Optimizing Cascade Classifiers

Brendan McCane MCCANE@CS.OTAGO.AC.NZ
Department of Computer Science
University of Otago, New Zealand

Kevin Novins NOVINS@CS.AUCKLAND.AC.NZ
Department of Computer Science
University of Auckland, New Zealand

Michael Albert MALBERT@CS.OTAGO.AC.NZ
Department of Computer Science
University of Otago, New Zealand

Editor: Leslie Pack Kaelbling

Abstract

We introduce a method for producing more efficient cascade classifiers. To do so, we empirically model the execution cost of AdaBoost classifiers with fixed height decision trees as the weak learners. Our empirical model is general enough to model many disparate problems. We use the empirical execution cost model to optimize the parameters required for building an optimal cost cascade classifier - rather than relying on ad hoc parameter settings. We demonstrate our method on several classification problems including rare event detection problems such as face detection and several non-rare event problems from the UCI repository. We show that cascading classifiers can provide significant speedups for non-rare event problems and our model provides good estimates for the attainable speedups.

Keywords: cascade classifiers

1. Introduction

In their influential papers, Viola and Jones (2001, 2004) describe a method for finding faces in images or video input. The method is very fast (achieving approximately 15 frames/sec on a 700MHz Pentium III), and quite accurate (according to Figure 7 in the paper, achieving greater than 90% correct detection rate for only 0.000053% false positives). One of the major contributions of their work is the use of a cascade of classifiers to significantly speed up run time classification. The use of a cascade is a very general idea and could be applied to a large class of detection problems, but has thus far been used only when the number of expected detections is small compared to the total number of samples — that is, rare event detection problems. Since the work of Viola and Jones (2001, 2004), several other authors have used cascaded classifiers for face detection and other tasks (Schneiderman, 2004; Grossmann, 2004; Wu et al., 2004; Lienhart et al., 2003; Zhang et al., 2004; Huang et al., 2004; Sun et al., 2004; McCane and Novins, 2003).

There are two main limitations to the work of Viola and Jones (2001, 2004). Firstly, the training algorithm they use is quite slow taking on the order of weeks according to the authors. Secondly, their method for determining the target false positive rate and detection rate at each stage in the cascade is ad hoc—it appears to be based on the experience of the developers. We focus predominantly on the second problem for producing near optimal cascades.

This paper makes several important contributions to the theory and practice of cascaded classifiers. We develop an empirical model of the performance of cascaded AdaBoost decision trees that takes into account both false positive and true positive rates (Viola and Jones (2001, 2004) only used false positive rates). Given this model, we show how significant speedups can be obtained for a wide class of classification problems — even for those problems which do not have a significantly larger proportion of negative examples. We demonstrate and provide source code for a method of automatically setting the parameters of the cascade rather than the ad hoc manner used by Viola and Jones (2001, 2004). We also provide general guidelines for manually setting such parameters. Minor contributions include the development of a simple optimization procedure for problems with product equality constraints, a method for estimating the execution cost of a cascaded classifier and the likely speedup obtained. Finally, we demonstrate our method on several classification problems.

2. Prior Work

As far as we are aware, only four papers address the problem of automatically learning the cascade parameters (Grossmann, 2004; Sun et al., 2004; McCane and Novins, 2003; Chen and Yuille, 2005). In Grossmann (2004) a single stage classifier is initially built using the standard AdaBoost algorithm. The weak classifiers used in the single classifier are then partitioned using dynamic programming to produce a cascaded classifier of near optimal speed with almost identical behaviour to the original classifier. This is a very elegant solution to the problem since there is no need to specify the number of levels in the cascade a priori. Further, different cascaded classifiers which perform at different points on the receiver operating characteristic (ROC) curve can easily and quickly be generated, allowing for significant flexibility of the final classifier. However, one of the reasons that the method of Viola and Jones (2001, 2004) could produce such good results is that new negative data could be sampled at each new cascade stage. This allowed the method to achieve very low false positive rates by covering a much larger portion of the input space than might otherwise have been possible. It is unclear whether a single boosted classifier could achieve similar results, although techniques such as that proposed by Chawla et al. (2004) might be applicable. Still, the practical efficacy of the method of Grossmann (2004) is yet to be established.

Sun et al. (2004) introduce the idea of cascade risk into a cost function which is then used to automatically determine the optimal trade-off between false positives and false negatives at each stage. The cost function includes a weighted sum of cascade risk and computational complexity where the weight is determined heuristically. As such, the optimization produces an heuristic trade-off between the most accurate classifier and classifier efficiency. The training algorithm in this case requires setting an upper limit on the number of simple features for each cascade stage so that the algorithm will terminate. The ROC curve produced was superior to that of Viola and Jones (2001, 2004), but no indication of classifier efficiency was given in the paper. At best, their work includes classifier efficiency only heuristically as this is not their main focus.

Chen and Yuille (2005) solve the problem in a greedy manner. However they seek to build a cascade which satisfies a given time constraint, and so they choose the next classifier in the cascade as one which leaves the maximum time available for later stages, subject to the number of negatives eliminated from consideration. It is clear that this method is not optimal as one could imagine a situation where a zero-cost classifier with a false positive rate of 1.0 is chosen as the next one.

Further, they require that the detection rate of the overall classifier is fixed to 1.0 on the training data, and do not consider problems where that condition may be relaxed.

In previous work, two of us (McCane and Novins, 2003) introduced a technique for producing optimal speed cascade classifiers by modeling the relationship between classifier cost and false positive rate assuming a fixed detection rate. We extend this work in three ways in this paper. Firstly, we apply the idea of cascaded classifiers to problems which are not necessarily "rare event", and show that significant speedups can still be obtained. Secondly, we model classifier cost as a function of both false positive rate and detection rate which allows for increased flexibility in producing fast classifiers. This is discussed in Section 5. Finally, we use the results of previous cascade layers at the current boosting level. This guarantees the cascaded classifier will produce similar results to an equivalent monolithic classifier.

3. Method Overview

Our method for building near optimal cascades involves four basic stages:

1. Incrementally build a monolithic AdaBoost classifier in the usual way, but generate ROC curves for the classifier as each new simple classifier is added to the ensemble,
2. Estimate a cost model from the family of ROC curves,
3. Estimate parameters for an optimal cascade using the cost model,
4. Build the final near-optimal cascade classifier using either an existing monolithic classifier (from stage 1), or building the cascade anew.

Each of these steps is discussed in the following sections.

4. Generating ROC curves

A monolithic AdaBoost classifier is constructed in the usual way (Freund and Schapire, 1996) up to a maximum length. We have found that 100 boosting rounds is typically enough to adequately perform the modeling. As each new weak classifier is added to the ensemble, an ROC curve is generated by adjusting the AdaBoost threshold to achieve a given detection rate on an independent test set. This results in a family of ROC curves that illustrate the evolution of the AdaBoost classifier as more weak classifiers are added. Figure 1 shows an example for the covtype data set. This family of ROC curves creates an approximation to an ROC surface. We show in the next section how we can model the ROC surface.

