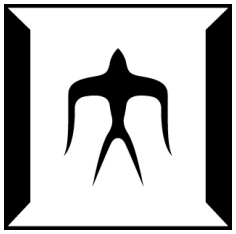
Online boosting algorithm based on two-phase SVM training and its application to image processing



Author:
Yugov Vsevolod

Supervisor:
Itsuo Kumazawa

Department of Information Processing

Tokyo Institute of Technology

A thesis submitted for the degree of

Doctor of Philosophy

September 2012

Abstract

We describe and analyze a simple and effective two-step online boosting algorithm that allows us to utilize highly effective stochastic gradient descent based methods developed for online SVM training without the need to fine-tune kernel parameters, and show its efficiency by several experiments. Our method is similar to the AdaBoost in that it trains additional classifiers according to the weights provided by previously trained classifiers, but unlike AdaBoost we utilize hinge loss rather than exponential loss, and modify algorithm for online setting, allowing for varying number of classifiers. We show the effectiveness of our method by developing applying it to the task of object tracking on the mobile device (iPhone). In order to achieve the real-time processing speed we furthermore describe a set of compact features in order to fully utilize the parallel processing capabilities of the device GPU. We then show that utilizing our algorithm with such features allows for a high discrimination rate even with a small number of features being utilized.

To ...

Acknowledgements

I would like to acknowledge all the people that have helped me with this research problem. First of all, I thank my academic advisor, prof. Kumazawa, for giving me a chance to live and study in Japan, and for his support during my research. I would also like to thank other member of my laboratory, especially Fukushi-san and Matsumura san, as well as recently graduated Quivy-san, for their help in discussing my work and assisting me in data collection, and just for being good friends. I recognize that this research would not have been possible without the financial assistance of Japans Ministry of Education, Culture, Sports, Science and Technology (MEXT), to which I express my gratitude.

Also, I want to thank my family, especially my grandmother who has raised and always supported me, my mother, whose initiative has made my study here possible, and my brother, who has always helped me in tight spots.

Last, but not least, I would like to express my gratitude to Hirasawa Makiko and Saeko, who has provided me home away from home in Japan, and without whose moral support I'd have never made it so far.

Contents

List of Figures

LIST OF FIGURES

List of Tables

1

Introduction

1.1 Aims of research

Recently, there has been several breakthroughs in the area of online learning and classification. This was partly driven by the massive amount and ever increasing speed of data acquisition in our information-driven society. The amount of data, especially in applications related to image processing quickly outstrips the capacity of many common learning algorithms that require all of the data to fit in memory or to be available at the same time. Futhermore, many of them have computational requirements that are polynomial of degree two or above on the amount of data. In particular, algorithms related to Support Vector Machines (SVM), a common and effective classification tool introduced by Vapnik =ref=, are usually quite computationally expensive.

For these reasons, several low-complexity, linear- or near-linear time algorithms has been developed specifically for the tasks that either have online limitations, i.e. only a limited number of samples available at the time and only sequential access to data, or need to process the amount of data that cannot fit in memory.

Two related types of classifier methods enjoy increased attention lately. For one, Support Vector Machines, primarily binary classifiers that have been successfully used for various tasks in image and signal processing, speech recognition and DNA analysis, have been successfully adapted to the online setting by using Stochastic Gradient Descent methods on their primal formulation. However, while incredibly efficient in the case of linear classification, introducing kernels to achieve nonlinearity in the online setting results in the rapid increase of computational complexity, as kernel expansion

coefficients are accumulated.

On the othe hand, boosting algorithm for aggregating several simple classifiers into one stronger has also been adapted for various online and adaptation tasks, especially in the areas like object tracking in the video sequence, where the adaptability of a model to changing conditions is paramount. Most of such methods, however, limit the complexity, and, as a result, possible accuracy, by fixing the amount of added classifiers and fixing the feature pool.

Our Goal In this thesis we aim to bridge the gap between boosting and online SVM learning by exploiting the simolarities between both that allow us to introduce a new boosting algorithm based on two-step SVM training. The proposed algorithm allows for greater flexibility, and has smaller computational costs than traditional kernel-based methods while achieving similar or greater accuracy. To our knowledge this is the first such algorithm proposed.

Contrbitions Our contributions in this paper, are therefore as follows. First we propose a new learning method and compare it against existing methods in terms of accuracy and computational complexity, as well as proposing several variations of the method useable for various applications. Then, we introduce a specific application of the descibed method to the task of video tracking on the mobile device. We show that reduced computational costs of our method, as well as highly parallel processing on the device's GPU allows such complex applications to run in the real time. Also, for this application a new set of simple features is introduced and evaluated.

