Open Set Forests

Ben Steele
UCCS Machine Learning REU
Benjamin.Steele@ColoradoCollege.edu

***Abstract*—Current ensemble classifiers use closed set decisions for each individual classifier in the ensemble. This applies to random forests, where each decision tree is a closed set classifier. In this article we propose using statistical extreme value theory to determine the relevance of the input to each decision tree in a random forest. this allows us to make open set decisions for the forest as a whole by looking at the average similarity between each tree and the given datapoint. We found that it is difficult to preserve the closed set accuracy when working with open set data in a random forest. However, it is possible to preserve the open set accuracy as unknown classes are added to testing.**

I. Introduction

Classification problems are currently set up in a closed set world. In closed set, any input to a model is assumed to belong to one the the classes it was trained on. This works well when this assumption holds true, which is often the case for specific classification problems. However, when we do classification of real world problems, we often cannot account for all of the classes we may see. Open set is a form of modeling in which the model is able to determine whether an input is similar to the data it was trained on.

We call this similarity to the training data pertinence. If the pertinence for a point is too low, the model cannot accurately predict the point because the point is likely not in any of the classes it was trained on. A classic example of an open set problem is object recognition in images. There is no way to account for every image the model might see during training, so it is beneficial for a model to be able to determine if an input is similar enough to the training data to be considered a known object.

Random forests are among the highest performing machine learning techniques. Because they are comprised of decision trees, they inherently have very low bias. By averaging the output of each tree they reduce variance which results in a relatively low bias, low variance model. The original implementation of random forests took the majority vote from all the trees to decide which class to output [3]. This means each tree is given equal weight in the forest's decision.

Probability estimation trees (PETs) are able to estimate the probability that a given data point is in a class. PETs give an estimation of the probability that the class they have chosen is correct, which allows the tree to weigh its decision depending on whether it has a high or low probability. However, these PETs are very poor estimators in general. Bagging, the basis of random forests, substantially improves PETs classification accuracy [5]. While bagging is accepted to improve classification accuracy in PETs, the best algorithm to do so is still an open question [2]. We believe the same concepts used in PETs can be used to create an open set tree.

Instead of predicting the probability that a point is in a class, we will predict the probability that a point is in any of the classes. Most implementations of PETs only compute probability at the leaf node, but this will not be sufficient for our algorithm. We will be using decision trees in a random forest, so the branching decisions will be limited to linear boundaries parallel to the axes. If we only check at the leaf we will only be checking pertinence along one axis, meaning we only get the pertinence of one feature. Because of this we will calculate pertinence at each node in the tree. We can then combine the pertinence of each node a point goes through to determine the overall pertinence.

Extreme value theory (EVT) [6] has been used for open set implementations before, so we will be using the libMR library to create EVT models for each tree. Decision trees are well suited for EVT because for data to be fit to an EVT model, it must first be represented in 1 dimension. In the case of a decision tree, each decision boundary is parallel to an axis, so only one feature is used in the decision. Because only one feature is used, each decision down the tree reduces the problem to a single dimension. This means the tree is already constructed in a way that EVT can easily be implemented at each node.

II. Method

We will be modifying the random forest from the Sci Kit learn library [4] to produce an open set forest. A decision tree in a random forest produces disjoint classes with linear boundaries parallel to the axes. At each node of a decision tree a linear boundary called the decision boundary is created, separating the data points of that node. Our modification adds EVT models to each of these decision boundaries in order to compute pertinence.

A. Node Pertinence

To create the model we must first reduce the points at a node to 1 dimension. This is done by taking the distance from the points to the decision boundary. Because the decision boundaries are parallel to the axis, the boundary is really just a threshold for a single feature. To calculate distance we only need to subtract the threshold from the point's value for that feature. We consider points greater than the decision boundary to be positive and points less than the decision boundary to be negative. Once all of the points are represented in a single dimension, we can fit an EVT model to them. We fit two models at each node, one to fit the high end (greatest points) and one to fit the low end (smallest points). The EVT models are monotonic, so a single model can only describe one side of the dataset. These two models allow us to compute the pertinence of any point that reaches the node.

To compute the pertinence of a point for a single node we first find its distance from the decision boundary. We can then plug that distance into each EVT model. We use the minimum of the two models as the final pertinence. The minimum is used because the model on the opposite side of the data from where the point is will give the point a very high pertinence (because the model is monotonic). The minimum of the two models is the pertinence that we are interested in because it will belong to the model that is on the same side of the data as the point.

B. Tree Pertinence

To find the pertinence of a point using the entire tree we combine the pertinence of each node that the point passes through in a few ways. Choosing a method of aggregating the pertinences from each node a point passes through was nontrivial and we chose 3 to determine experimentally which is best. One method was simply taking the minimum of all the pertinences. This would accentuate a single feature that is significantly not pertinent but would not be good at distinguishing points that are only slightly not pertinent for many features.