5. Cost Model

As noted in the introduction the goal of using a cascade of classifiers is to improve the average run speed of the classifier. Suppose that cascade stage i accepts the proportion p_i of examples passed to it, where:

$$p_i = (f_i n_f + d_i n_p)/(n_f + n_p) \tag{1}$$

where f_i is the false positive rate of cascade stage i, n_f is the number of false examples, d_i is the detection rate of cascade stage i, and n_p is the number of positive examples. Note that this is

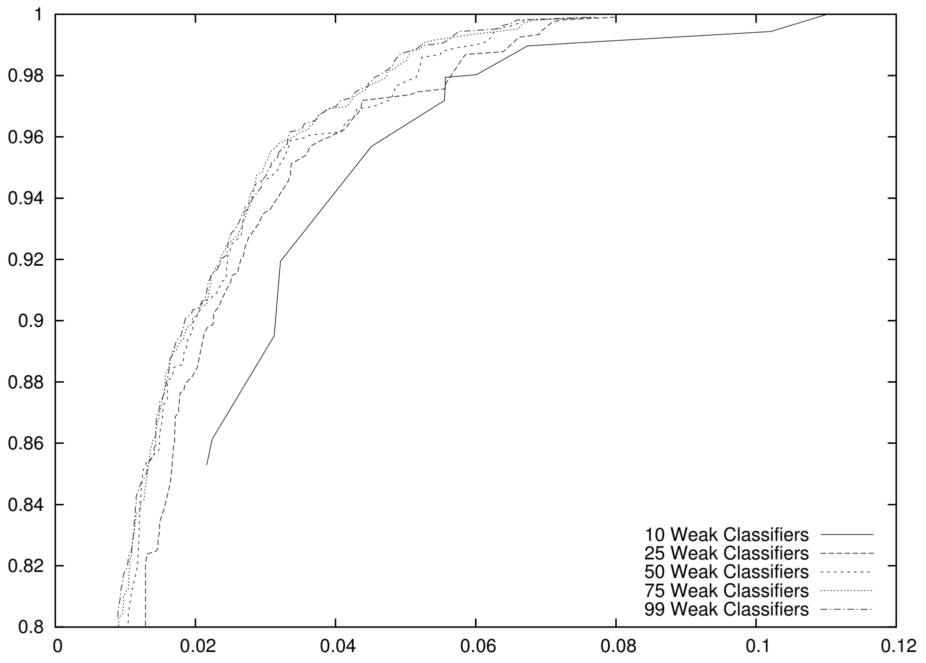


Figure 1: A family of ROC curves for the covtype data set.

a more complete model than that used by Viola and Jones (2001, 2004) since they assumed that $n_p \approx 0$, which is valid for rare event detection such as face detection, but not valid for more general classification problems. Associated with each stage is a certain cost of execution, C_i. Stage 1 is always executed. Stage 2 is executed only for those examples that successfully passed through stage 1. Similarly, stage 3 is only executed for those examples which successfully passed through stages 1 and 2, and so on. This then gives us a clear expression for the average cost of execution of the cascade:

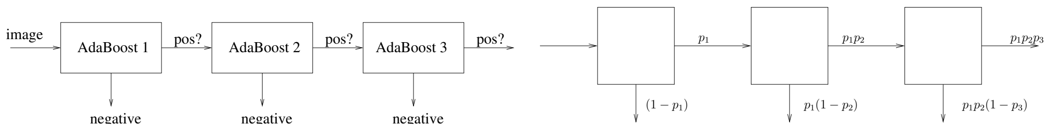
$$\begin{aligned} C_a &= C_1 + C_2 p_1 + C_3 p_1 p_2 + ... && (2) \\ &= C_1 + \sum_{i=2}^{N} \left[C_i \prod_{j=1}^{i-1} p_j \right] && (3) \end{aligned}$$

Figure 2(b) gives a pictorial view of how this works. We seek to minimize C_a, subject to relevant constraints, to find the optimal speed classifier.

5.1 The Cost of AdaBoost

To minimize C_a we need a model for the execution cost of each of the cascade stages C_i. In other words, we need to model the cost of an AdaBoost classifier. Freund and Schapire (1997) proved that the training error of an AdaBoost classifier is bounded above by:

$$E = \exp(-2G(T)) \quad (4)$$



(a) A classifier cascade (b) Cost of executing the classifier

Figure 2: Classifier cascades

where $G(T) = \sum_{t=1}^{T} \gamma_t^2$, T is the number of weak classifiers in the ensemble, $\gamma_t = 1/2 - \epsilon_t$, and ϵ_t is the training error of weak classifier t.

We have found that $G(T)$ can be approximated by:

$$G(T) \approx a(t-b)^c \tag{5}$$

where $a > 0$, $b > 0$, and $0 < c < 1.0$ are appropriate constants. Figure 3 shows examples of $G(t)$ on several of the problems from the UCI data repository. In this paper, we have restricted our study to weak learners which are height restricted decision trees, with heights of 1 (decision stumps), 2 or 3. Note that the approximation used in Equation 5 has similar properties to the function $G(T)$ - in particular the gradients of both approach 0 as T approaches infinity.

Now γ (or ϵ) is actually a combination of the false positive and the false negative rates. We are interested in not just modeling the error of AdaBoost with respect to the computational cost of the classifier, but rather the more detailed function of computational cost with respect to the false positive and false negative rates. In other words, we need to generate ROC curves for each cost point of an AdaBoost classifier. ROC curves can be generated for AdaBoost classifiers by altering the decision threshold of the ensemble as in Viola and Jones (2001, 2004). Therefore, for any AdaBoost classifier, we can choose an appropriate detection rate or false positive rate by adjusting the threshold appropriately. If we fix the detection rate, then the false positive rate will vary with cost in a similar way to the overall error rate as in Figure 3.

Ultimately, we seek a cost function that is dependent on both the false positive rate and the detection rate:

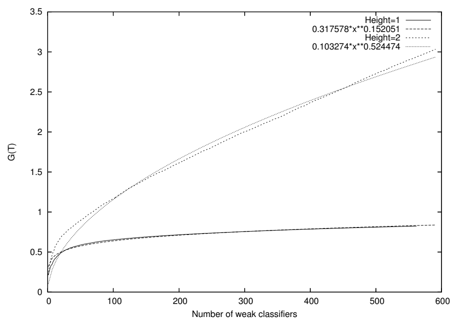
$$C_i = g(f,d),$$

where g is an appropriate two-dimensional function, f is the false positive rate and d is the detection rate of layer i in the cascade. Unfortunately, it is extremely difficult to directly estimate a two-dimensional function using non-linear regression. To overcome this problem, for each fixed detection rate, we estimate the function parameters independently and then perform a second regression on the parameters of the first function. For a fixed detection rate, we use a logarithmic function:

