Random Forests evaluated on crime data

Author: Gellért-Szabolcs Papp

1 Random Forest

This implementation of Random Forests was done for a university project.

1.1 Implementation details

This implementation of Random Forests was written in cython. The implementation has 4 hyperparameters: *the number of decision trees to be trained, the number of rows a tree should be trained on, the maximum height for a tree, and the method for choosing the best feature split, either Gini or information gain.* These can be specified during initialization.

The forest contains an array of decision trees that are trained during initialization. Each tree is trained on a different thread, so the training process will be faster if python's global interpreter lock is disabled.

Decision Trees are recursive data structures, meaning that one decision tree can have multiple child Decision Trees. During training, if a tree is not a leaf node (meaning the height limit is not reached and not all the data given to it has the same label) then it will search for the best feature to split the data it has been given based on the Gini index or information gain.

After initialization, a row can be given to the forest as input, and it will predict which class the row belongs to by evaluating the row with its trained trees and picking the majority vote.

The data on which it has been tested on contains only continuous features, but the implementation also supports categorical values (but has not been tested on them yet).

1.2 Experimental results

Experiments were conducted for 2 class classification on the dataset (but the implementation supports an arbitrary amount of classes), and the trees had a maximum depth of 20 nodes.

Hyperparameters have been tested using grid search. Metrics for these are depicted on figures: 1-11. From these results, we can see that some metrics benefit from either higher or lower tree counts and different amounts of data given to the trees. It can also be seen that forests with very few trees perform poorly; however they do not improve significantly with tree counts above 10 either. Even with a high tree count and large amounts of data given to each tree of the forest, test data accuracy only reaches 90% and training data accuracy 93%.

The overall performance was evaluated with 10 way cross-validation, against the sklearn implementation. The results for this can be found in table 1.2. It can be seen that our implementation is slightly slower to train than the sklearn implementation, and while on the test data they have comparable results, the training results are much better for the sklearn implementation. Our implementation outperforms other attempts for classifying this dataset in the related literature, which had accuracies between 64-84%, but it also has to be noted that those attempts were geared towards the fairness of AI.

Results				
	Our implementation with 10 trees	sklearn implementation with 10 trees	Our implementation with 100 trees	sklearn implementation with 100 trees
time to converge	1.02 s	0.61 s	7.94 s	5.97 s
test accuracy	89.3%	89.2%	89.6%	89.9%
test precision	66.9%	69.6%	68.3%	70.2%
test recall	44.9%	39.8%	45.1%	46.5%
test specificity	96.3%	97.1%	96.5%	96.7%
test f-score	55.8%	50.6%	54.4%	55.9%
train accuracy	89.8%	99.2%	90.8%	1.00%
train precision	71.4%	99.7%	74.7%	1.00%
train recall	45.8%	94.5%	51.2%	1.00%
train specificity	96.9%	99.9%	97.2%	1.00%
train f-score	55.8%	97.0$	60.7%	1.00%

