Improved Boosting Performance by Explicit Handling of Ambiguous Positive Examples

Miroslav Kobetski and Josephine Sullivan

Abstract Visual classes naturally have ambiguous examples, that are different depending on feature and classifier and are hard to disambiguate from surrounding negatives without overfitting. Boosting in particular tends to overfit to such hard and ambiguous examples, due to its flexibility and typically aggressive loss functions. We propose a two-pass learning method for identifying ambiguous examples and relearning, either subjecting them to an exclusion function or using them in a later stage of an inverted cascade. We provide an experimental comparison of different boosting algorithms on the VOC2007 dataset, training them with and without our proposed extension. Using our exclusion extension improves the performance of almost all of the tested boosting algorithms, without adding any additional test-time cost. Our proposed inverted cascade adds some test-time cost but gives additional improvements in performance. Our results also suggest that outlier exclusion is complementary to positive jittering and hard negative mining.

Keywords Boosting · Image classification · Algorithm evaluation · Dataset pruning · VOC2007

1 Introduction

Recent efforts to improve image classification performance have focused on designing new discriminative features and machine learning methods. However, some of the performance gains of many well-established methods are due to dataset augmentation such as hard negative mining, positive mirroring and jittering [1–4]. These data-bootstrapping techniques aim at augmenting sparsely populated regions of the dataset to allow any learning method to describe the class distributions more

M. Kobetski (✉) · J. Sullivan
Computer Vision and Active Perception, KTH, 114 28 Stockholm, Sweden
e-mail: kobetski@kth.se

J. Sullivan
e-mail: sullivan@nada.kth.se

© Springer International Publishing Switzerland 2015

A. Fred and M. De Marsico (eds.), *Pattern Recognition Applications and Methods*,
Advances in Intelligent Systems and Computing 318,
DOI 10.1007/978-3-319-12610-4_2

accurately, and they have become standard tools for achieving state-of-the-art performance for classification and detection tasks. In this paper we revisit the dataset augmentation idea, arguing and showing that pruning the positive training set by excluding hard-to-learn examples can improve performance for outlier-sensitive algorithms.

We focus on the boosting framework and propose a method to identify and exclude positive examples that a classifier is unable to learn—to make better use of the available training data rather than expanding it. We refer to the non-learnable examples as outliers and we wish to be clear that these examples are not label noise (such as has been studied in [5–7]), but rather examples that with a given feature and learner combination are ambiguous and too difficult to learn. We also propose an inverted cascade that allows inclusion of these hard examples at a later stage of the classification.

One of the main problems with most boosting methods is their sensitivity to outliers such as atypical examples and label noise [5, 8–10]. Some algorithms have tried to deal with this problem explicitly [6, 10–14], while others, such as LogitBoost [15] are less sensitive due to their softer loss function.

The boosting methods with aggressive loss functions give outliers high weight when fitting the weak learner, and therefore potentially work poorly in the presence of outliers. Softer loss functions as seen in the robust algorithms can on the other hand result in low weights for all examples far from the margin, regardless if they are noisy outliers or just data to which the current classifier has not yet been able to fit. This can be counter-productive in cases of hard inliers, which is illustrated in Fig. 2a. Another problem that soft loss functions are not able to solve, is that outliers are still able to affect the weak learners during the early stages of the training, which due to the greedy nature of boosting can only be undone later on by increasing the complexity of the final classifier. In this paper we provide an explicit analysis on how various boosting methods relate to examples via their weight functions and we argue that a distinct separation in the handling of inliers and outliers can help solve these problems that current robust boosting algorithms are facing.

Following this analysis we propose our two pass boosting algorithm extension, that explicitly handles learnable and non-learnable examples differently. We define outliers as examples that are too hard-to-learn for a given feature and weak learner set, and identify them based on their classification score after a first training round. A second round of training is performed, where the outliers are subjected to a much softer loss function and are therefore not allowed to interfere with the learning of the easier examples, in order to find a better optimum. This boosting algorithm extension consistently gives better test performance, with zero extra test-time costs at the expense of increased training time. Some examples of found inliers and outliers can be seen in Fig. 1. We also propose a method for reintroducing the hard examples without reducing the performance of the classifier. We call this an inverted cascade.

Fig. 1 Examples of outliers and inliers. The top rows of (a) and (b) show outliers while the bottom rows show inliers. We focus on how to detect the outliers and how their omission from training improves test results. The images are from the VOC2007 dataset. **a** Chair class, **b** Bottle class

1.1 Relation to Bootstrapping Methods

To further motivate our data-centric approach to learning, we illustrate the problems that different dataset augmentation techniques address. In regions where the positive training examples are dense and the negatives are existent but sparse, hard negative mining might improve the chances of finding the optimal decision boundary. In regions where positives are sparse and negatives existent, jittering and mirroring might have some effect, but the proper analogue to hard negative mining is practically much harder, since positive examples need to be labelled. At some scale this can be done by active learning [16], where labelling is done iteratively on selected examples.

Our approach tries to handle the regions where positives are sparse but additional hard positive mining is not possible, either due to limited resources or because all possible positive mining has already been done. We address this problem by restricting hard-to-learn positives from dominating the training with their increasingly high weights by excluding them from the training.

Our algorithm can be considered as dataset pruning and makes us face the high-level question of more data versus better data—something that has recently been addressed by [17]. It has been shown that in cases where huge labelled datasets are available, even simple learning methods perform very well [18–20]. We address the opposite case, where a huge accurately labelled data set cannot be obtained—a common scenario both in academic and industrial computer vision.

1.2 Contributions

We propose a two-pass boosting extension algorithm, suggested by a weight-centric theoretical analysis of how different boosting algorithms respond to outliers. We also demonstrate that it is important to distinguish between "hard-to-learn" examples and "non-learnable" outliers in vision as examples easily identified as positive by humans could be non-learnable given a feature and weak-learner set, and demonstrate that the different classes in VOC2007 dataset indeed have different fractions of hard-to-learn examples using HOG as base feature. We also propose the inverted cascade—that allows the hard positive examples to be re-introduced at a later stage. Finally we provide extensive experimental comparison of different boosting algorithms on real computer vision data and perform experiments using dataset augmentation techniques, showing that our method is complementary to jittering and hard negative mining.

