Constructing Orthogonal Latent Features for Arbitrary Loss

Michinari Momma[1] and Kristin P. Bennett[2]

[1] Fair Isaac Corporation, San Diego, CA 92130 USA mommam@alum.rpi.edu
[2] Department of Mathematical Sciences , Rensselaer Polytechnic Institute, Troy, NY 12180 USA bennek@rpi.edu

Summary. A boosting framework for constructing orthogonal features targeted to a given loss function is developed. Combined with techniques from spectral methods such as PCA and PLS, an orthogonal boosting algorithm for linear hypothesis is used to efficiently construct orthogonal latent features selected to optimize the given loss function. The method is generalized to construct orthogonal nonlinear features using the kernel trick. The resulting method, Boosted Latent Features (BLF) is demonstrated to both construct valuable orthogonal features and to be a competitive inference method for a variety of loss functions. For the least squared loss, BLF reduces to the PLS algorithm and preserves all the attractive properties of that algorithm. As in PCA and PLS, the resulting nonlinear features are valuable for visualization, dimensionality reduction, improving generalization by regularization, and use in other learning algorithms, but now these features can be targeted to a specific inference task/loss function. The data matrix is factorized by the extracted features. The low-rank approximation of the data matrix provides efficiency and stability in computation, an attractive characteristic of PLS-type methods. Computational results demonstrate the effectiveness of the approach on a wide range of classification and regression problems.

1 Introduction

We consider the problem of feature construction targeted towards a given inference task. The class of features considered are linear combinations of the input attributes. The quintessential unsupervised feature extraction method for such linear features is principal component analysis (PCA). PCA constructs orthogonal features consisting of linear combinations of the input vectors called principal components. In PCA, the principal components are chosen to maximize the explained variance and to be orthogonal to each other. The resulting principal components are useful for visualization, understanding importance of variables, and as a form of dimensionality reduction or regularization when used in inference functions, e.g principal component regression. PCA takes into account only the input variables and does not use the response variables. While PCA is widely used, it is not appropriate for every task. The approach is entirely based on the least squares loss, which may not be

appropriate when the underlying noise is non-Gaussian. PCA may not produce good features or may require more features for supervised learning tasks such as classification, regression, and ranking.

Our goal is to construct orthogonal features that are targeted toward a given inference task. The features are assumed to be linear combinations of the input data that are orthogonal to each other. The features constructed from factorizations of the data and hypothesis space are targeted toward a specific inference task as defined by the loss function. We seek a small set of features that span the portion of the hypothesis space of interest for the specific inference task. The features act as a form of regularization: the loss function is minimized with respect to the smaller set of features instead of in the original typically much higher dimensional input space. This relatively small set of features can then be used for many purposes such as predictive models, visualization, and outlier detection.

The primary motivation for this approach comes from boosting which can be viewed as a method for constructing features as well as predictive models. Ensemble methods such as AdaBoost [10] and Gradient Boost [12] construct a linear combination of hypotheses in order to optimize a specified loss function. But the resulting hypotheses do not meet our goals. Ensemble methods typically use many weak hypotheses. But our goal is to create a few orthogonal features that span the space of interest. The key missing property in ensemble methods, for the purpose of reducing dimensionality of the feature space, is orthogonality. Forcing orthogonality can dramatically increase the convergence speed of the boosting algorithm. Thus much fewer features need to be constructed to obtain the same decrease in loss function. By simply adding orthogonality to the ensemble method, the desirable properties of PCA are once again regained – a small set of orthogonal features are identified that explains properties of the data of interest; but now the definition of interest can go beyond explaining variance.

Thus, we propose a boosting method with orthogonal components as its weak hypotheses. Boosting has been shown to be equivalent to gradient descent projected into a hypothesis space. At each iteration, the hypothesis or feature is constructed to maximize the inner product with the gradient. In orthogonal boosting, we construct the feature that is orthogonal to all previous features that maximizes this inner product. For linear hypotheses, this constrained optimization problem can be very efficiently and exactly solved by projecting the data into the orthogonal subspace.

Orthogonal boosting of linear hypotheses is very closely related to spectral or data factorization methods such as PCA, canonical correlation analysis, factor analysis, and partial least squares regression (PLS) analysis [17, 29]. For least squares regression, orthogonal boosting with linear regression of linear hypotheses reduces exactly to PLS. The proposed approach generalizes PLS to an arbitrary loss function. In PLS, the linear hypotheses are called latent variables or factors. Thus we call boosting of such linear hypotheses, Boosted Latent Factors (BLF).

Many papers have been written on PLS and its properties from many different perspectives. The boosting perspective given here is novel but our goal is not to give another perspective on PLS. Our goal is to create an efficient orthogonal feature construction method that maintains the benefits of PLS. BLF and PLS both construct a set of orthogonal linear factors that form a factorization of the input and response variables. The BLF algorithm shares with PLS efficiency and elegance for computing inference functions. PLS has been shown to be a conjugate gradient method applied to the least square regression. BLF is also a subspace method, closely

related to but not identical to nonlinear conjugate gradient algorithms. Previously CGBOOST used a nonlinear conjugate gradient algorithm in function space but it did not produce orthogonal hypotheses and computational results were mixed due to overfitting [19]. Like PLS, BLF is a form of regularization of the loss function that reduces the norm of the solution as compared to the unconstrained optimal solution, alleviating overfitting. An extensive discussion of regularization in PLS and alternative gradient based regularization strategies can be found in [13].

A major benefit of BLF is that it can be elegantly and efficiently extended to create nonlinear latent hypotheses and functions through the use of kernels. Using an approach very similar to that of kernel partial least squares [26], data can be mapped into a feature space and the orthogonal features are constructed in the feature space. Kernel methods exist for construction of orthogonal nonlinear feature such as kernel principal component analysis, kernel partial least squares, and kernel canonical correlation analysis, but they are all based on the least squares loss [28]. See [5] for discussions of their common ancestry. Neural network approaches have been used to generalize PLS to nonlinear functions (e.g. [20]). But these methods face the problems of neural networks: inefficiency, local minima, limited loss functions, and lack of generality that are largely eliminated in kernel methods. Iterative reweighted partial least squares generalized linear PLS to exponential loss functions [21] using a generalized linear model approach, but this approach is more computationally costly, less general, and theoretically somewhat more obtuse than BLF.

BLF is efficient and stable computationally for kernel logistic regression (KLR). KLR can be solved by a boosting-type method [30] with the regularization term explicitly included in its objective function but without orthogonalization. An open question is if there are additional advantages to orthogonalizing weak hypotheses in terms of generalization. Orthogonal least squares regression and its variants [7, 23] utilize a similar approach. They orthogonalize weak hypotheses each consisting of a *single* support vector, not a linear combination of support vectors. The BLF can also be applied to extend their methodologies to other loss functions.

This chapter is organized as follows. We review AnyBoost [22] to provide a general framework for ensemble methods for differentiable loss functions, and use this to introduce the orthogonalized version of AnyBoost, OrthoAnyBoost. Section 3 shows how OrthoAnyBoost can be efficiently implemented in linear hypothesis spaces forming the Boosted Latent Factor Algorithm. Section 4 examines the convergence properties of BLF. In Section 5, PLS is shown to be the special case within the BLF framework of the squared loss functions with linear weak hypotheses. Section 6 provides variants of BLF for three more loss functions. The kernel version of BLF is developed in Section 7. Computational results found in Section 8 illustrate the potential of kernel BLF for feature construction. In addition, standard benchmarks demonstrate the approach is quite competitive with existing classification and regression algorithms.

The following explains the notation used throughout the paper. Assume we are given a training data set of size m with a single response, $((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$, with the column vectors $\mathbf{x}_i \in R^n$ and $y_i \in R$. Although PLS and KPLS can be used for multivariate response, we focus on cases with univariate response variables. $\boldsymbol{A}^T$ denotes a vector/matrix transpose. The data matrix is $\boldsymbol{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m]^T$ and the response vector is $\mathbf{y} = [y_1, \ldots, y_m]^T$. After centering each of the predictive variables, the centered data matrix is denoted by either $\widetilde{\boldsymbol{X}}$ or $\boldsymbol{X}^1$. $||\boldsymbol{y}||$ denotes the 2-norm of $\boldsymbol{y}$. The dot product of two column vectors $\mathbf{u}$ and $\mathbf{v}$ is denoted by $\boldsymbol{u}^T\boldsymbol{v}$. The outer

product of $\mathbf{u}$ and $\mathbf{v}$ is denoted by $\boldsymbol{uv}^T$. The reader should be careful to distinguish the use of dot products from that of outer products. Iteration indices are expressed as superscripts. Subscripts indicate components of a matrix or vector. In a slight abuse of notation, we use $\nabla l(\mathbf{y}, \mathbf{f})$ to denote the gradient or the subgradient of the function l with respect to $\mathbf{f}$ if the gradient does not exist.

2 General Framework in BLF

This section investigates a boosting algorithm that combines properties of the orthogonal components of PCA and PLS with the flexibility of general ensemble and boosting methods with respect to a loss function. The goal is to create a set of orthogonal features or functions that explain the response according to some loss function. We adapt notation of the general ensemble method, AnyBoost [22].

2.1 AnyBoost

Boosting algorithms construct a set of features called hypotheses that try to span the response e.g. $\mathbf{y} \approx \boldsymbol{T}\mathbf{c}$ for regression or $\mathbf{y}^T\boldsymbol{T}\mathbf{c} > 0$ for classification. AnyBoost (Algorithm 1) [22] constructs an optimal linear combination of hypotheses to fit the response by performing gradient descent in function or hypothesis space. Let $\mathcal{T}$ be a set of real-valued functions that are the hypotheses. The span of $\mathcal{T}$ forms a linear function space. The inner product in this function space is defined as $\mathbf{t}^T\mathbf{f} \equiv \sum_{i=1}^m t(x_i)f(x_i)$. Thus we can also think of $\mathbf{t}$ as an m-vector with $t_i = t(x_i)$. A linear combination of the hypotheses, $\sum_{i=1}^N c^i\mathbf{t}^i$ for $\mathbf{t}^1, \mathbf{t}^2, \ldots, \mathbf{t}^N$, can be written as $\boldsymbol{T}\mathbf{c}$ where the columns of $\boldsymbol{T}$ are the hypotheses and the vector $\mathbf{c}$ contains their coefficients. We want to find the element, $\mathbf{t} \in \text{span}(\mathcal{T})$ that approximately minimizes some loss function $l(\mathbf{y}, \mathbf{f})$. AnyBoost accomplishes this by doing gradient descent in the hypothesis space. Note $\nabla l(\mathbf{y}, \boldsymbol{T}\mathbf{c}) = \nabla_{\mathbf{f}} l(\mathbf{y}, \boldsymbol{T}\mathbf{c})$ will be used to denote the gradient of the loss function in the function space. Ideally $\mathcal{T}$ spans the same space as $\mathbf{y}$ but this is generally not the case. Thus the linear functional is minimized with respect to the loss function to fit the response.

Given the current function $\mathbf{f} = \boldsymbol{T}\mathbf{c}$, a descent direction in the hypothesis space, $\mathbf{t}^{i+1}$ is constructed. Any function with a positive inner product with the negative gradient $(-\nabla l(\mathbf{y}, \boldsymbol{T}\mathbf{c})^T\mathbf{t}^{i+1})$ must be a descent direction. In AnyBoost, a weak learning algorithm is used to construct a hypothesis that approximately maximizes the inner product of the hypothesis with the negative gradient at each iteration. The hypothesis is added with an appropriate stepsize c^{i+1} to the current function to form $\boldsymbol{T}\mathbf{c} + c^{i+1}\mathbf{t}^{i+1}$. The algorithm terminates if the weak learner fails to produce a weak hypothesis that is a descent direction indicating convergence, or if it reaches the maximum number of iterations.

