SpeedBoost: Anytime Prediction with Uniform Near-Optimality

Alexander Grubb
Carnegie Mellon University

J. Andrew Bagnell
Carnegie Mellon University

Abstract

We present SpeedBoost, a natural extension of functional gradient descent, for learning *anytime predictors*, which automatically trade computation time for predictive accuracy by selecting from a set of simpler candidate predictors. These anytime predictors not only generate approximate predictions rapidly, but are capable of using extra resources at prediction time, when available, to improve performance. We also demonstrate how our framework can be used to select weak predictors which target certain subsets of the data, allowing for efficient use of computational resources on difficult examples. We also show that variants of the SpeedBoost algorithm produce predictors which are provably competitive with any possible sequence of weak predictors with the same total complexity.

1 Introduction

The number of machine learning applications which involve real time and latency sensitive predictions is growing rapidly. In areas such as robotics, decisions must be made on the fly and in time to allow for adaptive behaviors which respond to real-time events. In computer vision, prediction algorithms must often keep up with high resolution streams of live video from multiple sources without sacrificing accuracy. Finally, prediction tasks in web applications must be carried out with response to incoming data or user input without significantly increasing latency. For such applications, the decision to use a larger, more complex predictor with higher accuracy or a less accurate, but significantly faster predictor can be difficult.

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

In many of these situations however, it may often be useful to work with prediction algorithms capable of initially giving crude but rapid estimates and then refining the results as time allows. For example, in a robotics application such as autonomous navigation, it may sometimes be the case that the robot must rapidly respond to nearby obstacles, but can spend more time reasoning about distant ones. *Anytime algorithms* [18] exhibit this desirable trait of providing increasingly better results given more computation time.

We present here an approach to learning *anytime predictors*. These anytime predictors are hypotheses which can be evaluated for varying amounts of time, giving better prediction results with increased prediction time. Our approach naturally extends previous work on boosted ensemble learning [11] and functional gradient descent [6, 4] by building predictors which sequentially compute features or weak learners on the input examples and uses these weaker signals to make increasingly accurate predictions as time allows.

One common way of improving the prediction time performance of these additive functional models is to use a *cascade* [17]. A cascade uses a sequence of increasingly complex classifiers to sequentially select and eliminate examples for which the predictor has high confidence in the current prediction, and then continues improving predictions on the low confidence examples. The original formulation focuses on eliminating negative examples, for settings where positive examples are very rare such as face detection, but extensions that eliminate both classes [12] exist.

Many variations on building and optimizing cascades exist [10, 1], but all these methods typically target final performance of the learned predictor. Furthermore, due to the decision making structure of these cascades and the permanent nature of prediction decisions, these models must be very conservative in making early decisions and are unable to recover from early errors. All of these factors combine to make cascades poor anytime predictors.

Previous approaches to the anytime prediction problem have focused on instance-based learning algo-

rithms, such as nearest neighbor classification [16] and novelty detection [13]. These approaches use intelligent instance selection and ordering to acheive rapid performance improvements on common cases, and then typically use the extra time for searching through the 'long tail' of the data distribution and improving result for rare examples. In the case of the latter, the training instances are even dynamically reordered based on the distribution of the inputs to the prediction algorithm, further improving performance.

Also related is the large body of work on variable selection and sparse approximation. These approaches attempt to select a set of variables that will maximize predictive performance given a budget on total computation time. Most relevant to this work are greedy selection algorithms [15, 14, 2] and Rezyin's stochastic approach for budgeted prediction in boosted ensembles [9]. While these feature selection and budgeted approaches target fixed computation budgets, they could potentially be adapted into anytime algorithms by generating selection results for a number of budgets in sequence. Greedy algorithms are particularly suited to this approach as the sequence of selected features is independent of the final budget, so results can be re-used from previous elements in the sequence.

We will first discuss the target anytime predictors we seek to learn, and then present a simple complexity dependent method for learning these predictors. Then we will demonstrate two applications of our anytime prediction algorithms, including one example illustrating how this framework can be used in a cascade-like manner to selectively target computational resources at certain difficult examples. Finally, we will discuss a variant of our algorithm and show that it is provably a *uniformly anytime approximation*, that is, that it produces a sequence of predictions which are near optimal when compared with any other way to utilize the same amount of computation.

2 Anytime Prediction Framework

We consider predictors $f : \mathcal{X} \to \mathcal{V}$ which compute some prediction $f(x) \in \mathcal{V}$ for inputs $x \in \mathcal{X}$, and some associated objective which measures the penalty for a given predictor, usually evaluated elementwise over a set of training examples:

$$\mathcal{R}[f] = \sum_{n=1}^{N} l_n(f(x_n)).$$

For binary classification problems for example, $\mathcal{V} = \mathbb{R}$ and l is a margin-based loss function such as the exponential loss $l_n(f(x_n)) = \exp(-y_n f(x_n))$, where y_n is the class label.

We want to learn a predictor f which is a weighted combination of weaker predictors $h \in \mathcal{H}$

$$f(x) = \sum_i \alpha_i h_i(x), \tag{1}$$

where $\alpha_i \in \mathbb{R}$ and $h_i : \mathcal{X} \to \mathcal{V}$.