2 Relation to Previous Work

As previously mentioned AdaBoost has been shown to be sensitive to noise [8, 9]. Other popular boosting algorithms such as LogitBoost or GentleBoost [15] have softer loss functions or optimization methods and can perform better in the presence of noise in the training data, but they have not been specifically designed to handle this problem. It has been argued that no convex-loss boosting algorithm is able to cope with random label noise [5]. This is however not the problem we want to address, as we focus on naturally occurring outliers and ambiguous examples, which is a significant and interesting problem in object detection today.

BrownBoost [11] and RobustBoost [10] are adaptive extensions of the Boost-By-Majority Algorithm [21] and have non-convex loss functions. Intuitively these algorithms "give up" on hard examples and this allows them to be less affected by erroneous examples.

Regularized LPBoost, SoftBoost and regularized AdaBoost [14, 22] regularize boosting to avoid overfitting to highly noisy data. These methods add the concept of soft margin to boosting by adding slack variables in a similar fashion to soft-margin SVMs, and this decreases the influence of outliers. Conceptually these methods bear some similarity to ours as the slack variables reduce the influence of examples on the wrong side of the margin, and they define an upper bound on the fraction ν of misclassified examples, which is comparable to the fraction of the dataset excluded in the second phase of training.

There is recent work on robust boosting where new semi-convex loss functions are derived based on probability elicitation [6, 7, 12]. These methods have shown potential for high-noise problems such as tracking, scene recognition and artificial label noise. But they have not been extensively compared to the common outlier-sensitive algorithms on low-noise problems, such as object classification, where the existing outliers are ambiguous or uncommon examples, rather than actual label errors.

In all the mentioned robust boosting algorithms the outliers are estimated and excluded on the fly and these outliers are therefore able to affect the training in the early rounds. Also, as can be seen in Fig. 2, these algorithms can treat uncommon non-outliers as conservatively as actual outliers, resulting in suboptimal decision boundaries.

Reducing overfitting by pruning the training set has been studied previously [23, 24] but improved results have mostly been seen in experiments where training sets include artificial label noise. Reference [23] is the only method that we have found where pruning improves performance on a "clean" dataset. They use an approach very similar to ours, detecting hard-to-learn examples, then removing those examples from training. The base algorithm to which [23] apply dataset pruning is AdaBoost, which we show is the most noise-sensitive boosting algorithm and not the one that should be used for image classification. We propose a similar but more direct approach that improves results for both robust and non-robust algorithms, while still using a reasonable number of weak learners.

It is important to note that vision data is typically very high-dimensional and boosting therefore also acts as feature selection—learning much fewer weak learners than available dimensions. This is one of the key differences to typical machine learning datasets, such as the ones used for validating the method in [23]. Our experiments on the VOC2007 dataset verify that exclusion of ambiguous examples is useful for the high-dimensional problems found in computer vision. We also compare a number of well-known boosting algorithms using typical vision data, something that we have not seen previously.

A different but related topic that deals with label ambiguity is Multiple Instance Learning (MIL). Viola et al. [25] suggest a boosting approach to the MIL problem, applying their solution to train an object detector with highly unaligned training data.

Our idea is also conceptually similar to a simplified version of self-paced learning [26]. We treat the hard and easy positives separately and do not let the hard examples dominate the easy ones in the search for the optimal decision boundary. This can seen

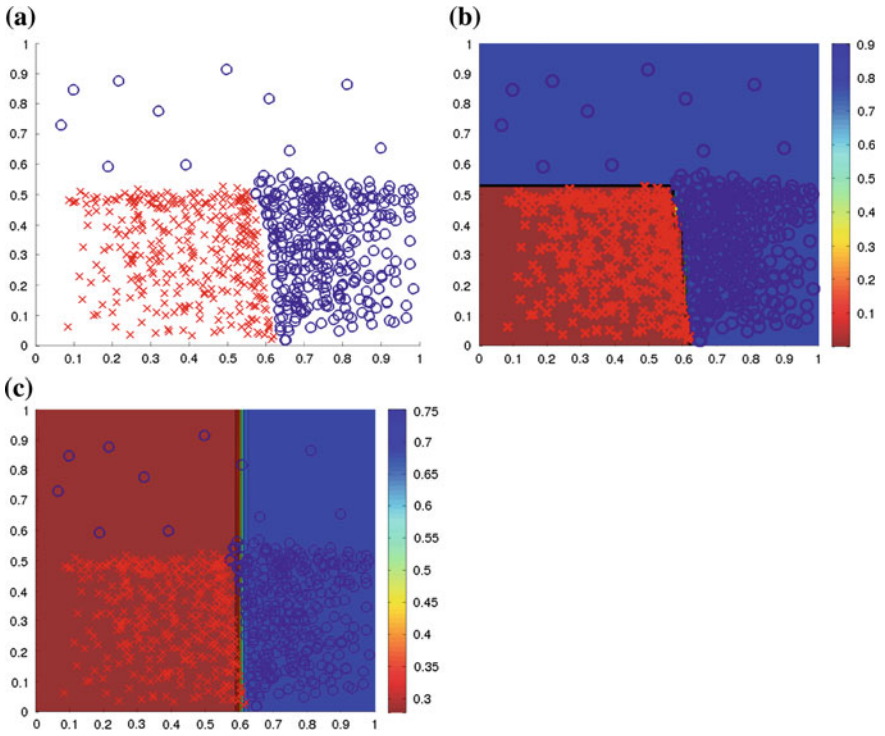

Fig. 2 Example with hard inliers. This toy problem shows how less dense, but learnable examples do not contribute to the decision boundary when learned using RobustBoost. The colour coding represents estimated probability $p(y = 1|x)$. (Best viewed in colour.) **a** Toy example without outliers, **b** Learnt AdaBoost classifier, **c** Learnt RobustBoost classifier

as a heavily quantized version of presenting the examples to the learning algorithm in the order of their difficulty.