Algorithm 1 (AnyBoost) *Given: class of weak hypotheses* $\mathcal{T}$, $l(\mathbf{y}, \mathbf{t})$ *with gradient* $\nabla l(\mathbf{y}, \mathbf{t})$, *weak learner that finds* $\mathbf{t} \in \mathcal{T}$ *maximizing* $\mathbf{u}^T\mathbf{t}$*:*

1. *Let* $\mathbf{f} =$ *constant or null hypothesis*
2. *Compute* $\mathbf{u}^1 = -\nabla l(\mathbf{y}, \mathbf{f})$
3. *For* $i = 1$ *to* N

4. *Let* $\mathbf{t}^i \in \arg\max_{\mathbf{t}\in\mathcal{T}} {\mathbf{u}^i}^T\mathbf{t}$
5. *if* $-{\mathbf{u}^i}^T\mathbf{t}^i < 0$ *then return* $\mathbf{f}$
6. *Choose* c^i *to reduce* $l(\mathbf{y}, \mathbf{f} + c^i\mathbf{t}^i)$
7. *Let* $\mathbf{f} = \sum_{j=1}^{i} \mathbf{t}^j c^j$
8. *Compute* $\mathbf{u}^{i+1} = -\nabla l(\mathbf{y}, \mathbf{f})$
9. *end for*
10. *return* $\mathbf{f}$

Variations can be developed by specifying the weak learner in Step 4, the loss function, and the algorithm to optimize the step size c^i (Step 6). For example, in [22], it is shown that AdaBoost [10] minimizes the exponential loss with exact line search, ConfidenceBoost [27] minimizes the exponential loss with an inexact line search, and LogitBoost [11] minimizes the negative binomial log-likelihood with a single Newton step used to compute c. Cost sensitive decision tree algorithms are commonly used as the weak learner.

2.2 AnyBoost with Orthogonal Weak Hypotheses

Our goal is to create a set of orthogonal features that explains the response, i.e. for regression $\mathbf{y} \approx \boldsymbol{T}\mathbf{c} = \boldsymbol{X}\boldsymbol{V}\mathbf{c}$, $\boldsymbol{T}^T\boldsymbol{T} = \boldsymbol{I}$, where $\boldsymbol{V}$ is a projection matrix with $\mathbf{v}^i$, $i = 1, \ldots, N$ as its column vectors, in such a way that a given loss function is minimized. By changing AnyBoost to produce orthogonal hypotheses, we can force AnyBoost to produce features that both factorize and explain the input space like PCA and span the response or output space as in original AnyBoost. The resulting algorithm becomes:

Algorithm 2 (OrthoAnyBoost) *Same as Algorithm 1 except for the following steps:*

4. Let $\mathbf{t}^i \in \begin{array}{l}\arg\max_{\mathbf{t}\in\mathcal{T}} {\mathbf{u}^i}^T\mathbf{t} \\ \textit{subject to} \quad \mathbf{t}^T\mathbf{t}^j = 0 \quad j = 1, \ldots, i-1.\end{array}$
6. Optimize $\mathbf{c}$ *to reduce* $l(\mathbf{y}, \sum_{j=1}^{i} \mathbf{t}^j c^j)$

This simple change has some immediate ramifications. We may now use more powerful hypotheses since the hypothesis subproblem is now regularized. Second, the approach is no longer a gradient descent algorithm. We have transformed it into a subspace algorithm [24] similar but not equivalent to a nonlinear conjugate gradient algorithm. At each iteration, the algorithm computes the optimal solution over the current subspace. This is a form of regularization of the original problem. Subspace methods such as conjugate gradient can be much faster than gradient methods particularly for ill-posed problems such as the ones that we are considering. Empirically BLF converges much faster than boosting without orthgonalization. In fact for linear hypotheses, we can prove finite termination of BLF at an optimal solution of the full problem. For the least squares loss with linear hypotheses, BLF reduces to the conjugate gradient method. We will not show this directly but instead show that the method reduces to partial least squares which has been previously shown to be a conjugate gradient algorithm [25]. Note for finite termination, the regression coefficients $\mathbf{c}$ must be reoptimized (refitted) at every iteration, an approach not typical of most boosting or ensemble algorithms.

3 BLF with Linear Functions

The OrthoAnyBoost framework can be applied to linear hypotheses. Linear hypotheses are of the form $t_k = \mathbf{x}_k^T \mathbf{v}$ in input space or $t_k = \phi(\mathbf{x}_k)^T \mathbf{v}$ in feature space for use in a kernel framework. We reserve discussion of the natural extension to kernels to Section 7. To apply OrthoAnyBoost to linear hypotheses requires that the weak learner (Step 4) and the hypotheses weighting computation (Step 6) be modified accordingly. The big advantage of linear hypotheses is that the optimal orthogonal linear function found in Step 4 can be efficiently computed in closed form by recasting the problem into the null space of the constraints using a widely-used procedure commonly known as "deflation." For clarity we first provide a detailed description of deflation since it has proven a source of confusion and an understanding of deflation is necessary for all other parts of the algorithm.

3.1 Enforcing Orthogonality in Linear Spaces

The requirements that the new hypotheses be orthogonal to all previous constraints form linear equality constraints in hypothesis space. For linear hypotheses, $\mathbf{t} = \boldsymbol{X}\mathbf{v}$, Step 4 in OrthoAnyboost reduces to

$$\begin{array}{ll} \max_{\mathbf{v}} & \mathbf{u}^{i\,T} \boldsymbol{X}\mathbf{v} \\ \textit{subject to} & \mathbf{t}^{j\,T} \boldsymbol{X}\mathbf{v} = 0 \quad j = 1, \ldots, i-1 \\ & \boldsymbol{X}\mathbf{v} \in \mathcal{T} \end{array} \tag{1}$$

The linear equalities can be eliminated by mapping the problem into the null space of the linear constraints, as in power methods for computing principal components. This also provides a way to naturally bound the hypothesis space $\mathcal{T}$ so that Problem (1) has a solution. Define $\boldsymbol{T}^i = [\mathbf{t}^1, \ldots, \mathbf{t}^i]$. Let $\boldsymbol{Z}^i$ be any null space matrix for the matrix $\boldsymbol{T}^{i\,T}\boldsymbol{X}$. Then for any $\mathbf{w}$, $\mathbf{v}^i = \boldsymbol{Z}^{i-1}\mathbf{w}^i$ implies $\boldsymbol{T}^{i\,T}\boldsymbol{X}\mathbf{v}^i = 0$. Note when $i = 1$, $\boldsymbol{Z}^0$ is defined to be $\boldsymbol{I}$ as a special case and $\boldsymbol{X}^1 = \boldsymbol{X}$. Problem (1) can be reparameterized as

$$\begin{array}{ll} \max_{\mathbf{w}} & \mathbf{u}^{i\,T} \boldsymbol{X}\boldsymbol{Z}^{i-1}\mathbf{w} \\ \textit{subject to} & \mathbf{w}^T\mathbf{w} = 1 \end{array} \tag{2}$$

Note that the constraint $\mathbf{w}^T\mathbf{w} = 1$ must be added to insure that the problem is bounded and that an optimal solution always exists. Problem (2) has a closed form solution $\mathbf{w}^i = \frac{\boldsymbol{Z}^{i-1\,T}\boldsymbol{X}^T\mathbf{u}^i}{\|\boldsymbol{Z}^{i-1\,T}\boldsymbol{X}^T\mathbf{u}^i\|}$.

An equivalent way to solve Problem (2) is to map the data into the null space $\boldsymbol{X}^i = \boldsymbol{X}\boldsymbol{Z}^{i-1}$ by "deflating" or orthogonalizing the data with respect to the constraints. At the i^{th} iteration, the linear hypothesis $\mathbf{t}^i = \frac{\boldsymbol{X}^i\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}$ is chosen. The deflated data matrix is $\boldsymbol{X}^{i+1} = (\boldsymbol{I} - \mathbf{t}^i\mathbf{t}^{i\,T})\boldsymbol{X}^i$.

While not obvious, deflating $\boldsymbol{X}$ in this way is equivalent to multiplying $\boldsymbol{X}$ by the appropriate null space matrix. At the i^{th} iteration, Problem (2) is equal to

$$\begin{array}{ll} \max_{\mathbf{w}} & \mathbf{u}^{i\,T} \boldsymbol{X}^i\mathbf{w} \\ \textit{subject to} & \mathbf{w}^T\mathbf{w} = 1. \end{array} \tag{3}$$

A major advantage of this form of the problem is that the optimal solution has a closed form $\mathbf{w}^i \propto \boldsymbol{X}^{i^T}\mathbf{u}^i$. The optimal solution in the transformed space is $\mathbf{w}^i$. The optimal solution is the original space is $\mathbf{v}^i = \boldsymbol{Z}^{i-1}\mathbf{w}^i$.

The fact that orthogonality is enforced, i.e. $\boldsymbol{T}^{i^T}\boldsymbol{X}^{i+i} = 0$, can be proved by induction. Consider the second iteration, where $\mathbf{t}^1 = \frac{\boldsymbol{X}\mathbf{w}^1}{\|\boldsymbol{X}\mathbf{w}^1\|}$

$$\boldsymbol{X}^2 = (\boldsymbol{I} - \mathbf{t}^1\mathbf{t}^{1^T})\boldsymbol{X} = \boldsymbol{X} - \tfrac{\boldsymbol{X}\mathbf{w}^1}{\|\boldsymbol{X}\mathbf{w}^1\|}\mathbf{t}^{1^T}\boldsymbol{X} = \boldsymbol{X}(\boldsymbol{I} - \tfrac{\mathbf{w}^1}{\|\boldsymbol{X}\mathbf{w}^1\|}\mathbf{t}^{1^T}\boldsymbol{X})$$

Define $\boldsymbol{Z}^1 = (\boldsymbol{I} - \frac{\mathbf{w}^1}{\|\boldsymbol{X}\mathbf{w}^i\|}\mathbf{t}^{1^T}\boldsymbol{X})$, then $\mathbf{t}^{1^T}\boldsymbol{X}^2 = \mathbf{t}^{1^T}\boldsymbol{X}\boldsymbol{Z}^1 = \mathbf{t}^{1^T}\boldsymbol{X} - \mathbf{t}^{1^T}\frac{\boldsymbol{X}\mathbf{w}^1}{\|\boldsymbol{X}\mathbf{w}^1\|}\mathbf{t}^{1^T}\boldsymbol{X} = 0$. Thus the optimal solution of Problem (3) for the second iteration (the first has no linear constraints) will satisfy the orthogonality constraints. For the i^{th} iteration assume $\boldsymbol{X}^i = \boldsymbol{X}\boldsymbol{Z}^{i-1}$ and $\mathbf{t}^{j^T}\boldsymbol{X}\boldsymbol{Z}^{i-1} = 0 \;\; j = 1, \ldots, i-1$. At iteration $i+1$,

$$\boldsymbol{X}^{i+1} = (\boldsymbol{I} - \mathbf{t}^i\mathbf{t}^{i^T})\boldsymbol{X}^i = \boldsymbol{X}^i - \tfrac{\boldsymbol{X}^i\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}\mathbf{t}^{i^T}\boldsymbol{X}^i = \boldsymbol{X}\boldsymbol{Z}^{i-1}(\boldsymbol{I} - \tfrac{\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}\mathbf{t}^{i^T}\boldsymbol{X}^i)$$

By the assumption, $\mathbf{t}^{j^T}\boldsymbol{X}^{i+1} = \mathbf{t}^{j^T}\boldsymbol{X}\boldsymbol{Z}^{i-1}(\boldsymbol{I} - \frac{\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}\mathbf{t}^{i^T}\boldsymbol{X}^i) = 0, \; j = 1, \ldots, i-1$, thus we need only worry about $\mathbf{t}^i = \frac{\boldsymbol{X}^i\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}$. Since

$$\mathbf{t}^{i^T}\boldsymbol{X}^{i+1} = \mathbf{t}^{i^T}\boldsymbol{X}^i(\boldsymbol{I} - \frac{\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}\mathbf{t}^{i^T}\boldsymbol{X}^i) = \mathbf{t}^{i^T}\boldsymbol{X}^i - \mathbf{t}^{i^T}\frac{\boldsymbol{X}^i\mathbf{w}^i}{\|\boldsymbol{X}^i\mathbf{w}^i\|}\mathbf{t}^{i^T}\boldsymbol{X}^i = 0.$$

Thus $\mathbf{v}^i = \boldsymbol{Z}^{i-1}\mathbf{w}^i$ satisfies the orthogonality constraints of Problems (1) and (2). Note that the optimal $\mathbf{w}^i$ at each iteration is in the deflated space. Steps at the end of BLF map the solution back to the original data space. As shown in the next section, this can be done efficiently using the stored vectors $\mathbf{w}^i$ and $\mathbf{p}^i = \boldsymbol{X}^{i^T}\mathbf{t}^i$ without explicitly deflating the data.

