**Recursive Random Forests by using Random Forests**

Arnav Jain, Vishal Kotha

Quarter 2 Intermediate Report

Machine Learning 1, Dr. Yilmaz Period 5

**Abstract**

Random forests currently utilize an ensemble of decision trees. In this paper, we propose using random forests of random forests in order to potentially achieve better performance. A parameter will be used to determine the depth for which random forests of random forests will be used, before standard random forests with decision trees are used. The algorithm’s performance and build time were evaluated using two different datasets: iris and kick. The goal of the proposed model is to increase the accuracy and other performance metrics compared to traditional random forests and modified decision trees like AdaBoost, in exchange for the speed.

**Introduction**

Currently, random forests are a state-of-the-art machine learning algorithm. However, some problems are too complex for them, resulting in poor performance. Furthermore, noisy data results in overfitting of the training data, meaning the algorithm generalizes poorly. Using an ensemble of “stronger” learners (random forests versus decision trees) may both improve performance and generalizability as compared to the random forest algorithm. The input to our algorithm is a list of attributes. We will evaluate performance using both the iris dataset, a small dataset that’s uniformly distributed, and the kick dataset, a large dataset with a non-uniform distribution. Any quantitative, continuous attributes will be made discrete using binning. We then used our modified random forest to output either a flower type (for iris dataset) or whether the car is a “kick” or not (for kick dataset). This means there are three possible outputs for the iris dataset and two possible outputs for the kick dataset.

