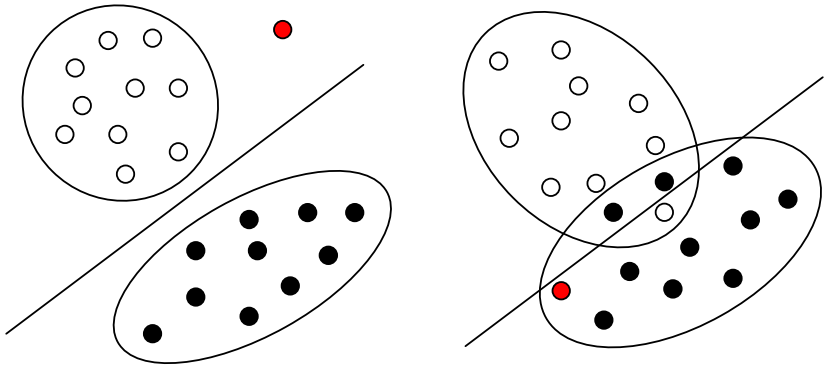
In summary, we have solved multiclass classification problems with STOCS. We evaluate and demonstrate the effectiveness of our proposed parameter self-tuning method step by step. Compared to the benchmark method LIBSVM, STOCS output shorter training time and fewer support vectors, while achieving comparable classification accuracy with LIBSVM's optimal parameter settings tuned for individual datasets. In the next chapter, online 1SVM learning is applied to solve binary classification problems in computer vision to demonstrate its real-world application ability.

Chapter 5

Computer Vision Application

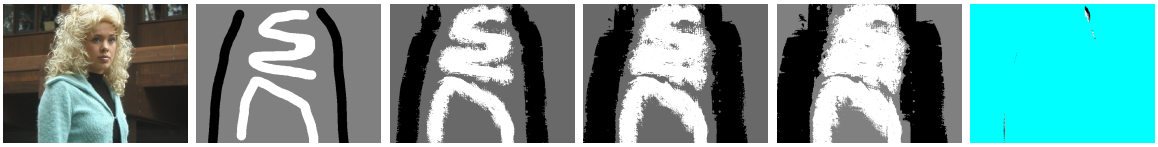
In the last chapter, we demonstrate the usefulness of applying STOCS to solve multiclass classification problem. Here we show an application in computer vision, where STOCS is used to solve foreground segmentation and boundary matting for live videos. Foreground segmentation, a.k.a, video cutout, studies how to extract objects of interest from input videos. It is a fundamental problem in computer vision and often serves as a pre-processing step for other video analysis tasks such as surveillance, teleconferencing, action recognition and retrieval. In the terminology of machine learning, it belongs to binary classification problems. Given a frame of a video, foreground segmentation aims to determine a pixel belongs to the class of foreground object or background. Boundary matting aims for recovering the transparency and corresponding color along foreground objects [60]. It is one of the key techniques in film production applications, especially handling scenarios such as fuzzy object boundaries (e.g. hair) and motion blur. Over the years a significant amount of related techniques have been proposed in both computer vision and machine learning

communities. However, some of them are limited to sequences captured by stationary cameras, while others require significant amount of training examples or cumbersome user interactions. Furthermore, most existing algorithms are rather complicated and computationally too demanding to be operated in real-time. As a result, there still lacks an efficient and powerful algorithm capable of processing challenging live video scenes with minimum user interactions.



(a) a separable case (b) an inseparable case

Figure 5.1: Comparison between a binary SVM and two 1SVMs under two situations. White circles and black dots represent the foreground and background training instances, respectively, while red dot denotes an unseen example. The straight line indicates the decision boundary of the binary SVM, whereas the ellipsoids show the boundaries of the two 1SVMs. In (a), binary SVM classifies the test example as foreground, whereas the 1SVMs labels it as unknown, since neither of the 1SVMs accepts it as inlier. In (b), binary SVM cannot confidently classify the test example since it is too close to the decision boundary, whereas 1SVMs is able to label it as background with confidence since only background 1SVM accepts it as inliner.