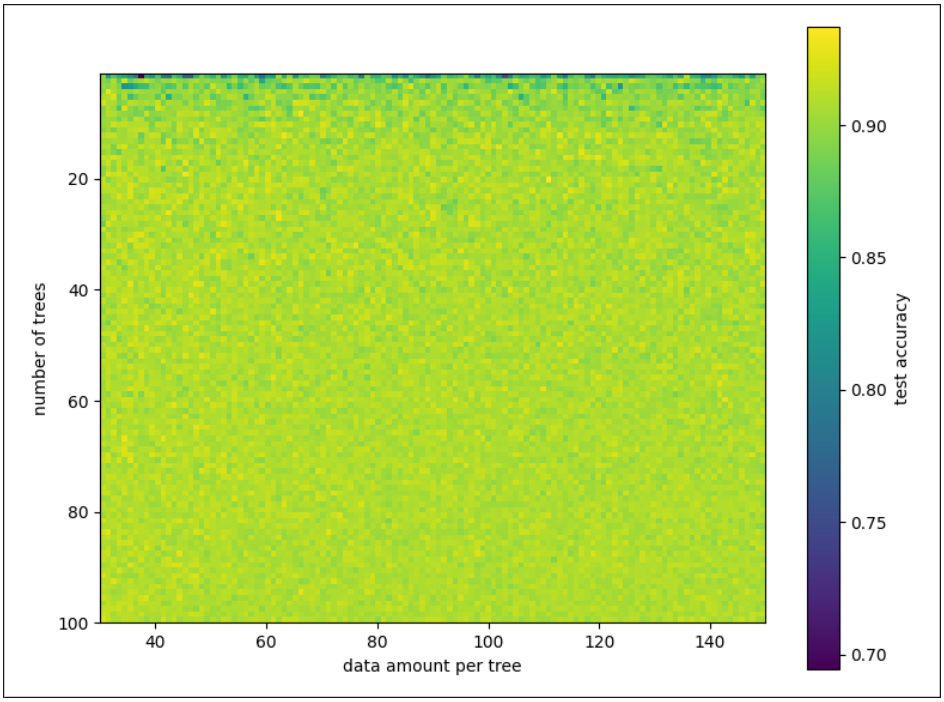

Figure 1: Test accuracy

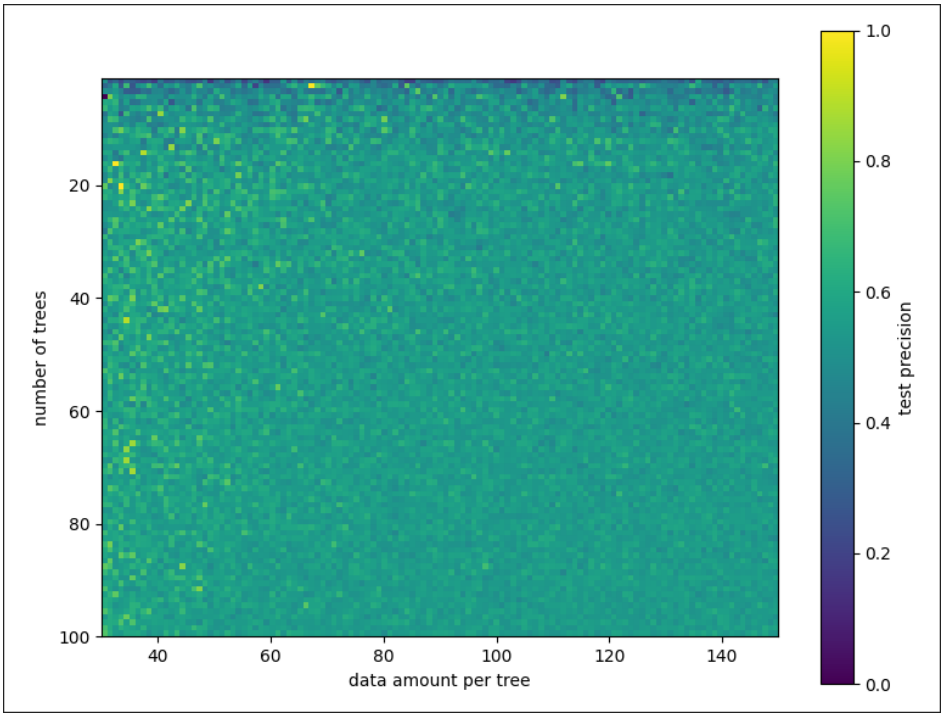

Figure 2: Test precision

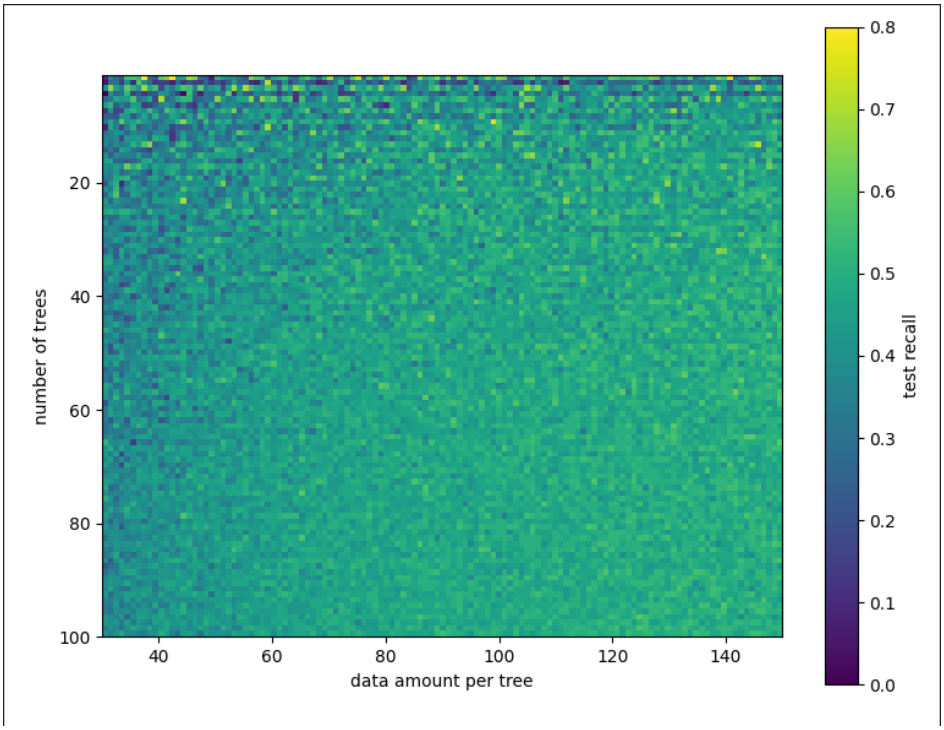

Figure 3: Test recall

