Trees of Predictors Implementation and Comparison with other Ensemble Methods

Filza Mazahir   Tarneem Barayyan

Abstract. This project is an implementation of a new supervised ensemble learning method called Trees of Predictors (ToPs), and its comparison with other well-known tree ensemble methods, that is, Random Forest, Extra Tree and AdaBoost. The scope of this project is to solve binary classification problem. Trees of Predictors (ToPs) works by creating a model that applies different predictive models to different subsets of the data. It does so by creating a tree that has predictors associated with each node, which is derived by recursively splitting the nodes using a specific feature and a threshold that is chosen based on minimum logarithmic loss. The tree is then further optimized by assigning weights to the predictors on the path for the final prediction. The evaluation is done using metrics of logarithmic loss, area under ROC curve, accuracy, training time, precision, recall, and f1-score values. It is shown through these metrics that ToPs has a very high training time compared to the other classifiers, but does well in terms of logarithmic loss, area under ROC curve, and accuracy. Two instances of ToPs were used for evaluation, one that uses a linear classifier as its base learner, and other that uses Random Forest, Extra Trees and AdaBoost as its base learner. Comparing the two showed that a good choice of base learners impacts the performance of ToPs greatly.

Keywords: ToPs, Tree-based, Ensemble Tree, Data Classification.

1 Introduction

Ensemble is a concept in which the idea is to train multiple models using the same learning algorithm. The aim of ensemble method is to decrease variance, as well as bias, and improve prediction [1]. The techniques to ensemble include bagging, boosting, and staching. Tree-based ensemble method uses decision trees as its base learners.

In this algorithm-oriented project to solve a binary classification problem, a new ensemble method called Trees of Predictors (ToPs) is implemented from scratch. This model differs from the other ensemble methods as it applies different predictive models to different subsets of the feature space [1]. It is also different than other tree-based methods, as usually tree-based approaches create splits in order to maximize homogeneity, ToPs creates splits to minimize loss value.

Since ToPs is similar to tree-based ensemble methods, it is compared with other well-known tree-based ensemble methods, which are Random Forest, Extra Trees, and AdaBoost.

2 Literature Review

2.1 Random Forest

Random Forest ensemble method consists of a collection of decision trees, where it trains each of these to get more accurate prediction. Each decision tree uses random vector generation to grow the tree. It then finds the popular class of the given input, and represents a unit vote [2]. It does so by selecting random samples from the given dataset, and then randomly selecting the number of input variables to split at each node. After growing all decision trees, the majority votes are considered to finalize the prediction class of the test object [3].

Random Forest can be used for both classification and regression problems. In classification problem, majority voting is used, while in regression a weighted average is considered to find the final prediction. For sampling, Random Forest uses bagging with replacement to produce various subsets for each decision tree [2].

As selection of sample and features are done randomly, it results in a model that is robust to outliers and overfitting. Another reason for the model to not overfit the data is that each tree is an independent model.

2.2 Extremely Randomized Tree Classifier

Extremely Randomized Tree classifier is another tree-based ensemble methods that can be used for both classification and regression problems [4]. This model differs than Random Forest as it does not sample the dataset to grow the decision trees; instead, it uses the whole dataset. Also, features and thresholds are randomly selected for each node to define the split at each node. At the end, majority voting is considered to make the final prediction [4].

As a result, Extra Tree shows significant improvement in term of accuracy and simplicity of implementation because it removes the needs of optimization of the decentralized threshold. Moreover, using entire dataset to train the trees minimizes bias. Similar to Random Forest, Extremely Randomized Trees model is not sensitive to outlier. In terms of computational cost, it is much faster than single tree and can get higher accuracy. It is also robust to irrelevant and redundant input, and it has very low variance.

2.3 AdaBoost Classifier

Boosting is an ensemble method that uses weak classifiers to come up with strong classifiers; it trains weak classifiers using training set, so it creates multiple models where each model corrects the errors (or misclassification) of the previous model, thus creating such a stronger classifier at the end [5].

