FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

PARAMETER SENSITIVE FEATURE SELECTION FOR LEARNING ON LARGE DATASETS

By

GARY GRAMAJO

A Dissertation submitted to the
Department of Statistics
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2015

Gary Gramajo defended this dissertation on June 12 2015.
The members of the supervisory committee were:

Adrian Barbu
Professor Directing Dissertation

Kumar Piyush
University Representative

Fred Huffer
Committee Member

Yiyuan She
Committee Member

Jinfeng Zhang
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

To my wife, who endured many nights and weekends without me as I finished my PhD

ACKNOWLEDGMENTS

Many thanks are due to many people. My major professor who had to put up with many questions from me. To friends and family who pulled together to help me finish. And most importantly, to my Lord Jesus Christ, who heard all of my reasons why I could not finish my PhD but refused to give up on me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

The following short list of symbols are used throughout the document. The symbols represent quantities that I tried to use consistently.

π	$3.1415926\ldots$
$\boldsymbol{x} \in \mathbb{R}^M$	vector of length M
$D = \{(\boldsymbol{x}, y)\}$	training dataset of observation $\boldsymbol{x}$ and its response y
$\boldsymbol{\beta}$	learned coefficients (in classification or regression)

ABSTRACT

Though there are many feature selection methods for learning, they might not scale well to very large datasets, such as those generated in computer vision data. Furthermore, it can be beneficial to capture and model the variability inherent to data such as face detection where a plethora of face poses (i.e. parameters) are possible. We propose a parameter sensitive learning method that can learn effectively on datasets that can be prohibitively large. Our contributions are the following. First, we propose an efficient feature selection algorithm that optimizes a differentiable loss with sparsity constraints. We note that any differentiable loss can be used and will vary depending on the application. The iterative algorithm alternates parameter updates with tightening the sparsity constraints by gradually removing variables based on the coefficient magnitudes and a schedule. Second, we show how to train a single parameter sensitive classifier that models the wide range of class variability. The sole classifier is important since this reduces the amount of data necessary for training compared to methods where multiple classifiers are trained for each parameter value. Third, we show how to use nonlinear univariate response functions to obtain a nonlinear decision boundary with feature selection; an important characteristic since the separation of classes in real world datasets is very challenging. Fourth, we show it is possible to mine hard negatives with feature selection, though it is more difficult. This is vital in computer vision data where 10^5 training examples can be generated per image. Fifth, we propose an approach to perform face detection using a 3D model on a number of face keypoints. We modify binary face features from the literature (generated using random forests) to fit into our 3D model framework. Experiments on detecting the face keypoints and on face detection using the proposed 3D models and modified face features show that the feature selection dramatically improve performance and come close to the state of the art on two standard datasets for face detection . We also apply our parameter sensitive learning method with feature selection to detect malicious websites, a dataset with approximately 2.4 million websites and 3.3 million features per website. We outperform other batch algorithms and obtain results close to a high performing online algorithm but using far fewer features.

CHAPTER 1

FEATURE SELECTION WITH ANNEALING

In a classification problem set-up, we have n observations, each represented as $\boldsymbol{x}_i$, with p predictors. Each observation $\boldsymbol{x}_i \in \mathbb{R}^p$, is labeled as either $y_i \in \{-1, +1\}$, and is drawn i.i.d from a probability distribution $P(X, \boldsymbol{y})$. The goal of feature selection would be to select a subset of the original p predictors while improving the discriminative ability of a classifier. A brute force search of all possible feature combinations is computationally infeasible [117, 78]. Thus, many feature selection methods have been developed in order to efficiently select features while preserving or reducing the misclassification error rate. We start by introducing the myriad of work done in feature selection.

1.1 Related Works

In [111], Weston et al. state that the problem of feature selection generally fall into two types of categories,

1. For a fixed $k << p$, find the k features that give the smallest generalization error.
2. Determine the smallest k that gives the maximum allowable generalization error γ.

Weston et al. note that the choices of k in the first category can usually be re-parameterized as choices for γ in the second category.

The first representation of the problem is very similar to our main approach of Feature Selection with Annealing (FSA), explained the Section 1.2, and thus, we will continue to develop the problem of feature selection through the first perspective. Weston et al. represent the approximation of the generalization error as follows. For a fixed set of functions $\boldsymbol{y} = f(\boldsymbol{x}, \alpha)$, the goal is to find a preprocessing of the data $\boldsymbol{x} \mapsto \boldsymbol{x} * \sigma$, $\sigma \in \{0, 1\}^p$, and the parameters α of the function f that minimize the value of

$$\tau(\sigma, \alpha) = \int V(\boldsymbol{y}, f((X * \sigma), \alpha)) dP(X, \boldsymbol{y}) \tag{1.1}$$

subject to $\|\sigma\|_0 = k$, where $P(X, \boldsymbol{y})$ is unknown, $\boldsymbol{x} * \sigma = (x_1\sigma_1, \ldots, x_p\sigma_p)$ denotes element wise product, $V(\cdot, \cdot)$ is a loss functional and $\|\cdot\|_0$ is the 0-norm.

1.1.1 Feature Selection

FSA shares some similarity to the Recursive Feature Elimination [49] (RFE) procedure, a wrapper method, which alternates training an SVM classifier on the current feature set and removing a percentage of the features based on the magnitude of the variable coefficients. However, our approach has the following significant differences:

1. It removes numerous junk variables long before the parameters $\boldsymbol{\beta}$ have converged, thus it is much faster than the RFE approach where all coefficients are fully trained at each iteration.
2. It can be applied to any loss function, not necessarily the SVM loss and we present applications in classification and regression. We refer the reader to [4] for applications in ranking as well as further examples in classification and regression.
3. In [4], we present rigorous theoretical guarantees of variable selection and parameter consistency.

FSA can be viewed as a backward elimination method [48]. But its variable elimination is built into the optimization process. Although there are numerous ways for variable removal and model update, our algorithm design by combining the optimization update and progressive killing is unique to the best of our knowledge. These principles enjoy theoretical guarantees of convergence, variable selection and parameter consistency.

In [40] a screening procedure analyzed each variable by fitting a model that depends only on that variable. In contrast, our screening procedure fits a model that depends on all variables and gradually removes them according to a schedule.

There exist feature selection methods such as MRMR [86] and Parallel FS [124] that only select features, independent of the model that will be built on those features. In contrast, our method simultaneously selects features and builds the model on the selected features in a unified approach aimed at minimizing a loss function with sparsity constraints.

The following description on previous work assumes knowledge of Support Vector Machine (SVM). For more details, please reference the Appendix where a SVM is explained and derived.

In [111], Weston et al. present a computationally feasible feature selection algorithm for SVM that is based on minimizing the generalization error bounds using gradient descent. That is, they propose a method to minimize (1.1) over σ and α. In order to do this, they state two theorems that

establish bounds on the expected error probability. Suppose M is the size of the maximal margin and the images $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$ of the training vectors are within a sphere of radius R. The first theorem states the bound:

$$E[P_{err}] \leq \frac{1}{N} E\left[\frac{R^2}{M^2}\right] = \frac{1}{N} E[R^2 W^2(\alpha^0)] \tag{1.2}$$

where W^2 is the dual problem that is maximized as is explained in the appendix.

This bound demonstrates that the performance of the SVM depends on the ratio $E[R^2/M^2]$ and not simply the large margin M where R is controlled by the mapping function $\phi(\cdot)$. Using the span of support vectors, Vapnik and Chapelle derived an estimate for the expected leave-one-out error probability [25]:

$$E[P_{err}^{N-1}] \leq \frac{1}{N} E\left[\sum_{i=1}^{N} \Psi\left(\frac{\alpha_i^0}{(K_{SV}^{-1})_{ii}}\right) - 1\right] \tag{1.3}$$

where

- Ψ is the step function
- K_{SV} is the matrix of dot products between support vectors
- P_{err}^{N-1} is the probability of test error for the machine trained on a sample of size $N-1$
- The expectations are taken over the random choice of the sample

Given that the support vector method attempts to find the function from the set $f(\boldsymbol{x}, \boldsymbol{\beta}, \beta_0) = \boldsymbol{\beta} \cdot \Phi(\boldsymbol{x}) + \beta_0$ (over $\{\boldsymbol{\beta}, \beta_0\}$) that minimizes the generalization error, [111] starts by enlarging the set of functions considered by the algorithm to $f(\boldsymbol{x}, \boldsymbol{\beta}, \beta_0) = \boldsymbol{\beta} \cdot \Phi(\boldsymbol{x} * \sigma) + \beta_0$. The mapping $\Phi_\sigma(\boldsymbol{x}) = \Phi(\boldsymbol{x} * \sigma)$ can be represented via the kernel K_σ (for any K):

$$K_\sigma(\boldsymbol{x}, \boldsymbol{z}) = K((\boldsymbol{x} * \sigma), (\boldsymbol{z} * \sigma)) = (\Phi_\sigma(\boldsymbol{x}) \cdot \Phi_\sigma(\boldsymbol{z})) \tag{1.4}$$

Since the bounds in (1.2) and (1.3) hold for these kernels, we can use (1.2) to minimize over σ:

$$R^2 W^2(\sigma) = R^2(\sigma) W^2(\alpha^0, \sigma) \tag{1.5}$$

Since finding the minimum of R^2W^2 over σ requires enumerating all possible n combinations, the approach of this thesis is to approximate σ via a real-valued vector. Then, the optimum value of σ can be found via gradient descent using the gradients explained in [25] and an extra constraint which approximates integer programming.

As Bradley and Mangasarian point out in [11], the SVM performs feature selection when an appropriate norm is used. They compare the performance of the SVM using 3 norms ($\|\cdot\|_1$, $\|\cdot\|_2^2$, $\|\cdot\|_\infty$) against their Feature Selection Concave (FSV). Similar to the objective function of the SVM that minimizes the distance of misclassified points, they introduce a term to the objective function that performs feature selection. They avoid introducing discontinuity into the objective via by a concave exponential approximation of this new term which essentially keeps track of the nonzero features. This approximation provides a smooth objective to optimize and they prove [12] that this smooth problem generates an exact solution to the non-smooth problem. In practice, they implement a fast successive linear approximation which terminates in a finite number of steps (usually less then 7) at a stationary point which they prove satisfies the minimum principle necessary optimality condition for the smooth problem.

Bradley and Mangasarian make their comparison over 6 public datasets. The SVMs with the $\|\cdot\|_2^2$ and $\|\cdot\|_\infty$ norms had the worst generalization performance and performed little to no feature selection. The poor generalization ability of these two norms is to be expected given their sensitivity FSV and the SVM with the $\|\cdot\|_1$ norm had comparable performance both in test error and feature selection performance. These results are expected given the popularity of the Hinge loss in SVMs.

In [76], Miranda et al. extend FSV to SVM with the added capacity for generalization by minimizing the norm of the predictors. Similarly to FSV, they approximate the sum of non-zero predictors by a concave exponential approximation. They refer to this model as a linearly penalized SVM (LP-SVM). In a similar fashion to FSV, they use Cross Validation to obtain the best model parameters.

In [53], predictors are ranked, after a single training run, according to their relative importance for classifying observations. Essentially, Hermes and Buhmann estimate the importance of individual feature components from the discriminant function $f(\boldsymbol{x})$ (A.18) provided by the SVM. The first training iteration provides optimal values for the Lagrange multipliers λ_i which constitute the support vectors and the decision boundary. To estimate the influence of the components of an observation $\boldsymbol{x}$ have on the classification decision, they compute the gradient of $\nabla f(\boldsymbol{x})$. They point out that since $f(\boldsymbol{x})$ is a linear combination of kernel products, $\nabla f(\boldsymbol{x})$ can be written as the weighted sum of kernel derivatives:

$$\nabla f(\boldsymbol{x}) = \sum_{i \in SV} \lambda_i y_i \nabla_{\boldsymbol{x}} K(\boldsymbol{x}_i, \boldsymbol{x}) \tag{1.6}$$

The unit vectors $\mathbf{e}_j$, where the $j = 1, \ldots, n$, represent the indices of the individual features, are compared to $\nabla f(\boldsymbol{x})$ (after normalization). They claim that if a feature component x_j is not important at position $\boldsymbol{x}$, then $\nabla f(\boldsymbol{x})$ should be roughly orthogonal to $\mathbf{e}_j$ (see their paper for the equation they use to calculate the angle). That is, x_j should not influence the distance to the decision hyper-plane. Furthermore, they perform a type of data reduction by only computing the angles for observations that are within a small distance of the decision boundary margins. These angles are averaged per feature component and only the features with the highest averages are kept.

In [49], Guyon et al. introduce SVM Recursive Feature Elimination (SVM-RFE). It is a wrapper-based approach that utilizes the objective function $\left(\frac{1}{2}\|\boldsymbol{\beta}\|^2\right)$ as a feature-ranking criterion to discover features with strong discriminatory abilities. Extending LeCun's idea in the Optimal Brain Damage algorithm of approximating the derivative of the cost function J using a Taylor series to second order, Guyon points out that at the optimum of J, the first order term can be neglected. This yields

$$DJ(i) = \frac{1}{2}\frac{\partial^2 J}{\partial w_i^2}(Dw_i)^2 \tag{1.7}$$

Guyon points out that the change in weight $Dw_i = w_i$ corresponds to removing feature i. Using the criteria of (1.7) to estimate the effect of removing one feature at a time on the objective function, Guyon states that the essence of the SVM-RFE is

i. The classifier is trained (i.e. the optimal w_i are obtained)

ii. Compute the ranking criterion

iii. Remove the feature(s) with the smallest rank

In [88], Rakotomamonjy extends the work in [49] and investigates three ranking criteria C_t which are based on either the weight vector $\|\boldsymbol{\beta}\|^2$ [49], the radius or margin bound $R^2\|\boldsymbol{\beta}\|^2$ [107] or the "span estimate" which involves the distance between a mapped support vector $\Phi(\boldsymbol{x})$ and the span of the other support vectors [106]. Rakotomamonjy uses two approaches for each criterion (inspired from the neural networks community). The "zero-order" is when the criterion is directly used for variable ranking, $R(i) = C_t^{(i)}$ where $C_t^{(i)}$ is the criterion value when the variable i is removed. The "first-order" method ranks a variable according to its influence on the absolute value of the derivative of the criterion. In the same nature as SVM-RFE, the optimum k variables are

found using backward selection. Overall, the first order method using the gradient of the weight vector performed the best both in accuracy and in computational complexity.

A convex framework for jointly training a SVM and choosing optimal features is proposed in [80]. They use a convex function to determine the weights of features which results in a convex optimization problem that can be solved using standard convex optimization software. They show both theoretically and through experiments on public datasets that their approach generates sparse features and require less support vectors compared to competing methods.

Other feature selection methods for SVM impose sparsity constraints on the SVM weights, being limited to linear SVM [79, 125] or polynomial kernels [110]. The L_1-norm SVM [125] optimizes the hinge loss with a L_1 penalty. This work shows that the regularization path is piecewise linear and offers an algorithm to compute it. The combined SVM method [79] optimizes the hinge loss with a combination of L_0 and L_2 penalties using the DC (difference of convex functions) optimization. Our work optimizes a differentiable approximation of the hinge loss with L_2 regularization and L_0 constraints but uses the FSA optimization method, which is very fast and scales well to large datasets.

1.1.2 Boosting

Boosting algorithms – such as Adaboost [97], Logitboost [43], Floatboost [69], Robust Logitboost [68] to cite only a few – optimize a loss function in a greedy manner in k iterations, at each iteration adding a weak learner that decreases the loss most. There are other modern versions such as LP-Adaboost [46], arc-gv [14], Adaboost* [90], LP-Boost [34], Optimal Adaboost [94], Coordinate Ascent Boosting and Approximate Coordinate Ascent Boosting [95], which aim at optimizing a notion of the margin at each iteration. Boosting has been regarded as a coordinate descent algorithm [93, 95] that optimizes a loss function, which can be margin based.

Boosting algorithms do not explicitly enforce sparsity but can be used for feature selection by using weak learners that depend on a single variable (feature). What feature will be selected in the next boosting iteration depends on what features have already been selected and their current coefficients. This dependence structure makes it difficult to obtain a general theoretical variable selection guarantee for boosting.

The approach introduced in this thesis is different from boosting because it starts with all the variables and gradually removes variables, according to an elimination schedule. Indeed, its top-

down design is opposite to that of boosting, but seems to be less greedy in feature selection based on our experiments in Section 1.8.

1.1.3 Penalized Loss Algorithms

Penalized loss algorithms add a sparsity inducing penalty such as the L_1[17, 36, 59, 121], SCAD [41] or MCP [120] and optimize the penalized loss in different ways. The proposed method is different from the penalized methods because variable selection is not obtained by imposing a sparsity prior on the variables, but by a direct optimization of the L_0 constrained loss function using a suboptimal algorithm.

By directly minimizing the constrained loss function, the approach proposed in this thesis does not introduce any undesired bias on the coefficients. Any desired biases (e.g. shrinkage) can be introduced as priors in the loss function $L(\boldsymbol{\beta})$. In contrast, the L_1 penalty introduces a bias in the coefficients that could lead to poor classification performance [16, 38, 41].

The closest related work is the Thresholding based Iterative Selection Procedure (TISP), HardTISP and QuantileTISP in particular [99, 98] which optimizes a penalized loss function by alternating coefficient updates and thresholding. However, it does not remove the variables according to a schedule.

The proposed method is inspired by TISP and by the Graduated Non-convexity Algorithm [9] that optimizes a non-convex Markov Random Field by first optimizing a convex surrogate and gradually changing the objective function towards the desired non-convex formulation, initializing each optimization step with the result from the previous step.

1.2 Feature Selection with Annealing

Our contribution to feature selection is via a method called *Feature Selection with Annealing* (FSA). This method is one that starts the selection process with all the variables or features. At each iteration, contrary to Boosting algorithms, a certain number of variables are removed based on coefficient magnitudes. Each iteration sees the optimization algorithm perform a one gradient update of the model parameters and a selection step that removes variables according to a predefined schedule (as in Annealing).

Let $D = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, N\}$, be training examples with $\boldsymbol{x}_i \in \mathbb{R}^P$. We will use a loss function $L_D(\boldsymbol{\beta})$ that is differentiable w.r.t. $\boldsymbol{\beta}$ over these training examples. We are interested in the constrained optimization in Equation (1.8)

$$\boldsymbol{\beta} = \underset{|\{i,\beta_i \neq 0\}| \leq k}{\operatorname{argmin}} L_D(\boldsymbol{\beta}) \tag{1.8}$$

where k is the number of specified relevant features. We note that this constrained form provides a more straight forward tuning process compared to penalty parameters such as λ in $\lambda\|\boldsymbol{\beta}\|_1$. We demonstrate empirically in Section 1.8 the there is a large range of values over which k can be chosen and good results can be obtained.

1.2.1 Algorithm Description

The main ideas in FSA is to use an annealing plan that will lessen the greediness in reducing the dimension from M to k and to facilitate computation by removing the most irrelevant variables. The description of the algorithm is as follows: The parameters $\boldsymbol{\beta}$ are initialized (typically $\boldsymbol{\beta} = 0$). Then, over N^{iter} iterations, our method alternates between an update step and a selection step. The parameter update step is done with the goal of minimizing the loss $L(\boldsymbol{\beta})$ by gradient descent, as in Equation (1.9).

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \frac{\partial L_D(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \tag{1.9}$$

The selection step only keeps the M_e of the variables with the largest coefficient magnitudes $|\beta_j|, j = \overline{1, M}$. The annealing schedule M_e allows us to have the sparsity constraint $|\{i, \beta_i \neq 0\}| \leq M_e$ after iteration e. After a large number iterations, the constraint is tightened until we obtain $|\{i, \beta_i \neq 0\}| \leq k$. In this manner, we achieve a suboptimal solution to the constrained problem (1.8).

The prototype algorithm is summarized in Algorithm 1 which is very easy to implement. The keep-or-kill rule is based on the magnitude of coefficients and does not involve any information about the objective function L. This is in contrast to many ad-hoc backward elimination approaches. Step 4 conducts an adaptive screening, which results in a nonlinear operator that increases the difficulty of the theoretical analysis. However in [4], we demonstrate that our approach has a rigorous guarantee of computational convergence and statistical consistency. This theoretical work is due to Dr. Yiyuan She.

Figure 1.1 provides an example of keep-or-kill process for a classification problem with $N = 1000$ observations and $M = 1000$ variables described in Section 1.8. Notice how some of the β_j are zeroed after each iteration. The algorithm stabilizes quickly, taking 80 iterations in the example presented in Figure 1.1. The problem size and thus the complexity keeps dropping due to the removal process. In an effort to save computational cost, "apparent junk" dimensions are eliminated early on while the features whose relevancy is harder to determine are handled later when the problem complexity has been lessened.

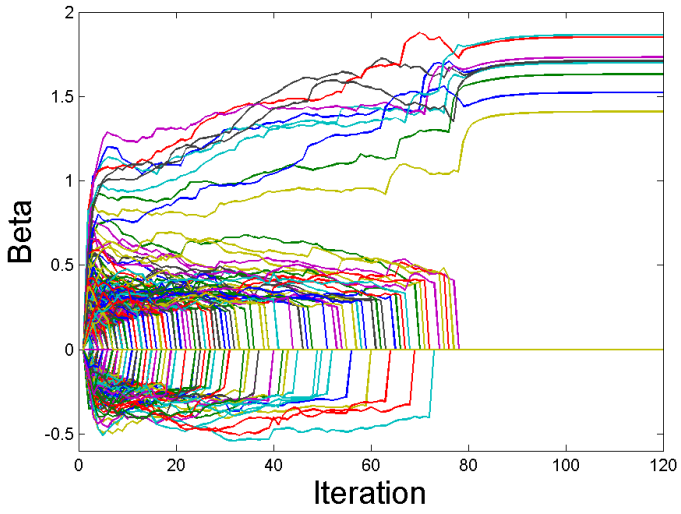

Figure 1.1: The value of $\beta_j, j = \overline{1, M}$ vs iteration number for simulated data with $N = 1000, M = 1000, k = 10$ with $\eta = 20, \mu = 300$.

The FSA algorithm is summarized in Algorithm 1.

Algorithm 1 Feature Selection with Annealing (FSA)

Input: Training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with $\boldsymbol{x}_i \in \mathbb{R}^M$.

Output: Trained classifier parameters β.

1: Initialize $\beta = 0$.
2: **for** e=1 to N^{iter} **do**
3: Update $\beta \leftarrow \beta - \eta \frac{\partial L_D(\beta)}{\partial \beta}$
4: Keep the M_e variables with highest $|\beta_j|$ and renumber them $1, ..., M_e$.
5: **end for**

1.2.2 Implementation Details

The inverse schedule M_e is used so that the algorithm hones in on a small number of important variables. Any annealing schedule $\{M_e\}$ slow enough works well in terms of estimation and selection accuracy. But a fast decaying schedule could reduce the computational cost significantly. Our experience shows that the inverse schedule in Equation (1.10) with a parameter μ provides a good balance between efficiency and accuracy:

$$M_e = k + (M - k) \max\left(0, \frac{N^{iter} - 2e}{2e\mu + N^{iter}}\right), e = \overline{1, N^{iter}} \tag{1.10}$$

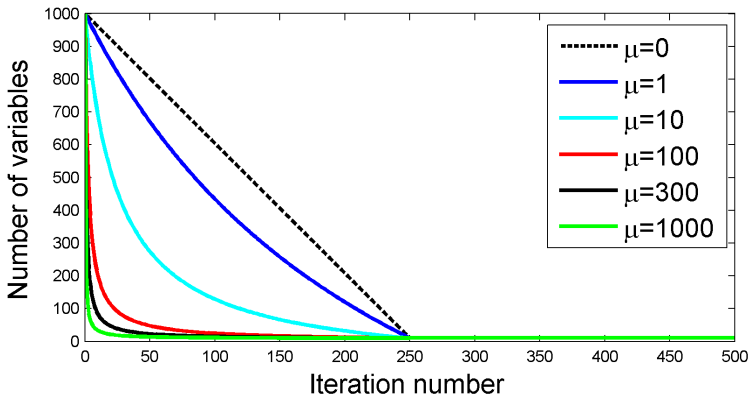

Figure 1.2: The number of kept features M_e vs iteration e for different schedules, with $M = 1,000, k = 10, N^{iter} = 500$.

Figure 1.2 plots the schedules for six difference choices of μ with $M = 1,000, k = 10$ and $N^{iter} = 500$. Different schedules can be chosen by choosing different values for μ, which is proportional to the number of variables that will be kept at each iteration. The computation time is proportional to the area under the graph of the schedule curve. Examples of computation times are given in Table 1.1. The overall computational complexity of FSA is linear in the problem size, MN.

The performance of the FSA algorithm depends on the following parameters:

- Gradient learning rate η, which can be arbitrarily small provided that the number of iterations is large enough. If η is too large, the coefficients β_i will not converge. In all our experiments we used $\eta = 0.5$.
- Annealing parameter μ for controlling the number of variables left at each iteration. Parameter μ should be proportional to the parameter η so that if the learning rate is small, the variables are removed at a slower pace.

Table 1.1: Computation times for selecting k variables using N observations of dimension M, when $N^{iter} = 500$.

Annealing param μ	Computation Time
$\mu = 0$	$125MN + kN^{iter}$
$\mu = 1$	$97MN + kN^{iter}$
$\mu = 10$	$41MN + kN^{iter}$
$\mu = 100$	$10MN + kN^{iter}$
$\mu = 300$	$5MN + kN^{iter}$
$\mu = 1000$	$2MN + kN^{iter}$

- Number of iterations N^{iter}, large enough to obtain in the end a desired number k of variables and to ensure the parameters have converged to a desired tolerance. In our experiments we used $N^{iter} = 500$.

We also point out the strength of FSA when it comes to tuning the parameters $\eta, \mu, N^{\text{iter}}$. As shown in Section 1.8, there is a large range of values over which the algorithm is stable.

In practice, the gradient update (1.9) has been replaced by one epoch of stochastic gradient updates. The optimization approach differs from backward selection in the following ways:

1. Variables are removed based on the magnitude of β_j not on p-values
2. Many variables are usually removed at each iteration.
3. Variables are removed long before the parameters $\boldsymbol{\beta}$ have converged.

The FSA algorithm can be parallelized for large scale problems by subdividing the $N \times M$ data matrix into a grid of sub-blocks that fit into the memory of the processing units. Then the per-observation response vectors can be obtained from a row-wise reduction of the partial sums computed by the units. The parameter updates are done similarly, via column-wise reduction. A GPU based implementation could offer further computation cost reductions.

As will be explained in the next two section, FSA can be used in regression or classification. FSA has also been used for ranking [4] but this is not the focus of this work.

1.3 FSA for Regression

The set-up to perform regression using FSA is as follows. Given training examples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^M \times \mathbb{R}, i = \overline{1, N}$, we have the penalized squared-error loss

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \sum_{j=1}^{M}\rho(\beta_j) \tag{1.11}$$

with a differentiable prior function ρ such as $\rho(\beta) = s\|\beta\|^2$. We refer the reader to [4] in order to see experiments on regression with FSA.

1.4 FSA for Classification

FSA can be used for classification and feature selection. The following three loss functions are examples of differentiable losses that can be used and are implemented extensively in our experiments. As stated before, we have training examples $D = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, N\}$ and $y_i \in \{0, 1\}$.

1.4.1 Penalized Logistic Loss

The logistic penalized loss is set-up as

$$L(\boldsymbol{\beta}) = -\sum_{n=1}^{N} w_n \ln\left[1 + \exp\left(-\tilde{y}_n(\beta_0 + \sum_{i=1}^{M}\boldsymbol{\beta}_i x_{ni})\right)\right] + \sum_{i=1}^{M}\rho(\boldsymbol{\beta}_i) \tag{1.12}$$

where $\tilde{y}_i = 2y_i - 1 \in \{-1, 1\}$. This loss has partial derivatives

$$\begin{aligned}\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}_j} &= \sum_{n=1}^{N} w_n \frac{\tilde{y}_n x_{nj}\exp\left(-\tilde{y}_n(\beta_0 + \sum_{i=1}^{M}\boldsymbol{\beta}_i x_{ni})\right)}{1 + \exp\left(-\tilde{y}_n(\beta_0 + \sum_{i=1}^{M}\boldsymbol{\beta}_i x_{ni})\right)} + \rho'(\boldsymbol{\beta}_j) \\ &= \sum_{n=1}^{N} w_n x_{nj}(y_n - p(\boldsymbol{x}_n)) + \rho'(\boldsymbol{\beta}_j)\end{aligned} \tag{1.13}$$

1.4.2 Penalized Huberized Loss

We use is a differentiable approximation of the primal SVM objective function from [24]

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} L_h(y_i\boldsymbol{x}_i^T\boldsymbol{\beta}) + \sum_{j=1}^{M}\rho(\boldsymbol{\beta}_j) \tag{1.14}$$

where $L_h : \mathbb{R} \to \mathbb{R}$ is the Huber-style differentiable approximation of the hinge loss [24]:

$$L_h(x) = \begin{cases} 0 & \text{if } x > 1 + h \\ \dfrac{(1 + h - x)^2}{4h} & \text{if } |1 - x| \leq h \\ 1 - x & \text{if } x < 1 - h \end{cases} \tag{1.15}$$

1.4.3 Penalized Lorenz Loss

We also experiment with the following loss function

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} L_h(y_i \boldsymbol{x}_i^T \boldsymbol{\beta}) + \sum_{j=1}^{M} \rho(\boldsymbol{\beta}_j) \tag{1.16}$$

where $L : \mathbb{R} \to \mathbb{R}$ is the following differentiable function:

$$L_h(x) = \begin{cases} 0 & \text{if } x > 1 \\ \ln(1 + (x-1)^2) & \text{else} \end{cases} \tag{1.17}$$

The Lorenz loss is differentiable everywhere, it is zero for $x \in [1, \infty)$ and grows logarithmically with respect to $|x|$ as $x \to -\infty$. These properties make the Lorenz loss (1.16) behave like the SVM loss in the sense that correctly classified examples that are far from the margin don't contribute to the loss. Moreover, the Lorenz loss is more robust to label noise than the SVM and logistic losses because the loss values for the misclassified examples that are far from the margin is not much higher than for those that are close to the margin. This loss is not convex, but it works well in practice together with the FSA algorithm, as it will be seen in experiments.

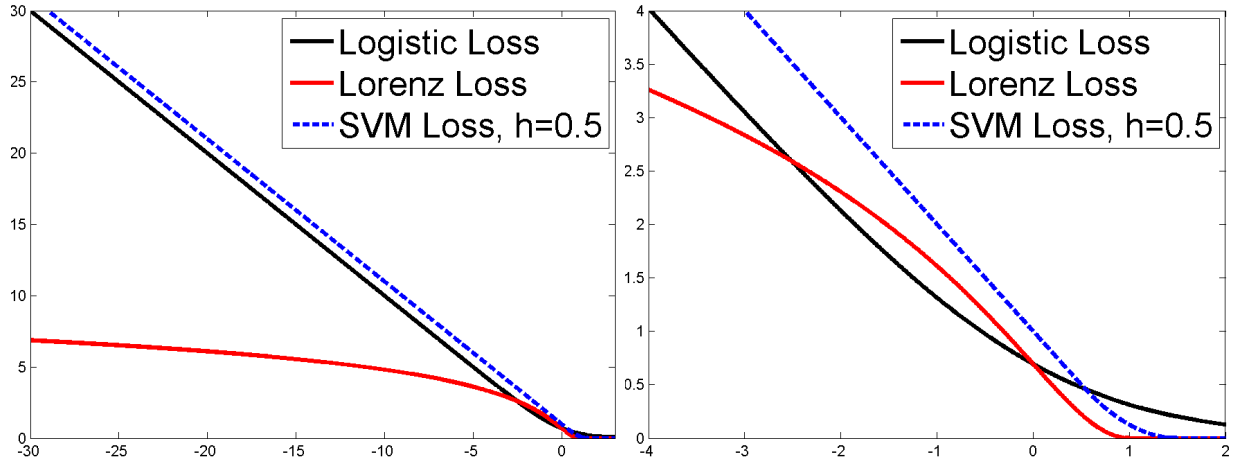

Figure 1.3: The loss functions from eq. (1.12), (1.15) and (3.4.2). Left: the losses on the interval $[-30, 3]$. Right: zoom in the interval $[-4, 2]$.