In the anytime setting we assume that each weak predictor h has an associated measure of complexity $\tau(h)$ where $\tau : \mathcal{H} \to \mathbb{R}$. This measure of complexity allows for weak predictors which trade accuracy for computational efficiency and vice versa.

For the case where each predictor h can have variable computational cost per example, such as a decision tree, we use the expected computation time. Let $\tau_x(h)$ be the cost of evaluating h on example x. Then:

$$\tau(h) = \mathbb{E}_{\mathcal{X}}[\tau_x(h)].$$

We further assume that calculating the weighted combination over values of α_t takes negligible computation. Using this complexity measure we can describe the predictions generated at a given time $\mathcal{T}$ as

$$f_{\langle \mathcal{T} \rangle} = \sum_{i=1}^{i^*} \alpha_i h_i(x), \quad i^* = \max \left\{ i' \;\middle|\; \sum_{i=1}^{i'} \tau(h_i) < \mathcal{T} \right\}$$

and the associated performance at time $\mathcal{T}$, $\mathcal{R}[f_{\langle \mathcal{T} \rangle}]$.

3 SpeedBoost

We now consider learning algorithms for generating anytime predictors. Formally, given a set of weak predictors $\mathcal{H}$ we want to find a sequence of weights and predictors $\{\alpha_i, h_i\}_{i=1}^{\infty}$ such that the predictor f constructed in (1) achieves good performance $\mathcal{R}[f_{\langle \mathcal{T} \rangle}]$ at all possible stopping times $\mathcal{T}$.

In his work on anytime algorithms, Zilbertstein [18] has identified a number of desirable properties for these algorithms to possess. These include: **interruptability**: a prediction can be generated at any time; **monotonicity**: the quality of a prediction is non-decreasing over time; and **diminishing returns**: prediction quality improves fastest at early stages.

An ensemble predictor as formulated in Section 2 naturally satisfies the interruptability property, by evaluating the weak predictors in sequence and stopping when necessary to output the final prediction.

To learn predictors which satisfy the last two properties, we present SpeedBoost (Algorithm 1), a natural greedy selection approach for selecting weak predictors. Much like AdaBoost greedily selects weak learners using improvement in loss function, this algorithm greedily selects weak predictors based on their

Algorithm 1 SpeedBoost

Given: starting point f_0, objective $\mathcal{R}$, number of stages I
for $i = 1, \ldots, I$ **do**
  Let $h_i, \alpha_i = \arg\max_{h \in \mathcal{H}, \alpha \in \mathbb{R}} \frac{[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]]}{\tau(h)}$
  Let $f_i = f_{i-1} + \alpha_i h_i$.
end for
return Predictor $\left(\{(h_i, \alpha_i)\}_{i=1}^{I}\right)$

improvement per unit-time. This type of greedy selection approach is common in feature selection [15, 2] and submodular optimization [14, 2].

SpeedBoost will select a sequence of feature functions h that greedily maximize the improvement in the algorithm's prediction per unit time. By using a large set $\mathcal{H}$ of different types of weak predictors with varying time complexity, this algorithm provides a simple way to trade computation time with improvement in prediction accuracy. Unfortunately, for many classes of functions where $\mathcal{H}$ is very large, Algorithm 1 can be impractical.

To address this issue, we use the weak learner selection methods of functional gradient descent and other boosting methods. As shown by Mason et al. [6] and Friedman [4], one can view boosting algorithms as a modified version of gradient descent in the space of functions. At each step the gradient is projected onto a set of allowable functions and the resulting direction is used for descent. For boosting, the set of allowable functions contains all the weak predictors being boosted. In this approach, the functional gradient is calculated as

$$\nabla \mathcal{R}[f](x_n) = \nabla l_n(f(x_n)),$$

where $\nabla l_n(f(x_n))$ is the gradient of the loss l_n with respect to the current prediction $f(x_n)$.

Given a function ∇ representing the functional gradient, the projection of ∇ on to a set of weak predictors $\mathcal{H}$ is defined using the functional inner product:

$$\text{Proj}\,(\nabla, \mathcal{H}) = \arg\max_{h \in \mathcal{H}} \frac{\sum_{n=1}^{N} h(x_n) \nabla(x_n)}{\sum_{n=1}^{N} h(x_n)^2}. \quad (2)$$

For classifiers with outputs in $h(x) \in \{-1, +1\}$ (2) is simply a weighted classification problem. Equivalently, when $\mathcal{H}$ is closed under scalar multiplication, the projection rule can minimize the norm in function space:

$$\text{Proj}\,(\nabla, \mathcal{H}) = \arg\max_{h \in \mathcal{H}} \sum_{n=1}^{N} (h(x_n) - \nabla(x_n))^2, \quad (3)$$

which corresponds directly to solving the least squares regression problem.