3.2 Boosting Latent Factors

Algorithm 3 provides the full algorithm for boosting orthogonal linear hypotheses or latent factors. To start the algorithm, the initial or "zeroth" weak hypothesis is taken to be a constant hypothesis $\mathbf{t} \propto \mathbf{e}$, a vector of ones. The step size on the constant column is obtained by solving $\min_c l(\mathbf{y}, c\mathbf{e})$, For example for the least squares loss function, $c = \mu_y = mean(\mathbf{y})$. The first hypothesis is then $f = c\mathbf{e}$. The data is deflated by the scaled $\mathbf{t}^0 = \frac{\mathbf{e}}{\|\mathbf{e}\|}$ so $\boldsymbol{X}^1 = \boldsymbol{X} - \mathbf{t}^0\mathbf{t}^{0^T}\boldsymbol{X} = \boldsymbol{X} - \mathbf{e}\boldsymbol{\mu}_X^T$ where $\boldsymbol{\mu}_X$ is the mean of $\boldsymbol{X}$. Thus deflation centers each attribute. Frequently centering is done as preprocessing, thus we treat it as a distinct first step.

Algorithm 3 (Boosted Latent Factors (BLF)) *Input: data* $\boldsymbol{X}$*; response* $\mathbf{y}$*; number of latent variables* N*.*

1. *Compute* $\mu_y = \arg\min_{\mu_y} l(\mathbf{y}, \mu_y\mathbf{e})$*. Set* $\boldsymbol{\mu}_X = \frac{1}{m}\boldsymbol{X}^T\mathbf{e}$*. Deflate* $\boldsymbol{X}^1 = \boldsymbol{X} - \mathbf{e}\boldsymbol{\mu}_X^T$*.* $\mathbf{u}^1 = -\nabla l(\mathbf{y}, \mu_y\mathbf{e})$*,* $\boldsymbol{T} = [\,]$*.*
2. *For* $i = 1$ *to* N
3. *Compute optimal solution to Problem (3):* $\mathbf{w}^i = \boldsymbol{X}^{i^T}\mathbf{u}^i$
4. *Compute linear hypothesis:* $\mathbf{t}^i = \boldsymbol{X}^i\mathbf{w}^i$*,* $\mathbf{t}^i \leftarrow \mathbf{t}^i/\|\mathbf{t}^i\|$*,* $\boldsymbol{T} = [\boldsymbol{T}\ \mathbf{t}]$
5. *Deflate:* $\mathbf{p}^i = \boldsymbol{X}^{i^T}\mathbf{t}^i$*,* $\boldsymbol{X}^{i+1} = \boldsymbol{X}^i - \mathbf{t}^i\mathbf{p}^{i^T}$

6. *Compute function:* $(\mu_y, \mathbf{c}) = \arg\min_{(\mu_y, \mathbf{c})} l(\mathbf{y}, \mu_y \mathbf{e} + \boldsymbol{T}\mathbf{c})$
7. *Compute negative gradient:* $\mathbf{u}^{i+1} = -\nabla l(\mathbf{y}, \mu_y \mathbf{e} + \boldsymbol{T}\mathbf{c})$
8. *end*
9. *Final features:* $T(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)^T \boldsymbol{W} \left(\boldsymbol{P}^T \boldsymbol{W}\right)^{-1}$ *where* $\boldsymbol{W}$ *and* $\boldsymbol{P}$ *have* $\mathbf{w}^i$ *and* $\mathbf{p}^i$ *as their columns, respectively.*
10. *Compute coefficients in original space:* $\mathbf{g} = \boldsymbol{W} \left(\boldsymbol{P}^T \boldsymbol{W}\right)^{-1} \mathbf{c}$
11. *Final function:* $f(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)^T \mathbf{g} + \mu_y$.

Using arguments much like those in Section 3.1, it can be shown that deflation forces orthogonality of many of the vectors constructed in the algorithm.

Lemma 1. *Orthogonality Properties of BLF. The following properties hold at termination of BLF:*

1. *The vectors* $\mathbf{w}^i$ *are mutually orthogonal:* ${\mathbf{w}^i}^T \mathbf{w}^j = 0$*, for* $i \neq j$.
2. *The vectors* $\mathbf{t}^i$ *are mutually orthogonal:* ${\mathbf{t}^i}^T \mathbf{t}^j = 0$*, for* $i \neq j$.
3. *The vectors* $\mathbf{w}^i$ *are orthogonal to the vectors* $\mathbf{p}^j$ *for* $i < j$*. Thus* $\boldsymbol{P}^T \boldsymbol{W}$ *is an upper triangular matrix.*

Proof. These properties also hold for PLS and the proofs for PLS in [15] apply to BLF without change. □

The end products of the algorithm are the final predictive function and the latent linear factors. Recall that the weights for the linear hypotheses were computed in the null space using the deflated data, thus Steps 9 and 10 are added to create the linear functions and final predictive function to the original space. As shown in the following theorem, it is not necessary to explicitly compute the null matrices or deflate the data to compute the final coefficients of the latent factors.

Theorem 1 (Mapping to original space). *The linear hypotheses* $\boldsymbol{T} = [\mathbf{t}^1, \ldots, \mathbf{t}^N]$ *in the original centered space are*

$$\boldsymbol{T} = \widetilde{\boldsymbol{X}} \boldsymbol{W} (\boldsymbol{P}^T \boldsymbol{W})^{-1} \tag{4}$$

Proof. The algorithm can run until the centered data matrix $\widetilde{\boldsymbol{X}}$ is fully factorized: $\widetilde{\boldsymbol{X}} = \sum_{i=1}^{\mathrm{rank}(\widetilde{\boldsymbol{X}})} \mathbf{t}^i {\mathbf{p}^i}^T$ and it can be divided into two parts.

$$\widetilde{\boldsymbol{X}} = \sum_{i=1}^{N} \mathbf{t}^i {\mathbf{p}^i}^T + \sum_{i=N+1}^{\mathrm{rank}(\widetilde{\boldsymbol{X}})} \mathbf{t}^i {\mathbf{p}^i}^T. \tag{5}$$

Multiplying by $\boldsymbol{W}$ on both sides yields:

$$\widetilde{\boldsymbol{X}} \boldsymbol{W} = \sum_{i=1}^{N} \mathbf{t}^i {\mathbf{p}^i}^T \boldsymbol{W} + \sum_{i=N+1}^{\mathrm{rank}(\widetilde{\boldsymbol{X}})} \mathbf{t}^i {\mathbf{p}^i}^T \boldsymbol{W} = \sum_{i=1}^{N} \mathbf{t}^i {\mathbf{p}^i}^T \boldsymbol{W}. \tag{6}$$

By Lemma 1, ${\mathbf{p}^i}^T \mathbf{w}^j = 0$ for $i = N+1, \ldots, \mathrm{rank}(\widetilde{\boldsymbol{X}})$ with $j = 1, \ldots, N$. So $\left(\widetilde{\boldsymbol{X}} - \sum_{i=1}^{N} \mathbf{t}^i {\mathbf{p}^i}^T\right) \boldsymbol{W} = \left(\widetilde{\boldsymbol{X}} - \boldsymbol{T}\boldsymbol{P}^T\right) \boldsymbol{W} = 0$ holds. Exploiting the fact that $\boldsymbol{P}^T \boldsymbol{W}$ is invertible (it is in fact a triangular matrix), solving for $\boldsymbol{T}$ yields the solution. □

The final steps in Algorithm 3 give the final latent factor function and predictive functions in the original space. These steps are always the same regardless of the particular loss function used. Step 9 computes the features or latent factors. The features may be scaled by the importance represented by the coefficients $\mathbf{c}$,

$$\hat{T}(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)\boldsymbol{W}\left(\boldsymbol{P}^T\boldsymbol{W}\right)^{-1} diag(\mathbf{c}). \tag{7}$$

As written, BLF does full refitting, i.e. the problem in Step 6 is solved to optimality. This allows us to prove some additional properties of the algorithm.

Lemma 2. Optimal Refit *Let l be a convex continuously (sub)differentiable function. At each iteration after Step 6 is successfully completed*

$$\boldsymbol{T}^T\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c}) = 0 \tag{8}$$

Proof. We prove this for the subgradient case since the differentiable functions are just a special case and let ∇ denote a subgradient. For convex unconstrained minimization with subgradients, a solution is optimal if and only if there exists a zero subgradient at that point [1]. The optimality condition for $\mathbf{c}$ is: $0 = \nabla_{\mathbf{c}} Loss(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c}) = \boldsymbol{T}^T\nabla Loss(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c})$ by the chain rule. □

Corollary 1. Orthogonality of U and T *At iteration i let $\mathbf{u}^{i+1} = -\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c})$, where $\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c})$ is the optimal subgradient found by refitting satisfying (8), then*

$${\mathbf{u}^i}^T\mathbf{t}^j = 0, \text{ for } i > j. \tag{9}$$

Proof. By Lemma 1, at the i^{th} step, ${\mathbf{u}^{i+1}}^T\boldsymbol{T} = 0$. Since $\boldsymbol{T} = [\mathbf{t}^1 \ \cdots \ \mathbf{t}^i]$, ${\mathbf{u}^{i+1}}^T\mathbf{t}^j = 0, \text{ for } i \geq j$ holds in general. □

This property permits $\boldsymbol{W}$ to be expressed in terms of the original variables.

Corollary 2. Alternate form of W *At iteration i let $\mathbf{u}^i = -\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c})$ where $\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \boldsymbol{T}\mathbf{c})$ is the optimal subgradient found by refitting satisfying (8), then $\mathbf{w}^i = \widetilde{\boldsymbol{X}}^T\mathbf{u}^i$ for $i = 1, \ldots N$, where $\widetilde{\boldsymbol{X}}$ is the centered data.*

For AdaBoost with discrete functions, the step sizes c^i are not reoptimized at later iterations. Step 6 in Algorithm 3 optimizes just one c^i. Therefore, the inequality $i > j$ in Equation (9) only holds with $i = j + 1$. The relation $\mathbf{u}^{j+1}\mathbf{t}^j = 0$ is known in AdaBoost as "the weak hypothesis will be exactly uncorrelated with the labels" [27], where "labels" corresponds to the pseudo residual $\mathbf{u}$.

4 Convergence Properties of BLF

If BLF is not stopped early by restricting the number of latent variables, BLF constructs a solution of the full model

$$\min_{\mathbf{g}, \mu_y} \ l(\mathbf{y}, \boldsymbol{X}\mathbf{g} + \mu_y\mathbf{e}) \tag{10}$$

in a finite number of iterations. Without loss of generality, assume that the threshold is eliminated by augmenting $\boldsymbol{X}$ with a constant column and that data centering is

optionally done as a preliminary step to form $\widetilde{\boldsymbol{X}}$. Equivalently we can say BLF constructs an optimal solution of

$$\min_{\mathbf{g}} \; l(\mathbf{y}, \widetilde{\boldsymbol{X}}\mathbf{g}) \tag{11}$$

Theorem 2. Finite Convergence of BLF to Full Model *Let l be a convex (sub)differentiable function and $\mathbf{u}^i$ be the negative (sub)gradient in Step 7 that is optimal for refitting. Assume BLF is modified to check for termination conditions at each iterations, e.g. at each iteration BLF calculates $\mathbf{g} = \boldsymbol{W}\left(\boldsymbol{P}^T\boldsymbol{W}\right)^{-1}\mathbf{c}$ and terminates if $\mathbf{g}$ is optimal for (11) or if $\widetilde{\boldsymbol{X}} = \boldsymbol{T}\boldsymbol{P}^T$. Then BLF terminates in a finite number of iterations and $\mathbf{g}$ is optimal for (11).*

Proof. The algorithm will terminate finitely since the number of iterations is limited by the rank of $\widetilde{\boldsymbol{X}}$. If $\mathbf{g}$ is optimal then the result holds. If the algorithm terminates because $\widetilde{\boldsymbol{X}} = \boldsymbol{T}\boldsymbol{P}^T$ then due to refitting $0 = \boldsymbol{T}^T\mathbf{u}^{i+1} = \boldsymbol{T}^T\nabla l(\mathbf{y}, \boldsymbol{T}\mathbf{c}) = \boldsymbol{T}^T\nabla l(\mathbf{y}, \widetilde{\boldsymbol{X}}\mathbf{g})$. Multiplying by $\boldsymbol{P}$, $\boldsymbol{P}\boldsymbol{T}^T\nabla l(\mathbf{y}, \widetilde{\boldsymbol{X}}\mathbf{g}) = \widetilde{\boldsymbol{X}}^T\nabla l(\mathbf{y}, \widetilde{\boldsymbol{X}}\mathbf{g}) = 0$, thus the (sub)gradient of the original problem is zero and thus optimal since the first order optimality conditions are satisfied and l is convex [1]. □

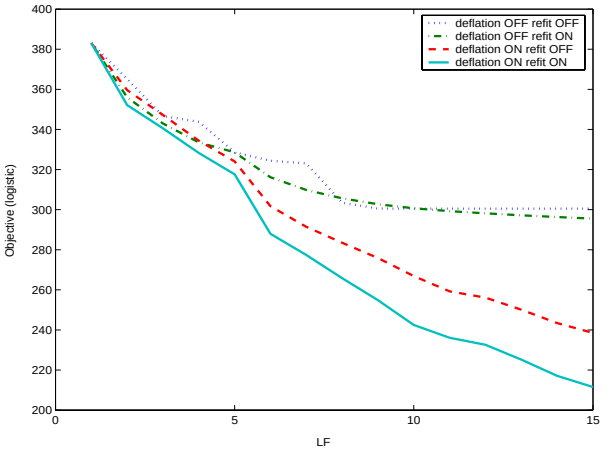

Fig. 1. Objective values of logistic loss for Diabetes data. Four possible options are examined: with/without deflation and with/without refit. Note the kernel trick is used to enrich the input space.