Figure 1.3 demonstrates the behavior of the loss functions the further an observation is classified on the wrong side of the margin. As we move from right to left on the horizontal axis, which corresponds to an observation being classified further in on the wrong side of the margin, a larger penalty is assigned to this misclassification (as demonstrated by all three losses). In the case of the SVM loss, misclassifications pay the highest penalty while the Lorenz loss pay the smallest loss.

1.5 Nonlinearity and Feature Selection

In most computer vision problems the data variability is very large and the training examples cannot be separated by a linear classifier based on a small number of features. We use a type of nonlinearity that is compatible with feature selection. For that we replace $\beta \boldsymbol{x}$ with a nonlinear response function that is a sum of a number of univariate functions

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{j=1}^{M} f_{\boldsymbol{\beta}_j}(x_j), \tag{1.18}$$

where $\boldsymbol{\beta}_j$ is a parameter vector characterizing the response function on variable j.

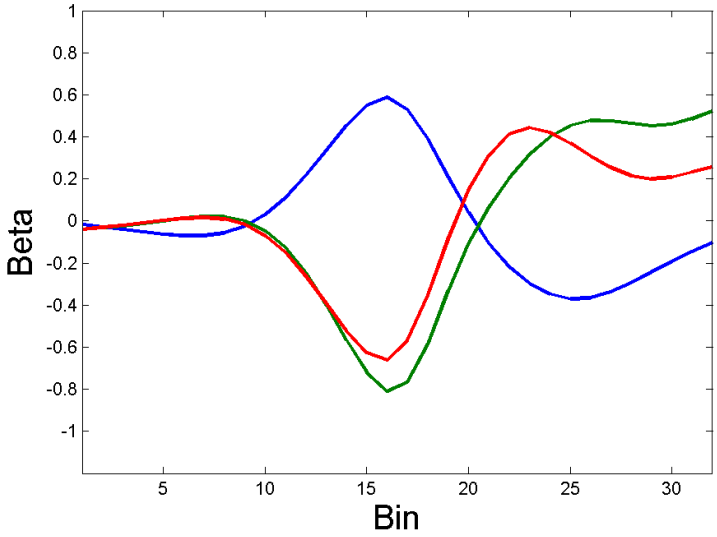

Figure 1.4: Piecewise linear response functions obtained for an eye detector.

We will use piecewise linear univariate response functions $f_{\boldsymbol{\beta}_k}(x_k)$ that can be written as $f_{\boldsymbol{\beta}_k}(x_k) = \boldsymbol{u}_k^T(x_k)\boldsymbol{\beta}_k$ where $\boldsymbol{u}_k(x_k)$ is the basis response vector and $\boldsymbol{\beta}_k \in \mathbb{R}^{B+1}$ is the coefficient vector of variable k. We obtain

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{k=1}^{M} \boldsymbol{u}_k^T(x_k)\boldsymbol{\beta}_k, \tag{1.19}$$

We offer the following explanation to provide further insight into how we implement these nonlinear functions. To simplify notation we will temporarily drop the index k and we will implicitly assume that we are working with variable x_k of the feature vector $\boldsymbol{x} = (x_1, ..., x_M)$.

A piecewise linear (PL) function $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ is defined based on the range $[x^{min}, x^{max}]$ of the variable x and a predefined number B of bins. Let $b = (x^{max} - x^{min})/B$ be the bin length.

For each value x, the function finds the bin index $j(x) = \left[(x - x^{min})/b\right] \in \{0, ..., B-1\}$ and the relative position in the bin $\alpha(x) = (x - x^{min})/b - j(x) \in [0, 1)$ and returns

$$f(x, \boldsymbol{\beta}) = \beta_{j(x)}(1 - \alpha(x)) + \beta_{j(x)+1}\alpha(x).$$

For $b \in \{0, ..., B\}$ let

$$u_b(x) = \begin{cases} 1 - \alpha(x) & \text{if } b = j(x) \\ \alpha(x) & \text{if } b = j(x) + 1 \\ 0 & \text{else.} \end{cases}$$

Then $u_b(x)$ are $B + 1$ piecewise linear basis functions and $f(x)$ can be written as a linear combination: $f(x, \boldsymbol{\beta}) = \sum_{b=0}^{B} \beta_b u_b(x) = \boldsymbol{\beta}^T \boldsymbol{u}(x$ where $\boldsymbol{u}(x) = (u_0(x), ..., u_B(x))^T$ is the vector of responses of the basis functions and $\boldsymbol{\beta} = (\beta_0, ..., \beta_B)^T$ is the parameter vector.

Some recent works [75, 55] use nonlinear additive models that depend on the variables through one dimensional smooth functions. In [75] it was proved that cubic B-splines optimize a smoothness criterion on these 1D functions. Variable selection was obtained by a group lasso penalty. A similar model is presented in [91] where a coordinate descent soft thresholding algorithm is used for optimizing an L_1 group-penalized loss function. Our work differs from these works by imposing constraints on the coefficients instead of biasing them with the L_1 penalty. Moreover, our optimization is achieved by a novel gradual variable selection algorithm that works well in practice and is computationally efficient.

In [40] a screening procedure analyzed each variable by fitting a model that depends only on that variable. In contrast, our screening procedure fits a model that depends on all variables and gradually removes them according to a schedule.

1.6 Mining Hard Negatives

As has been pointed out in the computer vision literature, training a model to detect object(s) can produce on the order of 10^5 examples (if not more). The training set $D = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^M \times \{-1, 1\}, i = \overline{1, N}\}$ contains billions of instances, which cannot be stored in the computer memory for training. One approach is to train the model on the positive examples and a subset of the negative examples known as "hard negatives."

1.6.1 Related Works

In [42], Felzenszwalb et al. develop a data-mining algorithm based on the bootstrapping idea for training a SVM. For a training set D and a classifier parameterized by $\boldsymbol{\beta}$, the hard H and easy E examples are defined as:

$$H(\boldsymbol{\beta}, D) = \{\langle \boldsymbol{x}, y\rangle \in D | y f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 1\} \tag{1.20}$$

$$E(\boldsymbol{\beta}, D) = \{\langle \boldsymbol{x}, y\rangle \in D | y f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 1\} \tag{1.21}$$

That is, $H(\boldsymbol{\beta}, D)$ are the examples that are either incorrectly classified or inside the margin defined by $\boldsymbol{\beta}$. The examples that form part of the set $E(\boldsymbol{\beta}, D)$ are both those that are correctly classified and are outside the margin. Since the dual function L_D defined over the training set D is convex, the solution $\boldsymbol{\beta}^*(D) = \text{argmin}_{\boldsymbol{\beta}} L_D(\boldsymbol{\beta})$ is unique. They start training with a subset of the original set D and remove easy examples and add hard examples at each iteration. The new subset is used to update $\boldsymbol{\beta}$. They prove that under certain conditions, the $\boldsymbol{\beta}$ found from the subset of hard examples is the same as $\boldsymbol{\beta}^*(D)$ and the algorithm stops in a finite number of steps.

1.6.2 Hard Data Mining With FSA

Since the Huberized SVM loss (1.15) is zero when $y f_{\boldsymbol{\beta}}(\boldsymbol{x}) > 1+h$, most of the training examples will not contribute to the loss function at its minimum, and could in principle be ignored. This idea has been used in [42] for mining "hard examples", which were defined for the hinge loss as

$$H(\boldsymbol{\beta}, D) = \{(\boldsymbol{x}, y) \in D, y f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 1\} \tag{1.22}$$

For the hinge loss without feature selection (i.e. no constraints on the number of nonzero coefficients), the authors give a procedure in [42] that finds all the hard examples in a finite number of steps and prove that the optimum obtained from minimizing the loss on the hard examples is the same as the optimum obtained from the entire training set D.

The same conclusions can carried over to the Huberized SVM loss (1.15) where the hard examples are

$$H_h(\boldsymbol{\beta}, D) = \{(\boldsymbol{x}, y) \in D, y f_{\boldsymbol{\beta}}(\boldsymbol{x}) < 1 + h\} \tag{1.23}$$

But after imposing the sparsity constraints for feature selection, the Huberized SVM loss is no longer strictly convex and no such guarantees can be obtained. However, the feature selection

consistency of the FSA with the logistic loss and the simulation experiments (shown in the supplementary material) suggest that FSA-SVM will find the true variables with high probability given enough training examples. This means that a large number of training examples is needed for FSA to select the appropriate variables, so it might not be enough to use only the hard examples.

In the case when there is a small number of positives, we use a standard approach that starts with a number N of negatives obtained randomly, and adds to the training set N false positives at each iteration until convergence or a maximum number of iterations. Thus at each iteration the set of negatives increases with harder and harder negatives.

The procedure is described in detail in Algorithm 2 and was found to work very well in practice as it will be seen in experiments.

Algorithm 2 Mining Hard Negatives

Input: Training images .

Output: Trained classifier parameters β.

1: Initialize set of negatives C_1 with $|C_1| = N$
2: **for** t=1 to T **do**
3: Train FSA-SVM with negative set C_t obtaining parameters $\beta = \beta^{(t)}$.
4: Generate set of false positives F_t of cardinality $|F_t| \leq N$ using classifier $f_\beta(\boldsymbol{x})$
5: Set $C_{t+1} = C_t \cup F_t$.
6: **if** $C_{t+1} = C_t$ **then**
7: Stop
8: **end if**
9: **end for**

Observe that when optimizing the loss (1.15) without sparsity constraints for feature selection, Algorithm 2 is equivalent to the mining procedure from [42] without step 3 (shrinking the cache). The proofs of Theorems 1 and 2 from [42] can easily be carried over to this case, which means that the algorithm will terminate in a finite number of steps and the minimum over the whole training set D is the same as the minimum over C_T when T is large enough. Thus without feature selection, Algorithm 2 will find the optimum from the entire set D after a finite number of steps. However, these guarantees cannot be offered under the feature selection sparsity constraints.

1.7 Confidence Weighted FSA

Inspired by [72] and the dataset the originated from that paper, we combined the Confidence-Weighted (CW) algorithm [28] with FSA (CW-FSA). We first summarize the online algorithm developed by Crammer et al. [28] and then explain how we exploited the features generated by this approach with FSA.

1.7.1 Confidence Weighted Algorithm Description

In [28], Crammer et al. introduce a very simple but powerful linear classifier for on-line learning. One of the strengths of their approach as well as one of the differences with passive-aggressive methods [27] is that they calculate the variance for each feature under Gaussian assumptions. Essentially, each feature j is represented as: (μ_j, Σ_j), the sufficient statistics of a Normal distribution. In their paper, Crammer et al. introduce (very easy to implement) closed form update equations for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ using Lagrangian Multipliers. Specifically, CW minimizes the KL divergence between Gaussians (at time t) which leads to the closed-form update of the mean and covariance:

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \boldsymbol{x}_t \tag{1.24}$$

$$\boldsymbol{\Sigma}_t^{-1} \leftarrow \boldsymbol{\Sigma}_t^{-1} + \alpha_t \phi u_t^{-1/2} \text{diag}^2(\boldsymbol{x}_t) \tag{1.25}$$

where the other parameters needed for the update can be found in [28]. Once the $\boldsymbol{\mu}$ have been obtained for all features that will be used to predict a response, the predictions are simply calculated by $\text{sign}(\mu_j \cdot \boldsymbol{x}_j)$. As will be discussed in Section 4.2.1, under certain assumptions, the cumulative error computed from learning with this on-line algorithm decreases with the number of training examples.

1.7.2 CW-FSA

When we combined CW with FSA, the on-line element of CW was lost since FSA discards variables according to a schedule, that is, in an iterative manner. The CW algorithm portion replaced the stochastic gradient update of the $\boldsymbol{\beta}$, after which, FSA was used to choose variables. We provide the structure for CW-FSA in Algorithm 3. Note that in the Update step of Algorithm 3, we intentionally use σ instead of $\boldsymbol{\Sigma}$ since one of the computational simplifications made in [72] is that the features are independent, thus, giving rise to a diagonal $\boldsymbol{\Sigma}$. The α in the update step is

the Lagrangian multiplier, y is the true label of the training example $\boldsymbol{x}$. We keep the M_e features with the smallest uncertainty (σ). The CW-FSA approach generated very good results on the data set URL Reputation [72] which will be discussed in Section 4.2.1.

Algorithm 3 Confidence-Weighted Feature Selection with Annealing (CW-FSA)

Input: Training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with $\boldsymbol{x}_i \in \mathbb{R}^M$.
Output: Trained classifier parameters β.

1: Initialize $\beta = 0$.
2: **for** e=1 to N^{iter} **do**
3: Update $\mu \leftarrow \mu + \alpha y \sigma \boldsymbol{x}$
4: Keep the M_e variables with lowest σ and renumber them $1, ..., M_e$.
5: Set $M = M_e$
6: **end for**

1.8 Experiments with FSA

1.8.1 Stability of Learning Parameters

In this experiment, we evaluate the stability of the FSA Algorithm 1 with respect to its tuning parameters: the learning rate η, the annealing rate μ and the number of iterations N^{iter}. The experiment was conducted on the linearly separable data with M=N=1000, k=k^*=10.

In Figure 1.5 are shown the dependence of the average area under the ROC curve (AUC) with respect to η (left), μ (middle) and N^{iter} (right). For the left plot, we had $\mu = 300, N^{iter} = 500$, for the middle plot $\eta = \mu/10, N^{iter} = 500$ and for the right plot $\mu = 300, \eta = 20$. The obtained curves are the averages of 10 runs.

One can see that all three parameters have a large range of values that yield optimal prediction performance. This robustness property is in contrast to the sensitivity issue of penalty parameters in L_1 or L_0 like methods. It greatly facilitates parameter tuning and reduces ad-hocness.

1.8.2 Simulations

In this experiment, we compare the variable selection and the prediction performance of the FSA algorithm with the Logitboost algorithm and various sparsity-inducing penalties that are popular in the literature. In calling Logitboost for feature selection, we require each weak learner

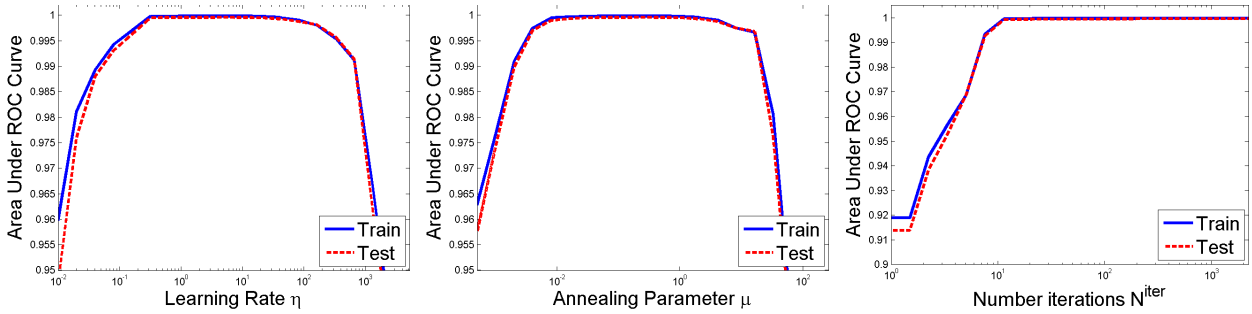

Figure 1.5: Sensitivity analysis for the 3 main FSA parameters (from left to right): η - rate of learning, μ - controls annealing schedule, and N^{iter} - number of epochs. The area under the ROC curve is plotted against a range of values for each parameter.

depends on only one variable. The data for simulations has correlated predictors sampled from a multivariate normal $\boldsymbol{x} \sim \mathcal{N}(0, \Sigma)$ where $\Sigma_{ij} = \delta^{|i-j|}$ and $\delta = 0.9$.

The label y for a data point $\boldsymbol{x} \in \mathbb{R}^M$ is

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{k^*} x_{10i} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.26}$$

Thus only the variables with index $10i, i = \overline{1,k}$ are relevant. We will also use a version of the data with noisy labels, where 10% of the examples had random labels, thus about 5% of the examples have incorrect labels.

Table 1.2: Classification experiments on simulated linearly separable data with $\delta = 0.9$, averaged over 100 runs.

			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1
300	1000	10	29	30	34	0	0	0	0	3	1	0	0	86.1	84.7	86.0	37.9	42.4	40.4	-	64.0	55.6	61.3	23.1
1000	1000	10	100	100	100	1	2	0	0	39	25	44	0	100	100	100	67.8	72.4	49.0	-	88.5	85.7	92.3	26.3
3000	1000	10	100	100	100	30	33	0	0	65	63	97	0	100	100	100	91.1	91.5	60.4	-	95.9	95.4	99.6	29.1
10000	1000	10	100	100	100	88	100	3	0	97	97	100	0	100	100	100	98.8	100	68.4	-	99.6	99.6	100	31.8
1000	1000	30	24	22	21	0	0	0	0	0	0	0	0	93.8	92.4	92.6	47.4	41.5	36.2	-	66.8	61.2	62.4	29.0
3000	1000	30	100	100	100	0	0	0	0	8	14	4	0	100	100	100	78.7	68.6	43.0	-	91.1	91.7	90.4	37.8
10000	1000	30	100	100	100	33	8	0	0	73	56	82	0	100	100	100	97.2	93.9	51.8	-	98.3	97.3	99.3	43.8
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1
300	1000	10	.992	.990	.990	.899	.915	.937	.922	.955	.934	.950	.923	0.03	0.03	0.04	0.02	17	35	.41	87	68	0.13	0.01
1000	1000	10	1.00	1.00	1.00	.947	.951	.950	.962	.965	.953	.967	.936	0.06	0.07	0.15	0.05	434	109	2.5	352	282	0.44	0.09
3000	1000	10	1.00	1.00	1.00	.987	.982	.962	.983	.973	.976	.971	.939	0.23	0.15	0.49	0.21	705	315	6	1122	1103	1.3	0.18
10000	1000	10	1.00	1.00	1.00	.998	.997	.972	.995	.979	.979	.971	.942	1.8	1.8	1.8	1.4	2151	962	20	3789	3725	4.9	0.49
1000	1000	30	.996	.995	.995	.919	.923	.943	.964	.954	.937	.956	.936	0.13	0.08	0.29	0.05	240	150	2.3	358	293	1.2	0.16
3000	1000	30	1.00	1.00	1.00	.969	.954	.955	.984	.979	.976	.975	.948	0.26	0.2	1.10	0.29	565	395	6	1840	1139	4.1	0.48
10000	1000	30	1.00	1.00	1.00	.997	.985	.965	.996	.987	.984	.980	.956	3.5	3.3	3.5	2.0	3914	1265	20	3860	3710	14	1.5

Table 1.3: Classification experiments on simulated data with noisy labels, $\delta = 0.9$, averaged over 100 runs.

			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1
300	1000	10	0	0	1	0	0	0	0	0	0	0	0	44.5	38.9	43.7	30.7	41.2	35.2	-	46.7	45.8	47.8	21.8
1000	1000	10	45	45	86	0	0	0	0	17	8	21	0	92.5	91.4	98.5	58.8	65.3	44.8	-	81.2	78.9	84.4	25.4
3000	1000	10	100	100	100	20	22	0	0	66	58	91	0	100	100	100	88.2	87.8	53.9	-	95.5	94.2	99.1	29.1
10000	1000	10	100	100	100	100	92	2	0	95	95	100	0	100	100	100	100	99.2	65	-	99.5	99.5	100	31.4
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	49.5	45.0	53.7	34.9	40.0	35.1	-	47.5	47.3	48.8	26.7
3000	1000	30	12	14	68	0	0	0	0	2	5	1	0	92.4	92.3	98.7	67.5	63.7	40.5	-	84.0	83.9	82.8	32.9
10000	1000	30	99	99	100	7	0	0	0	60	49	60	0	100	100	100	93.7	90.3	47.5	-	97.5	96.8	98.3	40.7
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1	FSA	FSV	FSL	QTP	L1	EL	L2	MCP	SCD	LB	LB1
300	1000	10	.890	.868	.885	.834	.880	.889	.834	.877	.865	.885	.863	0.04	0.04	0.04	0.04	22	67	.44	37	65	0.17	0.02
1000	1000	10	.943	.940	.946	.890	.907	.902	.892	.914	.906	.915	.888	0.14	0.13	0.14	0.12	412	120	2	218	211	0.53	0.06
3000	1000	10	.950	.950	.950	.935	.934	.913	.928	.927	.923	.924	.895	0.49	0.46	0.48	0.36	1094	321	8	385	367	1.6	0.17
10000	1000	10	.950	.950	.950	.950	.949	.923	.939	.933	.932	.924	.897	2.0	2.0	2.1	1.6	13921	940	20	653	599	5.3	0.5
1000	1000	30	.905	.889	.904	.845	.885	.895	.895	.876	.873	.898	.887	0.30	0.29	0.30	0.17	791	142	1.9	246	239	1.6	0.17
3000	1000	30	.945	.943	.949	.906	.911	.907	.929	.926	.919	.925	.902	1.0	1.0	1.0	0.55	1862	404	7.9	522	504	4.7	0.48
10000	1000	30	.950	.950	.950	.942	.938	.916	.940	.937	.933	.932	.908	3.8	3.8	3.8	2.1	15949	1213	21	763	719	16	1.6

The experiments are performed on the linearly separable data and its noisy version described above. The algorithms being compared are:

- FSA - The FSA Algorithm 1 for the logistic loss (1.12) with the $\mu = 300$ annealing schedule, $\eta = 20$.
- FSV, FSL - The FSA Algorithm 1 for the SVM loss (1.14) and Lorenz loss (1.16) respectively, with the $\mu = 300$ annealing schedule, $\eta = 1$.
- L1 - The interior point method [61] for L_1-penalized Logistic Regression using the implementation from *http://www.stanford.edu/∼boyd/l1_logreg/*. To obtain a given number k of variables, the value of the L_1 penalty coefficient λ is found using the bisection method [18]. The bisection procedure calls the interior point training routine about 9 times until a λ is found that gives exactly k nonzero coefficients. Then an unpenalized model was fitted on the selected variables..
- EL - Elastic net on the Logistic loss with $L_1 + L_2$ penalty using the stochastic gradient descent algorithm. We used the Python implementation `sklearn.linear_model.SGDClassifier` of [105], 1000 epochs for convergence, and the bisection method for finding the appropriate L_1 penalty coefficient. After feature selection, the model was refit on the selected variables with only the L_2 penalty $\alpha = 0.001$.
- L2 - SVM using the Python implementation `sklearn.linear_model.SGDClassifier` with 1000 epochs, and choosing the L_2 penalty coefficient $\alpha \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ that gave the best result.

- QTP - The quantile TISP algorithm with 10 thresholding iterations and 500 more iterations on the selected variables for convergence.
- MCP, SCD - Logistic regression using MCP (Minimax Concave Penalty)[120] and SCAD penalty respectively. Two implementations were evaluated: the `ncvreg` R package based on the coordinate descent algorithm [13] and the `cvplogistic` R package based on the the Majorization-Minimization by Coordinate Descent (MMCD) algorithm [57]. The `cvplogistic` package obtained better results, which are reported in this thesis.
- LB - Logitboost using univariate linear regressors as weak learners. In this version, all M linear regressors (one for each variable) are trained at each boosting iteration and the best one is added to the classifier.
- LB1 - Similar to LB, but only 10% of the learners were selected at random and trained at each boosting iteration and the best one was added to the classifier.

In Tables 1.2 and 1.3 are shown the *all-variable* detection rate (DR) and the average percent of correctly detected variables, (PCD) obtained from 100 independent runs. The PCD is the average value of $|\{j, \beta_j \neq 0\} \cap \{j, \beta_j^* \neq 0\}|/k^* \cdot 100$. A more stringent criterion is the DR which is the percentage of times when all k^* variables were correctly found i.e. $\{j, \beta_j \neq 0\} = \{j, \beta_j^* \neq 0\}$. The average area under the ROC curve on unseen data of same size as the training data, and the average training times for the methods being evaluated are also shown in Tables 1.2 and 1.3.

CHAPTER 2

FACE DETECTION USING A 3D MODEL

2.1 Related Work

One of the areas we apply our FSA framework is to object detection. This is still an area of active research due to the complexity of detecting objects in noisy or occluded settings, dealing with images of varying sizes, attempting to infer the 3D shape from 2D information, and the tremendous amount of data that can be generated from negative examples. Specifically, we have made contributions in detecting faces and their pose. In the related work to follow, we divide the literature review into two sections: Face Detection (where the methodology must detect the face with no prior information) and Face Alignment (where the location of a face is roughly known before the methodology is applied).

2.1.1 Face Detection

Felzenszwalb et al. develop a method for object detection using mixtures of multi-scale deformable parts [42]. In particular, from this paper, we extend the idea of mining the data for hard examples using the SVM. However, since we perform feature selection, we end up with a non-convex loss function. Thus, their theorems guaranteeing

1. that the solution obtained over the subset of hard examples is exactly the same as the solution obtained over the entire training set
2. the number of mining iterations is finite

do not necessarily extend to our methodology. However, the experimental results on large real-world datasets will demonstrate the effectiveness of our approach.

Zhu and Ramanan present a computationally tractable tree-based method [126] that uses a unified model approach for face detection, pose estimation, and landmark estimation in real-world data. Their model is based on a mixture of trees with a shared pool of parts V. These parts represent facial landmarks. They use the mixture of trees to capture changes in face topology due to a change in viewpoint. Furthermore, the use of tree mixtures allow them to find the globally

best configuration of parts via an inner maximization achieved via dynamic programming. They achieve comparable results with commercial state-of-the-art systems while training on hundreds of examples compared to the billions used in commercial software. We compare our approach against this paper on the same dataset they used and obtain better results.

In summary, many works [42, 52, 126] use computationally tractable tree-based models to represent the interactions between the locations of the object parts. In this thesis, we explore a model where the unknown face part locations are fully connected with each other into a simplex parameterized by the projected similarity parameters $\theta = (\boldsymbol{u}, s, R)$, as illustrated in Figure 2.12. Even though the proposed inference algorithm is not globally optimal, the model more than compensates this disadvantage, as shown in experiments.

In [52] a 2D part configuration is detected using a version of the deformable part model [42] and then a 3D pose and shape is inferred from the 2D configuration. In contrast, our work directly uses the 3D pose to represent the relative positions of the parts without going through an intermediate 2D model.

There have been 3D approaches to object detection [70, 85, 102] that use different types of features that are extracted at certain positions depending on the object pose. Our approach uses a different 3D representation on the object parts (keypoints) and probability model than these works, it does not need any synthetic 3D models, and uses a different inference algorithm. Moreover, none of these works was used for face detection.

2.1.2 Face Alignment

In [104], Sun et al. cascade three levels of convolutional networks in order to make refined predictions, at each level, of 5 face kepypoints. The first layer makes initial predictions of the keypoints using high-level (texture based) features. This first layer has a deep structure (4 convolutional layers) since it is making predictions over the entire face. The two remaining layers refine the initial predictions by using different convolutional networks which are fused for improved accuracy and reliability. Sun et al. point out that, unlike prior methods, they do not use the same regressor at different cascade levels. These two levels are shallow relative to the first since they receive as input small patches (thus a low-level task) centered around the initial keypoint predictions. These patches shrink as they pass through the cascades. The final prediction of a keypoint is computed by averaging the predictions made throughout the cascades. Figure 2.1 illustrates their algorithm

where the first row represents the keypoint predictions (as green dots) at the first level and the second row shows the refined predictions at the second and third level. The keypoints circled in a blue circle are examples of the improvement in predictions obtained through the multiple levels.

Figure 2.1: A figure from [104] that demonstrates the keypoint location predictions at different levels of the convolutional neural network. The first row corresponds to the first layer and the second row corresponds to sequential layers with improvement accentuated using blue circles.

Pose candidates have been previously proposed by image based regression in the shape regression machine [123] and for face alignment [20, 19]. However, they are not based on a 3D model, are not geared for face detection and don't predict the candidates from the keypoint locations and use other regression methods than this work. The face alignment by cascaded regression [20, 19] contain interesting ideas that could be adapted for refining the 3D pose candidates and further improve the performance of our algorithm.

Most face keypoint detection approaches such as [19, 20, 104] assume that the face has already been detected and detect the keypoints relative to the face bounding box. In order for such an approach to work it is necessary that the face detector be very good because if the face is missed, so are the keypoints. In this thesis, as in [126], we argue for an approach where the parts are detected together with the face, in a joint inference algorithm. Such a joint approach is more expensive than the cascade approach where the face is detected first, but it can be easily parallelized since the face part responses can be computed independently.

Image-based regression is used in face alignment approaches and we build off the work by Zhou and Comaniciu [123] where they present a method called *Shape Regression Machine* (SRM). They segment the left ventricle (LV) endocardium from an echocardiogram (medical image) in real-time. They dealt with the challenge posed by LV which can manifest arbitrary scale and orientation by developing a regression-based approach which took advantage of the computational efficiency of Haar features. In particular, they claim that their method only requires on scan (of the image), at least in theory. The basic idea is that they randomly sample image patches and then try to regress the location and shape of the LV using regression stumps (similar to [109]). We use image-based regression in order to generate potential 3D poses.

2.2 Image Features

The literature is rich with different kinds of image features that can be extracted to perform face detection. We focus on implementing four kinds of image features: Haar, Hogs, Difference Features (DF) and Local Binary Features (LBF). We describe these features in detail since they will be referenced in later sections. While there exist many kinds of image features that we did not use, and we might in the future, the purpose of this work is not to evaluate the performance of image features for object detection. Instead, we provide a framework where robust object detection can be performed. Indeed, our performance can be improved with more discriminating image features.

2.2.1 Histogram of Oriented Gradients

As Dalal and Triggs explain in [30], the main idea behind Histogram of Oriented Gradients (HOG) is that the contours of an object can be summarized robustly by the edge directions without knowing where these edges are located. The process by which these HOG *descriptors* are computed can be outlined as:

1. The image is divided into rectangle cells (8×8 in [30]).
2. For each cell, a histogram of gradient/edge directions is computed.
    - These gradients are computed by applying a derivative mask at each pixel in the cell.
3. Several adjacent cells are considered a block (16×16 in [30]). Robustness to changes in illumination, contrast, etc., can be achieved by concatenating the histograms from all the cells within one block and normalizing them.

- For detecting humans, the authors noted that they achieved better detection performances by restricting the angles between $[0°, 180°]$

These normalized descriptor blocks are referred to as HOG descriptors. As visualization of HOG features for a person, see Figure 2.2

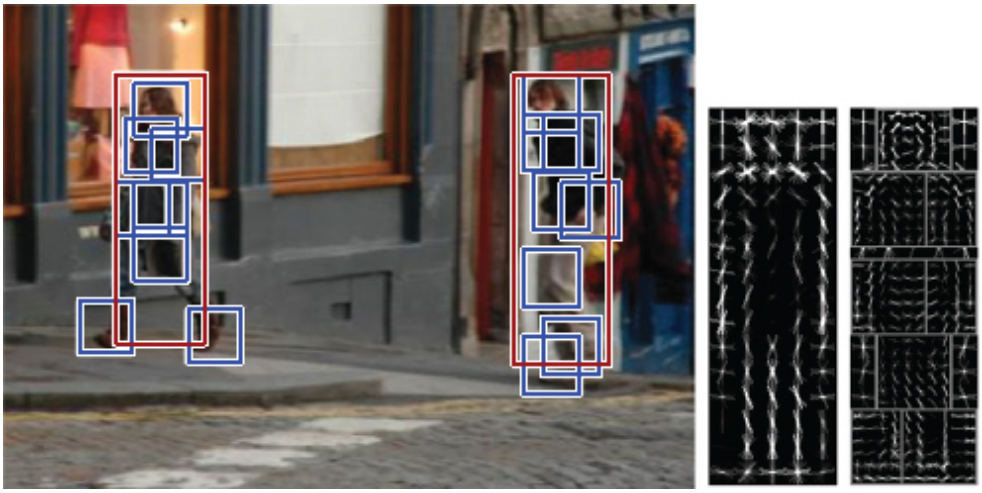

Figure 2.2: From [42] where the HOG features for each person identified view overlapping rectangles are demonstrated in the two rightmost images.