Algorithm 2 SpeedBoost.MP

Given: starting point f_0, objective $\mathcal{R}$, number of stages I
for $i = 1, \ldots, I$ **do**
  Compute gradient $\nabla_i = \nabla \mathcal{R}[f]$.
  Let $\mathcal{H}^* = \left\{ h_j^* \;\middle|\; h_j^* = \text{Proj}\,(\nabla_i, \mathcal{H}_j) \right\}$.
  Let $h_i, \alpha_i = \arg\max_{h \in \mathcal{H}^*, \alpha \in \mathbb{R}} \frac{[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]]}{\tau(h)}$
  Let $f_i = f_{i-1} + \alpha_i h_i$.
end for
return Predictor $\left(\{(h_i, \alpha_i)\}_{i=1}^{I}\right)$

Algorithm 2 gives a more tractable version of SpeedBoost for learning anytime predictors based on the projection strategy of functional gradient descent. Here we assume that there exist a relatively small number of weak prediction algorithms, $\{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$ representing classes of functions with similar complexity. For example, the classes may represent decision trees of varying depths or kernel-based learners of varying complexity. The algorithm first projects the functional gradient onto each individual class as in gradient boosting, and then uses the best result from each class to perform the greedy selection process described previously. This modification to the greedy algorithm can be viewed as a complexity-weighted version of matching pursuit [8] adapted to function spaces.

4 Case Studies

4.1 Classification

Our first application is a set of classification problems from the UCI Machine Learning Repository [3]. We use the multiclass extension [7] to the exponential loss

$$l_n(f(x_n)) = \sum_{l \neq y_n} \exp(f(x_n)_l - f(x_n)_{y_n}).$$

For weak predictors we use decision trees of varying depth up to 20 nodes deep. We use Algorithm 2 and the weighted classification form of gradient projection to select the sequence of trees for our anytime prediction algorithm.

As a point of comparison we use the AdaBoost.MM [7] implementation of multiclass boosting on the same set of trees. AdaBoost, when used in this manner to generate an anytime predictor, is effectively a variant on the greedy selection algorithm which does not consider the computation time $\tau(h)$ of the individual hypotheses.

Figure 1 shows the performance of our algorithm and AdaBoost as a function of the average number of features accessed per example. On these problems, the

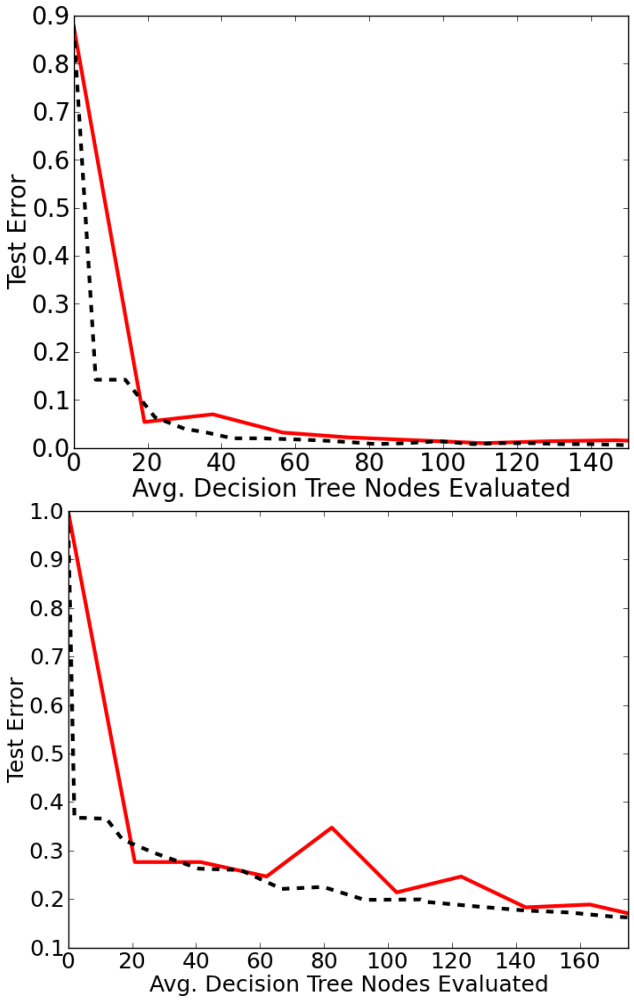

Figure 1: Test set error as a function of prediction time for the UCI 'pendigits' (top) and 'covertype' (bottom) dataset. The algorithms shown are SpeedBoost.MP (black dashed line), and AdaBoost.MM [7] (red solid line).

SpeedBoost generated predictor finds a reasonable prediction using fewer features than the AdaBoost alternative and remains competitive with AdaBoost as time progresses.

4.2 Object Detection

Our second application is a vehicle detection problem using images from onboard cameras on a vehicle on public roads and highways under a variety of weather and time-of-day conditions. The positive class includes all vehicle types, e.g., cars, trucks, and vans. Negative examples are drawn from non-vehicle regions of images taken from the onboard cameras.

4.2.1 Stuctured Prediction on Batch Data

In the previous application we consider weak predictors which will run for roughly the same amount of time on each example x and care about the performance of the learned predictor over time on a single example. In many settings, however, we often care about the computational requirements of a predictor on a batch of examples as a whole. For example, in ranking we care about the computation time required to get an accurate ranking on a set of items, and in computer vision applications many examples from a video or image are often processed simultaneously. Another way to view this problem is as a structured prediction problem where the goal is to make predictions on all pixels in an image simultaneously.

In these settings, it is often beneficial to allocate more computational resources to the difficult examples than the easy examples in a batch, so extra resources are not wasted improving predictions on examples that the algorithm already has high confidence in. In computer vision, in particular, cascades [17] are a popular approach to improving batch prediction performance. These prediction algorithms decrease the overall complexity of a predictor by periodically filtering out and making final predictions on examples, removing them from later prediction stages in the algorithm.

