Greedy Convex Ensemble

Thanh Tan Nguyen\(^1\), Nan Ye\(^2\) and Peter Bartlett\(^3\)

\(^1\)Queensland University of Technology
\(^2\)The University of Queensland
\(^3\)UC Berkeley

tan1889@gmail.com, nan.ye@uq.edu.au, bartlett@cs.berkeley.edu

Abstract

We consider learning a convex combination of basis models, and present some new theoretical and empirical results that demonstrate the effectiveness of a greedy approach. Theoretically, we first consider whether we can use linear, instead of convex, combinations, and obtain generalization results similar to existing ones for learning from a convex hull. We obtain a negative result that even the linear hull of very simple basis functions can have unbounded capacity, and is thus prone to overfitting; on the other hand, convex hulls are still rich but have bounded capacities. Secondly, we obtain a generalization bound for a general class of Lipschitz loss functions. Empirically, we first discuss how a convex combination can be greedily learned with early stopping, and how a convex combination can be non-greedyly learned when the number of basis models is known a priori. Our experiments suggest that the greedy scheme is competitive with or better than several baselines, including boosting and random forests. The greedy algorithm requires little effort in hyperparameter tuning, and also seems able to adapt to the underlying complexity of the problem. Our code is available at https://github.com/tan1889/gce.

1 Introduction

Various machine learning methods combine given basis models to form richer models that can represent more complex input-output relationships. These include random forests [Breiman, 2001] and boosting [Freund and Schapire, 1995; Mason et al., 2000a; Chen and Guestrin, 2016], which have often been found to work well in domains with good features. Interestingly, even combining simple basis models like decision stumps can work very well on hard problems [Viola and Jones, 2004].

In this paper, we consider learning an optimal convex combination of basis models [Lee et al., 1996; Mannor et al., 2003; Oglic and Gärtner, 2016; Wyner et al., 2017], and present new theoretical and empirical insights.

We first compare learning from convex hulls with learning from the closely related linear hulls. Learning from a convex hull can be seen as a regularized version of learning from the corresponding linear hull, where we enforce constraints on the weights of the basis functions. While linear hulls are known to provide universal approximations [Barron, 1993; Makovoz, 1996], our analysis shows that they can be prone to overfitting. Specifically, we show that the capacity of the linear hull of very simple functions can be unbounded, while the convex hull is still rich but has bounded capacity.

Our second contribution is a generalization result for a general class of Lipschitz loss functions. A number of works studied algorithms for learning a convex combination and analyzed their generalization performance. However, previous works mostly focused on generalization performance with quadratic loss [Lee et al., 1996; Mannor et al., 2003] or large margin type analysis [Koltchinskii and Panchenko, 2005] for classification problems. The quadratic loss is a special case of the class of Lipschitz loss functions considered in this paper. In addition, our result shows that we can obtain an \(O(1/\sqrt{n})\) convergence rate for log-loss in the classification setting.

Empirically, we present an extensive experimental evaluation of algorithms for learning from convex hulls. While previous works mainly focused on simple greedy algorithms to learn a convex combination, we leverage on the functional optimization versions of some sophisticated algorithms to develop algorithms for learning a convex combination. In particular, we consider the Frank-Wolfe (FW) algorithm and its variants [Jaggi, 2013], which provide natural ways to build convex combinations. We also show how a convex combination can be non-greedly learned when the number of basis functions is known a priori. Our experiments suggest that the greedy scheme is competitive with or better than several baselines, including boosting and random forests. The greedy algorithm requires little hyper-parameter tuning, and also seems to adapt to the underlying complexity of the problem.

Section 2 further discusses related works. Section 3 presents our theoretical analysis for learning from a convex hull. Section 4 discusses some greedy learning algorithms, and a non-greedy version. Section 5 empirically compares the algorithms for learning from convex hulls, and a few baselines. Section 6 concludes the paper.

2 Related Work

A number of works have studied the generalization performance of algorithms for learning convex combinations. Lee et al. [1996] considered learning a convex combination of
linear threshold units with bounded fan-in (#inputs) for binary classification using quadratic loss, and they showed that an optimal convex combination is PAC-learnable. Mannor et al. [2003] also considered binary classification, and obtained a generalization result for general basis functions and quadratic loss. They also obtained a consistency result for more general loss functions. Koltchinskii and Panchenko [2005] provided some generalization results for learning a convex combination by maximizing margin. Oglic and Gärtner [2016] considered regression with quadratic loss and presented a generalization analysis for learning a convex combination of cosine ridge functions. We obtained generalization bounds for a class of Lipschitz loss functions and general basis functions.

Various authors considered greedy approaches for learning a convex combination, which iteratively constructs a convex combination by choosing a good convex combination of the previous convex combination and a new basis function. Jones [1992] presented a greedy algorithm and showed that it converges at $O(1/k)$ rate for quadratic loss. This is further developed by Lee et al. [1996] and Mannor et al. [2003]. Zhang [2003] generalized these works to convex functionals. We leverage on the functional versions of more sophisticated greedy optimization algorithms; in particular, the FW algorithm and its variants, which have recently attracted significant attention in the numerical optimization literature [Jaggi, 2013]. Recently, Bach [2017] considered using the FW algorithm to learn neural networks with non-Euclidean regularizations, and showed that the sub-problems can be NP-hard. Besides greedy optimization approaches, we show how to non-greedily learn a convex combination given the number of basis functions. We empirically compared the effectiveness of these algorithms.

Works on random forests [Breiman, 2001] and boosting [Mason et al., 2000b] are also closely related. A random forest can be viewed as a convex combination of trees independently trained on bootstrap samples, where the trees have equal weights. Boosting algorithms greedily construct a conic, instead of convex, combination of basis functions, but for binary classification, a conic combination can be converted to a convex combination without changing the predictions. There are numerous related works on the generalization performance of boosting (e.g. see [Bartlett and Traskin, 2007; Schapire, 2013; Gao and Zhou, 2013]). Random forests are still less well understood theoretically yet [Wyner et al., 2017], and analysis can require unnatural assumptions [Wager and Walther, 2015]. We empirically compared algorithms for learning a convex combination with random forests and boosting.

There have been also several recent applications of boosting for generative models. Specifically, Locatello et al. [2018] show that boosting variational inference satisfies a relaxed smoothness assumption which is sufficient for the convergence of the functional Frank-Wolfe algorithm; Grover and Ermon [2018] consider Bayes optimal classification; and Tolstikhin et al. [2017] propose AdaGAN, which is adapted from AdaBoost for Generative Adversarial Networks. Our work is orthogonal to these works in the sense that we study discriminative models and learning from a convex hull, instead of a linear or a conic hull. Moreover, our bound might be interesting in comparison to vacuous bounds that grow rapidly in the number of parameters: while the number of parameters for a convex combination can be unbounded, our error bound depends only on the pseudodimension of the basis models.