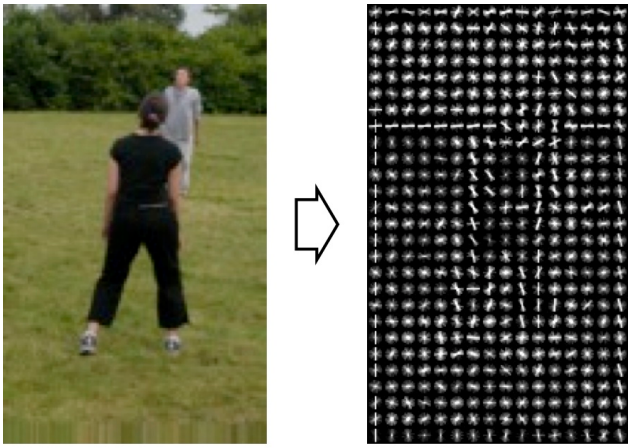

Figure 2.3: Illustration of HOG Features from **http://www.cs.cornell.edu/courses/cs4670/2012fa/projects/p5/index.html**

2.2.2 Haar-Like

We use Haar-like features as described by Viola-Jones [109]. The basic idea is that adjacent rectangles are applied to an image patch. It is common in the literature to visualize these adjacent rectangles as black and white rectangles, as in Figure 2.4. Keeping this figure in mind, the Haar feature will be the sum of the pixel values that fall into the black rectangle minus the sum of the pixel values that fall into the white rectangles.

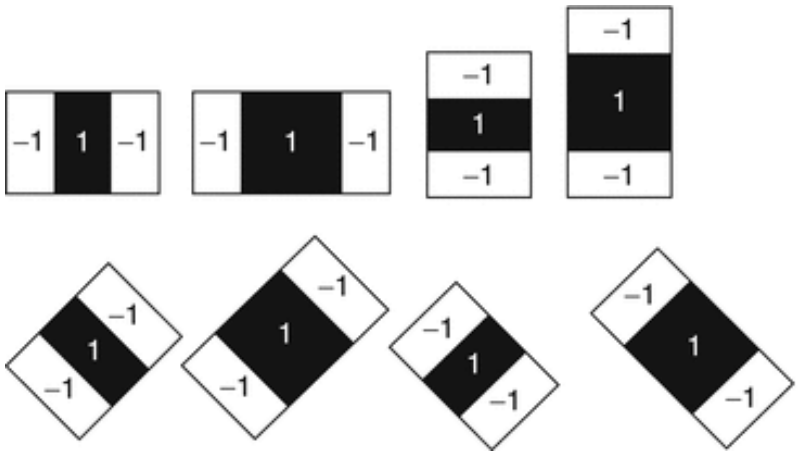

Figure 2.4: The rectangles in this figure (taken from **http://link.springer.com**) are commonly used to represent how pixel values will be treated as either negative or positive and then summed. These sums are used in forming Haar features.

The window that is comprised of these black and white rectangles slides across the image and the signed pixel values are summed. These summations can be used to determine how likely an image patch represents part of a face. In fact, these features proved to be a significant step towards detecting faces in real-time. One of the reasons Haar features turned out to be so effective was do to the observation that the eyes tend to be darker compared to other parts of the face. Thus, Haar features that exploit this property can be used to successfully detect a face. Examples of Haar features used for detecting faces are provided in Figure 2.5 taken from `https://code.google.com/p/scoialrobot/wiki/RGBDetector`.

2.2.3 Difference Features

The LBF are based on image pixel-differences [35, 82, 101, 21]; that is, the intensity difference of two pixels in the image. These features provide enough data to discriminate between keypoints

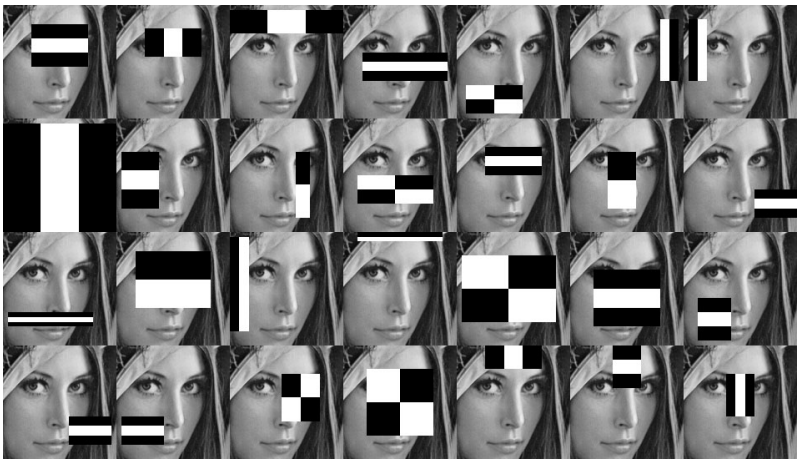

Figure 2.5: Examples of different kinds of adjacent rectangles and locations that can be used to form Haar features. Taken from **https://code.google.com/p/scoialrobot/wiki/ RGBDetector**

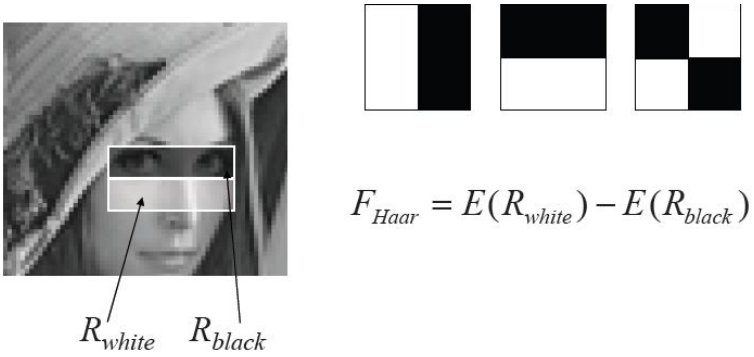

Figure 2.6: Example of summing signed pixel values as part of forming Haar features. Image is from **http://maraya.karo.or.id/haar-like-feature-pada-metode-viola-jones**

and are computationally efficient. The latter characteristic is very important to our work since we aim to compete against both the academic and industry standard. Figure 2.7 provides a visual example of how these pixel differences are computed on a grid centered at a keypoint. Depending on the radius set by the user, the grid will grow or decrease in size; thus, dictating how much data is generated. In Figure 2.7, the 21 face keypoints (elaborated in Section 2.3) are color coded on faces. The pixel difference grid for the right ear, with a radius of 4, is represented with a green grid. We provide further illustrations of the difference features over the 9 keypoints we use in our face detection process in Section 2.5.

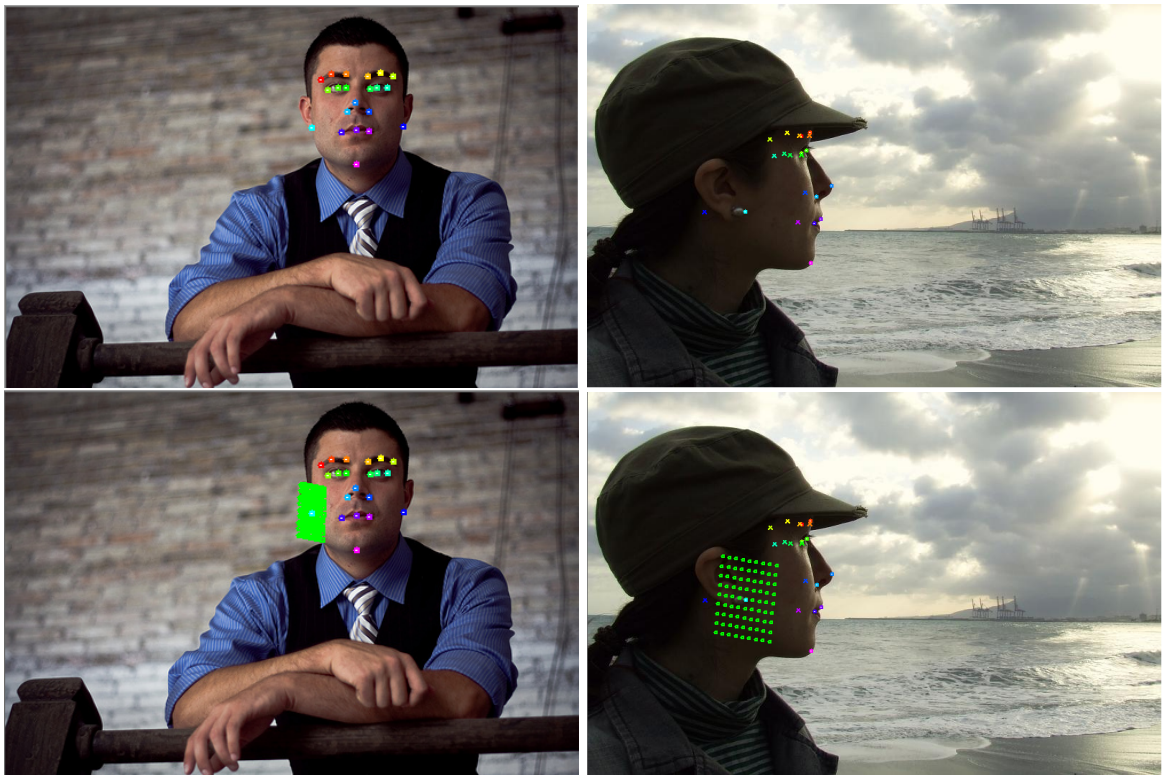

Figure 2.7: Visualizing the grid over which the pixel differences are computed for keypoint of the lower right ear.

2.2.4 Local Binary Features

In [92], Ren et al. argue that under certain conditions, the "signal-to-noise ratio" can be increased if the task of learning keypoints is kept local instead of global. Each local region is represented via a feature mapping function, Φ_ℓ^t - where ℓ represents a specific keypoint and t is the refinement iteration of their face detector, which they implement as regression random forest [15]. The pixel-differences are used to train the split nodes. Ren et al. explain that they select the features that provide the greatest reduction in model variance. To understand how we end up with binary features, suppose that after training a random forest, we sample pixel differences from a test image. This sample will traverse the tree until it reaches a leaf. Depending on which leaf it reaches, this leaf will be associated the value 1 and the rest with 0. Thus, the leaves in each tree of the random forest will be associated with a vector zeros and one 1. The vectors induced by each tree are concatenated in order to predict the location of the keypoints at a global level (i.e. global

shape estimation) via linear regression. Figure 2.8 is taken directly from the paper by Ren et al. which they use to visually explain how the LBF are computed.

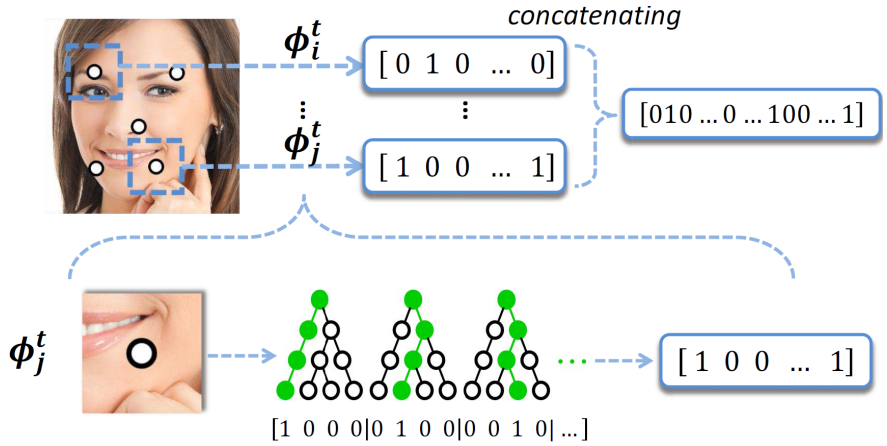

Figure 2.8: Visual example of how LBF are computed as presented in [92]

To further explain how LBF are computed and used, we provide more examples and expound on them. For every color channel, a grid of pixel differences are computed around an estimated keypoint location. A feature vector of length $N \times (N-1)/2$ is generated and it is this feature vector that traverses a tree. For each keypoint, a total of $N \times (N-1)/2 \times 3$ of pixel difference features are generated, as can be seen in Figure 2.9. To see this, we use the coordinate system $(x, y, Channel, Keypoint)$ where x and y represent the x and y coordinates of the pixel to which intensity differences are computed relative to, $Channel$ represents the RGB channel being used (represented as $0 = R, 1 = G, 2 = B$), and the last coordinate representing the keypoint in question. Each tree in the random forest is used to predict the location of a single keypoint.

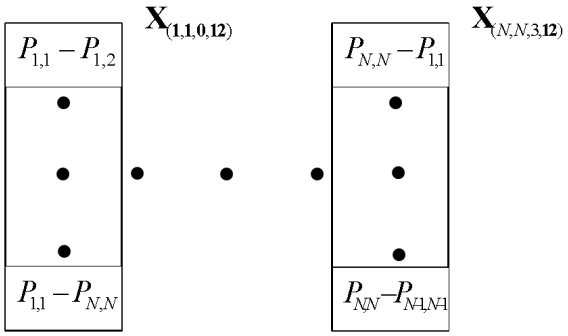

Figure 2.9: Visual example of pixel differences feature vectors

We note that each tree is associated with a single keypoint. Once the feature vector reaches the leaves, a binary vector whose length is the same size as the number of leaves is generated. Since we use trees of depth 5, there are 16 leaves. Thus, we end up with a binary vector of length 16 where only one of the values is 1, which corresponds to the destination of the feature vector. Each leaf is associated with a predicted keypoint location (d_x, d_y). The location predictions are averaged per keypoint for the final prediction. Since 100 trees are used with a depth of 5, this means that $100 \times 16 \times 9 = 14,400$ LBF are generated. Thus, approximately 160 LBF are generated per keypoint and about 9 trees are used to predict the location of a keypoint.

Figure 2.10 demonstrates an example for a single decision tree for the keypoint 12. The coordinates next to the nodes represent pixel-difference features that are used to "ask a question". That is, when a intensity difference vector from a test image traverses the tree, it is those training feature vectors that are used to determine the route of the test feature vector.

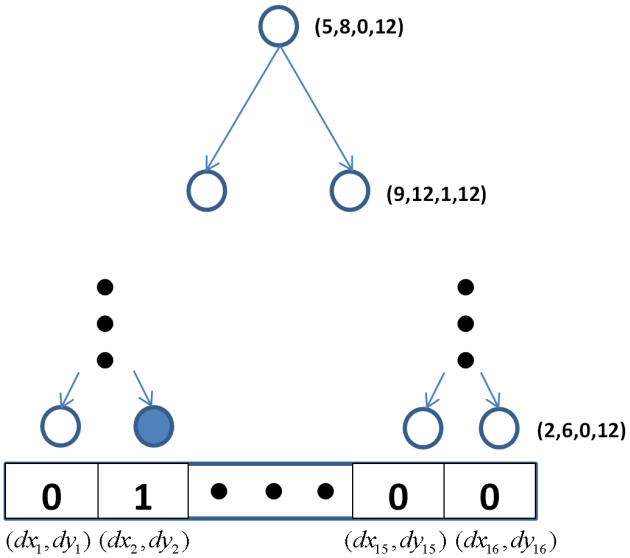

Figure 2.10: Visual example of how binary features are computed

2.3 Energy Model

Given an image, the goal is to find the faces and their 3D poses. The face 3D pose is represented as a projected rigid transformation T. Recall that a rigid transformation is a transformation that when applied to a vector $\boldsymbol{x}$, a transformed vector $T(\boldsymbol{x})$ of the form $T(\boldsymbol{x}) = R\boldsymbol{x} + \boldsymbol{u}$ is generated.

The matrix R is an orthogonal transformation and $\boldsymbol{u}$ is a translation from the origin. A proper rigid transformation is one where the determinant of R is 1 which indicates that R does not produce a reflection (and hence is an orientation-preserving orthogonal transformation).

This projected rigid transformation $T_\theta : \mathbb{R}^3 \to \mathbb{R}^2$ is defined as

$$T_\theta(\boldsymbol{x}) = \boldsymbol{u} + s\pi(R\boldsymbol{x}) \tag{2.1}$$

with parameters $\theta = (\boldsymbol{u}, s, R)$ consisting of a 2D translation $\boldsymbol{u} \in \mathbb{R}^2$, a scale s and a 3D rotation matrix R. The projection onto the (x, y) plane is represented by $\pi : \mathbb{R}^3 \to \mathbb{R}^2$, $\pi(x, y, z) = (x, y)$.

The face has L keypoints that form a rigid 3D configuration that can be written as a $3 \times L$ matrix $F = (F_1, \ldots, F_L)$ where $F_i \in \mathbb{R}^3$. We obtain the 2D configuration of the face keypoints in an image as

$$\{T_\theta(F_i) + \epsilon_i,\ i = \overline{1, L}\} \tag{2.2}$$

where T_θ is the 3D face pose (as previously defined) and $\epsilon_i \in \mathbb{R}^2$ are independent deformations for each keypoint. An example is depicted in the image below. The neon green dots that are connected in a simplex represent hypothetical keypoints. Each keypoint has a dot of a different color connected to it which represents a deformation of each keypoint's position. In this hypothetical case, there are 6 neon-green points which means there are 6 keypoints. This means that the 3D configuration matrix is a 3×6 matrix where each column of this matrix represents the $3D$ pose for each point.

$$F = \begin{array}{c|cccc} & F_1 & F_2 & \cdots & F_6 \\ \hline x & x_{1,1} & x_{1,2} & \cdots & x_{1,6} \\ y & y_{2,1} & y_{2,2} & \cdots & x_{2,6} \\ z & z_{3,1} & z_{3,2} & \cdots & x_{3,6} \end{array}$$

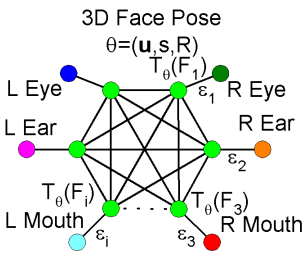

Figure 2.11: Example of keypoint representation as a column vectors (left table) and rigid model (right figure)

Thus, the 2D configuration for one of these points is found as:

$$\begin{aligned} T_\theta(F_i) &= \boldsymbol{u} + s\pi(R \cdot F_i) \\ &= [\mu_x, \mu_y]^T + s\pi(R \cdot [x_{1,i}, y_{2,i}, z_{3,i}]^T) \\ &= [\mu_x, \mu_y]^T + s[x', y']^T \end{aligned}$$

For any $\theta = (\boldsymbol{u}, s, R)$ let B_θ be the bounding box of the set of transformed keypoints $\{T_\theta(F_i), i = \overline{1, L}\}$. In other words, once we know the 3D pose T_θ, we can obtain a bounding box on that face using its 2D configuration (i.e. face detection). As will be explained in a later subsection, many 2D configurations will be generated for each face. We choose the best configuration via an energy minimization in a Bayesian framework as described in Equation 2.3.

$$(\theta_1, \ldots, \theta_n) = \underset{n, \theta_1, \ldots, \theta_n}{\operatorname{argmin}} E(n, \theta_1, \ldots, \theta_n) \tag{2.3}$$

$$E(n, \theta_1, ..., \theta_n) = E_{data}(\theta_1, ..., \theta_n) + E_{prior}(n, \theta_1, ..., \theta_n) \tag{2.4}$$

The data term depends directly on the detections of the L face keypoints in the image. A face scoring function $S(\theta_j)$ is used to determine how many keypoints are correctly detected. A parameter τ set a priori determines the minimum score needed for a valid detection. This data term is represented formally as:

$$E_{data}(\theta_1, ..., \theta_n) = \sum_{j=1}^{n} (\tau - S(\theta_j)) \tag{2.5}$$

where n represents the number of 3D poses generated. Even though we optimize over n, we use the maximum value of n to determine the number of candidates. The prior $E_{prior}(n, \theta_1, ..., \theta_n)$ enforces the constraint that the bounding boxes $B_{\theta_j}, j = \overline{1, L}$ generated for each 3D pose have small overlap with each other.

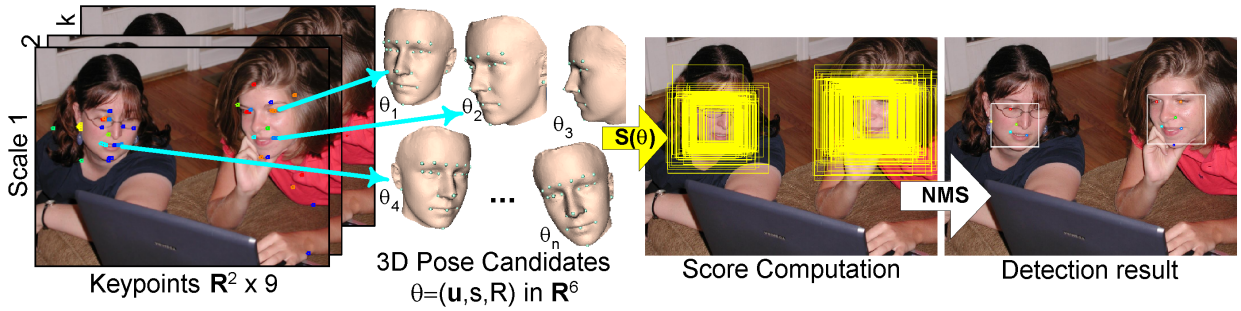

Figure 2.12: Face detection using a 3D model. The face keypoints are detected independently and used to propose 3D pose candidates $\theta = (\boldsymbol{u}, s, R) \in \mathbb{R}^6$. The 3D pose candidates are evaluated using the score $S(\theta)$ based on the detected keypoints. The detected faces are obtained by non-max suppression.

2.4 Inference Algorithm

The inference algorithm, a bottom-up face detection process which is illustrated in Fig. 2.12, contains the following steps:

1. Face keypoints are detected independent of each other.
2. Face 3D pose candidates are generated from the keypoints, obtaining 3D pose candidates $\theta_1, ..., \theta_n$.
3. For each 3D pose candidate θ_j, the face keypoints P_j are predicted using the 3D pose.
4. Face scores $S(\theta_j), j = \overline{1, n}$ are computed and low scoring candidates are removed.
5. Non-maximal suppression is applied to output a set of high score candidates that satisfy the overlap constraints, greedily minimizing the energy $E(n, \theta_1, ..., \theta_n)$.

We expand on these steps in the following subsections.

2.4.1 Detecting Keypoints

Both the keypoints and pose candidates are detected over a range of scales. The 3D pose candidates θ_i are generated by image-based regression from the detected keypoint locations. Since these θ_i depend on the detected keypoint detectors, we start by describing the keypoint detection.

Keypoint Detection. The face keypoints are detected using a sliding window classifier (see Figure 2.13) on a Gaussian pyramid (see Figure 2.14 for examples) with 4 scales per octave (i.e. resized by powers of $2^{1/4}$) down to a minimum of 24×24 pixels. This leads to a representation of the face keypoints as (x, y, s) where (x, y) is the pixel location and s is the index in the pyramid of the image containing the point. Appendix B provides more details on the use of Gaussian Pyramid in our work.

Feature Pool. The keypoint classifiers are trained using a feature pool consisting of $288 \times 3 = 864$ Histograms of Oriented Gradients [30](HOG) features and $61,000$ Haar features (see Section 2.2 for more details) extracted from the RGB channels in a 24×24 pixel window centered at the point of interest (x, y) in the appropriate image of a Gaussian pyramid.

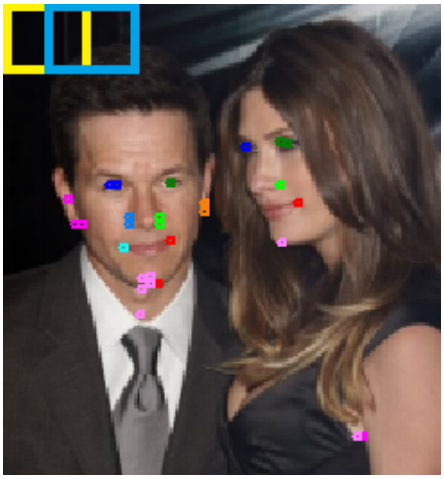

Figure 2.13: Illustration of Window Classifier

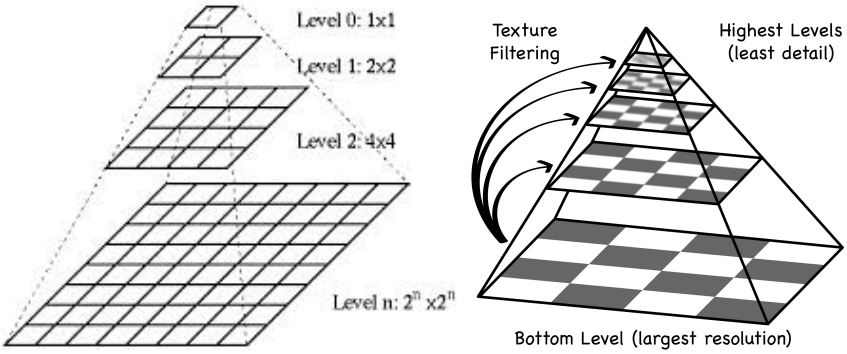

Figure 2.14: Illustration of Gaussian Pyramids. Left image from **http://fourier.eng.hmc.edu/e161 /lectures/canny/node3.html**. Right image from **https://elementalray.wordpress.com/ 2012/04/**

Training Details. Given our bottom-up approach to face detection, it is essential that we detect as many keypoints as possible since a single keypoint may fail. Furthermore, each keypoint implicitly contains information about the 3D pose. We point out that an advantage of using keypoints to predict face pose is that they exhibit smaller variation compared to the whole face.

As previously mentioned, the feature pool used to predict the 2D location of keypoints consists of $61,000$ Haar features and 864 Hog features. Each image is re-scaled using a Gaussian pyramid scheme over 4 octaves. The trainin set consists of $60,000$ positive examples and 2.5 billion negatives. Given memory limitations of 24 GBs, we are restricted to about $300,000$ training examples. We deal with memory constraints by implementing hard data mining in order to train on the most challenging negative examples. For example, in Figure 2.15, there are negative detections in the

Figure 2.15: Examples of training examples for keypoint detection. The positive examples (correct detections on the faces) and negative examples (e.g. detections in the trees) of keypoints.

trees. These negative examples may not prove to be very useful in improving the discriminatory power of our classifier. Thus, we use hard data mining to fill our limited memory space with the most relevant negative examples, such as examples on the face near keypoints that could confuse our classifier.

More specifically, we train a FSA classifier (see Algorithm 1) that minimizes the Lorenz loss of Equation (1.16). We initially train this classifier with all the positive examples and a subset of the negative examples. Then, with this trained classifier, we perform 10 iterations of hard data mining where each iteration sees about $20,000$ negatives added. This process is carried out for each of the 9 keypoints we use to detect a face.

2.4.2 Generating 3D Pose Candidates

Since the keypoints are detected for faces in a range of scales, the pose candidates are also obtained for faces in the same range. Specifically, the 3D pose candidates are generated by image-based regression from the detected keypoint locations. The 3D pose $\theta = (\boldsymbol{u}, s, R)$ has six parameters $(\boldsymbol{u}, s, \boldsymbol{\phi}) = (u^x, u^y, s, \varphi^x, \varphi^y, \varphi^z)$, where $\boldsymbol{\phi} = (\varphi^z, \varphi^x, \varphi^y)$ is the roll-pitch-yaw decomposition of the rotation matrix R.

Image based regression. The pose is predicted from a point (x_0, y_0, s_0) by predicting the relative vector

$$\boldsymbol{y} = (u^x/s_0 - x_0, u^y/s_0 - y_0, s/s_0, \varphi^x, \varphi^y, \varphi^z) \tag{2.6}$$

Training details. Given the 3D configuration F (of Equation (2.2)) and 2D coordinates P of the (previously found) keypoints, we obtain the ground truth 3D pose of each face by least squares energy minimization of Equation (2.7).

$$E(\boldsymbol{u}, s, R) = \|\boldsymbol{u}\mathbf{1} + s\pi(RF) - P\|^2 \tag{2.7}$$

We describe the energy minimization of Equation (2.7) through Algorithm 4. We note that the POSIT algorithm [33] could also be used for this purpose.

Algorithm 4 Fit Rigid Projection

Input: $3 \times L$ matrix F and $2 \times L$ matrix P.
Output: $\boldsymbol{\theta} = (\boldsymbol{u}, s, R)$

1: Initialize $L \times 3$ matrix $B = (P^T, 0)$.
2: **for** e=1 to N^{iter} **do**
3: Call Algorithm 5 to find $\boldsymbol{u}, s, R$ that minimize $\|\mathbf{1}^T\boldsymbol{u}^T + sF^TR - B\|^2$
4: Extract third column $\mathbf{c}_3 = (C_{i3})_i$ of $C = sF^TR$
5: Update $B = (P^T, \mathbf{c}_3)$
6: **end for**
7: Change R to R^T and discard the z-component of $\boldsymbol{u}$.

Note that in Algorithm 5, the input matrix A is F^T from Algorithm 4 and the input matrix B is the same B from Algorithm 4. Thus given these two matrices, we use Algorithm 5 to find the $\boldsymbol{\theta} = (\boldsymbol{u}, s, R)$ that minimizes Equation (2.8). On average about $3,000$ face poses $\boldsymbol{\theta}$ will be generated per face.

$$\|\mathbf{1}^T\boldsymbol{u}^T + sAR - B\|^2 \tag{2.8}$$

Next, the ground truth vectors for training the 3D pose regressors are obtained as in Equation (2.6) for each annotated face from the fitted 3D pose $(\boldsymbol{u}, s, \boldsymbol{\phi}) = (u^x, u^y, s, \varphi^x, \varphi^y, \varphi^z)$ and the keypoint location (x_i, y_i, s_i). A specific 6D pose regressor is trained for each keypoint, using the same feature pool as the keypoint detectors. The regressors for the 3D pose candidate generators are trained using FSA for the square loss

Algorithm 5 Fit Rigid Transformation

Input: Matrices A, B of size $p \times d$.

Output: $\boldsymbol{\theta} = (\boldsymbol{u}, s, R)$ to minimize $\|\mathbf{1}^T\boldsymbol{u}^T + sAR - B\|^2$

1: Compute the column means $\bar{\alpha} = \mathbf{1}A/p, \bar{\beta} = \mathbf{1}B/p$ and column centered matrices $A^* = A - \mathbf{1}^T\bar{\alpha}$ and $B^* = B - \mathbf{1}^T\bar{\beta}$

2: Decompose $A^{*T}B^* = UDV^T$ by SVD, where U, V are rotation matrices and D is a diagonal matrix.

3: Obtain $R = UV^T, \boldsymbol{u} = \bar{\beta} - s\bar{\alpha}R$ and $s = tr[R^TA^{*T}B^*]/tr(A^{*T}A^*)$

$$L_D(\boldsymbol{\beta}) = \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{f}_{\boldsymbol{\beta}}(\boldsymbol{x}_i)\|^2 + \sum_{j=1}^{M} \rho(\boldsymbol{\beta}_j) = \sum_{i=1}^{N}\sum_{k=1}^{d}(y_i^k - f_{\boldsymbol{\beta}}^k(\boldsymbol{x}_i))^2 + \sum_{j=1}^{M} \rho(\boldsymbol{\beta}_j) \tag{2.9}$$

where $\boldsymbol{y}_i = (y_i^1, ..., y_i^d) \in \mathbb{R}^d$ and $\boldsymbol{f}_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = (f_{\boldsymbol{\beta}}^1(\boldsymbol{x}_i)...f_{\boldsymbol{\beta}}^d(\boldsymbol{x}_i)) \in \mathbb{R}^d$ are piecewise linear in each dimension. Other regression methods could be employed to generate the 3D pose candidates, for example regression forests [29] or boosted regression [123].

2.4.3 The 3D Face Candidate Score $S(\theta)$

The score function $S(\theta)$ depends directly on the detections of the L face keypoints in the image. A sliding window classifier or a Hough Forest [44] could be used to detect the face keypoints on a Gaussian pyramid. The detected keypoints are rescaled to the original image scale, obtaining for each keypoint $i \in \{1, ..., L\}$ a set K_i of detections.

Modified LBF. To obtain the face score for a candidate $F = (\theta)$, we modified the LBF features [92] so that they align based on the 3D pose θ instead of the 2D shape. For that, an approximate tangent plane at each keypoint on the 3D face model is assigned a system of coordinates. This coordinate system is projected to a 2D coordinate system based on the 3D pose, which is used to define a skewed point grid centered at the predicted keypoint location pi, as illustrated in Figure 5. The modified LBF features are then obtained as the leaf indexes for the Random Forest trees trained with these modified features. The LBF were trained with 100 Random Trees of depth 6 for each of the 9 keypoints, for a total of $100 \cdot 32 \cdot 9 = 28,800$ features. Let $\boldsymbol{x}(\theta)$ be the vector of LBF features extracted from the image for the candidate $F = (\theta)$.

