Consistency of Online Random Forests

Misha Denil MDENIL@CS.UBC.CA
David Matheson DAVIDM@CS.UBC.CA
Nando de Freitas NANDO@CS.UBC.CA
University of British Columbia

Abstract

As a testament to their success, the theory of random forests has long been outpaced by their application in practice. In this paper, we take a step towards narrowing this gap by providing a consistency result for online random forests.

1. Introduction

Random forests are a class of ensemble method whose base learners are a collection of randomized tree predictors, which are combined through averaging. The original random forests framework described in Breiman (2001) has been extremely influential (Svetnik et al., 2003; Prasad et al., 2006; Cutler et al., 2007; Shotton et al., 2011; Criminisi et al., 2011).

Despite their extensive use in practical settings, very little is known about the mathematical properties of these algorithms. A recent paper by one of the leading theoretical experts states that

> Despite growing interest and practical use, there has been little exploration of the statistical properties of random forests, and little is known about the mathematical forces driving the algorithm (Biau, 2012).

Theoretical work in this area typically focuses on stylized versions of the random forests algorithms used in practice. For example, Biau et al. (2008) prove the consistency of a variety of ensemble methods built by averaging base classifiers. Two of the models they study are direct simplifications of the forest growing algorithms used in practice; the others are stylized neighbourhood averaging rules, which can be viewed as simplifications of random forests through the lens of Lin & Jeon (2002). An even further simplified version of random forests in one dimension is studied in Genuer (2010; 2012).

In this paper we make further steps towards narrowing the gap between theory and practice. In particular, we present what is, to the best of our knowledge, the first consistency result for online random forests.

2. Related Work

Different variants of random forests are distinguished by the methods they use for growing the trees. The model described in Breiman (2001) builds each tree on a bootstrapped sample of the training set using the CART methodology (Breiman et al., 1984). The optimization in each leaf that searches for the optimal split point is restricted to a random selection of features, or linear combinations of features.

The framework of Criminisi et al. (2011) operates slightly differently. Instead of choosing only features at random, this framework chooses entire decisions (i.e. both a feature or combination of features and a threshold together) at random and optimizes only over this set. Unlike the work of Breiman (2001), this framework chooses not to include bagging, preferring instead to train each tree on the entire data set and introduce randomness only in the splitting process. The authors argue that without bagging their model obtains max-margin properties.

In addition to the frameworks mentioned above, many practitioners introduce their own variations on the basic random forests algorithm, tailored to their specific problem domain. A variant from Bosch et al. (2007) is especially similar to the technique we use in this paper: When growing a tree the authors randomly select one third of the training data to determine the structure of the tree and use the remaining two thirds to fit the leaf estimators. However, the authors consider this only as a technique for introducing randomness

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

into the trees, whereas in our model the partitioning of data plays a central role in consistency.

In addition to these offline methods, several researchers have focused on building online versions of random forests. Online models are attractive because they do not require that the entire training set be accessible at once. These models are appropriate for streaming settings where training data is generated over time and should be incorporated into the model as quickly as possible. Several variants of online decision tree models are present in the MOA system of Bifet et al. (2010).

The primary difficulty with building online decision trees is their recursive structure. Data encountered once a split has been made cannot be used to correct earlier decisions. A notable approach to this problem is the Hoeffding tree (Domingos & Hulten, 2000) algorithm, which works by maintaining several candidate splits in each leaf. The quality of each split is estimated online as data arrive in the leaf, but since the entire training set is not available these quality measures are only estimates. The Hoeffding bound is employed in each leaf to control the amount of data which must be collected to ensure that the split chosen on the basis of these estimates is the true best split with high probability. Domingos & Hulten (2000) prove that under reasonable assumptions the online Hoeffding tree converges to the offline tree with high probability. The Hoeffding tree algorithm is implemented in the system of Bifet et al. (2010).

Alternative methods for controlling tree growth in an online setting have also been explored. Saffari et al. (2009) use the online bagging technique of Oza & Russel (2001) and control leaf splitting using two parameters in their online random forest. One parameter specifies the minimum number of data points which must be seen in a leaf before it can be split, and another specifies a minimum quality threshold that the best split in a leaf must reach. This is similar in flavor to the technique used by Hoeffding trees, but trades theoretical guarantees for more interpretable parameters.