By introducing orthogonality, BLF goes from being a gradient descent algorithm which may have slow convergence for ill-conditioned loss functions to a conjugate-gradient-like subspace method with finite termination. As we show in the next section, BLF with least squares loss reduces to PLS which has been previously shown to be a conjugate gradient algorithm. Thus we can view BLF as a novel nonlinear conjugate gradient algorithm. Empirically the difference in convergence rate of the

gradient/boosting method versus the orthogonal BLF method can be quite marked for some loss functions. Figure 1 illustrates the different convergence rates for the negative binomial or logistic loss evaluated on the training data with the kernel trick explained in detail in Section 7. Without deflation and refit the method is just boosting. Without deflation and with refit the method is a gradient method with a more extensive line search. Details about the logistic loss implementation are provided in Section 6.2. The figure shows that without deflation the algorithm stalls and fails to make progress, a classic problem with gradient descent methods. With deflation, convergence improves markedly, with the fastest algorithm being BLF with deflation and refitting.

In practice BLF would be halted after a few iterations and the difference between BLF with and without refitting is not large in the first five iterations. This suggestions that full refitting may not be necessary to achieve good testing set results, but we leave the computational study of the best BLF variant to future work.

5 PLS and BLF

For the least squares loss function, orthogonal boosting with linear hypotheses reduces to the well studied partial least squares regression (PLS) algorithm. We derive BLF with least squares loss and show it reduces to PLS (Algorithm 4).

Algorithm 4 (Linear PLS) *Input:* $\boldsymbol{X}$, $\mathbf{y}$, N.

1. *Center data and compute mean response:*
   $\boldsymbol{\mu}_X = \frac{1}{m}\boldsymbol{X}^T\mathbf{e}$, $\boldsymbol{X}^1 = \boldsymbol{X} - \mathbf{e}\boldsymbol{\mu}_X^T$, $\mu_y = \frac{1}{m}\mathbf{e}^T\mathbf{y}$.
   $\mathbf{u}^1 = \mathbf{y} - \mu_y\mathbf{e}$ $\boldsymbol{T} = [\ ]$.
2. *For* $i = 1$ *to* N
3. *Compute optimal solution to problem (3):* $\mathbf{w}^i = {\boldsymbol{X}^i}^T\mathbf{u}^i$
4. *Compute linear hypothesis:* $\mathbf{t}^i = \boldsymbol{X}^i\mathbf{w}^i$, $\mathbf{t}^i \leftarrow \mathbf{t}^i/\|\mathbf{t}^i\|$, $\boldsymbol{T} = [\boldsymbol{T}\ \mathbf{t}]$
5. *Deflate:* $\mathbf{p}^i = {\boldsymbol{X}^i}^T\mathbf{t}^i$, $\boldsymbol{X}^{i+1} = \boldsymbol{X}^i - \mathbf{t}^i{\mathbf{p}^i}^T$
6. *Compute function:* $c^i = {\mathbf{u}^i}^T\mathbf{t}^i$
7. *Compute negative gradient:* $\mathbf{u}^{i+1} = \mathbf{u}^i - \mathbf{t}^i c^i$
8. *end*
9. *Final features:* $T(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)^T\boldsymbol{W}\left(\boldsymbol{P}^T\boldsymbol{W}\right)^{-1}$ *where* $\boldsymbol{W}$ *and* $\boldsymbol{P}$ *have* $\mathbf{w}^i$ *and* $\mathbf{p}^i$ *as their columns, respectively.*
10. *Compute coefficients in original space:* $\mathbf{g} = \boldsymbol{W}\left(\boldsymbol{P}^T\boldsymbol{W}\right)^{-1}\mathbf{c}$
11. *Final function:* $f(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_X)^T\mathbf{g} + \mu_y$.

The PLS loss function is the squared loss: $l(\mathbf{y}, \mathbf{f}) = \|\mathbf{y} - \mathbf{f}\|_2^2$. The negative gradient in Step 7 in Algorithm 3 is simply the residual: $\mathbf{u}^1 = -\nabla l(\mathbf{y}, \mathbf{f}) = (\mathbf{y} - \mathbf{f})$. As discussed above, the first step of BLF uses the constant hypothesis so deflation is equivalent to centering the data. Also at the constant hypothesis, the negative gradient of the squared loss is $-\nabla l(\mathbf{y}, \mu_y\mathbf{e}) = \mathbf{y} - \mu_y\mathbf{e}$, which is precisely $\mathbf{u}^1$ in PLS (Step 1), the step that mean centers the response. In general, at the i^{th} iteration, the negative gradient is: $\mathbf{u}^i = -\nabla l(\mathbf{y}, \mu_y\mathbf{e} + \sum_{j=1}^{i-1}\mathbf{t}^j c^j) = \mathbf{y} - \mu_y\mathbf{e} - \sum_{j=1}^{i-1}\mathbf{t}^j c^j = \mathbf{y} - \mathbf{f}$. The negative gradient is exactly the residual for the squared loss. At the i^{th}

iteration, the algorithm finds the descent direction by maximizing the inner product between $\mathbf{u}^i$ and the data projection $\boldsymbol{X}^i\mathbf{w}^i$. Since, all the variables are centered, maximizing the inner product becomes equivalent to maximizing the covariance of the residual response with the linear hypothesis, which corresponds to the statistical interpretation of the PLS optimization problem.

Step 6 computes the regression coefficients. PLS optimizes one coefficient at a time in closed form, but because of the orthogonality of latent factors and the least squares loss function, the coefficient is globally optimal for the loss function and there is no need for refit. PLS will be equivalent to ordinary least squares or the full model when the full rank of the original matrix is used in the extracted features. As noted above, for a generic loss function, refit is needed to insure convergence to the optimal solution of for the full-model and can increase the convergence rate.

BLF can be regarded as a generalization of PLS to an arbitrary loss function. The algorithmic and regularization properties of PLS have been extensively studied. PLS is in fact a classic conjugate gradient or Lanczos method [25]. Thus for least squares loss, BLF reduces to a conjugate gradient method that exploits second order properties of the loss function. BLF is *not* a classical nonlinear conjugate gradient algorithm. Conjugate gradient boosting in hypothesis space has been tried previously with mixed results [19]. BLF requires the hypotheses to be orthogonal in the hypothesis space. A nonlinear conjugate gradient algorithm would not enforce this property. BLF is a novel subspace optimization algorithm. It would be interesting to investigate the convergence rate properties of BLF and to compare BLF with a nonlinear conjugate algorithm but we leave these for future work.

PLS is known to regularize the solution and the extent of the regularization depends on the number of features constructed. See [13] for a nice analysis of these properties. When the number of feature is equal to the rank of the data, the PLS solution will coincide with the ordinary least squares solution. The norm of the regression weights increases as more latent factors are added. This property is also maintained but we have not formally proven it. BLF is closely related to the gradient based regularization also proposed in [13].

6 BLF for Arbitrary Loss

As discussed, Algorithm 3 gives the general framework for BLF. Among many choices of loss functions, the following loss functions are examined:

1. Squared loss (PLS)
$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i^2 \left(y_i - f(\mathbf{x}_i)\right)^2, \tag{12}$$
2. Least absolute deviation (1-norm loss)
$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i |y_i - f(\mathbf{x}_i)|, \tag{13}$$
3. Exponential loss
$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i \exp\left(-y_i f(\mathbf{x}_i)\right), \tag{14}$$
4. Negative binomial log-likelihood (logistic loss)
$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i \ln\left(1 + e^{-2y_i f(\mathbf{x}_i)}\right).$$

Note we introduced local weighting γ_i (> 0) for each data point $\mathbf{x}_i$ to be applicable for more general settings such as cost sensitive learning. Additionally, there are many possible strategies for optimizing the loss function that effect how to optimize $\mathbf{w}$ for creating the weak hypotheses and the step size $\mathbf{c}$. Following OrthoAnyBoost, $\mathbf{w}$ is determined by maximizing the inner product between the latent factor determined by $\mathbf{w}$ and the negative gradient. The step size or hypothesis weight $\mathbf{c}$ is optimized by an exact line search or one iteration of a Newton method, ensuring the full model can eventually be obtained.

6.1 Least Absolute Deviation

The least absolute deviation loss is more robust to outlying points that the squared loss function. Here a Least Absolute Deviation (LAD) loss function is used for regression problems. Support Vector Machines (SVM) for regression is one example of a method that uses a more robust loss function than ordinary least squares [9]. The LAD loss can be seen as a special case of the ϵ-insensitive loss [9] in which ϵ is set to zero. In principle, it is possible to use the ϵ-insensitive loss for BLF and it may make the model more tolerant to noise. We leave this to future work.

The LAD loss function is defined as

$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i |y_i - f(\mathbf{x}_i)|, \tag{15}$$

where $f(\mathbf{x}_i)$ is the sum of weak hypotheses: $f(\mathbf{x}_i) = \sum_{j=1}^{N} f^j(\mathbf{x}_i)$, and $\mathbf{f}$ is a vector representation of the function f. The $\gamma_i > 0$ are fixed error weights for each point. The LAD loss function is not differentiable at every point, but an appropriate subgradient does exist. For BLA, the LAD subgradient is defined as:[3]

$$\nabla l(\mathbf{y}, \mathbf{f}) = -\left[\gamma_1 \text{sign}(y_1 - f_1), \ldots, \gamma_m \text{sign}(y_m - f_m)\right]^T, \tag{16}$$

where f_i is the i^{th} element of the vector $\mathbf{f}$, $sign(\eta) = 1$ if $\eta > 0$, $sign(\eta) = -1$ if $\eta < 0$, and $sign(\eta) = 0$ if $\eta = 0$. Note all convergence and orthogonality properties are maintained and the algorithm performs quite well.

The first step in BLF is to optimize a constant hypothesis. The solution of the hypothesis weight problem $\text{argmin}_{\mu_y} \sum_{i=1}^{m} \gamma_i |y_i - \mu_y|$is the weighted median of $\{y_i\}_{i=1}^{m}$ with weights $\{\gamma_i\}_{i=1}^{m}$, see for example [14]. Once μ_y is obtained, the negative gradient $\mathbf{u}$ is computed. Thus, $\mathbf{u}^1$ at the first iteration is written by

$$\mathbf{u}^1 = \left[\gamma_1 \text{sign}(y_1 - \mu_y), \ldots, \gamma_m \text{sign}(y_m - \mu_y)\right]^T \tag{17}$$

There are two options for optimizing the function coefficients $\mathbf{c}$. We can optimize one c^i associated with $\mathbf{t}^i$ then fix the value of c^i for later iterations.