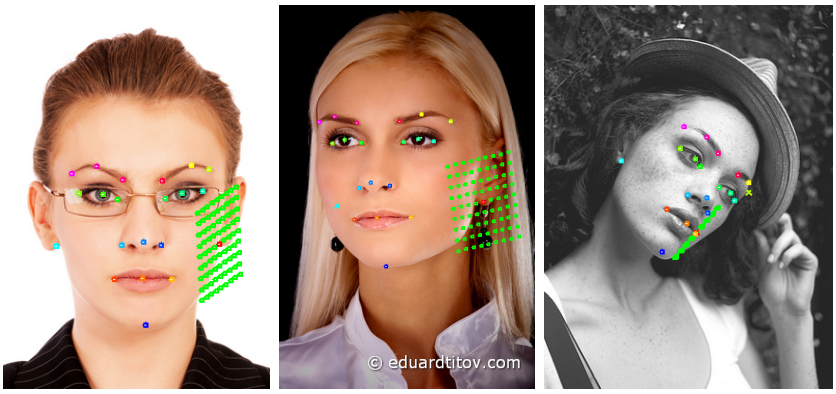

Figure 2.16: Examples of LBF sampling grid patterns obtained using the 3D pose of the face.

Then, for a face with 3D pose $\boldsymbol{\theta} = (\boldsymbol{u}, s, R)$, we compute the (modified) LBF $\boldsymbol{x}(\boldsymbol{\theta})$. The score $S(\theta) = S(\boldsymbol{u}, s, R)$ of the 3D pose $\theta = (\boldsymbol{u}, s, R)$ is given by:

$$S(\theta) = \boldsymbol{\beta}^T \boldsymbol{x}(\boldsymbol{\theta}) \tag{2.10}$$

Training the score function $S(\theta)$. The scoring function is a classifier trained to predict the candidates with large overlap between the face bounding box B_θ and the bounding box of an annotated face with largest overlap with B_θ. From all the face candidates of the training set, the candidates with overlap at least 0.7 are used as positives and the ones with overlap at most 0.3 as negatives. Given a training set of poses $\theta_j, j = \overline{1, N}$ and $\boldsymbol{y}_j$ (see Equation 2.6), learning the parameters $\boldsymbol{\beta}$ is obtained by minimizing the following quadratic energy:

$$E(\boldsymbol{\beta}) = \sum_{j=1}^{N} L(y_j S(\theta_j)) + \sum_{j=1}^{M} \rho(\boldsymbol{\beta}_j) = \sum_{j=1}^{N} L(\boldsymbol{y}_j \boldsymbol{\beta}_j^T \boldsymbol{x}(\theta_j)) + s\|\boldsymbol{\beta}\|^2 \tag{2.11}$$

where $L(\cdot)$ is the Lorenz loss (Equation (1.16)).

2.4.4 Non-Maximal Suppression

The non-maximal suppression repeats the following steps until convergence:

1. Select the pose candidate with the largest support above a threshold and finds the bounding box B of its projected points..
2. Remove the candidates that have high overlap with B.

2.5 Evaluation of Intermediate Face Detection Steps

2.5.1 Dataset Description

Before presenting evaluations on our face detection approach, we elaborate on the different datasets used in this section.

AFLW. The Annotated Facial Landmarks in the Wild [60] is a large scale database for facial landmark localization. This dataset was developed by Martin Koestinger, Paul Wohlhart, Peter M. Roth, Horst Bischof at the Institute for Computer Graphics and Vision housed within the Graz University of Technology. It is a multi-view (i.e. not limited to frontal faces), real-world face database, an examples demonstrated in the left image of Figure 2.17, with annotated facial features. The images were downloaded from *Flickr* (via face tags) which were then manually vetted.
The database has a total of $21,123$ images containing $24,386$ faces annotated with 21 points, as seen in right image of Figure 2.17. Of them, $16,207$ images were found to contain one face per image. There were $2,164$ images containing at least 2 annotated faces. Most of the images were color and some were gray-scale. They state that no rescaling or cropping was performed and that some images contain multiple faces. They also provide a gender break-down: 59% of the faces are female and 41% of the faces are male. In total, the dataset contains about $380K$ manually annotated facial landmarks.

AFLWMF. The AFLWMF dataset is subset of the AFLW dataset. By visual inspection, $1,555$ of them were found to have all the faces annotated and were used as the test dataset AFLWMF. These $1,555$ images contain $3,861$ faces.

LFPW. This dataset of $1,432$ images was collected via queries on search engines. A total of 35 face keypoints were annotated (unless the keypoint was occluded). Figure 2.18 provides an example of image collected in [7] with the ear keypoints omitted due to occlusion. The coordinates of each keypoint on a face is represented in a CSV file where the coordinate of each point is given followed by a number (0-3) to indicate whether it is occluded or not.

Helen. The Helen dataset [66] was also collected via Flickr using specific queries. The images were annotated using the Amazon Mechanical Turk and example of an image is presented in Figure 2.19. There is a total of $2,000$ training images and 330 test images.

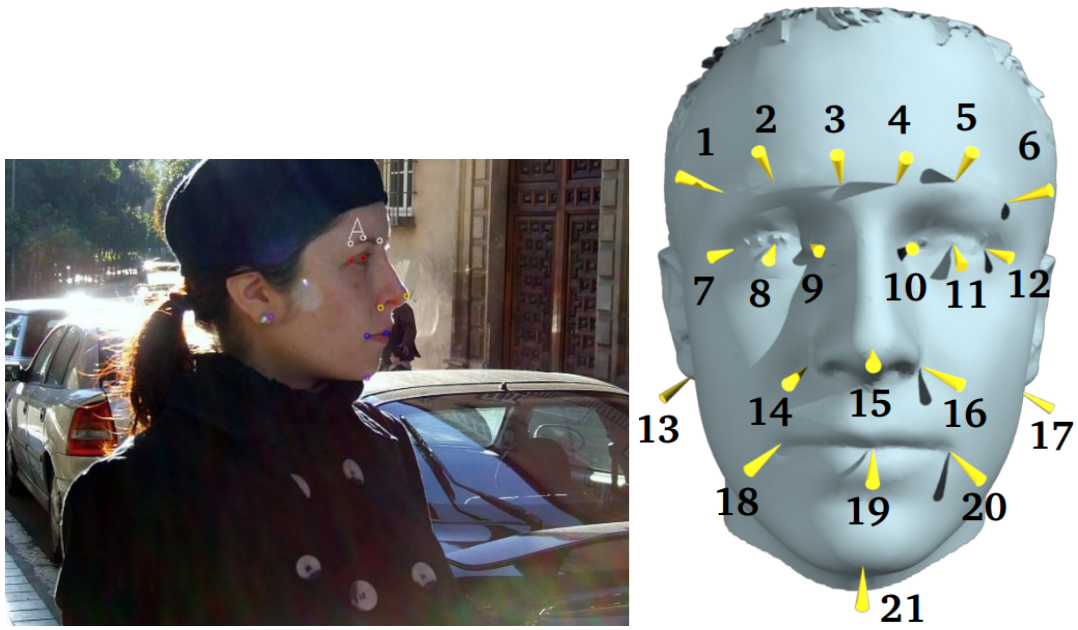

Figure 2.17: Examples of the real-world scenario face images (left) and the 21 face keypoint template of AFLW (right). Both images were taken from [60]

2.5.2 Face Keypoint Evaluation

We present experiments on detecting face keypoints from color images. The face keypoints such as eye centers, nose sides, mouth corners, chin, bottom of ears, are represented as 2D points (x, y). The training examples are points on the Gaussian pyramid, with the positives within one pixel from the keypoint annotation on the images of the pyramid where the inter-eye distance is in the $[20, 40]$ pixel range. The negatives are all points at least 4 pixels from the keypoint annotation. In total the 999 AFLWT training images contain about 1 billion negatives. All the negatives were used for training the classifiers through a negative mining procedure similar to [42], with the difference that about 20,000 hard negatives were added to the training set at each iteration, thus the set of training negatives increased with each mining iteration. All classifiers were trained with 10 iterations of mining hard negatives.

A separate classifier was trained for each keypoint being evaluated. All classifiers except SVM-PL HOG were trained as monolithic classifier with 1500 features or weak learners. The SVM-PL HOG classifier was trained on the 864 HOG features, without feature selection

The following criteria were used for evaluating detection performance. The visible face keypoint is

Figure 2.18: Example of image with most of the face keypoints used in LFPW (except the ear points) taken from [7]

Figure 2.19: Example of image from Helen dataset taken from [66]

considered detected in an image if a detection is found at most 5% of the IED (inter-eye distance, computed by fitting a rigid 3D face model) away in one of the images of the pyramid. A detected point p in one of the images of the pyramid is a false positive if it is at least 10% of the IED away from the face part being evaluated (visible or not) of any face of the image.

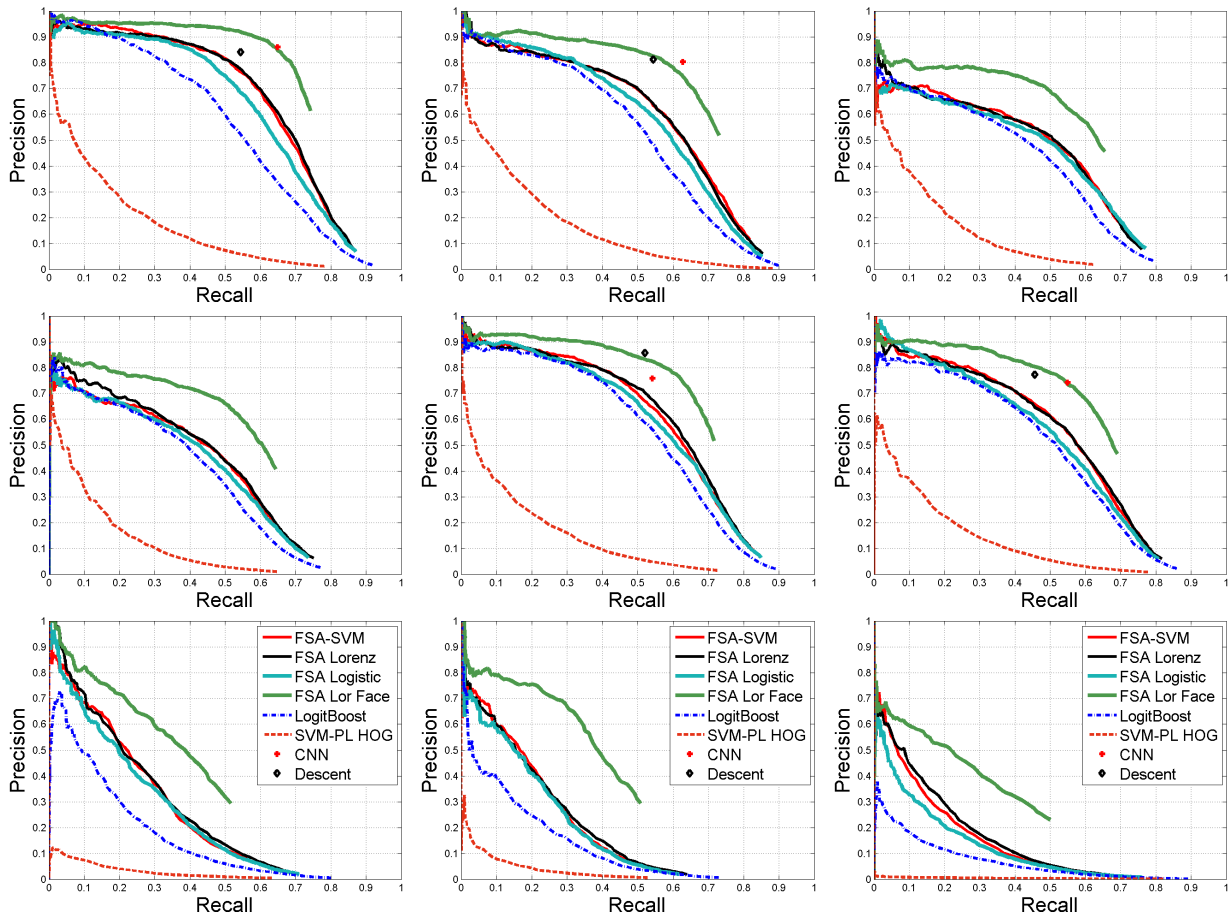

Figure 2.20: Precision-recall curves for face keypoint detection on the test set AFLWMF containing 1555 images and 3861 faces. From top to bottom, left to right: left/right eye center, left/right nose, left/right mouth corner, left/right ear, chin.

We compared the following learning algorithms:

1. FSA-Logistic - The FSA method on the Logistic loss Equation (1.12) with piecewise linear learners, $\mu = 300, N^{iter} = 500$.

2. FSA-SVM - The FSA method on the SVM loss Equation (1.14) with piecewise linear learners, $\mu = 300, N^{iter} = 500$.

3. FSA-Lorenz - The FSA method on the Lorenz loss Equation (1.16) with piecewise linear learners, $\mu = 300, N^{iter} = 500$.

4. LogitBoost using univariate piecewise constant regressors as weak learners. For speed reasons, only 10% of the learners were selected at random and trained at each boosting iteration and the best one was added to the classifier.

5. SVM-PL HOG - The SVM algorithm with piecewise linear response on each variable. The variables were the 864 HOG features.

In Figure 2.20, the precision-recall curves are shown for detecting nine keypoints on the AFLWMF data. One can see that the FSA-SVM and FSA-Lorenz perform similarly and slightly outperform the FSA on the logistic loss. All three FSA versions outperform Logitboost and greatly outperform the piecewise linear SVM on the HOG features. At the same time, the FSA algorithm is about 8 times faster than the LB algorithm, which is 10 times faster than the full LB version that trains all weak learners at each boosting iteration. The regression-based FSA-Lorenz method is at least as good as the sliding window classifiers, while being about 4 times faster.

Also shown are the supervised descent method [113] and the CNN based face point detection method [104] on the eye and mouth, which were the keypoints that were in common with the keypoints we evaluated.

These two methods outperform the classification and regression-based FSA detectors. However, we must point out that the two face alignment methods are top-down methods that rely on the face being detected first by a face detector, which in the case of the CNN method was trained with about 100k faces In contrast, our point detectors are bottom-up detectors that were trained with 999 faces to directly detect the keypoints without the intermediary step of finding the face. If we involve our own 3D-model based face detector [5] that uses all nine FSA-Lorenz keypoint detectors to detect the face and its 3D pose, we obtain the curve denoted as FSA-Lor Face. These results were obtained using a top-down pruning step that keeps only the keypoint detections that are within 0.5 IED (Inter-Eye Distance) from the predicted locations from the 3D pose. We see that using the top-down information we obtain results comparable to the CNN method [104] and slightly better than the supervised descent method [113].

2.5.3 Evaluating 3D Pose Candidate Generator

It would be useful to know how good is the 3D Pose Candidate Generator (3DCG) at predicting the correct 3D poses by regression. A 3D pose candidate $\theta = (\boldsymbol{u}, s, R)$ can be evaluated using the relative error:

$$err(\theta) = \frac{1}{D|I|} \sum_{i \in I} \|T_\theta(F_i) - G_i\|$$

which is the average point-to-point distance between the 2D points $T_\theta(F_i)$ predicted by the pose θ and the corresponding ground truth (GT) annotation points G_i, divided by the inter-eye distance

D of the GT face. Then for a set of 3D pose candidates in an image one could compute the smallest relative error towards each of the faces present in the image, obtaining the relative error for predicting each face.

The errors percentiles for these relative errors are shown in Figure 2.21. On the left are shown the percentiles for fitting the 3D model to the GT location of the nine keypoints in different datasets, showing that the 3D model fits the GT well. To evaluate only the 3DCG while removing the effect of missed keypoint detections, the 3DCG was run from all points at distance at most 2 pixels from the true keypoint locations and the error percentiles of the closest candidate to the GT are shown in Figure 2.21, middle. This was only possible on the LFPW and AFLW datasets, which had all 9 keypoints annotated. In Figure 2.21, right are evaluated the 3DCG together with the keypoint detection by FSA-SVM, showing the relative error percentiles for predicting the GT faces. It is clear that the LFPW dataset is much easier than the AFLW dataset, because it contains more frontal faces. One could also see that the 3D pose candidate generator does a good job in predicting the 3D pose on about 90% of the AFLWMF faces and on all the LFPW faces.

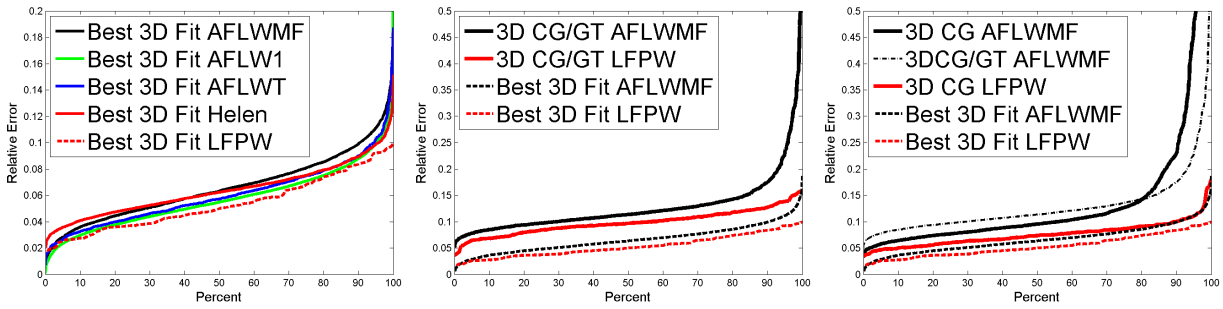

Figure 2.21: Candidate Generator Evaluation. Left: errors of the best fit of the 3D model to the ground truth (GT). Middle: errors of the best 3D pose candidate predicted from the true keypoint locations. Right: errors of the best 3D pose candidate.

2.5.4 Illustration of Keypoints and Difference Features

In Figure 2.22, we provide an illustration of the 9 keypoints (of 21) we use to perform our bottom-up face detection. We accentuate each point by centering a grid of radius 4 over which we compute the difference features of Section 2.2.3.

Figure 2.22: Illustration of keypoints and difference features (with radius of 4). Top Image: template face with 21 keypoints. From top left to bottom right: bottom right ear, middle right eye, middle left eye, bottom left ear, right nose edge, left nose edge, right mouth corner, left mouth corner, chin

CHAPTER 3

PARAMETER SENSITIVE CLASSIFIERS WITH FEATURE SELECTION WITH ANNEALING

3.1 Motivation

The idea for Parameter Sensitive Feature Selection with Annealing (PFSA) was motivated by the data we analyze. For example, in the case of face detection, one of the difficulties lies in the various possible poses. As can be seen in Figure 3.1, there are many in-plane and out-of-plane face rotations that can occlude vital face features. These rotations are commonly known as roll, pitch, and yaw angles.

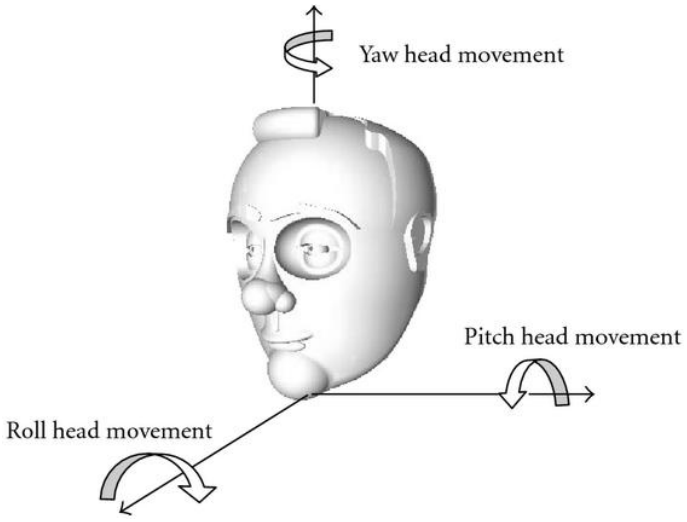

Figure 3.1: The image demonstrates the different angle rotations for a 3D object (image taken from **http://www.hindawi.com/journals/mse/2009/245606/fig7/**).

In the URL Reputation dataset we analyze, and describe in Section 4.2, malicious websites only stay on-line for a limited time due to various reasons (e.g. government agencies shutting them down). That is, time is a natural parameter that can be used to better understand the characteristics of malicious websites. We hypothesized that if we could capture the inherent variability in our data, such as the Yaw angle demonstrated in Figure 3.2, we would be able to improve classification.

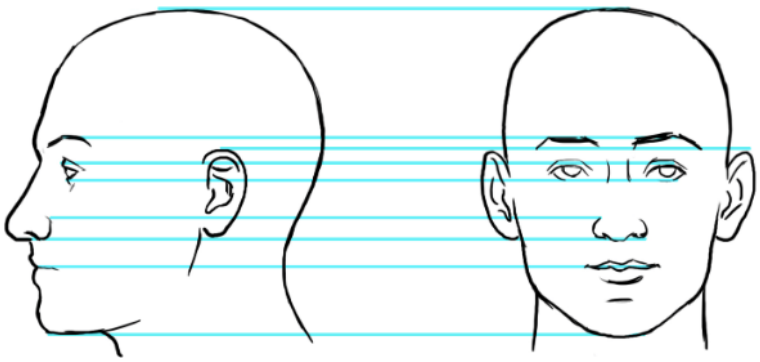

Figure 3.2: The image demonstrates the variability in face detection that we wish to account for in our model (image taken from **http://imgarcade.com/1/frontal-face-drawing/**) - the motivation for PFSA.

We saw three advantages to parameterizing a classifier trained using FSA. First, by making this classifier dependent on a parameter that describes variability in the data (such as time or the Yaw angle), we would be able to perform classification with a single classifier as opposed to several classifiers. As will be discussed in the following Related Works section, prior work has involved training several classifiers for different parameter values. Instead, we use bins to describe the variability. For example, in Figure 3.3, the Yaw angle might be a natural parameter to discretize into bins. The optimal number of bins can be fined tuned empirically.

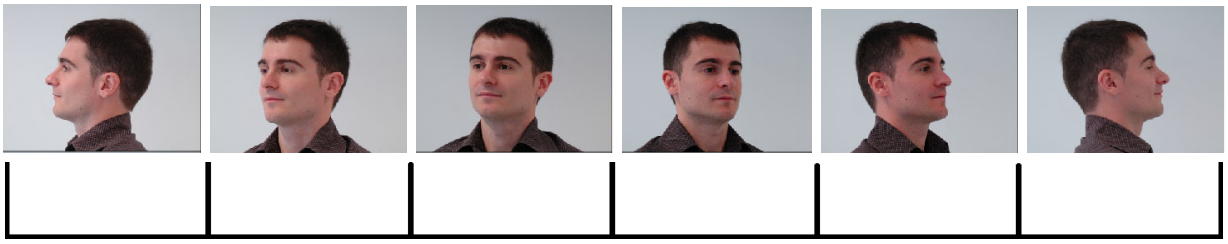

Figure 3.3: Example of how bins can be used to capture variability in face pose (face-images taken from **http://i.cs.hku.hk/cisc/projects/websiteITS08211/Background.html**)

We begin by summarizing prior work done in this area.

3.2 Related Works

Yuan et al. propose classifiers that are associated with continuous parameters [118] in order to perform object detection. These parameters are used to describe large within class variability

inherent to problems such as object detection. Since these classifiers reside in a parameterized function space, these parameters can be learned from the training data in an unsupervised fashion. The advantage of this approach is that it resolves the necessity of partitioning the continuous parameter space. Their method allows them to exploit the correlation that exists between classifiers that share features since their parameterized classifiers are learned jointly. Their work classifies faces, hand shapes and pedestrians in images and the classification function is solved using Support Vector Machine (SVM) or Adaboost. Of particular interest to us is to be able to extend this idea to accurately detecting profile faces (i.e. a face with only one side showing). As pointed out in [118], detection rates are lower for profile faces compared to frontal faces [58, 81].

The work in [119] builds upon [118] and uses the product of two kernels within a SVM training. One kernel is used for pose estimation as well as feature sharing. The other kernel handles the classification of foreground and background. The support vectors obtained from the SVM training are reweighed so that only those support vectors having a common parameter value (θ) will have a large weights (hence, feature sharing). This allows [119] to "generate" individual classifiers that are associated with a particular weighting of support vectors and thus, a specific parameter value. He et al. also learn detection and pose estimation via a product of two kernels [51]. One of the differences with [119] is that He et al. learn continuous pose estimation. The joint kernel approach [51] produces a non-convex optimization objective that they solve in a cascaded manner. More specifically, one kernel they refer to as a "structural kernel" which is used for detection and the other kernel the refer to as a "pose kernel" which is used for continuous parameterization. In order to reduce the complexity of the search space, their two-step cascaded algorithm, prune + refine, makes several proposals to the algorithm. The end game is to return the best bounding box and object view angle. We provide the steps taken by [51] in order to reach this final solution:

1. **Prune**: They obtain a smaller search space that consists of candidate pairs of bounding boxes and ranges for possible poses.
    - The possible view angles are uniformly divided into M intervals. These intervals are represented by a single *theta* value such as the geometric mean.
    - Once a specific θ value is given, their model reduces to a single linear classifier.
    - Gradient based optimization is used to generate bounding box and view angles pairs.
2. **Refine**: They optimize over the possible candidates and return a final pair.
    - During this stage, the reduced search space permits a gradient ascent optimization of the objective function.

- This part of the algorithm requires careful tuning so that the possible range for the view angles is small enough to allow few local optima but enough representative bounding boxes to permit a thorough exploration of the search space.

We note that similar to our approach, He et al. use non-maximum suppression in order to insure that a diverse and predetermined number of candidates are generated. They modify the loss function in their structural SVM in order to account for both detection and pose. They also train their model in an on-line fashion by providing small batches of training examples at each model update iteration.

The work by Gu and Ren [47] builds upon the discriminate part based model of [42] for object detection by adding pose estimation. In fact, in their paper, they present two different viewpoint models: discrete and continuous. We summarize their contributions for each category below.

1. **Discrete Viewpoint**: The final output of this framework returns a confidence score for the presence of the object and a viewpoint label.
   (a) A pre-determined number ($\mathcal{V}$) of canonical viewpoints of the object is saved in order compare against when objects are detected. Each viewpoint will have a label associated with it.
   (b) Several canonical viewpoints, for a specific object, are represented using a mixture model of HOG-based templates.
      - NOTE: they point out that they use HOG based features to represent the spatial layout of the object because of its robustness to intra-class and intra-viewpoint variation.
   (c) A sliding window classifier is used to detect the object where non-maximal suppression is used to enforce a diverse candidate set (i.e. remove redundant image patches).
   (d) Their large margin optimization extends what is done in [42]. One of their contributions is learning the viewpoint labels in the supervised, semi-supervised, and unsupervised case.

2. **Continuous Viewpoint**: Here the output, in addition to the BB, is a viewpoint angle $\boldsymbol{\theta} \in \mathbb{R}^3$.
   - After modifying the discrete mixture model, they partition the continuous viewpoint space into small chunks that are each associated with a canonical viewpoint.
   - Assuming small chunks, a viewpoint is approximated by a linear deformation of the canonical template w.r.t. the difference of viewpoint angles and canonical angles.
   - A window classifier is also employed to detect the object

There has also been work in multi-view object detection with a 3D Geometric model [83, 70]. A three step algorithm for 3D multi-view object detection is introduced in [83]. They address the computationally intensive demands of searching over an image for various image scales by first generating a bounding box for the object. Then, within this image patch, they construct features expressed as histogram pyramids [65] that are robust against small errors in the bounding box size and object view variance. A Naive Bayes classifier is trained using these spatial pyramid histograms to learn the object pose. Finally, a SVM is used to determine the presence of the object (which in this work is cars).

Liebelt and Schmidt learn a 3D model and object classifier separately and then combine them [70]. They rely on a few synthetic 3D models to learn a 3D representation of the geometry of the object. This allows them to estimate a 3D object pose which they use to evaluate the 2D part detection. Their approach allows them to avoid tedious manual annotations of the individual object parts, only a 2D bounding box and the viewpoint of the object is needed. In summary, their detection process is as follows:

1. Pre-Detection: Initial 2D detection of regions of interest using sliding windows
   - Sliding windows are used and then mean-shift mode estimation
   - They point out that sliding windows cannot capture all possible window layouts on all possible scales.
2. Object-Parts Detection: Sub-Grids are made on the regions of interest to focus on the object and remove background variability.
3. 3D Pose Estimation: Using an EM-like procedure (Genetic algorithm), the optimize the likelihood of the detected 2D parts over spherical camera parameters.

A probabilistic framework for 3D object classification is proposed in [103] where image features and geometric constraints are combined to ensure detection across different viewpoints. Sun et al. model can generate a distribution of the parts that represent an object in a Bayesian manner. We summarize how Sun et al. train their generative model:

1. They initialize their model using a single object instance over all viewpoints. In order to match features across different views, they employ the multiRANSAC algorithm.
   (a) They sample a viewpoint (from K) using a Uniform Distribution.
   (b) After generating image features, for image feature
      i. generate the expected (object) part proportion for the current viewpoint. Use this to obtain the labels for the parts. (Note: these labels have a Multinomial Distribution).

ii. generate the feature appearance and location. Sun et al. point out that the object parts are represented by appearance elements (codewords).

(c) For each image, they introduce an affine transformation variable. Sun et al. state that a contribution of their work is the ability to automatically compute affine transformation so that object detection can be performed under any viewpoint. This of course is done within their Bayesian framework.

(d) They reduce the complexity of the parameter space by enforcing the following geometric constraints for parts with nearby viewpoints: they are assumed to have similar appearances/codewords AND their 2D configuration is constrained using epipolar geometry.

2. Updates are done incrementally. For a new training image, its affine transformation (relative to the base instance) is estimated.
3. Sun et al. predict where new parts might be based on existing parts.

Su et al. build upon [103] by developing a model that requires less supervision and superior viewpoint depiction [102]. Su et al. argue that a dense representation of poses can lead to robust object detection and avoid the tedious manual labeling of 3D poses by initializing their model using a video clip from a cell phone. They learn the model as follows:

1. Starting with a viewing sphere centered at the object, they parameterize the sphere using triangles. This sphere is "built" in practice by using a cell phone to capture a 360◦ video of the object, lowering and raising the cell phone as filmer walks around the object
   - They enforce feature-level correspondence across frames using the Lucas-Kanade tracker. They say that on average, about 100 viewpoints are sampled from each video clip.
   - They triangulate the sphere by requiring the neighboring viewpoints by grouped into the same triangles. **Note**: They claim that this parameterization permits synthesis of new viewpoints within each triangle.
2. They apply modified version of the J-Linkage clustering algorithm to generate object parts. Su et al. provide a very specific definition of what they mean by an object part.
3. They learn a probabilistic model for a 3D object class similar to [103] with the two differences that they introduce a morphing variable in order to recognize and generate new poses and their triangular representation of the viewpoint sphere permits more robust geometric constraints.

Ali et al. jointly learn pose-indexed image features and pose estimators in a boosted manner: each boosting iteration chooses the best pose estimator and pose-indexed pair that addresses errors from the previous iteration [1]. This allows the classifier to deform the features according to computed pose parameters. While they use AdaBoost, they point out that other classification methods could

be used, such as SVM or decision trees. This method is similar to our approach in that a single classifier is used and different learning algorithms can be used.

3.3 Parameter Sensitive Classifiers with FSA

The formal representation of our PFSA framework is as follows. Let $(\boldsymbol{x}_i, y_i, \theta_i)$, $i = \overline{1, N}$ be training examples with $\boldsymbol{x}_i \in \mathbb{R}^M$, a loss function $L(\boldsymbol{\beta})$ based on these examples. Then, the modified constrained optimization problem we solve in PFSA is

$$\boldsymbol{\beta} = \underset{|\{j:\beta_j \neq 0\}| \leq k}{\operatorname{argmin}} \; L(\boldsymbol{\beta}) \tag{3.1}$$

where the number k of relevant features is set a-priori, and the loss function $L(\boldsymbol{\beta})$ is differentiable with respect to $\boldsymbol{\beta}$. The $\boldsymbol{\beta}$ learned in this scenario is a matrix that can be visualized as in Figure 3.4. Each row represents a bin that is used to discretize the parameter θ. All learned values for the bin ℓ is represented by the vector $\boldsymbol{\beta}_\ell$. Note that there are M columns representing the number of features.