In AdaBoost, the given dataset is sampled, and weight is assigned to each sample set. As AdaBoost designed for binary classification [6], each learner will make a decision for the input. Then, the misclassification error is calculated, and weight is updated as it goes higher for those sets that has misclassification, so these sets will appear in the second iteration. In other words, higher weight is given for wrong prediction while low weight is given for correct prediction, hence boosting the performance of the weak classifier. This process continues until no further improvement can be done, and final output is determined.

AdaBoost has many advantages; it is simple, easy to implement, and does not require prior knowledge about weak learners as well as it tries to correct its mistake. However, AdaBoost is sensitive to noise, and weak learners might lead to overfitting.

3 Description of Method Selected

3.1 Trees of Predictors

Tree of Predictors (ToPs) is a novel method of ensemble learning that applies different predictive models to different subsets of the dataset [1]. It differs than existing method as it constructs a tree using a splitting criterion that depends both on labels and predictions of its base learners [1]. The subset of feature tree along with predictor for each node in the tree. Even though ToPs is similar to tree-based ensemble method, ToPs split feature space that minimizes logarithmic loss instead of homogeneity. ToPs splits feature space that maximize predictive accuracy instead of homogeneity.

ToPs divides training sets into further two validation sets. The first validation set is used during the construction of the tree (in the first step) while second validation set is used during weight assignment (in the second step).

ToPs algorithm consists of three steps. The first step defines how to create a tree. The tree is created by iterating through all the features, then iterating through each threshold for that feature in the range of 0.1 to 0.9, and then iterating through all classifiers in the base learner set. The log loss value is compared for these iterations and the splitting decision is made based minimum logarithmic loss value. For features with binary class, threshold value of 0.5 is used.

After growing the tree, step two assigns weight for each predictor on the path from root node to leaf node, such that the log loss is minimized. In this step, second validation set is used for the computation of log loss. The weights are assigned such that the sum of weights from root node to leaf node is 1. Finally step three is testing, where splitting of the data is done based on the features, threshold and predictor found in step 1, and a final prediction is made by using the optimized weights found in step 2 for the predictions on root to leaf path.

Figure 1 illustrates how ToPs works. Each node has a different instance of a classifier trained (shown), with different features and threshold selected (not shown in figure). As an example, the weight for root to the left most leaf node is also shown with *'w'*.

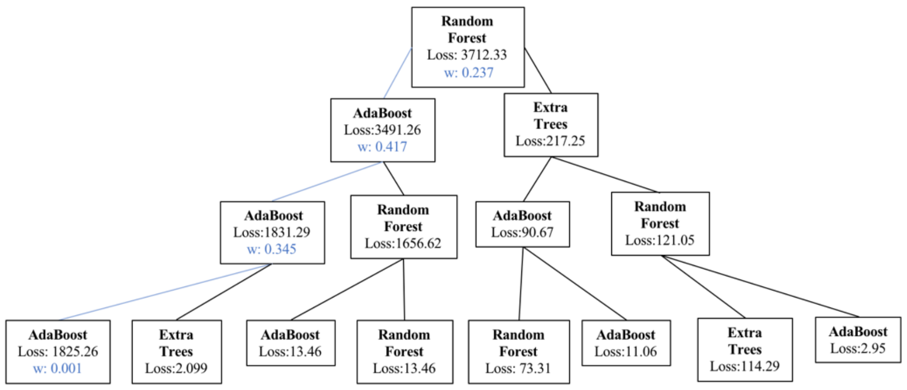

Figure 1: Tree of Predictors, using Random Forest, Extra Trees, and AdaBoost as base learners

The ToPs method does not take any extra parameters, but a maximum depth parameter was introduced to avoid overfitting, so the ToPs instance created with Random Forest, Extra trees, and AdaBoost had a maximum depth of 3. The parameters used for the base learners for ToPs was the same as outlined in section 3.2.

3.2 Other Ensemble Tree Methods

The other ensemble tree methods selected for comparison are Random Forest, Extra Trees, and AdaBoost classifier. Table 1 shows the parameters used for each of these methods.