3 Boosting Theory

Boosted strong classifiers have the form $H_m(x) = \sum_i^m \alpha_i h(x; \beta_i)$, where $h(x)$ is a weak learner, with multiplier α_i and parameters β_i. To learn such a classifier one wishes to minimize the average loss $\frac{1}{N} \sum_{j=1}^{N} L(H(x_j), y_j)$ over the N input data points (x_j, y_j) where each data label $y_j \in \{-1, 1\}$. Learning the classifier that minimizes the average loss by an exhaustive search is infeasible, so boosting algorithms do this in a greedy stepwise fashion. At each iteration the strong classifier is extended with the weak learner that minimizes the loss given the already learned strong classifier

$$\alpha^*, \beta^* = \underset{\alpha,\beta}{\operatorname{argmin}} \frac{1}{N} \sum_{j=1}^{N} L(H_m(x_j) + \alpha h(x_j; \beta), y_j). \tag{1}$$

Equation 1 is solved by weighting the importance of the input data by a weight function $w(x, y)$ when learning α and β. This $w(x_j, y_j)$ represents how poorly the current classifier $H_m(x_j)$ is able to classify example j.

Different boosting algorithms have different losses and optimizations procedures, but the key mechanism to their behaviour and handling of outliers is the weight function $w(x, y)$. For this reason we believe that analyzing the weight functions of different losses gives an insight to how different boosting algorithms behave in the presence of hard and ambiguous examples. So in order to compare a number of boosting algorithms in a consistent framework we re-derive $w(x, y)$ for each of the algorithms by following the GradientBoost approach [27, 28].

The GradientBoost approach views boosting as a gradient based optimization of the loss in function space. According to the GradientBoost framework a boosting algorithm can be constructed from any differentiable loss function, where each iteration is a combination of a least squares fitting of a weak regressor $h(x)$ to a target $w(x, y)$

$$\beta^* = \underset{\beta}{\operatorname{argmin}} \left(\sum_j \left(w(x_j, y_j) - h(x_j; \beta) \right)^2 \right), \tag{2}$$

and a line search $\alpha = \operatorname{argmin}_\alpha (L(H(x) + \alpha h(x; \beta)))$ to obtain α. The loss function is derived with respect to the current margin $v(x, y) = yH(x)$ to obtain the negative target function

$$w(x, y) = -\frac{\partial L(x, y)}{\partial v(x, y)}. \tag{3}$$

Equation 2 can then be interpreted as finding the weak learner that points in the direction of the steepest gradient of the loss, given the data.

3.1 Convex-Loss Boosting Algorithms

3.1.1 Exponential Loss Boosting

AdaBoost and GentleBoost [15, 29] are the most notable algorithms with the exponential loss

$$L_e(x, y) = \exp(-v(x, y)). \tag{4}$$

AdaBoost uses weak classifiers for $h(x)$ rather than regressors and directly solves for α, while GentleBoost employs Newton-step optimization for the expected loss. In the original algorithms $w(x, y)$ is exponential and comes in via the weighted fitting

of $h(x)$, but we obtain

$$w_e(x, y) = \exp(-v(x, y)), \tag{5}$$

from the GradientBoost approach to align all analyzed loss functions in the same framework. $w_e(x, y)$ has a slightly different meaning than the weight function of the original algorithms since it is the target of a non-weighted fit, rather than the weight of a weighted fit. However, its interpretation is the same—the importance function by which an example is weighted for the training of the weak learner $h(x)$. Also, it should be noted that we have omitted implementation-dependent normalization of the weight function.

3.1.2 Binomial Log-Likelihood Boosting

LogitBoost is a boosting algorithm that uses Newton stepping to minimize the expected value of the negative binomial log-likelihood

$$L_l(x, y) = \log(1 + \exp(-2v(x, y))). \tag{6}$$

This is potentially more resistant to outliers than AdaBoost or GentleBoost as the binomial log-likelihood is a much softer loss function than the exponential one [15].

Since the original LogitBoost optimizes this loss with a series of Newton steps, the actual importance of an example is distributed between a weight function for the weighted regression and a target for the regression—both varying with the margin of the example. We derive $w(x, y)$ by applying the GradientBoost approach to the binomial log-likelihood loss function to collect the example weight in one function

$$w_l(x, y) = \frac{1}{1 + \exp(v(x, y))}. \tag{7}$$

Figure 3a shows the different weight functions and suggests that LogitBoost should be affected less by examples far on the negative margin than the exponential-loss algorithms.

3.2 Robust Boosting Algorithms

3.2.1 RobustBoost

RobustBoost is specifically designed to handle outliers [10]. RobustBoost, a variation of BrownBoost, is based on the Boost-by-Majority algorithm and has a very soft and non-convex loss function

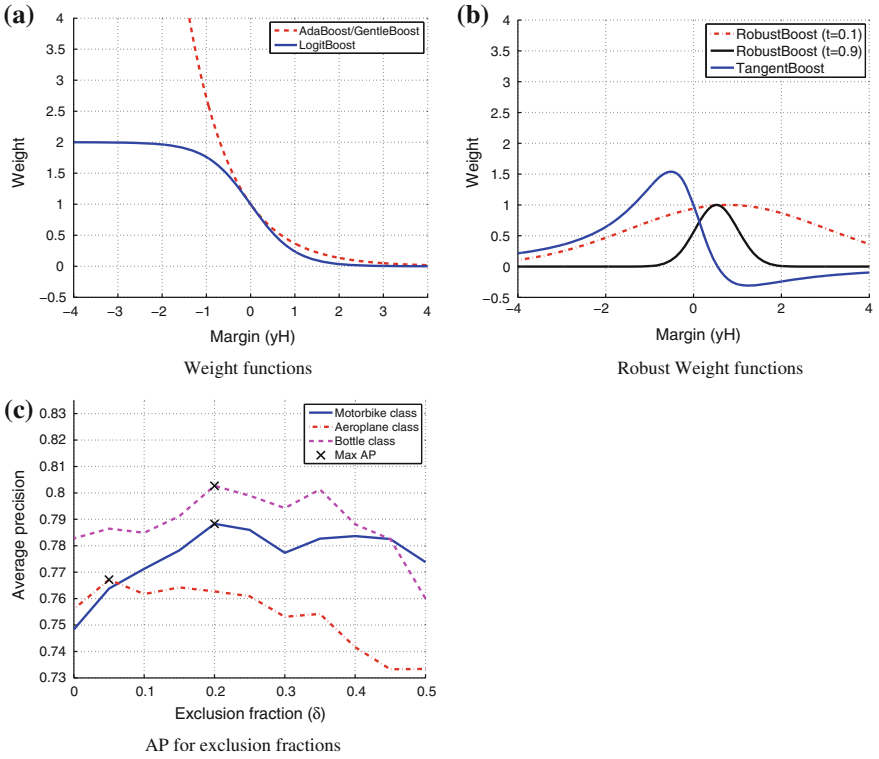

Fig. 3 **a, b** Weight functions with respect to the margin. This illustrates how much examples at different distances from the margin are able to affect the decision boundary for the different algorithms. **c** Performance for different exclusion fractions δ. Average precision on the test set, using the two-pass extension with LogitBoost for different exclusion fractions δ and different classes. This figure illustrates that different classes have different optimal exclusion fractions δ