Or we can refit, e.g. re-optimize all the regression coefficients $[\mu_y \ \mathbf{c}] = \left[\mu_y, c^1, \ldots, c^i\right]$ associated with the weak hypotheses selected so far:

$$(\mu_y, \mathbf{c}) = \arg\min \ \|\text{diag}(\boldsymbol{\gamma})(\mathbf{y} - \sum_{j=1}^{i} \mathbf{t}^j c^j - \mu_y)\|_1, \tag{18}$$

[3] In a slight abuse of notation, we use ∇ to denote the subgradient whenever the gradient does not exist.

where $\boldsymbol{\gamma}$ is a vector representation of $\{\gamma_i\}_{i=1}^m$. In general, this problem can be solved using linear programming. Since $(\mu_y, \mathbf{c})$ are incrementally optimized, column generation methods can be used to efficiently update the solution at each iteration. Note that if reoptimization/refitting is not chosen, a closed-form solution can be utilized. See [12] for more detail. Once the regression coefficients are solved, the negative gradient, or pseudo response, is updated:

$$\mathbf{u}^{i+1} = \left[\gamma_1 \text{sign}(y_1 - \mu_y - \sum_{j=1}^{i} t_1^j c^j), \ldots, \gamma_m \text{sign}(y_m - \mu_y - \sum_{j=1}^{i} t_m^j c^j)\right]^T, \quad (19)$$

where t_j^i is the j^{th} element of the i^{th} weak hypothesis $\mathbf{t}^i$. In summary, the steps in Algorithm 3 for LAD loss are specified as follows:

- Step 1: $\mu_y = \text{median}_{\boldsymbol{\gamma}}(\mathbf{y})$ and $\mathbf{u}^1 = [\gamma_1 \text{sign}(y_1 - \mu_y), \ldots, \gamma_m \text{sign}(y_m - \mu_y)]^T$
- Step 6: LAD loss is minimized by linear program (18)
- Step 7: $\mathbf{u}^{i+1} = \left[\gamma_1 \text{sign}(y_1 - \mu_y - \sum_{j=1}^{i} t_1^j c^j), \ldots, \gamma_m \text{sign}(y_m - \mu_y - \sum_{j=1}^{i} t_m^j c^j)\right]^T$

6.2 Exponential Loss

The exponential loss function was used in AdaBoost [10] for binary classification problems. AdaBoost changes the weights on the data points at each iteration; more difficult instances are given larger weights. The algorithm can be understood as minimizing the exponential loss functional [11, 18]:

$$l(\mathbf{y}, \mathbf{f}) = \sum_{i=1}^{m} \gamma_i \exp\left(-y_i f(\mathbf{x}_i)\right), \quad (20)$$

where the responses are given in binary coding: $y_i \in \{+1, -1\}$, $i = 1, \ldots, m$.

Computing Descent Directions in BLF

Let's discuss how to formulate the exponential loss in BLF. At first a constant weak hypothesis is added to the additive model. Then the regression coefficient μ_y on the constant hypothesis is determined by optimizing

$$\begin{aligned} \min_{\mu_y} l(\mathbf{y}, \mu_y \mathbf{e}) &= \min_{\mu_y} \sum_{i=1}^{m} \gamma_i \exp\left(-y_i \mu_y\right) \\ &= \min_{\mu_y} e^{-\mu_y} \sum_{i \in \mathcal{C}^+} \gamma_i + e^{\mu_y} \sum_{i \in \mathcal{C}^-} \gamma_i, \end{aligned} \quad (21)$$

where $\mathcal{C}^+$ is a set of positive data points and $\mathcal{C}^-$ is a set of negative data points. With this simplification, it is now easy to solve the optimality condition $\frac{\partial l(\mathbf{y}, \mu_y)}{\partial \mu_y} = 0$, to compute the solution:

$$\mu_y = \frac{1}{2} \ln \frac{\sum_{i \in \mathcal{C}^+} \gamma_i}{\sum_{i \in \mathcal{C}^-} \gamma_i}. \quad (22)$$

The next step computes the negative gradient of the loss function:

$$u_k^{i+1} = \left(-\nabla l(\mathbf{y}, \mu_y + \sum_{j=1}^{i} \mathbf{t}^j c^j) \right)_k = \gamma_k e^{-y_k \left(\mu_y + \sum_{j=1}^{i} t_k^j c^j \right)} y_k. \tag{23}$$

As seen in Equation (23), the weak hypotheses in previous iterations are absorbed in the "weight" defined as $\gamma_k e^{-y_k \left(\mu_y + \sum_{j=1}^{i} t_k^j c^j \right)}$. This weighted response becomes a new response variable to fit in creating a new weak hypothesis, which corresponds to the weight updates on each of the data points in the AdaBoost algorithm.

Solving for Function Coefficients

At every iteration i, BLF computes the function coefficients $\mathbf{c}$ when a new weak hypothesis or latent factor is added to the model. Refitting for exponential loss minimizes the following loss function with respect to μ_y and $\mathbf{c}$:

$$l(\mathbf{y}, \sum_{j=1}^{i} \mathbf{t}^j c^j) = \sum_{k=1}^{m} \gamma_k \exp\left(-y_k (\mu_y + \sum_{j=1}^{i} t_k^j c^j) \right). \tag{24}$$

It is convenient to define a matrix notation. Let

$$\mathbf{d} = \left[\gamma_1 \exp\left(-y_1 (\mu_y + \sum_{j=1}^{i} t_1^j c^j) \right), \ldots, \gamma_m \exp\left(-y_m (\mu_y + \sum_{j=1}^{i} t_m^j c^j) \right) \right]^T \tag{25}$$

Then the loss function is simply expressed by

$$l(\mathbf{y}, \mu_y + \sum_{j=1}^{i} \mathbf{t}^j c^j) = \mathbf{e}^T \mathbf{d}. \tag{26}$$

The gradient with respect to $(\mu_y, \mathbf{c})$ is given by

$$\begin{aligned} \nabla_{(\mu_y, \mathbf{c})} l(\mathbf{y}, \mu_y + \sum_{j=1}^{i} \mathbf{t}^j c^j) &= \begin{bmatrix} -\sum_{k=1}^{m} \gamma_k e^{-y_k (\mu_y + \sum_{j=1}^{i} t_k^j c^j)} y_k \\ -\sum_{k=1}^{m} \gamma_k e^{-y_k (\mu_y + \sum_{j=1}^{i} t_k^j c^j)} y_k t_k^1 \\ \vdots \\ -\sum_{k=1}^{m} \gamma_k e^{-y_k (\mu_y + \sum_{j=1}^{i} t_k^j c^j)} y_k t_k^i \end{bmatrix} \\ &= -[\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{d}) \mathbf{y}. \end{aligned} \tag{27}$$

The Hessian is given as follows:

$$\nabla^2_{(\mu_y, \mathbf{c})} l(\mathbf{y}, \mu_y + \sum_{j=1}^{i} \mathbf{t}^j c^j) = [\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{y}) \mathrm{diag}(\mathbf{d}) \mathrm{diag}(\mathbf{y}) [\mathbf{e}\ \boldsymbol{T}] \tag{28}$$

Since $y_i \in \{+1, -1\}$ for binary classification, the Hessian can be written as

$$\nabla^2_{(\mu_y, \mathbf{c})} l = [\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{d}) [\mathbf{e}\ \boldsymbol{T}]. \tag{29}$$

Thus the Newton step is given by:

$$\begin{bmatrix} \mu_y \\ \mathbf{c} \end{bmatrix} = \left([\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{d}) [\mathbf{e}\ \boldsymbol{T}] \right)^{-1} [\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{d}) \mathbf{y}, \tag{30}$$

which is just a weighted least squares problem with the vector $\mathbf{d}$ determining the weights on data points. Since μ_y and $\mathbf{c}$ are incrementally optimized, the Newton step can be started from the previously optimized value to reduce the number of iterations to converge. In practice, a few iterations are sufficient to get a good fit to the response variable. In summary, the steps in Algorithm 3 for the exponential loss are specified as follows:

- Step 1: $\mu_y = \frac{1}{2} \ln \frac{\sum_{i \in \mathcal{C}+} \gamma_i}{\sum_{i \in \mathcal{C}-} \gamma_i}$
- Step 1 and 7:
$$u_k^{i+1} = \gamma_k e^{-y_k \left(\mu_y + \sum_{j=1}^{i} t_k^j c^j\right)} y_k$$
- Steps 6: the step size $(\mu_y, \mathbf{c})$ optimized by the Newton step

6.3 The Negative Binomial Log-likelihood

The binomial likelihood is parameterized by

$$\begin{aligned} p(y=1|\mathbf{x}) &= \frac{e^{f(\mathbf{x})}}{e^{f(\mathbf{x})} + e^{-f(\mathbf{x})}} = \frac{e^{f(\mathbf{x})}}{2\cosh\left(f(\mathbf{x})\right)} \\ p(y=-1|\mathbf{x}) &= \frac{e^{-f(\mathbf{x})}}{e^{f(\mathbf{x})} + e^{-f(\mathbf{x})}} = \frac{e^{-f(\mathbf{x})}}{2\cosh\left(f(\mathbf{x})\right)}. \end{aligned} \tag{31}$$

Equivalently, $f(\mathbf{x})$ is expressed by $p(y=1|\mathbf{x})$ and $p(y=-1|\mathbf{x})$:

$$f(\mathbf{x}) = \frac{1}{2} \ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})}. \tag{32}$$

The negative log-likelihood loss, including local weighting γ_i, is given by

$$\begin{aligned} l(\mathbf{y}, \mathbf{f}) &= -\sum_{i \in \mathcal{C}+} \gamma_i \ln\left(p(y=1|\mathbf{x}_i)\right) - \sum_{i \in \mathcal{C}-} \gamma_i \ln\left(p(y=-1|\mathbf{x}_i)\right) \\ &= -\sum_{i=1}^{m} \gamma_i \left[y_i f(\mathbf{x}_i) - \ln\left(2\cosh f(\mathbf{x}_i)\right)\right] \\ &= \sum_{i=1}^{m} \gamma_i \ln\left(1 + e^{-2y_i f(\mathbf{x}_i)}\right). \end{aligned} \tag{33}$$

Note that this loss function is used for additive logistic regression [11,18] and is just a factor of 2 different from the loss in logistic regression. It is also known that negative binomial loss is equivalent to the exponential loss up to the second order [11, 18].

Computing the Descent Directions in the Negative Binomial Log-likelihood

In Step 1, a constant hypothesis is added to the model:

$$\begin{aligned} \min l(\mathbf{y}, \mu_y \mathbf{e}) &= \sum_{i=1}^{m} \gamma_i \ln\left(1 + e^{-2y_i \mu_y}\right) \\ &= \ln\left(1 + e^{-2\mu_y}\right) \sum_{i \in \mathcal{C}+} \gamma_i + \ln\left(1 + e^{2\mu_y}\right) \sum_{i \in \mathcal{C}-} \gamma_i. \end{aligned} \tag{34}$$

Solving $\frac{\partial l(\mathbf{y},\mu_y\mathbf{e})}{\partial \mu_y} = 0$ yields

$$\mu_y = \frac{1}{2}\ln\frac{\sum_{i\in\mathcal{C}^+}\gamma_i}{\sum_{i\in\mathcal{C}^-}\gamma_i}. \tag{35}$$

The next step computes the negative gradient of the loss function.

$$u_k^{i+1} = \left(-\nabla l(\mathbf{y},\mu_y\mathbf{e}+\sum_{j=1}^{i}\mathbf{t}^j c^j)\right)_k = \gamma_k\left(y_k - \tanh\left(\mu_y + \sum_{j=1}^{i} t_k^j c^j\right)\right). \tag{36}$$

This equation means that the new pseudo response is given by the residual between $\mathbf{y}$ and the hyperbolic tangent of the linear combination of the weak hypotheses, just like in neural networks.