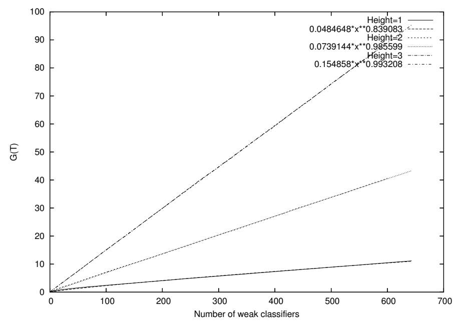
$$g_d(f) = a_d + (\log(f)/(-c_d))^{1.0/b_d}, \tag{6}$$

where g_d is the cost, a_d, c_d and b_d are our constant parameters that must be estimated, and the subscript d is used to indicate that these parameters are in fact dependent on the detection rate. For ease of modeling, we can invert Equation 6 to give the false positive rate as a function of the cost:

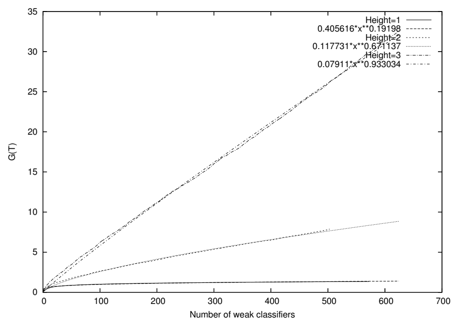
$$f(g_d) = e^{-c_d(g_d-a_d)^{b_d}}. \tag{7}$$



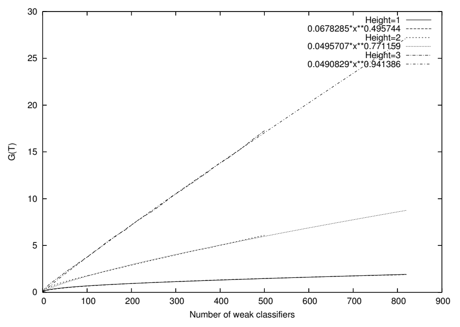
(a) Covtype



(b) Isolet

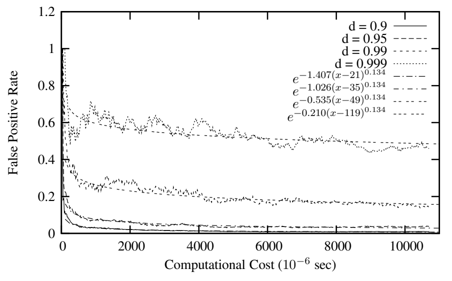


(c) Letter

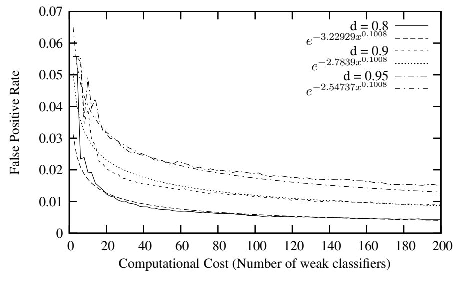


(d) Musk

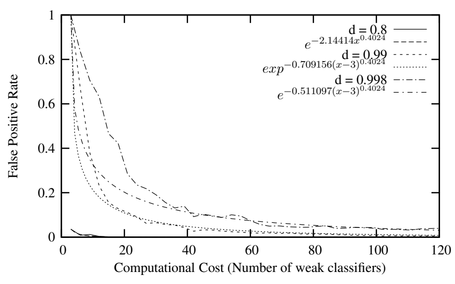
Figure 3: Model fit for the sum of gammas function. The data is shown using AdaBoost with fixed height decision trees on several examples in the UCI data repository. Where the problem is a multi-class one, one of the classes is chosen as the class to be detected and all others are considered negative examples.



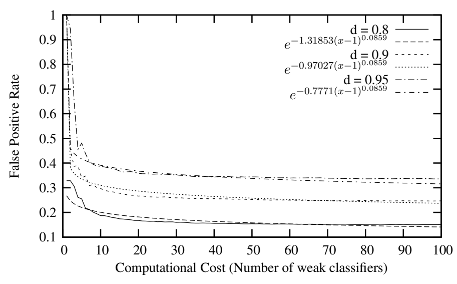
(a) Faces



(b) Covtype



(c) Letter



(d) Splice

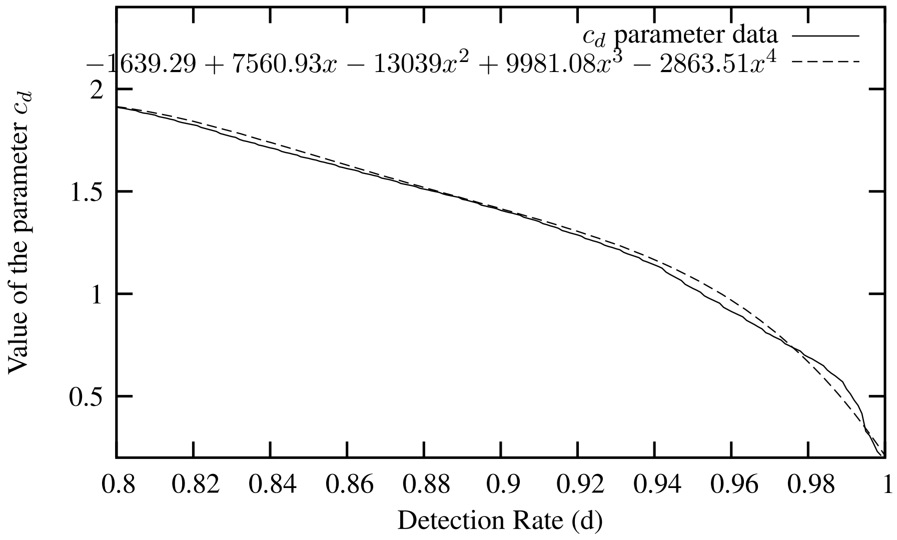
Figure 4: Cost as a function of false positive rate, for several fixed detection rates.

Note that this equation is of the same form as the combination of equations 4 and 5. The parameter a_d simply shifts the curve along the axis, and can be estimated directly from the data as the cost point, or number of weak classifiers, where the false positive rate first begins to drop, and is therefore a discrete parameter. For high detection rates, this may take several weak classifiers, and hence a_d increases as the detection rate increases. We estimate a_d in this way to simplify the non-linear regression problem, similar to an expected-value parameter (Ratkowsky, 1990).

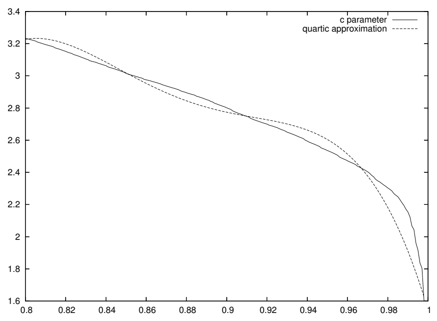
The parameter b_d is best estimated as a constant, so we perform two rounds of regression - the first is used to estimate b_d and the second to estimate c_d given a fixed b_d. The problem is solved using the Levenberg-Marquardt method (in our case, using Mathematica's NonLinearRegress function Wolfram Research (2003)). Figure 4 shows several curves at different detection rates and their associated best fit curves. The curves shown here are generated from an independent test set and therefore show test error rather than training error. Consequently, we can see some examples of overtraining, but the general form of the approximation appears to remain valid. To further improve the generality of the model, a nine-fold cross-validation with a random partition into a 50% training and 50% test set is used where feasible (for the face detection problem this isn't feasible due to the long training times). The false positive rate displayed in feasible cases is simply the average rate across the different runs.