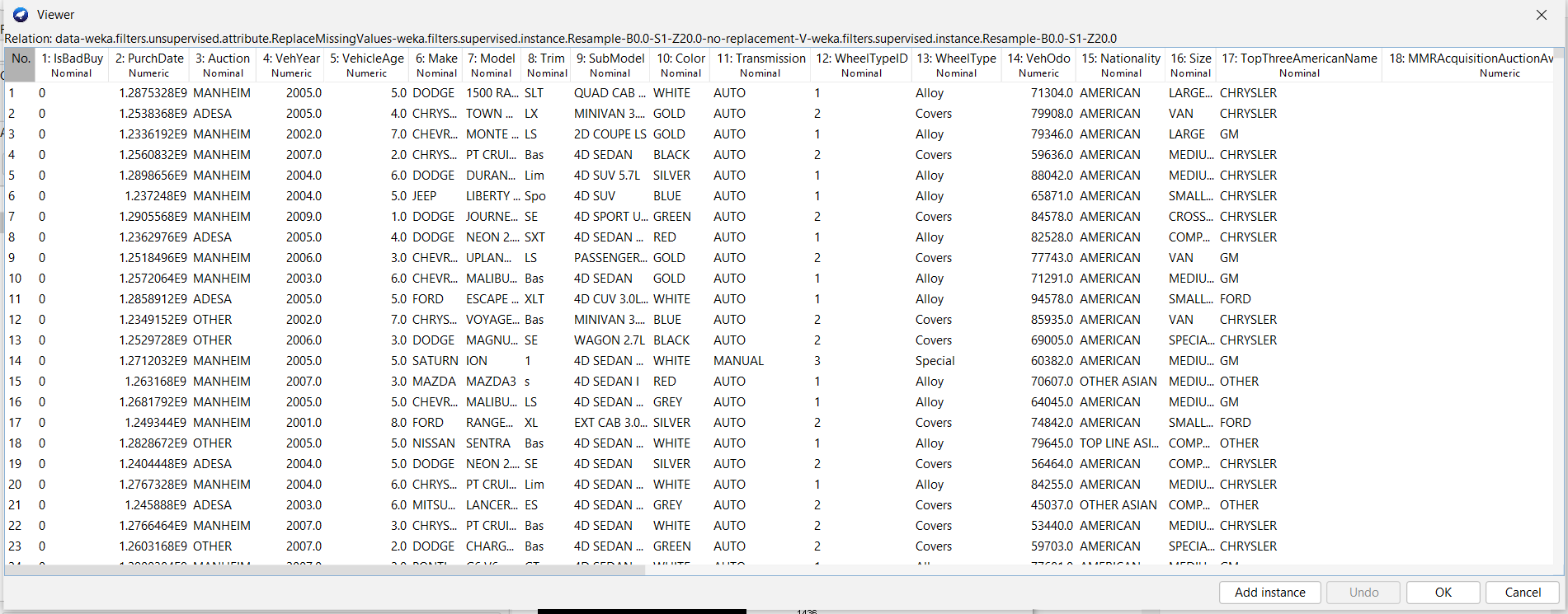
**Related work**

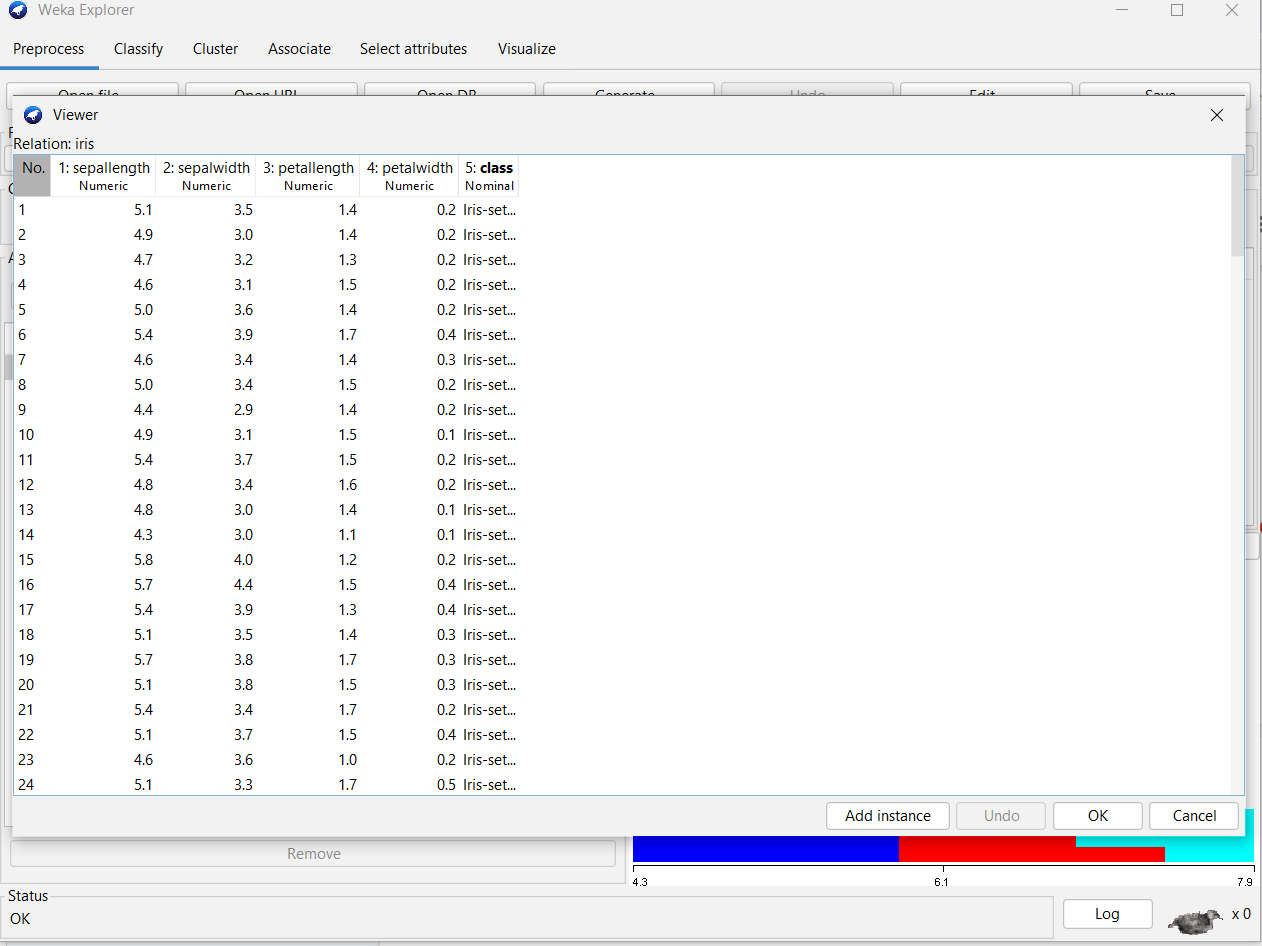
This algorithm builds-off the original Random Forest algorithm first proposed in Breiman’s 2001 paper [[1]](https://drive.google.com/file/d/1cTOwboU8uI3wDB8iAeQGQaLYDTXfQESf/view?usp=sharing), which is a state-of-the-art ensemble classification algorithm. Instead of an ensemble of decision trees, up to a certain depth, we use an ensemble of random forests before using the standard random forest algorithm until all the allocated attributes have been used for splitting. However, everything else remains the same, including the metric for evaluating split performance and data aggregation. The original Random Forest utilizes a Decision Tree, which is constructed using the ID3 algorithm outlined in Quinlan’s 1986 research paper [[4]](https://drive.google.com/file/d/1druopRWAh8sVPE7NEHU2xEa5r8D510pV/view?usp=sharing). Our algorithm utilizes decision trees at its core, just like the random forest. However, by utilizing an ensemble of decision trees, both random forests and our modified random forest both perform and generalize better. An improvement to the random forest algorithm was outlined in Robnik-Šikonja’s 2004 research paper [[5]](https://drive.google.com/file/d/1KAOhjOHES_oiZZmt-8FycPasEdrMLVrm/view?usp=sharing). His approach involved two phases: instead of using Gini index as the sole evaluation heuristic for every tree, create a fifth using these five metrics respectively: Gini index, gain ratio, MDL, Myopic ReliefF, and ReliefF. Furthermore, he weighted various trees’ importance in the overall classification by using their performance on the most similar instances in the training dataset. This combination improved the random forest’s performance, with a drawback of computation time. We stuck with the standard random forest for our algorithm, but using this modified Random Forest is an avenue for further potential performance improvements. The aforementioned approach improved performance at the cost of computation time; however, another avenue for improvement is speeding up the computation of Random Forests while minimizing performance decrease. One approach to do so was proposed by Paul et al. in their 2018 research paper [[3]](https://drive.google.com/file/d/11cJO7I8FZzp12ccL8PNBqm6NYDDt8kax/view?usp=sharing). They determine which features are important and unimportant, and using that, develop an upper limit for the number of trees to be included in the model. Furthermore, they reduce some features, resulting in a reduced but important feature set. Both improvements speed up the build time while minimizing performance decrease. They also prove that further attribute reduction and additional trees don’t significantly improve performance. Utilizing these metrics to reduce computation time could be another potential implementation for our algorithm in the future.