Table 1. Ensemble Classifiers parameters values.

Ensemble Classifier	Parameters
Random Forest	n_estimators=50, max_depth=8
Extra Tree	n_estimators=50, max_depth=8
AdaBoost	n_estimators=50

The number of trees is set as 50 for all these classifiers. Considering that the dataset has over 37,000 instances, 50 seems a sufficient number to use. The maximum depth of these trees in the Random Forest and Extra Tree classifiers were set to 8, which is expected to be sufficient as doing more would mean overfitting the data.

4 Implementation

4.1 Dataset and Preprocessing

The dataset selected for this project is Online News Popularity dataset from the UCI Machine Learning Repository [7]. It has 39797 instances and 58 predictive features. The target feature is the number of shares for each article, which has been converted to binary classification as values greater or equal to1400 are classified as “popular” and values less than 1400 are classified as “unpopular”. This results in balanced classes as the ratio between popular and unpopular is 1.12 : 1.

The given dataset has a lot of outliers. Removing them all meant having a very imbalanced class, so only some outliers, defined as values more than 5 standard deviations away from the mean were removed. Min-max normalization is applied so that all the features have the same scale. The main reason for using min-max normalization with a range of 0 to 1 as compared to z-score method was to make it easier to do the ToPs implementation, as the threshold used for all features could then be between 0.1 to 0.9.

Feature reduction is done by finding key features and dropping the least important ones. Using Random Forest classifier and its feature selection model, a threshold of 0.02 was used for feature importance, hence reducing the total number of features to 15. The following are the 15 features used: data_channel_is_entertainment, data_channel_is_socmed, data_channel_is_world, kw_avg_min, kw_min_avg, kw_max_avg, kw_avg_avg, self_reference_min_shares, self_reference_max_shares, self_reference_avg_sharess, weekday_is_saturday, is_weekend, LDA_01, LDA_02, and LDA_04.

Finally, the given dataset is split into train and test sets, assigning 20% for testing, and 80% for training.

4.2 Tree of Predictors Implementation

The Tree of Predictors classifier is implemented in a ToPs class that is initialized by passing a training dataset, testing dataset, and a list of classifiers to it. The list of classifiers are the base learners that are used for that particular instantiation of ToPs. It utilizes another class Node for each node that it creates in the tree. A node is initialized using the Node class by passing to it the training data, first validation data, second validation data, test data, predictor used on that node, and current depth of that node in relation to the tree in ToPs. The Node class also has left and right attributes for its children, a feature to split, threshold, and predictor name. The leaf nodes also have a list of all predictors on their path from the root, and optimized weights for each predictor.

When an instance of ToPs is initialized, it first splits the training data further to get two validation sets. After that, it creates a root node that has the entire training dataset available to it. When creating the root node, it iterates through the list of classifiers passed to it, trains each of them on the training set, and checks the log loss value on the first validation set. It then picks its predictor as the classifier instance that gave it the minimum log loss value.

After creating the root node, it creates a ToPs tree. It does that by calling a recursive function that creates a sub tree to its root node, and then calling another recursive function which adds weights to the predictors on root to leaf path. In the function that creates a sub tree, there is a triple nested loop. In the first loop, it iterates through all the features in the given dataset, then for each feature, it iterates through a range of thresholds from 0.1 to 0.9. If the feature is of a binary class, then only a threshold of 0.5 is used. Then for each threshold, it iterates through the list of classifiers used for this ToPs instance. In all these iterations, the node is split into a left and right node, and their log loss value is calculated on the first validation set. The feature, threshold, and predictor with the minimum log loss is kept track of through the iterations, and then once all the iterations are done, the children are assigned to the node which area created by a split at that particular feature, threshold and predictor. The combined log loss value of the children and parent node is compared, and a stopping criterion is that if the combined log loss of children is smaller than the parent's, then the splitting stops. Another stopping criterion is if the maximum depth of the tree is achieved. This maximum depth parameter was added here to decrease the training time for a ToPs classifier, as without it, the instantiation of ToPs with three classifiers took longer than 12 hours and went up to a depth of 28, which was overfitting.