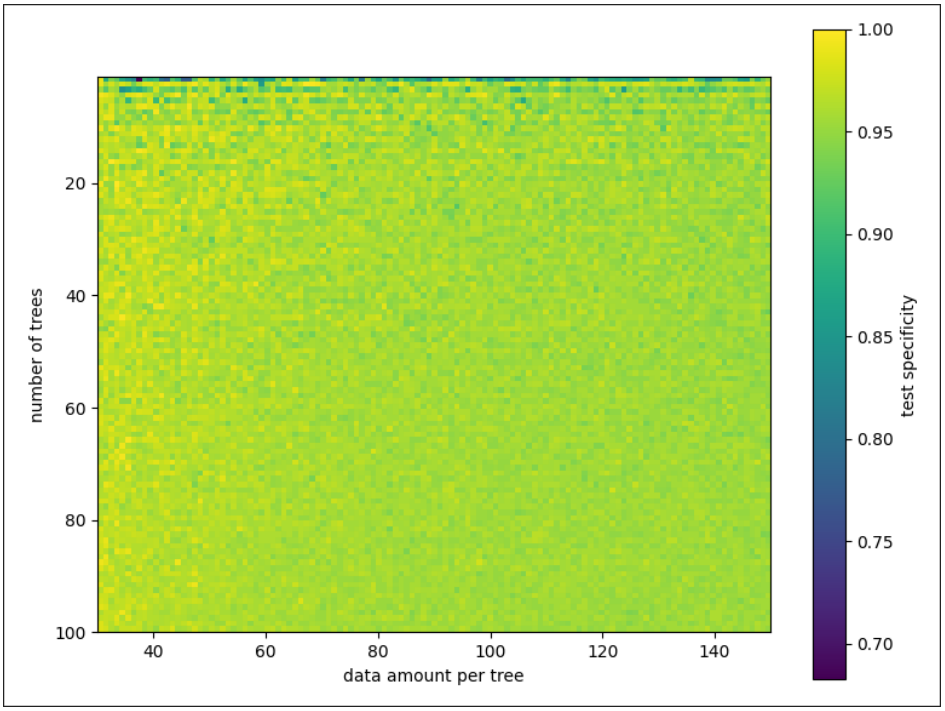

Figure 4: Test specificity

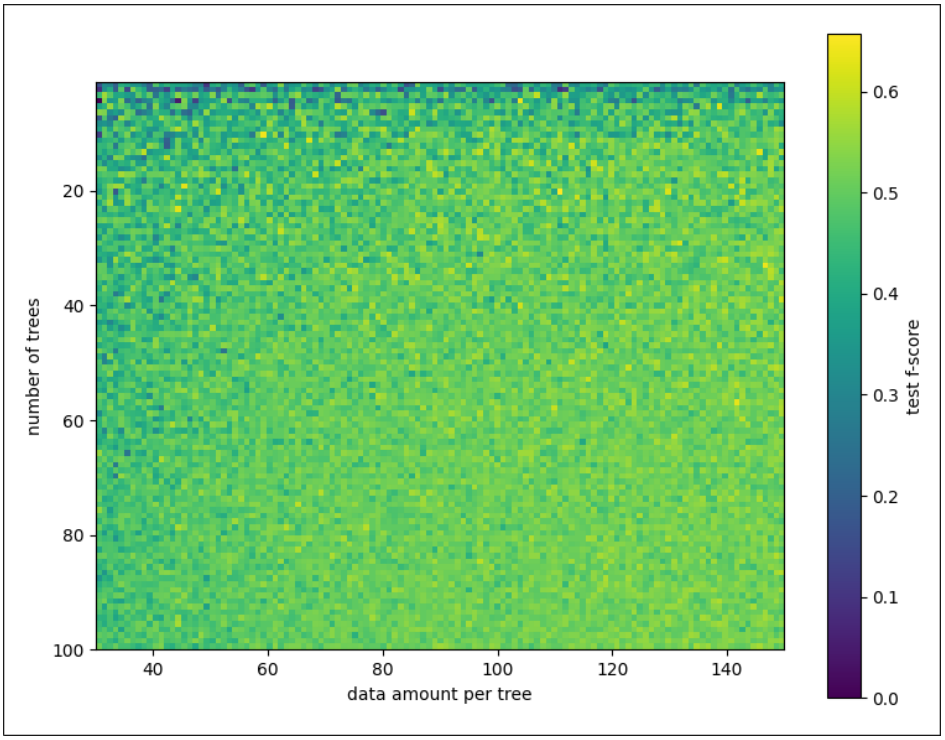

Figure 5: Test f-score

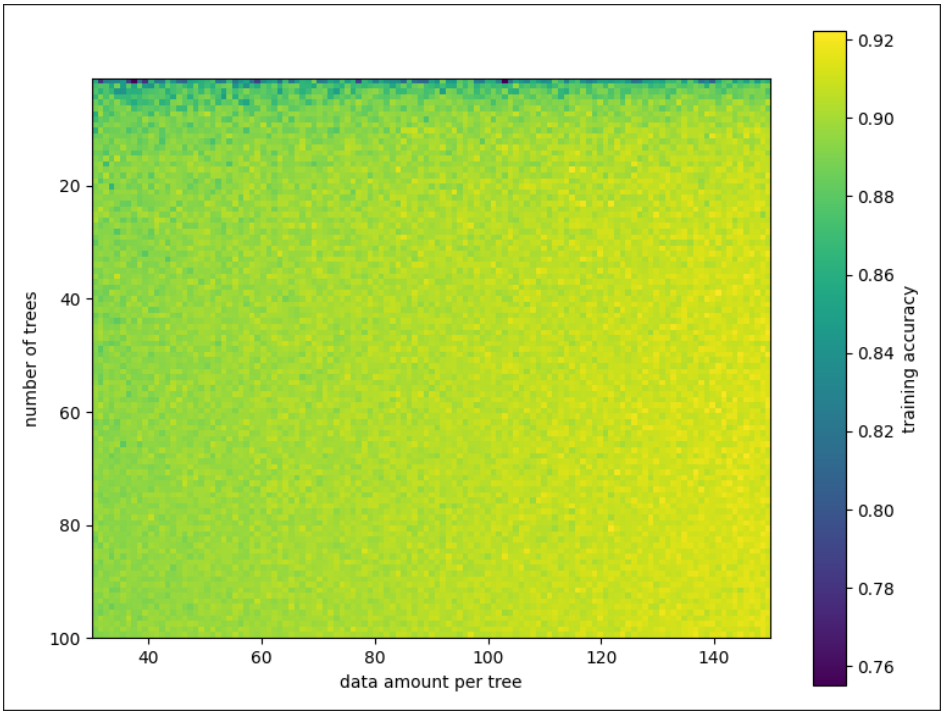

Figure 6: Training accuracy