(a) input frame 0 (b) input stroke (c) after 1 iteration (d) after 2 iterations (e) after 3 iterations (f) matting pixels

(g) after convergence (h) matting pixels (i) binary segmentation (j) composite (k) input frame 1 (l) initial label

(m) alpha matte (n) composite (o) input frame 104 (p) initial label (q) alpha matte (r) composite

Figure 5.2: Handling the "kim" sequence [14], which is challenging due to fuzzy object boundaries and camera motions. The user is only required to label the first frame (a) using strokes (b). Local classifiers are trained at each pixel location and then used to relabel the center pixel (c). Iterative training and relabeling leads to convergence (d, e, & g), even though ambiguous (grey) areas still exist. At each iteration, pixels along fore/background boundaries (non-cyan pixels in f & h) are detected, for which matting is performed. The final binary segmentation is computed through graph-cut optimization (i). Combining binary segmentation (i) with boundary matte (h) produces the full alpha matte, which is used to generate a blue screen composite (j). When new frames (k & o) arrive, they are initially labeled (l & p) using the classifiers trained by previous frames, before the same train-relabel-matting procedure takes place to produce the alpha mattes (m & q) and composites (n & r).

To fill that niche, this chapter applies the STOCS learning model introduced in Chapter 3 to solve the binary classification problem. Compared to the conventional binary SVMs, STOCS uses two online 1SVMs, which learn foreground and background distributions separately. We hypothesize that better performance can be achieved using two 1SVMs in the application. Here are the reasons: First, foreground and background may not be well separable in the color feature space. For example, the black sweater and the dark background shown in Figure 5.2(a) share a similar appearance. As a result, it is not proper to deal with this scenario by means of training a global binary SVM and use it to classify the entire image. Second, trying to train local binary SVMs at each pixel location is problematic as well since in most cases merely one of the two (foreground or background) types of observations is locally available. In fact, even in areas that both foreground and background examples are available, modeling the two sets separately using two 1SVMs produces two hyperplanes that enclose the training examples more tightly. As illustrated in Figure 5.1, this helps toward better detecting and handling of ambiguous cases.

The presented approach solves video foreground segmentation and boundary matting in an integrated manner. As shown in Figure 5.2, with only a few strokes from user on the first frame of the video (users do not need to provide any interface in the remaining frames), the algorithm is able to propagate labeling information to neighboring pixels through a simple train-relabel-matting procedure, resulting in a proper segmentation of the frame. This same procedure is used to further propagate labeling information across adjacent frames, regardless of the foreground or background motions. Furthermore, by exploiting the parallel structure of the proposed algorithm, real-time processing speed of 14 frames per second (FPS) is achieved for VGA-sized

videos when matting is not applied, with the frame rate dropping to 8 FPS when matting over large fuzzy areas is needed. Details of the train-relabel-matting procedure used for each frame is discussed in the following sections.

5.1 Foreground Segmentation

In this section, the method of applying STOCS for foreground segmentation is presented. Followed by the idea of data classification with one-class SVM (see Section 1.1), the key insight of our approach is to train and maintain two one-class SVM models at every pixel location. The two 1SVMs capture the local foreground and background color densities separately, but determine a proper label for the pixel jointly. By iterating between training local 1SVMs and applying them to label the pixels, the algorithm effectively propagates initial user labeling to the entire image, as well as to consecutive frames.

As shown in Figure 5.2 and Algorithm 1, the core of our approach is a train-relabel-matting procedure: Based on the online 1SVM model introduced in Chapter 2 and 3, two STOCS models ($\mathcal{F}_p$ for foreground and $\mathcal{B}_p$ for background) are *trained* locally for each pixel p using known foreground and background colors within the local window Ω_p. Once trained, $\mathcal{F}_p$ and $\mathcal{B}_p$ are used to jointly *label* p as either foreground, background, or unknown. Please note that p becomes a labeled pixel and is used for training in the later train-relabel-matting iterations, allowing STOCS works in a semi-supervised learning manner. Pixels along the boundary between foreground and background regions are then detected and form a *matting pixel* set M, on which matting operation is performed (Details about matting operation is discussed in the

Algorithm 1 Foreground segmentation from user strokes

for each input frame I^t **do**

if $t == 0$ **then**

Initialize the label map L^0 based on input stroke;

Initialize the matting pixel set M to empty;

else

Train $\mathcal{F}$ & $\mathcal{B}$ using I^{t-1} & G^{t-1} for $p \notin M$ (Eq. 3.3);

Label I^t using $\mathcal{F}$ & $\mathcal{B}$ to obtain L^t (Eq. 3.3);

Apply temporal decay to all support vectors in $\mathcal{F}$ & $\mathcal{B}$;