$$L_r(x, y, t) = 1 - \text{erf}\left(\frac{v(x, y) - \mu(t)}{\sigma(t)}\right), \tag{8}$$

where $\text{erf}(\cdot)$ is the error function, $t \in [0, 1]$ is a time variable and $\mu(t)$ and $\sigma(t)$ are functions

$$\sigma^2(t) = (\sigma_f^2 + 1)\exp(2(1 - t)) - 1 \tag{9}$$

$$\mu(t) = (\theta - 2\rho)\exp(1 - t) + 2\rho, \tag{10}$$

with parameters θ, σ_f and ρ. Equation 8 is differentiated with respect to the margin to get the weight function

$$w_r(x, y, t) = \exp\left(-\frac{(v(x, y) - \mu(t))^2}{2\sigma(t)^2}\right). \tag{11}$$

Figure 3b shows Eq. 11 for some values of t. From these we can see the RobustBoost weight function changes over time. It is slightly more aggressive in the beginning and as $t \to 1$, it focuses less and less on examples far away from the target margin θ. One interpretation is that the algorithm focuses on all examples early in the training stage, and as the algorithm progresses it starts ignoring examples that it has not been able to push close to the target margin.

RobustBoost is self-terminating in that it finishes when $t \geq 1$. In our experiments we follow Freund's example and set $\sigma_f = 0.1$ to avoid numerical instability for t close to 1 and we obtain the parameters θ and ρ by cross-validation.

3.2.2 TangentBoost

TangentBoost was designed to have a positive bounded loss function for both positive and negative large margins, where the maximum loss for large positive margins is smaller than for large negative margins [12]. To satisfy these properties the method of probability elicitation [6] is followed to define TangentBoost to have a tangent link function

$$f(x) = \tan\left(p(x) - 0.5\right), \tag{12}$$

and a quadratic minimum conditional risk

$$C_L^*(x) = 4p(x)(1 - p(x)), \tag{13}$$

where $p(x) = \arctan(H(x)) + 0.5$. is the intermediate probability estimate. Combining the above equations results in the Tangent loss

$$L_t(x, y) = \left(2\arctan\left(v(x, y)\right) - 1\right)^2. \tag{14}$$

We immediately see that the theoretical derivation of TangentBoost and its implementation may have to differ as the probability estimates $p(x) \in [-\pi/2+0.5, \pi/2+0.5]$ are not proper, so that we only have proper probabilities $p(x) \in [0, 1]$ for $|H(x)| < 0.546$. This means that $|H(x)| > 0.546$ has to be handled according to some heuristic, which is not presented in the original paper [12]. In the original paper the Tangent loss is optimized through Gauss steps, which similarly to LogitBoost divides the importance of examples into two functions. So as with the other algorithms we re-derive $w_t(x, y)$ by using the GradientBoost method, and obtain

$$w_t(x, y) = -\frac{4\left(2\arctan\left(v(x, y)\right) - 1\right)}{1 + \left(v(x, y)\right)^2}. \tag{15}$$

As seen in Fig. 3b this weight function gives low weights for examples with large negative margin, but it also penalizes large positive margins by assigning negative weight to very confident examples. Since $w_t(x, y)$ is actually the regression target

this means that the weak learner tries to fit very correct examples to an incorrect label.

4 A Two-Pass Exclusion Extension

Our main point is that some fraction of the data, that is easy to learn with a given feature and weak-learner set, defines the core shape of the class—we call these examples inliers. Then there are examples that are ambiguous or uncommon so that they cannot be properly learned given the same representation. Trying to do so might lead to overfitting, create artefacts or force a poorer definition of the shape of the core of the class. We call these examples non-learnable or outliers and illustrate their effect on training in Fig. 4. It is important to note that there might be hard examples with large negative margin during some parts of training, but that eventually get learned without overfitting. We refer to these examples as hard inliers, and believe they are important for learning a well performing classifier.

Figure 4 illustrates that even if robust algorithms are better at coping with outliers, they are still negatively affected by them in two ways; The outliers still have an effect on the decision boundary learnt, even if their effect is reduced. Hard inliers are also subject to the robust losses, thus having less influence over the decision boundary than for non-robust losses, illustrated in Fig. 2.

We propose that outliers and inliers should be identified and handled separately so that the outliers are only allowed to influence the training when already close to the decision boundary and therefore can be considered as part of the core shape of the class. This can be achieved with a very soft loss function, such as the logistic loss or the Bayes consistent Savage loss [6]. We use the logistic loss, since the Savage loss gives more importance to slightly misclassified examples, rather than being symmetric around the margin.

Differentiating the logistic loss

$$L_s(x, y) = \frac{1}{1 + \exp\left(-\eta v(x, y)\right)}, \tag{16}$$

with respect to the margin results in the weight function

$$w_{excl}(x, y) = \eta\, \sigma(-\eta\, v(x, y))\, \sigma(\eta\, v(x, y)) \tag{17}$$

where $\sigma(\cdot)$ is the sigmoid function. This weight function can be made arbitrarily thin by increasing the η parameter. We call this function the *exclusion function*, as its purpose is to exclude outliers from training.

Since the inlier examples are considered learnable we want the difficult examples in the inlier set to have high weight, according to the original idea of boosting. For this reason all inliers should be subjected to a more aggressive loss such as the exponential loss or the binomial log-likelihood loss.