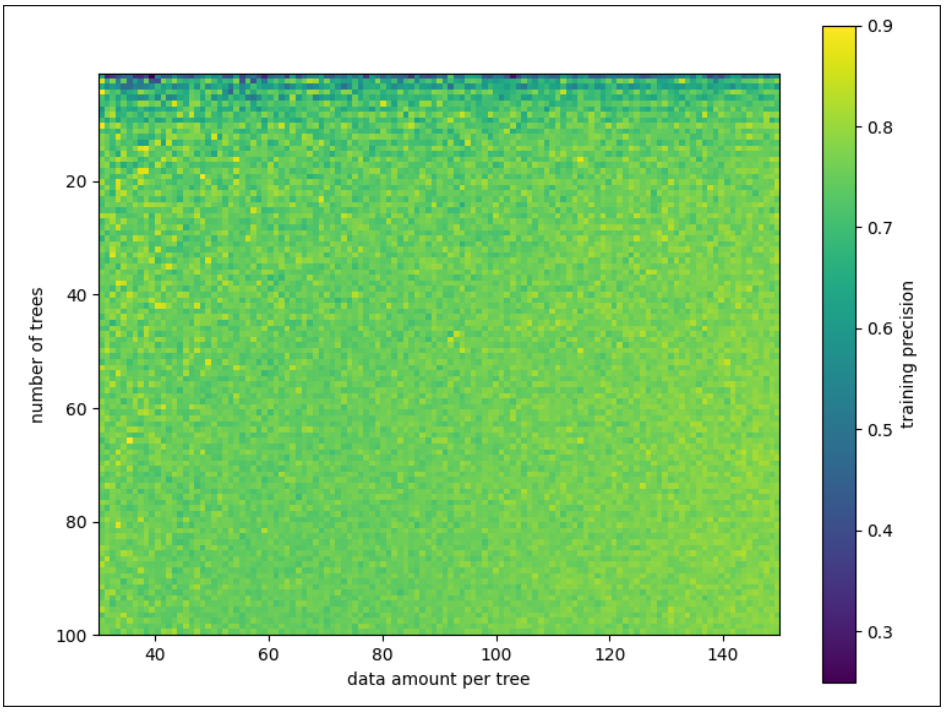

Figure 7: Training precision

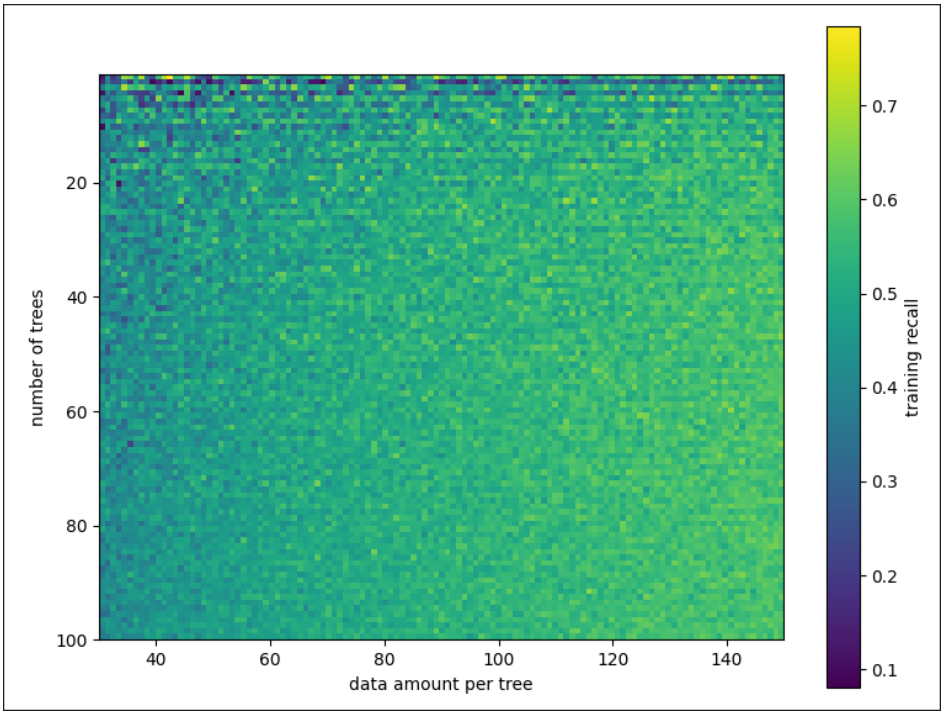

Figure 8: Training recall

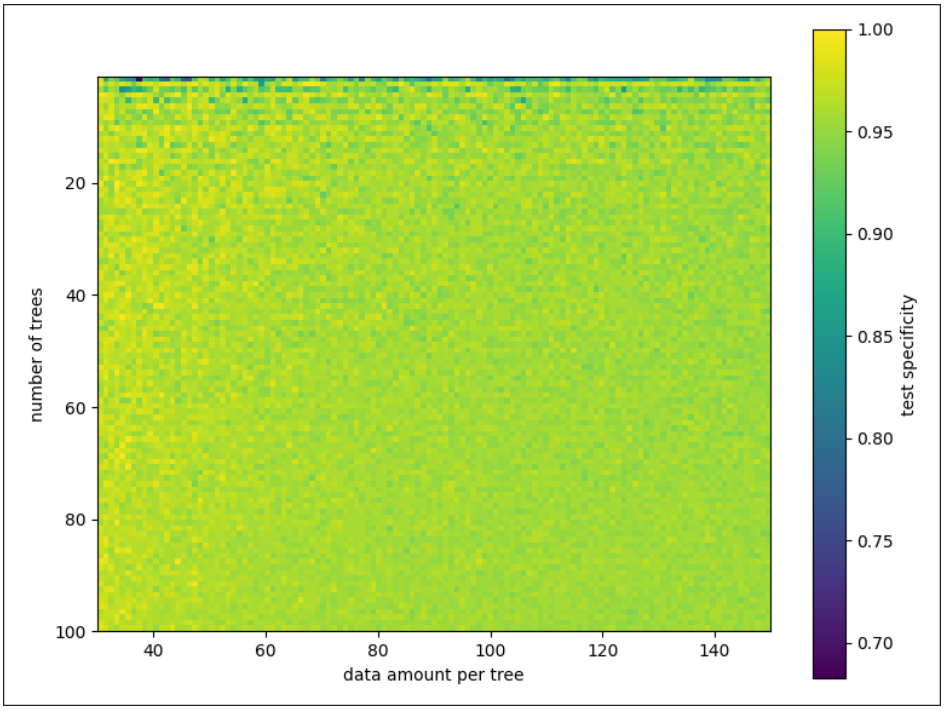

Figure 9: Training specificity