Reset the matting pixel set M to empty;

end if

repeat

Train $\mathcal{F}$ & $\mathcal{B}$ using I^t & L^t for $p \notin M$ and using α^t, F^t, & B^t for $p \in M$ (Eq. 3.3);

Relabel I^t using $\mathcal{F}$ & $\mathcal{B}$ and update L^t (Eq. 3.3);

if matting is needed **then**

Update the matting pixel set M;

Estimate α^t, F^t, & B^t for $p \in M$;

end if

until there are no more changes in L^t

Find optimal binary segmentation G^t using graph cuts;

end for

next section). The train-relabel-matting operation at each pixel is independent, which runs in parallel on GPUs in our experiments.

Since the knowledge learned from neighboring pixels in Ω_p is considered in labeling p, the above procedure effectively propagates known foreground and background

information to its neighborhood. As a result, based on only a few initial strokes, the algorithm can segment the whole image and perform matting along object boundaries (see Figure 5.2(a-i)).

The same train-relabel-matting procedure is employed for handling temporal changes as well. When a new frame $t+1$ arrives, the label $L^{t+1}(p)$ is initialized automatically using the existing $\mathcal{F}_p$ and $\mathcal{B}_p$ from the last frames. The initial labels, together with newly observed colors, are then utilized to conduct the train-relabel-matting process. Since $\mathcal{F}_p$ and $\mathcal{B}_p$ are trained using all pixels within Ω_p of frame t, if any of those pixels in Ω_p moves to p, $\mathcal{F}_p$ and $\mathcal{B}_p$ are able to classify it properly. Consequently, the algorithm can cope with arbitrary foreground and background movement without *a prior* motion information, as long as the amount of movement is less than the radius of neighboring window Ω. For the setting of Ω value in our implementation, the same subgrouping approach in [30] is used in this thesis.

Finally, under ideal situations, where the appearance distributions of foreground and background pixels are locally separable, the above baseline procedure is sufficient. However, since users provide foreground and background examples using only a few strokes, there may be pixels in the frame with colors that are not recognized by either foreground or background 1SVMs. Similar situation also occurs when pixels with new foreground or background colors show up in the frame due to motions. These pixels are labeled as unknown at the end of the train-relabel-matting process as shown in Figure 5.2(g). To label these unknown pixels, as well as pixels along fore/background boundaries so that a clean binary segmentation can be generated, our final step is to compute the globally consistent and smooth solution G based on Graph Cut [6]. In practice, we use a GPU version of the push-relabel algorithm to compute the min

cuts [29], and limit the number of push-relabel steps to 20, which is found sufficient throughout our experiments.

5.2 Boundary Matting

Both motion blur and fuzzy foreground objects such as hair strands may cause pixels near foreground boundary having a mixture of foreground and background colors. In the foreground segmentation problem, it is classified as foreground or background by enforcing smoothness of segmentation. Here we decompose the observed colors for these pixels into fore/background values and the alpha mattes, directly producing a soft segmentation for the foreground, which is known as image matting in computer vision.

It is well-known that matting is an ill-posed problem with possibly multiple solutions. At each pixel, the unknowns on the right side of the following image compositing equation need to be estimated using the known observed color I:

$$I = \alpha F + (1 - \alpha)B, \tag{5.1}$$

where $\alpha \in [0, 1]$ is the alpha matte; F and B are the real foreground and background colors for the pixel, respectively.

To solve the problem, we add a further constrain that F and B should fit the local foreground and background 1SVMs ($\mathcal{F}_p$ and $\mathcal{B}_p$) as much as possible. That is, the scores $f_{\mathcal{F}_p}(F)$ and $f_{\mathcal{B}_p}(B)$ should be high. Consequently, we optimize the following the energy function at each pixel:

$$\arg\max_{F,B,\alpha} e^{-(\alpha F+(1-\alpha)B-I)^2/2\sigma_c^2} + f_{\mathcal{F}_p}(F) + f_{\mathcal{B}_p}(B), \tag{5.2}$$

where parameter σ_c controls the support of the Gaussian and $\sigma_c = 5$ in our experiments. $f_{\mathcal{F}_p}(\cdot)$ and $f_{\mathcal{B}_p}(\cdot)$ are foreground and background scores, which are computed based on the Equation 3.3. When differentiable kernels such as Gaussian kernel are used for 1SVMs, Equation 5.2 is differentiable, which allows us to use gradient-based approaches to search for the optimal F, B and α values.