3 Theoretical Analysis

Given an i.i.d. sample $z = ((x_1, y_1), \ldots, (x_n, y_n))$ drawn from a distribution $P(X, Y)$ defined on $\mathcal{X} \times \mathcal{Y} \subseteq \mathcal{X} \times \mathcal{R}$, we want to learn a function $f$ to minimize the risk $R(f) = \mathbb{E}_P(L(Y, f(X)))$, where $L(y, \hat{y})$ is the loss that $f$ incurs when predicting $y$ as $\hat{y}$, and the expectation is taken w.r.t $P$. The empirical risk of $f$ is $R_n(f) = \mathbb{E}_nL(Y, f(X)) = \frac{1}{n} \sum_{i=1}^n L(y_i, f(x_i))$. Without loss of generality, we assume $\mathcal{X} \subseteq \mathcal{R}^d$.

Given a class of basis functions $\mathcal{G} \subseteq \mathcal{Y}^X$, we use $\text{co}_k(\mathcal{G})$ to denote the set of convex combinations of $k$ functions in $\mathcal{G}$, that is, $\text{co}_k(\mathcal{G}) = \{ \sum_{i=1}^k \alpha_i g_i : \sum_{i} \alpha_i = 1, each \alpha_i \geq 0, each g_i \in \mathcal{G} \}$. The convex hull of $\mathcal{G}$ is $\text{co}(\mathcal{G}) = \cup_{k \geq 1} \text{co}_k(\mathcal{G})$. We will also use $\text{lin}_k(\mathcal{G})$ to denote the set of linear combinations of $k$ functions in $\mathcal{G}$, that is, $\text{lin}_k(\mathcal{G}) = \{ \sum_{i=1}^k \alpha_i g_i : \alpha_1, \ldots, \alpha_k \in \mathcal{R}, g_1, \ldots, g_k \in \mathcal{G} \}$. The linear hull of $\mathcal{G}$ is $\text{lin}(\mathcal{G}) = \cup_{k \geq 1} \text{lin}_k(\mathcal{G})$. The basis functions are assumed to be bounded, with $\mathcal{G} \subseteq \mathcal{Y}^X \subseteq [-B, B]^X$ for some constant $B > 0$.

Capacity measures. A function class needs to be rich to be able to fit observed data, but cannot be too rich so as to make generalization possible, that is, it needs to have the right capacity. Commonly used capacity measures include VC-dimension, pseudodimension, and Rademacher complexity.

VC-dimension is defined for binary valued functions. Specifically, for a class $\mathcal{F}$ of binary valued functions, its VC-dimension $d_{VC}(\mathcal{F})$ is the largest $m$ such that there exists $m$ examples $x_1, \ldots, x_m$ such that the restriction of $\mathcal{F}$ to these examples contains $2^m$ functions. Equivalently, for any $y_1, \ldots, y_m \in \{0, 1\}$, there is a function $f \in \mathcal{F}$ such that $f(x_i) = y_i$ for all $i$. $x_1, \ldots, x_m$ is said to be shattered by $\mathcal{F}$.

Pseudodimension [Pollard, 1984] is a generalization of VC-dimension to real-valued functions. The pseudodimension $d_{p}(\mathcal{F})$ of a class of real-valued functions $\mathcal{F}$ is defined as the maximum number $m$ such that there exists $m$ inputs $x_1, \ldots, x_m \in \mathcal{X}$, and thresholds $t_1, \ldots, t_m \in \mathcal{R}$ satisfying $\{ \{f(x_1) \geq t_1\}, \ldots, \{f(x_m) \geq t_m\} : f \in \mathcal{F} \} = \{0, 1\}^m$. If each $f \in \mathcal{F}$ is binary-valued, then $d_{p}(\mathcal{F}) = d_{VC}(\mathcal{F})$.

Rademacher complexity is defined as $\mathbb{E} \sup_{f \in \mathcal{F}} \mathbb{R}_n(f)$, where $\mathbb{R}_n$ is the Rademacher process defined by $\sum_{i} \epsilon_i f(x_i, y_i)$, with $(x_i, y_i)$’s being an i.i.d. sample, and $\epsilon_i$’s being independent Rademacher random variables (i.e., they have probability 0.5 to be -1 and 1). Expectation is taken w.r.t both the random sample and the Rademacher variables.

We refer the readers to the book of Anthony and Bartlett [2009] and the article of Mendelson [2003] for excellent discussions on these capacity measures and their applications in generalization results.

3.1 A Regularization Perspective

Several authors showed that linear hulls of various basis functions are universal approximators [Barron, 1993; Makovoz, 1996]. Naturally, one would like to learn using linear hulls if possible. On the other hand, the richness of the
Assume that $\text{lin}(T)$ is $\ell_2$ regularized in the sense that $\ell_2$ regularization constrained the weights to be inside an $\ell_2$ ball, while here we constrain the weights of the basis model to be inside a simplex. A key difference is that standard $\ell_2$ regularization is often applied to a parametric model with fixed number of parameters, but here the number of parameters can be infinite.

We compare the capacities of the linear hull and the convex hull of a class of basis functions $G$ with finite pseudodimension, and demonstrate the effect of the regularizer $I_\infty$ in controlling the capacity: while the convex hull can still be rich, it has a more adequate capacity for generalization.

For a class of functions $\mathcal{F}$, we shall use $\text{bin}(\mathcal{F}) = \{x \mapsto I(f(x) \geq t) : f \in \mathcal{F}, t \in \mathbb{R}\}$ to denote the thresholded binary version of $\mathcal{F}$. Consider the set of threshold linear functions $T = \{\langle \theta, x \rangle \geq t : \theta \in \mathbb{R}^d, t \in \mathbb{R}\}$. It is well-known that the VC-dimension of the thresholded versions of $k$ linear threshold functions can grow quickly.

**Proposition 1.** ([Anthony and Bartlett, 2009], Theorem 6.4) The VC-dimension of $\text{bin}(\text{lin}_k(T))$ is at least $\frac{dk}{\delta} \log_2 \left(\frac{\delta}{\epsilon}\right)$ for $d > 3$ and $k \geq 2d/2$.

The above result implies that $d_P(\text{bin}(\text{lin}_k(T)))$ is at least $\frac{dk}{\delta} \log_2 \left(\frac{\delta}{\epsilon}\right)$. A natural question is whether the VC-dimension still grows linearly when $k > 2d/2$. We give an affirmative answer via a constructive proof, and provide counterexamples for the convex hull.

**Proposition 2.** (a) Assume that $d \geq 2$. Then $d_{VC}(\text{bin}(\text{lin}_k(T))) \geq k$, thus $d_P(\text{bin}(\text{lin}_k(T))) \geq k$, and $d_P(\text{lin}(T)) = \infty$. In addition, the Rademacher complexity of $\text{bin}(\text{lin}(T))$ is infinite.

(b) Assume that $d \geq 2$. Then $d_{VC}(\text{bin}(\text{co}_{k+1}(T))) \geq k$, thus $d_P(\text{co}_{k+1}(T)) \geq k$, and $d_P(\text{co}(T)) = \infty$, but the Rademacher complexity of $\text{co}(\text{lin}(T))$ is finite.

