Model

Best parameters: {'max_depth': None, 'n_estimators': 50} Best cross-validation score: 0.838178871138416

Best estimator: RandomForestClassifier(n_estimators=50, random_state=42) CV Results: mean_fit_time std_fit_time mean_score_time std_score_time
0 388.361178 1.316630 18.459254 0.067612
1 775.966639 3.836029 36.261779 0.158859
2 1546.735567 1.099148 72.351171 0.201649
3 387.077123 2.431197 18.395992 0.065002
4 765.819646 3.443475 36.144048 0.129246
5 1534.708805 8.858862 71.736950 0.447102
6 385.781756 2.816198 18.546033 0.159117
7 767.741088 2.092667 36.290477 0.128457
8 1515.395600 30.513255 70.541677 1.401866
9 385.295443 5.159583 18.421564 0.080699
10 749.348484 4.340930 35.605123 0.092405
11 1506.804124 23.657984 72.895656 0.794143

param_max_depth param_n_estimators
0 None 50
1 None 100
2 None 200
3 5 50
4 5 100
5 5 200
6 10 50
7 10 100
8 10 200
9 20 50
10 20 100
11 20 200

params split0_test_score \

0 {'max_depth': None, 'n_estimators': 50} 0.838179
1 {'max_depth': None, 'n_estimators': 100} 0.838179
2 {'max_depth': None, 'n_estimators': 200} 0.838179
3 {'max_depth': 5, 'n_estimators': 50} 0.838179
4 {'max_depth': 5, 'n_estimators': 100} 0.838179
5 {'max_depth': 5, 'n_estimators': 200} 0.838179
6 {'max_depth': 10, 'n_estimators': 50} 0.838179
7 {'max_depth': 10, 'n_estimators': 100} 0.838179
8 {'max_depth': 10, 'n_estimators': 200} 0.838179
9 {'max_depth': 20, 'n_estimators': 50} 0.838179
10 {'max_depth': 20, 'n_estimators': 100} 0.838179
11 {'max_depth': 20, 'n_estimators': 200} 0.838179

split1_test_score split2_test_score split3_test_score \

0 0.838179 0.838179 0.838179
1 0.838179 0.838179 0.838179
2 0.838179 0.838179 0.838179
3 0.838179 0.838179 0.838179
4 0.838179 0.838179 0.838179
5 0.838179 0.838179 0.838179
6 0.838179 0.838179 0.838179
7 0.838179 0.838179 0.838179
8 0.838179 0.838179 0.838179
9 0.838179 0.838179 0.838179
10 0.838179 0.838179 0.838179
11 0.838179 0.838179 0.838179

split4_test_score mean_test_score std_test_score rank_test_score

0 0.838179 0.838179 5.326493e-08 1
1 0.838179 0.838179 5.326493e-08 1
2 0.838179 0.838179 5.326493e-08 1
3 0.838179 0.838179 5.326493e-08 1
4 0.838179 0.838179 5.326493e-08 1
5 0.838179 0.838179 5.326493e-08 1
6 0.838179 0.838179 5.326493e-08 1
7 0.838179 0.838179 5.326493e-08 1
8 0.838179 0.838179 5.326493e-08 1
9 0.838179 0.838179 5.326493e-08 1
10 0.838179 0.838179 5.326493e-08 1
11 0.838179 0.838179 5.326493e-08 1
Scorer function: <function _passthrough_scorer at 0x7fa19f83eb60> Refit time (seconds): 446.5034511089325

Best parameters: The grid search identified that using 'max_depth' of None and 'n_estimators' of 50 leads to the best performance. This means using a RandomForestClassifier with no limit on the depth of the trees and 50 trees in the forest provides the highest score.

Best score: The highest cross-validation score achieved by the best parameters is approximately 0.8382. This is a reasonably high score, indicating that your model performs well.

CV Results: This is a summary of the grid search results, including the mean fit time, standard deviation of fit time, mean score time, and standard deviation of score time for each combination of parameters.

Interestingly, all the tested configurations have exactly the same mean test score, which is quite unusual. It may indicate that your model is insensitive to the changes in 'max_depth' and 'n_estimators'. This could potentially suggest that the features you are using to predict deforestation don't require complex modeling, or it could mean your labels are heavily imbalanced, with one class dominating the others, which can lead to misleadingly high accuracy scores.

Scorer function: The scoring function used was a pass-through scorer. This means that the estimator's default scoring method was used, which in the case of RandomForestClassifier is mean accuracy.