In practice, the matting module starts with determining the matting pixel set M. We treat a pixel p as a matting pixel *iff.* sufficient number of examples are used to train both the foreground and the background 1SVMs, $\mathcal{F}_p$ and $\mathcal{B}_p$. Here we require the numbers of both foreground and background examples within the 33×33 local window to be greater than 50. Once the matting pixels are determined, a nonlinear conjugate gradient technique is applied to these pixels in parallel. The initial F and B values are set to the mean colors of foreground and background training examples within the local 33×33 window, respectively. The initial α is set to the local mean of the label map L for the first train-relabel-matting pass and to the mean of previously estimated alpha matte for the following passes. During the optimization, we explicitly enforce F, B to remain in range $[0, 255]$ and α to remain in range $[0, 1]$. Please note that, after optimizing the Equation 5.2, the training examples in the next iteration would be the estimated fore/background colors of matting pixels; see Figure 5.3(a & b).

It is worth noting that, even though the matting is performed for different pixels in parallel without explicitly enforcing the smoothness, the estimated alpha matte is

(a) F examples (b) B examples (c) alpha matte

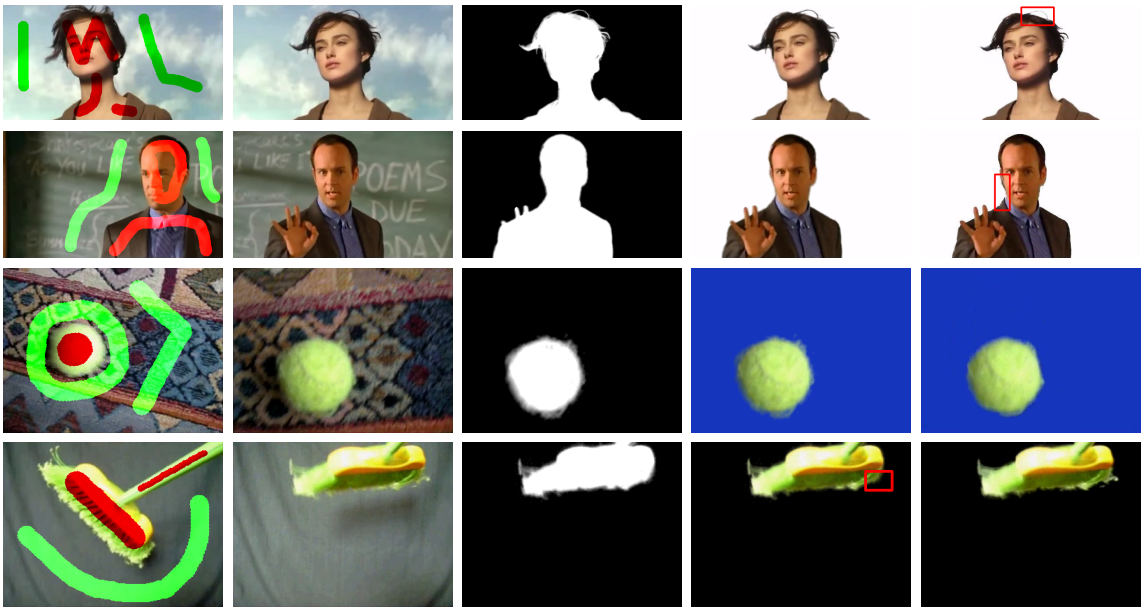
Figure 5.3: Alpha matte estimation. Cyan color in (a) and (b) indicates no foreground or background training examples are available for the corresponding pixels. Cyan in (c) indicates non-matting pixels, where the number of foreground or background examples within the local neighborhoods is insufficient.

still smooth due to the coherence among neighboring 1SVMs that models foreground and background colors. In a similar manner, the temporal coherence among the alpha mattes of adjacent frames is also enforced implicitly.

5.3 Results

To exploit the inherit parallel structure of the proposed algorithm, we implement it on GPU using DirectCompute, which is proposed by Microsoft as an alternative to CUDA and is included in the Direct3D11 API. For the VGA-sized "kim" sequence with large fuzzy area, our current implementation runs at 14 FPS without matting and 8 FPS with matting on a Lenovo ThinkStation S20 with nVidia GeForce GTX 480 GPU. Besides, Gaussian kernel is used as the kernel function in the application.

