CS 529 Project 1: Random Forests

Christopher Tye
& Gabriel Urbaitis

Random Forests are an ensemble method used in Machine Learning for classification using decision trees. In our project, we wrote algorithms in Python to construct decision trees based on information gain from one of three impurity measures, Entropy, Misclassification Error, and Gini Index. The Goal was to use these trees to classify “Vesta's real-world e-commerce transactions [containing] features describing aspects of the transactions like: product features, card features, and other coefficients that are kept [masked] for privacy reasons.” [1] The two possible classifications were 0 (not fraud) and 1 (fraud).

The algorithm for building the trees started at the root node, and took in the 25 feature column data set. The feature with the highest information gain (described below) was selected, and the data set was split according to rows that had that feature’s value if it was a categorical feature, or according to rows that had less than, or greater or equal to a selected midpoint if the data was continuous. Selecting a categorical feature would create a child node for each possible value of the feature, and that feature’s column was dropped as the split given for that node to evaluate would have a repeated value for the selected feature. Selecting a continuous feature resulted in two children, less than, and greater or equal to the midpoint, but in these cases the column was preserved, as another midpoint in the column might lead to the highest information gain at the next stage.

The equations for information gain and the three impurity measures are described below.

Information Gain(D, A) = Impurity(D) - $\sum_{v \in A}((N_v/N)*\text{Impurity}(D_v))$
Where D is the original data, A is the Attribute to be tested, v is a value in the attribute, N_v is the number of occurrences of value v in attribute A, N is the total occurrences for all values v in attribute A, and Impurity is one of the three equations:
Entropy: $-\sum_c(P(c)*\log(P(c)))$
Misclassification Error: 1 - max(c1,c2, … cn)
Gini index: $1 -\sum_c((P(c))^2)$
Where c is a specific value for the feature in question. (c is equal to v from the Information Gain equation.)

Additionally Entropy’s Information Gain was divided by the split information test:
$-\sum^n_{i=1}((N_v/N)*\log(N_v/N))$

This divides the information gain by the subset’s intrinsic information to remove bias for features with many values.

The multitude of values in continuous features led to slow performance. While some approaches to continuous data test every possible value as a midpoint, this quickly becomes computationally expensive, especially when considering it has to be done every time information gain is calculated for that feature. We optimized by selecting a midpoint halfway between every appearance of a 1 for “isFraud” in the selected continuous feature. This was still very slow, so

we ended up taking a random sample of 20 midpoints in continuous features and taking the one with the highest information gain for comparison to the other features when building the tree.

We tried a variety of sample sizes of our midpoints, and we actually found that smaller sample sizes down to at least 20 actually improved accuracy in our random forests. This was counter intuitive, as according to theory, larger sample sizes have greater resolution of the set and therefore are more likely to find the perfect split. However it seems that finding the general trend of a good split is more important than finding the best one, as the best one may overfit or "memorize" a data point that may not be all that much more significant than the next best ones.

Missing Data in the training and test sets was labeled as "notFound." To our observation, this only occurred with categorical features. We compared removing rows with "NotFound" and treating "notFound" as a category. We found higher accuracy when treating "notFound" as a category. Our assumption is the absence of some information may be a good indicator of fraud cases for some features. If "notFound" or "NaN" ever appear in continuous features, their values are removed when calculating midpoints, however, we observed no such instances in the data.

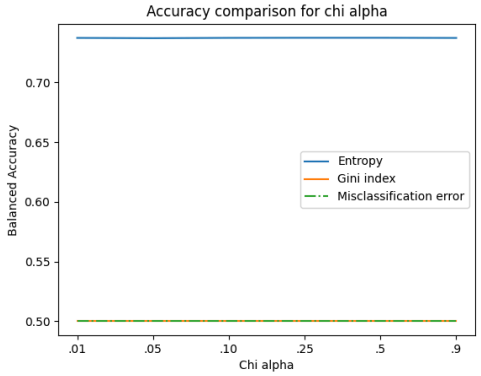
The data had an approximate 95% "isFraud" = 0 to 5% "isFraud" = 1 imbalance. When data has such a disparity in the amount of the minority class to the majority class, not addressing the imbalance can lead to models which are biased towards predicting the majority class. When detecting fraud, a false positive is much less costly than a false negative as a false negative results in theft, and a false positive merely results in possibly slightly annoying the client with a confirmation request. As fraud cases are the minority class, it was important to create a more balanced data set. We did this through undersampling the majority class. We used imblearn's RandomUndersampler method to create a 50-50 split between majority and minority classes. This removed any inherent bias in one class over the other by having equal amounts of each one. We then used the undersampled data to build our tree and tested it on the regularly sampled data. Additionally, when calculating accuracy, we use scikit learn's confusion matrix to get the true negative, false positive, false negative and true positive values. From these values we were able to get the recall (recall = (tp) / (tp + fn)) and true negative rate (tnr = (tn) / (tn + fp)) which we then used to get the balanced accuracy (balacc = (recall + tnr) / 2). By using balanced accuracy to evaluate our trees, we ensured that both our rates of true positives and true negatives were weighted equally, despite the necessary imbalance in testing data which had to approximate the likeliest unknown distribution. So in summary, class imbalance was handled through undersampling the data when building the model, and through using balanced accuracy when evaluating the model.