We can use our anytime framework and algorithms to consider running each weak predictor on subsets of the data instead of every example. Given a set of weak predictors $\mathcal{H}$ to optimize over, we can create a new set of predictors $\mathcal{H}'$ by introducing a set of filter functions $\phi \in \Phi$:

$$\phi : \mathcal{X} \to \{0, 1\},$$

and considering the pairing of every filter function and weak predictor

$$\mathcal{H}' = \Phi \times \mathcal{H}$$
$$h'(x) = \phi(x)h(x).$$

These filters ϕ represent the decision to either run the weak predictor h on example x or not. Unlike cascades, these decision are not permanent and apply only to the current stage. This property very nicely allows the anytime predictor to quickly focus on difficult examples and gradually revisit the lower margin examples, whereas the cascade predictor must be highly-confident that an example is correct before halting prediction on that example.

Assuming that the filter function is relatively inexpensive to compute compared to the computation time of the predictor, the new complexity measure for predictors h' is

$$\tau(h') = \mathbb{E}_{\mathcal{X}} \left[\phi(x)\tau_x(h)\right].$$

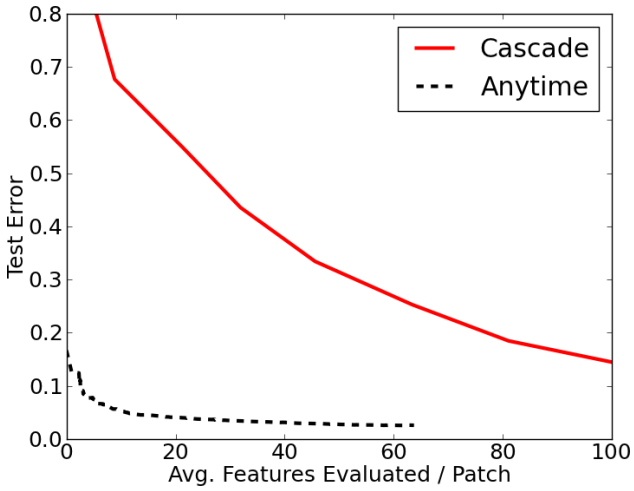

Figure 2: Test set error for the vehicle detection problem as a function of the average number of features evaluated on each image patch.

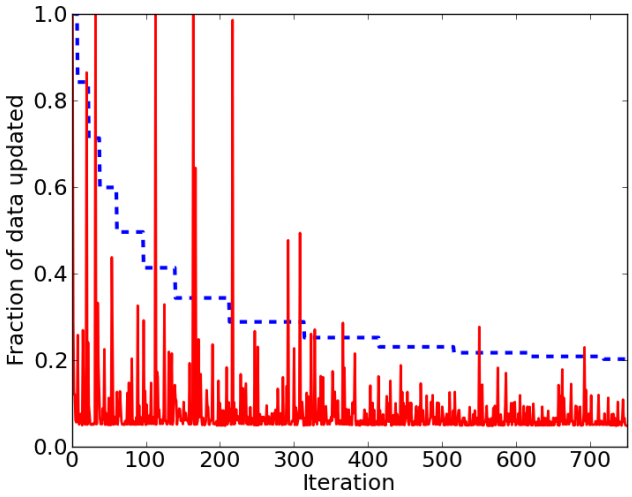

Figure 3: Fraction of data updated by each iteration for the cascade (dashed blue line) and anytime predictor (solid red line).

4.2.2 Implementation

Similar to previous work in object detection, we use Haar-like features computed over image patches for weak predictors. We search over margin-based filter functions ϕ, such that the filters at stage i are

$$\phi_i(x) = \mathbf{1}(|f_{i-1}(x)| < \theta),$$

leveraging the property that examples far away from the margin are (with high probability) already correcly classified.

Computing these filters can be made relatively efficient in two ways. First, by storing examples in a priority queue sorted by current margin the updates and filtering at each stage can be made relatively cheap. Second, after learning the anytime predictor using Algorithm 2, all future filters are known at each stage, and so the predictor can quickly determine the next stage an example will require computation in and handle the example accordingly.

We compare against both a standard AdaBoost implementation and a cascade implementation for this detection dataset. Figure 2 gives the error on a test dataset of 10000 positive and 50000 negative examples as a function of computation time. In this setting the cascade is at a significant disadvantage because it must solidly rule out any negative examples before classifying them as such, while the AdaBoost and anytime predictors can initially declare all examples negative and proceed to adjust prediction on positive examples.

To further illustrate the large benefit to being able to ignore examples early on and revisit them later, Figure 3 gives a per iteration plot of the fraction of test data updated by each corresponding feature. This demonstrates the large culling early on of examples that allows the anytime predictor to improve performance much more rapidly. Finally, Figure 4 displays the ROC curve for the anytime predictor at various complexity thresholds against the ROC curve generated by the final cascade predictions and Figure 5 shows the visual evolution of the cascade and anytime predictions on a single test image.

5 Theoretical Analysis

We will now analyze a variant of the SpeedBoost algorithm and prove that the predictor produced by this algorithm is near optimal with respect to any sequence of weak predictors that could be computed in the same amount of time, for a common set of loss functions and certain classes of weak predictors.