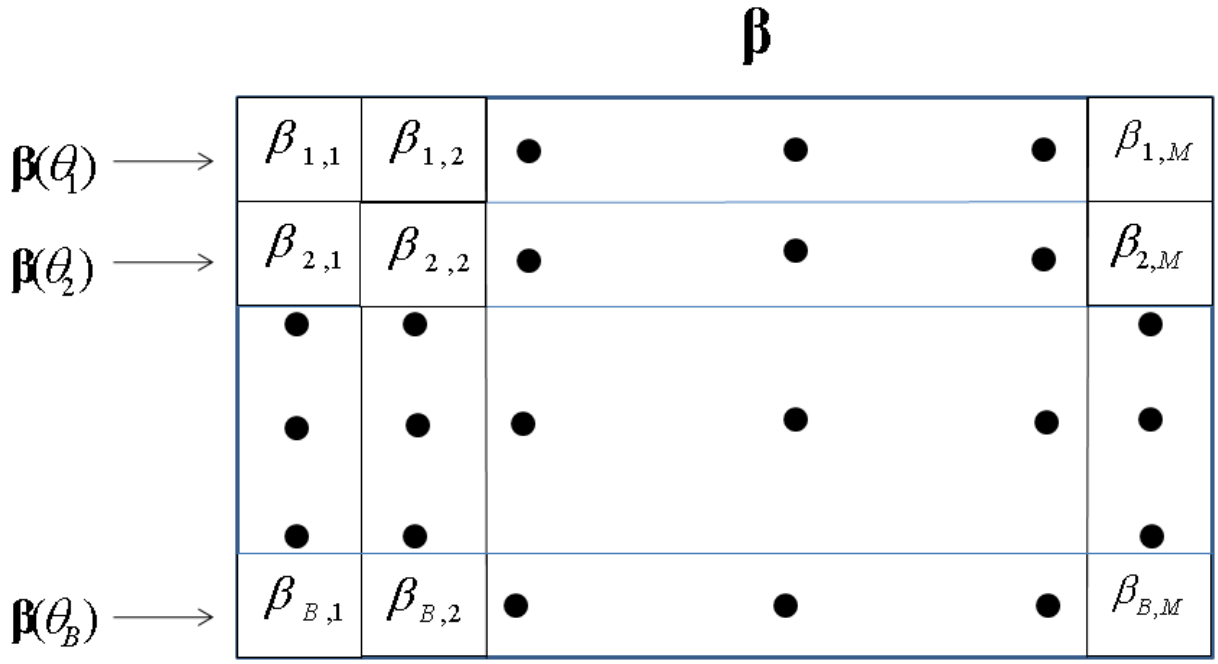

Figure 3.4: Example of the $\boldsymbol{\beta}$ that is learned in PFSA. Note how the bins are represented via rows and the number of $\boldsymbol{\beta}$ vectors learned are represented via columns (a total of M)

The classifier can be expressed as eq. (3.2)

$$f_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = \sum_{j=1}^{M} \boldsymbol{\beta}_j(\theta_i) \cdot x_j \tag{3.2}$$

There are several ways to parameterize the classifier $f_{\boldsymbol{\beta}}$ such as discretizing $\boldsymbol{\theta}$ into bins that cover the range of the parameter space or using splines. In order to enforce smooth $\boldsymbol{\beta}_j$, we enforce a second order prior across all bins (see Equation (3.3)). An example of the smooth $\boldsymbol{\beta}_j$ that we generated during simulations (for 30 bins) can be seen in Figure 3.5, we will discuss these simulations in more detail in the section on Experiments.

$$\rho(\boldsymbol{\beta}_j) = c \sum_{j=2}^{M} (\beta_{j,b+1} + \beta_{j,b-1} - 2\beta_{j,b})^2 + s\|\boldsymbol{\beta}\|^2 \tag{3.3}$$

With the previous notations in mind, we can state the loss function being optimized in Equation (3.4).

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{N} L(y_i \boldsymbol{\beta}^T(\theta_i)\boldsymbol{x}_i) + \sum_{j=1}^{M} \rho(\boldsymbol{\beta}_j) \tag{3.4}$$

In Algorithm 1, during the update step, depending on which bin the current training example falls into, we update that bin. The criterion we use to determine which $\boldsymbol{\beta}_j$ to keep depends on the sum of squares across the bin values: $\|\boldsymbol{\beta}_j\|_2^2$

3.4 PFSA for Face Detection

3.4.1 Overview

As stated in the Motivation section, we wish to capture the variability in face poses in order to improve our accuracy in face detection. Recall that the process of detecting the face and its pose begins with detecting the face keypoints, then predict the 3D pose from those keypoints (i.e. generate 3D candidates), score those candidates and remove those that have significant overlap, and finally, use a global model to make a (bounding box) prediction. The entire process can be visualized in Figure 2.12.

The parameter sensitive classifier comes into use after we have generated 3D face pose candidates (i.e. the second image in Figure 2.12). Specifically, our goal was to have a more robust method for scoring the pose candidates. From experiments, we realized that the yaw angle had the largest

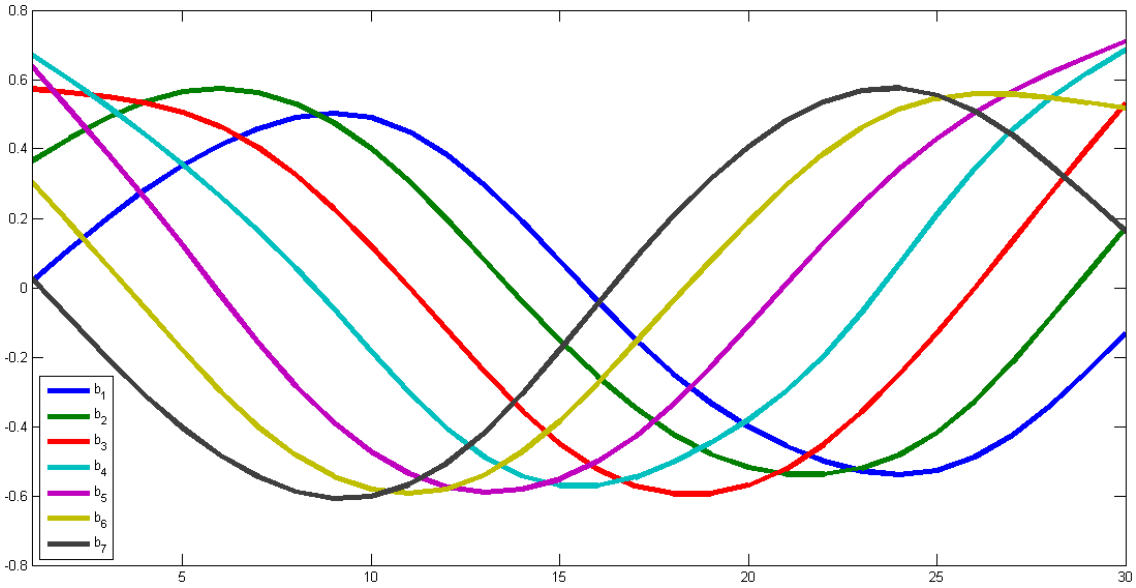

Figure 3.5: Example of smooth $\boldsymbol{\beta}$ obtained with our second order prior

face pose variability. We pursued the hypothesis that a yaw-angle dependent model would improve how we scored candidates.

The yaw-angle dependent image-based score function can be represented in Equation (3.5). For each feature $\boldsymbol{x}$ extracted from image I at yaw angle θ^y, we assign it the weight $\boldsymbol{\beta}(\theta^y)$,

$$S(dx, dy, ds, \theta^r, \theta^p, \theta^y) = \boldsymbol{\beta}(\theta^y)^T \boldsymbol{x} \tag{3.5}$$

where dx, dy is the x, y coordinate of the keypoint in the image, ds is the image scale, θ^r is the roll angle and θ^p is the pitch angle.

3.4.2 Training Details for the Scoring Function of Face Candidates

We expand on the scoring of face candidates since this is where we leverage our parameterized classifier. The Face candidate $F = (\theta)$ contains the 3D pose $\theta = (\boldsymbol{u}, s, R)$ and predicted locations $P = (\boldsymbol{p}_1, ..., \boldsymbol{p}_L)$ of the L keypoints. The score is obtained by a parameter-sensitive classifier that depends on the yaw angle φ^y.

Score function. The score $S(\theta)$ of the 3D face candidate with face keypoints P and pose θ is based on the LBF feature vector $\boldsymbol{x}(\theta)$:

$$S(\theta) = S(\boldsymbol{x}) = \boldsymbol{\beta}(\varphi^y)^T \boldsymbol{x}(\theta). \tag{3.6}$$

The coefficients $\boldsymbol{\beta}(\varphi^y)$ depend parametrically on the yaw angle φ^y of the rotation A. The yaw angle ranges between $-\pi$ and π, being 0 for frontal faces and $\pm\pi/2$ for profile faces. For this application, it is discretized into $B = 16$ bins, so there are parameter vectors $\boldsymbol{\beta}_k, k = \overline{1, B}$, one for each yaw angle bin.

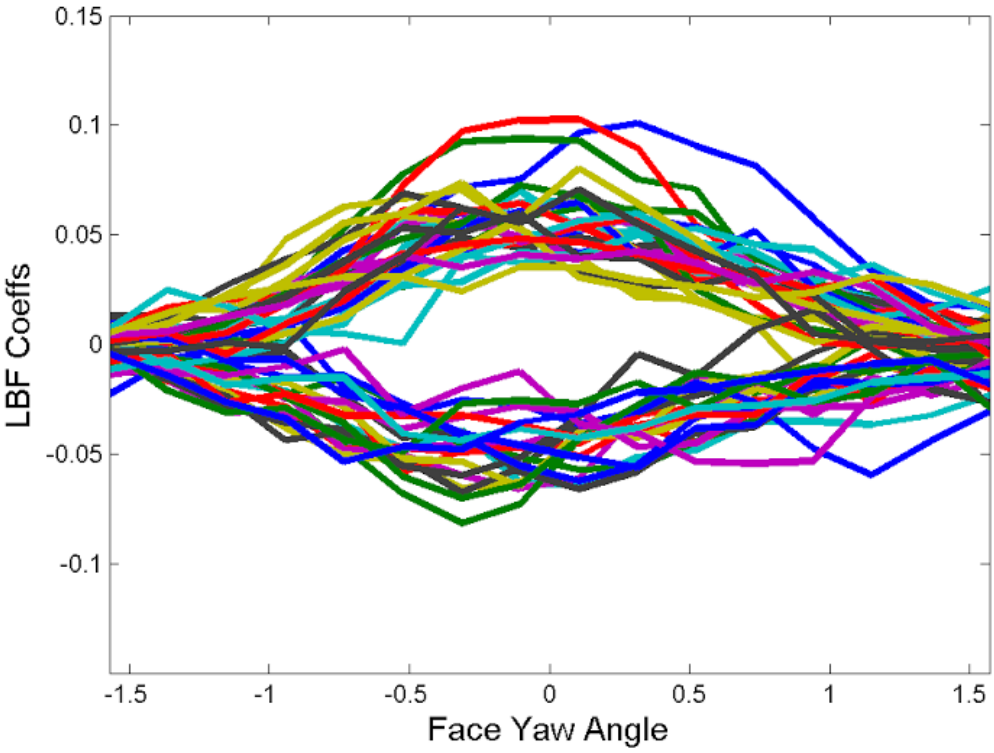

Figure 3.6: Top 50 LBF coefficients by total variation, out of 28,800.

Training the score function $S(\theta)$. The scoring function is a classifier trained to predict the candidates with large overlap between the face bounding box B_θ and the bounding box of an annotated face with largest overlap with B_θ. From all the face candidates of the training set, the candidates with overlap at least 0.7 are used as positives and the ones with overlap at most 0.3 as negatives. We obtain this way a training set of face candidates $F_j = (P_j, \theta_j), j = \overline{1, N}$ with yaw angle bins $b_j \in \{1, ..., B\}$, LBF feature vectors $\boldsymbol{x}_j$, and labels $y_j \in \{-1, 1\}$. Learning the parameters $\boldsymbol{\beta}$ of the face score is obtained by minimizing the classification loss:

$$E(\boldsymbol{\beta}) = \sum_{j=1}^{N} L(y_j S(F_j)) + \sum_{k=1}^{B} \rho(\boldsymbol{\beta}_k) = \sum_{j=1}^{N} L(y_j \boldsymbol{\beta}_{b_j}^T \boldsymbol{x}_j) + \sum_{k=1}^{B} \rho(\boldsymbol{\beta}_k) \tag{3.7}$$

where $L(x)$ is the Lorenz loss [4]

$$L(x) = \begin{cases} \ln(1 + (x-1)^2) & \text{if } x < 1 \\ 0 & \text{else} \end{cases}$$

and the prior $\rho(\boldsymbol{\beta})$ encourages smooth changes of the coefficients between adjacent bins

$$\rho(\boldsymbol{\beta}) = s\|\boldsymbol{\beta}\|^2 + c\sum_{i=2}^{B-1}(\beta_{i+1} + \beta_{i-1} - 2\beta_i)^2. \tag{3.8}$$

The loss function $E(\boldsymbol{\beta})$ is differentiable but non convex. It is minimized by 50 epochs of stochastic gradient descent with momentum $\mu = 0.99$ and learning rate $\eta = 10$.

CHAPTER 4

EXPERIMENTS

4.1 Simulations

In this chapter, we build upon the simulation structure from Chapter 1.

4.1.1 Background

Regular FSA has strong classification performance on Normally simulated data compared to other classification algorithms (given sufficient training examples), with or without noisy labels. We proceeded to simulate a classification problem that could potentially benefit from our parameterized classifier. As in Chapter 1.8.2, the data is simulated using a Normal distribution where only 10 or 30 of the 1000 features are used to compute the true signal.

However, these simulations add a weight factor to Equation (1.26) that wasn't present before. Specifically, the responses are computed as in Equation (4.1).

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{k^*} x_{10i} \cdot \beta_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

We use two different kinds of weights $\boldsymbol{\beta}$: one comes from a sine function, Equation (4.2), and the other comes from a sine function that has been modified to be zero whenever the original sine function takes on negative values, Equation (4.3). We note that both weight functions have a constant phase shift.

$$\beta_i = \sum_{j=1}^{B} \sin(2j\pi/B) + c \tag{4.2}$$

$$\beta_i = \begin{cases} \sum_{j=1}^{B} \sin(2j\pi/B) + c & \text{if } \sum_{j=1}^{B} \sin(2j\pi/B) + c > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

The phase shift c can be set by the user to be as much as desired. We set our phase shift to be a multiple of $\pi/6$. The max summation index B is the number of bins. The tables in this chapter that summarize the simulation results demonstrate that we experimented with three different bin

values: 3, 10 and 50. We illustrate these weights in Figures 4.1 and 4.2. We only present graphs for bins of size 10 and 50 since it is difficult to visualize the graphs at 3 bins (as we can start to see in the plot for 10 bins, fewer bins lead to a more distorted graph).

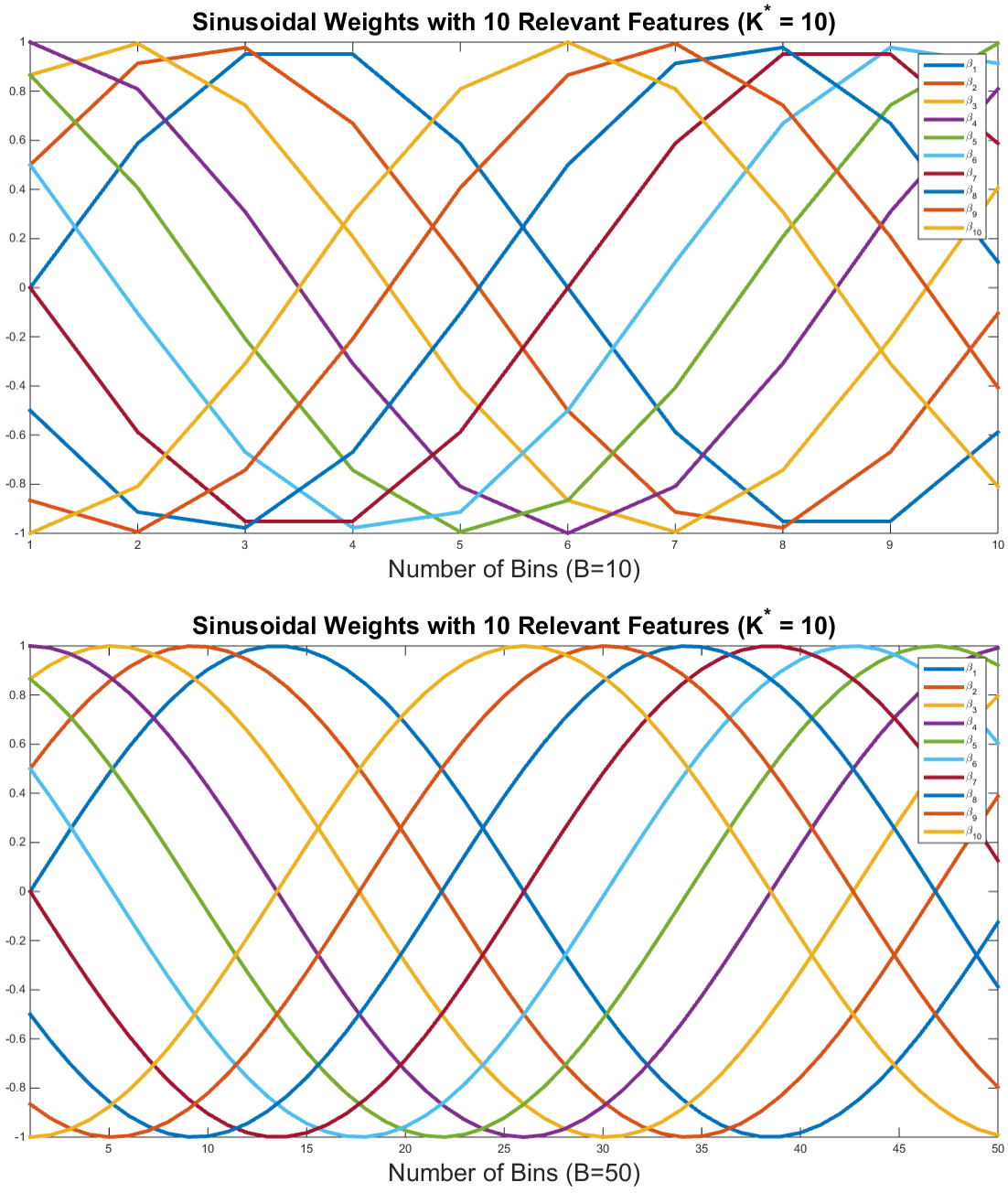

Figure 4.1: Illustration of Sinusoidal Weights

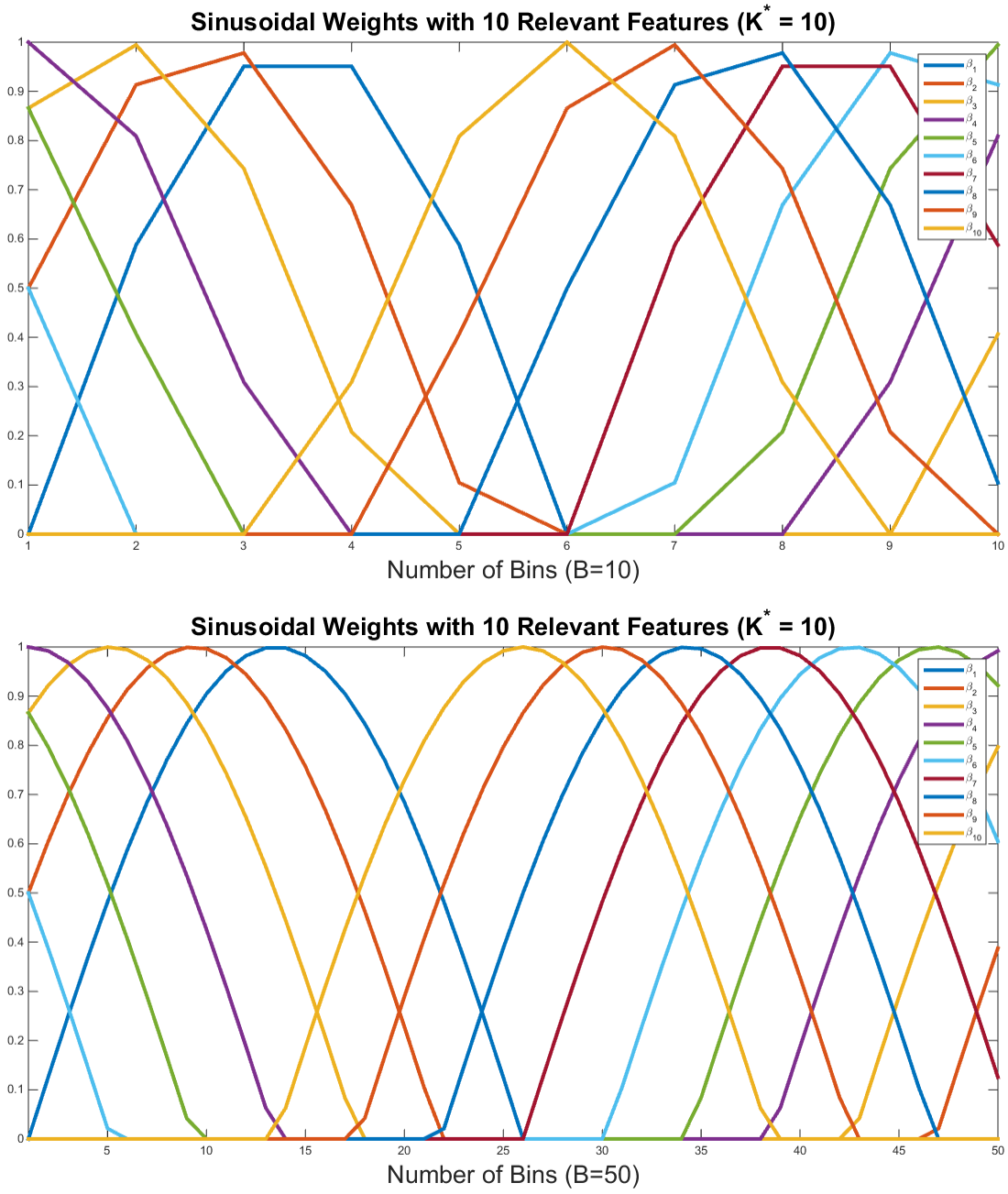

Figure 4.2: Illustration of Zero Sub-Domain Weights

An important goal of these experiments is to be able to recover the original weights that give rise to the response values as closely as possible. The estimates of the weights are represented as b_i. Thus, an accurate recovery of these weights will permit us to recover the true signal. Given the

variability of the sinusoidal weights or the weights the vary between sine values and zero values, we hypothesized that our parametric classifier would be able to model this behavior better than other existing methods. The results in the tables that follow give empirical evidence to support our belief.

4.1.2 Simulation Results

For various combinations of N and M, we present the results for a FSA classifier as well as our parameter sensitive classifiers:

1. Parameter sensitive classifier with FSA and logistic loss (Equation (1.13)) - PFSA.
2. Parameter sensitive classifier with FSA and huberised loss (Equation (1.14)) - PFSV.
3. Parameter sensitive classifier with FSA and lorenz loss (Equation (1.16)) - PFSL.

In Figures 4.4 - 4.9, we present examples of the b_i our parameterized classifier recovered for the case $K^* = 10$, 10 bins, and noiseless weights. As expected, as the number of observations increases, our recovered weights behave more closely to the true weights. We also point out that of the three sensitive parameter classifiers (PFSA, PFSV, PFSL), the one based on the Lorenz loss does a better job at staying within the upper and lower limits of the ground truth weights.

An interesting finding is the difficulty that most methods had with the weights that were strictly sinusoidal. As the tables demonstrate, all of the non-parameterized algorithms (explained in Chapter 1.8.2) failed to obtain a single all-variable detection, regardless of the sample size. However, for the zero sub-domain experiments, once there were sufficient observations, the non-parameterized approaches were able to increase their all-variables detection rate. We hypothesize that it has to do with the segments where the zero sub-domain weights have a constant behavior which makes it easier for the non-parameterized methods to approximate.

Conversely, our parameter sensitive classifiers did not perform as well in the case of Zero Sub-Domain weights. We believe that this might be do to the fact that an extended trough, the 'bottom' part of the waves in Figure 4.3 where our weights take on zero values, is needed to account for the zero domain. This constant behavior might diminish our parameter sensitive classifier's ability to capitalize on variability. That is, our classifier ends up acting like a parameter insensitive classifier.

It is worth noting that the choice of weights was motivated from a dataset we applied our method to: URL Reputation [71] (described in Section 4.2). One of the challenges posed by this dataset is

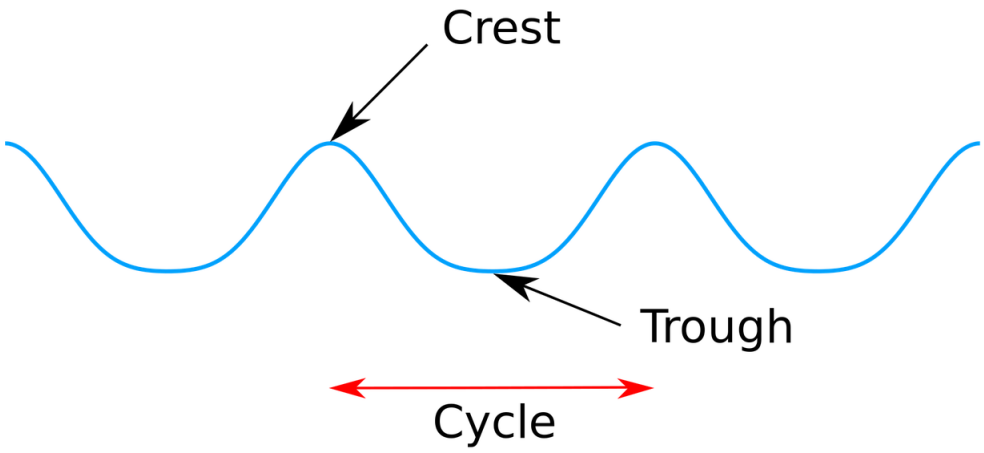

Figure 4.3: Visual description of waves from Wikipedia

that features describing URLs were time dependent and could vary between being zero and non-zero. For example, a website being used for criminal activity could be shut down after some time; thus, there wouldn't be more data from this site. Our goal was to simulate this variability and evaluate the performance of our parameter sensitive classifier.

4.2 URL Reputation Dataset

The use of the dataset generated by Justin Ma et al. in [72]. Essentially, the data set represents URLs that are either benign or malicious. The goal is to classify the URLs using the features extracted for each website (which we will explain shortly). Ma et al. collected websites over 121 days with approximately $20,000$ websites per day.

4.2.1 Description of URL Reputation Dataset

For reasons of protecting user sensitive information, prevent illegal transactions, or deter less than reputable activities, it is vital that methods continually improve to detect websites that fall into either of the previously mentioned categories. This problem is accentuated by the fact the these websites are ever evolving to avoid being detected. As pointed out in [72], these malicious websites can be engaging in:

- Marketing counterfeit goods
- Financial fraud

Table 4.1: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 3 bins, averaged over 100 runs. Sinusoidal weights were used as illustrated in Figure 4.1

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	62	82	86	0	0	0	0	0	0	0	0.8	96.1	98.2	98.6	0	1.9	-	1.7	1.7	1.9	1.7
3000	1000	10	0	100	100	100	0	0	0	0	0	0	0	2.2	100	100	100	0	2.3	-	3.5	3.4	2.3	2.9
10000	1000	10	0	100	100	100	0	0	0	0	0	0	0	2.9	100	100	100	0	9.3	-	8.4	8.8	4.5	6.0
30000	1000	10	0	100	100	100	0	0	0	0	0	0	0	9.2	100	100	100	0	21.5	-	20.7	22.6	14.0	7.9
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.7	72.3	75.8	78.0	0	2.2	-	2.6	2.6	2.8	2.7
3000	1000	30	0	40	46	60	0	0	0	0	0	0	0	2.9	97.0	97.5	98.3	0	2.7	-	3.3	3.3	3.2	2.7
10000	1000	30	0	100	100	100	0	0	0	0	0	0	0	2.3	100	100	100	0	2.5	-	3.1	3.3	3.3	3.0
30000	1000	30	0	100	100	100	0	0	0	0	0	0	0	1.8	100	100	100	0	1.8	-	3.5	3.5	3.3	3.4
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.501	.993	.997	.997	.500	.500	.502	.500	.500	.502	.503	0.09	0.09	0.08	0.08	329	206	3.9	1142	1262	0.44	0.05
3000	1000	10	.502	.999	.999	.999	.500	.505	.506	.507	.507	.505	.505	0.23	0.21	0.18	0.21	1853	619	12.0	1538	1343	1.4	0.15
10000	1000	10	.506	1.00	1.00	1.00	.500	.506	.508	.514	.514	.509	.509	0.91	0.89	0.79	0.86	11040	2301	47.2	2358	2649	4.6	0.47
30000	1000	10	.511	1.00	1.00	1.00	.502	.509	.513	.521	.521	.513	.514	3.1	3.1	2.9	3.1	2.3e4	6379	127	5954	6486	14.0	1.4
1000	1000	30	.498	.966	.971	.975	.500	.499	.500	.500	.500	.498	.499	0.12	0.11	0.10	0.11	436	230	2.0	1138	1097	1.3	0.14
3000	1000	30	.501	.996	.997	.997	.499	.503	.506	.501	.501	.500	.501	0.32	0.30	0.28	0.29	4084	712	7.4	1445	1478	4.1	0.42
10000	1000	30	.501	1.00	.999	.999	.500	.501	.509	.500	.500	.500	.501	1.3	1.2	1.1	1.2	6807	2472	29.1	2727	2807	13.8	1.4
30000	1000	30	.500	1.00	1.00	1.00	.500	.499	.511	.501	.501	.500	.500	4.0	3.9	3.7	3.9	2.6e4	8308	66.3	6060	6355	41.7	4.2
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	16	38	48	0	0	0	0	0	0	0	0.7	87.3	92.0	93.9	0	1.4	-	1.4	1.4	1.6	1.1
3000	1000	10	0	95	97	99	0	0	0	0	0	0	0	1.5	99.5	99.6	99.9	0	2.6	-	3.2	3.3	2.5	2.1
10000	1000	10	0	100	100	100	0	0	0	0	0	0	0	2.2	100	100	100	0	8.4	-	7.4	8.0	5.0	5.7
30000	1000	10	0	100	100	100	0	0	0	0	0	0	0	7.7	100	100	100	0	16.3	-	18.7	19.6	12.5	7.1
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.4	54.1	58.9	61.2	0	3.0	-	3.4	3.6	2.9	3.3
3000	1000	30	0	3	9	25	0	0	0	0	0	0	0	2.8	89.5	93.2	95.7	0	2.7	-	3.1	3.1	2.9	3.2
10000	1000	30	0	94	95	99	0	0	0	0	0	0	0	2.7	99.8	99.8	100	0	2.5	-	3.3	3.3	3.2	3.3
30000	1000	30	0	100	100	100	0	0	0	0	0	0	0	1.9	100	100	100	0	2.0	-	3.4	3.3	3.5	3.2
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.501	.932	.938	.941	.500	.503	.503	.502	.502	.504	.503	0.08	0.08	0.07	0.08	250	198	3.8	1493	990	0.49	0.05
3000	1000	10	.501	.948	.948	.949	.500	.504	.502	.504	.504	.503	.504	0.24	0.23	0.21	0.24	2544	679	12.5	1160	1434	1.4	0.15
10000	1000	10	.504	.949	.949	.949	.500	.506	.506	.511	.511	.507	.507	0.94	0.91	0.85	0.93	6203	2460	45.1	1777	1758	4.7	0.48
30000	1000	10	.509	.950	.950	.950	.501	.509	.511	.518	.518	.511	.511	3.1	3.0	2.8	3.1	2.4e4	7266	153	5097	4991	14.1	1.4
1000	1000	30	.501	.892	.897	.902	.500	.500	.501	.501	.501	.500	.499	0.11	0.10	0.09	0.11	301	238	2.7	1508	941	1.3	0.14
3000	1000	30	.502	.939	.942	.945	.501	.501	.501	.501	.501	.500	.501	0.33	0.31	0.29	0.32	2588	748	12.4	1136	1466	4.1	0.42
10000	1000	30	.501	.948	.948	.949	.500	.501	.507	.500	.500	.501	.500	1.3	1.2	1.2	1.2	8583	3621	32.4	1799	1794	14.0	1.4
30000	1000	30	.500	.950	.950	.950	.500	.501	.509	.500	.500	.500	.500	4.0	4.1	3.8	4.1	3.3e4	8006	90.7	5463	5190	42.0	4.3