1.2 Overview of common methods

As mentioned above, the algorithm introduced in this paper is based partly on new SGD-based training methods, as well as well known algorithms, such as Adaboost. In particular, our work has been inspired by the following algorithms:

NORMA Naive Online Risk Minimization Algorithm =ref= is one of the first and most generics algorithms based on stochastic gradient descent. It can be applied to various online task that require nonlinear separation of data, including kernel-based SVM, regression and novelty detecion. This paper has served as a basis to many other related algorithms, and provides a solid theoretical background by defining bounds on error and convergence rates of SGD-based methods.

Pegasos Primal Estimated sub-GrAdient SOlver for SVM (PEGASOS) (==ref==) is a more recent algorithm that bridges the span between online and batch learning by allowing several samples to be processed at once. It gives significantly iproves convergence speed compared to NORMA at the price of a slight increase of a computational complexity of a single iteration.

Online AdaBoost In a series of papers, =ref=, and online boosting algorithm for feature selection is introduced, and several applications of it are discussed. This algorithm, and its limitation, are what has originall inspired us to work on a boosting-related methods.

1.3 Algorithm outline

Our algorithm takes at its basis the offline Adaboost algorithm and transfers it to online setting by utilizing its similarity to the SVM formulation.

1.4 Possible applications

In the introduction, we briefly state the need and motivation of our research. We describe the current state of the art techniques for the SVM training and boosting, and show some of their drawback that we aim to address by introducing our method. We then briefly outline our algorithm and its relation to the previously described algorithms. We further describe possible applications of our algorithm, particularly in the field of image processing.

2

Related works

Both the SVM and boosting-based learning algorithms, and their applications to various tasks in image processing are extremely widespread. In this chapter we outline the works most closely related to the presented methods, since the size constraints of the paper do not allow for a detailed review of this area.

2.1 Overview

In general, classification is a problem of identifying to which set, or cathegory belongs the next observation. The cathegories may be given beforehand, or derived from the data itself by application of a chosen clustering method. Usually, a certain set of data samples is given beforehand (training dataset), which serves as a basis by which the membership of the new sample is determined. The data samples are transformed into a set of explanatory variables, or features, and from them the cathegory-defining model, or classifier, is built.

While the distinction between offline and online algorithms for training classifiers is not clearly defined, it is usually accepted that the offline systems have random access to all training data at the same time, and that the model resulting from this data should asymptotically converge to the equilibrium. In this setting, the training time is less important. On the other hand, online systems have only limited access to the data, usuallly to a single sample at a time, or a few consecutive samples, and it is preferrable for the learning iteration to run in the real time, i.e. that the update iteration should take less time then the acquisition of the next data sample.

There are several types of classifiers, such as bunary or multiclass, linear, nonlinear and cathegorical, etc. In this paper, we focus on the linear binary classifiers, which can be used for nonlinear classification by transforming feature space.

2.2 Offline classification algorithms

2.2.1 Support vector machines

The support vector machines are a class of linear binary classifiers that attempt to maximize the minimal distance (margin) between classes, i.e. to construct a hyperplane in the feature space that separates the two classes and is located, intuitively, exactly in the "middle" between them (see =fig=). The mathematical formulation for this problem is: given a set of training samples $\vec{x}_i$ and associated labels y_i, $i \in [1..n]$, minimize in$\vec{w}, b$ $\frac{1}{2}||\vec{w}||^2$ subject to $y_i(\vec{w}\vec{x}_i - b) \geq 1$. Introducing Lagrangian multiplers α_i, the primal formulation then becomes as follows:

$$\min_{\vec{w},b} \max_{\alpha_i} \left\{ \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w}\vec{x}_i - b) - 1) \right\} \tag{2.1}$$

The classifier output is then the sign of the confidence function $f(\vec{x}, \vec{w}) = \vec{w} * \vec{x} - b$,

$$H(\vec{x}) = sign(f(\vec{x}, \vec{w}))$$

The dual formulation of the above problem

$$\max_{\alpha_i} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum j = 1^n \alpha_i \alpha_j y_i y_j k(\vec{x_i}, \vec{x_j}) \right) \tag{2.2}$$

subject to $\alpha_i \geq 0$,where, in original linear case, $k(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$, $\cdot$ denoting an inner product, and $\vec{w} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$. In this formulation, data points $\vec{x}_i$ for which $\alpha_i > 0$ are called the support vectors, giving rise to the name of the method. Intuitively, these are the points that lie on the margin hyperplanes.

Original =ref= formulation assumed that the data was linearly separable, and couldn't be solved for the noisy data. One of the most important results related to this version of classifier was that, when solvable, it was shown =ref= to minimize the theoretical upper bound on the testing error rate. This bound is related to the VC dimension of the classifier, and governs the relation between the capacity of a learning machine and its performance. The bound is as follows: if the classifier with parameters

$\vec{\alpha}$ achieves empirical error rate, i.e. error rate on a set used for training, $R_{emp}(\vec{\alpha})$, then with probability $1 - \eta$, the following bound holds:

$$R(\vec{\alpha}) \leq R_{emp}(\vec{\alpha}) + \sqrt{\left(\frac{h(log(2l/h) + 1) - log(\eta/4)}{l}\right)} \tag{2.3}$$

where l is the number of training samples and h is the VC dimension of a classifier function, defined as the number maximum cardinality of a data point set that can be shattered (separated for any assignment of labels $y \in -1; 1$ to data points). For example, a linear classifier of dimension n has VC dimension of $n + 1$.