**Proof.** (a) Consider an arbitrary unit circle centered at the origin, and any $k$ points $x_1, \ldots, x_k$ which are equally spaced on the circle. Let $\theta_i = x_i$ and $b_i = 1$ for $i = 1, \ldots, k$. For any $y_1, \ldots, y_k \in \{0, 1\}$, consider the linear combination $f(x) = \sum_{i=1}^k w_i I(\theta_i^\top x \geq b_i)$, with $w_i = y_i$. We have $f(x_i) = y_i$. The classifier $t(x) = \mathbb{I}(f(x) \geq 1)$ is a thresholded classifier obtained from $f$, and thus $t \in \text{bin}(\text{lin}_k(T))$. In addition, $t(x) = I(y_i \geq 1) = y_i$. In short, for any $y_1, \ldots, y_k \in \{0, 1\}$, there is a classifier $t \in \text{bin}(\text{lin}_k(T))$ such that $t(x_i) = y_i$. Thus $d_{VC}(\text{bin}(\text{lin}_k(T))) \geq k$. It follows that $d_P(\text{lin}_k(T)) \geq k$, and thus $d_P(\text{lin}(T)) = \infty$.

The Rademacher complexity of $\text{lin}(T)$ is infinite, because for any $c > 0$, the Rademacher complexity of $c \text{lin}(T)$ is $c$ times that of $\text{lin}(T)$. On the other hand, $c \text{lin}(T) = \text{lin}(T)$. Hence the Rademacher complexity of $\text{lin}(T)$ can be arbitrarily large, and is thus infinite.

(b) Consider the function $h(x) = w_0 I(0.5 \leq x) + \sum_{i=1}^k w_i I(\theta_i^\top x \geq b_i)$, where $w_0 = 1/(1+\sum_i y_i)$ and $w_i = y_i/(1+\sum_i y_i)$ for $i \geq 1$. The function $h(x)$ is a convex combination of $I(0.5 \geq x)$, $I(\theta_i^\top x \geq b_i)$, where the first one is always 0. For any $x_i$, we have $h(x_i) = y_i/(1+\sum_i y_i) y_i/(k+1)$, because each $y_i$ is either 0 or 1. Hence we have $I(h(x_i) \geq 1/(k+1)) = y_i$. It follows that $x_1, \ldots, x_k$ can be shattered by the thresholded version of $\text{co}_{k+1}(T)$.

The Rademacher complexity of the convex hull is equal to that of $T$ according to Theorem 2.25 in [Mendelson, 2003], which is finite as $d_{VC}(T) = d + 1$ is finite.

Proposition 2 shows that the linear hull has infinite capacity, both in terms of pseudodimension and in terms of Rademacher complexity. Thus, it may easily overfit a training dataset. On the other hand, the convex hull is more restricted with a finite Rademacher complexity, but still rich because it has infinite pseudodimension. This can be attributed to regularization effect imposed by the convex coefficients constraints.

### 3.2 Generalization Error Bounds

Let $f^*(x) = \min_{y \in \mathcal{Y}} E[L(y, f(x))|x = x]$ be the Bayes optimal function for binary classification problems (that is, $\mathcal{Y} = \{-1, 1\}$) using the quadratic loss $L(y, f(x)) = (f(x) - y)^2$. Mannor et al. [2003] obtained the following uniform convergence rate with an assumption on the uniform entropy $H(\epsilon, \text{co}(G), n)$ of $\text{co}(G)$, where $G \subseteq [-B, B]^d$.

**Theorem 1.** (Theorem 9 in [Mannor et al., 2003]) Assume that for all positive $\epsilon$, $H(\epsilon, \text{co}(G), n) \leq K(2B/\epsilon)^{2c}$ for $\epsilon \in (0, 1)$ and $K > 0$. Then there exist constants $c_0, c_1 > 0$ that depend on $\xi$ and $K$ only, such that $\delta \geq c_0$, with probability at least $1 - e^{-\delta}$, for all $f \in \text{co}(G)$,

$$R(f) - R(f^*) \leq 4(R_n(f) - R_n(f^*)) + \frac{c_0 4B^2}{\xi} \left(\frac{\delta}{n}\right)^{1/(1+\xi)}.$$

When $d_P(G) = p$, then the assumption on the metric entropy $H(\epsilon, \text{co}(G), n)$ is satisfied with $\xi = \frac{p}{p-2}$ [Weller and Song, 2002]. In Theorem 2, we prove a more general bound for a class of Lipschitz losses that includes the quadratic loss considered in Theorem 1 as a special case. Omitted proofs are available in the appendix.

**Theorem 2.** Assume that $d_P(G) = p < \infty$, and $L(y, f(x)) = \phi(f(x) - y)$ for a $c_\phi$-Lipschitz nonnegative function $\phi$ satisfying $\phi(0) = 0$. With probability at least $1 - \delta$, for all $f \in \text{co}(G)$, $R(f) - R(f^*) \leq R_n(f) - R_n(f^*) + \frac{\sqrt{2c_\phi B \left(2\ln(1/\delta) + D\sqrt{p} + 2\right) + D}}{n}$, where $c = 2c_\phi B \left(2\ln(1/\delta) + D\sqrt{p} + 2\right)$, and $D$ is an absolute constant.

The Bayes optimal function $f^*$ is generally not in $\text{co}(G)$, thus the chosen convex combination $f$ may not reach the level of performance of $f^*$. Thus we are often interested in the convergence of the empirical minimizer to the optimal model in $\text{co}(G)$. We can obtain an $O(1/\sqrt{n})$ convergence rate by closely following the proof of Theorem 2.

**Theorem 3.** Assume that $d_P(G) = p < \infty$, $L(y, f(x)) = \phi(f(x) - y)$ for a $c_\phi$-Lipschitz nonnegative function $\phi$ satisfying $\phi(0) = 0$. Let $\hat{f} = \arg\min_{f \in \text{co}(G)} R_n(f)$, and $h^* = \arg\min_{f \in \text{co}(G)} R(f)$, then with probability
at least $1 - \delta$, $R(\hat{f}) \leq R(h^*) + \frac{\sqrt{c}}{\sqrt{n}}$, where $c = 2c_0B\left(\sqrt{2\ln(1/\delta)} + D\sqrt{p} + 2\right)$.

As a special case, we have the result below for $\ell_p$ regression.

**Corollary 1.** For $L(y, f(x)) = |f(x) - y|^q$, $q \geq 1$, the bounds in Theorem 2 and 3 hold with $c_0 = q(2B)^q - 1$.

Donahue et al. [1997] showed that tighter bounds can be obtained for $\ell_p$ regression by exploiting the specific form of $\ell_p$ loss. Our analysis provides a looser bound, but is simpler and can be applied to the classification setting below.

Specifically, for binary classification with $Y = \{-1, 1\}$, we can also obtain an $O(1/\sqrt{n})$ generalization bound for a class of Lipschitz loss as a corollary of the proof of Theorem 2. The loss in this case is Lipschitz in $yf(x)$ (not $f(x) - y$ as in the regression case), with a positive value indicating that $f(x)$ is better aligned with $y$.

**Corollary 2.** Assume that $d_p(G) = p < \infty$, $Y = \{-1, 1\}$, $L(y, f(x)) = \phi(yf(x))$ for a $c_0$-Lipschitz non-negative function $\phi$ satisfying $\phi(0) = 0$. Let $\hat{f} = \arg\min_{f \in co(G)} R_n(f)$, and $h^* = \arg\min_{f \in co(G)} R(f)$, then with probability at least $1 - \delta$, $R(\hat{f}) \leq R(h^*) + \frac{c}{\sqrt{n}}$, where $c = 2c_0B\left(\sqrt{2\ln(1/\delta)} + D\sqrt{p} + 2\right)$.