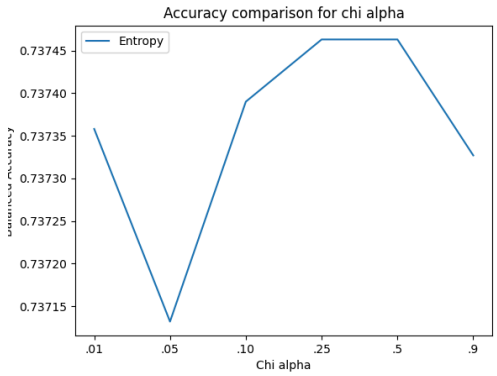
In addition to the random sampling of 20 midpoints, we implemented an ignore_feature parameter, that randomly skips a fifth, quarter, or log or square root proportions of the features at each level to increase the diversity of trees in the random forest and prevent single features from dominating the node structure of the tree. We found that skipping a fifth produced optimal accuracy in random forests. The skips also improved performance as the features skipped didn't have to be evaluated for information gain. Our final performance optimization (other than chi squared which is discussed below) was implementing a max depth, to prevent trees from becoming too deep. We found that accuracy increased when we increased the max depth, but the gains dropped off at 20. Average depth increases also dropped off at a max depth of 20.

In our tests, we compared trees built on different information gains and for different chi termination values. We standardized a random seed value for the train_test_split, undersampling and midpoint sample methods so the data interpreted by each information gain method would be identical. All trees were trained on the same 80% of the data and tested on the remaining 20%. We used the 20 midpoint samples for each tree, and a maximum depth of 30 to help illustrate any structural differences. The ignore_feature logic was not invoked as for single trees this would create differences in variance not relevant to either comparison.

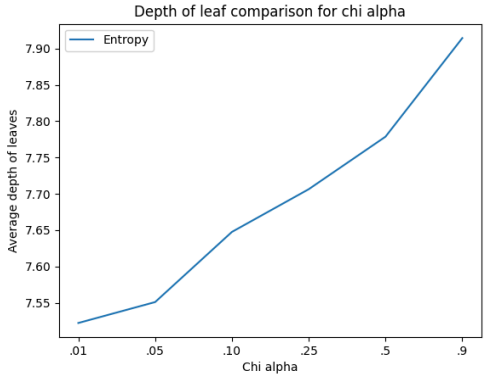
Chi squared results are described below:

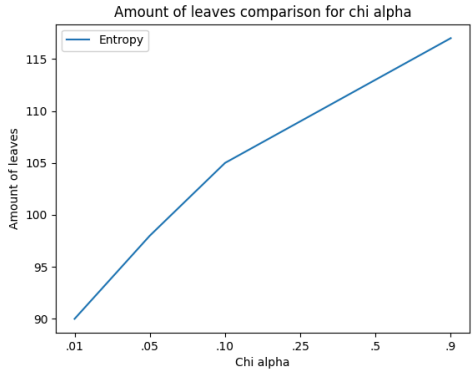
No critical value for gini and misclassification surpassed any of the 6 thresholds, so the trees produced had one node, and because the undersampling produced equal amounts of values they were all classified as negative per the tiebreaker criteria. Thus both gini and misclassification had a balanced accuracy of .5 for all 6 values. For entropy, the mean across the 6 values of chi was .737355 and all values fell in ±.000224 of the mean.