Refit time: The time taken to refit the best estimator (found during grid search) to the entire dataset was approximately 446.5 seconds.

Given the uniform scores across the parameter grid, you may want to investigate the distribution of your target variable. If there is a class imbalance, consider using techniques such as upsampling the minority class, downsampling the majority class, or using a different metric that is more informative for imbalanced datasets, such as precision, recall, F1-score or area under the receiver operating characteristic curve (AUC-ROC).

Accuracy, F1, Classification Report

Accuracy: 0.8382829951075697 F1-score: 0.7645377580674374 /Users/romero61/.conda/envs/pyforest/lib/python3.11
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result)) /Users/romero61/.conda/envs/pyforest/lib/p
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result)) Classification report: precision recall f1-score support

0	0.84	1.00	0.91	8077959
1	0.00	0.00	0.00	1558356
accuracy			0.84	9636315

macro avg 0.42 0.50 0.46 9636315 weighted avg 0.70 0.84 0.76 9636315

/Users/romero61/.conda/envs/pyforest/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result))

It appears that your model is having trouble predicting the positive class (1 - denoting deforestation) correctly. The model is likely predicting most of the instances as the negative class (0), which is why the precision, recall and f1-score for the positive class are 0.00. This also results in a high accuracy, since the majority class (negative class) is correctly predicted most of the time.

This might be happening due to a few reasons:

Imbalanced Classes: The number of instances of the negative class (0) is significantly higher than the number of instances of the positive class (1). The model, as a result, may have a bias towards the negative class because it has more examples of it to learn from.

Insufficient Representation: It might be possible that the features for the positive class instances are not sufficiently different from those of the negative class for the model to learn to distinguish them.

Inadequate Model Complexity: The RandomForest model may not be complex enough to capture the pattern associated with the positive class instances.

There are a few strategies to deal with this:

Resampling the Data: You can try oversampling the minority class (1) or undersampling the majority class (0) to balance the classes.

Using Different Models: You might want to try other models which can handle imbalanced classes better, such as gradient boosting models (like XGBoost or LightGBM) or even neural networks.

Tuning Class Weights: Some models (including RandomForest) allow you to adjust the "class_weight" parameter, which can help in cases of imbalanced classes. By assigning a higher weight to the minority class, you are essentially telling the model to "pay more attention" to it.

Collect More Data: If possible, collecting more instances of the positive class can help the model learn to recognize it.

Feature Engineering: Creating new features or transforming existing ones may help improve the model's performance. For instance, you could create interaction terms, polynomial features, or other transformations that might help distinguish the classes.

Confusion Matrix

/Users/romero61/.conda/envs/pyforest/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result)) /Users/romero61/.conda/envs/pyforest/lib/python3.11/site-packages/sklearn/metrics/_class UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result)) Training confusion matrix: [[32307820 0] [6237437 0]] Training classification report: precision recall f1-score support

```
       0       0.84      1.00      0.91  32307820
       1       0.00      0.00      0.00   6237437

accuracy                           0.84  38545257
```

macro avg 0.42 0.50 0.46 38545257 weighted avg 0.70 0.84 0.76 38545257

/Users/romero61/.conda/envs/pyforest/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior. _warn_prf(average, modifier, msg_start, len(result))

Confirms the previous diagnosis: your model is predicting every instance as the negative class (0). That's why the recall for the positive class (1) is 0.00 - the model isn't correctly identifying any true positive instances.

As I mentioned before, this might be due to several reasons such as imbalanced classes, insufficient representation of the positive class in the feature space, or an inadequate model.

To improve your model, you could try the following:

Data Resampling: You can try to balance your classes using oversampling or undersampling techniques.

Changing Models: Try a different model that might be more suitable for imbalanced datasets, like Gradient Boosting models (XGBoost or LightGBM) or neural networks.

Class Weight Tuning: Modify the "class_weight" parameter in your RandomForest model. This tells the model to pay more attention to the minority class.

More Data: If possible, collect more data, especially instances of the positive class. This helps the model learn to recognize it.

Feature Engineering: This might help the model distinguish between classes. You can create new features, transform existing ones, or do both. This includes creating interaction terms, polynomial features, or other transformations.

Remember to adjust and try these solutions based on your specific problem and data. Also, keep monitoring the model's performance to see if the adjustments are improving its ability to correctly classify instances of the positive class.