SVM were then originally derived as a family of classifiers minimizing right-hand part of =equation=, thus minimizing the expected risk and generalization error. The original classification, however, was too rigid and not very usable on sets of data from the real world, so in =ref=, Eq. (2.1) was rewritten, replacing hard constraints with a loss function:

$$\min_{\vec{w}} \left\{ \frac{\lambda}{2}||\vec{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} l(\vec{w}, (\vec{x}, y) \right\} \tag{2.4}$$

where $l(\vec{w}, (\vec{x}, y)$ is a loss function determining penalty for the outlier and λ is a parameter that determines the "softness" of the margin, with large λ favoring larger number of outliers with smaller margin.

The most commonly used loss function l is a hinge-loss function, that linearly punishes the outliers and margin errors, i.e. data points that are classified correctly but lie between margin hyperplanes

$$l(\vec{w}, (\vec{x}, y) = \max(0, \rho - y(\vec{w} \cdot \vec{x}))$$

where ρ is a margin parameter, usually assumed to be 1 for maximum margin algorithms.

When using Eq. (2.2.1) , dual form of Eq. (2.4) is the same as Eq. (2.2) , with the constraints changed to $0 \leq \alpha_i \leq C$, $C \propto \frac{1}{\lambda}$. This constrained quadratic problem is the one that is usually solved by most common offline methods.

2.2.1.1 Using kernel trick to create nonlinear classifier

It is easy to see that all of the above formulas can be expressed in terms of linear combinations of kernel function $k(\cdot, \cdot)$ on the input data samples and coefficients α_i.

For instance, confidence function of the classifier $f(\vec{x}, \vec{w}) = \vec{w} * \vec{x}$ can be replaced with $f(\vec{x}; \{\vec{x_i}, \alpha_i\, y_i\}) = \sum_{i=1}^{n} \alpha_i y_i k(\vec{x}, \vec{x_i})$. The linear function $k(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$ of the original formulation can then be replaced by any function satisfying Mercers condition, i.e. any positive definite kernel. There also exists a body of research (such as =ref=) that deals with practical applications of non-positive definite kernels (such as well known sigmoid function), but that lies beyond the scope of a current work.

The Mercers condition, in essence, guarantees that there exists a feature space V, that has an operation of inner product defined in it, and a map from the input data space S, $\phi : S \leftarrow V$so that kernel function between two vectors $\vec{x}, \vec{y} \in S$ is equivalent to inner product: $k(\vec{(}x, \vec{y}) = \phi(\vec{x}) \cdot \phi(\vec{y})$. Essentially, the kernel trick maps input vectors $\vec{x_i}$ into larger-dimensional feature vector space V , where, hopefully, the data becomes linearly separable (see =fig= for illustration).

This techniques allows using SVM as nonlinear classifier, and greatly expands the variety of possible applications since as long as the kernel function is defined, the input vectors $\vec{x_i}$ do not even have to be numbers,a nd can instead be words or area descriptors in an image (=ref=]). However, its application also increases the costs associated with learning and using the classifier, since instead of a single vector $\vec{w}$ we have to store all nonzero α_i and associated $\vec{x}_i$, and the increased dimensionality of the feature space gurantees the larger possible amount of support vectors as compared to the linear formulation. Also, the same increased dimensionality increases the VC dimension of the classifier, raising bounds on the generalization error, i.e. increasing the risk of overfitting. In addition, the kernel formulation makes the primal problem much harder to solve, biasing the existing research towards quadratic dual formulation, although the methods of dealing with it exist and are introduced in section 2.3.1.1 below.

The most commonly used kernels are listed in =table=.

2.2.1.2 Common methods for offline SVM training

Most common methods for SVM training deal with the solution to the Eq. (2.2) , and as such are usually applicable to the Quadratic Programming problems in general. Several classes of such silution techniques should be mentioned in this overview.

Interior point methods IP methods (for example, =ref=) replace linear constraints of the primal with a barrier function, similar to the IP methods for linear

programming. The result is a sequence of unconstrained problems which can be optimized very efficiently using Newton or Quasi-Newton methods. The advantage of IP methods is that they achive rapid convergence to a given accuracy bound in terms of a number of iterations. Unfortunately, they typically require run time which is cubic in the number of data samples n. Moreover, the memory requirements of IP very large, so such methods are not suitable for the training sets with large number of samples.