Our second method takes the average of all the nodes. This is better when many features are slightly not pertinent, but greatly softens the impact a single variable can have on the overall pertinence.

The third method is taking the product of each node's pertinence. This should allow a single extreme feature to have a larger impact than using the average, while maintaining the ability to find many features that are slightly not pertinent. this is also a logical way to combine the nodes because when you find the probability of multiple independent things happening together, you multiply their individual probabilities. This is essentially what we are doing.

Not every leaf in a decision tree is at the same level in the tree. This complicates the product because points with more nodes will generally have a smaller product. To correct for this we use a log function.

N = number of nodes

P = product of pertinences of each node

$$\log_{10^N} P$$

C. Forest Pertinence

The forest averages the pertinences from each of its trees to determine the overall pertinence. Once we calculated the overall pertinence, we needed a way to decide what is pertinent and what not. For this we trained a threshold. Using linear regression, we trained the threshold to maximize accuracy on a validation set. Once the threshold is trained the model is completely trained and can be tested.

III. Evaluation

Our primary focus in this study is to create a foundation for open set in random forests. To evaluate the model we use the accuracy score of the original forest against the open set forest using min pertinence, average pertinence, and product pertinence. We found during testing that the more continuous features the data has, the better our algorithm will perform. Because many datasets have large numbers of non continuous data we had to choose specific datasets and generate our own. We used the mnist dataset from the UCI Repository [1] as well as generated datasets. The mnist dataset has 70,000 data points, 780 features, and 10 non separable classes.

The generated datasets were non separable and used either normal distributions or gamma distributions for each class. They contained 1,000 points per class, 50 features, and 100 classes.

classes	original	min	average	product
0	0.96	0.71	0.78	0.71
1	0.76	0.69	0.72	0.68
2	0.64	0.69	0.68	0.67
3	0.55	0.67	0.64	0.65

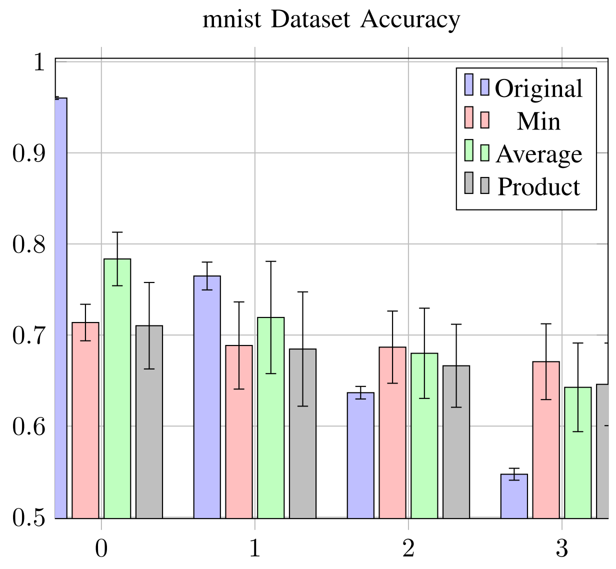

Fig. 1. A graph of the accuracy of all 4 algorithms with different numbers of unknown classes. The x-axis is the number of unknown classes being tested. At 0 on the x-axis it is using the 4 classes the model was trained on, but no unknown classes. The y-axis is the % accuracy of the model.

IV. Results

In testing we found that our algorithm is extremely variable depending on the dataset that is used. When using the mnist dataset we constructed a forest with 40 trees and used 4 classes to train and 3 classes to validate. we tested using the remaining 3 classes in 5 trials. When the model was tested using only the classes used during training the original forest was able to get 96% accuracy while the open set forest got at best 78% with average pertinence, as seen in figure 1.

As unknown classes were added, the accuracy of the original forest dropped much faster than the open set implementations and once two classes had been added all of the open set implementations outperformed the original forest

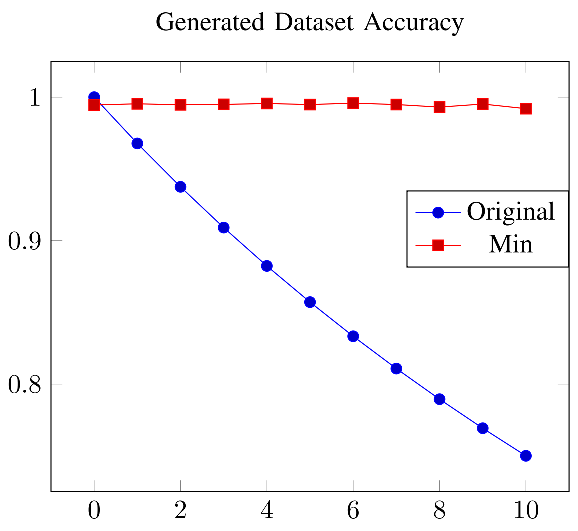

Fig. 2. A graph of the accuracy of the original forest and the min pertinence random forest. The x-axis is the number of unknown classes being tested. At 0 on the x-axis it is using the 30 classes the model was trained on, but no unknown classes.