For analysis, one can interpret Algorithm 2 as a time-based version of matching pursuit [5]. Unfortunately, the sequence of weak predictors selected by matching pursuit can perform poorly with respect to the optimal sequence for some fixed time budget $\mathcal{T}$ when faced with highly correlated weak predictors [8]. A modification of the matching pursuit algorithm called orthogonal matching pursuit [8] addresses this flaw, and can be shown to be a competitive approximation to any sequence that could be selected [2].

To that end, Algorithm 3 gives a modification of Algorithm 2 which is analogous to orthogonal matching pursuit, SpeedBoost.OMP. The key difference between these algorithms is the re-fitting of the weights on every weak predictor selected so far at every iter-

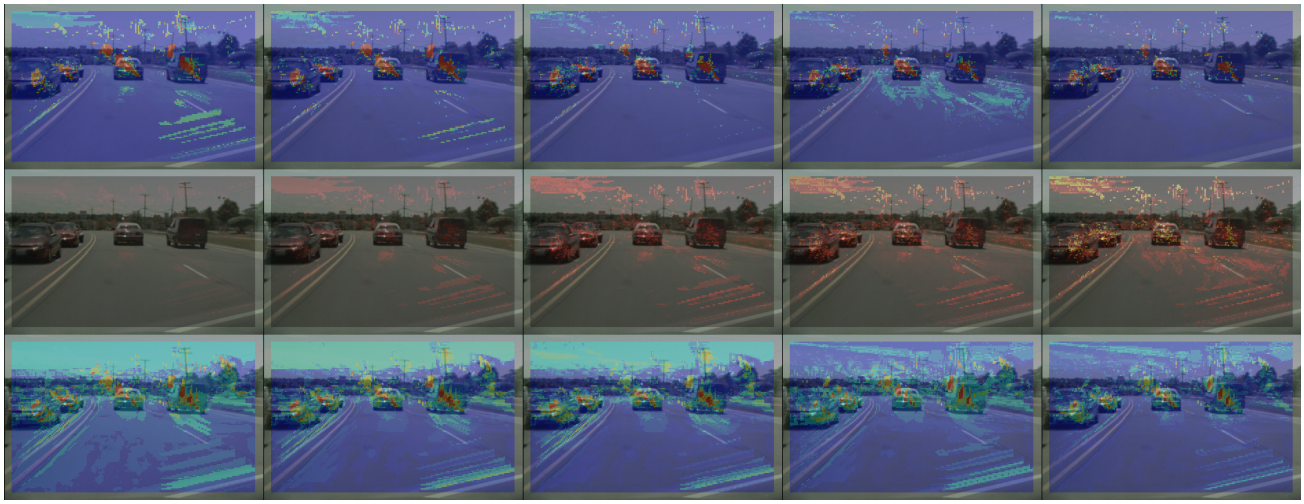

Figure 5: Images displaying the detection activations on a test image for the anytime predictor produced by SpeedBoost (top) and the cascade (bottom). Displayed in the middle is a heat map of the number of features evaluated by the SpeedBoost predictor for each pixel. Images are arranged left to right through time, at intervals of 7 average feature evaluations per pixel.

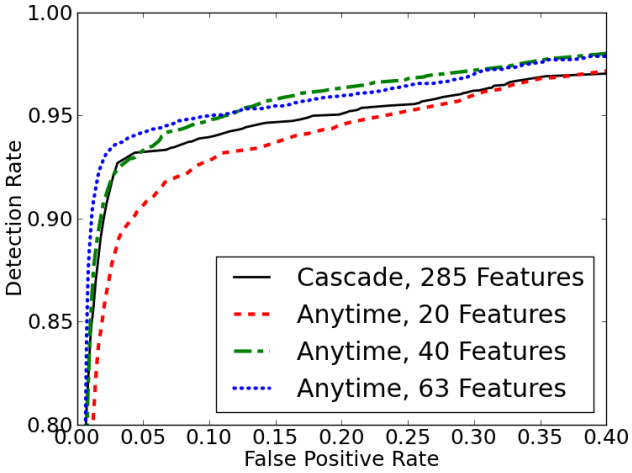

Figure 4: ROC curves for the final cascade predictions and anytime algorithm predictions at various computation thresholds. Computation is measured using the average number of features computed on each patch.

ation of the algorithm. The key disadvantage of using this algorithm in practice is that the output of all previous weak predictors must be maintained and the linear combination re-computed whenever a final prediction is desired. In practice, we found that SpeedBoost and SpeedBoost.MP performed nearly as well as Algorithm 3 in terms of improvement in the objective function, while being significantly cheaper to implement. Figure 6 shows a comparison of the test error on the UCI 'covertype' dataset for SpeedBoost and SpeedBoost.OMP. In this case, while the training objective performances were nearly indistinguishable (not shown), Algorithm 3 overfit to the training data much more rapidly.

Algorithm 3 SpeedBoost.OMP

Given: starting point f_0, objective $\mathcal{R}$, number of stages I

for $i = 1, \ldots, I$ **do**

Compute gradient $\nabla_i = \nabla \mathcal{R}[f]$.