AdaBoost was outlined in Freund et. al’s 1999 paper [[2]](https://drive.google.com/file/d/1P4v9C4mJr2xIOUQcMw9PiZExe22V5ZSu/view?usp=sharing). AdaBoost can be applied to create an ensemble of decision trees. All the data subsets are created using sampling with replacement. The first decision tree is created on a random subset; each subsequent tree created using a data subset weighted to contain the attributes most incorrectly classified by the previous decision tree. This process is repeated until the number of decision trees is satisfied and the output of all these decision trees is aggregated. It’s similar to our approach in that we both ensemble decision trees, but the method in which these decision trees are created is different. It’s a clever approach, using successive decision trees to ensure the faults of the preceding ones are accounted for. Each tree’s output can also be weighted based on its training dataset accuracy, while our current approach weights each tree equally.

**Dataset and Features**

Before we could create and train our models, we had to pre-process data: this included things like sampling our data and splitting the data into a train and test set. We used a larger, non-uniform dataset and a smaller, uniform dataset to ensure we tested our model with a variety of datasets. For the kick dataset (obtained from <https://www.openml.org/search?type=data&sort=runs&id=41162&status=active>), we first filled the missing values in the original data using the ReplaceMissingValues function in WEKA. Its methods of imputing are mean for numerical variables and mode for categorical variables. Our original dataset contained 72,893 instances: each instance represents a car. Due to the large dataset size, we first did a random sample without resampling, taking only 20% of the original data. The new sample contained 14,596 instances (12,801 class label 0 and 1,795 of class label 1, which is 87.702% of class label 0). Then, we split this sample into train and test sets using an 80-20 split. The training set contains 11,676 instances (10,240 of class label 0 and 1,436 of class label 1, which is 87.701% of class label 0). The test set contains 2,920 instances (2,561 of class label 0 and 359 of class label 1, which is 87.705% of class label 0). The original dataset, containing 72,983 instances, had 64,007 instances with a class label of 0. This is about 87.701%. The sample, training dataset, and testing dataset all had class imbalances of similar percentage distributions. There were 32 attributes and we did not use any attribute selection algorithms; the attributes were randomly selected by the random forest algorithm. We also did not do any data normalization; we one-hot encoded the categorical variables and, using three equal width bins, smoothed the numerical variables. Here is a snippet of the kick dataset (“0” represents not a kick while “1” represents a kick):



For the iris dataset (built into WEKA; can be found at <https://www.openml.org/search?type=data&sort=runs&id=61&status=active>), it contains 150 instances: each one represents a flower. There are exactly fifty instances of each of the three flower types (Iris virginica, Iris versiocolor, and Iris setosa), meaning the dataset is uniformly distributed. There are four quantitative attributes: sepal length, sepal width, petal length, and petal width. To discretize the data, we binned each attribute using three equal width bins. There were no missing values, so no imputation was necessary. We did not do any data normalization. We used stratified random sampling and an 80-20 train-test split: there were 120 instances in the training dataset and 30 instances in the testing dataset. The training dataset contained 40 instances from each class while the testing dataset contained 10 instances from each class. This means there were 33.33% of each class in both the training and testing dataset, which is consistent with the uniform distribution of the original dataset. Here is a snippet of the iris dataset:

**Methods [≈ 1 − 1.5 pages]**

To create this random forest of random forests, we begin by using the same recursive structure of a random forest. A random forest works by taking the dataset, and bagging and bootstrapping the data to get multiple views of the data, on which a decision tree is trained by recursively splitting the data to make it pure.When generating a prediction, a random forest combines the output values of all the decision trees using either mean, median, or mode depending on the input type.

Figure 1: A diagram depicting the structure of a random forest.

Our proposed random forest of random forests uses the ensemble structure of a random forest. It works by bagging and bootstrapping the dataset to get multiple views, just as the original random forest did. Then, rather than training a decision tree on each view, it trains a random forest on the bootstrapped data. We can control the depth of this recursive random forest which will determine when it switches back to a forest of decision trees. This proposed model has the potential to outperform traditional random forests because it has access to more views of the dataset, and rather than ensembling multiple weak models to create a classification, it ensembles multiple ensembles of weak models, resulting in a more robust model. However, the expected accuracy increase will come at the tradeoff of speed and space. This is why the user has the ability to control the depth of the random forest layers so they can tune these parameters to have a balance of speed and accuracy.



Figure 2: A diagram depicting the proposed structure of the random forest of random forests with random\_forest\_depth = 1.

In order to maintain consistency, both models will still use gini impurity as the information gain metric and use the same ensembling methods. Furthermore, it will run on the same hardware and be given the same random state when doing bagging and bootstrapping.

**Experimental Design**

As stated previously, we will test our model on multiple different datasets. The two datasets we are using for testing are the Iris dataset, because of its small size and simple classification, and the Kick dataset, which is more representative of a large, real world dataset, with missing features and class imbalance. Using these two different datasets allows us to gauge the strengths and weaknesses of our proposed model.