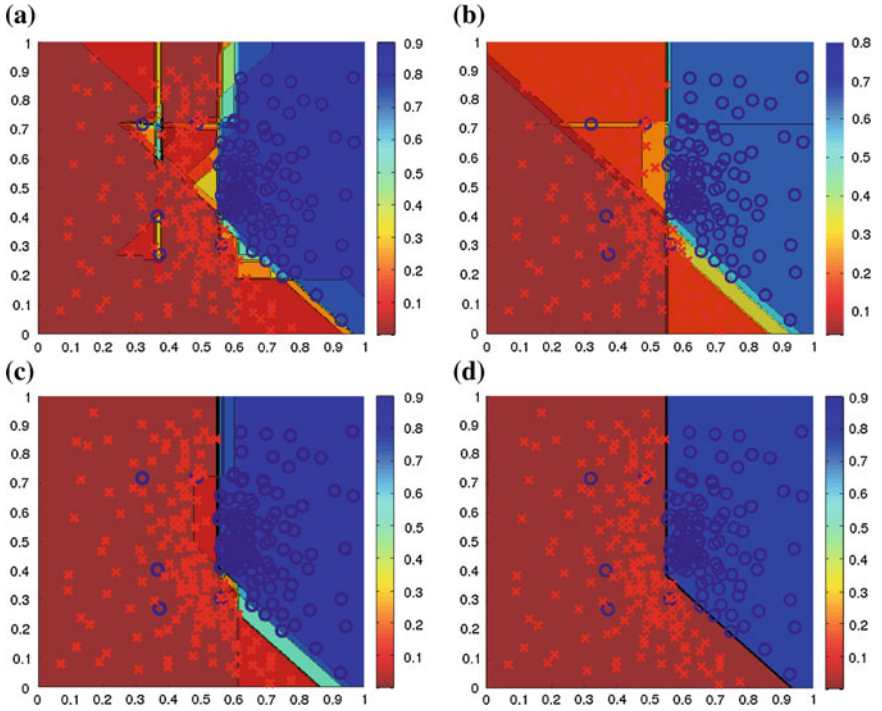

Fig. 4 Example with five outliers. The decision boundaries the different algorithms produce in the presence of a few outliers. The colour coding represents estimated probability $p(y = 1|x)$. We can see how AdaBoost overfits to the outliers, TangentBoost overfits slightly less and RobustBoost is able to handle the problem, even if it is less certain around the boundary. Applying the two-pass method to this problem results in a decision boundary that completely ignores the outliers. (Best viewed in colour.) **a** AdaBoost, **b** TangentBoost, **c** RobustBoost, **d** AdaBoost with two-pass method

The main challenge is to identify the outliers in a dataset. To do this we follow our definition of outliers as non-learnable and say that they are the examples with the lowest confidence after completed training. We therefore define the steady-state difficulty $d(x_j)$ of the examples as their negative margin $-v(x_j, y_j)$ after a fully completed training round, and normalize to get non-negative values.

$$d(x_j) = \begin{cases} \max(H(x)) - H(x_j) & \text{if } y_j = 1 \\ H(x_j) - \min(H(x)) & \text{if } y_j = -1, \end{cases} \tag{18}$$

where $H(x_j)$ is the classification score of example j. This is referred to as the first pass.

We order the positive examples according to their difficulty $d(x_j)$ and re-train the classifier, assigning a fraction δ of the most difficult examples to the outlier set and subjecting them to the logistic loss function. This second iteration of training

is what we call the second pass. Figure 1 shows some inlier and outlier examples for the bottle and chair classes. As we have mentioned, what will be considered an outlier depends on the features used. We use HOG in our experiments [2], so it is expected that the tilted bottles and the occluded ones are considered outliers, since HOG cannot capture such variation well.

As previously mentioned, our model for outlier exclusion has two parameters: δ and η, where δ controls how many examples will be considered as outliers and η controls how aggressively the outlier examples will be down-weighted. In our experiments we choose a large value for η—effectively ignoring outliers completely in the second round. The actual fraction of outliers is both class and feature dependent, so δ needs to be properly tuned. We tune δ by cross-validation, yet we have noticed that simple heuristics seem to work quite well too. Figure 3c shows how the performance is affected by δ for three different classes. We can clearly see that different classes have different optimal values for δ, which is related to the number of outliers in their datasets, given the used features and learners.

4.1 Inverted Cascade

The excluded examples are not necessarily label noise, but are defined as examples that increase the cross-validation error, they can still hold valuable information about the class. One way of making use of this information without reducing the performance of the exclusion-type classifier is to learn separate classifiers, and take the *max* of their classification scores as output. Since there is a difficulty ordering between the two classifiers, it is more natural to approach this problem as an inverted cascade. An inverted cascade is a cascade where each positive classification aborts the cascade instead of negative classifications as in standard cascades. Each negative classification continues to the next classifier. This way the learnable examples can be correctly classified by the first classifier while hard false negatives can be subjected to stricter classification criteria in a later cascade stage.

5 Experiments

We perform experiments on a large number of classes to reduce the influence of random performance fluctuations. For this reason, and due to the availability of a test set we select the VOC2007 dataset for our experiments. Positive examples have bounding-box annotations, so we crop and resize all positive examples to have the same patch size. Since our main focus is to investigate how our two-pass exclusion method improves learning, we want to minimize variance or tweaking in the areas not related to learning, and therefore choose a static non-deformable feature. We use the HOG descriptor [2] to describe the patches, since it has shown good performance

in the past, is popular within the vision community, and its best parameter settings have been well established.

To get the bounding boxes for the negative examples, we generate random positions, widths and heights. We make sure that box sizes and aspect ratios are restricted to values that are reasonable for the positive class. Image patches are then cropped from the negative training set according to the generated bounding boxes. The patches are also resized to have the same size as the positive patches, after which the HOG features are computed. We then apply our two-pass training procedure described in Sect. 4 to train a boosted stump classifier.

We train boosted stumps using four different boosting algorithms: AdaBoost, LogitBoost, TangentBoost and RobustBoost. AdaBoost and LogitBoost are chosen for their popularity and proliferation in the field. TangentBoost and RobustBoost are chosen as they explicitly handle outliers. We also train an SVM classifier to use as reference.