- Propagating malware

Brief Overview of CW On-line Approach. Ma et al. developed a real-time classification of the URLs as to whether the associated website was malicious or not. Their approach can be visualized in Figure 4.10. That is, they implemented an on-line learning algorithm to classify websites with very good results (1% misclassification rate). They built upon their batch learning approach [71] with the reasoning that on-line algorithms are better suited to handle large number

Table 4.2: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 3 bins, averaged over 100 runs. Zero sub-domain weights were used as illustrated in Figure 4.2

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	1	1	29	43	0	0	0	0	0	0	0	59.6	87.8	92.1	93.9	0	40	-	53.9	51.1	54.6	21.0
3000	1000	10	23	1	45	76	0	0	0	10	3	20	0	86.1	90.1	94.5	97.6	0.1	52.1	-	79.9	74.6	83.9	26.0
10000	1000	10	92	0	71	96	0	0	0	91	78	95	0	92.0	90.0	97.1	99.6	7.3	61.1	-	99.1	97.8	99.5	29.9
30000	1000	10	100	0	97	100	0	2	0	100	100	100	0	100	90.0	99.7	100	10	67.3	-	100	100	100	26.9
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	25.5	66.2	70.2	72.7	0	25.9	-	30.1	28.9	24.7	17.9
3000	1000	30	0	4	25	34	0	0	0	0	0	0	0	58.6	92.2	96.0	97.1	0	35.4	-	53.7	50.3	52.2	27.9
10000	1000	30	2	86	99	100	0	0	0	1	0	1	0	90.3	99.5	100	100	0	44.6	-	84.1	79.7	87.0	35.8
30000	1000	30	94	100	100	100	0	0	0	58	25	82	0	99.8	100	100	100	0.3	50.1	-	98.3	96.2	99.3	40.2
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.803	.985	.990	.992	.654	.788	.737	.780	.768	.781	.765	0.09	0.09	0.08	0.09	352	151	3.1	1183	809	0.43	0.05
3000	1000	10	.817	.989	.994	.997	.660	.794	.782	.794	.782	.798	.775	0.25	0.23	0.21	0.23	2307	520	10.2	875	1105	1.4	0.15
10000	1000	10	.821	.990	.997	.999	.657	.802	.807	.798	.794	.800	.780	0.98	0.95	0.85	0.93	6233	1597	35.1	1305	1088	4.7	0.48
30000	1000	10	.822	.991	1.00	1.00	.656	.805	.814	.798	.793	.801	.781	3.0	3.0	2.8	3.0	2.6e4	3918	102	1477	1834	13.7	1.4
1000	1000	30	.758	.965	.969	.972	.594	.768	.728	.757	.746	.756	.755	0.12	0.12	0.12	0.12	302	188	3.3	1354	898	1.3	0.14
3000	1000	30	.797	.991	.996	.997	.580	.784	.776	.772	.758	.783	.775	0.34	0.32	0.31	0.32	2418	609	10.9	1233	1430	4.1	0.42
10000	1000	30	.812	.999	.999	.999	.594	.791	.801	.792	.782	.800	.786	1.3	1.3	1.2	1.2	7032	1998	37.9	2018	1802	14.0	1.4
30000	1000	30	.815	1.00	1.00	1.00	.612	.793	.808	.805	.801	.804	.790	4.0	3.8	3.6	3.8	3.1e4	5289	91.2	3186	2763	40.8	4.1
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	1	18	29	0	0	0	0	0	0	0	49.5	80.6	88.2	90.5	0	38.6	-	51.2	49.1	44.6	18.6
3000	1000	10	5	1	50	77	0	0	0	3	2	5	0	77.2	89.5	95.0	97.7	0	46.9	-	72.3	69.4	74.4	22.7
10000	1000	10	78	1	78	93	0	0	0	74	59	86	0	97.6	90.1	97.8	99.3	4.4	56.7	-	96.7	94.9	98.5	28.6
30000	1000	10	100	1	95	100	0	2	0	100	100	100	0	100	90.1	99.5	100	10	67.3	-	100	100	100	25.7
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	19.2	51.4	55.4	57.1	0	21.8	-	26.5	26.6	18.4	14.7
3000	1000	30	0	0	4	9	0	0	0	0	0	0	0	48.1	84.1	89.1	92.2	0	32.0	-	47.2	44.3	41.7	24.9
10000	1000	30	2	54	88	97	0	0	0	0	0	0	0	84.4	98.3	99.6	99.9	0	39.4	-	75.4	71.2	79.7	33.4
30000	1000	30	80	100	100	100	0	0	0	58	25	61	0	99.3	100	100	100	0.3	50.1	-	98.3	96.2	98.4	37.5
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.764	.926	.935	.939	.639	.761	.699	.751	.742	.743	.735	0.09	0.09	0.08	0.09	225	151	3.4	903	772	.43	.05
3000	1000	10	.782	.934	.944	.947	.642	.764	.746	.764	.754	.764	.747	0.24	0.24	0.21	0.24	2366	453	10.9	840	1180	1.5	0.16
10000	1000	10	.788	.940	.947	.949	.640	.768	.772	.768	.763	.769	.748	0.95	0.92	0.84	0.94	7566	1342	38.6	1144	1135	4.7	0.48
30000	1000	10	.790	.942	.949	.950	.656	.805	.814	.798	.793	.770	.751	3.1	3.0	2.8	3.0	1.8e4	4015	89.9	1784	1984	13.9	1.4
1000	1000	30	.715	.894	.896	.900	.586	.735	.693	.729	.721	.720	.720	0.12	0.12	0.11	0.12	304	201	3.2	813	924	1.3	0.14
3000	1000	30	.759	.934	.939	.943	.574	.752	.741	.742	.729	.747	.742	0.34	0.32	0.30	0.33	2718	609	10.6	1286	1435	4.3	0.44
10000	1000	30	.779	.947	.948	.949	.583	.758	.767	.757	.747	.768	.754	1.3	1.2	1.2	1.2	7806	2006	38.4	1950	1872	14.1	1.4
30000	1000	30	.784	.949	.949	.950	.612	.793	.808	.805	.801	.774	.759	4.0	3.9	3.6	3.9	2.2e4	5795	97.6	3151	3406	41.7	4.2

of observations and the need to adapt to evolutions in malicious websites and their corresponding features.

Features Description. The features extracted for each website fall into two categories: lexical and host-based. A general description of the type of features obtained can be seen in Figure 4.7. These features are obtained using crawling techniques. The lexical features are used to describe

Table 4.3: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 10 bins, averaged over 100 runs. Sinusoidal weights were used as illustrated in Figure 4.1

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	24	45	55	0	0	0	0	0	0	0	0.7	88.8	93.0	94.8	0	0.8	-	0.5	0.6	1.0	0.6
3000	1000	10	0	88	96	100	0	0	0	0	0	0	0	0.7	98.8	99.6	100	0	0.8	-	0.6	0.6	0.5	0.9
10000	1000	10	0	100	100	100	0	0	0	0	0	0	0	0.7	100	100	100	0	0.6	-	0.9	0.8	0.8	0.7
30000	1000	10	0	100	100	100	0	0	0	0	0	0	0	0.3	100	100	100	0	0.6	-	1.1	1.0	0.8	0.9
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.9	57.3	63.4	67.8	0	2.8	-	2.7	2.5	2.8	2.7
3000	1000	30	0	2	6	17	0	0	0	0	0	0	0	3.2	88.2	92.1	94.8	0	2.9	-	3.5	3.4	3.3	3.5
10000	1000	30	0	93	96	99	0	0	0	0	0	0	0	2.5	99.7	99.8	100	0	3.0	-	3.4	3.6	2.7	3.0
30000	1000	30	0	100	100	100	0	0	0	0	0	0	0	1.8	100	100	100	0	2.5	-	2.6	2.7	3.0	3.0
			Area under the ROC curve (AUC)											**Training Time** (sec)										
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.499	.976	.984	.988	.499	.500	.500	.501	.501	.499	.500	0.09	0.09	0.08	0.09	252	209	2.4	1013	1008	0.50	0.36
3000	1000	10	.501	.996	.997	.998	.500	.502	.501	.501	.501	.502	.501	0.23	0.22	0.20	0.23	2524	642	7.2	1025	1111	1.4	0.14
10000	1000	10	.500	.999	.999	.999	.500	.501	.500	.500	.500	.500	.500	0.93	0.91	0.84	0.94	8616	2199	27.4	2006	2186	4.6	0.47
30000	1000	10	.499	1.00	1.00	1.00	.500	.500	.501	.500	.500	.500	.500	3.1	3.0	2.8	3.1	2.7e4	8.0e3	69.5	6.2e3	4.9e3	14.0	1.4
1000	1000	30	.499	.935	.951	.957	.498	.500	.502	.500	.500	.500	.501	0.11	0.11	0.11	0.12	284	231	4.1	1036	1056	1.3	0.15
3000	1000	30	.500	.981	.989	.992	.500	.501	.501	.500	.500	.501	.500	0.34	0.32	0.30	0.33	2559	700	7.5	1056	1104	4.1	0.42
10000	1000	30	.500	.998	.998	.998	.500	.500	.501	.499	.499	.499	.500	1.3	1.2	1.1	1.2	8507	2468	27.2	1986	2104	13.7	1.4
30000	1000	30	.500	.999	.999	.999	.500	.500	.500	.500	.500	.500	.500	4.0	3.9	3.7	4.0	2.7e4	8.7e3	63.0	5.5e3	5.4e3	42.3	4.3
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	4	19	33	0	0	0	0	0	0	0	0.3	80.2	87.3	91.2	0	1.2	-	1.0	0.9	0.7	0.9
3000	1000	10	0	72	91	98	0	0	0	0	0	0	0	0.7	97.0	99.1	99.8	0	0.8	-	1.0	1.0	1.0	0.6
10000	1000	10	0	100	100	100	0	0	0	0	0	0	0	0.7	100	100	100	0	1.1	-	0.7	0.6	1.7	1.2
30000	1000	10	0	100	100	100	0	0	0	0	0	0	0	0.3	100	100	100	0	0.4	-	1.0	0.8	0.5	0.8
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	3.0	46.4	51.3	53.1	0	3.1	-	2.7	2.6	3.1	2.4
3000	1000	30	0	0	1	2	0	0	0	0	0	0	0	2.8	79.6	85.3	89.3	0	2.8	-	3.2	3.2	3.2	3.2
10000	1000	30	0	58	77	92	0	0	0	0	0	0	0	2.2	98.3	99.2	99.7	0	2.5	-	3.4	3.5	3.1	2.8
30000	1000	30	0	99	100	100	0	0	0	0	0	0	0	1.6	100	100	100	0	2.2	-	2.9	3.0	3.1	3.1
			Area under the ROC curve (AUC)											**Training Time** (sec)										
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.499	.914	.925	.932	.500	.501	.502	.502	.502	.501	.501	0.11	0.11	0.11	0.12	270	171	3.8	1051	1078	0.47	0.06
3000	1000	10	.500	.942	.946	.947	.500	.500	.501	.501	.501	.501	.500	0.24	0.23	0.21	0.25	2971	642	9.1	1144	1157	1.5	0.17
10000	1000	10	.499	.949	.949	.949	.500	.499	.500	.499	.499	.499	.500	0.95	0.93	0.87	0.98	1.2e4	2087	28.8	2347	2163	4.6	0.47
30000	1000	10	.500	.949	.950	.950	.500	.500	.500	0.500	.500	.500	.501	3.1	3.1	2.9	3.2	1.9e4	8.0e3	100.0	5.1e3	5.8e3	14.0	1.4
1000	1000	30	.499	.872	.881	.884	.501	.501	.503	.500	.500	.500	.500	0.15	0.15	0.14	0.15	524	215	3.8	1130	1040	1.4	0.16
3000	1000	30	.499	.922	.932	.937	.501	.501	.500	.499	.499	.500	.501	0.34	0.33	0.31	0.35	2343	791	13.5	1222	1133	4.6	0.47
10000	1000	30	.500	.945	.947	.948	.500	.500	.500	.500	.500	.500	.500	1.3	1.2	1.1	1.3	1.1e4	2774	29.9	2294	2260	13.9	1.4
30000	1000	30	.500	.949	.949	.949	.500	.500	.500	0.500	.500	0.500	0.500	4.0	3.8	3.7	4.0	2.4e4	8.8e3	93.3	5.4e3	6.0e3	41.8	4.2

the characteristic that malicious websites "look different" while host-based features describe the website host. Ma et al. represent lexical features using bag-of-words for tokens in the URL. The 3.2 million features are accumulated over the 121 with each day adding features to existing ones.

Data Collection. They obtain the malicious URLS from a large Webmail provider (the provider requested a to be kept anonymous). The benign URLs were drawn randomly from Yahoo's directory listing. The average ratio of benign-to-malicious URLS is 2-to-1. These sources only provide the

Table 4.4: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 10 bins, averaged over 100 runs. Zero sub-domain weights were used as illustrated in Figure 4.2

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	14	31	40	0	0	0	0	0	0	0	59.6	85.8	90.4	92.2	0	40.4	-	52.9	50.3	53.9	20.4
3000	1000	10	30	77	86	97	0	0	0	13	6	24	0	89.5	97.7	98.6	99.7	0.1	50.6	-	80.6	76.6	86.5	25.1
10000	1000	10	99	100	100	100	0	0	0	91	80	93	0	99.9	100	100	100	7.4	57.5	-	99	97.7	99.3	28.2
30000	1000	10	100	100	100	100	0	2	0	100	100	100	0	100	100	100	100	10.0	65.8	-	100	100	100	29.9
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	27.2	54.1	62.2	64.9	0	27.4	-	31.6	31.3	24.0	17.5
3000	1000	30	0	0	1	8	0	0	0	0	0	0	0	59.1	86.0	90.4	93.7	0	35.1	-	53.4	50.0	53.2	29.5
10000	1000	30	9	67	96	98	0	0	0	1	0	3	0	91.7	98.8	99.9.	99.9	0	44.2	-	84.1	80	88.3	36.5
30000	1000	30	95	100	100	100	0	0	0	59	30	79	0	99.8	100	100	100	0.4	48.5	-	98.3	96.3	99.3	39.3
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.799	.974	.981	.984	.580	.784	.732	.773	.762	.776	.767	0.08	0.08	0.08	0.09	239	157	3.2	771	832	0.48	0.09
3000	1000	10	.813	.989	.994	.996	.583	.790	.773	.788	.779	.792	.775	0.30	0.29	0.26	0.30	2636	431	9.7	920	874	1.5	0.17
10000	1000	10	.818	.997	.999	.999	.607	.795	.802	.796	.796	.797	.777	0.94	0.93	0 .86	0.95	7619	1368	38.1	1139	1182	4.6	0.47
30000	1000	10	.818	.998	1.00	1.00	.626	.800	.810	.796	.797	.798	.778	3.1	3.1	2.9	3.2	2.1e4	4.0e3	96.1	1.8e3	1.5e3	14.0	1.4
1000	1000	30	.761	.930	.946	.950	.593	.775	.733	.765	.752	.758	.760	0.15	0.15	0.14	0.15	310	197	3.3	1021	884	1.4	0.15
3000	1000	30	.801	.979	.986	.989	.582	.787	.781	.777	.762	.787	.780	0.38	0.38	0.35	0.38	2791	677	11.3	1283	1314	4.6	0.49
10000	1000	30	.817	.996	.997	.998	.590	.795	.805	.795	.784	.804	.789	1.2	1.2	1.1	1.2	1.0e4	2162	38.5	1677	1870	13.7	1.4
30000	1000	30	.819	.999	.999	.999	.619	.796	.812	.810	.807	.807	.793	4.1	3.9	3.7	4.0	2.3e4	5.9e3	92.1	2.9e3	3.1e3	42.0	4.2
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	1	10	19	0	0	0	0	0	0	0	50.3	78.0	83.4	86.8	0	37.8	-	48.1	45.7	43.5	20.0
3000	1000	10	18	44	67	85	0	0	0	7	2	8	0	84.1	93.6	96.5	98.5	0	45.3	-	75.1	70.5	77.7	22.8
10000	1000	10	89	99	100	100	0	1	0	86	73	77	0	98.9	99.9	100	100	6.5	54.7	-	98.6	97.2	97.6	28.0
30000	1000	10	100	100	100	100	0	1	0	100	100	100	0	100	100	100	100	10.0	61.6	-	100	100	100	26.3
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	21.2	46.7	49.7	51.5	0	23.2	-	18.7	15.1	20.1	15.2
3000	1000	30	0	0	0	1	0	0	0	0	0	0	0	50.1	77.4	83.6	87.8	0	32.6	-	47.0	30.8	42.9	25.4
10000	1000	30	2	39	75	93	0	0	0	0	0	0	0	86.9	97.2	99.0	99.8	0	39.3	-	77.2	49.6	82.2	34.9
30000	1000	30	83	99	100	100	0	0	0	44	16	60	0	99.4	100	100	100	0.1	45.6	-	97.3	94.0	98.3	38.4
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.762	.915	.923	.927	.569	.756	.697	.743	.732	.741	.737	0.12	0.12	0.12	0.13	381.1	129.4	3.6	855.6	905.8	0.46	0.06
3000	1000	10	.780	.936	.942	.943	.576	.758	.742	.756	.747	.758	.744	0.32	0.31	0.29	0.33	2.8e3	499.1	11.2	1.0e3	956.2	1.6	0.19
10000	1000	10	.786	.945	.948	.949	.589	.764	.769	.766	.765	.766	.748	0.92	0.91	0.84	0.96	8.8e3	1.6e3	38.4	1.3e3	1.2e3	4.6	0.47
30000	1000	10	.786	.947	.949	.949	.614	.768	.778	.767	.767	.768	.749	3.1	3.0	2.8	3.1	1.8e4	4.3e3	102.6	2.2e3	1.9e3	13.9	1.5
1000	1000	30	.718	.872	.877	.880	.580	.737	.693	.731	.730	.721	.722	0.11	0.11	0.10	0.11	434.1	185.3	3.7	1.1e3	1.0e3	1.3	0.15
3000	1000	30	.764	.919	.927	.932	.577	.756	.745	.744	.732	.751	.745	0.34	0.33	0.31	0.35	3.1e3	675.8	11.1	1.3e3	1.1e3	4.5	0.46
10000	1000	30	.784	.943	.945	.947	.580	.763	.771	.760	.752	.772	.759	1.2	1.2	1.1	1.2	1.3e4	2.4e3	38.5	1.9e3	2.1e3	13.7	1.4
30000	1000	30	.787	.948	.948	.949	.602	.766	.780	.777	.772	.776	.762	4.0	3.8	3.7	4.0	2.3e4	6.8e3	111.2	3.6e3	3.2e3	41.7	4.2

URLs; they wrote the web crawling techniques that query for DNS, WHOIS, geographic information servers, etc., for every incoming URL.

On-line Algorithm. Ma et al. tested several on-line algorithms and settled upon the Confidence-Weighted algorithm [28, 37]. The punchline behind this algorithm is that it represents confidence for each feature via an uncertainty weight w_i with a Gaussian distribution $N(\mu_i, \Sigma_i)$ where $\boldsymbol{\mu}$ is the vector of feature means and $\boldsymbol{\Sigma}$ is the diagonal covariance matrix. For an observation $\boldsymbol{x}$, the

Table 4.5: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 50 bins, averaged over 100 runs. Sinusoidal weights were used as illustrated in Figure 4.1

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	0	1	7	0	0	0	0	0	0	0	0.9	66.7	76.1	81.3	0	1.3	-	0.7	0.6	0.8	0.5
3000	1000	10	0	2	27	68	0	0	0	0	0	0	0	0.9	81.8	90.2	96.4	0	0.9	-	0.7	0.7	0.5	0.6
10000	1000	10	0	10	85	99	0	0	0	0	0	0	0	0.2	87.7	98.5	99.9	0	0.7	-	1.3	1.2	1.2	1.0
30000	1000	10	0	24	100	100	0	0	0	0	0	0	0	0.5	91.0	100	100	0	0.6	-	0.9	0.9	0.9	1.1
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.9	40.8	45.6	50.0	0	2.7	-	2.4	2.5	2.5	2.8
3000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.8	66.0	71.8	79.7	0	3.2	-	1.9	2.3	3.0	2.7
10000	1000	30	0	0	4	43	0	0	0	0	0	0	0	2.5	88.5	93.4	97.7	0	2.2	-	2.4	2.5	2.7	2.7
30000	1000	30	0	0	1	69	0	0	0	0	0	0	0	1.9	93.5	94.2	98.9	0	2.3	-	2.8	2.9	2.4	2.7
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.502	.922	.944	.959	.500	.501	.501	.503	.502	.500	.501	0.10	0.10	0.10	0.11	279.1	210.7	2.4	1.4e3	1.2e3	0.45	0.08
3000	1000	10	.501	.955	.975	.989	.500	.502	.501	.502	.502	.502	.502	0.26	0.26	0.24	0.29	2.4e3	649.8	7.4	1.2e3	1.3e3	1.5	0.38
10000	1000	10	.500	.970	.994	.998	.500	.500	.500	.500	.501	.500	.500	0.93	0.94	0.87	1.0	6.6e3	2.6e3	29.2	2.3e3	2.3e3	4.7	0.55
30000	1000	10	.500	.976	.999	.999	.500	.500	.500	.500	.500	.500	.500	3.0	3.1	2.9	3.3	3.0e4	7.0e3	99.8	6.6e3	6.6e3	14.0	2.8
1000	1000	30	.503	.872	.901	.912	.499	.502	.503	.499	.499	.499	.500	0.11	0.11	0.11	0.12	327.8	245.3	3.1	1.2e3	1.3e3	1.3	0.15
3000	1000	30	.501	.929	.950	.966	.500	.501	.502	.502	.500	.502	.501	0.36	0.36	0.34	0.39	2.7e3	738.6	10.6	1.1e3	1.0e3	4.3	0.71
10000	1000	30	.500	.973	.982	.992	.500	.500	.501	.500	.500	.500	.499	1.3	1.3	1.2	1.3	8.7e3	3.1e3	21.4	2.2e3	2.4e3	14.0	1.4
30000	1000	30	.500	.985	.988	.996	.501	.500	.500	.499	.500	.500	.499	3.9	3.9	3.7	4.1	2.8e4	8.6e3	98.7	7.1e3	6.4e3	41.8	6.5
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	0	0	1	0	0	0	0	0	0	0	0.8	59.5	68.1	73.5	0	1.0	-	1.1	0.9	0.6	1.1
3000	1000	10	0	2	12	34	0	0	0	0	0	0	0	1.0	80.2	87.0	92.5	0	1.0	-	1.3	1.2	1.0	1.0
10000	1000	10	0	11	63	96	0	0	0	0	0	0	0	0	86.9	96.0	99.6	0	0.6	-	0.9	1.0	0.9	0.6
30000	1000	10	0	21	99	100	0	0	0	0	0	0	0	0.3	89.9	99.9	100	0	0.9	-	1.3	1.3	1.1	1.0
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	2.8	34.1	38.7	42.4	0	2.8	-	2.8	2.6	2.7	2.7
3000	1000	30	0	0	0	0	0	0	0	0	0	0	0	3.1	59.6	64.3	72.1	0	2.5	-	2.9	2.8	2.9	2.9
10000	1000	30	0	0	0	20	0	0	0	0	0	0	0	2.5	85.4	89.8	95.6	0	2.2	-	2.3	2.2	2.7	2.8
30000	1000	30	0	0	1	54	0	0	0	0	0	0	0	1.9	93.5	94.0	98.3	0	2.2	-	2.9	2.8	2.7	2.6
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.500	.866	.885	.898	.500	.501	.502	.502	.502	.502	.501	0.10	0.10	0.10	0.11	418	181.4	3.2	1.1e3	1.2e3	0.46	0.06
3000	1000	10	.501	.905	.920	.931	.500	.501	.501	.502	.502	.501	.500	0.25	0.25	0.23	0.28	2.3e6	648.3	.6.5	1.4e3	1.3e3	1.4	0.18
10000	1000	10	.500	.922	.941	.948	.500	.499	.500	.500	.500	.500	.499	0.93	0.95	0.88	0.97	8.7e3	2.7e3	29.8	2.2e3	2.1e3	4.8	0.64
30000	1000	10	.500	.928	.949	.949	.500	.500	.500	.500	.500	.500	.500	3.1	3.1	2.9	3.4	2.1e4	7.3e3	152.7	7.1e3	6.8e3	14.0	1.8
1000	1000	30	.501	.820	.841	.851	.500	.501	.503	.499	.499	.500	.500	0.13	0.14	0.13	0.14	451	229.0	3.1	1.1e3	1.1e3	1.3	0.15
3000	1000	30	.500	.874	.892	.907	.500	.501	.501	.499	.499	.500	.501	0.34	0.34	0.33	0.37	2.7e3	756.4	10.8	1.e3	1.1e3	4.2	0.44
10000	1000	30	.500	.919	.928	.939	.500	.500	.501	.500	.500	.499	.500	1.3	1.3	1.2	1.4	9.4e3	2.8e3	24.5	2.2e3	2.1e3	14.2	1.5
30000	1000	30	.500	.936	.938	.945	.500	.500	.500	.500	.500	.500	.500	4.0	4.0	3.8	4.2	2.3e4	8.3e3	103.7	7.1e3	6.2e3	42.0	4.9

prediction of its class (-1 or $+1$) is given by the signed dot product of the observation and the mean: $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\mu} \cdot \boldsymbol{x})$.

Evaluation of Batch and On-line Algorithms. Ma et al. found that "continuous" training produced superior results compared to "interval-based" training. The continuous approach, the typical method for on-line algorithms, allows the classifier to retrain after each observation. The interval-based approach is more typical of batch algorithms (retraining occurs after an interval of

Table 4.6: Classification experiments on simulated linearly separable data (top table) and noisy data (bottom table) with $\delta = 0.9$, 50 bins, averaged over 100 runs. Zero sub-domain weights were used as illustrated in Figure 4.2

Noiseless																							
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	0	4	6	0	0	0	0	0	0	0	61.0	68.3	75.9	79.9	0	41.6	-	54.0	50.8	54.2	22.1
3000	1000	10	30	4	16	31	0	0	0	15	9	31	0	89.6	82.8	88.8	91.9	0.3	49.8	-	81.3	77.9	87.9	25.8
10000	1000	10	98	42	56	69	0	0	0	83	81	95	0	99.8	93.6	95.6	96.9	6.6	57.5	-	1.1e3	1.1e3	99.5	28.5
30000	1000	10	100	84	89	89	0	4	0	100	100	99	0	100	98.4	98.9	98.9	10.0	66.5	-	100	100	99.9	29.8
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	27.1	38.8	43.9	48.2	0	25.1	-	31.5	30.8	24.7	18.2
3000	1000	30	0	0	0	0	0	0	0	0	0	0	0	59.2	63.6	70.8	78.1	0	30.2	-	53.3	50.0	52.6	27.8
10000	1000	30	2	0	0	15	0	0	0	1	0	0	0	91.7	82.7	90.6	95.6	0	44.0	-	56.5	53.5	88.4	36.4
30000	1000	30	95	0	0	39	0	0	0	58	27	73	0	99.8	90.9	93.4	97.6	0.4	48.5	-	98.2	95.5	99.0	40.6
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.799	.935	.953	.963	.561	.785	.730	.773	.760	.774	.765	0.08	0.09	0.08	0.10	295	141.8	2.7	1.0e3	911.4	0.44	0.06
3000	1000	10	.814	.966	.977	.985	.583	.789	.778	.787	.779	.792	.773	0.25	0.25	0.23	0.28	2.4e3	494.7	12.6	980.6	938.7	1.4	0.25
10000	1000	10	.819	.987	.992	.994	.611	.797	.803	.797	.797	.797	.778	.984	1.0	.94	1.1	6.8e3	1.2e3	30.7	1.1e3	1.1e3	4.8	0.50
30000	1000	10	.819	.995	.997	.997	.629	.801	.811	.795	.796	.799	.778	3.1	3.2	3.0	3.4	2.0e4	4.5e3	111.8	2.2e3	2.2e3	14.1	2.3
1000	1000	30	.762	.879	.906	.918	.570	.775	.736	.766	.753	.760	.760	0.11	0.12	0.11	0.13	389	186.2	2.6	1.1e3	1.0e3	1.3	0.15
3000	1000	30	.803	.929	.951	.965	.584	.788	.781	.777	.763	.787	.779	0.35	0.35	0.33	0.38	2.4e3	605.9	13.3	1.4e3	1.5e3	4.2	0.48
10000	1000	30	.817	.962	.979	.989	.594	.795	.805	.795	.784	.805	.790	1.3	1.3	1.2	1.4	8.4e3	1.9e3	29.6	2.0e3	1.7e3	14.1	1.4
30000	1000	30	.819	.979	.986	.994	.619	.797	.812	.810	.805	.807	.792	4.0	4.0	3.8	4.2	2.7e4	5.9e3	114.5	4.0e3	3.7e3	41.9	6.2
Noisy																								
			All-variable detection rate (DR)											**Percent correctly detected (PCD)**									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	0	0	1	3	0	0	0	0	0	0	0	49.3	62.6	68.3	73.4	0	38.5	-	50.1	46.5	43.7	19.9
3000	1000	10	17	2	6	19	0	0	0	2	2	11	0	83.6	78.8	84.4	89.3	0.1	44.9	-	74.8	69.9	77.9	24.2
10000	1000	10	85	29	46	58	0	0	0	68	62	77	0	98.5	91.7	94.4	95.7	4.7	54.8	-	96.6	95.3	97.7	28.9
30000	1000	10	100	78	82	84	0	1	0	100	100	100	0	100	97.7	98.2	98.4	9.9	62.4	-	100	100	100	0
1000	1000	30	0	0	0	0	0	0	0	0	0	0	0	20.4	34.3	37.7	41.0	0	23.0	-	27.1	27.3	20.5	14.6
3000	1000	30	0	0	0	0	0	0	0	0	0	0	0	49.8	57.7	63.2	69.5	0	27.0	-	46.8	44.2	43.0	25.0
10000	1000	30	0	0	0	2	0	0	0	0	0	0	0	87.3	79.8	87.1	93.1	0	40.4	-	53.6	50.3	82.2	34.7
30000	1000	30	75	0	0	20	0	0	0	33	13	52	0	99.2	89.2	92.8	96.4	0.1	44.8	-	96.5	4.1e3	97.8	39.0
			Area under the ROC curve (AUC)											**Training Time** (sec)									
N	M	k=k^*	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1	FSA	PFSA	PFSV	PFSL	L1	EL	L2	MCP	SCD	LB	LB1
1000	1000	10	.760	.883	.895	.905	.572	.755	.696	.741	.729	.737	.736	0.09	0.10	0.09	0.11	367	140.1	3.1	983.7	952.7	0.43	0.07
3000	1000	10	.780	.911	.923	.929	.583	.759	.743	.754	.746	.758	.746	0.26	0.28	0.25	0.30	2.1e3	513.3	10.6	1.2e3	1.2e3	1.6	0.79
10000	1000	10	.787	.934	.941	.942	.594	.766	.770	.767	.766	.766	.749	0.97	0.98	0.92	1.1	7.0e3	1.4e3	31.7	1.4e3	1.2e3	4.7	0.48
30000	1000	10	.787	.944	.946	.947	.615	.769	.778	.766	.766	.769	.749	3.0	3.1	2.9	3.4	1.7e4	4.7e3	110.2	2.9e3	2.3e3	13.7	1.4
1000	1000	30	.717	.829	.846	.854	.564	.738	.695	.732	.724	.721	.721	0.12	0.13	0.12	0.14	488	202.7	2.7	1.0e3	989	1.3	0.14
3000	1000	30	.765	.876	.894	.907	.577	.757	.745	.746	.733	.750	.746	0.36	0.36	0.35	0.40	2.4e3	613.2	10.9	1.4e3	1.4e3	4.4	1.1
10000	1000	30	.784	.910	.925	.936	.580	.764	.772	.762	.752	.772	.759	1.3	1.3	1.2	1.4	8.1e3	1.9e3	33.5	2.0e3	1.7e3	13.9	1.4
30000	1000	30	.787	.929	.937	.942	.602	.766	.780	.776	.771	.776	.763	4.0	3.9	3.7	4.2	2.0e4	5.5e3	116.1	4.6e3	4.1e3	41.0	4.2

time has passed).