In our experiment, we will measure the time taken to run the models, and various accuracy metrics, including accuracy, precision, recall, and the confusion matrix. In order to measure the time, we will run the models on the same machine available on Google Colab notebooks and use the built in python time package to measure when the model begins training and when it finishes. This will minimize variability in these measurements.

We will compare our proposed model with a baseline random forest for these metrics, as well as improved decision trees like AdaBoost. We will also experiment with how changing the depth of the random forest layers will affect the performance. A significant speedup was achieved by using sklearn’s RandomForestClassifier in the final layer, rather than our implementation from scratch.

**Results**

| **Model** | **Accuracy** | **TP Rate** | **FP Rate** | **ROC Area** | **Time Taken (s)** |
| --- | --- | --- | --- | --- | --- |
| **Random Forest**  **(trees = 100)** | 100.0% | 1.0 | 0.0 | 1.00 | 0.61937 |
| **AdaBoost** | 100.0% | 1.0 | 0.0 | 1.00 | 0.1816 |
| **Random Forest of Random Forests (depth = 1)** | 100.0% | 1.0 | 0.0 | 1.00 | 5.0684 |
| **Random Forest of Random Forests (depth = 2)** | 100.0% | 1.0 | 0.0 | 1.00 | 25.3216 |

Table 1: Results on the iris dataset. Note: TP and FP are weighted

| **Model** | **Accuracy** | **TP Rate** | **FP Rate** | **ROC Area** | **Time Taken (s)** |
| --- | --- | --- | --- | --- | --- |
| **Random Forest (trees = 100)** | 87.4957% | 0.875 | 0.851 | 0.649 | 1.64 |
| **AdaBoost** | 88.1507% | 0.877 | 0.877 | 0.679 | 5.59 |
| **Random Forest of Random Forests (depth = 1)** | 88.8698% | 0.8913 | 0.8791 | 0.701 | 11.52 |
| **Random Forest of Randoms Forest (depth = 2)** | 88.2642% | 0.8898 | 0.8801 | 0.694 | 40.18 |

Table 2: Results on the “kick” dataset. Note: TP and FP are weighted

**Discussion**

Based on the results in Tables 1 and 2, we can see that the random forests and AdaBoosts perform well, quickly with accuracies of 100% on the small Iris dataset, and around 87% on the “kick” dataset. On the iris dataset, our model achieves the same 100% accuracy at both random\_forest\_depth = 1 and random\_forest\_depth = 2. This shows its ability to learn on small, balanced datasets with simple correlations between input features and the output. However, it still is slower by a factor of 10x compared to a traditional random forest. This makes sense, as we trained 10 random forests in the first layer, which then branched to normal random forests in the next. However, the “kick” dataset has a large class imbalance, and the previous models ended up predicting the majority class every time. Our model had a slight improvement over the previous models in this department, with improved predictions on predicting the “kick” class. This is likely because the model is more complex and can learn deeper correlations than just predicting a majority class. This comes with the tradeoff of speed, resulting in a slowdown which becomes more evident as we increase the depth of random forests. Our models can be optimized for speed, as sklearn utilizes many computational speed ups to make their models faster than models coded from scratch. This is a potential area of improvement for our model.

**Conclusion**

In this paper, we propose a variant of random forests, random forest of random forests, which aims to increase the accuracy and robustness of random forests in exchange for speed. Our results show success on small datasets as well as improvement on larger, real-world datasets. It can learn complex patterns in a dataset without a significant slowdown in computation time.

**Works Cited**

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.

Paul, A., Mukherjee, D. P., Das, P., Gangopadhyay, A., Chintha, A. R., & Kundu, S. (2018). Improved random forest for classification. *IEEE Transactions on Image Processing*, *27*(8), 4012-4024.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106.

Robnik-Šikonja, M. (2004, September). Improving random forests. In *European conference on machine learning* (pp. 359-370). Springer, Berlin, Heidelberg.