Segmentation-based methods To overcome the quadratic memory requirement of IP methods, decomposition methods such as as well-known SMO =ref= and SVM-Light =ref= work with dual variables α_i, constrained by a set of conditions that are derived from the current state of the solution, and thus change every iteration. In the extreme case, the active set consists of a single constraint. Therefore, the larger quadratic problem is segmented into a number of smaller ones, SMO emplying the smalles subset of just two variabes being optimized at a time. While algorithms of this family are simple to implement and have general asymptotic convergence properties, the time complexity of is still typically super linear in the training set size n.

Some of the decomposition methods can be adapted to the online learning setting, but the results are typically inferior in terms of convergence rate and accuracy to the methods specifically developed for the online setting.

Gradient-based methods Unconstrained gradient methods used to be common before the emergence of the more modern methods. While gradient based methods are usually known to exhibit slow convergence rates, the computational demands imposed by large scale classification problems of high dimension feature space, such as the ones common in image processing, has revived the theoretical and applied interest in gradient methods. Many of the online methods described below, as well as our proposed algorithm, were based on the modifications of gradient methods.

2.2.2 Boosting

Boosting is a name of a family of meta-learning algorithms that boost the performance of several low- accuracy claasifiers by combining them into a single classifier with increased accuracy. Usually, a linear combination of the classifiers' outputs is used, with the goal of the corresponding algorithm being the assigment of the weights to each classifier. In other words, given a set, or a pool of M classifiers $h_i(\vec{x})$, each with error ratio ϵ_i being arbitrarily close to a result of a random classification, 0.5, over a training

dataset (the error ratio may be unknown), the boosting algorithm attempts to find a set of boosting coeffficients $\{\beta_i\}$ to form a confidence function $F(\vec{x}, \{\beta_i\}) = \sum_{i=1}^{M} \beta_i h_i(\vec{x})$, so that a classifier

$$H(\vec{x}, \{\beta_i\}) = sign(F(\vec{x}, \{\beta_i\})) \tag{2.5}$$

would achieve error rate below arbitrary threshold.

Boosting algorithms were originally an answer to the question posed by Kearns (=ref=, of whether a set of weak classifiers can be combined to form an arbitrary strong classifier. The proof of the possiblity delivered by Shapite =ref= has significant impact on the field of classification, and has lead to many related algorithms being developed. Amongst the most effective and popular is the AdaBoost, first introduced in =ref=, which shall be reviewed in more detail below, since it forms part of the basis of our proposed method.

2.2.3 AdaBoost

In this section, we shall briefly describe the AdaBoost algorithm for later reference. For the detailed derivation, please refer to te =ref=.

AdaBoost, short for Adaptive Boosting, is a greedy algorithm formulated by Yoav Freund and Robert Schapire, that can be used in conjunction with many other lesrning algorithms serving as a source for the set of the weak classifiers. AdaBoost is adaptive in the sense that subsequent weak classifiers added to the solution are tweaked in favor of those instances misclassified by previous classifiers. For this reason, AdaBoost can be sensitive to noisy data, however, it performs well on most datasets, and have been successfully used for the variety of tasks. Of the particuar interest to our work is its application to image processing and feature selection, deescribed in =ref=.

AdaBoost adds a new weak classifier in each of a series of iterations $t = 1, \ldots, T$. On each iteration, a distribution of weights D_t is updated that indicates the importance of examples in the data set for the classification. On each round, the weights of each incorrectly classified example are increased, and the weights of each correctly classified example are decreased, so the new classifier focuses on the examples which have been misclassified by the previous classifiers.

For binary classifications, the algorithm is given input data samples, $x_i \in X$, corresponding labels $y_i \in \{-1; 1\}$, $i = 1, \ldots, n$, and a family of weak classifiers $\mathfrak{H}$. It

initiazises a weight $D_i = 1$ for each sample. Then, for each iteration, the algorithm proceeds as described below.

1. A weak classifier is selected from a provided family $\mathfrak{H}$ that minimizes the weighted error rate over the training dataset:

$$h_t = \underset{h_t \in \mathfrak{H}}{\operatorname{argmax}} \ |0.5 - \epsilon_t|$$

$$\epsilon_t = \frac{\sum_{i=1}^{n} D_i I(h_t(\vec{x}_i) y_i < 0)}{\sum_{i=1}^{n} D_i}$$

2. Set $\beta_i = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

3. Update for all i: $D_i = D_i e^{-\beta_i y_i h_t(\vec{x_i})}$. This step decreases the weigths of the successfully classified samples, and decreases the weight of misclassified ones.

After T iterations, the resulting strong classifier can be calculated by Eq. (2.5) Adaboost can be seen as a minimization of the

2.2.4 Other boosting algorithms

Boosting algorithms mainly differ in te way they esmimate weights β_i. Some of them prioritize misclassified examples, same as the AdaBoost, while others, like BrownBoost =ref=, attempt to increase robustness to noise and outliers by "giving up", and decreasing weights of the samples that has been repeatedly misclassified.