Once a tree is created, weights are added to the predictors from root to leaf node. This is done by traversing the tree from root to leaf, and then making a list of all the predictors on path at the leaf node. After that, at the leaf node, a prediction probability is computed from each of the predictors on the second validation set. An initial equal weight is assigned to each predictor at this point. Based on the prediction probability of each of these predictors, an optimize function is used that assigns the weight such that it minimizes the log loss value on these predictions, and the sum of the weights on each root to leaf path is 1. These optimized weights are then saved on the leaf node.

The third step of the ToPs implementation is to then test it using the test data set. This is done by splitting the test data set at the feature to split and threshold found in the first step that gave the minimum log loss, then training the chosen predictor on it. Then it traverses the tree created using the test data set, and adds weights to the predictors on root to leaf path using the optimized weights saved on the leaf node. Based on that, it gives a final prediction probability for the test data set. This prediction probability along with the true values of the test data set is returned as the result. The reason true test values are returned as well is because the order of instances changed during the implementation as the dataset was being split, so the true test values give the same order as the result, making it easier to run evaluation metrics on them.

4.3 Software and Data structures used

The implementation is done using Python 3.6, with the help of pandas, numpy, scipy, and scikit-learn libraries. The data structure of pandas data frame is used for all the training, validation and test data sets at each node. The classifiers used as base learners for the ToPs method, and the logarithmic loss calculated for the validation sets are called through the scikit-learn library. Numpy library is used to create the list of optimized weights on the leaf node, to make mathematical calculations such as matrix multiplication easier. The optimize function used to optimize the weights is called upon through the scipy library.

4.4 Overall Program Structure

First, data preprocessing is done as outlined in section 4.1, then Random Forest classifier, AdaBoost classifier, and Extra Trees classifier are imported from scikit-learn and trained on the datasets. The Trees of Predictors classifier is implemented separately in a class called ToPs as outlined in section 4.2. This class is imported into the program, and then two instances of ToPs were created and trained. First one uses linear classifier (with stochastic gradient descent) as its base learner, and the second one uses three predictors using Random Forest, Extra Tree and AdaBoost as its base learners.

These classifiers are called in a for loop to do multiple experiments, and evaluation metrics were then calculated on all these classifiers, while storing the metric in a nested dictionary to make it easier to compute the mean and standard deviation after.

5 Testing and Evaluation Results

For testing purposes, all these ensemble methods were compared to a baseline classification method of Nearest Neighbors. The experiment was run 20 times to obtain stable and accurate results. As mentioned earlier in section 4, testing was done on the Online News Popularity dataset, where 20% of the data set was used for testing.

In order to compare ToPs method with other tree-based ensemble method, the following performance metrics are used to test each method: logarithmic loss, area under ROC curve, accuracy, training time, precision, recall, and f1-score. The results are shown in tables 2 and 3.

Table 2. Area under ROC (AUC), Log Loss, Accuracy, and Training Time for all methods

Classifier	AUC	Log Loss	Accuracy	Training Time
K Nearest Neighbors	0.648 ($1.11\text{x}10^{16}$)	2.712 (0.00)	61.46% (0.00)	4.0003 sec (0.1004)
Random Forest	0.709 (0.0006)	0.623 (0.004)	65.28% (0.0017)	1.429 sec (0.0333)
Extra Tree	0.694 (0.0008)	0.636 (0.0003)	64.02% (0.0025)	0.338 sec (0.0148)
AdaBoost	0.708 ($1.11\text{x}10^{-16}$)	0.689 ($4.97\text{x}10^{-17}$)	65.29% (0.00)	1.514 sec (0.0296)
ToPs (linear classifier)	0.697 (0.0045)	0.633 (0.002)	65.04% (0.0055)	13.54 sec (4.3209)
ToPs (3 classifiers)	0.708 (0.0017)	0.623 (0.0006)	65.27% (0.0013)	10.39 min (4.3209)

Table 3. Precision, Recall and F1-Score values for all classifiers.