Solving for Function Coefficients

Refitting re-optimizes the function coefficients $\mathbf{c}$ at each iteration. The loss function is written by

$$\begin{aligned} &l(\mathbf{y},\mu_y\mathbf{e}+\sum_{j=1}^{i}\mathbf{t}^j c^j) = \\ &-\sum_{k=1}^{m}\gamma_k\left[y_k(\mu_y+\sum_{j=1}^{i}t_k^j c^j) - \ln\left(2\cosh\left(\mu_y+\sum_{j=1}^{i}t_k^j c^j\right)\right)\right]. \end{aligned} \tag{37}$$

The gradient with respect to $(\mu_y, \mathbf{c})$ is

$$\begin{aligned} &\nabla_{(\mu_y,\mathbf{c})} l(\mathbf{y},\mu_y\mathbf{e}+\sum_{j=1}^{i}\mathbf{t}^j c^j) \\ &= \begin{bmatrix} -\sum_{k=1}^{m}\gamma_k\left(y_k - \tanh(\mu_y+\sum_{j=1}^{i}t_k^j c^j)\right) \\ -\sum_{k=1}^{m}\gamma_k t_k^1\left(y_k - \tanh(\mu_y+\sum_{j=1}^{i}t_k^j c^j)\right) \\ \vdots \\ -\sum_{k=1}^{m}\gamma_k t_k^i\left(y_k - \tanh(\mu_y+\sum_{j=1}^{i}t_k^j c^j)\right) \end{bmatrix} \\ &= -[\mathbf{e}\ \boldsymbol{T}]^T\mathbf{r}, \end{aligned} \tag{38}$$

where $r_k = \gamma_k\left(y_k - \tanh(\mu_y+\sum_{j=1}^{i}t_k^j c^j)\right)$. Furthermore, the Hessian is given by

$$\nabla^2_{(\mu_y,\mathbf{c})} l = [\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}(\mathbf{d})[\mathbf{e}\ \boldsymbol{T}], \tag{39}$$

where $\mathbf{d}$ is defined as follows:

$$\mathbf{d} = \left(\left[\gamma_1\cosh^{-2}(\mu_y+\sum_{j=1}^{i}t_1^j c^j),\ldots,\gamma_m\cosh^{-2}(\mu_y+\sum_{j=1}^{i}t_m^j c^j)\right]^T\right). \tag{40}$$

Thus the Newton step is given by

$$\begin{bmatrix} \mu_y \\ \mathbf{c} \end{bmatrix} = \left([\mathbf{e}\ \boldsymbol{T}]^T \mathrm{diag}\,(\mathbf{d})\,[\mathbf{e}\ \boldsymbol{T}] \right)^{-1} [\mathbf{e}\ \boldsymbol{T}]^T \mathbf{r}. \tag{41}$$

This optimization is done in the same fashion as for exponential loss.

The Newton step may not always lead to a decrease in the objective function, leading to a condition that can be readily detected while training. A modified Newton step, which adds a multiple of the identity matrix to the Hessian before computing the Newton step, can be used. The following heuristic for a Modified Newton step was found to work well in practice based on the parameter $0 \leq \lambda \leq 1$, Hessian $\boldsymbol{H} = \nabla^2_{(\mu_y, \mathbf{c})} l$, and iteration number i:

$$\widehat{\boldsymbol{H}} = (1-\lambda)\boldsymbol{H} + \frac{\lambda\, trace(\boldsymbol{H})}{i+1} I \tag{42}$$

Then $\widehat{\boldsymbol{H}}$ is used to compute a modified Newton step in Step 6 instead of the original Newton step.

In summary, the steps in Algorithm 3 for the negative binomial log-likelihood loss are specified as follows:

- Step 1: $\mu_y = \frac{1}{2} \ln \frac{\sum_{j \in \mathcal{C}^+} \gamma_j}{\sum_{j \in \mathcal{C}^-} \gamma_j}$
- Steps 1 and 7:
$$u_k^{i+1} = \gamma_k \left(y_k - \tanh \left(\mu_y + \sum_{j=1}^{i} t_k^j c^j \right) \right)$$
- Step 6: the step size $(\mu_y, \mathbf{c})$ optimized by the modified Newton step

7 Kernel BLF

The kernel extension of BLF is similar to that of kernel PLS [26]. BLF is expressed in terms of inner products of data points, or equivalently in terms of the Gram matrix $\boldsymbol{X}\boldsymbol{X}^T$. We then transform the input space into the kernel-defined feature space: $\boldsymbol{X}\boldsymbol{X}^T \mapsto \boldsymbol{K}$. With this basic rule in mind, we can modify the BLF algorithm (Algorithm 3). In Step 1, the deflation of the data matrix should be replaced by centering the kernel matrix. That is, $\boldsymbol{K}^1 = \widetilde{\boldsymbol{K}} = \left(\boldsymbol{I} - \frac{1}{m}\mathbf{e}\mathbf{e}^T\right) \boldsymbol{K} \left(\boldsymbol{I} - \frac{1}{m}\mathbf{e}\mathbf{e}^T\right)$. Steps 3 and 4 are combined to avoid expressing $\mathbf{w}$. Namely, the latent factor is computed by $\mathbf{t}^i \in \{\mathbf{t} | \mathbf{u}^{iT} \boldsymbol{K}^i \mathbf{t} > 0\}$. Step 5 is replaced by the kernel deflation: $\boldsymbol{K}^{i+1} = \left(\boldsymbol{I} - \mathbf{t}^i \mathbf{t}^{iT}\right) \boldsymbol{K}^i \left(\boldsymbol{I} - \mathbf{t}^i \mathbf{t}^{iT}\right)$.

The step computing the function coefficients associated with the original data matrix $\boldsymbol{X}$ is now computed using the dual function coefficients, $\widetilde{\boldsymbol{\beta}}$. The formula in Step 9 in Algorithm 3 is expressed in terms of dual variables $\mathbf{t}$, $\mathbf{u}$, and the kernel $\boldsymbol{K}$. As shown in Section 3, the properties $\boldsymbol{P} = \widetilde{\boldsymbol{X}}^T \boldsymbol{T}$ and $\boldsymbol{W} = \widetilde{\boldsymbol{X}}^T \boldsymbol{U}$ hold. Thus the formula $\mathbf{g} = \boldsymbol{W} \left(\boldsymbol{P}^T \boldsymbol{W}\right)^{-1} \mathbf{c}$ can be rewritten by $\mathbf{g} = \widetilde{\boldsymbol{X}}^T \boldsymbol{U} \left(\boldsymbol{T}^T \widetilde{\boldsymbol{X}} \widetilde{\boldsymbol{X}}^T \boldsymbol{U}^T\right)^{-1} \mathbf{c}$, which gives the formula for the kernel PLS regression coefficients: $\widetilde{\boldsymbol{\beta}} = \boldsymbol{U} \left(\boldsymbol{T}^T \widetilde{\boldsymbol{K}} \boldsymbol{U}^T\right)^{-1} \mathbf{c}$. The first property $\boldsymbol{P} = \widetilde{\boldsymbol{X}}^T \boldsymbol{T}$ results from deflation. However, as shown in Corollary 2, $\boldsymbol{W} = \widetilde{\boldsymbol{X}}^T \boldsymbol{U}$ only holds if the step sizes are refitted. Thus, early termination of

the Newton iteration to determine $\mathbf{c}^i$ or fixing the step size for earlier iterations will violate Equation (9). Therefore, we need to find a matrix $\boldsymbol{A}$ such that $\boldsymbol{W} = \widetilde{\boldsymbol{X}}^T \boldsymbol{A}$ in order to express the inference functions in terms of the original data. Using the deflation formula, we can readily find a formula for an i^{th} column of the matrix $\boldsymbol{A}$:

$$\mathbf{a}^i = \mathbf{u}^i - \sum_{j=1}^{i-1} (\mathbf{t}^{j\,T} \mathbf{u}^i) \mathbf{t}^j. \tag{43}$$

Using the matrix $\boldsymbol{A}$, we can write the formula for the function coefficients as follows:

$$\widetilde{\boldsymbol{\beta}} = \boldsymbol{A} \left(\boldsymbol{T}^T \widetilde{\boldsymbol{K}} \boldsymbol{A} \right)^{-1} \mathbf{c}. \tag{44}$$

Using the dual regression coefficient $\widetilde{\boldsymbol{\beta}}$, the final prediction is written by

$$f(\mathbf{x}) = \sum_{i=1}^{m} \tilde{K}(\mathbf{x}, \mathbf{x}_i) \tilde{\beta}_i + \mu_y, \tag{45}$$

where the centered test kernel is denoted by $\widetilde{\mathbf{k}} \equiv [\tilde{K}(\mathbf{x}, \mathbf{x}_1), \ldots, \tilde{K}(\mathbf{x}, \mathbf{x}_m)]^T$. The formula for the centered test kernel is easily derived first by considering a linear kernel, then extending to general kernels:

$$\begin{aligned} \widetilde{\mathbf{k}}^T &= \left(\mathbf{x} - \frac{1}{m} \boldsymbol{X}^T \mathbf{e} \right)^T \boldsymbol{X}^T \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right) \\ &= \left(\mathbf{x}^T \boldsymbol{X}^T - \frac{1}{m} \mathbf{e}^T \boldsymbol{X} \boldsymbol{X}^T \right) \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right) \\ &\mapsto \left(\mathbf{k}^T - \boldsymbol{\mu}_K^T \right) \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right), \end{aligned} \tag{46}$$

where $\boldsymbol{\mu}_K$ is a mean vector of the kernel matrix: $\boldsymbol{\mu}_K = \frac{1}{m} \boldsymbol{K} \mathbf{e}$. Thus, by putting $\boldsymbol{\beta} = \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right) \widetilde{\boldsymbol{\beta}}$ and $\mu_K = \boldsymbol{\mu}_K^T \boldsymbol{\beta}$, the final regression function is simply expressed by

$$f(\mathbf{x}) = \sum_{i=1}^{m} K(\mathbf{x}, \mathbf{x}_i) \beta_i - \mu_K + \mu_y. \tag{47}$$

With Equation (47), prediction can be done using the uncentered kernel, modified dual regression coefficient $\boldsymbol{\beta}$ and a constant bias term $(-\mu_K + \mu_y)$.

Algorithm 5 (Kernel Boosted Latent Factors) *Input:* $\boldsymbol{K}$, $\mathbf{y}$, N.

1. *Compute* $\mu_y = \arg\min_{\mu_y} l(\mathbf{y}, \mu_y \mathbf{e})$.
   Deflate $\boldsymbol{K}^1 = \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right) \boldsymbol{K} \left(\boldsymbol{I} - \frac{1}{m} \mathbf{e}\mathbf{e}^T \right)$. $\mathbf{u}^1 = -\nabla l(\mathbf{y}, \mu_y \mathbf{e})$, $\boldsymbol{T} = [\,]$.
2. *For* $i = 1$ *to* N
3. *Compute latent factors:*
   $\mathbf{t}^i = \boldsymbol{K}^i \mathbf{u}^i$, $\mathbf{t}^i \leftarrow \mathbf{t}^i / \|\mathbf{t}^i\|$, $\boldsymbol{T} = [\boldsymbol{T}\ \mathbf{t}]$
4. *Deflate:* $\boldsymbol{K}^{i+1} = \left(\boldsymbol{I} - \mathbf{t}^i \mathbf{t}^{i\,T} \right) \boldsymbol{K}^i \left(\boldsymbol{I} - \mathbf{t}^i \mathbf{t}^{i\,T} \right)$
5. *Compute the function:*
   $(\mu_y, \mathbf{c}) = \arg\min_{(\mu_y, \mathbf{c})} l(\mathbf{y}, \mu_y \mathbf{e} + \boldsymbol{T}\mathbf{c})$

6. Compute: $\mathbf{u}^{i+1} = -\nabla l(\mathbf{y}, \boldsymbol{T}\mathbf{c})$
7. end
8. For $i = 1$ *to* N
9. $\mathbf{a}^i = \mathbf{u}^i - \sum_{j=1}^{i-1} (\mathbf{t}^{j^T} \mathbf{u}^i)\mathbf{t}^j$
10. end
11. Final features: $T(\mathbf{x}) = K^1(\mathbf{x}, \boldsymbol{X})\boldsymbol{A}\left(\boldsymbol{T}^T\boldsymbol{K}^1\boldsymbol{A}\right)^{-1}$, *where* $\boldsymbol{A}$ *has* $\mathbf{a}^i$ *as its columns.*
12. Compute the coefficients and bias:
$\boldsymbol{\beta} = \left(\boldsymbol{I} - \frac{1}{m}\mathbf{e}\mathbf{e}^T\right)\boldsymbol{A}\left(\boldsymbol{T}^T\boldsymbol{K}^1\boldsymbol{A}\right)^{-1}\mathbf{c}$, $\mu_K = \boldsymbol{\mu}_K^T\boldsymbol{\beta}$.
13. Final prediction function is:
$f(\mathbf{x}) = \sum_{i=1}^{m} K(\mathbf{x}, \mathbf{x}_i)\boldsymbol{\beta} - \mu_K + \mu_y$.