Figure 5 shows how the parameter c_d varies as the detection rate increases. We model this curve with a quartic function using linear regression.

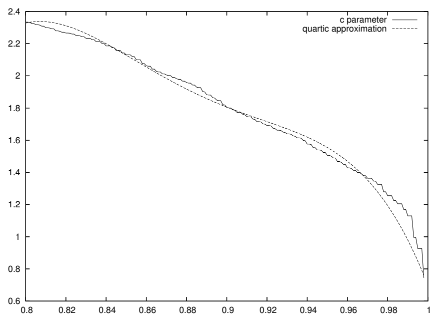
Modeling the a_d parameter from Equation 7 is more difficult since it is a discrete parameter. We need our 2D model to be continuous to allow for optimization however, so we use a sum of logistic



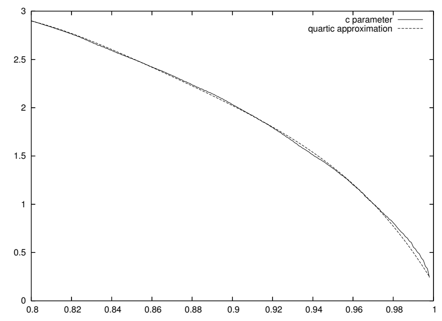
(a) Faces



(b) Covtype



(c) Letter



(d) Splice

Figure 5: The variation of parameter c_d with respect to the detection rate for various classification problems. The data contains 200 plot points.

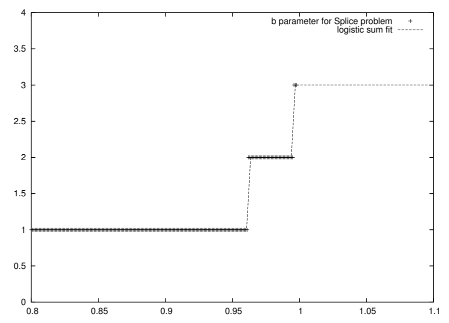


Figure 6: The variation of the parameter a with respect to the detection rate for the Splice classification problem.

step functions of the form:

$$a_d = \alpha \sum_{i=1}^{N} \left[\beta_i/(1.0 + \exp(-\sigma_i * (x - \delta_i)))\right],$$

where α is the starting height, β_i is the step size, σ_i is the step slope (which is very large in all cases), and δ_i is the step position. Figure 6 shows example data and its corresponding fit function.

Our component cost function, then is:

$$K(f, d) = a(d) + (\log(f)/ - c(d))^{-1.0/b(d)}$$

The cost of each stage of the cascade is equal to the cost of getting from the current false positive rate to the new false positive rate. This cost can be estimated as follows:

$$C_i(f, d) = K(\prod_{j=1}^{i} f_j, \prod_{j=1}^{i} d_j) - K(\prod_{j=1}^{i-1} f_j, \prod_{j=1-1}^{i} d_j) \qquad (8)$$

where f_i, d_i are the component false positive rates and detection rates respectively of each stage in the cascade.

6. Optimizing the Cost Function

We are now able to optimize Equation 3. Rewriting it to illustrate the final form of the function gives:

$$C_a(f_1, f_2, ..., f_N, d_1, d_2, ..., d_N) = C_1(f_1, d_1) + \sum_{i=2}^{N} \left[C_i(f_i, d_i) \prod_{j=1}^{i-1} p_j \right] \qquad (9)$$

where $C_i(f_j, d_j)$ is defined in Equation 8, and p_j is defined as in Equation 1. We solve Equation 9 as a constrained numerical nonlinear optimization problem. The constraints applicable to the problem are:

$$\begin{aligned} f_l \leq & f_i \leq 1 && \text{for } 1 \leq i \leq N \\ 0 \leq & d_i \leq d_u && \text{for } 1 \leq i \leq N \\ \prod_{i=1}^{N} & f_i \leq F \\ \prod_{i=1}^{N} & d_i \geq D \end{aligned}$$

where F is the target false positive rate, D is the target detection rate, f_l is some achievable minimum false positive rate ($f_l \approx 0$), and d_u is some achievable maximum detection rate ($d_u \approx 1$). In practice, the nonlinear inequality constraints can be replaced by equality constraints by noting that if the products are not equal to the constraint values, then the method is clearly doing more work than it needs to and a classifier which is not less efficient can always be found that satisfies the equality constraints. To find the optimal speed classifier, we need to optimize over each of the f_i, d_i and over N, the number of stages in the cascade.

The problem as specified can be solved using reduced gradient type methods (Gill et al., 1981) and we have used a variation of these types of methods. In typical reduced gradient methods, a step is made in the tangent space of the constraints, potentially producing an infeasible solution, and a further correction step is made to recover feasibility. In our solution we maintain feasibility at all times by taking advantage of the special nature of the constraints. The algorithm is based on the following insight. Consider the constrained minimization problem:

$$\text{minimize } f(x_1, x_2) \tag{10}$$

$$\text{subject to } x_1 x_2 = k \tag{11}$$

where k is some constant. Using the constraint, we can transform this problem to an unconstrained minimization:

$$\text{minimize} f(x_1, k/x_1). \tag{12}$$

For problems with more than two variables, we can fix all but two of the variables, perform a minimization, then fix all but another two and iterate until convergence. The final algorithm is given in Algorithm 1. A potentially more efficient version would actively choose which pairs of variables to optimize over rather than simply choosing the neighbours, but in all cases we found, the algorithm converged in a reasonable amount of time. We have also empirically investigated the convergence of the algorithm and optimality of the solution using different starting points for the search by randomly perturbing the starting false positive and detection rates (while maintaining feasibility). For each optimization we have tested 10 different starting points and in all cases the algorithm converged on the same solution. We have also experimented with Matlab's `fmincon` optimization function but found it to be less robust than our method, but when it did converge, it found the same solution.