in accuracy. when 2 and 3 unknown classes are added to the 4 known classes min pertinence has the best accuracy while the original forest does the worst. The min pertinence implementation did drop in accuracy as unknown classes were added, but no more than 2% for each unknown class.

using the generated data we only tested the original forest against the minimum pertinence open set implementation. We used forests with 20 trees, training on 30 classes and validating with 8 classes. When testing on only the training classes, the original forest had 100% accuracy while the open set forest had 99.4% accuracy as seen in figure 2. This is a far more acceptable loss of accuracy compared to the mnist dataset. As unknown data is added, we see a similar trend as in the mnist dataset. The original forest drops in accuracy far faster than the open set forest. after adding 29 unknown classes, the original dataset performed at 43.3% accuracy while the open set forest was still at 98.1% accuracy.

While the accuracy of the open set forest is far greater in the generated data than in the mnist data, the behavior when unknown data is added is similar. In both cases the accuracy falls slowly, showing that accuracy can be preserved as unknown data is added, even if that accuracy is far less than in a closed set implementation.

V. Conclusions

Open set forests succeed in maintaining accuracy in the presence of unknown classes, but cause a loss in their base accuracy (accuracy on closed set test) to do so. It is promising that our implementations are able to maintain a relatively stable accuracy. In both datasets, the accuracy does not significantly decrease as more unknown classes are added. In the generated dataset we show that even with very high base accuracy, adding unknown classes will not cause it to drop.

The 3 implementations we used for open set forests all performed similarly compared to the original forest. Of the three, min pertinence seems most stable because it has the smallest decrease in accuracy as unknown data is added. However, with small amounts of unknown data the average pertinence performs better. Depending on the nature of the data, one may be better than the other. Product pertinence consistently performs poorly in these tests, making it the worst of the 3 implementations.

The generated dataset's results are far different from the mnist dataset. We believe the generated data is probably very close to separable because the high number of dimensions creates a large space even when the range in each dimension is small. This does not make the results insignificant. the generated dataset shows that our algorithm can work very effectively with medium size dimensionalty and many classes when the classes are near separable.

The mnist dataset had much poorer results than the generated data. The base accuracy dropped over 20% on known classes. even though the accuracy was better when a few unknown classes were added, this is a tradeoff most users would opt not to take. We believe this drop in accuracy is caused because the trees are unable to produce good estimations of pertinence.

The decision trees only see one feature at a time, one feature at each node. This is because only looking at a single feature for each node makes a forest much faster than if multiple features could be considered at the same time. This means the algorithm is most fit at finding outliers that have one or more features that are grossly different than those features in the training data. This may be why min pertinence seems to be the best metric, it captures the single feature that separates a datapoint from the rest of the data.

This is more of a problem in higher dimensional spaces because a point can easily have similar feature values to the training points when looking at the features individually, but that points euclidean distance from the training points can be very large at the same time. A simple example of this in 2d is if you have a circle inside a square, both with the same width. take a point on the corner of the square. if you look at that point from either axis it looks like it is right on the edge of the circle, but in 2d it is far from the edge of the circle. In much higher dimensions this error space becomes very large and this hinders the ability of these trees to predict pertinence. This phenomena is somewhat mitigated because as the tree gets deeper, the dataset is cut into smaller and smaller sections and the error space becomes increasingly small. However, this is not the only issue in higher dimensions.

With higher dimensions, the tree is going to make decisions on only a small portion of the features before it can determine a class for any point. This means some features that obviously make a point not pertinent may not be the feature that the model is using to determine that point's class. These issues can be fixed by allowing the use of any hyperplane at each node, not only planes parallel to the axes.

A significant limitation to open set forests is that they perform extremely poorly without a majority of continuous features. This occurs because only one feature can be evaluated at a time. A binary feature cannot give much information on pertinence on its own, and similarly any feature with a low number of possible values becomes a very poor indicator of pertinence. This also means any non numeric feature is insignificant in determining pertinence. This would not be as big an issue if the decision boundaries were not parallel to the axis, so future work could be done on decision trees that can use any hyperplane as a decision boundary. This would be far more computationally intensive, but is likely to be much more accurate and robust.

VI. Acknowledgments

We would like to thank the University of Colorado at Colorado Springs and the National Science Foundation's Research Experiences for Undergraduates program, who's Grant 1359275 funded this research.

References

[1] CL Blake and C Merz. Uci repository of machine learning databases. *University of California, Department of Information and Computer Science*, 1998.

[2] Henrik Bostrom. Estimating class probabilities in random forests. In *Sixth International Conference on Machine Learning and Applications*, pages 211–216. IEEE, 2007.

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

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[6] Walter J Scheirer, Anderson Rocha, Ross J Micheals, and Terrance E Boult. Meta-recognition: The theory and practice of recognition score analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1689–1695, 2011.