For LogitBoost, TangentBoost and AdaBoost there are no parameters to be set. For RobustBoost two parameters have to be tuned: the error goal ρ and the target margin θ. We tune them by holdout cross-validation on the training set. With TangentBoost we encountered an implementation issue, due to the possibility of negative weights and improper probability estimates. After input from the author of [12] we manually truncate the probability estimates to make them proper. Unfortunately this forces example weights to zero for margins $|v(x, y)| > 0.546$, which gives poor results as this quickly discards a large portion of the training set. To cope with this and to obtain reasonable performance we lower the learn rate of the algorithm. The linear SVM is trained with *liblinear* [30], with normalized features and using 5-fold cross-validation to tune the regularization term C.

Jittering the positive examples is a popular way of bootstrapping the positive dataset, but we believe that this can also generate examples that are not representative of that class. For this reason our two-pass approach should respond well to jittered datasets. We therefore redo the same experiments for the best performing classifier, augmenting the positive sets by randomly generating 2 positive examples per labelled positive example, with small random offsets in positions of the bounding box. We also mine for hard negatives to get a complete picture of how the outlier exclusion extension interacts with bootstrapping methods.

6 Results

A summary of our results is that all boosting algorithms except TangentBoost show consistent improvements for the experiments using our two-pass extension, which can be seen in Table 1. Before employing our two-pass extension LogitBoost performs best with 11 wins over the other algorithms. After the two-pass outlier exclusion LogitBoost dominates even more with 15 wins over other outlier-excluded algorithms and 13 wins over all other algorithms, including LogitBoost without

Table 1 Performance of our experiments

Class	Mean average precision for each algorithm																	
	Adaboost			LogitBoost			RobustBoost			TangentBoost			Linear SVM			LogitBoost inverted cascade		
	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Result	Improvement
Plane	**74.27**	73.87	−0.40	73.13	**74.09**	0.97	**72.49**	72.23	−0.27	**73.86**	72.88	−0.98	**74.04**	72.01	−2.03	79.03	5.90
Bike	87.59	**88.24**	0.66	87.85	**88.43**	0.58	86.86	**87.98**	1.12	**86.81**	86.80	−0.01	**85.12**	84.47	−0.66	88.57	0.72
Bird	46.69	**48.14**	1.45	48.08	**49.87**	1.79	46.66	**48.69**	2.03	46.97	**49.45**	2.48	**46.67**	46.32	−0.34	51.05	2.97
Boat	57.33	**58.96**	1.63	59.01	**60.81**	1.80	57.92	**58.31**	0.39	**58.61**	58.02	−0.59	**57.00**	56.94	−0.06	63.46	4.45
Bottle	74.57	**78.56**	3.99	76.02	**80.00**	3.98	72.74	**78.94**	6.20	76.79	**78.32**	1.53	76.22	**76.97**	0.75	80.23	4.21
Bus	**86.79**	86.18	−0.62	86.54	**86.96**	0.42	83.86	**86.82**	2.96	85.97	**86.26**	0.29	**82.07**	81.85	−0.22	88.63	2.09
Car	87.05	**87.73**	0.68	**88.32**	**88.32**	0.00	**88.26**	88.14	−0.11	**88.34**	88.30	−0.04	**86.22**	85.93	−0.29	89.45	1.13
Cat	**49.45**	49.18	−0.27	50.15	**52.44**	2.29	48.30	**52.21**	3.91	45.93	**50.90**	4.97	45.45	**45.48**	0.02	54.79	4.64
Chair	67.68	**69.36**	1.68	68.26	**69.93**	1.66	66.06	**68.91**	2.85	67.42	**68.09**	0.67	**64.08**	63.94	−0.14	71.69	3.43
Cow	81.90	**83.25**	1.35	81.99	**83.85**	1.87	81.83	**82.71**	0.87	**81.50**	80.63	−0.87	82.01	**82.44**	0.43	85.32	3.33
Table	40.89	**44.50**	3.61	43.68	**47.91**	4.24	39.89	**46.26**	6.37	**47.52**	36.34	−11.18	**28.35**	28.09	−0.26	50.95	7.27
Dog	47.83	**51.56**	3.73	51.27	**51.68**	0.41	47.66	**52.24**	4.58	**51.25**	50.21	−1.04	46.16	**46.94**	0.78	56.24	4.97
Horse	76.09	**76.10**	0.01	78.83	**78.84**	0.01	77.72	**78.28**	0.56	75.20	**76.20**	1.00	75.55	**75.77**	0.21	78.83	0.00
Motorbike	74.15	**77.15**	3.00	77.39	**78.33**	0.94	76.75	**79.20**	2.45	75.16	**75.46**	0.30	**73.31**	72.56	−0.76	79.31	1.92

(continued)

Table 1 (Continued)

Class	Mean average precision for each algorithm																
	Adaboost			LogitBoost			RobustBoost			TangentBoost			Linear SVM			LogitBoost inverted cascade	
	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Use all	Exclude	Diff	Result	Improvement
Person	58.56	**63.64**	5.08	65.16	**67.03**	1.87	60.17	**65.09**	4.92	66.04	**67.02**	0.98	60.86	**61.84**	0.98	67.89	2.73
Plant	**58.08**	57.60	−0.48	60.46	**60.51**	0.04	56.87	**59.62**	2.75	**59.28**	58.23	−1.05	58.42	**58.90**	0.48	60.10	−0.36
Sheep	80.60	**84.41**	3.81	81.59	**83.11**	1.53	**80.78**	80.11	−0.66	**84.38**	82.01	−2.37	80.07	**80.17**	0.10	84.49	2.90
Sofa	61.83	**66.28**	4.44	62.35	**66.66**	4.31	63.92	**67.81**	3.89	56.73	**63.06**	6.33	**62.33**	62.18	−0.15	67.50	5.15
Train	73.18	**76.91**	3.74	73.98	**76.74**	2.75	73.13	**77.26**	4.13	73.87	**74.74**	0.87	71.82	**72.10**	0.28	78.74	4.76
Tv	**92.11**	**92.11**	0.00	**92.57**	**92.57**	0.00	**91.97**	**91.97**	0.00	**91.69**	**91.69**	0.00	**91.28**	**91.28**	0.00	92.65	0.08
Mean	68.83	**70.69**	1.85	70.33	**71.90**	1.57	68.69	**71.14**	2.45	69.67	**69.73**	0.06	**67.35**	67.31	−0.04	73.45	3.11
WWA	4	15	–	0	18	–	3	16	–	9	10	–	10	9	–	–	–
WBA	2	1	–	11	15	–	1	4	–	5	0	–	1	0	–	–	–