Comparison with existing approaches: Figures 5.2 and 5.4 show the results of our algorithm for several sequences used by previous video matting papers [14, 2,



(a) first frame with user strokes (b) test frame (c) alpha matte (d) composite (e) other approaches

Figure 5.4: Results on testbed sequences referred to as (from top to bottom) "wind", "class" [2], "ball", and "broom" [27]. The results show that our algorithm can properly handle background motions (in "class" & "ball") and strong motion blur (in "ball" & "broom"). The results of geodesic matting [2] (for "wind" & "class") and shared matting [27] (for "ball" & "broom") are generated by the authors (e). For better comparison, matching background colors are used in our composites (d). Areas with suboptimal matte results are highlighted with red boxes.

31, 27, 13]. Visual inspection suggests that the alpha mattes estimated are smooth and rich in details. Available results generated by authors of existing approaches are also shown in Figures 5.4 and 5.5 for comparison. They suggest that the performance of our algorithm is on par with existing approaches, which often require additional steps to compute dense trimaps [27, 31] or background colors [14] for each frame.

Some of these approaches also require long computational time [14, 13] or process all frames in a batch manner [2], making them hard to be applied to real-time video processing. In comparison, our approach only requires user to provide few key strokes and is designed for parallel execution on GPUs, which is simple and efficient.

(a) test frame (b) our result (c) Bayesian matting [14] (d) multi-chan. Poisson [31] (e) non-local matting [13]

Figure 5.5: Comparison on alpha mattes generated by different approaches. Areas with suboptimal matte results are highlighted with red boxes.

Robustness w.r.t. input stroke variations: To evaluate the impact of the users' input stroke variations on the matting results, here we also test our algorithm under different input settings. As shown in Figure 5.6, changing input stroke locations affects how the labeling information is propagated across the image, but has very little impact on the final alpha matte generated. This suggests that our program is robust against users' input variations.

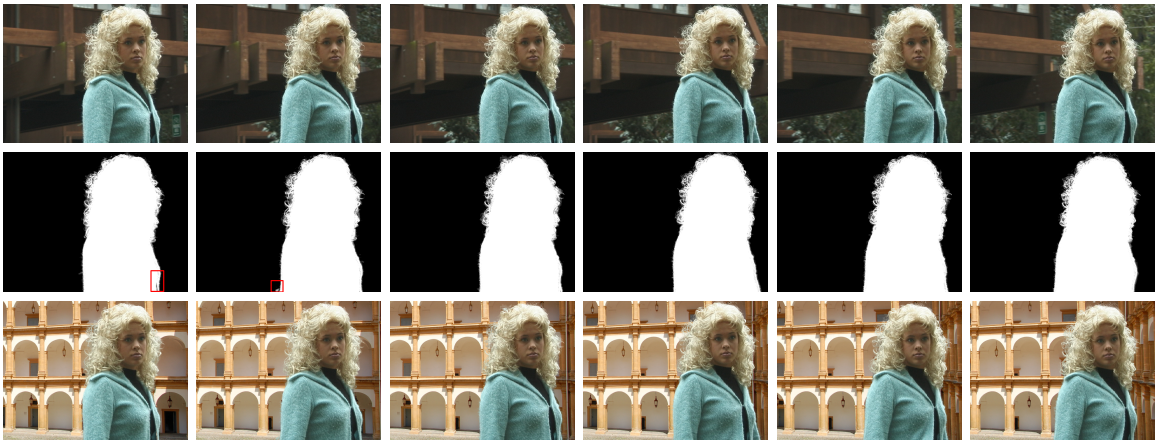
Evaluation on temporal coherence: Unlike some existing video matting approaches [41, 3, 13], our approach does not explicitly enforce the temporal coherence among adjacent frames. Instead, it relies on the local fore/background classifiers trained at each pixel location, $\mathcal{F}$ and $\mathcal{B}$, being stable across multiple frames. As a result, for pixels from different frames but with similar observed colors, the optimal alpha values found by maximizing Equation 5.2 will be similar. To evaluate

(a) first frame w/ strokes (b) after 10 iterations (c) after convergence (d) labeling for test frame (e) alpha matte for test frame

Figure 5.6: Matting results obtained under different stroke inputs than the one shown in Figure 5.4(a). Under different stroke inputs (a), labeling information is propagated differently across the image (b), but the final per-pixel labeling results (c) are similar. The impact of the stroke variations on the first frame is even less noticeable in the per-pixel labeling results for the test frame (d). As a result, the alpha mattes generated for the test frame (e) are nearly identical to the one shown in Figure 5.4(c).

the effectiveness of this strategy, Figure 5.7 shows the matting results for six nearby frames of the "kim" dataset, which contains a large and detailed fuzzy area. The results demonstrate that our approach is able to generate coherent alpha mattes for the foreground object even in front of a changing background.