Let $\mathcal{H} = \left\{ h_j^* \;\middle|\; h_j^* = \text{Proj}\left(\nabla_i, \mathcal{H}_j\right) \right\}$.

Let $h_i, \alpha_i = \underset{h' \in \mathcal{H}, \alpha' \in \mathbb{R}^i}{\arg\max} \frac{\left[\mathcal{R}[f_{i-1}] - \mathcal{R}\left[\sum_{j=1}^{i-1} \alpha'_j h_j + \alpha'_i h'\right]\right]}{\tau(h)}$

Let $f_i = \sum_{j=1}^{i} \alpha_{ij} h_j$.

end for

return Predictor $\left(\{(h_i, \alpha_i)\}_{i=1}^{I} \right)$

We will now proceed by first adapting current results in submodular maximization to handle the computation time-dependent case, followed by an analysis showing that loss functions that are strongly-smooth and strongly-convex are submodular. Combining these two results, we will then show that SpeedBoost.OMP, which approximately greedily maximizes the improvement in loss $\mathcal{R}$, is competitive with any other sequence of weak predictors one could select.

5.1 Greedy Submodular Maximization

We can build a set function $z_{\mathcal{R}} : 2^{\mathcal{H}} \to \mathbb{R}$ from our loss $\mathcal{R}$, turning our loss minimization problem into a maximization problem over sets:

$$z_{\mathcal{R}}(S) = \max_f \mathcal{R}[f] - \mathcal{R}[f_S], \tag{4}$$

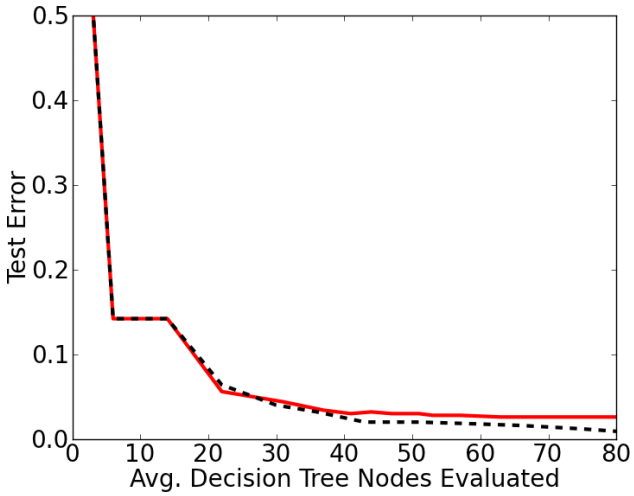

Figure 6: Test set error as a function of complexity for the UCI 'pendigits' dataset, comparing SpeedBoost.MP (Algorithm 2) (black dashed line) to SpeedBoost.OMP (Algorithm 3) (solid red line).

where

$$f_S = \underset{\alpha \in \mathbb{R}^{|S|}, f = \sum_{h \in S} \alpha_h h}{\arg\min} \mathcal{R}[f]. \tag{5}$$

This set function models the best improvement in loss we can obtain from any linear combination of a set S of selected weak predictors.

Definition 5.1 (Submodularity Ratio [2])**.** *A function z has submodularity ratio $\gamma_{U,k}$ if for some set U and some $k \geq 1$:*

$$\sum_{h \in S} \left[z(\{h\} \cup L) - z(L) \right] \geq \gamma_{U,k} \left[z(L \cup S) - z(L) \right],$$

for all $L \subset U$, and S such that $|S| \leq k$ and $S \cap L = \emptyset$.

To simplify the following derivations, we will only refer to γ such that $\gamma \leq \gamma_{U,k}$ for all U and k.

Now we would like to analyze the time based greedy approximation algorithm, extending Streeter and Golovin's analysis of the offline greedy maximization algorithm [14] to include the submodularity ratio of Das and Kempe [2].

Given elements $h \in \mathcal{H}$ and some time $\tau(h)$ associated with each element, the greedy algorithm iteratively selects elements g_j using:

$$g_j = \underset{h \in \mathcal{H}}{\arg\max} \left[\frac{z(G_j \cup h) - z(G_j)}{\tau(h)} \right], \tag{6}$$

where $G_j = (g_1, g_2, \ldots, g_{j-1})$. Following the same structure as the analysis in [14], we can now prove that the set selected by the greedy algorithm approximates the optimal sequence of elements for functions which satisfy Definition 5.1.

Lemma 5.2. *If a function z has submodularity ratio γ then for any $L, S \subset \mathcal{H}$:*

$$\gamma \frac{z(L \cup S) - z(L)}{\tau(S)} \leq \max_{h \in \mathcal{H}} \left[\frac{z(L \cup \{h\}) - z(L)}{\tau(h)} \right].$$

Proof. By submodularity ratio:

$$\gamma \frac{z(L \cup S) - z(L)}{\tau(S)} \leq \frac{\sum_{h \in S} \left[z(L \cup S) - z(L) \right]}{\sum_{h \in S} \tau(h)}.$$

Taking the max of the right hand side proves the Lemma. □

Proposition 5.3. *Let s_j be the value of the maximum in (6) evaluated by the greedy algorithm at timestep j. Then for all ordered sets S:*

$$z(S_{\langle t \rangle}) \leq z(G_j) + \frac{t s_j}{\gamma}.$$

Theorem 5.4. *For any list of elements S, the greedy algorithm selects a list G such that:*

$$z(G_{\langle \mathcal{T} \rangle}) > \left(1 - e^{-\gamma}\right) z(S_{\langle \mathcal{T} \rangle}),$$

where $\mathcal{T} = \sum_{j=1}^{L} \tau(g_j)$ and $S_{\langle \mathcal{T} \rangle} = (s_1, \ldots, s_k)$ such that $\sum_{i=1}^{k} \tau(s_i) \leq \mathcal{T}$.