Average Precisions of different classifiers, when applied to the VOC2007 test set. The gray box on each row indicates the best performing classifier for the object class. Boldface numbers indicate best within-algorithm-performance for excluding outliers or not. Gray cells indicate total best performance for a given class—not including the inverted cascade. *Wins within algorithm* (*WWA*) summarizes how often a learning method is improved by our extension, and *Wins between algorithms* (*WBA*) summarizes how often an algorithm outperforms the others when having the same strategy for handling outliers. Note that these results cannot be directly compared to results from the original VOC2007 challenge since we are performing image patch classification, using the annotated bounding boxes to obtain positive object positions. The inverted cascade achieves the highest performance and the last column shows its improvement over the base algorithm

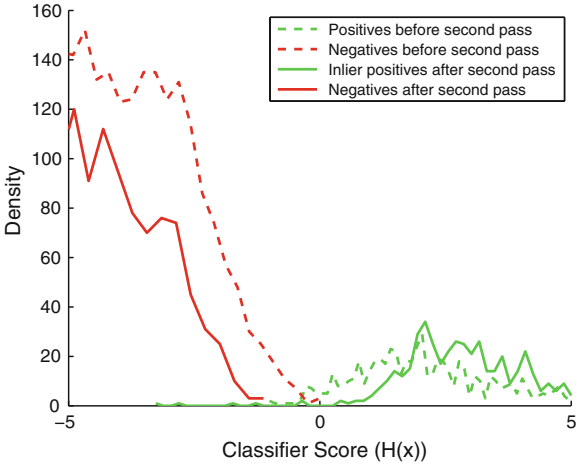

Fig. 5 Margin before and after outlier exclusion. The margin for the learnable examples becomes wider when using the outlier exclusion extension

outlier exclusion. Figure 5 shows an example of how the margin becomes much wider when using the extension.

The inverted cascade with LogitBoost performs the best overall with the most significant improvements, both over the base algorithm but also over the outlier-excluded one.

6.1 Comparison of Boosting Algorithms

Table 1 also includes a comparison of the performance between algorithms, showing performance differences with and without the outlier exclusion. Among the convex-loss algorithms LogitBoost performs better than AdaBoost. The difference in performance shrinks when our two-pass method is applied, which suggests that naturally occurring outliers in real-world vision data affects the performance of boosting algorithms and that those better able to cope with such outliers have a greater potential for good performance.

Even so, the robust algorithms perform worse than LogitBoost. One reason for this could be that the robust algorithms make no distinction between outliers and hard inliers, as previously discussed. Our two-pass algorithm only treats "non-learnable" examples differently, not penalizing unkommon learnable examples for being difficult in the early stages of learning.

Although RobustBoost has inherent robustness, it is improved the most by our extension. One explanation is its variable target error rate ρ, which after the exclusion of outliers obtains a lower value through cross-validation. RobustBoost with a small

value ρ is more similar to a non-robust algorithm, and should not suffer as much from the hard-inlier-problem demonstrated in Fig. 2c.

The SVM classifier is provided as a reference and sanity check, and we see that the boosting algorithms give superior results even though only decision stumps are used.

The higher performance of the boosted classifier is likely due to that combination of decision stumps can produce more complex decision boundaries than the hyperplane of a linear SVM. It is not surprising that the linear SVM is not improved by the outlier exclusion as it has a relatively soft hinge loss, tuned soft margins, and lacks the iterative reweighting of examples and greedy strategy, that our argumentation is based on. SVMs suffer from outliers too [17], but our method is not optimal for SVMs, since some of the excluded examples could be important support vectors.

6.2 Bootstrapping Methods in Relation to Outlier Exclusion

We can see in Table 2 that jittering has a positive effect on classifier performance and that our outlier exclusion method improves that performance even more. This shows that our two-pass outlier exclusion is complementary to hard negative mining and positive jittering and could be considered as a viable data augmentation technique when using boosting algorithms.

7 Discussion and Future Work

We show that all boosting methods perform better when handling outliers separately during training. As RobustBoost and TangentBoost do not reach the performance of LogitBoost they might be too aggressive in reducing the importance of hard inliers and not aggressive enough for outliers. We must remember that the problem posed by the VOC2007 dataset does not include label noise, but does definitely have hard and ambiguous examples that might interfere with learning the optimal decision boundary. RobustBoost and other robust boosting algorithms have previously shown good results on artificially flipped labels, but we believe that a more common problem in object classification is naturally occurring ambiguous examples and we have therefore not focused on artificial experiments where labels are changed at random.

We notice that the improved performance from excluding hard and ambiguous examples is correlated with the severity of the loss function of the method. AdaBoost, with its exponential loss function, shows large improvement while LogitBoost has less average gain and TangentBoost gains almost nothing from the exclusion of outliers. More surprising is that RobustBoost is improved the most, in spite of its soft loss function. One explanation is that there is an additional mechanism at work in improving the performance of RobustBoost. RobustBoost is self-terminating, stopping when it has reached its target error. When training on a dataset with a