As a special case, the above rate holds for the log-loss.

**Corollary 3.** When $y \in \{-1, 1\}$, $L(y, f(x)) = -\ln \frac{1}{1 + e^{-yf(x)}}$, the bound in Corollary 2 holds with $c_0 = 1$.

### 4 Algorithms

We consider algorithms for finding the empirical risk minimizer $\hat{f} = \arg\min_{f \in co(G)} R_n(f)$, when $G$ consists of parametric basis models, i.e., $G = \{g_\theta : \theta \in \mathbb{R}^p\}$, where $g_\theta$ denotes a model with parameters $\theta$.

#### 4.1 Greedy Algorithms

The convexity of $co(G)$ allows the following greedy scheme. Start with some $f_0 \in H$. At iteration $k$, we choose appropriate $\alpha_t \in [0, 1]$ and $g_t \in G$, for the new convex combination

$$f_{t+1} = (1 - \alpha_t)f_t + \alpha_t g_t.$$  \hfill (1)

We run the algorithm up to a maximum number of iterations $T$, or do early stopping if the improvements in the last few iterations is negligible (less than a small threshold).

Such scheme generates sparse solutions in the sense that at iteration $t$, the convex combination consists of at most $t$ basis functions, even though the optimal combination can include arbitrarily large number of basis functions.

We present several instantiations of this scheme based on results from functional optimization. The derived sub-problems, while being functional optimization problems, are equivalent to finite-dimensional numerical optimizations which can be solved using stochastic gradient descent. Some of them have interesting forms that differ from standard risk minimization problems.

A **nonlinear greedy algorithm.** One natural way to choose $g_t$ and $\alpha_t$ is to choose them jointly so as to maximize the decrease in the empirical risk [Jones, 1992; Lee et al., 1996; Mannor et al., 2003; Zhang, 2003]. Specifically,

$$\theta_t, \alpha_t \leftarrow \arg\min_{\theta \in \mathbb{R}^p, \alpha \in [0, 1]} R_n((1 - \alpha)f_{t-1} + \alpha g_t)$$  \hfill (2)

For common loss functions, the RHS is usually a differentiable function of $\theta$ and $\alpha$, and thus the problem can be solved using first-order methods. We used Adam [Kingma and Ba, 2014] in our experiments.

When $L(y, f(x))$ is convex and smooth functional of $f$, it is known, e.g. from [Zhang, 2003], that $R_n(f_t) - R_n(\hat{f}) \leq O(1/t)$. In fact, we can still achieve a convergence rate of $O(1/t)$, as long as we can solve the greedy step with an error of $O(1/t^2)$, that is, if we can choose $g_t$ and $\alpha_t$ such that $R_n((1 - \alpha_t)f_{t-1} + \alpha_t g_t) \leq \min_{g \in G, \alpha \in [0, 1]} R_n((1 - \alpha)f_{t-1} + \alpha g) + \frac{c}{t^2}$, for some constant $c > 0$ [Zhang, 2003]. In particular, this result applies to the quadratic loss.

**The FW algorithm.** The FW algorithm [Frank and Wolfe, 1956] does not choose $g_t$ to directly minimize the risk functional at each iteration, but chooses it by solving a linear functional minimization problem

$$g_t = \arg\min_{g \in G} DR_n(f_{t-1}, g).$$  \hfill (3)

where the step size $\alpha_t$ can be taken as $\alpha_t = \frac{1}{t+1}$ or chosen using line search, and $DR_n(f)$ denotes the functional gradient of $R_n$ with respect to $f$, which is only non-zero at the points in the sample and is thus finite. For the quadratic loss $L(y, f(x)) = (f(x) - y)^2$, $g_t$ is $g_{y_t}$ with $\theta_t$ chosen by

$$\theta_t = \arg\min_{\theta \in \mathbb{R}^p} \sum_i 2(f_{t-1}(x_i) - y_i)g_{\theta}(x_i).$$  \hfill (4)

This optimization problem has an interesting form different from standard risk minimization problems. For quadratic loss, we can also derive the closed form solution for the line-search for $\alpha_t$. For other risk criteria, there is no closed form solution, and we treat that as a parameter in the numerical optimization problem in each iteration. FW also converges at an $O(1/t)$ rate as for smooth and convex functionals (e.g., see Jaggi [2013]), with the constant in the rate being of the same order as that for the nonlinear greedy algorithm.

**Away-step and Pairwise FW.** The away-step Frank-Wolfe (AFW) [Guélat and Marcotte, 1986], and the pairwise Frank-Wolfe (PFW) [Lacoste-Julien and Jaggi, 2015] are faster variants which can converge at a linear rate when the solution is not at the boundary.

AFW either takes a standard FW step or an away step which removes a basis network from current convex combination and redistributes the weight to remaining basis networks. Specifically, at each iteration, it finds $g_t \in \hat{G}$ that is most aligned with the negative gradient $DR_n(f_{t-1})$ as in the FW algorithm, and a basis function $a_t$ that is most misaligned with the negative gradient $DR_n(f_{t-1})$ from the set of basis functions in $f_{t-1}$. Here, the inner product of two vectors measures the degree of alignment between them. It then constructs a FW direction $d_t^{AFW}$ that moves towards $g_t$, and an away-step direction $d_t^{AW}$ that moves away from $a_t$. The direction that is better aligned with the negative gradient is taken. For the away-step, the step size
is restricted to be in \( [0, \frac{\alpha_i}{1-\alpha_i}] \) so that the weight of \( a_t \) remains non-negative in \( f_t \).

PFW swaps the weight of \( a_t \) and \( g_t \) determined in AFW by moving along the direction \( g_t - a_t \). Line search is used to determine the optimal step size.

### 4.2 A Non-greedy Algorithm

If we know the number of basis models required a priori, we can train the weights of the basis models and the convex coefficients simultaneously. Instead of using constrained optimization techniques, we can use a softmax normalization. However, we found this not working well in practice.

We propose a simple unconstrained parametrization of the convex coefficients that have been observed to perform well in our experiments. Specifically, if we know that the number of basis model is \( k \), we reparameterize the convex coefficients \( \alpha_1, \ldots, \alpha_k \) as a function of the unconstrained parameter vector \( v \in \mathbb{R}^k \) with \( \alpha_t = \frac{1}{1+\sum_{i=1}^k |v_i|} \). The model \( \sum_t \alpha_t g_t(x) \) can be seen as a neural network that can be trained conventionally.

### 4.3 Implementation

In practice, a lot of interesting functions are unbounded, and we may need multiple output.

We can convert unbounded functions to bounded ones by using the scaled hard tanh \( \text{hardtanh}(y) = B \max(-1, \min(y, 1)) \) to clamp the output to \([-B, B]\). Sometimes it is beneficial to choose the scaling factor \( B \) to be larger than the actual possible range: when \( G \) contains the zero function, \( G \) is a larger set when \( B \) is larger.