They also found that allowing the number of features to change dynamically with new observations ("variable") encountered generated better results than using a fixed number of features. Thus, the variable feature approach permitted the model to grow with newly encountered features.

In Figure 4.11, Ma et al. demonstrate the performance of the on-line algorithm CW versus varying batch algorithms that use a SVM classifier. Their figure demonstrates that CW can adapt to new

Table 4.7: Example of a $\boldsymbol{\beta}$ recovered with $k^* = 1000$ and 100 days

Lexical		Host-Based	
Feature type	**Count**	**Feature type**	**Count**
Hostname	835,764	WHOIS info	917,776
Primary domain	738,201	IP prefix	131,930
Path tokens	124,401	AS number	39,843
Last path token	92,367	Geographic	28,263
TLD	522	Conn. speed	52
Lexical misc.	6	Host misc.	37
Lexical	1,791,261	Host-Based	1,117,901

features over time which allows it to obtain better results than the SVM batch algorithms. Note that the vertical axis represents cumulative error and the horizontal axis represents the days used for training. To be clear, the error here represents the percentage of misclassified examples for all URLS encountered up to that date. To understand the meaning of each batch algorithm, we refer the user to their paper. Ma et al. believe that the success of the CW algorithm comes from treating the features differently - a characteristic we hope to exploit in PFSA.

4.2.2 Application of PFSA to URL Reputation

We provide the details of how the parameter sensitive classifiers are used in the URL Reputation dataset. As described in Section 4.2.1, this dataset has the characteristic that variables can have non-zero values for some days and then be completely zero for other days. For example, a variable representing a particular geographic location from where a malicious website is originating can have non-zero values on Day N. A few days later, perhaps due to the developer's desire to not be caught, this location is no longer generating a malicious website which leads to this variable taking on zero values for the remaining days. We want to take this erratic behavior in account when training our classifiers.

Training Details. We provide a visual of this parameterized $\boldsymbol{\beta}$ over time in Table 4.8. The idea is that there can be different coefficient values depending on day t. More specifically, suppose we are trying to recover the $\boldsymbol{\beta}$ over 100 days of data with $k^* = 1,000$. Each row will represent a selected feature and each column of a row represents the value of the feature at particular day t.

Table 4.8: Example of a $\boldsymbol{\beta}$ recovered with $k^* = 1000$ and 100 days

$\boldsymbol{\beta}_1$	$b_{1,1}$	$b_{1,2}$	$\cdots$	$b_{1,100}$
$\boldsymbol{\beta}_2$	$b_{2,1}$	$b_{2,2}$	$\cdots$	$b_{2,100}$
$\vdots$			$\vdots$	
$\boldsymbol{\beta}_{1000}$	$b_{1000,1}$	$b_{1000,2}$	$\cdots$	$b_{1000,100}$
	t_1	t_2	$\cdots$	t_{100}

Recall that the data is provided in CSV files with $20,000$ website observations per file. The files have a naming convention associated with the day the data was collected: Day0.csv, Day1.csv, etc. We load Day0 - Day99 as our training set and train our Lorenz based classifier over 500 epochs. Then, just as Ma et al. did, we test on Day100. We used a learning rate (η) of 0.5, a second order penalty value of 0.01, and a ridge value of 0.001.

4.2.3 Results

We now present the results from applying FSA and PFSA to the URL Reputation dataset. We also provide comparison results on the methods CW, SVM and Logistic Regression with Stochastic Gradient descent as reported by [72] . In Table 4.9, we can see that the CW method has the best test error while SVM performs the worst. We note that while FSA and PFSA have a higher test error than CW, they reduce the number of features by 85%. We thought it was curious that PFSA didn't outperform FSA but the reason may be similar to why FSA performed better than our parameter sensitive classifier in the simulations with zero sub-domain data. Recall that we mentioned that during the portion of the weights that are zero, our PFSA acted like a parameter insensitive classifier. The nature of the URL Reputation dataset has similar behavior in that a feature may suddenly disappear (and most do) with no smooth transition. This abruptness is a behavior our PFSA probably has challenges with.

Table 4.9: Experiments results on URL dataset. Our implementations are third and fourth methods. We use linear FSA algorithms with a Lorenz loss. Note that while we do not outperform CW (the on-line algorithm), we come close using less than one-fifth the number of features. We also have a version of FSA that uses the features from the CW implementation that obtains the same test error.

Method	Number of features k	Error train %	Error valid %	Error test %
SVM	all	-	-	1.8
Log Reg-SGD	all	-	-	1.6
Linear PFSA-Lorenz (3 bins)	75,000	0.49	-	1.16
Linear FSA-Lorenz	75,000	0.50	-	1.15
CW [72]	500,000	0.23	-	1.0

4.3 Face Detection Results

We now present results on two standard face detection datasets. The detection time is about 3 seconds for a 480×320 image with unoptimized C++ code, with the most time used for detecting the keypoints. We expect to obtain speedups of $10 - 100$ times with a GPU implementation and code optimization.

4.3.1 FDDB Dataset

Dataset Description. In the technical report [56], Jain and Learned-Miller present the Face Detection Data Set and Benchmark (FDDB) with the aim of introducing a database for unconstrained face detection that permits efficient comparison of face detection methods. The database has $2,845$ images with a total of $5,171$ faces. Furthermore, their evaluation process has the goal of enforcing:

1. Reproducibility
2. Wide range of face poses and occlusions
3. Detailed annotations
4. Gray-scale and color images.

On their website at *http://vis-www.cs.umass.edu/fddb/*, researchers can download a script to evaluate their method against other state-of-the-art. They explain their paper how they used elliptical regions to specify face regions, as visualized in Figure 4.12.

Results. Our results are: "Ours, N^{supp}=" are obtained by pruning the candidates with the support threshold N^{supp} and scoring them with the parameter sensitive classifier. "Ours, Center only" has candidates predicted only from the face center detections, with $N^{supp} = 1$."Ours, LBF" uses a parameter insensitive classifier trained with the logistic loss.

The results on the FDDB dataset are shown in Figure 4.13. Also shown are results from the Joint Cascade [26], HeadHunter [74], Boosted Exemplar [67], ACF [116], Yan et al [114] and Zhu [126]. One can see that the proposed method obtains very good results close to the state of the art. For at least 1000 false positive it lags behind only the HeadHunter [74] and for false positives below 800 it is also outperformed by the Joint Cascade [26].

In Figure 4.14 are shown more results with different values of the support parameter N^{supp} on FDDB. One can see that the performance increases up to $N^{supp} = 4$, but the range of false positives shrinks. It is also clear that by using multiple keypoints to propose the candidates instead of just the face center the detection accuracy increases considerably. The parameter sensitive classifier also brings some improvement, though not as much.

4.3.2 AFW Dataset

Dataset Description. In [126], Zhu and Ramanan present an annotated faces in the wild (AFW) built from Flickr images. Their dataset is composed of 205 images with at least one large face (a total of 468 faces). They label each face with a bounding box and 6 landmarks. They offer code on their website where anyone can train and test their method.

Results. The results on the AFW dataset are shown in Figure 4.17. Also shown are results from the Head Hunter [74], Shen et al [100], Structured Models [115] and Zhu [126]. Again, the algorithm performs well in the high recall regime and lags a little behind in the high precision regime. What this means is that we can detect many of the faces but also label non-face image patches as faces. Some improvements that we can make include:

1. Use more discriminating face features
2. Train our score function using CNN

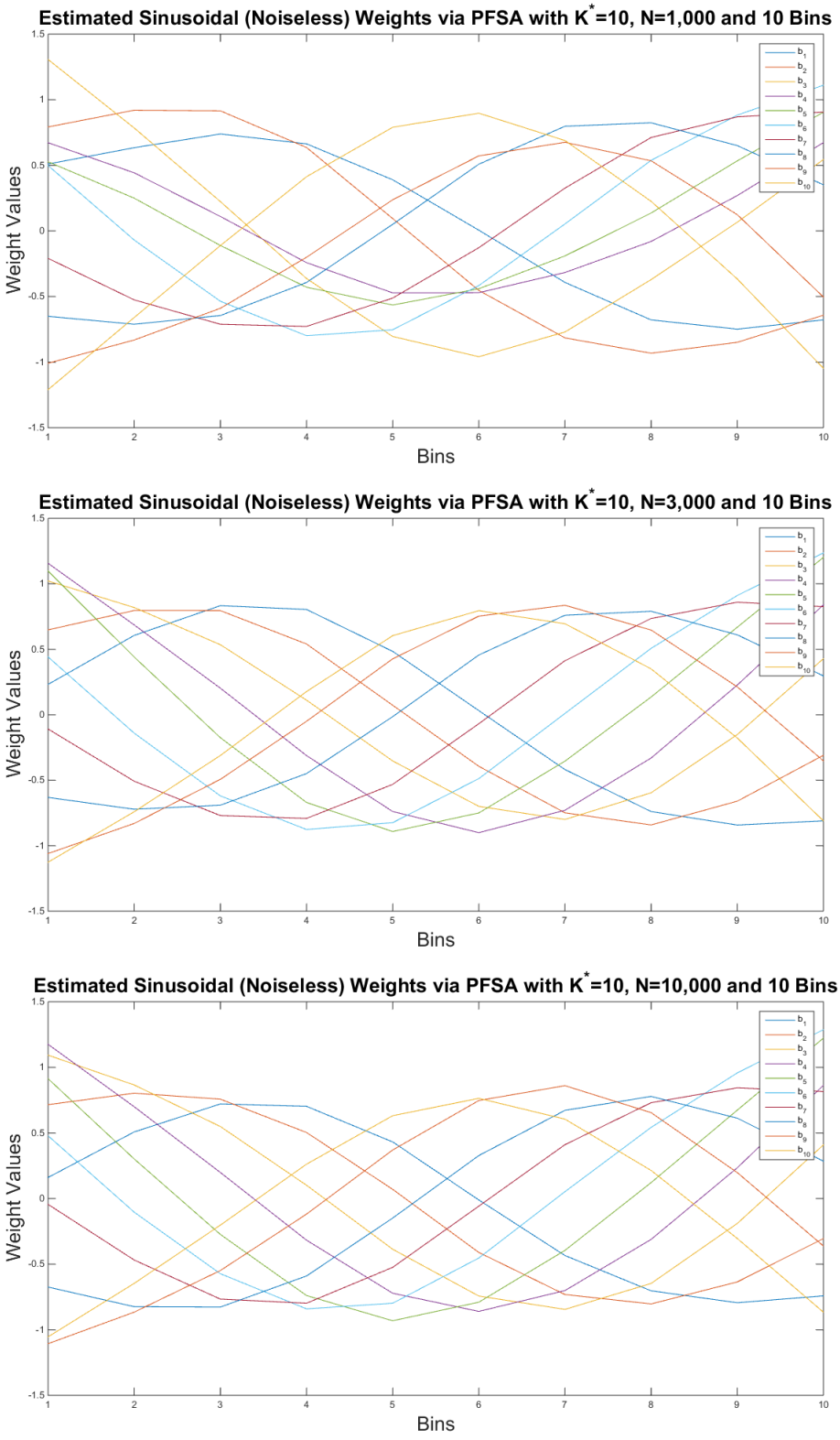

Figure 4.4: Recovered weights in the case of Sinusoidal Noiseless weights using a parameter sensitive classifier and FSA with a logistic loss (PFSA)

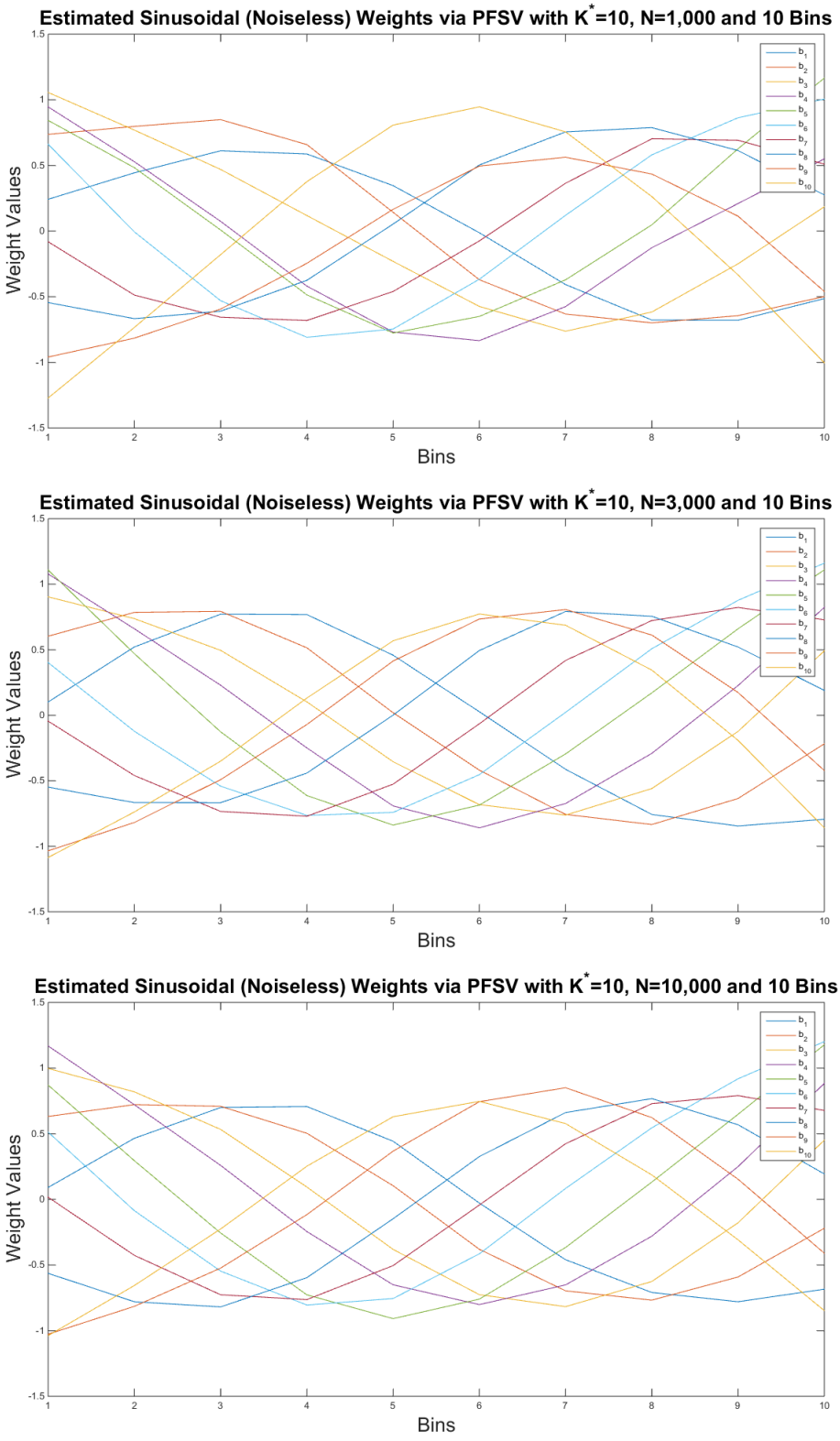

Figure 4.5: Recovered weights in the case of Sinusoidal Noiseless weights using a parameter sensitive classifier and FSA with a huberised loss (PFSV)

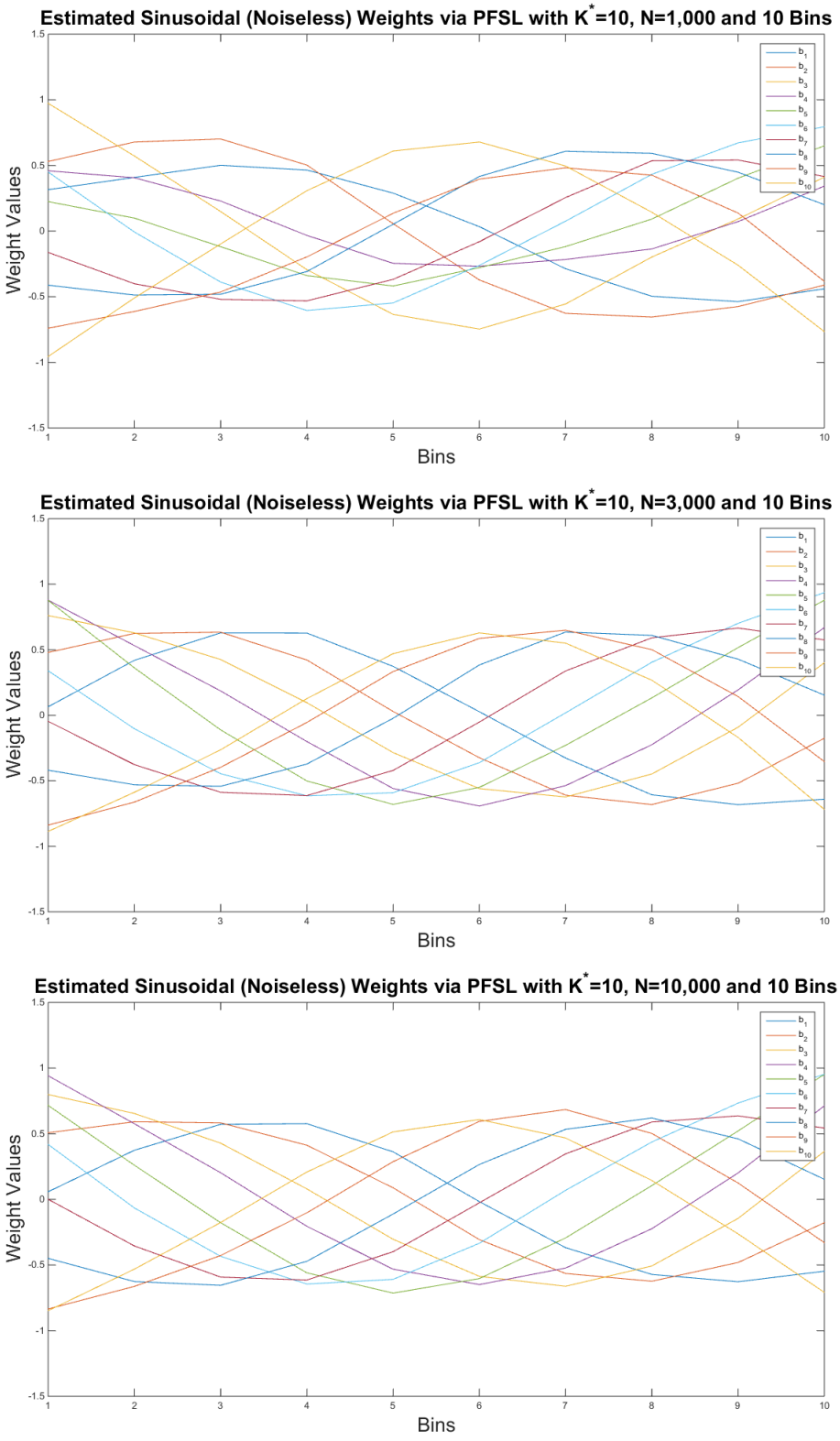

Figure 4.6: Recovered weights in the case of Sinusoidal Noiseless weights using a parameter sensitive classifier and FSA with a lorenz loss (PFSL)

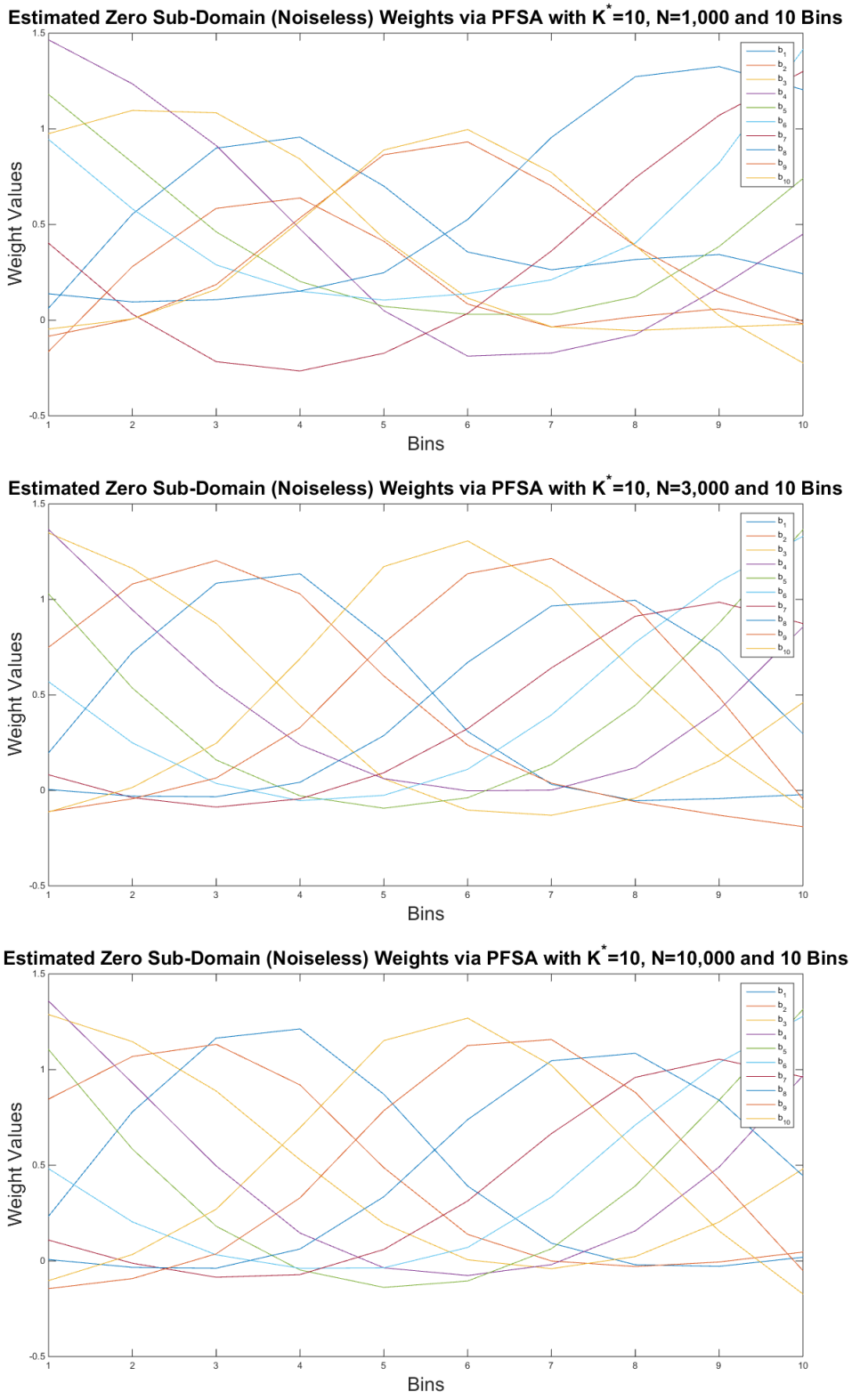

Figure 4.7: Recovered weights in the case of Zero Sub-Domain Noiseless weights using a parameter sensitive classifier and FSA with a logistic loss (PFSA)

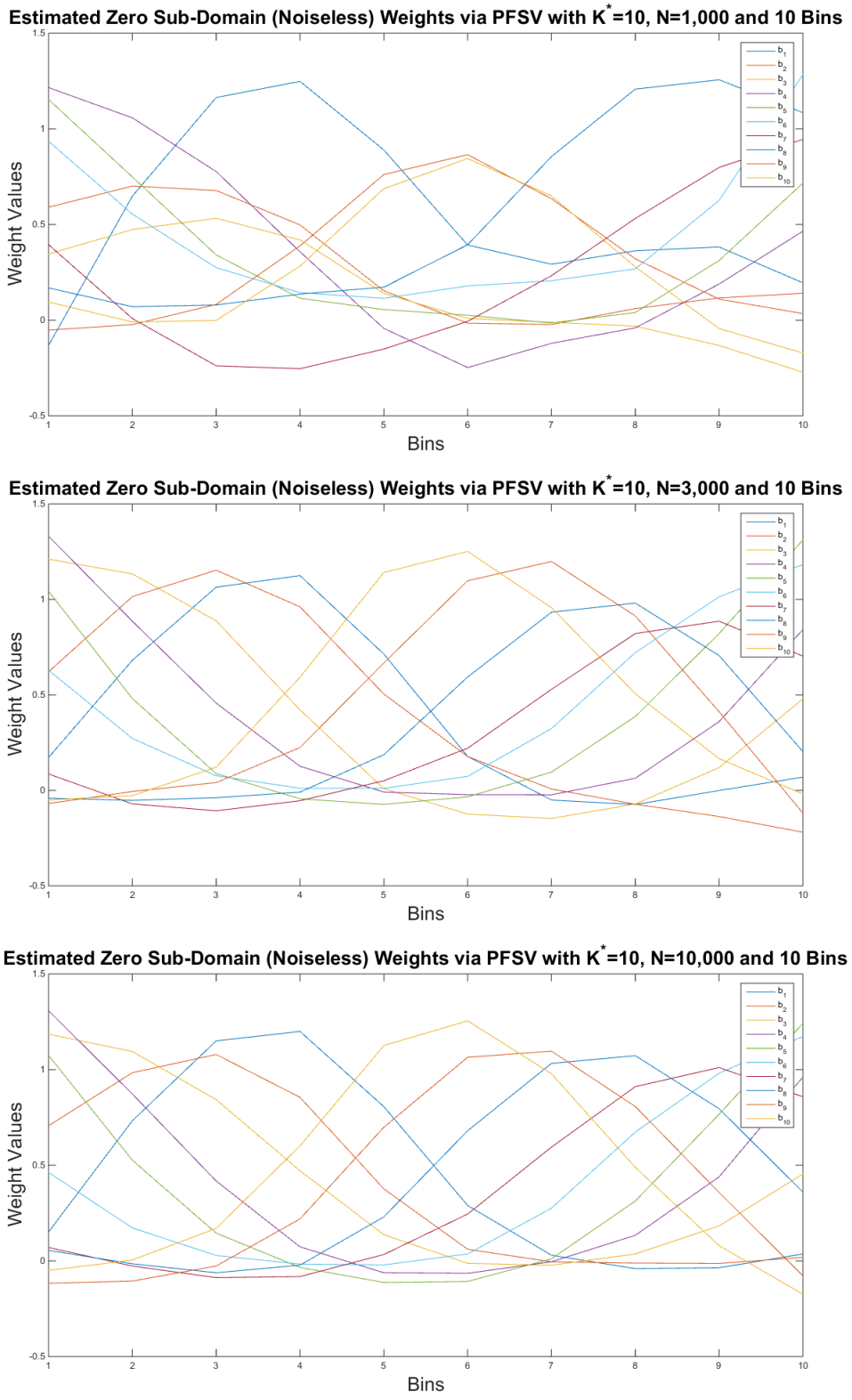

Figure 4.8: Recovered weights in the case of Zero Sub-Domain Noiseless weights using a parameter sensitive classifier and FSA with a huberised loss (PFSV)

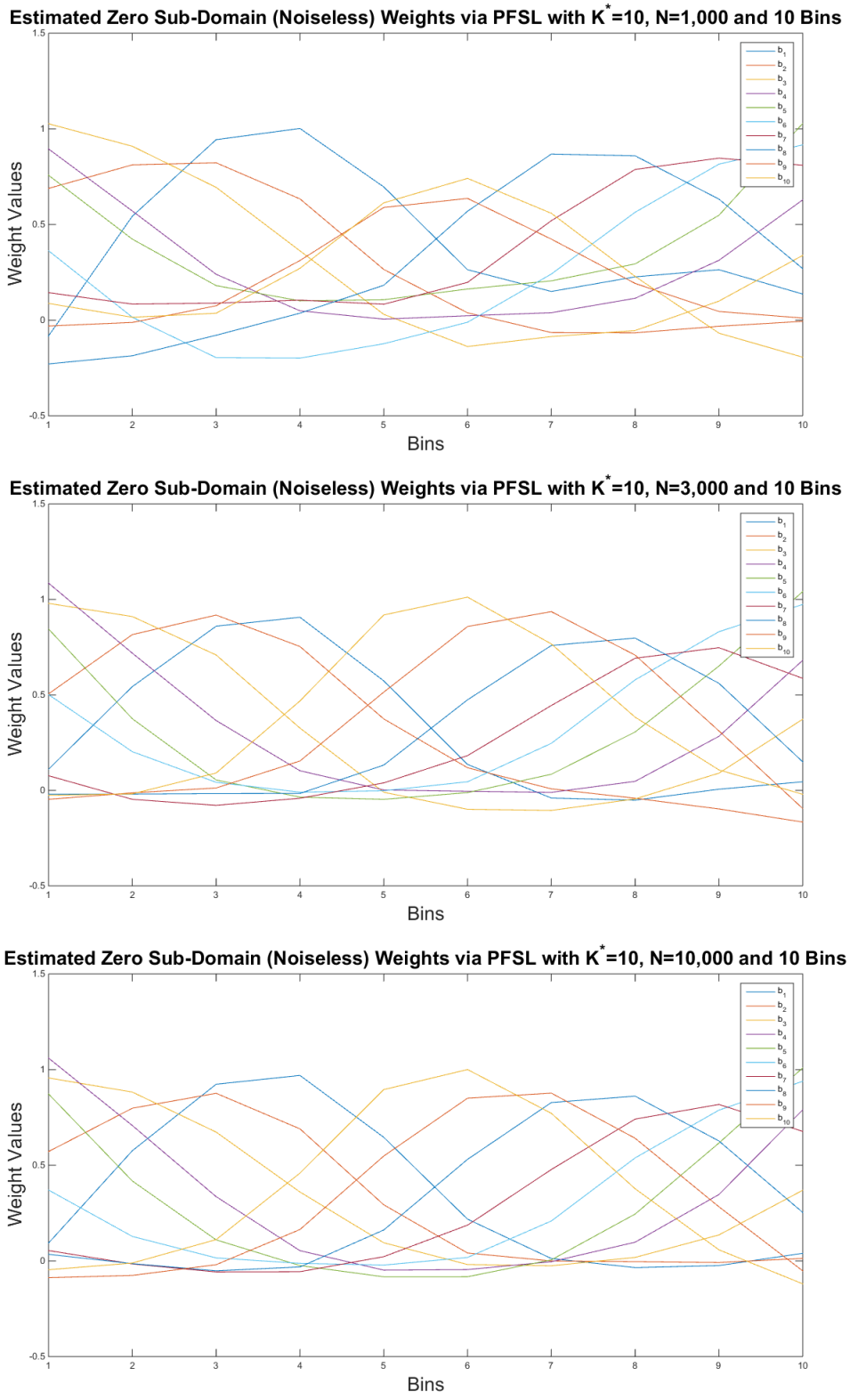

Figure 4.9: Recovered weights in the case of Zero Sub-Domain Noiseless weights using a parameter sensitive classifier and FSA with a lorenz loss (PFSL)

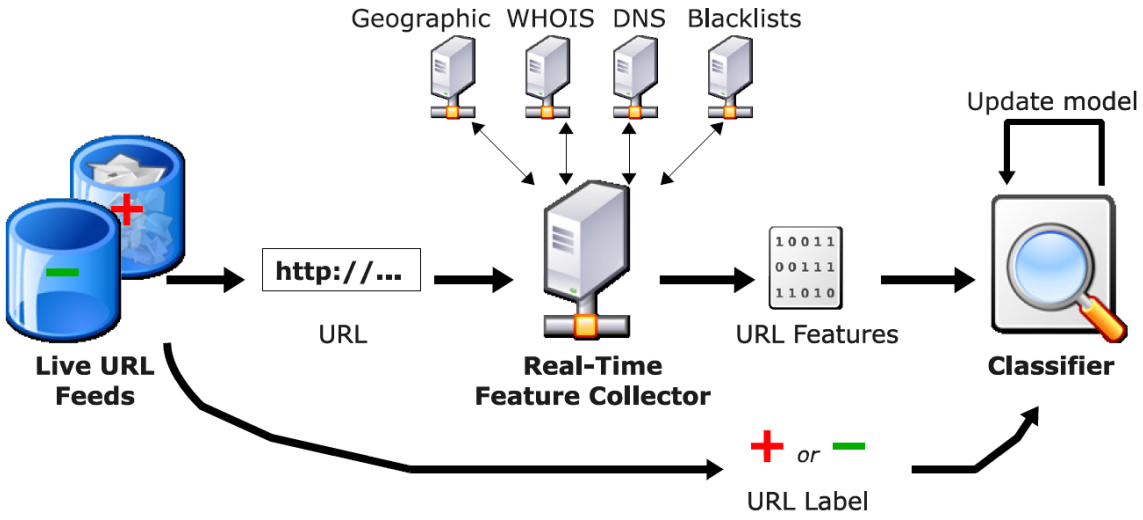

Figure 4.10: Visual of URL Classification Steps in [72]