While our proposed learning technique uses AdaBoost as the basis for sample weighting, it is flexible enough to be easily adjusted to other methods.

2.3 Online classification algorithms

As mentioned above, many of the offline classification algorithms suffer from superlinear computational costs in the number of training samples. As the amount of training data increases, the training quickly becomes unfeasible. Also, with the increasing amount of common devices capable of data acquisition, such as mobile phones wit video cameras, etc, real-time classification tasks that allow for the adaptation to the incoming data stream and operate on a limited amount of data available dureing a single frame become

more and more relevant. This is one of the reasons for the development of our proposed algorithm and sample application.

In this section therefore, we introduce the algorithms specifically developed or modified for the online setting, that can be used as the replacement of the algorithms described above.

2.3.1 Online SVM

In this section, we review two algorithms for online SVM training that have a direct bearing on out work. Both of these algorithms use a variant of the Stochastic Gradient Descent in order to upadate the solution on each iteration.

2.3.1.1 NORMA

NORMA =ref= is a generic method for onilne risk minimization using stochastic gradient descent, with a particular focus on its application to kernel-based Support Vector Machines and regression. In the founding paper, Kivinen et.al. both describe the method and give theoretical bounds on its accuracy and convergence rate. They show that the convergence rate of NORMA is independent of the size of the dataset, if a limited dataset is used, and is instead of the order $O(\frac{X}{\epsilon^2})$, where X is the bound on the absolute value of the kernel function, and ϵ is an error rate. They also show that the truncation error resulting from removing older kernel expansion vectors decreases exponentially with the number of preserved vectors-coefficient pairs, as long as the kernel fuction values over the space of input data samples are bounded.

The agorithm itself attempts to minimize the primal formulation of an SVM problem (Eq. (2.4)) rewritten to allow the usage of the kernel trick. To do that, the concept of Reproducible Kernel Hilbert Spaces (RKHS) with defined inner product is used to replace $\vec{w}$ of the linear SVM with the function f int the RKHS $\mathscr{H}$ defined by kernel $k(\cdot,\cdot)$

$$\min_{f\in\mathscr{H}} \left\{ \frac{\lambda}{2}||f||^2_{\mathscr{H}} + \frac{1}{n}\sum_{i=1}^{n} l(f(\vec{x}_i, y_i)) \right\} \tag{2.6}$$

where $||f||^2_{\mathscr{H}} = f \cdot f$ However, in the online setting, assuming that a single data sample $\vec{x}_t$ and label y_t are provided on the iteration t, and other data samples unavailable Eq.

(2.6) instead takes the form of instantanious risk:

$$\min_{f \in \mathscr{H}} \left\{ \frac{\lambda}{2} ||f||^2_{\mathscr{H}} + l(f(\vec{x}_t, y_t)) \right\} \tag{2.7}$$

The above expression is then minimalized by utilising a simple subgradient descent (gradient if the function l is differentiable) with a learning rate η_t, i.e., on each iteration the function f is updated:

$$f_{t+1} = f_t - \eta_t \partial_f \left(\frac{\lambda}{2} ||f||^2_{\mathscr{H}} + l(f(\vec{x}_t, y_t)) \right)$$

which can be simplified to

$$f_{t+1} = (1 - \eta_t) f_t - l'_f(f(\vec{x}_t, y_t) k(\cdot, \vec{x}_t) \tag{2.8}$$

Since as long as the kernel used satisfies Mercer's condition, function f can be represented in the form already mentioned above, i. e., on the iteration t

$$f(\vec{x}) = \sum_{i=1}^{t} \alpha_i k(\vec{x}, \vec{x_i}) \tag{2.9}$$

Substituting Eq. (**??**) Fexpansion) into Eq. (2.8) , the final formula dealing with the updates of α_i becomes:

$$\alpha_t = -\eta_t l'(f(\vec{x}_t, y_t) \quad i = t$$

$$\alpha_i = (1 - \eta_t)\alpha_i \quad i < t$$

Here one of the drawbacks of the kernel bases-methods becomes obvious, since it can be seen that on each iteration an additional kernel expansion vector is added, quickly increasing the storage requirements.

2.3.1.2 Pegasos

Pegasos algorithm, introduced in =ref=, is conceptually similar to the NORMA, with two key differences, one being that it incorporates an additional parameter k, which denotes the number of input samples accepted at a single iteration, and averages the loss function in Eq. (2.7) over k. The other key difference is the additional scaling step after each iteration, which drastically increases the convergence rate.

$$f_{t+1} = min(1, \frac{1}{\sqrt{\lambda ||f_{t+0.5}||^2_{\mathscr{H}}}}) f_{t+0.5}$$

Using these steps allow Pegasos to effectively use agressive update schedule $\eta_t = \frac{1}{t}$, and achieve conergence rates proportional to $O(\frac{1}{\epsilon})$. Unlike NORMA, pegasos is primarily oriented toward linear SVM optimization. This fact, combined with the increased convergence rate and simplicity of implementation lead to us adopting it over NORMA as a basis of our method.