Please note that our approach is mainly designed for extracting alpha mattes along the boundaries of fuzzy objects, whereas it cannot be used to handle large semi-transparent regions since the matting relies on locally trained foreground and background classifiers. For highly transparent object such as the ones used for image matting evaluation [51], a more global classifier trained using non-local examples would be needed.



(a) frame 86 (b) frame 89 (c) frame 92 (d) frame 95 (e) frame 98 (f) frame 101

Figure 5.7: Alpha mattes (middle row) and composites (bottom row) generated for adjacent frames. Despite that the background behind the fuzzy hair changes from bright window to brown building then to green leaves, our approach extracts temporal coherent alpha mattes. Please note that although artifacts show up in the estimated alpha mattes (highlighted with red boxes), they are hardly noticeable in the final composites.

In summary, STOCS is successfully applied to solve foreground segmentation and boundary matting in an integrated manner. Finally, our algorithm is shown to be particularly competent at processing a wide range of videos with complex backgrounds from freely moving cameras. This is usually achieved with minimum user interactions. Near real-time processing speed is achieved for VGA-sized videos.

Chapter 6

Conclusions

The thesis proposes a novel data classification approach, STOCS, that is parameter-free, efficient, and capable of dealing with label noise. Different from the conventional method of 1SVM model for data classification, which is trained only based on positive examples, the decision boundary of STOCS is learned from both positive and negative examples. Three parameters, including the decay parameter, the kernel bandwidth and the cut-off value, are either removed or adaptively selected. Self-tuning cut-off values for different support vectors also helps us better handling label noise in training data. We also improve the original online learning method by removing the decay parameter, which makes STOCS achieve faster convergence. By further exploiting the online learning framework, STOCS has the ability to work well on dynamic and large-scale data.

Our empirical study demonstrate STOCS's effectiveness of improving convergence speed, parameter self-tuning and robustness to label noise. Compared with LIBSVM, arguably the most widely-used data classification system, STOCS almost always out-

performs the LIBSVM under the default parameter settings, while uses much fewer support vectors at the same time. Its performance is also very close to LIBSVM's optimal results obtained by cross-validation for individual datasets, but uses only a fraction of the processing time.

Besides working on multiclass classification, a binary classification application in computer vision is presented. A novel video segmentation and matting solution is proposed that is able to efficiently and effectively deal with live videos. The solution is easy to implement, simple to use, and capable of handling a variety of difficult scenarios, such as dynamic background, camera motion, topology changes, and fuzzy objects. For fuzzy objects, the integrated boundary matting step can effectively pull the matte along object boundary, allowing seamless composites over new backgrounds. Furthermore, by introducing novel acceleration techniques and by exploiting the parallel structure of the algorithm, near real-time processing speed (14 FPS without matting and 8 FPS with matting on a mid-range PC & GPU) is achieved for VGA-sized videos. Finally, with the assistance of the STOCS training framework, experiments on standard testbed videos demonstrate that our solution possesses comparable or superior performance comparing to the state-of-the-art approaches [14, 31].

6.1 Future Works

Supervised learning requires completely labeled data. However, the labeling processes always require human annotations, which is boring and may require experts, or physical experiments, which may require special devices. The cost associated with labeling makes acquisition of a sufficient labeled dataset hard and even infeasible, whereas

getting unlabeled data is relatively inexpensive. To cope with this problem, semi-supervised learning [10] is proposed, which makes use of unlabeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. In the presented application, large scale dynamic video data classification is successfully solved in a semi-supervised manner. However, extending the current approach to general semi-supervised learning based classification problems requires further study. In the future, we plan to apply STOCS to solve general semi-supervised problems. The possible route is to first train 1SVM models using labeled data, and then use the 1SVMs to label the data that we have the most confidence on. The above process will be repeated until all data are labeled.