One active avenue of research in online random forests involves tracking non-stationary distributions, also known as concept drift. Many of the online techniques incorporate features designed for this problem (Gama et al., 2005; Abdulsalam, 2008; Saffari et al., 2009; Bifet et al., 2009; 2012). However, tracking of non-stationarity is beyond the scope of this paper.

The most well known theoretical result for random forests is that of Breiman (2001), which gives an upper bound on the generalization error of the forest in terms of the correlation and strength of trees. Following Breiman (2001), an interpretation of random forests as an adaptive neighborhood weighting scheme was published by Lin & Jeon (2002). This was followed by the first consistency result in this area from Breiman (2004), who proves consistency of a simplified model of the random forests used in practice. In the context of quantile regression the consistency of a certain model of random forests has been shown by Meinshausen (2006). A model of random forests for survival analysis was shown to be consistent in Ishwaran & Kogalur (2010).

Significant recent work in this direction comes from Biau et al. (2008) who prove the consistency of a variety of ensemble methods built by averaging base classifiers, as is done in random forests. A key feature of the consistency of the tree construction algorithms they present is a proposition that states that if the base classifier is consistent then the forest, which takes a majority vote of these classifiers, is itself consistent.

The most recent theoretical study, and the one which achieves the closest match between theory and practice, is that of Biau (2012). The most significant way in which their model differs from practice is that it requires a second data set which is not used to fit the leaf predictors in order to make decisions about variable importance when growing the trees. One of the innovations of the model we present in this paper is a way to circumvent this limitation in an online setting while maintaining consistency.

3. Online Random Forests with Stream Partitioning

In this section we describe the workings of our online random forest algorithm. A more precise (pseudo-code) description of the training procedure can be found in Appendix A.

3.1. Forest Construction

The random forest classifier is constructed by building a collection of random tree classifiers in parallel. Each tree is built independently and in isolation from the other trees in the forest. Unlike many other random forest algorithms we do not perform bootstrapping or subsampling at this level; however, the individual trees each have their own optional mechanism for subsampling the data they receive.

3.2. Tree Construction

Each node of the tree is associated with a rectangular subset of $\mathbb{R}^D$, and at each step of the construction the collection of cells associated with the leafs of the tree forms a partition of $\mathbb{R}^D$. The root of the tree is $\mathbb{R}^D$ itself. At each step we receive a data point (X_t, Y_t) from the environment. Each point is assigned to one of two possible streams at random with fixed probability. We denote stream membership with the variable $I_t \in \{s, e\}$. How the tree is updated at each time step depends on which stream the corresponding data point is assigned to.

We refer to the two streams as the *structure* stream and the *estimation* stream; points assigned to these streams are structure and estimation points, respectively. These names reflect the different uses of the two streams in the construction of the tree:

Structure points are allowed to influence the structure of the tree partition, i.e. the locations of candidate split points and the statistics used to choose between candidates, but they are not permitted to influence the predictions that are made in each leaf of the tree.

Estimation points are not permitted to influence the shape of the tree partition, but can be used to estimate class membership probabilities in whichever leaf they are assigned to.

Only two streams are needed to build a consistent forest, but there is no reason we cannot have more. For instance, we explored the use of a third stream for points that the tree should ignore completely, which gives a form of online sub-sampling in each tree. We found empirically that including this third stream hurts performance of the algorithm, but its presence or absence does not affect the theoretical properties.

3.3. Leaf Splitting Mechanism

When a leaf is created the number of candidate split dimensions for the new leaf is set to $\min(1 + \text{Poisson}(\lambda), D)$, and this many distinct candidate dimensions are selected uniformly at random. We then collect m candidate splits in each candidate dimension (m is a parameter of the algorithm) by projecting the first m structure points to arrive in the newly created leaf onto the candidate dimensions. We maintain several structural statistics for each candidate split. Specifically, for each candidate split we maintain class histograms for each of the new leafs it would create, using data from the estimation stream. We also maintain structural statistics, computed from data in the structure stream, which can be used to choose between the candidate splits. The specific form of the structural statistics does not affect the consistency of our model, but it is crucial that they depend only on data in the structure stream.