Proof. Define $\Delta_j = S_{\langle \mathcal{T} \rangle} - z(G_j)$. By the previous proposition $S_{\langle \mathcal{T} \rangle} \leq z(G_j) + \frac{\mathcal{T} s_j}{\gamma}$. Then:

$$\Delta_j \leq \frac{\mathcal{T} s_j}{\gamma} = \frac{\mathcal{T}}{\gamma} \left(\frac{\Delta_j - \Delta_{j+1}}{\tau(g_j)} \right).$$

Rearranging we get $\Delta_{j+1} \leq \Delta_j (1 - \frac{\tau_j \gamma}{\mathcal{T}})$. Unroll to get

$$\Delta_{L+1} \leq \Delta_1 \left(\prod_{j=1}^{L} 1 - \frac{\tau_j \gamma}{\mathcal{T}} \right).$$

As per [14], this is maximized by $\tau_j = \frac{\mathcal{T}}{L}$, giving:

$$\begin{aligned} z(S_{\langle \mathcal{T} \rangle}) - z(G_{L+1}) = \Delta_{L+1} &\leq \Delta_1 \left(1 - \frac{\gamma}{L}\right)^L \\ &< z(S_{\langle \mathcal{T} \rangle}) e^{-\gamma}, \end{aligned}$$

or $z(G_{\langle \mathcal{T} \rangle}) > z(S_{\langle \mathcal{T} \rangle})(1 - e^{-\gamma})$. □

5.2 Submodularity Ratio of Smooth Losses

We can now show that the set function $z_{\mathcal{R}}$ in (4) has a submodularity ratio as in Definition 5.1. The submodularity ratio of $z_{\mathcal{R}}$ will rely on the convexity and smoothness of the loss function $\mathcal{R}$.

A functional $\mathcal{R}$ is λ-*strongly convex* if $\forall f, f'$:

$$\mathcal{R}[f'] \geq \mathcal{R}[f] + \langle \nabla \mathcal{R}[f], f' - f \rangle + \frac{\lambda}{2} \|f' - f\|^2,$$

for some $\lambda > 0$, and Λ-*strongly smooth* if

$$\mathcal{R}[f'] \leq \mathcal{R}[f] + \langle \nabla \mathcal{R}[f], f' - f \rangle + \frac{\Lambda}{2} \|f' - f\|^2,$$

for some $\Lambda > 0$.

Theorem 5.5. *Let C be the covariance matrix for the weak predictors $h \in \mathcal{H}$ transformed to have zero mean and unit variance. Let $\lambda_{min}(C)$ be the minimum eigenvalue of C. Let loss $\mathcal{R}$ be a λ-strongly convex and Λ-strongly smooth functional. Then for all $L, S \subset H$ with $L \cap S = \emptyset$ there exists $\gamma \geq \frac{\lambda}{\Lambda}\lambda_{min}(C)$ such that*

$$\gamma \leq \frac{\sum_{h \in S} [z_{\mathcal{R}}(\{h\} \cup L) - z_{\mathcal{R}}(L)]}{[z_{\mathcal{R}}(L \cup S) - z_{\mathcal{R}}(L)]}.$$

Proof. Starting with the definition of the submodularity ratio γ from Definition 5.1:

$$\begin{aligned}\gamma &= \min_{L,S} \frac{\sum_{h \in S} [z_{\mathcal{R}}(\{h\} \cup L) - z_{\mathcal{R}}(L)]}{[z_{\mathcal{R}}(L \cup S) - z_{\mathcal{R}}(L)]} \\ &= \min_{L,S} \frac{\sum_{h \in S} \left[\mathcal{R}[f_L] - \mathcal{R}[f_{\{h\} \cup L}]\right]}{[\mathcal{R}[f_L] - \mathcal{R}[f_{L \cup S}]]}.\end{aligned}$$

Let $z'(S) = \|f_L + \nabla \mathcal{R}[f]\|^2 - \|f_L + \nabla \mathcal{R}[f] - f_S\|^2$ be another submodular set function representing a quadratic loss centered at $f_L + \nabla R[f]$. Using the strong-convexity and strong-smoothness of $\mathcal{R}$, we can then express the submodularity ratio of $z_{\mathcal{R}}$ in terms of the submodularity ratio of f_L:

$$\gamma \geq \min_{L,S} \frac{\lambda}{\Lambda} \frac{\sum_{h \in S} [z'(\{h\} \cup L) - z'(L)]}{[z'(L \cup S) - z'(L)]}.$$

Using the result from Lemma 2.4 of [2], on the right hand side of this equation (the submodularity ratio of z') gives $\gamma \geq \frac{\lambda}{\Lambda}\lambda_{\min}(C)$. □

5.3 Uniformly Anytime Near-Optimality

We can now combine the results from Sections 5.1 and 5.2 to obtain an approximation guarantee for the performance of Algorithm 3.