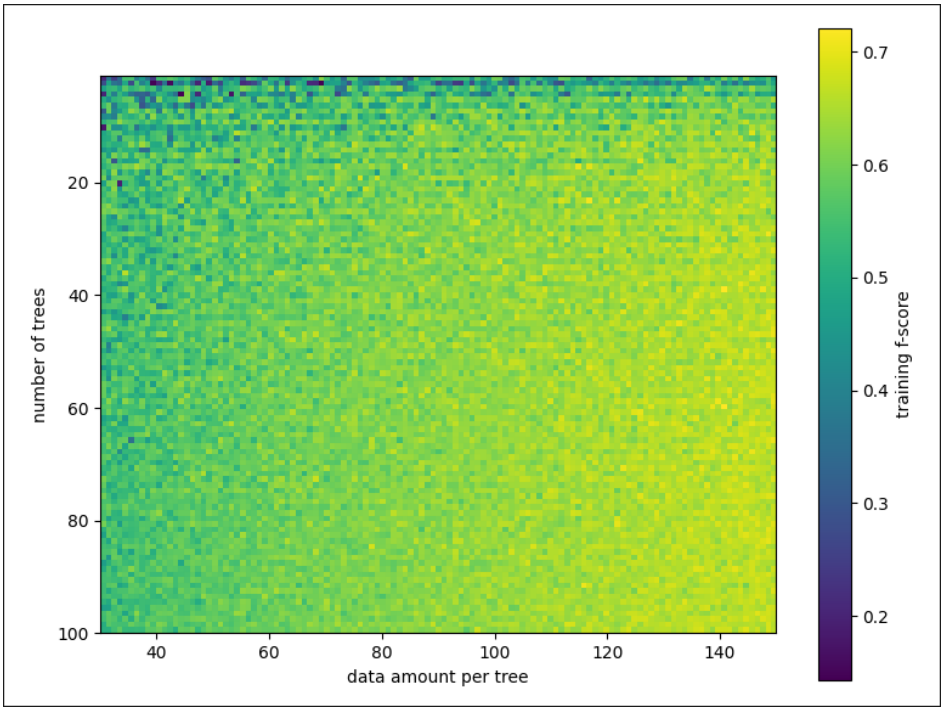

Figure 10: Training f-score

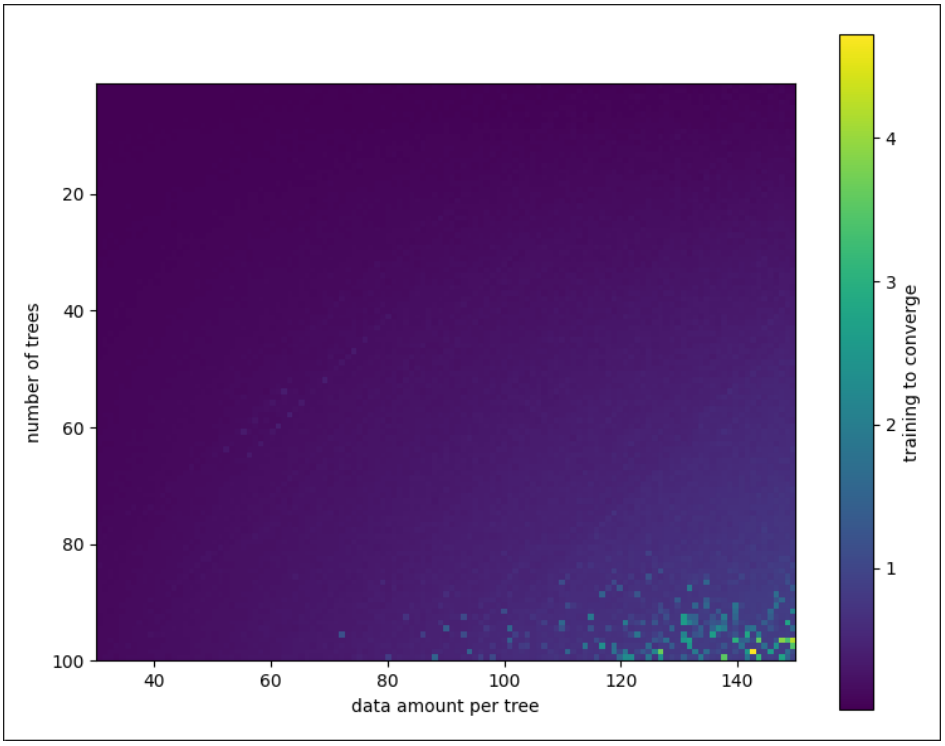

Figure 11: Time to converge

1.3 Discussion

As stated in the previous section, our implementation reaches