7. Building the Classifier Cascade

Algorithm 1 returns the target false positive and detection rates that a cascade should achieve to minimize the execution cost of the classifier. We use a similar technique to that described by Viola

Algorithm 1 Minimizing the Cost Function

```
Require: $N > 0, 0 \leq F \leq 1, 0 \leq D \leq 1$
Ensure: $C_a$ is minimized
  $f_i \leftarrow F^{1/N} \forall i \in [1, ..., N]$
  $d_i \leftarrow D^{1/N} \forall i \in [1, ..., N]$
  while not converged do
      for $i = 1$ to $N - 1$ do
          minimize $C_a(f_i, k_f/f_i, d_i, k_d/d_i)$ where:
              $k_f = F f_i f_{i+1} / (\prod_{i=1}^{N} f_i)$, and
              $k_d = D d_i d_{i+1} / (\prod_{i=1}^{N} d_i)$, and
              $f_1, ..., f_{i-1}, f_{i+2}, ..., f_N, d_1, ..., d_{i-1}, d_{i+2}, ..., d_N$ are held constant.
      end for
  end while
  return $f_1, ..., f_N, d_1, ..., d_N$
```

and Jones (2001, 2004) to build our cascades except for two important differences. First, when the number of negative examples is limited as in most classification problems (unlike face detection for which an almost infinite number of negative examples can be generated), it is not helpful to eliminate negative examples correctly classified by previous stages as this can lead to overtraining problems for later stages (due to having very few negative training examples to work with).

We have also independently discovered a technique of reusing previous cascade stages similar to that reported by Sochman and Matas (2004). In our case, we use the evaluation from the previous level in the cascade as a starting point for the current level. If we consider the output hypothesis for level i of the cascade as:

$$h_i(x) = \begin{cases} 1 & \text{if } e^i(x) > B^i \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where:

$$e^1(x) = \sum_{t=1}^{T^1} (\log(\frac{1}{\beta_t^1})) h_t^1(x), \tag{14}$$

$$B^1 = \frac{1}{2} \sum_{t=1}^{T^1} \log(\frac{1}{\beta_t^1}), \tag{15}$$

$$e^i(x) = e^{i-1}(x) + \sum_{t=1}^{T} (\log(\frac{1}{\beta_t^i})) h_t^i(x), \tag{16}$$

$$B^i = B^{i-1} + \frac{1}{2} \sum_{t=1}^{T^i} \log(\frac{1}{\beta_t^i}), \tag{17}$$

and $\beta_t^i = \epsilon_t^i / (1 - \epsilon_t^i)$, ϵ_t^i is the error of weak classifier t in cascade level i, and $h_t^i(x)$ is the output of weak classifier t in cascade level i. Equations 14 and 15 are the equations for a monolithic AdaBoost classifier. Equations 16 and 17 use the evaluations from the previous layer and thus produce an evaluation identical to an equivalent monolithic AdaBoost classifier.

The final algorithm for building the cascade classifier is given in Algorithm 2. The function *AdaBoostAdd* simply adds a weak classifier (second parameter) to an existing AdaBoost classifier (first parameter) given the training data (third parameter). The call to *AdaBoostAdd* on line 4 allows the new layer to access the evaluations from the previous layer. The function *AdjustThreshold*, adjusts the threshold of the input AdaBoost classifier so the required detection rate is achieved for the given test data. Note that in this version of the algorithm we do not eliminate already rejected negative examples, nor generate new negative examples for each layer of the cascade. However, with problems with a very large number of negative examples (such as face detection) it is still desirable to do this.

Algorithm 2 Build a Cascade Classifier

```
Require: Target false positive rates: $f_1, ..., f_N$
Require: Target detection rates: $d_1, ..., d_N$
Require: Training examples: $T$
Require: Weak classifier learning algorithm: WeakLearn
 1: Cascade $\leftarrow \{\}$
 2: $(T_{train}, T_{test}) \leftarrow$ RandomPartition($T$)
 3: for $i = 1$ to $N$ do
 4:     $C_i \leftarrow$ AdaBoostAdd($NULL, C_{i-1}, T_{train}$)
 5:     AdjustThreshold($C_i, T_{test}, d_i$)
 6:     $(f'_i, d'_i) \leftarrow$ Test($C_i, T_{test}$)
 7:     while $f'_i < f_i$ do
 8:         $C_i \leftarrow$ AdaBoostAdd($C_i, WeakLearn, T_{train}$)
 9:         AdjustThreshold($C_i, T_{test}, d_i$)
10:         $(f'_i, d'_i) \leftarrow$ Test($C_i, T_{test}$)
11:     end while
12:     Cascade $\leftarrow$ {Cascade, $C_i$}
13: end for
14: return Cascade
```

8. Results

8.1 Data Sets

We have tested our model on eight data sets from the UCI machine learning repository (Newman et al., 1998). Table 1 documents the sets used and which class was used as the positive class in multi-class problems. We have also tested our techniques using a face detection problem. We use the CMU training database to provide the positive examples for training the cascade and the CMU/MIT upright face set for testing (the combined test sets of Sung and Poggio, 1998; Rowley et al., 1998). Both sets were provided by Henry Schneiderman (Schneiderman and Kanade, 2000). The negative examples are sampled from a further 2146 independent images which do not have faces in them.

Problem Set	Positive Class	Number of Positives	Number of Negatives
adult	>50K	7841	24720
covtype	7	2160	17840
isolet	B	240	5998
letter	A	789	19211
musk	musk	1017	5581
page blocks	2	329	5144
spambase	Not Spam	2788	1813
splice	Neither	1648	1527
statlog/sat	1	1072	3363
face detection	faces	2380	10000

Table 1: Data sets used from the UCI machine learning repository.

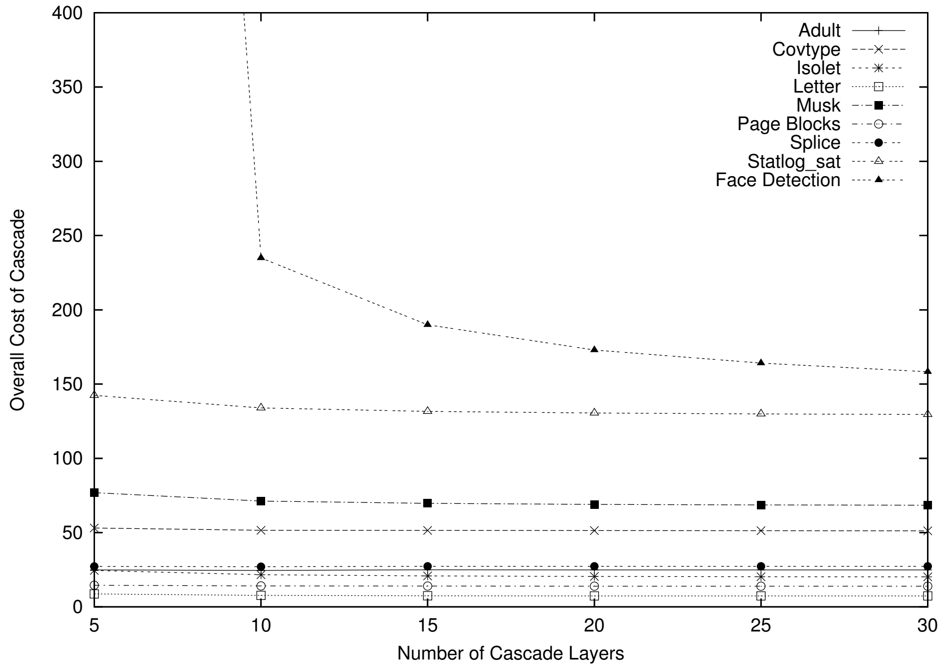
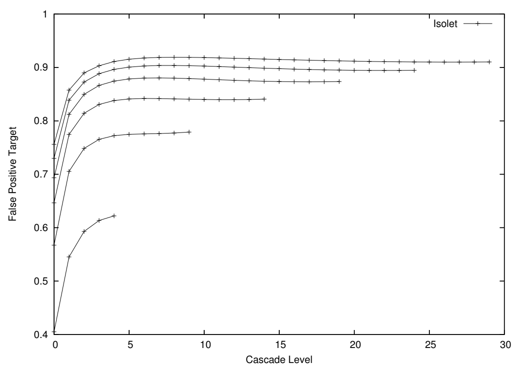