When multiple outputs are needed, we can train a model for each output separately. If the convex hull contains the true input-output function for each output channel, then the generalization theory for the case of single output guarantees that we will learn all the input-output functions eventually.

### 5 Experiments

We choose \( g_0 \) as a small neural network in our experiments. We compare the performance of the greedy algorithms (nicknamed as GCE, which stands for greedy convex ensemble) in Section 4 to study whether it is beneficial to use more sophisticated greedy algorithms. We also compare the greedy algorithms with XGBoost (XGB) and Random Forest (RF) to study how the convex combination constructed, which can be seen as a weighted ensemble, fare. Both XGB and RF provide strong baselines and are state-of-the-art ensemble methods that won many Kaggle competitions for non-CV non-NLP tasks.

In addition, we also compare the greedy algorithms with the non-greedy method in Section 4.2 (NGCE, and a standard regularized neural network (NN). Both NGCE and NN need to assume a given number of basis functions. Comparison with NN sheds light on how the regularization effect of learning from a convex hull compare with standard \( \ell_2 \) regularization.

We used 12 datasets of various sizes and tasks (#instances and dimensions in brackets): diabetes \((442/10)\), boston \((506/13)\), ca_housing \((20,640/8)\), msd \((515,345/90)\) for regression; iris \((150/4)\), wine \((178/13)\), breast_cancer \((569/30)\), digits \((1.797/64)\), cifar10_f \((60,000/342)\), mnist \((70,000/784)\), coverytype \((581,012/54)\), kddcup99 \((4,898,431/41)\) for classification. Most of the datasets are from UCI ML Repository [Dheeru and Karra Taniskidou, 2017]. All data attributes are normalized to have zero mean and unit variance. Further details on the datasets, their training/validation/test splits, hyper-parameter tuning and experimental setup are available in the appendix.

### 5.1 Comparison of GCE Algorithms

Fig. 1 shows the training and test error (MSE) on the msd dataset for the four greedy algorithms discussed in Section 4. For each variant, we train 100 modules, with each being a single neuron of the form \( B \tanh(u^T x) \). Interestingly, the non-linear greedy variant, which is most commonly studied, is significantly slower than other variants. The PFW variant has the best performance. We observed similar behavior of the algorithms on other datasets and settings, thus we only report the results for the PFW variant in subsequent experiments.

### 5.2 Comparison of GCE with Other Algorithms

We compare GCE (using PFW to greedily choose new basis functions), NGCE, XGB, RF, and NN below.

**Experimental setup.** To ensure a fair comparison between algorithms, we spent a significant effort to tune hyper-parameters of competing algorithms. Particularly, XGB and RF are tuned over 2000 hyper-parameters combinations for small datasets (has less than 10000 training samples), and over 200 combinations for large datasets.

The basis module for GCE is a two-layer network with 1 or 10 hidden neurons for small datasets and 100 hidden neurons for other datasets. GCE grows the network by adding one basis module at a time until no improvement on validation set is detected, or until reaching the maximum limit of 100 modules. NGCE and NN are given the maximum capacity achievable by GCE. For NGCE, it is 100 modules, each of 10/100 hidden neurons for small/large datasets. NN is a two layers neural net of 10/100 hidden neurons for small/large datasets, respectively. NGCE and NN are tuned using grid search over \( \text{learning_rate} \in \{0.01, 0.001\} \), \( \text{regularization} \in \{0, 10^{-6}, \ldots, 10^{-4}\} \), totaling in 14 combinations.
14 combinations. GCE grows the model by one basis module at a time. While we focused on the case with neural networks as the basis models, the greedy algorithms in Section 4 can be applied with trees as basis models. In the case of the nonlinear greedy algorithm and a quadratic loss, the greedy step involves a greedy algorithm and a quadratic loss, the greedy step involves a step no longer corresponds to a standard loss function though.

### Results and Discussion

For each algorithm, the model using the hyper-parameters with the best validation performance is selected. Its test set performance is reported in Table 1. We ran the experiments on train/test splits fixed across all methods in comparison, instead of using multiple train/test splits, which can be useful for further comparison, but is beyond our computing resources.

From Table 1, GCE is competitive with the baselines on datasets with different numbers of features and examples. For regression problems, GCE uses up the maximum number of basis functions, while for classification problems, GCE often terminates way earlier than that, suggesting that it is capable to adapt to the complexity of the task.

GCE uses a fixed set of hyper-parameters without tuning: \( \text{learning\_rate} = 0.001, \text{regularization} = 0 \). All these three algorithms use ReLU activation, MSE criterion for training regression problem, cross entropy loss for classification. The training uses Adam SGD with \( \text{learning\_rate} \) reduced by 10 on plateau (training performance did not improve for 10 consecutive epochs) until reaching the minimum \( \text{learning\_rate} \) of \( 10^{-5} \), at which point the optimizer is ran for another 10 epochs and then returns the solution with the best performance across all training epochs.

### Results and Discussion

For each algorithm, the model using the hyper-parameters with the best validation performance is selected. Its test set performance is reported in Table 1. We ran the experiments on train/test splits fixed across all methods in comparison, instead of using multiple train/test splits, which can be useful for further comparison, but is beyond our computing resources.

From Table 1, GCE is competitive with the baselines on datasets with different numbers of features and examples. For regression problems, GCE uses up the maximum number of basis functions, while for classification problems, GCE often terminates way earlier than that, suggesting that it is capable to adapt to the complexity of the task.

We have chosen datasets from small to large in our experiments. The small datasets can be easily overfitted, and are useful for comparing how well the algorithms avoid overfitting. Empirical results show that GCE builds up the convex ensemble to just a right capacity, but not more. Thus, it has good generalization performance despite having almost little parameter tuning, even for the smaller datasets, where overfitting could easily occur. On the other extreme, overfitting a very large datasets, like kddcup99, is hard. So, empirical results for this dataset show that all models, including ours, have adequate capacity, as they have similar generalization performance. All algorithms were able to scale to the largest dataset (kddcup99), but GCE usually requires little hyper-parameter tuning, while XGB and RF involve significant tuning.

### Table 1: Summary of the empirical results

| Datasets     | GCE   | XGB   | RF  | NN   | NGCE |
|--------------|-------|-------|-----|------|------|
| diabetes     | 42.706| 46.569| 49.519| 43.283| 44.703|
| boston       | 2.165 | 2.271 | 2.705| 2.217 | 2.232 |
| ca_housing   | 0.435 | 0.393 | 0.416| 0.440 | 0.437 |
| msd          | 6.084 | 6.291 | 6.462| 6.186 | 7.610 |
| iris         | 0.00  | 6.67  | 6.67 | 3.33  | 10.00 |
| wine         | 0.00  | 2.78  | 2.78 | 0.0   | 0.0   |
| breast_cancer| 3.51  | 4.39  | 8.77 | 3.51  | 4.39  |
| digits       | 2.78  | 3.06  | 2.50 | 3.33  | 3.06  |
| cifar10_f    | 4.86  | 5.40  | 5.16 | 5.00  | 4.92  |
| mnist        | 1.22  | 1.66  | 2.32 | 1.24  | 1.11  |
| covertype    | 26.70 | 26.39 | 27.73| 26.89 | 26.56 |
| kddcup99     | 0.01  | 0.01  | 0.01 | 0.01  | 0.01  |

(b) Comparison of GCE against XGBoost and RandomForest. GCE shows clear performance advantage over XGBoost and RandomForest on most domains, except that XGBoost has a clear advantage on ca_housing, and RandomForest performing slightly better on digits.