Finally, we require two additional conditions which control when a leaf at depth d is split:

1. Before a candidate split can be chosen, the class histograms in each of the leafs it would create must incorporate information from at least $\alpha(d)$ estimation points.
2. If any leaf receives more than $\beta(d)$ estimation points, and the previous condition is satisfied for *any* candidate split in that leaf, then when the next structure point arrives in this leaf it must be split regardless of the state of the structural statistics.

The first condition ensures that leafs are not split too often, and the second condition ensures that no branch of the tree ever stops growing completely. In order to ensure consistency we require that $\alpha(d) \to \infty$ monotonically in d and that $d/\alpha(d) \to 0$. We also require that $\beta(d) \geq \alpha(d)$ for convenience.

When a structure point arrives in a leaf, if the first condition is satisfied for some candidate split then the leaf may optionally be split at the corresponding point. The decision of whether to split the leaf or wait to collect more data is made on the basis of the structural statistics collected for the candidate splits in that leaf.

3.4. Structural Statistics

In each candidate child we maintain an estimate of the posterior probability of each class, as well as the total number of points we have seen fall in the candidate child, both counted from the structure stream. In order to decide if a leaf should be split, we compute the information gain for each candidate split which satisfies condition 1 from the previous section,

$$I(S) = H(A) - \frac{|A'|}{|A|} H(A') - \frac{|A''|}{|A|} H(A'') \ .$$

Here S is the candidate split, A is the cell belonging to the leaf to be split, and A' and A'' are the two leafs that would be created if A were split at S. The function $H(A)$ is the discrete entropy, computed over the labels of the structure points which fall in the cell A.

We select the candidate split with the largest information gain for splitting, provided this split achieves a minimum threshold in information gain, τ. The value of τ is a parameter of our algorithm.

3.5. Prediction

At any time the online forest can be used to make predictions for unlabelled data points using the model built from the labelled data it has seen so far. To make a prediction for a query point x at time t, each tree computes, for each class k,

$$\eta_t^k(x) = \frac{1}{N^e(A_t(x))} \sum_{\substack{(X_\tau, Y_\tau) \in A_t(x) \\ I_\tau = e}} \mathbb{I}\{Y_\tau = k\} \quad ,$$

where $A_t(x)$ denotes the leaf of the tree containing x at time t, and $N^e(A_t(x))$ is the number of estimation points which have been counted in $A_t(x)$ during its lifetime. Similarly, the sum is over the labels of these points. The tree prediction is then the class which maximizes this value:

$$g_t(x) = \arg\max_k \{\eta_t^k(x)\} \quad .$$

The forest predicts the class which receives the most votes from the individual trees.

Note that this requires that we maintain class histograms from both the structure and estimation streams separately for each candidate child in the fringe of the tree. The counts from the structure stream are used to select between candidate split points, and the counts from the estimation stream are used to initialize the parameters in the newly created leafs after a split is made.

3.6. Memory Management

The typical approach to building trees online, which is employed in Domingos & Hulten (2000) and Saffari et al. (2009), is to maintain a fringe of candidate children in each leaf of the tree. The algorithm collects statistics in each of these candidate children until some (algorithm dependent) criterion is met, at which point a pair of candidate children is selected to replace their parent. The selected children become leafs in the new tree, acquiring their own candidate children, and the process repeats. Our algorithm also uses this approach.

The difficulty here is that the trees must be grown breadth first, and maintaining the fringe of potential children is very memory intensive when the trees are large. Our algorithm also suffers from this deficiency, as maintaining the fringe requires $O(cmd)$ statistics in each leaf, where d is the number of candidate split dimensions, m is the number of candidate split points (i.e. md pairs of candidate children per leaf) and c is the number of classes in the problem. These statistics can be quite large and for deep trees the memory cost becomes prohibitive.

In practice the memory problem is resolved either by growing small trees, as in Saffari et al. (2009), or by bounding the number of nodes in the fringe of the tree, as in Domingos & Hulten (2000). Other models of streaming random forests, such as those discussed in Abdulsalam (2008), build trees in sequence instead of in parallel, which reduces the total memory usage.

Our algorithm makes use of a bounded fringe and adopts the technique of Domingos & Hulten (2000) to control the policy for adding and removing leafs from the fringe.