2.3.2 Online boosting

Online boosting algorithms, such as the ones presented in =ref=, =ref=, are a modification of AdaBoost approach with the exact error rate of the classifier ϵ_t being replaced with the online estimate $\tilde{\epsilon_t}$ that is updated on each iteration. In essense, they propose rerunning AdaBoost algorithm on limited subsets of weak classifiers called selectors on each iteration. Since this method is not essential to the understanding of our proposed algorithm, we shall refrain from desribing it in further detail here.

2.4 Overview of applications

Below are several examples of the practical applications of the described algorithms. While the applications for large scale and online SVM learning and boosting are spanning most of the areas of modern research that deal with large amounts of data, the examples presented below are most relevant to our study topic.

- **Object recognition**

  As mentioned in =ref= and briefly described in =ref=, the above mentioned methods or theor derivatives can be used to quickly and adaptively organize various image features (HOG in the case of =ref=, or an ensemble of simple features in =ref=) to reliably detect certain objects. The adaptive nature of such algorithms allows the detection to remain stable even for an object with changing form.

- **Object tracking** Similar to above, only in this case the original position of an object is given beforehand and has to be updated each frame as it moves.As shown in =ref=, online boosting for simple features, in particular, seems to be well suited for such a task.

- **Text classification** Text classification, i.e. the task of defining whether the text belongs to a certain cathegory or not, often has to be updates online, as the user inserts new texts or updates old labels. The online SVM-based methods are well suited for such tasks both due to adaptability and their ability to process large amounts of data easily. Due to near-linearity of many text classification based tasks, Pegasos is better suited to such applications than Norma.

- **Advertisement selection** Once again, a modification of the above mentioned text classification, this application deals with determining whether a certain link is relevant or not to a particular user based on his history of previous searches and network surfing.

3

Description of the proposed two-step algorithm

3.1 Derivation of the algorithm

In this section, we describe in detail the way our proposed algorithm was conceived and derived.

3.1.1 Drawbacks of the existing online algorithms

The method described in chap. 2 represent, in a certain way, two extremes of a spectrum. On one hand, the SVM training algorithms like Pegasos and NORMA may be used for either linear classification, which is fast and has low memory requirements, but has the drawback of being of limited utility, since the majority of data distributions in the natural datasets is nonlinear, or nonlinear kernel-based classification, which is imprecise in the case the optimal kernel for the data distribution is unknown, and, even in the optimal case, often exhibits growth of computational complexity by accumulating kernel expansion coefficients, that is roughly proportional to half the the processed data (see =fig==). The reason for this growth is partially explained by the fact that the provided data points cannot perfectly describe distribution of data in the transformed feature space, and thus the algorithms has to compensate with the larger number of samples to provide approximation (see =fig=). In this way, the kernel versions of the SVM training algorithms represent algorithms with unlimited complexity growth over time.

On the other hands, online boosting algorithms presented in several sources (for example, see =ref=, =ref==, etc) usually adopt the approach of a fixed complexity opposite to that of the offline boosting by fixing the number of classifiers being added to the model at the beginning of the training, which limits the possibly accuracy. They are also limitied by the need to estimate and update a large number of pre-selected classifiers on each iteration, although this problem is somewhat mitigated by the simplicity of the classifiers.

Our goal in this paper is to develop a middle-ground classifier that can provide adjustable levels of complexity growth or decay depending on the needs of the application. For that reason, we took a look at a structure of a single AdaBoost update and have adapted it to the online setting by using SVM training as a method of choosing the next classifier.

3.1.2 Similarity of AdaBoost and linear SVM

We shall start the explanation of our algorithm by noting the similarity between a classifier resulting from a linear SVM training and strong classifier of AdaBoost (this similaity being analyses in more detail in =ref=). It is easy to see, that Eq. (2.2.1) and Eq. (2.5) are identical, both being the sign value of a confidence function, which in turn is a linear combination of an input. The main difference is in the fact that in case of boosting the original inputs $\vec{x}_i$ were transformed by a set of classifiers to a kind of bimary-valued feature space. However, one cannot assume that this makes boosting completely equivalent to the kernel-based SVM, since the *sign* function, as well ass sigmoid function, the extreme case of which it is does, in fact, not satisfy Mercer condition, this fact proven in =ref=. This makes direct application of kernel SVM-related methods unpredictable. However, if we were to treat a vector of weak classifier outputs $h_t(\vec{x})$ as a kind of input vectors to the linear SVM classifier, we could use online SVM adaptation to iteratively change the weights β_i in the Eq. (2.5) .