While RandomForest and XGBoost have quite a few parameters to tune, and a proper tuning often requires searching a large number of hyper-parameters, GCE works well across datasets with a default setting for basis module optimization (no tuning) and two options for module size. Overall, GCE is often more efficient than RandomForest and XGBoost as there is little tuning needed. In addition, for large datasets, RandomForest and XGBoost are slow due to the lack of mechanism for mini-batch training and no GPU speed-up.

### 6 Conclusion

This paper presents some new results and empirical insights for learning an optimal convex combination of basis models. While we focused on the case with neural networks as the basis models, the greedy algorithms in Section 4 can be applied with trees as basis models. In the case of the nonlinear greedy algorithm and a quadratic loss, the greedy step involves training a regression tree given a fixed step size. For FW variables, the objective function at each greedy step no longer corresponds to a standard loss function though. Regarding generalization, since trees are nonparametric, our generalization results do not hold, and we may face similar challenges as analyzing random forests. There are interesting questions for further work.
References

[Anthony and Bartlett, 2009] Martin Anthony and Peter L. Bartlett. Neural network learning: Theoretical foundations. Cambridge University Press, 2009.

[Bach, 2017] Francis Bach. Breaking the curse of dimensionality with convex neural networks. The Journal of Machine Learning Research, 18(1):629–681, 2017.

[Barron, 1993] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. Information Theory, IEEE Transactions on, 39(3):930–945, 1993.

[Bartlett and Traskin, 2007] Peter L Bartlett and Mikhail Traskin. Adaboost is consistent. Journal of Machine Learning Research, 8(Oct):2347–2368, 2007.

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

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In KDD, pages 785–794. ACM, 2016.

[Dheeru and Karra Taniskidou, 2017] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[Donahue et al., 1992] Lee K. Jones. A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. The annals of Statistics, pages 608–613, 1992.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[Koltchinskii and Panchenko, 2005] Vladimir Koltchinskii and Dmitry Panchenko. Complexities of convex combinations and bounding the generalization error in classification. The Annals of Statistics, 33(4):1455–1496, 2005.

[Koltchinskii, 2001] Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. IEEE Transactions on Information Theory, 47(5):1902–1914, 2001.

[Lacoste-Julien and Jaggi, 2015] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In NIPS, 2015.

[Lee et al., 1996] Wee Sun Lee, Peter L Bartlett, and Robert C Williamson. Efficient agnostic learning of neural networks with bounded fan-in. Information Theory, IEEE Transactions on, 42(6):2118–2132, 1996.

[Locatello et al., 2018] Francesco Locatello, Gideon Dresdner, Rajiv Khanna, Isabel Valera, and Gunnar Rätsch. Boosting black box variational inference. In Advances in Neural Information Processing Systems, 2018.

[Makovoz, 1996] Yuly Makovoz. Random approximants and neural networks. Journal of Approximation Theory, 85(1):98–109, 1996.

[Mannor et al., 2003] Shie Mannor, Ron Meir, and Tong Zhang. Greedy algorithms for classification–consistency, convergence rates, and adaptivity. Journal of Machine Learning Research, 4(Oct):713–742, 2003.

[Mason et al., 2000a] Llew Mason, Jonathan Baxter, Peter L Bartlett, Marcus Frean, et al. Functional gradient techniques for combining hypotheses. In Advances in Large Margin Classifiers, pages 221–246. MIT, 2000.

[Mason et al., 2000b] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean. Boosting algorithms as gradient descent. In NIPS, pages 512–518, 2000.

[Mendelson, 2003] Shahar Mendelson. A few notes on statistical learning theory. In Advanced lectures on machine learning, pages 1–40. Springer, 2003.

[Oglic and Gärtner, 2016] Dino Oglic and Thomas Gärtner. Greedy feature construction. In NeurIPS, 2016.

[Paszke et al., 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In NIPS-W, 2017.

[Pollard, 1984] David Pollard. Convergence of stochastic processes. Springer, 1984.

[Schapire, 2013] Robert E Schapire. Explaining adaboost. In Empirical inference, pages 37–52. Springer, 2013.

[Tolstikhin et al., 2017] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. Adagan: Boosting generative models. In NeurIPS, 2017.

[Viola and Jones, 2004] Paul Viola and Michael J Jones. Robust real-time face detection. IJCV, 57(2):137–154, 2004.

[Wager and Walther, 2015] Stefan Wager and Guenther Walther. Adaptive concentration of regression trees, with application to random forests. arXiv preprint arXiv:1503.06388, 2015.

[Wellner and Song, 2002] Jon A. Wellner and Shuguang Song. An upper bound for uniform entropy numbers, 2002.
[Wyner et al., 2017] Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1):1558–1590, 2017.

[Zhang, 2003] Tong Zhang. Sequential greedy approximation for certain convex optimization problems. *IEEE Transactions on Information Theory*, 49(3):682–691, 2003.
A Proofs

For a real-valued function $r$ defined on a set $Z$, and a random variable $Z$ taking values from $Z$, we use $\mathbb{P}r$ to denote $\mathbb{E}r(Z)$, and $P_n r$ to denote $\mathbb{E}n r(Z)$.

**Lemma 1.** (Koltchinskii [2001], Lemma 2.5) Let $R$ be a class of real-valued functions defined on $Z$, then

$$\mathbb{E} sup_{r \in R} |Pr - P_n r| \leq 2 \mathbb{E} sup_{r \in R} n r.$$

**Lemma 2.** (Mendelson [2003], Theorem 2.25)

(a) If $\phi : R \to R$ is $c_\phi$-Lipschitz and $\phi(0) = 0$, then

$$\mathbb{E} sup_{r \in R} n \phi \circ r \leq 2c_\phi \mathbb{E} sup_{r \in R} n r.$$

(b) If $g : Z \to [-M, M]$, then

$$\mathbb{E} sup_{r \in R} n (r + g) \leq \mathbb{E} sup_{r \in R} n r + \frac{M}{\sqrt{n}}.$$

**Theorem 2.** Assume that $d_p(G) = p < \infty$, and $L(y, f(x)) = \phi(f(x) - y)$ for a $c_\phi$-Lipschitz nonnegative function $\phi$ satisfying $\phi(0) = 0$. With probability at least $1 - \delta$, for all $f \in co(G)$, $R(f) - R(f^*) \leq R_n(f) - R_n(f^*) + \frac{c_\phi}{\sqrt{n}}$, where $c = 2c_\phi B \left( \sqrt{2 \ln(1/\delta)} + D \sqrt{\ln 2} \right)$, and $D$ is an absolute constant.

**Proof.** For any function $f$, define its regret version $r_f$ by

$$r_f(x, y) = L(y, f(x)) - L(y, f^*(x)).$$

We call $R = \{r_f : f \in H\}$ the regret class of $H$.