In each tree we partition the leafs into two sets: we have a set of *active* leafs, for which we collect split statistics as described in earlier sections, and a set of *inactive* leafs for which we store only two numbers. We call the set of active leafs the *fringe* of the tree, and describe a policy for controlling how inactive leafs are added to the fringe.

In each inactive leaf A_t we store the following two quantities

- $\hat{p}(A_t)$ which is an estimate of $\mu(A_t) = \mathbb{P}(X \in A_t)$, and
- $\hat{e}(A_t)$ which is an estimate of $e(A) = \mathbb{P}(g_t(X) \neq Y \mid X \in A_t)$.

Both of these are estimated based on the estimation points which arrive in A_t during its lifetime. From these two numbers we form the statistic $\hat{s}(A_t) = \hat{p}(A_t)\hat{e}(A_t)$ (with corresponding true value $s(A_t) = p(A_t)e(A_t)$) which is an upper bound on the improvement in error rate that can be obtained by splitting A_t.

Membership in the fringe is controlled by $\hat{s}(A_t)$. When a leaf is split it relinquishes its place in the fringe and the inactive leaf with the largest value of $\hat{s}(A_t)$ is chosen to take its place. The newly created leafs from the split are initially inactive and must compete with the other inactive leafs for entry into the fringe.

Unlike Domingos & Hulten (2000), who use this technique only as a heuristic for managing memory use, we incorporate the memory management directly into our analysis. The analysis in Appendix B shows that our algorithm, including a limited size fringe, is consistent.

4. Theory

In this section we state our main theoretical results and give an outline of the strategy for establishing consistency of our online random forest algorithm. In the interest of space and clarity we do not include proofs

in this section. Unless otherwise noted, the proofs of all claims appear in Appendix B.

We denote the tree partition created by our online random forest algorithm from t data points as g_t. As t varies we obtain a sequence of classifiers, and we are interested in showing that the sequence $\{g_t\}$ is consistent, or more precisely that the probability of error of g_t converges in probability to the Bayes risk, i.e.

$$L(g_t) = \mathbb{P}\left(g_t(X, Z) \neq Y \,|\, D_t\right) \to L^* \ ,$$

as $t \to \infty$. Here (X, Y) is a random test point and Z denotes the randomness in the tree construction algorithm. D_t is the training set (of size t) and the probability in the convergence is over the random selection of D_t. The Bayes risk is the probability of error of the Bayes classifier, which is the classifier that makes predictions by choosing the class with the highest posterior probability,

$$g(x) = \arg\max_k \mathbb{P}\left(Y = k \,|\, X = x\right) \ ,$$

(where ties are broken in favour of the smaller index). The Bayes risk $L(g) = L^*$ is the minimum achievable risk of any classifier for the distribution of (X, Y). In order to ease notation, we drop the explicit dependence on D_t in the remainder of this paper. More information about this setting can be found in Devroye et al. (1996).

Our main result is the following theorem:

Theorem 1. *Suppose the distribution of X has a density with respect to the Lebesgue measure and that this density is bounded from above and below. Then the online random forest classifier described in this paper is consistent.*

The first step in proving Theorem 1 is to show that the consistency of a voting classifier, such as a random forest, follows from the consistency of the base classifiers. We prove the following proposition, which is a straightforward generalization of a proposition from Biau et al. (2008), who prove the same result for two class ensembles.

Proposition 2. *Assume that the sequence $\{g_t\}$ of randomized classifiers is consistent for a certain distribution of (X, Y). Then the voting classifier, $g_t^{(M)}$ obtained by taking the majority vote over M (not necessarily independent) copies of g_t is also consistent.*

With Proposition 2 established, the remainder of the effort goes into proving the consistency of our tree construction.

The first step is to separate the stream splitting randomness from the remaining randomness in the tree construction. We show that if a classifier is conditionally consistent based on the outcome of some random variable, and the sampling process for this random variable generates acceptable values with probability 1, then the resulting classifier is unconditionally consistent.