3.1.3 Adapting AdaBoost to online setting by using modified Pegasos algorithm

AdaBoost iteration consists, essentially, from the two parts - the selection of the weak classifier $h_t(\vec{x})$ minimizing error in respect to the weights provided by the iteration's strong classifier, and selection of the weight for that classifier.

In the online setting, however, we do not have the option of evaluating each datapoint for each classifier to determine the optimum, although error values on the unweighted dataset can be estimated simply by adding the error of each consecutive iteration. It is also not possible to reach back and reevaluate data points when the weights of the classifiers placed earlier in the AdaBoost iteration chain change due to the changed accuracy estimate. The solution to this problem employed in =ref= is to both fix the number of classifiers and to separate them into unrelated sets called selectors (although the latter requirement is relaxed in the later articles =ref=), resulting in the totality of classifiers in each selector to slowly adapt their accuracy values according to their position in the booster chain.

Our solution, however, is to add a single classifier (linear SVM) at a time and then train it according the the weights provided by the current strong classifier, which is also being trained to better fit the global data distribution. This training can be achieved in the variety of ways, but the similarity between the Adaboost and the SVM described in the previous section has lead us to consider one of the online SVM training algorithm for adjusting the boosting weights. Given the simplicity and the higher convergence rate we have chosen Pegasos as our basis training method.

The online setting also naturaly raises the question of calculating the weight of each incoming data point for training the additional classifier. To answer this, we consider the weights provided by the AdaBoost algorithm at a given iteration t

$$D_{i,t} = e^{-\beta_1 h_1(x_i) y_i} e^{-\beta_2 h_2(x_i) y_i} \cdots e^{-\beta_t h_t(x_i) y_i} = e^{(} - y_i F(\vec{x_i}))$$

That is, the weight for each data point is a constant e to the power of the confidence function of the currently available multiplied by the opposite of a given label, which results in a positive value in case the current version of the strong classifier misclassifies the sample and negative if the available classification if correct. Since in our algorithm the we have constant access to the confidence function of the updated strong classifier, the calculation of the weight is relatively straightforward. Several loss functions can be used, including exponential function as in AdaBoost, or the hinge loss function (Eq. (2.2.1)) common to SVM, with varying results, which will be explored later in more detail.

function PEGASOS($\vec{w}, \lambda, \vec{x}, y, t_w, c$)
$l = \max(0, 1 - \vec{w} \cdot \vec{x}y)$
$\sigma = I(l > 0)$
$\eta = \frac{1}{\lambda t}$
$\vec{w} = (1 - \eta\lambda)\vec{w} + c\sigma\eta y\vec{x}$
$\vec{w} = min\left(1, \frac{1}{||\vec{w}||_2\sqrt{\lambda}}\right)\vec{w}$
end function

Figure 3.1: The Pegasos algorithm with weighted samples. Parameters: $\vec{w}$ - weight vector for the linear SVM, λ - regularization parameter, $\vec{x}$ -input training sample, y - training label, t_w - number of current iteration for weak classifier, c - weight parameter

3.1.4 Linear SVM as weak classifiers

The last step in defining our algorithm is the definition of the weak classifier. In our work, we use a linear SVM for that purpose. Even a random linear classifier would provide an error rate differing from random classification value of 0.5, in all but extremely rare degenerate cases, which is the only requirement for the weak classifier in AdaBoost (see =fig= for illustration).

The accuracy of the SVM is then improved by it being trained online by the same Pegasos algorithm as the global classifier with a single difference being an adaptation to the weights provided by the global classifier. Pseudocode and required parameters for the weighted for Pegasos iteration are shown on fig. 3.1. It should be noted that this is not the only way to incorporate weights into the online update algorithm. For example, =ref=, as well as =ref= propose using several updates in a row with the number being drawn according to the poisson distribution. However, we find our adaptation much simpler, and sufficient in terms of accuracy.

3.2 Description of the resulting algorithm

3.2.1 Algorithm description

The proposed algorithm can the be summarized as follows:

- It takes 3 parameters, λ, λ_H, and parameter r regulating the complexity growth.
- On each iteration, it takes the following inputs: number of iteration t, the number of iterations the weak classifier has been trained t_w, $\vec{x}$, y, and the outputs of the

previous iterations: vectors $\vec{w}_k$, $k = 1 \cdots K$, K being the number of SVM serving as weak classifiers already incorporated into a strong classifier, and $\vec{\beta} = \beta_1 \cdots \beta_K$ as boosting coefficients for weak classifiers. On the first iteration, $\vec{w}_1 = \vec{0}$, $K = 1$, $\beta_1 = 1$ that is, the classifier being trained is considered incorporated into a strong classfier.