For any i.i.d. sample $z = ((x_1, y_1), \ldots, (x_n, y_n))$, we first show that $h(z) = sup_{r \in R} |Pr - P_n r|$ is concentrated around its expectation. As a first step, we verify that $\mathbb{E} sup_{r \in R} n r$ satisfies the bounded difference property with a bound $4c_\phi B/n$. By McDiarmid’s inequality, with probability at least $1 - \delta$,

$$\mathbb{E} sup_{r \in R} |Pr - P_n r| \leq \mathbb{E} sup_{r \in R} |Pr - P_n r| + \frac{\sqrt{8c_\phi^2 B^2 \ln(1/\delta)}}{\sqrt{n}}. \tag{I}$$

By Lemma 1, we have

$$\mathbb{E} sup_{r \in R} |Pr - P_n r| \leq 2\mathbb{E} sup_{r \in R} n r. \tag{II}$$

The Rademacher complexity can be bounded as follows

$$\mathbb{E} sup_{r \in R} n r = \mathbb{E} sup_{f \in co(G)} n f = \mathbb{E} sup_{f \in co(G)} n (L(\cdot, f(\cdot)) - L(\cdot, f^*(\cdot))) \leq \mathbb{E} sup_{f \in co(G)} n (f(\cdot) - f^*(\cdot)) + \frac{2c_\phi B}{\sqrt{n}} \leq 2c_\phi \mathbb{E} sup_{f \in co(G)} n f + \frac{2Bc_\phi}{\sqrt{n}} + \frac{2c_\phi B}{\sqrt{n}} = 2c_\phi \mathbb{E} sup_{f \in co(G)} n f + \frac{4c_\phi B}{\sqrt{n}},$$

where 1st and 3rd inequalities follow from Lemma 2 (b), and 2nd inequality from Lemma 2 (a).

We bound the Rademacher complexity $\mathbb{E} sup_{f \in G} n f$ using a covering number argument, as follows. For any sequence $x = (x_1, \ldots, x_n) \in X^n$, any $f, g \in H$, let $d(x_1, y) = (\frac{1}{n} \sum (f(x_i) - g(x_i))^2)^1/2$. Since $d_p(G) = p$, for any $x$,

$$N(\epsilon, G, d_\phi) \leq \left( \frac{C}{\epsilon} \right)^{2p}, \tag{5}$$

where $C > 0$ is an absolute constant. Using Dudley’s entropy integral bound, we have

$$\mathbb{E} sup_{f \in G} n f \leq C' B\mathbb{E}_\phi \int_0^\infty \sqrt{\frac{\ln N(\epsilon, G, d_\phi)}{n}} d\epsilon = DB \sqrt{\ln n}. \tag{IV}$$

where $C'$ and $D$ are absolute constants.

Combining the inequalities (I)-(IV), we have with probability at least $1 - \delta$, for any $r \in R$, we have

$$\mathbb{P}r \leq P_n r + sup_{r' \in R} |Pr' - P_n r'| \leq P_n r + \frac{2c_\phi B}{n} \left( \sqrt{2 \ln(1/\delta)} + D \sqrt{\ln n} + 2 \right).$$

□
Theorem 3. Assume that \( d_p(G) = p < \infty \), \( L(y, f(x)) = \phi(f(x) - y) \) for a \( c_\phi \)-Lipschitz nonnegative function \( \phi \) satisfying \( \phi(0) = 0 \). Let \( \hat{f} = \arg \min_{f \in \Omega(G)} R_n(f) \), and \( h^* = \arg \min_{f \in \Omega(G)} R(f) \), then with probability at least \( 1 - \delta \), \( \hat{R}(f) \leq R(h^*) + \frac{c}{\sqrt{n}} \), where \( c = 2c_\phi B \left( \sqrt{2 \ln(1/\delta)} + D\sqrt{\beta} + 2 \right) \).

Proof sketch. By inspecting the proof of Theorem 3, we can see that the proof still works if we replace \( f^* \) by an arbitrary function, including \( h^* \). Thus with probability at least \( 1 - \delta \), we have
\[
R(\hat{f}) - R(h^*) \leq R_n(\hat{f}) - R_n(h^*) + \frac{c}{\sqrt{n}} \leq \frac{c}{\sqrt{n}}.
\]

Corollary 1. For \( L(y, f(x)) = |f(x) - y|^p, q \geq 1 \), the bounds in Theorem 2 and 3 hold with \( c_\phi = q(2B)^{q-1} \).

Proof. We have \( L(y, f(x)) = |f(x) - y| \) where \( \phi(u) = |u|^p \). \( \phi \) has a Lipschitz constant of \( q(2B)^{q-1} \) in our case, because we only consider \( u \) of the form \( f(x) - y \), and thus \( u \in [-2B, 2B] \), and for any \( u, v \in [-2B, 2B] \), we have \( |\phi(u) - \phi(v)| = |q^{q-1}(|u| - |v|)| \leq q(2B)^{q-1} |u - v| \), where \( e \) is between \( |u| \) and \( |v| \). The first equation is obtained by applying the mean value theorem to the function \( u^{q-1} \), and taking absolute values on both sides. The second inequality follows because both \( |u| \) and \( |v| \) are not more than 2B, and \( |u| - |v| \) is between \( |u - v| \).

Corollary 2. Assume that \( d_p(G) = p < \infty \), \( Y = \{-1, 1\} \), \( L(y, f(x)) = \phi(yf(x)) \) for a \( c_\phi \)-Lipschitz nonnegative function \( \phi \) satisfying \( \phi(0) = 0 \). Let \( \hat{f} = \arg \min_{f \in \Omega(G)} R_n(f) \), and \( h^* = \arg \min_{f \in \Omega(G)} R(f) \), then with probability at least \( 1 - \delta \), \( \hat{R}(f) \leq R(h^*) + \frac{c}{\sqrt{n}} \), where \( c = 2c_\phi B \left( \sqrt{2 \ln(1/\delta)} + D\sqrt{\beta} + 2 \right) \).

Proof sketch. First note that the loss \( L \) is bounded by \( \sqrt{2B} \) because \( \phi \) is Lipschitz, \( \phi(0) = 0 \), and the margin \( yf(x) \) is in \([-B, B]\). Thus \( L \) is also bounded by \( 2\sqrt{2B} \). This allows the McDiarmid’s inequality in the proof of Theorem 2 to go through. Secondly, Eq. (*) in the proof of Theorem 2 becomes
\[
\mathbb{E} \sup_{f \in \Omega(G)} \mathbb{R}_n(\phi(\cdot f(\cdot))) + \frac{2c_\phi B}{\sqrt{n}},
\]
where the first \( \cdot \) is the output \( y \), and the second \( \cdot \) is the input \( x \). Now we can apply the Lipschitz property of \( \phi \) to bound the first term by \( 2c_\phi \mathbb{E} \sup_{f \in \Omega(G)} \mathbb{R}_n(\cdot f(\cdot)) \). This is equal to \( 2c_\phi \mathbb{E} \sup_{f \in \Omega(G)} \mathbb{R}_n(\cdot f(\cdot)) \), because \( y \in \{-1, 1\} \). Thus the bound (III) still holds. The remaining steps go through without any modification.