Proposition 3. *Suppose $\{g_t\}$ is a sequence of classifiers whose probability of error converges conditionally in probability to the Bayes risk L^* for a specified distribution on (X, Y), i.e.*

$$\mathbb{P}\left(g_t(X, Z, I) \neq Y \,|\, I\right) \to L^*$$

for all $I \in \mathcal{I}$ and that ν is a distribution on I. If $\nu(\mathcal{I}) = 1$ then the probability of error converges unconditionally in probability, i.e.

$$\mathbb{P}\left(g_t(X, Z, I) \neq Y\right) \to L^*$$

In particular, $\{g_t\}$ is consistent for the specified distribution.

Proposition 3 allows us to condition on the random variables $\{I_t\}_{t=1}^{\infty}$ which partition the data stream into structure and estimation points in each tree. Provided that the random partitioning process produces acceptable sequences with probability 1, it is sufficient to show that the random tree classifier is consistent conditioned on such a sequence. In particular, in the remainder of the argument we assume that $\{I_t\}_{t=1}^{\infty}$ is a fixed, deterministic sequence which assigns infinitely many points to each of the structure and estimation streams. We refer to such a sequence as a *partitioning sequence*.

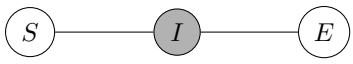

Figure 1. The dependency structure of our algorithm. S represents the randomness in the structure of the tree partition, E represents the randomness in the leaf estimators and I represents the randomness in the partitioning of the data stream. E and S are independent conditioned on I.

The reason this is useful is that conditioning on a partitioning sequence breaks the dependence between the structure of the tree partition and the estimators in the leafs. This is a powerful tool because it gives us access to a class of consistency theorems which rely on this type of independence. However, before we are able to apply these theorems we must further reduce our problem to proving the consistency of estimators of the posterior distribution of each class.

Proposition 4. *Suppose we have regression estimates, $\eta_t^k(x)$, for each class posterior $\eta^k(x) = \mathbb{P}(Y = k \,|\, X = x)$, and that these estimates are each consistent. The classifier*

$$g_t(x) = \arg\max_k \{\eta_t^k(x)\}$$

(where ties are broken in favour of the smaller index) is consistent for the corresponding multiclass classification problem.

Proposition 4 allows us to reduce the consistency of the multiclass classifier to the problem of proving the consistency of several two class posterior estimates. Given a set of classes $\{1, \ldots, c\}$ we can re-assign the labels using the map $(X, Y) \mapsto (X, \mathbb{I}\{Y = k\})$ for any $k \in \{1, \ldots, c\}$ in order to get a two class problem where $\mathbb{P}(Y = 1 \,|\, X = x)$ in this new problem is equal to $\eta^k(x)$ in the original multiclass problem. Thus to prove consistency of the multiclass classifier it is enough to show that each of these two class posteriors is consistent. To this end we make use of the following theorem from Devroye et al. (1996).

Theorem 5. *Consider a partitioning classification rule which builds a prediction $\eta_t(x)$ of $\eta(x) = \mathbb{P}(Y = 1 \,|\, X = x)$ by averaging the labels in each cell of the partition. If the labels of the voting points do not influence the structure of the partition then*

$$\mathbb{E}\left[|\eta_t(x) - \eta(x)|\right] \to 0$$

provided that

1. $\operatorname{diam}(A_t(X)) \to 0$ *in probability,*
2. $N^e(A_t(X)) \to \infty$ *in probability.*

Proof. See Theorem 6.1 in Devroye et al. (1996). □

Here $A_t(X)$ refers to the cell of the tree partition containing a random test point X, and $\operatorname{diam}(A)$ indicates the diameter of set A, which is defined as the maximum distance between any two points falling in A,

$$\operatorname{diam}(A) = \sup_{x,y \in A} ||x - y|| \ .$$

The quantity $N^e(A_t(X))$ is the number of points contributing to the estimation of the posterior at X.

This theorem places two requirements on the cells of the partition. The first condition ensures that the cells are sufficiently small that small details of the posterior distribution can be represented. The second condition requires that the cells be large enough that we are able to obtain high quality estimates of the posterior probability in each cell.

The leaf splitting mechanism described in Section 3.3 ensures that the second condition of Theorem 5 is satisfied. However, showing that our algorithm satisfies the first condition requires significantly more work. The chief difficulty lies in showing that every leaf of the tree will be split infinitely often in probability. Once this claim is established a relatively straightforward calculation shows that the expected size of each dimension of a leaf is reduced each time it is split.