- Outputs of weak classifiers are calculated and accumulated into a vector $\vec{h} = h_1 \cdots h_K$, $h_k = sign(\vec{w}_k \cdot \vec{x})$
- Confidence function for the resulting global classified is calculated and used to estimate the weight for the training of weak classifier: $F(\vec{h}) = \vec{\beta} \cdot \vec{h}$, $c = l(F(\vec{h})$, l defined as a hinge-loss function (Eq. (2.2.1))
- Weak classifier currently in training (defined by vector $\vec{w}_K$) is updated according to the algorithm illustrated on fig. 3.1.
- Vector β of the strong classifier is updated using Pegasos algorithm without weight modification, using the calculated confidence value F and regularization parameter λ_H.
- Parameters t and t_w are increased: $t = t + 1$, $t_w = t_w + 1$.
- If $t_w > r$, the values of $\vec{w}_K$ are fixed and a new weak classifier is initialized: $K = K + 1$, $\vec{w}_K = \vec{0}$, $\beta_K = 0$. Since the classifier is initialized with 0, it does not start to affect strong classifier H until the next update.

Furthermore, to increase the flexibility of the resulting algorithm, both the weak and the strong classifiers are assumed to incorporate a bias term into an input vector, according to the method described in =ref=, that is, both $\vec{x}$ and $\vec{\beta}$ are assumed to have an additional element equal to 1.

From the above definition, it is easy to see that at each iteration our algorithm has a string classifier ready, same as =ref=. The additional parameter r is used to control the complexity and non-linearity of the output, although the experiments have shown that for most non-linear application, the constant value of r about 100 is sufficient. Higher values of r result in a fewer number of better-trained classifiers combined into a strong classifiers, limiting possible task complexity, while lower result in the weak

classifiers being closer to random and increase complexity without appreaciable incease in accuracy for most datasets.

The result is an algorithm with controllable storage and computational requirements that can be used for online training of strong classifiers on nonlinear data distribution without the use of kernels, and therefore without the need to adjust or fine-tune kernel parameters. Experiments in section =secref= show tha a constant value of parameters works well enough for most complex datasets, proving the simplicity and large applicability range of proposed method.

The convergence of our meethod on a sample 2-dimentional dataset is illustrated on =fig=

3.2.2 Possible modifications

The above algorithm can be modified in several ways to adapt for a specific application requirements or for increased flexibility

Different loss functions for weak classifier training. The function l used to calculate weight for weak classifier training can be changed to, essentially, any monotonically increasing function. Several loss functions popular in optimization are shown on =fig=.

Removal of weak classifiers While our algorithm allows for accuracy levels comparable or higher to that of kernel-based SVM, with much lower computational complexity, computational and storage costs of our algorithm still grow with input samples. One way to stop this growth is to remove those weak classifiers absolute values of coefficients for which fall below a certain threshold, resulting in eventual replacement of ill-fitting classifiers. This technique also increases the flexibility of the method

Limiting the global iteration number t Since the learning rate of the algorithm decreases proportionally to teration number t, for applications that deal with constantly changing data distribution it is beneficial to limit the growth of the number t, stopping the increase at certain threshold corresponding to the level of flexibility desired.

Varying number and kind of weak classifiers trained simultaneously Since the trainer for strong classifier is unaware of a kind and number of weak classifiers being trained, different features may be added when each new weak classifier is initialized, and several distinct classifiers may be trained at once, especially if the setting favors

parallel processing. This particular modification is discussed in more detail in the next section.

3.2.3 Comparison to other algorithms

Online SVM trainning methods (NORMA and Pegasos) Our methods compares favorably to both NORMA and Pegasos in case of nonlinear data distribution, since they allow for much lower computational and storage requirements. If, however, the parameter r is set to values near 1, our method is reduced to, essentially, fitting a set of random classifiers to data distribution, with the storage and computational requirement approaching those of kernel methods. However, the extimated computational requrements are still somewhat lower that for most commonly used kernels due to the simplicity of the sign function.

For the case of linarly separable data, our algorithm is essentially indistinguishable from Pegasos, on which it is based, with additional overhead due to added classifiers.
Online boosting methods It is difficult to compare our method to the online boosting methods due to the difference in methodology. In many ways, our methods are quite opposite. Methods proposed in =ref= apply learning updates to the preselected set of classifiers based on random features, while our method updates usually only one classifier per iteration. Also, their algorithm assigns weights to classifiers in a manner similar to Adaboost, all at the same iteration, whle ours adjusts the weights iteratively over time.

3. DESCRIPTION OF THE PROPOSED TWO-STEP ALGORITHM

4

Application of proposed method to object tracking on a mobile device

4. APPLICATION OF PROPOSED METHOD TO OBJECT TRACKING ON A MOBILE DEVICE

5

Evaluation of related algorithms

6

Conclusion