better performance than the related literature, which might be because those approaches took fairness into account.

From the perspective of explainability, we've used the LIME algorithm to explain our results. With this we obtained the most important features which are depicted in figure 12.

One can also find single instance explanations such as the ones presented by figure 13 and 14

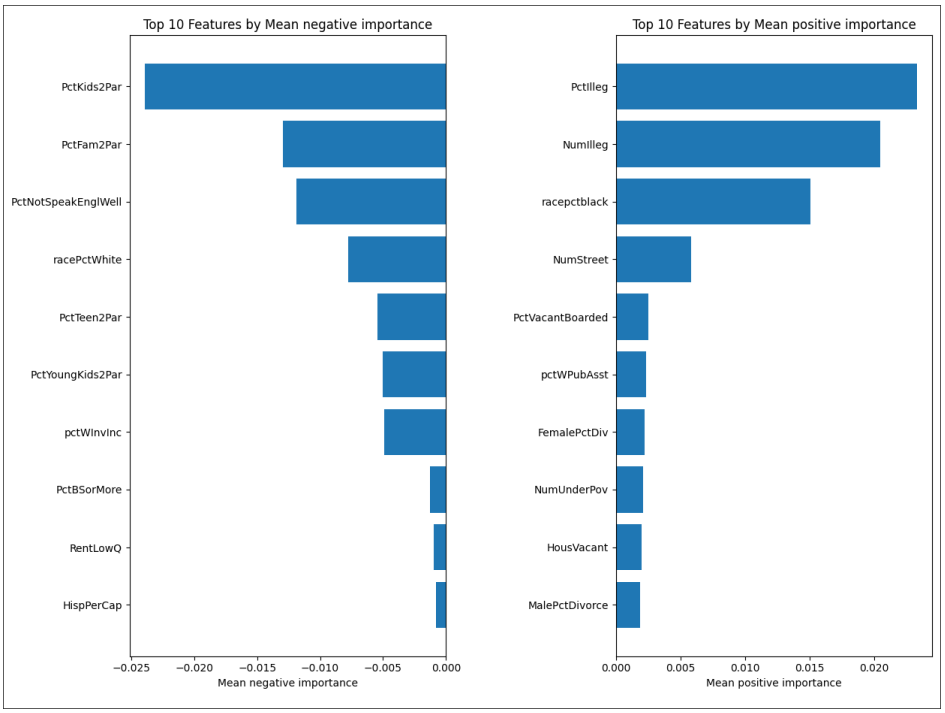

Figure 12: Most important features

Figure 13: Explanation for a single instance of low crime rate

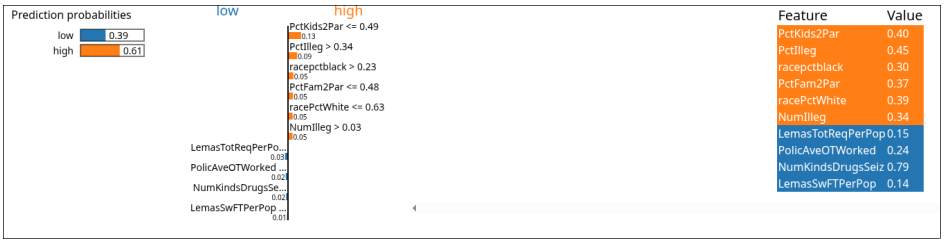

Figure 14: Explanation for a single instance of high crime rate