So far we have described the approach to proving consistency of our algorithm with an unbounded fringe. If the tree is small (i.e. never has more leafs than the maximum fringe size) then the analysis is unchanged. However, since our trees are required to grow to unbounded size this is not possible.

In order to apply Theorem 5 in the case of an unbounded fringe we have shown that every leaf will be split in finite time with arbitrarily high probability. To extend consistency to this setting we need only show that the probability of an inactive leaf not being activated goes to zero as $t \to \infty$. This is sufficient, since once a leaf is activated it remains in the fringe until it is split and the argument from the unbounded fringe setting applies.

In order to show that any leaf will be eventually added to the fringe, we consider an arbitrary leaf A and show that we can make the probability that $\hat{s}(A)$ is not the largest $\hat{s}(A)$ value among inactive leafs arbitrarily small by making t sufficiently large.

These details are somewhat lengthy, so we refer the interested reader to Appendix B a full presentation, including proofs of the propositions stated in this section.

5. Experiments

In this section we demonstrate some empirical results in order to illustrate the properties of our algorithm. Code to reproduce all of the experiements in this section is available online[1].

5.1. Advantage of a Forest

Our first experiment demonstrates that although the individual trees are consistent classifiers, empirically the performance of the forest is significantly better than each of the trees for problems with finite data.

[1] https://github.com/david-matheson/rftk-colrf-icml2013

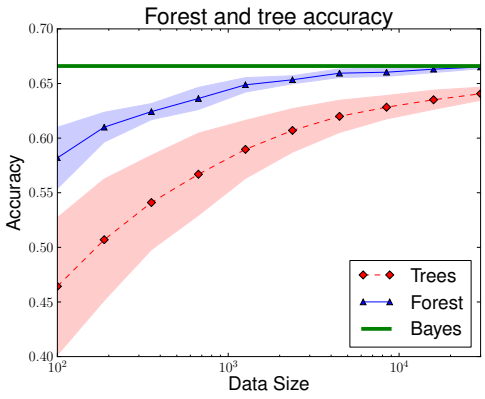

Figure 2. Prediction accuracy of the forest and the trees it averages on a 2D mixture of Gaussians. The horizontal line shows the accuracy of the Bayes classifier on this problem. We see that the accuracy of the forest consistently dominates the expected accuracy of the trees. Shaded regions show one standard deviation computed over 10 runs.

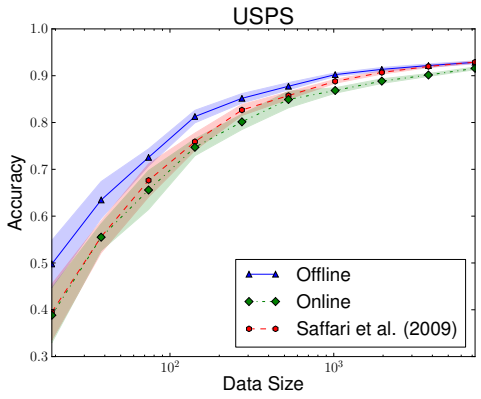

Figure 3. Comparison between offline random forests and our online algorithm on the USPS data set. The online forest uses 10 passes through the data set. The third line is our implementation of the algorithm from Saffari et al. (2009). Shaded regions show one standard deviation computed over 10 runs.

We demonstrate this on a synthetic five class mixture of Gaussians problem with significant class overlap and variation in prior weights. For this experiment we used 100 trees and set $\lambda = 1$, $m = 10$, $\tau = 0.001$, $\alpha(d) = 1.1^d$, $\beta(d) = 1000\alpha(d)$.

From Figure 2 it is clear that the forest converges much more quickly than the individual trees. Result profiles of this kind are common in the boosting and random forests literature; however, in practice one often uses inconsistent base classifiers in the ensemble (e.g. boosting with decision stumps or random forests where the trees are grown to full size). This experiment demonstrates that although our base classifiers provably converge, empirically there is still a benefit from averaging in finite time.

5.2. Comparison to Offline

In our second experiment, we demonstrate that our online algorithm is able to achieve similar performance to an offline implementation of random forests and also compare to an existing online random forests algorithm on a small non-synthetic problem.