Table 2 Outlier exclusion with other dataset augmentation techniques

	Mean average precision for each object class																			
Method	Plane	Bike	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	Motor-bike	Person	Plant	Sheep	Sofa	Train	Tv	Mean
Default	73.13	**87.85**	48.08	59.01	76.02	86.54	88.32	50.15	68.26	81.99	43.68	51.27	78.83	77.39	65.16	60.46	81.59	62.35	73.98	92.57	70.33
J	75.64	87.33	49.34	57.59	76.68	88.69	88.29	49.95	68.28	83.63	48.25	51.87	79.78	77.83	66.17	**63.59**	82.86	64.51	**78.08**	92.55	71.55
J + OE	77.18	87.56	50.30	60.32	79.54	**89.73**	89.62	50.64	68.75	84.73	50.39	**55.09**	79.33	**79.68**	**67.73**	62.83	83.71	**67.65**	75.79	92.55	72.66
J + HN	76.31	87.35	**50.85**	**61.10**	80.08	89.41	89.62	51.23	68.93	84.88	49.35	53.08	80.90	79.67	65.78	61.98	81.39	64.07	76.18	**92.98**	72.26
J + HN + OE	**77.73**	87.46	48.60	59.76	**80.67**	89.37	**90.07**	**53.76**	**70.11**	**86.55**	**51.24**	53.99	**80.92**	77.78	67.54	62.39	**85.60**	66.83	77.38	92.98	**73.04**

Positive jittering (J) and hard negative mining (HN) improves performance even more in combination with outlier exclusion (OE)

smaller fraction non-learnable examples, a lower target error is better suited and the cross-validation process will ensure that a lower target error is selected. This will result in later termination and an overall more aggressive loss function.

We have seen that pruning of the hard-to-learn examples in a dataset without label noise can lead to improved performance, contrary to the "more data is better" philosophy. This point has also been examined by [17], who demonstrate that removing perfectly fine examples can improve overall classification performance in the case where model flexibility is insufficient to adapt to all training data.

Our belief is that our method removes bad data, but also that it reduces the importance of examples that make the learning more difficult, in this way allowing the boosting algorithms to find better local minima. We still believe that more data is better if properly handled, so this first approach of selective example exclusion should be extended in the future, and might potentially combine well with positive example mining, especially in cases where the quality of the positives cannot be guaranteed.

8 Conclusions

We provide an analysis of several boosting algorithms and their sensitivity to outlier data. Following this analysis we propose a two-pass training extension that can be applied to boosting algorithms to improve their tolerance to naturally occurring outliers. We show experimentally that excluding the hardest positives from training by subjecting them to an exclusive weight function is beneficial for classification performance. Introducing an inverted cascade to re-include the hard positives we improve the performance even more. We also show that this effect is complementary to jittering and hard negative mining, which are common bootstrapping techniques.

The main strength of our approach is that classification performance can be improved without any extra test-time cost, only at the expense of training-time cost. If test time can be increased slightly the inverted cascade allows additional performance. We believe that handling the normal and hard examples separately might allow bootstrapping of training sets with less accurate training data.

We also present results on the VOC2007 dataset, comparing the performance of different boosting algorithms on real world vision data, concluding that LogitBoost performs the best and that some of this difference in performance can be due to its ability to better cope with naturally occurring outlier examples.

Acknowledgments This work has been funded by the Swedish Foundation for Strategic Research (SSF); within the project VINST

References

1. Felzenszwalb, P., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. PAMI (2010)
2. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR (2005)
3. Laptev, I.: Improving object detection with boosted histograms. IVC (2009)
4. Kumar, M.P., Zisserman, A., Torr, P.H.S.: Efficient discriminative learning of parts-based models. In: ICCV (2009)
5. Long, P.M., Servedio, R.A.: Random classification noise defeats all convex potential boosters. In: ICML (2008)
6. Masnadi-shirazi, H., Vasconcelos, N.: On the design of loss functions for classification: theory, robustness to outliers, and savageboost. In: NIPS (2008)
7. Leistner, C., Saffari, A., Roth, P.M., Bischof, H.: On robustness of on-line boosting—a competitive study. In: ICCV Workshops (2009)
8. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: bagging, boosting, and variants. MLJ (1999)
9. Dietterich, T.: An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. MLJ (2000)
10. Freund, Y., Science, C.: A more robust boosting algorithm. arXiv:0905.2138 (2009)
11. Freund, Y.: An adaptive version of the boost by majority algorithm. In: COLT (1999)
12. Masnadi-Shirazi, H., Mahadevan, V., Vasconcelos, N.: On the design of robust classifiers for computer vision. In: CVPR (2010)
13. Grove, A., Schuurmans, D.: Boosting in the limit: maximizing the margin of learned ensembles. In: AAAI (1998)
14. Warmuth, M., Glocer, K., Rätsch, G.: Boosting algorithms for maximizing the soft margin. In: NIPS (2008)
15. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. AOS (2000)
16. Vijayanarasimhan, S.: Large-scale live active learning: training object detectors with crawled data and crowds. In: CVPR (2011)
17. Zhu, X., Vondrick, C., Ramanan, D., Fowlkes, C.C.: Do we need more training data or better models for object detection? In: BMVC (2012)
18. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: CVPR (2011)
19. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: a large data set for nonparametric object and scene recognition. PAMI (2008)
20. Hays, J., Efros, A.: Scene completion using millions of photographs. TOG (2007)
21. Freund, Y.: Boosting a weak learning algorithm by majority. IANDC (1995)
22. Rätsch, G., Onoda, T., Müller, K.: Soft margins for AdaBoost. MLJ (2001)
23. Vezhnevets, A., Barinova, O.: Avoiding boosting overfitting by removing confusing samples. In: ECML (2007)
24. Angelova, A., Abu-Mostafam, Y., Perona, P.: Pruning training sets for learning of object categories. In: CVPR (2005)
25. Viola, P., Platt, J.: Multiple instance boosting for object detection. In: NIPS (2006)
26. Kumar, M.P., Packer, B.: Self-paced learning for latent variable models. In: NIPS (2010)
27. Friedman, J.: Greedy function approximation: a gradient machine. AOS (2001)
28. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent in function space. In: NIPS (1999)
29. Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. In: COLT (1995)
30. Fan, R., Chang, K., Hsieh, C., Wang, X., Lin, C.: LIBLINEAR: a library for large linear classification. JMLR (2008)

http://www.springer.com/978-3-319-12609-8

Pattern Recognition Applications and Methods
International Conference, ICPRAM 2013 Barcelona, Spain, February 15-18, 2013 Revised Selected Papers
Fred, A.; De Marsico, M. (Eds.)
2015, XV, 312 p. 124 illus., 70 illus. in color., Softcover
ISBN: 978-3-319-12609-8