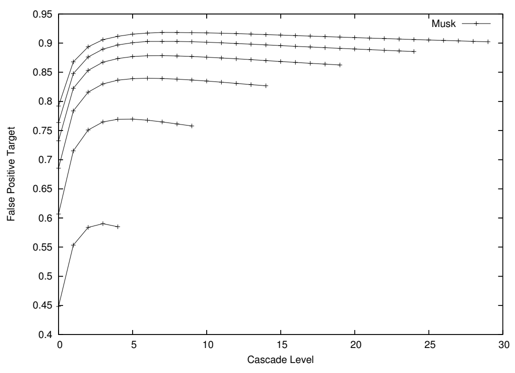
Figure 7: Estimated optimal costs for different cascade lengths for each of the problems studied.

8.2 Cascade Length

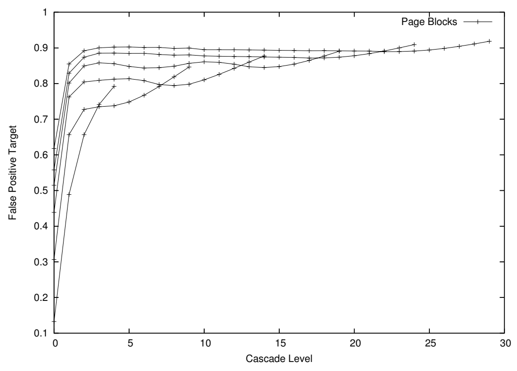
Figure 7 shows the estimated optimal cost of cascades of different lengths for each of the problems studied. In our model, longer cascades are always more efficient, although the efficiency gains quickly drop off as the length increases. For problems that are not rare events, such as most of the UCI problems, cascade lengths greater than 5 or 10 levels do not offer much advantage. For rare event problems, such as face detection, longer cascades are more useful - in our case, little can be gained from cascades larger than 20 layers.



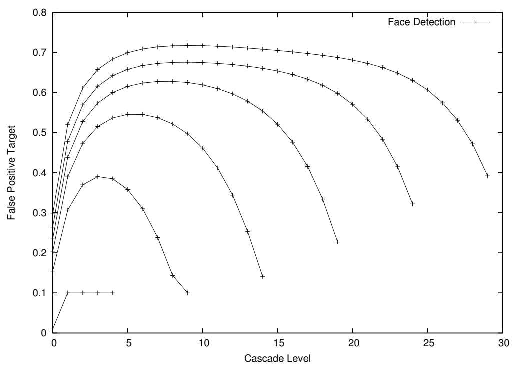
(a) Isolet



(b) Musk



(c) Page Blocks



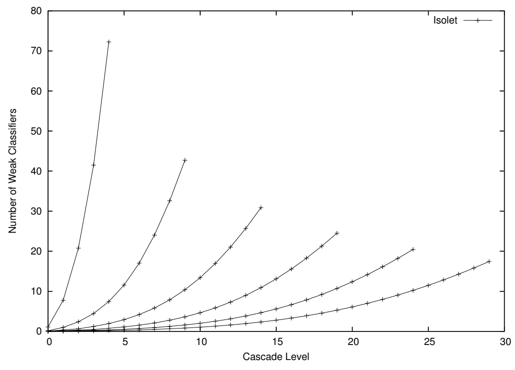
(d) Face Detection

Figure 8: Target false positive rates for some of the problems with different number of levels.

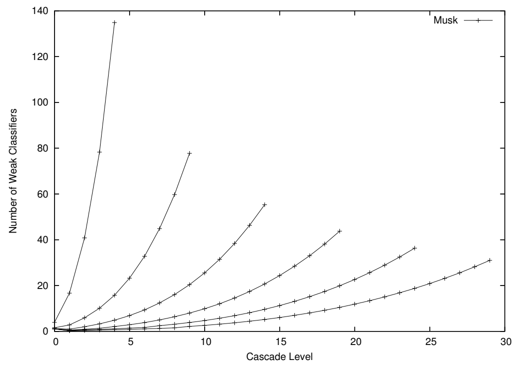
8.3 Optimal Parameters

It is worthwhile examining the optimal parameters which are produced as these typically follow a generic form and can give guidance to practitioners who want to use a cascade without the effort of producing an optimal one. The optimal detection rate settings are simple and obvious in retrospect. If the overall target detection rate is x say, then the first level of the cascade should have a target detection rate of x and all other levels should have a target detection rate of 1.0. Our method always converged on this solution regardless of the problem, the target detection rate or number of levels in the cascade. This is clearly optimal since the only situation in which it would not be would be the case where detecting more positives allows for the rejection of more negatives. But moving any decision boundary to include more positives will at best not include more negatives.

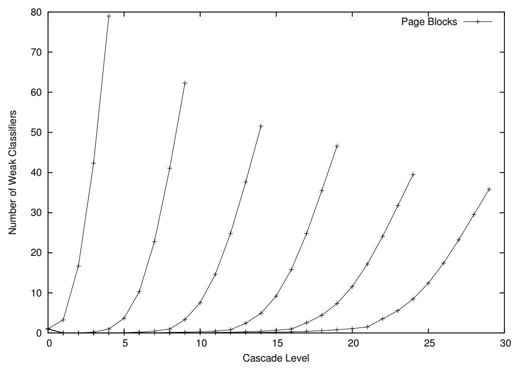
The target false positive ratio is a more complex situation. Figure 8 shows several examples with varying length cascades. More informative graphs are shown in Figure 9 which shows the number of weak classifiers required for each level of the cascade. The message for setting the cascade parameters is therefore quite simple and intuitive - do as little as possible early in the cascade, and progressively do more work in later levels of the cascade.



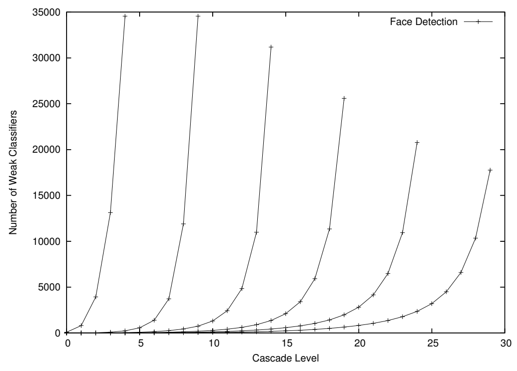
(a) Isolet



(b) Musk



(c) Page Blocks



(d) Face Detection

Figure 9: Target size for each cascade level for some of the problems with different number of levels.