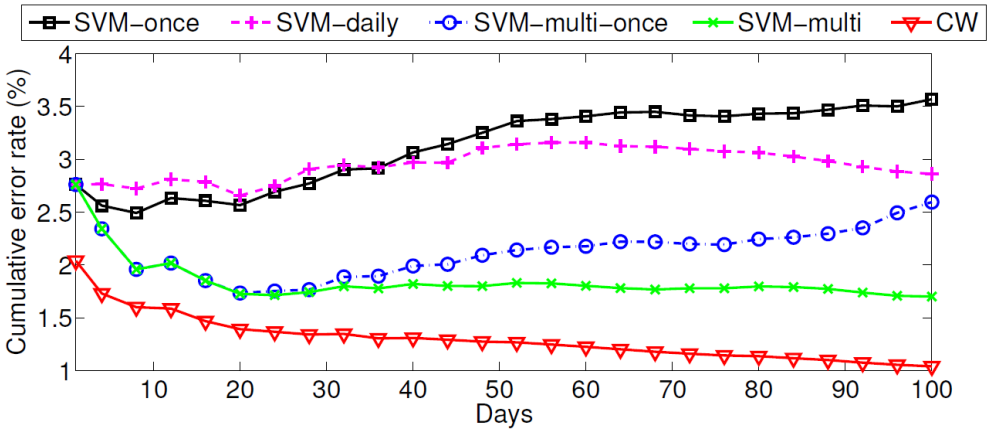

Figure 4.11: Figure from [72] Comparing CW and Batch Algorithms

Figure 4.12: Example of an image and annotation used in FDDB dataset

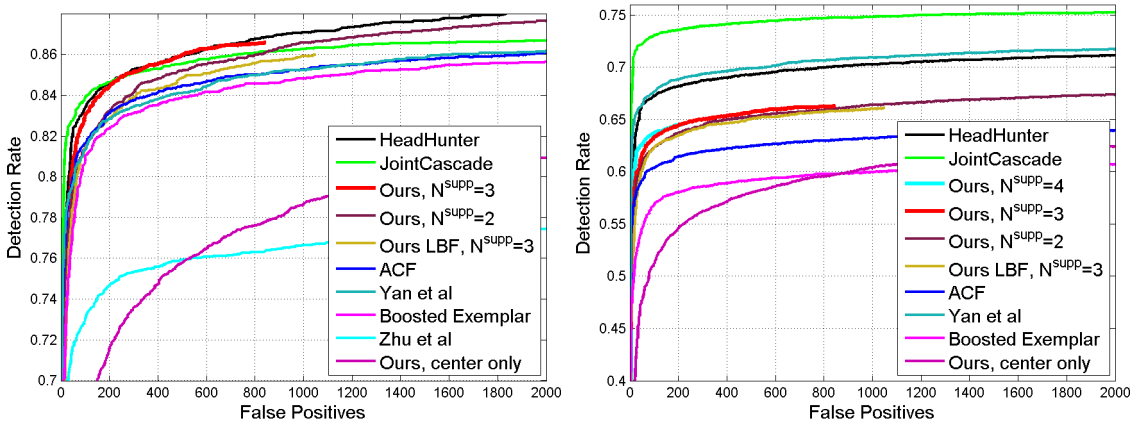

Figure 4.13: Results and comparisons on the FDDB dataset

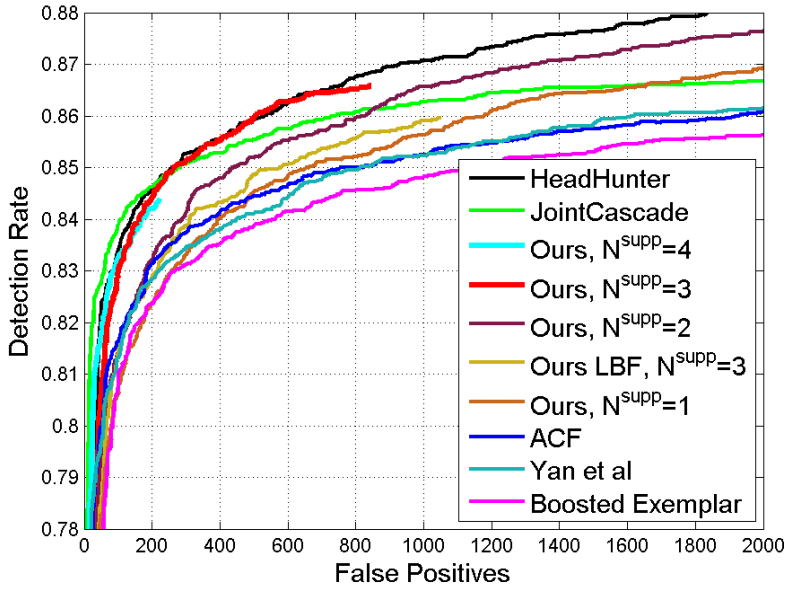

Figure 4.14: Detection results on the FDDB dataset for different values of the support parameter N^{supp}.

Figure 4.15: Example of images used in AFW dataset and presented in [126]

Figure 4.16: Detected faces on the AFW dataset using the 3D model and FSA-SVM keypoint detectors. Also shown are the detected keypoints that were closest to the 3D pose of the detected face. The annotations are shown as thin yellow boxes.

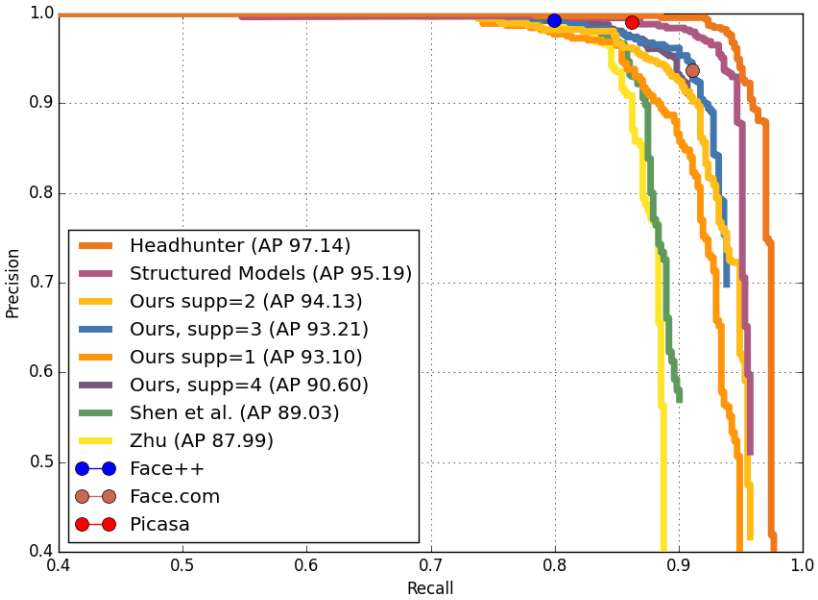

Figure 4.17: Results and comparisons on the AFW dataset (205 images with 486 faces).

APPENDIX A

SUPPORT VECTOR MACHINES

An overview of support vector machines (SVM) is provided. Several papers, textbooks (in particular [50]), and online resources were used in order to develop this section.

SVM generalize the linear decision boundary for classification of the nonseparable case. The SVM produce nonlinear boundaries by constructing a linear boundary in a transformed version of the feature space [107]. That is, they map observations into a high (possibly infinite) dimensional space and construct an optimal hyperplane in this space.

A.1 Constructing the Support Vector Classifier

Suppose our data consists of N pairs $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$ with p predictors, where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. The goal of SVM is to find a hyperplane that correctly separates the training examples into their two classes of -1 and $+1$. We will represent a hyperplane by,

$$\{\boldsymbol{x} : f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\beta} + \beta_0 = 0\} \tag{A.1}$$

where $\boldsymbol{\beta}$ is a unit vector. A property of using the hyperplane representation of Equation (A.1) is that a $f(\boldsymbol{x})$ induces a classification rule (see Equation (A.2)). That is, for an observation $\boldsymbol{x}$, $G(\boldsymbol{x})$ will assign this observation to a class.

$$G(\boldsymbol{x}_i) = \text{sign}[\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0] \tag{A.2}$$

As previously mentioned, the goal is to find the hyperplane that "best" separates the two classes among the data. SVM achieves this best hyperplane by finding the hyperplane the creates the largest margin (from the hyperplane) between both classes. In fact, this is why SVM is sometimes referred to as a Max-Margin method. Figure A.1 demonstrates the ideal case where two classes can be separated completely.

This optimization problem can be represented using Equation (A.3).

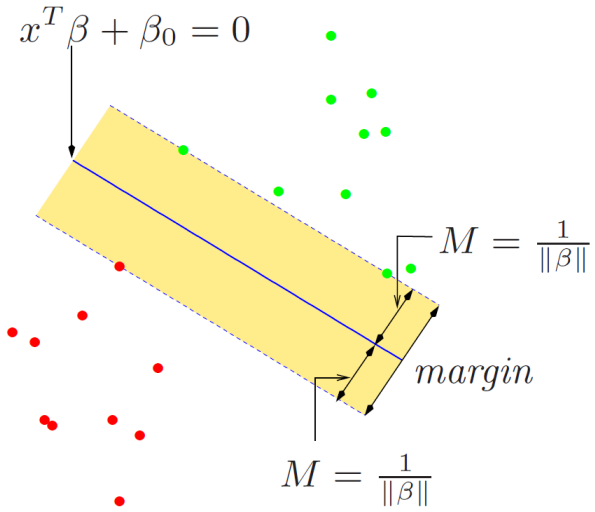

Figure A.1: Figure taken from [50]

$$\begin{aligned} \max_{\boldsymbol{\beta},\beta_0,\|\boldsymbol{\beta}\|=1} \quad & M \\ \text{subject to} \quad & y_i(x_i^T\boldsymbol{\beta} + \beta_0) \geq M \\ & i = 1, \ldots, N. \end{aligned} \tag{A.3}$$

Or more conveniently, we will represent this optimization using Equation (A.4) (let $M = 1/\|\boldsymbol{\beta}\|$).

$$\begin{aligned} \min_{\boldsymbol{\beta},\beta_0} \quad & \frac{1}{2}\|\beta\|^2 \\ \text{subject to} \quad & y_i(x_i^T\boldsymbol{\beta} + \beta_0) \geq 1 \\ & i = 1, \ldots, N. \end{aligned} \tag{A.4}$$

The more interesting and realistic case is when the two classes in question are not separable, as in Figure A.2. Define the slack variables $\xi = (\xi_1, \xi_2, \ldots, \xi_N)$.

- Points labeled ξ^* are on the wrong side of their margin by $\xi^* = M\xi$
- Points on the correct side have $\xi^* = 0$.
- We wish to maximize margin with constraint $\sum \xi \leq C$.
- Thus, $\sum \xi^*$ is the total distance of points on the wrong side.

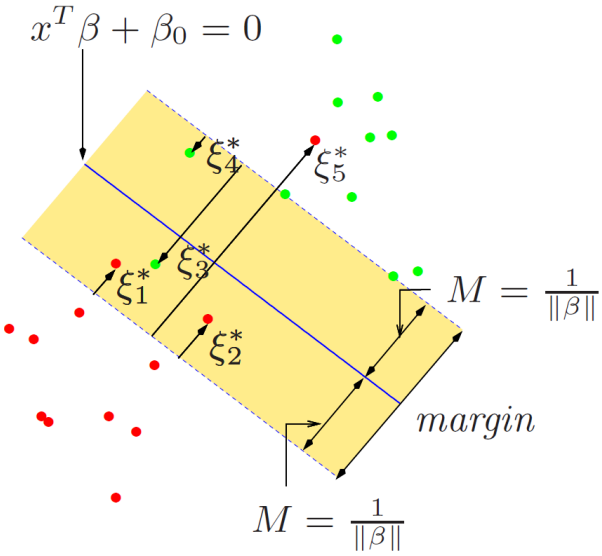

Figure A.2: Figure taken from [50]

There are two ways to modify the constraint in (A.3):

$$y_i(\boldsymbol{x}^T\boldsymbol{\beta} + \beta_0) \geq M - \xi_i \tag{A.5}$$

or

$$y_i(\boldsymbol{x}^T\boldsymbol{\beta} + \beta_0) \geq M(1 - \xi_i) \tag{A.6}$$

$$\forall i, \ , \xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C$$

The first choice measures overlap in actual distance from the decision boundary. However, this choice leads to a nonconvex optimization. The second choice measures overlap in relative distance (changes with M). This choice leads to a convex optimization.

- The value ξ_i is the proportional amount by which the prediction $f(\boldsymbol{x})$ is on the wrong side.
- Misclassification occurs when $\xi_i > 1$. By bounding $\sum \xi_i \leq V$, we limit the number of misclassifications to about V.

As before, we can let $M = 1/\|\boldsymbol{\beta}\|$ and now write (A.4) as

$$\begin{aligned} \min_{\boldsymbol{\beta},\beta_0} \quad & \frac{1}{2}\|\beta\|^2 \\ \text{subject to} \quad & y_i(x_i^T\boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i \quad \forall i \\ & \xi_i \geq 0, \ \sum \xi_i \leq C. \end{aligned} \tag{A.7}$$

Since we will use Lagrange Multipliers to arrive at a solution, we will re-express (A.7) as:

$$\begin{aligned} \min_{\boldsymbol{\beta},\beta_0} \quad & C\sum_{i=1}^{N}\xi_i + \frac{1}{2}\|\beta\|^2 \\ \text{subject to} \quad & \xi_i \geq 0,\ y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0) \geq 1-\xi_i \quad \forall i \end{aligned} \tag{A.8}$$

We now set-up the Lagrange function:

$$L_P(\boldsymbol{\beta},\beta_0,\xi) = C\sum_{i=1}^{N}\xi_i + \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)-(1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i \tag{A.9}$$

We minimize w.r.t. $\boldsymbol{\beta},\beta_0,\xi$ and obtain:

$$\boldsymbol{\beta} = \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i \tag{A.10}$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i \tag{A.11}$$

$$\alpha_i = C - \mu_i, \quad \forall i \tag{A.12}$$

with positivity constraints $\alpha_i,\mu_i,\xi_i \geq 0$. By substituting (A.10)-(A.12) into A.9, this yields the Lagrangian dual function:

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j \tag{A.13}$$

The dual (A.13) gives a lower bound on the objective function (A.8) for any feasible point. We maximize (A.13) subject to $0 \leq \alpha_i \leq C$ and (A.11). This can be achieved using standard software packets. The Karush–Kuhn-Tucker (KKT) conditions generalize the method of Lagrange multipliers, which allows only equality constraints. The KKT conditions for our problem include the constraints:

$$\alpha_i[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)-(1-\xi_i)] = 0 \tag{A.14}$$

$$\mu_i\xi_i = 0 \tag{A.15}$$

$$y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)-(1-\xi_i) \geq 0 \tag{A.16}$$

The partial derivatives we found previously and these KKT conditions uniquely characterize the solution. From (A.10), we see that the solution for $\boldsymbol{\beta}^*$ has the form:

$$\hat{\boldsymbol{\beta}} = \sum_{i=1}^{N}\hat{\alpha}_i y_i \boldsymbol{x}_i \tag{A.17}$$

with nonzero coefficients $\hat{\alpha}_i$ ONLY for observations that satisfy (A.16). These observations are called **support vectors** since $\hat{\boldsymbol{\beta}}$ is represented in terms of them alone. Once Equation (A.13) has been optimized in α, a classification function can be used to classify new observations $\boldsymbol{z}$ from the support vectors $\boldsymbol{x}_i$.

$$f(\boldsymbol{z}) = \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i^T\boldsymbol{z} + \beta_0 \tag{A.18}$$

APPENDIX B

EXPLANATION OF THE GAUSSIAN PYRAMID

We provide a brief tutorial on the use of a Gaussian Pyramid when downsamlping images as well as details on how we use this technique in our work. Since our work is implemented in C++, we relied on OpenCV's implementation of the Gaussian Pyramid as well as their tutorials (found at `http://docs.opencv.org/doc/tutorials/imgproc/pyramids/pyramids.html`) from which this appendix draws heavily from.

B.1 Background

The idea behind the Gaussian Pyramid is that each successive and higher level in the pyramid represents a scaled down version of the original image. That is, at level i of the pyramid, the pixels in this image are "averaged" from neighboring pixels in the image right below it $(i-1)$. Figure B.1 from the tutorial `http://opencv-code.com/tutorials/fast-template-matching-with-image-pyramid/` is provided as an illustration. At higher levels in pyramid, each pixel in the eyes of the baboon, for example, are "averaged" from eye pixels from the most previous level.

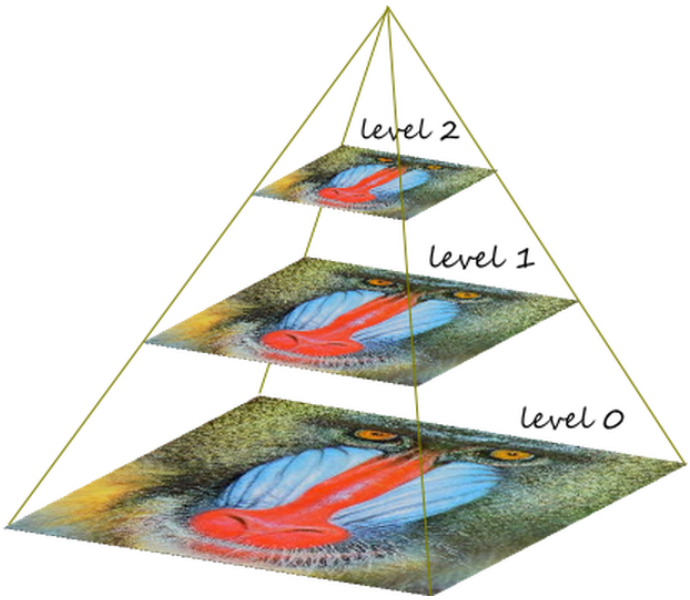

Figure B.1: Illustration of Gaussian Pyramids (taken directly off the web)

What we mean by an average is that each successive level is obtained by convolving the prior level image with a Gaussian Kernel. A typical Gaussian kernel used is the one presented in Figure B.2

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figure B.2

BIBLIOGRAPHY

[1] Karim Ali, François Fleuret, David Hasler, and Pascal Fua. Joint pose estimator and feature learning for object detection. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1373–1380. IEEE, 2009.

[2] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[3] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 1999.

[4] A Barbu, She Y., D. Liangjing, and G. Gramajo. Feature selection with annealing for big data learning. *arXiv preprint arXiv:1310.2880*, 2014.

[5] Adrian Barbu and Gary Gramajo. Face detection using a 3d model on face keypoints. *arXiv preprint arXiv:1404.3596*, 2014.

[6] Peter Bartlett. Statistical learning theory (lectures), 2008.

[7] P.N. Belhumeur, D.W. Jacobs, D.J. Kriegman, and N. Kumar. Localizing parts of faces using a consensus of exemplars. In *CVPR*, pages 545–552, 2011.

[8] DA Belsley, Edwin Kuh, and Roy E Welsch. Regression diagnostics: identifying influential data and sources of collinearity, 1980.

[9] Andrew Blake and Andrew Zisserman. *Visual reconstruction*, volume 2. MIT press Cambridge, 1987.

[10] Avrim L Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.

[11] Paul S Bradley and Olvi L Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pages 82–90, 1998.

[12] PS Bradley, OL Mangasarian, and JB Rosen. Parsimonious least norm approximation. *Computational Optimization and Applications*, 11(1):5–21, 1998.

[13] P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Ann. App. Statistics*, 5(1):232–253, 2011.

[14] Leo Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[15] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] Andrew G Bruce and Hong-Ye Gao. Understanding waveshrink: variance and bias estimation. *Biometrika*, 83(4):727–745, 1996.

[17] Florentina Bunea, Alexandre Tsybakov, Marten Wegkamp, et al. Sparsity oracle inequalities for the lasso. *Electronic Journal of Statistics*, 1:169–194, 2007.

[18] R.L. Burden and J.D. Faires. Numerical analysis. 2001. *Brooks/Cole, USA*.

[19] Xavier P Burgos-Artizzu, Pietro Perona, and Piotr Dollár. Robust face landmark estimation under occlusion. ICCV, 2013.

[20] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. Face alignment by explicit shape regression. In *CVPR*, pages 2887–2894, 2012.

[21] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. Face alignment by explicit shape regression. *International Journal of Computer Vision*, 107(2):177–190, 2014.

[22] H. Cevikalp and B. Triggs. Efficient object detection using cascades of nearest convex model classifiers. In *CVPR*, pages 3138–3145, 2012.

[23] Hakan Cevikalp, Bill Triggs, and Vojtech Franc. Face and landmark detection by using cascade of classifiers. In *ICAFGR*, 2013.

[24] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[25] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1-3):131–159, 2002.

[26] Dong Chen, Shaoqing Ren, Yichen Wei, Xudong Cao, and Jian Sun. Joint cascade face detection and alignment. In *Computer Vision–ECCV 2014*, pages 109–122. Springer, 2014.

[27] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[28] Koby Crammer, Mark Dredze, and Fernando Pereira. Exact convex confidence-weighted learning. In *Advances in Neural Information Processing Systems*, pages 345–352, 2009.

[29] Antonio Criminisi, Jamie Shotton, Duncan Robertson, and Ender Konukoglu. Regression forests for efficient anatomy detection and localization in ct studies. In *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging*, pages 106–117. Springer, 2011.

[30] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[31] Oscar Danielsson and Stefan Carlsson. Projectable classifiers for multi-view object class recognition. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 577–584. IEEE, 2011.

[32] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *CVPR*, pages 2578–2585, 2012.

[33] Daniel F DeMenthon and Larry S Davis. Model-based object pose in 25 lines of code. In *Computer VisionECCV'92*, pages 335–343. Springer, 1992.

[34] Ayhan Demiriz, Kristin P Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[35] Piotr Dollár, Peter Welinder, and Pietro Perona. Cascaded pose regression. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1078–1085. IEEE, 2010.

[36] David L Donoho, Michael Elad, and Vladimir N Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *Information Theory, IEEE Transactions on*, 52(1):6–18, 2006.

[37] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM, 2008.

[38] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[39] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[40] Jianqing Fan, Yang Feng, and Rui Song. Nonparametric independence screening in sparse ultra-high-dimensional additive models. *Journal of the American Statistical Association*, 106(494), 2011.

[41] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

[42] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[43] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[44] Juergen Gall and Victor Lempitsky. Class-specific hough forests for object detection. In *Decision Forests for Computer Vision and Medical Image Analysis*, pages 143–157. Springer, 2013.

[45] Gene H Golub and Charles F van Van Loan. Matrix computations (johns hopkins studies in mathematical sciences). 1996.

[46] Adam J Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pages 692–699, 1998.

[47] Chunhui Gu and Xiaofeng Ren. Discriminative mixture-of-templates for viewpoint classification. In *Computer Vision–ECCV 2010*, pages 408–421. Springer, 2010.

[48] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. 2006.

[49] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[50] Trevor. Hastie, Robert. Tibshirani, and J Jerome H Friedman. *The elements of statistical learning*, volume 1. Springer New York, 2001.

[51] Kun He, Leonid Sigal, and Stan Sclaroff. Parameterizing object detectors in the continuous pose space. In *Computer Vision–ECCV 2014*, pages 450–465. Springer, 2014.

[52] M. Hejrati and D. Ramanan. Analyzing 3d objects in cluttered images. In *NIPS*, pages 602–610, 2012.

[53] Lothar Hermes and Joachim M Buhmann. Feature selection for support vector machines. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 712–715. IEEE, 2000.

[54] G.B. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. 2007.

[55] Jian Huang, Joel L Horowitz, and Fengrong Wei. Variable selection in nonparametric additive models. *Annals of statistics*, 38(4):2282, 2010.

[56] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[57] D. Jiang and J. Huang. Majorization minimization by coordinate descent for concave penalized generalized linear models. *Technical Report*, 2011.

[58] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3:14, 2003.

[59] Keith Knight and Wenjiang Fu. Asymptotics for lasso-type estimators. *Annals of statistics*, pages 1356–1378, 2000.

[60] Martin Koestinger, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011.

[61] K. Koh, S.J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *JMLR*, 8(8):1519–1555, 2007.

[62] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[64] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *JMLR*, 10:777–801, 2009.

[65] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[66] V. Le, J. Brandt, Z. Lin, L. Bourdev, and T.S. Huang. Interactive facial feature localization. In *ECCV*, pages 679–692. 2012.

[67] Haoxiang Li, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Gang Hua. Efficient boosted exemplar-based face detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1843–1850. IEEE, 2014.

[68] Ping Li. Robust logitboost and adaptive base class (abc) logitboost. *arXiv preprint arXiv:1203.3491*, 2012.

[69] Stan Z Li and ZhenQiu Zhang. Floatboost learning and statistical face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1112–1123, 2004.

[70] Joerg Liebelt and Cordelia Schmid. Multi-view object class detection with a 3d geometric model. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1688–1695. IEEE, 2010.

[71] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[72] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 681–688. ACM, 2009.

[73] Jerrold E Marsden and Anthony Tromba. *Vector calculus*. WH freeman, 2003.

[74] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. Face detection without bells and whistles. In *Computer Vision–ECCV 2014*, pages 720–735. Springer, 2014.

[75] Lukas Meier, Sara Van de Geer, Peter Bühlmann, et al. High-dimensional additive modeling. *The Annals of Statistics*, 37(6B):3779–3821, 2009.

[76] Jaime Miranda, Ricardo Montoya, and Richard Weber. Linear penalization support vector machines for feature selection. In *Pattern Recognition and Machine Intelligence*, pages 188–192. Springer, 2005.

[77] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*, volume 821. Wiley, 2006.

[78] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *Computers, IEEE Transactions on*, 100(9):917–922, 1977.

[79] Julia Neumann, Christoph Schnörr, and Gabriele Steidl. Combined svm-based feature selection and classification. *Machine Learning*, 61(1-3):129–150, 2005.

[80] Minh Hoai Nguyen and Fernando De la Torre. Optimal feature selection for support vector machines. *Pattern recognition*, 43(3):584–591, 2010.

[81] Margarita Osadchy, Yann Le Cun, and Matthew L Miller. Synergistic face detection and pose estimation with energy-based models. *The Journal of Machine Learning Research*, 8:1197–1215, 2007.

[82] Mustafa Ozuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):448–461, 2010.

[83] Mustafa Ozuysal, Vincent Lepetit, and Pascal Fua. Pose estimation for category specific multiview object localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 778–785. IEEE, 2009.

[84] Mahesh Pal and Giles M Foody. Feature selection for classification of hyperspectral data by svm. *Geoscience and Remote Sensing, IEEE Transactions on*, 48(5):2297–2307, 2010.

[85] Nadia Payet and Sinisa Todorovic. From contours to 3d object detection and pose estimation. In *ICCV*, pages 983–990, 2011.

[86] Hanchuan Peng, Fulmi Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[87] Bojan Pepik, Michael Stark, Peter Gehler, and Bernt Schiele. Teaching 3d geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3362–3369. IEEE, 2012.

[88] Alain Rakotomamonjy. Variable selection using svm based criteria. *The Journal of Machine Learning Research*, 3:1357–1370, 2003.

[89] Gunnar Rätsch, Takashi Onoda, and Klaus R Müller. Regularizing adaboost. 1999.

[90] Gunnar Rätsch and Manfred K Warmuth. Maximizing the margin with boosting. In *Computational Learning Theory*, pages 334–350, 2002.

[91] Pradeep D Ravikumar, Han Liu, John D Lafferty, and Larry A Wasserman. Spam: Sparse additive models. In *NIPS*, 2007.

[92] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 fps via regressing local binary features. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1685–1692. IEEE, 2014.

[93] Saharon Rosset, Ji Zhu, and Trevor Hastie. Boosting as a regularized path to a maximum margin classifier. *The Journal of Machine Learning Research*, 5:941–973, 2004.

[94] Cynthia Rudin, Ingrid Daubechies, and Robert E Schapire. The dynamics of adaboost: Cyclic behavior and convergence of margins. *The Journal of Machine Learning Research*, 5:1557–1595, 2004.

[95] Cynthia Rudin, Robert E Schapire, and Ingrid Daubechies. Boosting based on a smooth margin. In *Learning theory*, pages 502–517. Springer, 2004.

[96] Stefan Schaal and Christopher G Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

[97] R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[98] Yiyuan She. An iterative algorithm for fitting nonconvex penalized generalized linear models with grouped predictors. *Computational Statistics & Data Analysis*, 56(10):2976–2990, 2012.

[99] Yiyuan She et al. Thresholding-based iterative selection procedures for model selection and shrinkage. *Electronic Journal of Statistics*, 3:384–415, 2009.

[100] Xiaohui Shen, Zhe Lin, Jonathan Brandt, and Ying Wu. Detecting and aligning faces by image retrieval. In *CVPR*, pages 3460–3467, 2013.

[101] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[102] Hao Su, Min Sun, Li Fei-Fei, and Silvio Savarese. Learning a dense multi-view representation for detection, viewpoint classification and synthesis of object categories. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 213–220. IEEE, 2009.

[103] Min Sun, Hao Su, Silvio Savarese, and Li Fei-Fei. A multi-view probabilistic model for 3d object classes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1247–1254. IEEE, 2009.

[104] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3476–3483. IEEE, 2013.

[105] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of the AFNLP/ACL*, pages 477–485, 2009.

[106] Vladimir Vapnik and Olivier Chapelle. Bounds on error expectation for support vector machines. *Neural computation*, 12(9):2013–2036, 2000.

[107] Vladimir N Vapnik. Statistical learning theory. 1998.

[108] Sethu Vijayakumar, Aaron D'souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural computation*, 17(12):2602–2634, 2005.

[109] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[110] Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero norm with linear models and kernel methods. *The Journal of Machine Learning Research*, 3:1439–1461, 2003.

[111] Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, and Vladimir Vapnik. Feature selection for svms. In *NIPS*, volume 12, pages 668–674, 2000.

[112] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *JMLR*, 11:2543–2596, 2010.

[113] Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 532–539. IEEE, 2013.

[114] Junjie Yan, Zhen Lei, Longyin Wen, and Stan Z Li. The fastest deformable part model for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2497–2504. IEEE, 2014.

[115] Junjie Yan, Xuzong Zhang, Zhen Lei, and Stan Z Li. Face detection by structural models. *Image and Vision Computing*, 32(10):790–799, 2014.

[116] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014.

[117] Bin Yu and Baozong Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, 26(6):883–889, 1993.

[118] Quan Yuan, Ashwin Thangali, Vitaly Ablavsky, and Stan Sclaroff. Parameter sensitive detectors. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. IEEE, 2007.

[119] Quan Yuan, Ashwin Thangali, Vitaly Ablavsky, and Stan Sclaroff. Learning a family of detectors via multiplicative kernels. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(3):514–530, 2011.

[120] C.H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Ann. of Statistics*, 38(2):894–942, 2010.

[121] Peng Zhao and Bin Yu. On model selection consistency of lasso. *The Journal of Machine Learning Research*, 7:2541–2563, 2006.

[122] Jun Zheng, Furao Shen, Hongjun Fan, and Jinxi Zhao. An online incremental learning support vector machine for large-scale data. *Neural Computing and Applications*, 22(5):1023–1035, 2013.

[123] Shaohua Kevin Zhou and Dorin Comaniciu. Shape regression machine. In *Information Processing in Medical Imaging*, pages 13–25. Springer, 2007.

[124] Yingbo Zhou, Utkarsh Porwal, Ce Zhang, Hung Q Ngo, Long Nguyen, Christopher Ré, and Venu Govindaraju. Parallel feature selection inspired by group testing. In *Advances in Neural Information Processing Systems*, pages 3554–3562, 2014.

[125] Ji Zhu, Saharon Rosset, Trevor Hastie, and Robert Tibshirani. 1-norm support vector machines. In *NIPS*, volume 15, pages 49–56, 2003.

[126] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012.

BIOGRAPHICAL SKETCH

After finishing high school in Florida, the author completed a Bachelor of Science degree at Florida State University in mathematics. Right after, the author enrolled in the statistics graduate program at FSU to pursue a PhD. The author would have never made it this far without his Lord and Savior Jesus Christ.