Corollary 3. When \( y \in \{-1, 1\} \), \( L(y, f(x)) = -\ln \left( \frac{1}{1 + e^{-yf(x)}} \right) \), the bound in Corollary 2 holds with \( c_\phi = 1 \).

Proof. Consider the loss \( L'(y, f(x)) = -\ln \left( \frac{1}{1 + e^{-yf(x)}} \right) \). Then \( L'(y, f(x)) = L(y, f(x)) - 2 \), that is, \( L \) and \( L' \) differ by only a constant, and thus it is sufficient to bound the loss only for \( L' \). The modified loss \( L'(y, f(x)) \) has the form \( \phi(yf(x)) \) where \( \phi(u) = -\ln \frac{1}{1 + e^{-u}} \). We have \( \phi(0) = 0 \). In addition, \( \phi \) is 1-Lipschitz because the absolute value of its derivative is \( |\phi'(u)| = \frac{e^{-u}}{1 + e^{-u}} \leq 1 \). Hence \( L' \) satisfies the condition in Corollary 2, and the \( O(1/\sqrt{n}) \) bound there holds for \( L' \).

B Datasets and Experimental Setup

We used 12 datasets of various sizes and tasks (both regression and classification) in the experiments. Most of the datasets are from UCI ML Repository [Dheeru and Karra Taniskidou, 2017]. Table 2 provides a summary of the datasets.

Table 2: Summary of datasets used in the experiments. †: number of classes for classification and max [output] for regression. ‡: a predefined set chosen by the author of the respective dataset. If not predefined, training/test set is a random split of 80%/20% of the dataset. Then, the training set is again split into 80% for training and 20% for validation. boston, diabetes, iris, digits, wine, breast_cancer, covertype, kddcup99, ca_housing are loaded using data utilities in scikit-learn package. (Year prediction) mbsd is taken directly from UCI’s website. mnist and cifar10 are well known datasets and are loaded using PyTorch data utilities. cifar10_f is created by transforming cifar10 using a trained DenseNet (95.16% accuracy on test set) and taking the values of 342 features in the last convolutional layer. Training/validation/test set remain identical across all experiments. All data attributes are normalized to have 0-mean 1-standard deviation.

To ensure a fair comparison between algorithms, we spent a significant effort to tune the hyper-parameters of competing algorithms. For each algorithm, the model with the best validation performance is selected and the performance on test set is reported. Small datasets (less than 10000 training samples, i.e. boston, diabetes, iris, digits, wine, breast_cancer), allow for a larger number of hyper-parameter tuning combinations. The details of hyper-parameter tuning for each algorithm is as follows:

- XGBoost: Evaluation metric eval_metric=error for classification and eval_metric=rmse for regression. num_boost_round=2000, early_stopping_rounds=50. For small datasets, we start with eta=0.1 and do 100 random searches over the two most important hyper-parameters in the following ranges: max_depth \( \in \{3, 5, 7, 9, 11, 13\} \), min_child_weight \( \in \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\} \). The random search is followed by a 5 \times 5 fine tuning grid search around the best value for each parameters. Next, we tune gamma \( \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\} \).
grid search for $\text{subsample} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and $\text{colsample_bytree} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, followed by another grid search for $\text{reg\_lambda} \in \{0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$ and $\text{reg\_alpha} \in \{0, 0.1, 1\}$. Next, we tune the learning rate $\text{eta} \in \{0.2, 0.15, 0.05, 0.01, 0.005, 0.001\}$. Then finally we do a 1000 random searches in the neighborhood of the best value for all parameters. This process generates in total about 2298 combinations of hyper-parameters settings. For large datasets, the procedure is similar, with more restrictive range of values for secondary parameters: $\text{min\_samples\_split} \in \{0, 0.05, 0.1, 0.2\}$, $\text{subsample} \in \{0.5, 0.6, 0.7, 0.8\}$, $\text{colsample\_bytree} \in \{0.5, 0.6, 0.7, 0.8\}$, $\text{reg\_lambda} \in \{0, 0.5, 1, 1.5\}$, $\text{eta} \in \{0.15, 0.05, 0.01\}$, which are also optimized separately instead of jointly in pairs as before.

- **RandomForest**: Training criterion is $\text{gini}$ for classification and $\text{mse}$ for regression. The maximum number of trees is 2000, with early stopping if there is no improvement over 50 additional trees. For Random Forest, we found that a large number of random searches is often the most effective strategy. So, we do 4000/200 random searches for small/large datasets respectively. The hyper-parameters and their range are as follows.

  - $\text{max\_features} \in \{\text{auto, sqrt, log2, 1, 3, 5, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7}\}$.
  - $\text{min\_samples\_leaf} \in \{1, 2, 5, 10, 20, 30, 50, 80, 120, 170, 230\}$.
  - $\text{max\_depth} \in \{1, 2, 5, 10, 20, 30, 50, 80, 120\}$.
  - $\text{min\_samples\_split} \in \{2, 4, 6, 8, 10, 12, 14, 16\}$.
  - $\text{no\_bootstrap} \in \{\text{True, False}\}$.

- **GCE, NGCE, NN**: The basis module for GCE is a two layers network with 1 or 10 hidden neurons for small datasets and 100 hidden neurons for other datasets. The output of the basis module is bounded using the $\text{hardtanh}$ function scaled by the bound $B$. For classification we set $B = 10$, for regression, $B = \frac{4}{3} \text{max\_output}$. GCE grows the network by adding one basis module at a time until no improvement on validation set is detected, or until reaching the maximum limit of 100 modules. We use Brent's method as line search for parameter $\alpha_t$ of GCE. NGCE and NN are given the maximum capacity achievable by GCE. For NGCE, it is 100 modules, each of 10/100 hidden neurons for small/large datasets. NN is a two layers neural net of 1000/10000 hidden neurons for small/large datasets, respectively. For both small and large datasets, NGCE and NN are tuned using grid search over $\text{learning\_rate} \in \{0.01, 0.001\}$, $\text{regularization} \in \{0, 10^{-6}, \ldots, 10^{-1}\}$, totaling in 14 combinations. GCE uses a fixed set of hyper-parameters without tuning: $\text{learning\_rate} = 0.001$, $\text{regularization} = 0$. All these three algorithms use ReLU activation, MSE criterion for training regression problem, cross entropy loss for classification. The training uses Adam SGD with $\text{learning\_rate}$ reduced by 10 on plateau (training performance did not improve for 10 consecutive epochs) until reaching the minimum learning rate of $10^{-6}$, at which point the optimizer is ran

for another 10 epochs and then returns the solution with the best performance across all training epochs.

All experiments are implemented using Python and its interface for XGBoost. Random Forest is from scikit-learn package. Greedy variants and NN are implemented using PyTorch [Paszke et al., 2017] and are ran on a machine with Intel i5-7600K CPU @ 3.80GHz (4 cores) and 1x NVIDIA GEFORCE GTX 1080 Ti GPU card. XGBoost and Random Forest are run on cloud machines with Intel CPU E5-2650 v3 @ 2.30GHz (8 cores) and one NVIDIA GEFORCE RTX 2080 Ti GPU.