In particular, we demonstrate this on the USPS data set from the LibSVM repository (Chang & Lin, 2011). We have chosen the USPS data for this experiment because it allows us to compare our results directly to those of Saffari et al. (2009), whose algorithm is very similar to our own. For both algorithms we use a forest of 100 trees. For our model we set $\lambda = 10$, $m = 10$, $\tau = 0.1$, $\alpha(d) = 10(1.00001^d)$ and $\beta(d) = 10^4\alpha(d)$. For the model of Saffari et al. (2009) we set the number of features and thresholds to sample at 10, the minimum information gain to 0.1 and the minimum number of samples to split at leaf at 50. We show results from both online algorithms with 15 passes through the data.

Figure 3 shows that we are able to achieve performance very similar to the offline random forest on the full data. The performance we achieve is similar to the performance reported by Saffari et al. (2009) on this data set.

5.3. Microsoft Kinect

For our final experiment we evaluate our online random forest algorithm on the challenging computer vision problem of predicting human body part labels from a depth image. Our procedure closely follows the work of Shotton et al. (2011) which is used in the commercially successful Kinect system. Applying the same approach as Shotton et al. (2011), our online classifier predicts the body part label of a single pixel P in a depth image. To predict all the labels of a depth image, the classifier is applied to every pixel in parrallel.

For our dataset, we generate pairs of 640x480 resolution depth and body part images by rendering random poses from the CMU mocap dataset. The 19 body parts and one background class are represented by 20 unique color identifiers in the body part image. Figure 4 (left) visualizes the raw depth image, ground truth body part labels and body parts predicted by our classifier for one pose. During training, we sample 50 pix-

Figure 4. **Left:** Depth, ground truth body parts and predicted body parts. **Right:** A candidate feature specified by two offsets.

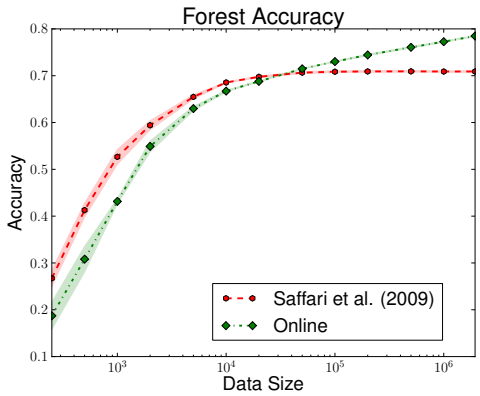

Figure 5. Comparison of our online algorithm with Saffari et al. (2009) on the kinect application. Error regions show one standard deviation computed over 5 runs.

els without replacement for each body part class from each pose; thus, producing 1000 data points for each depth image. During testing we evaluate the prediction accuracy of all non background pixels as this provides a more informative accuracy metric since most of the pixels are background and are relatively easy to predict. For this experiment we use a stream of 2000 poses for training and 500 poses for testing.

Each node of each decision tree computes the depth difference between two pixels described by two offsets from P (the pixel being classified). At training time, candidate pairs of offsets are sampled from a 2-dimensional Gaussian distributions with variance 75.0. The offsets are scaled by the depth of the pixel P to produce depth invariant features. Figure 4 (right) shows a candidate feature for the indicated pixel. The resulting feature value is the depth difference between the pixel in the red box and the pixel in the white box.

In this experiment we construct a forest of 25 trees with 2000 candidate offsets (λ), 10 candidate splits (m) and a minimum information gain of 0.01 (τ). For Saffari et al. (2009) we set the number of sample points required to split to 25 and for our own algorithm we set $\alpha(d) = 25 \cdot (1.01^d)$ and $\beta(d) = 4 \cdot \alpha(d)$. With this parameter setting each active leaf stores $20 \cdot 10 \cdot 2000 \cdot 2 = 400,000$ statistics which requires 1.6MB of memory. By limiting the fringe to 1000 active leaves our algorithm requires 1.6GB of memory for leaf statistics. To limit the maximum memory used by Saffari et al. (2009) we set the maximum depth to 8 which uses up to $25 \cdot 2^8 = 6400$ active leaves which requires up to 10GB of memory for leaf statistics.