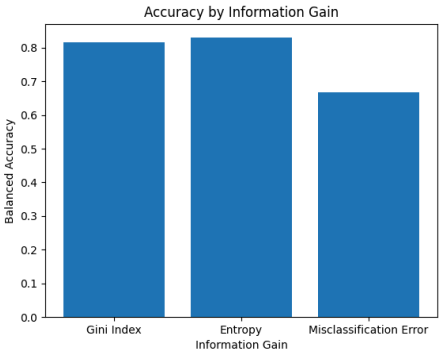


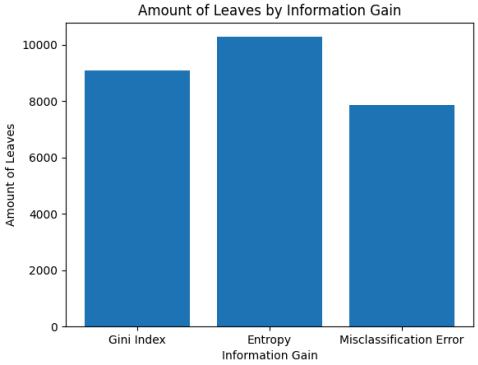
Structurally, all six trees each produced by gini and misclassification had one node, a leaf node, so the maximum and average depth were 0. Entropy had a maximum depth of 13 for all 6 alpha values, and average depth steadily increased from 7.52 at alpha = .01 to 7.92 at alpha = .90. This meant the greater the threshold, the more early termination of branches by chi squared, which was to be expected. This could also be seen in the amount of leaves that steadily increased from 90 at alpha = .01 to 117 at alpha = .90.



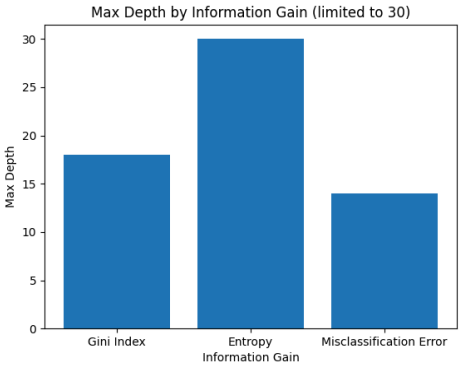


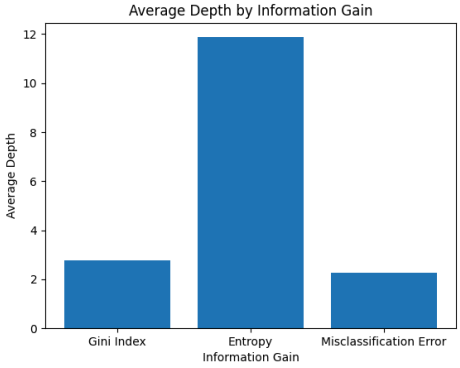
Since the chi squared termination rule provided the absolute minimum accuracy for misclassification and gini index with only one leaf node trees, and it provided no clear trend in the accuracy for entropy, we ran trees on the same data as before but with chi alpha = 1, meaning the termination rule was turned off. This allowed us to compare the trees made by each method with regard to structure and accuracy.





Entropy was the most accurate at 82.95%, followed by Gini index at 81.57% and Misclassification Error at 66.82%. In the same order, Entropy had the most leaves at 10280, followed by Gini Index with 9085, and Misclassification with 7861.





Average depth had the starkest contrast with Entropy nearing a depth of 12 per leaf and Gini Index and Misclassification between 2 and 3. Entropy reached the maximum depth of 30 while Gini Index and Misclassification Error terminated earlier at depths of 18 and 14 respectively.

Overall the increases from using a chi squared termination rule to disabling the feature led to accuracy increases of ~8% for entropy and 31.57% and 16.82% for Gini Index and Misclassification Error respectively. Thus our final optimization for accuracy was to use 1 as the alpha for chi squared, effectively not using the chi squared termination rule.

As Entropy had higher accuracy in our tests, we chose it for our final Random forest submitted in Kaggle. It makes sense for both Entropy and Gini Index to be much better than

Misclassification Error, as the maxing function ignores how balanced the data splits are beyond the maximum probability. It also makes sense for Entropy to be slightly better than Gini index as the logarithmic function entropy uses is more sensitive (between 0 and 1 where probabilities lie) to changes than the squaring function gini uses.

Our final random forest consisted of 11 trees built on random samples of 70% of the data, skipping a fifth of features randomly, randomly selecting 20 midpoint samples, with a maximum depth of 25 and a chi alpha value of 1 (equivalent to not using the chi squared test for termination). It scored 0.87373 on the 83% of the test data available at submission time.

References:

[1] Estrada, Trilce. *Kaggle*. 2024,
www.kaggle.com/competitions/cs429529-project1-random-forests/data
Accessed 10 March 2024.