Problem	Pos. Ratio	F.P. Target	D.R Target	Tree Height	Est. Speedup	Actual Speedup	Opt. Speedup	Est Cost/ Act. Cost
statlog_sat	0.24	0.03	0.99	3	3.20	2.12 ± 0.19	1.06	0.24 ± 0.03
splice	0.52	0.03	0.95	3	1.91	1.73 ± 0.55	1.06	0.89 ± 0.78
spambase	0.61	0.07	0.95	3	1.57	1.58 ± 0.06	1.05	1.14 ± 0.72
page_blocks	0.06	0.03	0.95	2	10.97	4.74 ± 2.83	1.60	1.75 ± 1.32
musk	0.15	0.02	0.95	3	4.07	4.21 ± 0.28	1.30	1.22 ± 0.46
letter	0.04	0.01	0.99	3	7.30	6.10 ± 1.20	1.33	0.73 ± 0.15
isolet	0.04	0.05	0.95	2	7.04	6.17 ± 1.76	1.46	1.12 ± 0.51
covtype	0.11	0.04	0.99	2	5.68	4.84 ± 0.67	1.14	0.66 ± 0.28
adult	0.24	0.34	0.95	1	1.87	1.87 ± 0.13	1.11	2.81 ± 3.17
faces	10^{-6}	0.01	0.8	1	20.4	9.25	2.1	1.0

Table 2: Comparison of estimated and actual speedups obtained by using a cascade classifier. Column 2 shows the ratio of positive examples in the data, column 3 shows the false positive rate target, column 4 shows the detection rate target, column 5 the height of the decision tree which is boosted (the most efficient height for each problem is shown), column 6 shows the speedup which is estimated by the model, column 7 shows the actual speedup obtained, column 8 shows the estimated speedup obtained by the optimized cascade over a non-optimized cascade, and column 9 shows the ratio of the estimated cascade cost to the actual cascade cost. Speedups are shown on test data and the numbers are averaged over 9 cascade training runs with a 50/50 training/test set split.

8.4 Cascade Performance

Table 2 shows speedups for several problems from the repository. In all of these problems, only 5 layer cascades have been used - larger cascades become more useful as the ratio of positive examples falls close to 0. There are several things to note about the data. The amount of speedup is loosely correlated with the ratio of positive examples as expected. Significant speedups can be obtained even when the positive ratio is quite high - from 30-50% for some evenly weighted problems, and more than double the speed for most problems. The optimization procedure can also produce a significant cost reduction - again this seems to be loosely correlated with the positive ratio. The table also shows that the model is less good at estimating actual cost often with large variances between runs. We believe this is due largely to the sensitivity of the combination of AdaBoost and decision trees to the training and test sets.

Figure 10 shows an ROC scatter-plot for each of the problems listed above. As can be seen from the plot, the cascade always returns an ROC point that is to the left and below the corresponding monolithic classifier (i.e. fewer false positives and fewer detections). This is entirely expected due to the early reject nature of the cascade.

For the face detection problem, we have also built a useful cascade classifier with 20 layers. The constraints on the optimization included an overall false positive rate of 10^{-6} and an overall detection rate of 0.8. On the CMU/MIT test set using 10 scale levels with window sizes between 25×25 pixels and 200×200 pixels and a simple mechanism for merging overlapping detections, we achieved a false positive rate of 5.9×10^{-6} and a detection rate of 0.77, which is reasonably

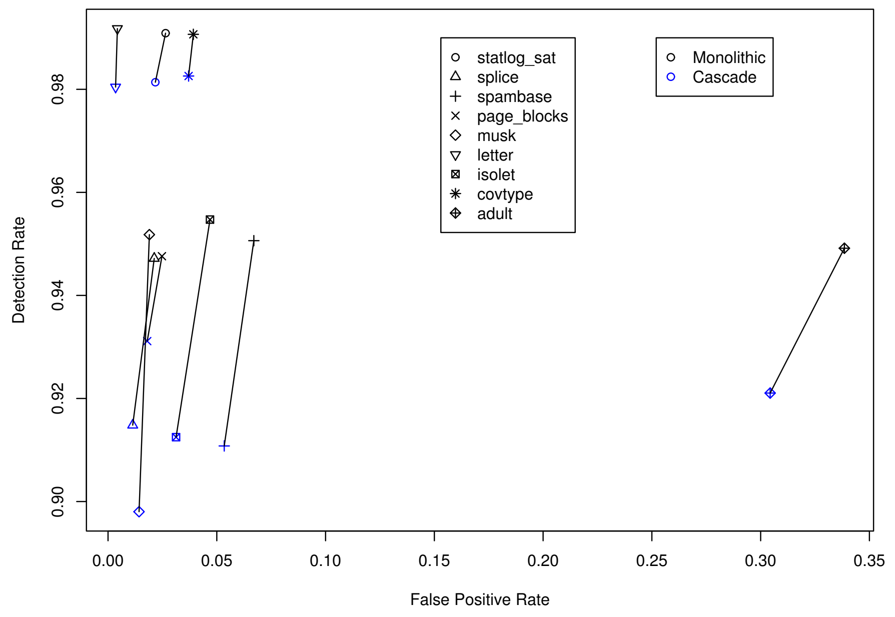


Figure 10: An ROC scatter-plot comparing the cascade classifiers with their corresponding monolithic classifiers. The same problems are symbol matched and connected by lines with different colours for monolithic and cascade classifiers.

close to the initial constraints. Our model also estimates on average 13.6 feature evaluations per window, where evaluating a classifier with 2 weak classifiers results in 2 feature evaluations. On the CMU/MIT test set, our classifier performed on average 13.0 feature evaluations per window. Note that this is slower than the results reported by Viola and Jones (2001, 2004) who report an average of 8 feature evaluations per window — since we did not have access to their training data or the particular set of features they used, we cannot do a direct comparison with their method. A naive cascade with a goal false positive rate of 0.5012 and goal detection rate of 0.9889 for each layer would require 93.8 feature evaluations per window according to our model.

9. Limitations

Cascade classifiers suffer from the same problems as the underlying monolithic classifier. In particular, it is possible to choose an overall target false positive rate and detection rate that cannot be achieved given the current training data and weak learning algorithm. There is no real solution to this problem — some care must be taken in choosing appropriate overall targets.

The model we use is quite complex and lacks some theoretical justification — why does Equation 5 seem to hold and why should the parameters be modeled as they are. In many respect our model is purely empirical and parsimonious. Because the model is empirical, there are inevitable inaccuracies. Nevertheless, we believe it has shed some light on the problem of cascade optimization. In particular, it has given insight into how to set the cascade parameters even if a practitioner is not prepared to go through the process of optimization. Although the modeling and optimization process is complex, we have fully automated it and have released Mathematica source code for performing the optimization.