Classifier	Precision-Popular	Precision-Unpopular	Recall-Popular	Recall-Unpopular	F1-Score-Popular	F1-Score Unpopular
K Nearest Neighbors	0.632 (1.11e^{-16})	0.594 ($1.11\text{x}10^{-16}$)	0.647 ($1.11\text{x}10^{-16}$)	0.577 (0.0001)	0.639 (0.0001)	0.585 ($1.11\text{x}10^{-16}$)
Random Forest	0.657 (0.0016)	0.646 (0.0023)	0.717 (0.0033)	0.58 (0.0036)	0.685 (0.0018)	0.612 (0.0022)
Extra Tree	0.634 (0.0042)	0.650 (0.0022)	0.753 (0.0076)	0.513 (0.0129)	0.688 (0.0015)	0.573 (0.0077)
AdaBoost	0.66 (0.0003)	0.642 (0.0001)	0.705 ($1.11\text{x}10^{-16}$)	0.593 ($1.11\text{x}10^{-16}$)	0.682 (0.0001)	0.617 ($1.11\text{x}10^{-16}$)
ToPs (linear classifier)	0.658 (0.0076)	0.641 (0.0176)	0.704 (0.0239)	0.589 (0.0238)	0.68 (0.0096)	0.613 (0.0102)
ToPs (3 classifiers)	0.658 (0.0021)	0.644 (0.0018)	0.711 (0.0043)	0586 (0.0058)	0.687 (0.0015)	0.614 (0.0029)

6 Discussion of Results

6.1 Logarithmic Loss

Since the ToPs algorithm is mainly based on minimizing logarithmic loss, this is where it was expected to do the best. Comparing it with the other models, it was seen that ToPs with the three classifiers had the lowest log loss of 0.623, which is the same as

Random Forest. All other models had a higher log loss. It is observed that ToPs with just a linear classifier had a higher log loss compared to ToPs with the three classifiers, showing that the performance of ToPs is highly dependent on its base learners.

6.2 Area under ROC Curve

Receive Operating Characteristic (called ROC Curve) is created by plotting true positive rate against false positive rate using different threshold values [8]. ROC curve can help in selecting optimal classification models, as it shows the trade-offs between true positive (benefit) and false positive (cost). Thus, best prediction model would have (0,1) coordinate of the ROC space, with 1 as the area under ROC curve, representing no false negatives and no false positives. This is a good metric to compare as it is independent of the threshold used to convert a prediciton probability to the actual predicted class.

As shown in figure 2, ToPs with the three classifiers, AdaBoost, and Random Forest have higher ROC curve (with area under the curve as 0.71), followed by ToPs with linear classifier (area under the curve as 0.70), then Extra Tree (with area under the curve as 0.69). This shows that even though ToPs did not improve significantly over the other methods, it performed as well as those. Once again, it was observed that the performance of ToPs is dependent on its base learners as the ToPs with linear classifier as its base learner had a lower area under the ROC curve.

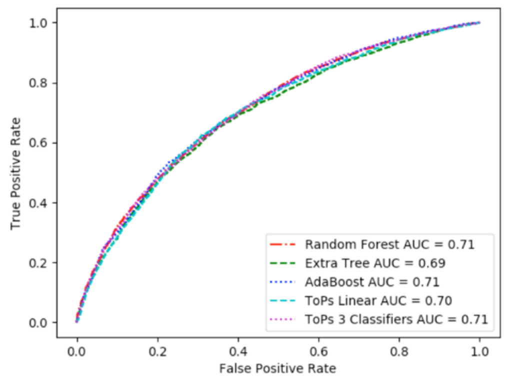

Figure 2: ROC Curve results for all ensemble methods.