Large-scale and high-dimensional data classification such as document classification is useful in many applications, but training large quantities of data remains an important research issue. The most popular big data classification solver is LIBLINEAR [22], whereas it cannot handle large data that does not fit into memory well. Unlike LIBLINEAR that uses batch learning, STOCS uses online learning, where training examples are shown to the learner one by one. STOCS only needs to store support vectors without loading all training examples into memory, which motivates us to plan to apply STOCS to handle big data classification. The possible solution is that examples are randomly shown to the learner until the classifier can label a random data collection correctly. Hence, it is possible that the training model achieves convergence with loading a fraction of examples only. Furthermore, to avoid support vectors cannot fit into memory and accelerate training speed, only the dominant support vectors are kept. This strategy is also used in the presented application for the purpose of real-time processing.

Our current approach can learn an online 1SVM model for each class, which secures a compact area belonging to this class with high confidence. There is still a large area in-between different classes that cannot be labeled with confidence. Figure 5.1(a) illustrates the simplest case, i.e. binary classification, where the two online 1SVMs cannot confidently classify data points between the two ellipsoids. To make STOCS more powerful especially on reducing ambiguities on labeling the in-between areas, we plan to further improve our current approach in the future. A possible route is to incorporate STOCS as weak classifiers into the AdaBoost [24] learning scheme. By applying AdaBoost, we can combine multiple weak classifiers, i.e. STOCS in our case, into a single strong classifier, which integrates information of STOCS learned from different training sets in different stages and may label the ambiguous areas with confidence.

Bibliography

[1] D. Angluin and P. Laird. Learning from noisy examples. *Mach. Learn.*, 2(4):343–370, 1988.

[2] X. Bai and G. Sapiro. Geodesic matting: A framework for fast interactive image and video segmentation and matting. 82(2):113–132, 2009.

[3] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video snapcut: robust video object cutout using localized classifiers. 2009.

[4] M. Boullé. A parameter-free classification method for large scale learning. *Journal of Machine Learning Research*, 10:1367–1385, 2009.

[5] O. Bousquet and L. Bottou. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 161–168. 2007.

[6] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, Nov. 2001.

[7] T. Bylander. Learning linear threshold functions in the presence of classification noise. In *COLT*, 1994.

[8] N. Cesa-Bianchi, S. Shalev-Shwartz, and O. Shamir. Online learning of noisy data. *IEEE Transactions on Information Theory*, 57(12):7907–7931, 2011.

[9] C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[10] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[11] L. Cheng, M. Gong, D. Schuurmans, and T. Caelli. Real-time discriminative background subtraction. *Trans. Img. Proc.*, 20(5):1401–1414, May 2011.

[12] L. Cheng, S. Vishwanathan, D. Schuurmans, S. Wang, and T. Caelli. Implicit online learning with kernels. In *Advances in Neural Information Processing Systems (NIPS)*, pages 249–256. MIT Press, Cambridge, MA, 2007.

[13] I. Choi, M. Lee, and Y.-W. Tai. Video matting using multi-frame nonlocal matting laplacian. pages 540–553, 2012.

[14] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. pages 243–248, 2002.

[15] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[16] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, Mar. 2002.

[17] P. Craven and G. Wahba. Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math.*, 31:377–403, 1979.

[18] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, first edition, 2000.

[19] J. Dai, S. Yan, X. Tang, and J. Kwok. Locally adaptive classification piloted by uncertainty. In *ICML*, 2006.

[20] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286, Jan. 1995.

[21] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *KDD*, 2008.

[22] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

[23] B. Frenay and M. Verleysen. Classification in the presence of label noise: a survey. *IEEE Trans. Neural Networks and Learning Systems*, 2014.

[24] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, Aug. 1997.

[25] T. Gao and D. Koller. Multiclass boosting with hinge loss based on output coding. In *Proceedings of International Conference on Machine Learning (ICML)*, 2011.

[26] N. García-Pedrajas and A. de Haro-García. Scaling up data mining algorithms: review and taxonomy. *Progress in AI*, 1(1):71–87, 2012.

[27] E. S. L. Gastal and M. M. Oliveira. Shared sampling for real-time alpha matting. 29(2):575–584, May 2010.

[28] C. Gold, A. Holub, and P. Sollich. Bayesian approach to feature selection and parameter tuning for support vector machine classifiers. *Neural Networks*, 18(5-6):693–701, 2005.

[29] M. Gong and L. Cheng. Real-time foreground segmentation on GPUs using local online learning and global graph cut optimization. pages 1–4, 2008.