10. Conclusion

Our empirical model provides an interesting glimpse into the workings of AdaBoost, but we do not have a strong theoretical justification for why such a model can be used so effectively. More work is required on the theoretical aspects hinted at by the model. We are strong advocates of the cascade architecture and think it would be interesting to apply the method to other learning algorithms such as support vector machines.

We have introduced a method for optimizing cascade classifiers by empirically modeling the execution cost of AdaBoost classifiers and have made several contributions to the state of knowledge on cascade classifiers. We have: shown that cascades can provide effective speedups for non-rare event problems; provided very strong evidence to support the intuitions of Viola and Jones (2001, 2004) to do as little work as possible early in the cascade; shown that even naively setting cascade parameters can provide significant speedups over monolithic classifiers; shown how cascade parameters can be optimized and that this optimization can provide significant efficiency gains over naively setting parameters in many cases.

References

Nitesh V. Chawla, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, April 2004.

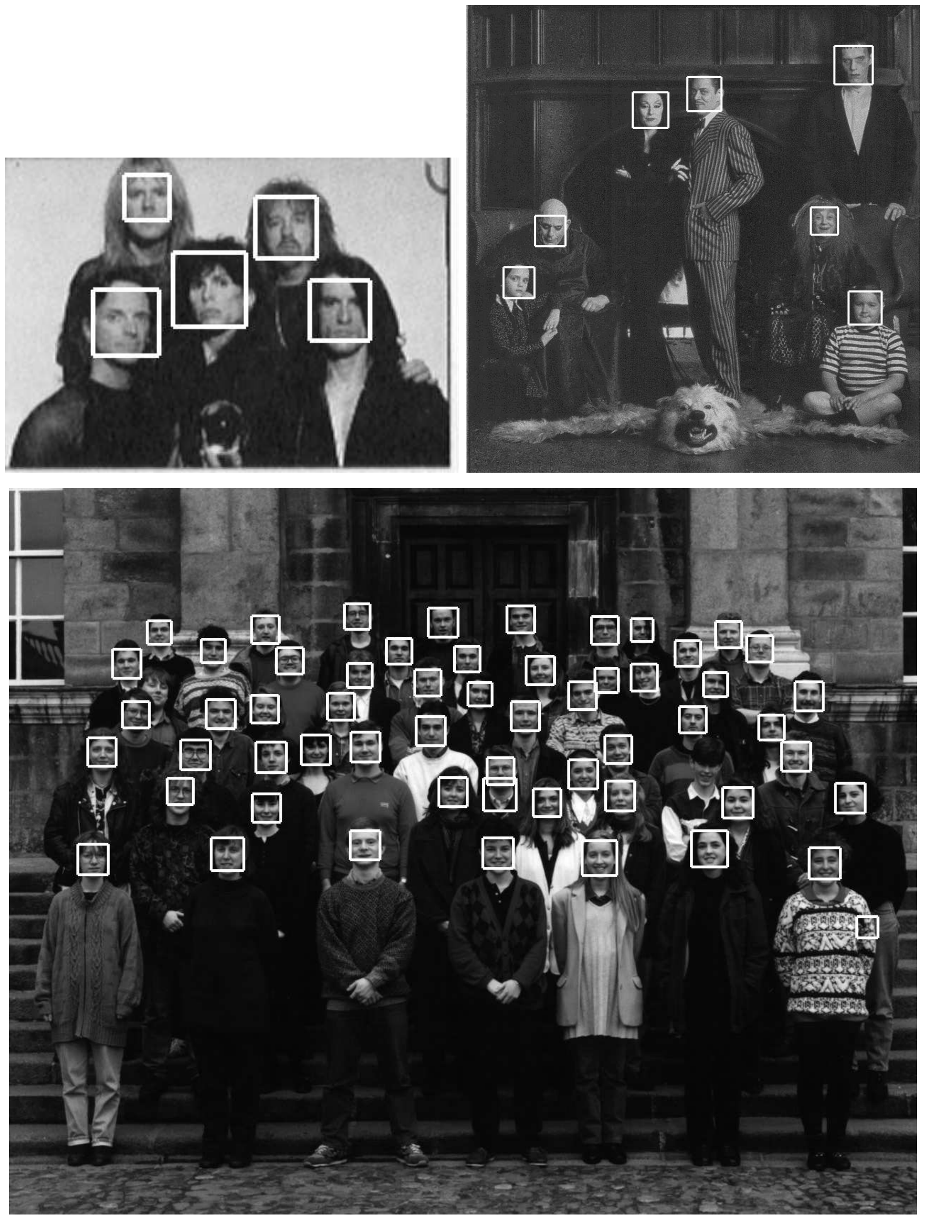


Figure 11: Some example detections

Xiangrong Chen and Alan L. Yuille. A time-efficient cascade for real-time object detection: With applications for the visually impaired. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 28, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2-3. doi: http://dx.doi.org/10.1109/CVPR.2005.399.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996. URL `citeseer.nj.nec.com/freund96experiments.html`.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*, chapter 6, pages 219–224. Academic Press, 1981.

E. Grossmann. Automatic design of cascaded classifiers. In *International IAPR workshop on statistical pattern recognition*, 2004.

Xiangsheng Huang, Stan Z.Li, and Yangsheng Wang. Learning with cascade for classification of non-convex manifolds. In *Proc. of CVPR Workshop on Face Processing in Video (FPIV'04), Washington DC, 2004*, 2004.

Rainer Lienhart, Alexander Kuranova, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM'03 25th Pattern Recognition Symposium*, volume 2781 of *Lecture Notes in Computer Science*, pages 297–304. Springer Verlag, 2003.

Brendan McCane and Kevin Novins. On training cascade face detectors. In *Image and Vision Computing New Zealand*, pages 239–244, 2003.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

David A. Ratkowsky. *Handbook of nonlinear regression models*, volume 107 of *Statistics: textbooks and monographs*, chapter 2, pages 31–33. Marcel Dekker, Inc., 1990.

H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:22–38, 1998.

H. Schneiderman and T. Kanade. A statistical method for 3D object detection applied to faces and cars. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1746–, 2000.

Henry Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

Jan Sochman and Jiri Matas. Inter-stage feature propagation in cascade building with adaboost. In *17th International Conference on Pattern Recognition (ICPR'04)*, volume 1, pages 236–239, 2004.

Jie Sun, James M. Rehg, and Aaron Bobick. Automatic cascade training with perturbation bias. In *IEEE International Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Press, 2004. To Appear.

Kah Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, 1998. URL `citeseer.ist.psu.edu/sung95example.html`.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, volume I, pages 511–518. IEEE Computer Society, 2001.

Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

Inc. Wolfram Research. Mathematica. Wolfram Research, Inc., Champaign, Illinois, 2003.

Jianxin Wu, James M. Rehg, and Matthew D. Mullin. Learning a rare event detection cascade by direct feature selection. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Lei Zhang, Stan Z. Li, and Zhi Yi Qu. Boosting local feature based classifiers for face recognition. In *Proc. of CVPR Workshop on Face Processing in Video (FPIV'04), Washington DC, 2004*, 2004.