8 Computational Results

This section presents computational results for linear and kernel BLF. Recall BLF can be used both to construct orthogonal features targeted to a particular loss function and to create predictive models for a loss function. Figure 2 illustrates the value of orthogonal features targeted to a specific loss. The Cancer data set from the UCI machine learning repository [4] is used. The first two features of the nine dimensional Cancer data constructed by PCA, BLF with least squares (BLF-LS) loss, which is equivalent to PLS, and BLF with logistic loss (BLF-LOG) are shown. The PCA plot does not use information from the response and does not capture the class structure as well. The PCA plot looks rather noisy since the outliers have more influence; outliers account for the high variance. BLF-LS and BLF-LOG discriminate the two classes more clearly and could be used as input another modeling method or the BLF function could be used for classification. BLF-LOG constructs a nice Gaussian blob for each class.

The next two subsections are devoted to quantitative studies of BLF. The first set of experiments illustrates BLF's competitive performance as a predictive modeling tool on standard machine learning test beds using common cost functions. In the second case study, we investigate the use of kernel BLF on a very high dimensional unbalanced classification problem, *Thrombin*, from the KDD cup 2001 (also known as the Dorothea dataset in the NIPS 2003 Feature Selection Challenge).

8.1 The Benchmark Data Sets

In this subsection, real world data sets, Boston Housing, Cancer, Diabetes, Liver, and Wisconsin Breast Cancer (WBC) from the UCI machine learning repository [4], and Albumin (a drug discovery dataset) [2], are examined. Boston Housing and Albumin are regression data sets. All others are binary classification problems. We report both the mean and the standard deviation of the mean squared error in case of regression or accuracy in percentages in case of classification. Except for Boston Housing, all the experiments are repeated 100 times. For Boston Housing, 10-fold cross validation (CV) is adopted. Uniform weighting of data is always used.

For each of the trials, 10% of the data are held out as a test set and the rest are used for training. Every experiment uses the same splits of training and test sets. The number of latent features N is determined by 10-fold CV inside the training set. N is selected to be the number of latent features that minimizes a moving average

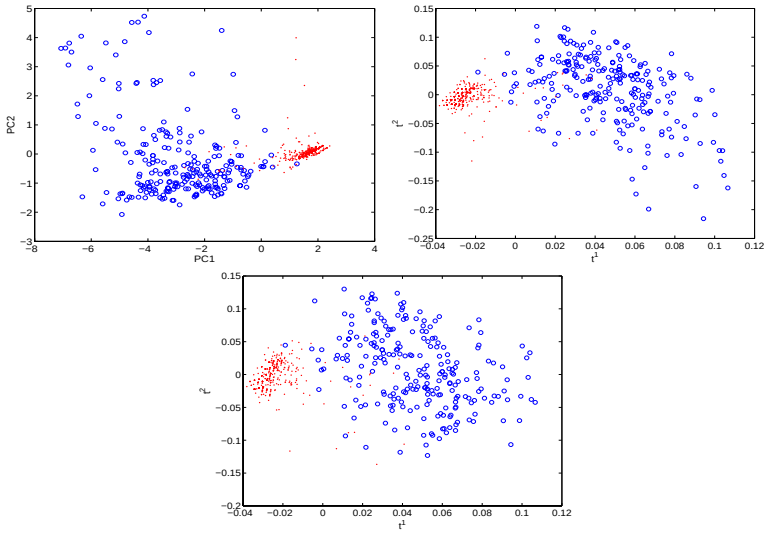

Fig. 2. The first two components of PCA ($upper - left$), BLF-LS (PLS) ($upper - right$) and BLF-LOG ($lower$). The positive examples are dots and the negative examples are circles.

error curve, thus N is different among the different train-test splits. Since the error curve is based on random splits of the 10-fold training data CV, the error for a given latent feature i is averaged with the next smaller $i-1$ and next larger $i+1$ results for all the experiments.

For kernel BLF, the radial basis function (RBF) kernel,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}),$$

is used. The kernel parameter σ is determined so that the value of σ creates reasonably good models for *all the loss functions* by 10-fold cross-validation runs over all the data points. Five to ten different σs are tried depending on the data set.

Refitting the step size is used with in all of the loss functions except least squares. But least squares, not refitting is mathematically equivalent to refitting. LAD refit was solved exactly by linear programming. For exponential and logistic loss, the Newton step is iterated only once for computational efficiency. The modified Newton step in Equation (42) with $\lambda = 0.1$ is used. For the RBF kernel, the kernel matrix has full rank m, and without the regularization provided by the single modified Newton Step the model overfits the training set quickly. We leave a more formal treatment of regularization within BLF to future work.

As a reference, we also perform the same experiments with SVMs [6]. The experimental settings are the same as other BLF models except for a selection of trade-off parameter C. Depending on the data set, a wide range of values of C (about 10) is examined by 10-fold CV inside the training data and the one that minimizes the CV error is chosen. SVM^{light} [16] is used for all the SVM experiments.

The paired student t-test is used to test significance. Since the baseline method is BLF-LS (PLS), the t-test is performed against the results of BLF-LS. Bold numbers indicate the method was significantly better than BLF-LS with quadratic loss using a significance level of 0.05. Note that SVM models are not compared in the t-test.

		Quadratic		LAD		SVM	
		Train	Test	Train	Test	Train	Test
Boston Housing	Mean	21.8173	23.4910	24.6214	25.6909	22.5446	24.1008
(506 × 13)	STD	1.1102	10.1094	1.4260	14.0496	1.1150	11.6537
	LF	8 (0)		6.8 (1.40)		—	
Albumin	Mean	0.1586	0.4044	0.1245	**0.3201**	0.0937	0.4331
(94 × 524)	STD	0.0999	0.2165	0.0243	0.2037	0.0443	0.2012
	LF	3.39 (1.39)		4.95 (0.77)		—	

Table 1. Regression results for linear models for the quadratic and least absolute deviation (LAD) loss functions. SVM results provided for reference. Data sizes ($m \times n$) are also shown.

		Quadratic		Exponential		Logistic		SVM	
		Train	Test	Train	Test	Train	Test	Train	Test
Cancer	Mean	95.97	96.00	96.57	96.22	96.91	**96.74**	97.03	96.72
(699 × 9)	STD	0.30	0.24	0.30	2.25	0.22	2.13	0.24	1.95
	LF	4.08 (1.03)		5.84 (0.85)		5.41 (1.08)		—	
Diabetes	Mean	78.08	76.01	77.62	75.80	78.26	**76.33**	77.93	76.04
(768 × 8)	STD	0.51	4.03	0.58	3.90	0.52	4.02	0.62	4.08
	LF	5.20 (0.70)		5.15 (0.78)		5.39 (0.71)		—	
Ionosphere	Mean	90.18	85.86	91.64	85.97	93.47	**86.83**	93.12	86.06
(351 × 34)	STD	1.04	5.86	1.30	5.90	1.31	5.86	1.74	5.54
	LF	5.69 (1.62)		7.40 (1.26)		7.11 (1.27)		—	
Liver	Mean	69.93	**69.24**	69.40	68.53	96.98	69.00	70.93	69.15
(345 × 6)	STD	1.33	8.15	1.45	8.76	1.37	8.92	1.27	7.62
	LF	4 (0)		4 (0)		4 (0)		—	
WBC	Mean	96.64	95.91	98.10	**97.14**	98.65	**97.80**	98.50	97.52
(569 × 30)	STD	0.31	2.33	0.33	1.93	0.22	1.85	0.34	1.88
	LF	6.16 (0.75)		5.34 (0.54)		5.38 (0.49)		—	

Table 2. Classification results for linear models for the quadratic, exponential and logistic loss functions. SVM results provided for reference. Data sizes ($m \times n$) are also shown.

As shown in Table 1, BLF works well for all of the loss functions. For regression, BLF with LAD loss (BLF-LAD) and BLF-LS achieve results for Boston Housing that are not significantly different. Note that only 10 trials were performed for Boston Housing, which might be why the difference is not significant by the t-test.

For Albumin, the results for the LAD loss show significantly better performance. For classification (Table 2), BLF-LOG performs better than BLF-LS on all of the data sets except for Liver. The exponential loss (BLF-EXP) is not as good as BLF-LOG for all of the data sets. Figure 3 shows accuracy versus number of latent factors for BLF-LOG and BLF-LS on the cancer data. BLF-LS's quadratic loss function penalizes points that are classified"too well", which serves as a form of capacity control for BLF-LS. BLF-LS does not always fit training and test sets well but the behavior is stable. BLF-LOG, however, does not have such penalization via the loss function, so BLF-LOG can fit the training data better than BLF-LS. Eventually BLF-LOG ovefits. Selecting the number of latent factors in BLF serves as regularization, and the experiments show that the tuning number of latent factors by CV succeeds in avoiding over-fitting – an advantage over regular logistic regression.

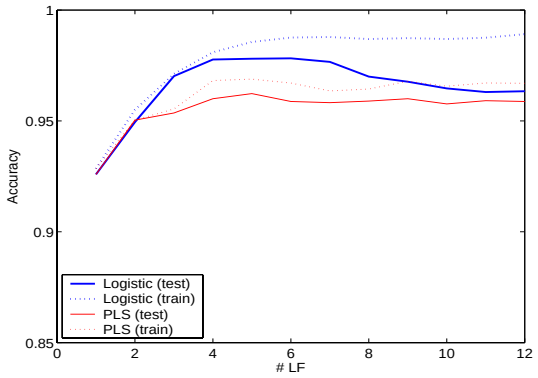

Fig. 3. Testing accuracy for Wisconsin Breast Cancer for logistic and least squares loss.

The kernel version of BLF is also examined for the same data sets. The RBF kernel is used with parameters, σ, as follows: 50 for Albumin, 4.24 for Boston Housing, 5 for Cancer, 5 for Diabetes, 3 for Ionosphere, 6 for Liver and 9 for WBC. In kernel models, especially with the RBF kernel, regularization plays a key role for capacity control, and capacity control is much more important for nonlinear models. As mentioned in linear models, BLF-LS has a natural penalization for overfitting, but the classification loss functions do not have such penalization except for early stopping. As seen in Table 4, the advantage observed in linear models seems to be reduced and the differences between the models are less significant. This observation suggests that the kernel BLF with explicit regularization in the objective function may be better to exploit the logistic and other loss function without overfitting.

Comparison with PCA

To evaluate the effectiveness of BLF at feature construction, we repeated the kernel classification experiments using principal components instead of latent factors. The

		Quadratic		LAD		SVM	
		Train	Test	Train	Test	Train	Test
Boston Housing	Mean	3.5723	9.8334	4.8788	10.4530	3.8194	10.1425
(506 × 13)	STD	0.4464	4.3849	0.9215	7.6215	0.5040	7.3367
	LF	19.2 (1.81)		29.6 (4.97)		—	
Albumin	Mean	0.0854	0.3764	0.0992	**0.3266**	0.1017	0.4190
(94 × 524)	STD	0.0098	0.2054	0.0228	0.1983	0.0460	0.1894
	LF	4.14 (1.15)		4.95 (0.70)		—	

Table 3. Regression results for kernel models for quadratic and least absolute deviation (LAD) loss functions. SVM results are also shown as a reference. Data sizes ($m \times n$) are also shown.