Figure 5 shows that our algorithm achieves significantly better accuracy while requiring less memory.

6. Discussion and Future Work

In this paper we described an algorithm for building online random forests and showed that our algorithm is consistent. To the best of our knowledge this is the first consistency result for online random forests.

Growing trees online in the obvious way requires large amounts of memory, since the trees must be grown breadth first and each leaf must store are large number of statistics in each of its potential children. We incorporated a memory management technique from Domingos & Hulten (2000) in order to limit the number of leafs in the fringe of the tree. This refinement is important, since it enables our algorithm to grow large trees. The analysis shows that our algorithm is still consistent with this refinement.

The analysis we presented in this paper shows that our algorithm is consistent, but does not give rates of convergence to the Bayes risk. Analyzing the convergence rate of our algorithm is clear direction for future work, but the way to proceed does not appear to be straightforward.

Finally, our current algorithm is restricted to axis aligned splits. Many implementations of random forests use more elaborate split shapes, such as random linear or quadratic combinations of features. These strategies can be highly effective in practice, especially in sparse or high dimensional settings. Understanding how to maintain consistency in these settings is another potentially interesting direction of inquiry.

Acknowledgements

Some of the data used in this paper was obtained from `mocap.cs.cmu.edu` (funded by NSF EIA-0196217).

References

H. Abdulsalam. *Streaming Random Forests*. PhD thesis, Queens University, 2008.

G. Biau. Analysis of a Random Forests model. *JMLR*, 13 (April):1063–1095, 2012.

G. Biau, L. Devroye, and G. Lugosi. Consistency of random forests and other averaging classifiers. *JMLR*, 9:2015–2033, 2008.

A. Bifet, G. Holmes, and B. Pfahringer. MOA: Massive Online Analysis, a framework for stream classification and clustering. In *Workshop on Applications of Pattern Analysis*, pp. 3–16, 2010.

A. Bifet, E. Frank, G. Holmes, and B. Pfahringer. Ensembles of Restricted Hoeffding Trees. *ACM Transactions on Intelligent Systems and Technology*, 3(2):1–20, February 2012.

A. Bifet, G. Holmes, and B. Pfahringer. New ensemble methods for evolving data streams. In *ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, 2009.

A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *International Conference on Computer Vision*, pp. 1–8, 2007.

L. Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001.

L. Breiman. Consistency for a Simple Model of Random Forests. Technical report, University of California at Berkeley, 2004.

L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. CRC Press LLC, Boca Raton, Florida, 1984.

C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2-3):81–227, 2011.

D. Cutler, T. Edwards, and K. Beard. Random forests for classification in ecology. *Ecology*, 88(11):2783–92, November 2007.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, New York, USA, 1996.

P. Domingos and G. Hulten. Mining high-speed data streams. In *International Conference on Knowledge Discovery and Data Mining*, pp. 71–80. ACM, 2000.

J. Gama, P. Medas, and P. Rodrigues. Learning decision trees from dynamic data streams. In *ACM symposium on Applied computing*, SAC '05, pp. 573–577, New York, NY, USA, 2005. ACM.

R. Genuer. Risk bounds for purely uniformly random forests. Technical report, Institut National de Recherche en Informatique et en Automatique, 2010.

R. Genuer. Variance reduction in purely random forests. *Journal of Nonparametric Statistics*, 24(3):543–562, 2012.

H. Ishwaran and U. Kogalur. Consistency of random survival forests. *Statistics and Probability Letters*, 80:1056–1064, 2010.

Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. Technical Report 1055, University of Wisconsin, 2002.

N. Meinshausen. Quantile regression forests. *JMLR*, 7: 983–999, 2006.

N. Oza and S. Russel. Online Bagging and Boosting. In *Artificial Intelligence and Statistics*, volume 3, 2001.

A. Prasad, L. Iverson, and A. Liaw. Newer Classification and Regression Tree Techniques: Bagging and Random Forests for Ecological Prediction. *Ecosystems*, 9(2):181–199, March 2006. ISSN 1432-9840.

A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1393–1400. IEEE, 2009.

J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. *CVPR*, pp. 1297–1304, 2011.

V. Svetnik, A. Liaw, C. Tong, J. Culberson, R. Sheridan, and B. Feuston. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947–58, 2003.