6.3 Accuracy

Accuracy measures the total number of correct prediction with respect to total number of samples, regardless of the classes. As shown in table 2, ToPs with 3 classifiers, Random Forest, and AdaBoost had the highest accuracy of 65.3%. This is again expected as ToPs is based on minimizing loss, hence improving accuracy. Comparing it with the base method of K Nearest Neighbors, it can be seen that ToPs has a much better accuracy as K Nearest Neighbors had an accuracy of 61.5%.

6.4 Training Time

Training times was computed for each of the classification methods. This is where ToPs does the worst as ToPs with the three classifiers had a training time of 10.4 minutes, compared to the other ensemble methods which were all under 2 seconds. This is over 300 times more time consuming in comparison. Even K Nearest Neighbors that was used as a baseline had a training time of 4 seconds. This long training time shows that ToPs is extremely computationally intensive. It should also be noted that initially, without specifying the maximum depth for ToPs and using all 58 features without feature reduction resulted in a training time of over 12 hours, which was then not completed.

6.5 Precision, Recall, and F1-Score

Precision measures the classifier's ability to not label negative samples as positive, and recall measures the classifier's ability to find all positive samples [9]. F1-Score is defined as weighted harmonic mean of precision and recall [9]. Looking at the values in table 3, it can be seen that ToPs with the three classifiers had overall good precision, recall and f1-score values. The average precision, recall and f1-support was actually the same for ToPs with the three classifiers, Random Forest, and AdaBoost as 0.66, followed closely by ToPs with linear classifier and Extra Trees as 0.65

7 Conclusion

Trees of Predictors (ToPs) is a new ensemble learning algorithm that was implemented from scratch, then compared with other well-known tree ensemble methods such as Random Forest, Extra Trees, and AdaBoost Classifier. Two instances of ToPs were created for this comparison, one with just a linear classifier as its base learner, and another with three classifiers as its base learners, which were again, Random Forest, Extra Trees, and AdaBoost. Comparing these ensemble methods showed that ToPs (along with Random Forest) had the lowest logarithmic loss and highest area under ROC curve and accuracy. However, ToPs did very poorly in terms of training time as it is very computationally intensive.

It was also observed that ToPs algorithm does not have a good stopping criterion as it continues splitting until log loss cannot be further minimized. This leads to overfitting and even much longer traininig time. This was overcome by introducing a parameter of maximum depth to the ToPs algorithm. It was also observed that the evaluation metrics of ToPS were highly dependent on the classifiers its used as its base learners.

Based on the results seen, it is concluded ToPs is a good classification method, but needs significant improvement in making it less computationally intensive to improve training time. Another improvement would be to add accuracy as its metric when choosing the splits at each node, as it did not do so great in terms of accuracy either.

References

1. Jinsung Yoon, William R. Zame, Mihaela van der Schaar, *Tops: Ensemble Learning with Trees of Predictors*. 2018. [Online] Available: https://arxiv.org/abs/1706.01396v2
2. Breiman, L., *Random Forests*. Machine Learning (2001) 45: 5. DOI: https://doi.org/10.1023/A:1010933404324
3. Goel, E., Abhilasha, Er.: *Random Forest: A Review.* International Journal of Advanced Research in Computer Science and Software Engineering vo.7-1, (2017).
4. Geurts, P., Ernst, D., Wehenkel. L.: *Extremely Randomized Trees*. Machine Learning, 63(1), 3-42, (2006), https://doi.org/10.1007/s10994-006-6226-1
5. Freund, Y., Schapire, R.: *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. Journal of Computer and System Sciences. 55, 119-139 (1997). DOI: https://doi.org/10.1006/jcss.1997.1504
6. Boosting and AdaBoost for Machine Learning. [Online] https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning.
7. Lichman, M. *Online News Popularity Dataset*. UCI Machine Learning Repository. 2015. [Online] Available: https://archive.ics.uci.edu/ml/datasets/online+news+popularity
8. The Area Under an ROC Curve. [Online] http://gim.unmc.edu/dxtests/roc3.htm
9. Scikit-Learn, *Classification Reports*. [Online] http://www.scikit-yb.org/en/latest/api/classifier/classification_report.html