		Quadratic		Exponential		Logistic		SVM	
		Train	Test	Train	Test	Train	Test	Train	Test
Cancer	Mean	96.95	96.46	96.87	96.51	96.70	**96.71**	96.16	96.87
(699 × 9)	STD	0.47	1.99	0.35	2.15	0.28	1.98	0.27	1.82
	LF	2.94 (3.72)		3.98 (0.32)		1.17 (0.45)		—	
Diabetes	Mean	79.60	75.76	78.67	75.01	79.58	**76.36**	79.06	76.03
(768 × 8)	STD	0.66	4.63	1.55	4.68	0.62	4.64	0.82	4.01
	LF	4.17 (0.57)		3.28 (1.34)		4.14 (0.38)		—	
Ionosphere	Mean	99.18	94.11	99.45	**94.80**	98.99	**94.66**	97.94	94.14
(351 × 34)	STD	0.45	3.49	0.51	3.60	0.53	3.51	0.88	3.50
	LF	8.73 (2.56)		8.02 (3.71)		5.85 (1.42)		—	
Liver	Mean	76.45	72.62	76.03	72.26	76.54	72.94	75.90	73.26
(345 × 6)	STD	1.22	7.09	1.10	6.06	0.90	6.74	1.03	6.92
	LF	5.56 (1.48)		5.76 (1.13)		5.62 (0.65)		—	
WBC	Mean	98.80	97.88	99.45	97.37	98.87	97.70	98.67	97.95
(569 × 30)	STD	0.20	1.81	0.53	2.07	0.30	1.90	0.21	1.78
	LF	10.53 (2.05)		8.89 (4.35)		5.87 (0.92)		—	

Table 4. Classification results for kernel models for quadratic, exponential, and logistic loss functions. Data sizes ($m \times n$) are also shown.

results show that much fewer latent features than PCA latent features are required to achieve comparable accuracies. The experiments for kernel models in Table 4 are repeated using the principal component from PCA instead of the latent factors from BLF. For the least squares loss, this becomes the standard principal component regression algorithm. In terms of generalization error, the methods with PCA and BLF features were very similar: the paired t-test resulted in 2 wins (significantly better results) for BLF, 2 wins for PCA, and 8 ties for Cancer, Diabetes, Ionosphere, and Liver. For Wisconsin Breast Cancer, BLF wins for all the loss functions. Significantly fewer boosted latent features than principal components were required to achieve similar/slightly better performance. Figure 4 plots the number of latent factors used for BLF and PCA. PCA always required more features.

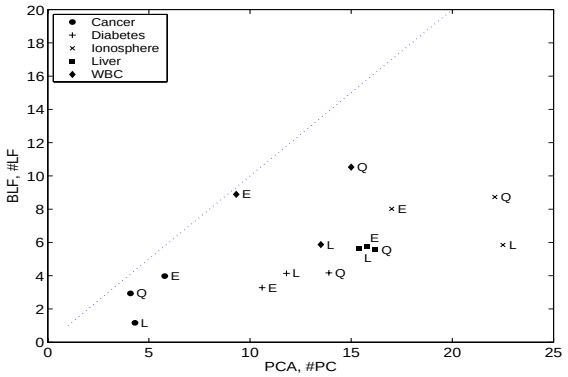

Fig. 4. Number of the principal components and latent factors for kernel classification models. Quadratic (Q), exponential (E), and logistic (L) loss functions are used for both BLF and PCA. PCA requires more dimensions to obtain similar performance to BLF.

8.2 Case Study on High Dimensional Data

We use a very high dimensional data set, *Thrombin* from KDD Cup 2001, to illustrate BLF as a feature construction method. *Thrombin* is a classification problem to predict a compound's binding activity (active/inactive) to Thrombin. The original training/test split for the 2001 KDD Cup is very challenging: 42 active and 1867 inactive compounds (actives to inactives ratio is about 2.25%) with 139,351 binary features. However the test set is more balanced: 150 actives and 484 inactives (31%). In order to slightly reduce the difficulty in the training set, the original training and test sets are merged and randomly divided so that the new training/test split is 100 actives and 330 inactives. This gives the active/inactive ratio 4.55% in the new training split and 30% in the new test split. We repeat the experiment 100 times to compare models. Note that all 100 experiments use the same traininig/test splits. Because of the unbalanced class distribution, we adopt a local weighting γ_i for each data point $\mathbf{x}_i$: $\gamma_i = 1/|\mathcal{C}^+|$, $\forall i \in \mathcal{C}^+$. $\gamma_i = 1/|\mathcal{C}^-|$, $\forall i \in \mathcal{C}^-$. This weighting has been previously used in [3]. For numerical stability in the weighted logistic loss function, the modified Newton step in equation (42) with $\lambda = 0.1$ was used to optimize the function coefficients in Step 6 of Algorithm 3.

In order to compare the quality of the orthogonal latent features created by different loss functions in BLF, SVM models are created using the BLF features. The quality of the features is evaluated by the performance of SVM on the test set. BLF with logistic loss and squared loss (BLF-LS) are used for constructing the features. Since our goal is to show BLF is an effective feature construction method, the full 139,351 binary features, without any preprocessing, are used for the input space of the BLF models. Figure 5 shows the area under ROC curve (AUC) for BLF-LOG and BLF-LS, as a function of the number of latent factors. As seen in the previous section, logistic loss fits the data faster than squared loss: the "peak" appears at a smaller number of latent factors in logistic loss than for squared loss. We pick 5 and 15 as the number of orthogonal features that are used in the next stage of SVM

experiments. Obviously, five features is before the peak and the dimensionality may have been reduced too much to have good predictive ability. But for the case with 15 latent factors, the curves for both squared loss and logistic loss seem to be stabilized and we can expect better predictability. We use the nonlinear orthogonal features,

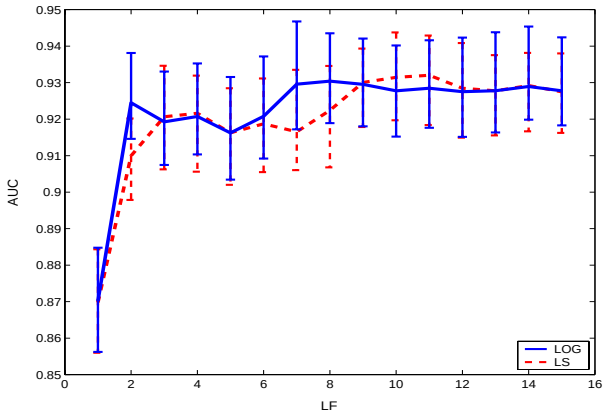

Fig. 5. Median of area under ROC curve with respect to the number of latent factors for BLF with logistic loss and squared loss. The error bars correspond to 1/4-th and 3/4-th order statistics. The statistics are based on 100 randomized experiments.

$\boldsymbol{T}\mathrm{diag}(\mathbf{c})$ as in Equation (7), as the input data to a classic linear SVM [8]. Since features in $\boldsymbol{T}$ are normalized to have length one, we need to use feature weighting $\mathrm{diag}(\mathbf{c})$ for better performance especially for the relatively small dimensional space used here. Figure 6 illustrates the AUC for SVM models with a wide spectrum of the cost parameter C in the SVM. As baseline cases, results of SVM models trained with the full data set are also shown in the figure. SVM is given a better metric space trained by BLF so it can overfit as the value C gets large. However, the SVM models that use the original features stay at almost the same value of AUC within the value of C shown in Figure 6. As seen in Figure 5, at LF=5 and 15, BLF-LOG and BLF-LS have very similar performance. However, after training by SVM, the features from BLF-LOG are slightly better than those from BLF-LS. Overall, with a reasonably good choice of parameter C, SVM using reduced features by BLF can improve models with the original features. Further, features created by logistic loss perform slightly better than those constructed using the squared loss over a wide range of C.

9 Conclusion

In this chapter, a framework for constructing orthogonal features by boosting, OrthoAnyBoost, was proposed. Using techniques from spectral methods such as PCA

and PLS, OrthoAnyboost can be very efficiently implemented in linear hypothesis spaces. The resulting method, BLF, was demonstrated to both construct valuable orthogonal features and to be a competitive predictive method by itself for a variety of loss functions. BLF performs feature construction based on a given (sub)differentiable loss function. For the least squared loss, BLF reduces to PLS and preserves all the attractive properties of that algorithm. As in PCA and PLS, the resulting nonlinear features are valuable for visualization, dimensionality reduction, improving generalization, and use in other learning algorithms, but now these features can be targeted to a specific inference task. The data matrix is factorized by the extracted features. The low-rank approximation of the data matrix provides efficiency and stability in computation. The orthogonality properties of BLF guarantee that it converges to the optimal solution of the full model in a finite number of iterations. Empirically, orthogonality makes BLF converge much faster than gradient boosting. The predictive model is constructed in a reduced dimensionality space thus providing capacity control leading to good generalization. The method is generalized to nonlinear hypotheses using kernels.

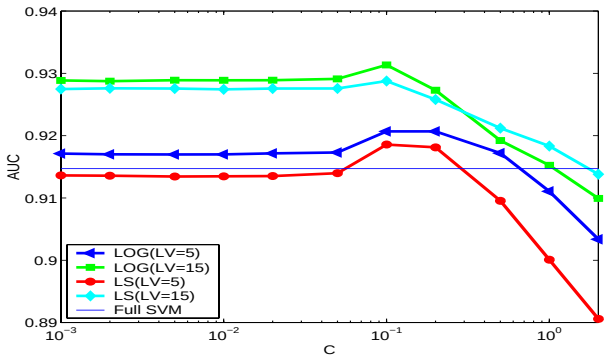

Fig. 6. Area under ROC curve for SVM models using features created by BLF with squared loss and logistic loss for LF=5 and 15. The dotted line shows SVM models using the original full features.

Computational results demonstrate how BLF can be applied to a wide range of commonly used loss functions. The results illustrate differences in loss functions. As always, the best loss function depends on the data and inference task. The least absolute deviation is more robust than the squared loss and the version of BLF using the LAD loss showed some improvements for drug discovery data where attributes are inter-correlated and noisy. Classification loss functions such as exponential loss and logistic loss are more natural for classification problems and the loss functions can be weighted to handle problems with unbalanced data or unequal misclassification costs. Future work is need to investigate the theoretical properties of BLF from both optimization and learning points of view and to apply the approach to other learning tasks/loss functions such as ranking.

References

1. M. Bazaraa, H. Sherali, and C. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, 1993.
2. K. P. Bennett, C. M. Breneman, and M. J. Embrechts. DDASSL project, 2000. `http://www.drugmining.com`.
3. K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.
4. C. L. Blake and C. J. Merz. UCI Repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
5. M. Borga, T. Landelius, and H. Knutsson. A unified approach to PCA, PLS, MLR and CCA. Report LiTH-ISY-R-1992, ISY, SE-581 83 Linköping, Sweden, November 1997.
6. Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.
7. S. Chen. Local regularization assisted orthogonal least squares regression, 2003. `http://www.ecs.soton.ac.uk/~sqc/`.
8. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
9. H. Drucker, C. Burges, L. Kaufman, A. Smola, , and V. Vapnik. Support vector regression machines.
10. Y. Freund and R. E. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.
11. J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, (28):337–307, 2000.
12. J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.
13. Jerome H. Friedman and Bogdan E. Popescu. Gradient directed regularization of linear regression and classification. Technical report, Stanford University, 2004.
14. D. C. Hoaglin, F. Mosteller, and J. W. Tukey. *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, 1982.
15. A. Höskuldsson. PLS regression methods. *Journal of Chemometrics*, 2:211–228, 1988.
16. T. Joachims. Making large-scale svm learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. 1999.
17. Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1992.
18. John Lafferty. Additive models, boosting, and inference for generalized divergences. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 125–133, New York, NY, USA, 1999. ACM Press.
19. L. Li, Y. S. Abu-Mostafa, and A. Pratap. Cgboost:conjugate gradient in function space. Technical Report Computer Science Technical Report CaltechCSTR:2003.007, CalTech, 2003.
20. Edward Malthouse, Ajit Tamhane, and Richard Mah. Nonlinear partial least squares. *Computers in Chemical Engineering*, 12(8):875–890, 1997.

21. Brian Marx. Iteratively reweighted partial least squares for generalized linear models. *Technometrics*, 38(4):374–381, 1996.
22. L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space. Technical report, RSISE, Australian National University, 1999.
23. P. B. Nair, A. Choudhury, and A. J. Keane. Some greedy learning algorithms for sparse regression and classification with mercer kernels. *Journal of Machine Learning Research*, 3:781–801, 2002.
24. Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, 1999.
25. A. Phatak and F. de Hoog. Exploiting the connection between PLS, lanczos, and conjugate gradients: Alternative proofs of some properties of PLS. *Journal of Chemometrics*, 16:361–367, 2002.
26. R. Rosipal and L. T. Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. *Journal of Machine Learning Research*, 2:97–123, 2001.
27. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297–336, 1999.
28. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
29. H. Wold. Estimation of principal components and related models by iterative least squares. In *Multivariate Analysis*, pages 391–420, New York, 1966. Academic Press.
30. J. Zhu and T. Hastie. Kernel logistic regression and the import vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1081–1088, Cambridge, MA, 2002. MIT Press.