Theorem 5.6. *Uniformly Anytime Approximation Let C be the covariance matrix for the weak predictors $h \in \mathcal{H}$ transformed to have zero mean and unit variance. Let $\lambda_{min}(C)$ be the minimum eigenvalue of C. Let loss $\mathcal{R}$ be a λ-strongly convex and Λ-strongly smooth functional. Let S be any sequence of elements in $\mathcal{H}$. Let $\gamma = \left(\frac{\lambda}{\Lambda}\lambda_{min}(C)\right)^2$. Algorithm 3 selects a sequence of weak predictors $G = \{h_i \mid h_i \in \mathcal{H}\}_{i=1}^{I}$ such that for any time $\mathcal{T} = \sum_{i=1}^{i'} \tau(h_j)$,*

$$z_{\mathcal{R}}(G_{\langle \mathcal{T} \rangle}) > \left(1 - e^{-\gamma}\right) z_{\mathcal{R}}(S_{\langle \mathcal{T} \rangle}),$$

where $S_{\langle \mathcal{T} \rangle} = (s_1, \ldots, s_k)$ such that $\sum_{i=1}^{k} \tau(s_i) \leq \mathcal{T}$.

Proof. Starting with the result from Theorem 5.5 we know that the submodularity ratio of $z_{\mathcal{R}}$ is $\gamma \geq \frac{\lambda}{\Lambda}\lambda_{\min}(C)$.

By Theorem 5.4, the strictly greedy maximization algorithm selects a list G' such that:

$$z(G'_{\langle \mathcal{T} \rangle}) > \left(1 - e^{-\gamma}\right) z(S_{\langle \mathcal{T} \rangle}),$$

where $\mathcal{T} = \sum_{j=1}^{L} \tau(g_j)$ and $S_{\langle \mathcal{T} \rangle} = (s_1, \ldots, s_k)$ such that $\sum_{i=1}^{k} \tau(s_i) \leq \mathcal{T}$.

The key difference between the orthogonal matching pursuit algorithm and the greedy algorithm is the selection of elements in the weak predictor subset $\mathcal{H}_j$ is done using gradient projection instead of greedy selection. Let h_j and be the element selected by the OMP algorithm and h'_j be the element selected by the greedy algorithm.

Let $z'(S) = \|f_L + \nabla \mathcal{R}[f]\|^2 - \|f_L + \nabla \mathcal{R}[f] - f_S\|^2$ be another submodular set function representing a quadratic loss centered at $f_L + \nabla R[f]$. Using the result from Theorem 3.7 in [2], we have $z'(L \cup h_j) \geq \lambda_{\min} z'(L \cup h'_j)$. By strong-convexity and strong-smoothness:

$$\begin{aligned}z_{\mathcal{R}}(L \cup h_j) &\geq \frac{1}{\Lambda} z'(L \cup h_j) \\ &\geq \frac{1}{\Lambda} \lambda_{\min} z'(L \cup h'_j) \\ &\geq \frac{\lambda}{\Lambda} \lambda_{\min} z_{\mathcal{R}}(L \cup h'_j).\end{aligned}$$

Because this holds for every element in the sequence selected by Algorithm 3, we have the result of the theorem. □

Theorem 5.6 states that, for all times $\mathcal{T}$ that correspond to the computation times that weak learners selected by Algorithm 3 finish, the resulting improvement in loss $\mathcal{R}$ is approximately as large as any other sequence of weak learners that could have been computed up to that point. This means that the anytime predictor generated by Algorithm 3 is competitive even with sequences specifically targeting fixed time budgets, uniformly across all times at which the anytime predictor computes new predictions.

Acknowledgements

We would like to thank the AISTATS reviewers for their helpful feedback. This work was conducted through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016.

References

[1] S. Brubaker, Jianxin Wu, Jie Sun, Matthew Mullin, and James Rehg. On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, pages 65–86, 2008.

[2] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[3] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[4] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[5] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397 – 3415, dec 1993.

[6] L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

[7] I. Mukherjee and R. E. Schapire. A theory of multiclass boosting. In *Advances in Neural Information Processing Systems 22*, Cambridge, MA, 2010. MIT Press.

[8] Y. C. Pati, R. Rezaiifar, Y. C. Pati R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

[9] Lev Reyzin. Boosting on a budget: Sampling for feature-efficient prediction. In *Proceedings of the 28th Internation Conference on Machine Learning (ICML)*, 2011.

[10] Mohammad J. Saberian and Nuno Vasconcelos. Boosting classifier cascades. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

[11] R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[12] Jan Sochman and Jiri Matas. Waldboost: Learning for time constrained sequential detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 150–156, 2005.

[13] Boris Sofman, J. Andrew Bagnell, and Anthony Stentz. Anytime online novelty detection for vehicle safeguarding. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (ICRA)*, 2010.

[14] Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS)*, 2008.

[15] Joel A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory*, 50:2231–2242, 2004.

[16] Ken Ueno, Xiaopeng Xi, Eamonn Keogh, and Dah-Jye Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. *Data Mining, IEEE International Conference on*, 0:623–632, 2006.

[17] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2001.

[18] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.