[30] M. Gong and L. Cheng. Foreground segmentation of live videos using locally competing 1svms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[31] M. Gong, L. Wang, R. Yang, and Y.-H. Yang. Real-time video matting using multichannel Poisson equations. pages 89–96, 2010.

[32] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, NY, USA, 2009.

[33] E. Herrmann. Local bandwidth choice in kernel regression estimation. *Journal of Computational and Graphical Statistics*, 6(1):35–54, 1997.

[34] M. Izbicki. Algebraic classifiers: a generic approach to fast cross-validation, online training, and parallel training. In *International Conference on Machine Learning*, 2013.

[35] R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. *J. Mach. Learn. Res.*, 8:227–248, 2007.

[36] J. Kim and C. Scott. Variable kernel density estimation. *Annals of Statistics*, 20:1236–1265, 1992.

[37] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, 1995.

[38] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[39] N. D. Lawrence and B. Scholkopf. Estimating a kernel fisher discriminant in the presence of label noise. In *ICML*, 2001.

[40] J.-S. Lee and I.-S. Oh. Binary classification trees for multi-class classification problems. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, pages 770–, Washington, DC, USA, 2003. IEEE Computer Society.

[41] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *ACM Trans. Graph.*, 24(3):595–600, July 2005.

[42] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, Feb. 1994.

[43] A. Lorbert and P. J. Ramadge. Descent methods for tuning parameter refinement. In *AISTATS*, 2010.

[44] Z. Lu and Y. Peng. Robust image analysis by l1-norm semi-supervised learning. *CoRR*, abs/1110.3109, 2011.

[45] L. M. Manevitz and M. Yousef. One-class svms for document classification. *J. Mach. Learn. Res.*, 2:139–154, Mar. 2002.

[46] A. Muoz and J. M. Moguerza. One-class support vector machines and density estimation: The precise relation. In A. Sanfeliu, J. F. M. Trinidad, and J. A. Carrasco-Ochoa, editors, *CIARP*, volume 3287 of *Lecture Notes in Computer Science*, pages 216–223. Springer, 2004.

[47] N. Natarajan, I. Dhillon, P. Ravikumar, and A. Tewari. Learning with noisy labels. In *NIPS*. 2013.

[48] D. F. Nettleton, , A. Orriols-Puig, and A. Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33:275–306, 2010.

[49] J. C. Platt, N. Cristianini, and J. Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems 12*, pages 547–553, 2000.

[50] R. B. Rao and G. Fung. On the dangers of cross-validation. an experimental evaluation. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, pages 588–596, 2008.

[51] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. pages 1826–1833, 2009.

[52] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, Dec. 2004.

[53] N. V. Sawant, K. Shah, and V. A. Bharadi. Survey on data mining classification techniques. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, pages 1380–1380, New York, NY, USA, 2011. ACM.

[54] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001.

[55] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 582–588. 1999.

[56] C. Scott, G. Blanchard, and G. Handy. Classification with asymmetric label noise: Consistency and maximal denoising. In *COLT*, 2013.

[57] M. Stone. Cross-validatory choice and assessment of statistical predictions. *J. of Roy. Stat. Soc. (JRSS) Series B*, 36:111–147, 1974.

[58] W. Sun, J. Wang, and Y. Fang. Consistent selection of tuning parameters via variable selection stability. *Journal of Machine Learning Research*, 14:3419–3440, 2013.

[59] D. M. J. Tax and R. P. W. Duin. Support vector data description. *Mach. Learn.*, 54(1):45–66, Jan. 2004.

[60] J. Wang and M. F. Cohen. Image and video matting: a survey. 3(2):97–175, Jan. 2007.

[61] W. Wang and A. Gelman. A problem with the use of cross-validation for selecting among multilevel models. Technical report, Columbia Univ., New York, 2013.

[62] Y. Yajima. One-class support vector machines for recommendation tasks. In *Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'06, pages 230–239, Berlin, Heidelberg, 2006. Springer-Verlag.

[63] T. Yang, M. Mahdavi, R. Jin, L. Zhang, and Y. Zhou. Multiple kernel learning from noisy labels by stochastic programming. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning*

(ICML-12), ICML '12, pages 233–240, New York, NY, USA, July 2012. Omnipress.

[64] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study of their impacts. *Artif. Intell. Rev.*, 22(3):177–210, 2004.

[65] H. Zou. The adaptive lasso and its oracle properties. *Jounal of the American Statistical Association*, 101:476, 2006.