Multinomial Classification of Forest Cover Types Using Random Forest and Boosting Methods

Ryan Elson

ISYE 7406 DMSL: Spring 2022

Abstract

Using satellite imagery, it is informative to be able to predict the forest cover type from spectral characteristics without having to manually label each observation.

To try and predict “Forest Cover Type” for a multi-class dataset, I built boosting, random forest, LDA, multinomial logistic regression, Naïve Bayes, and single tree models. Using cross-validation for all model types, I found that random forest performed best and had the lowest test error rate. Boosting provided the second-best results.

Introduction

For this homework, I am using random forest and boosting methods with a “Forest Cover Type” mapping dataset from UC Irvine (1). Each observation includes multiple spectral features (predictors) and is labeled as a type of forest or non-forest land that needs to be predicted as part of a multi-class classification problem.

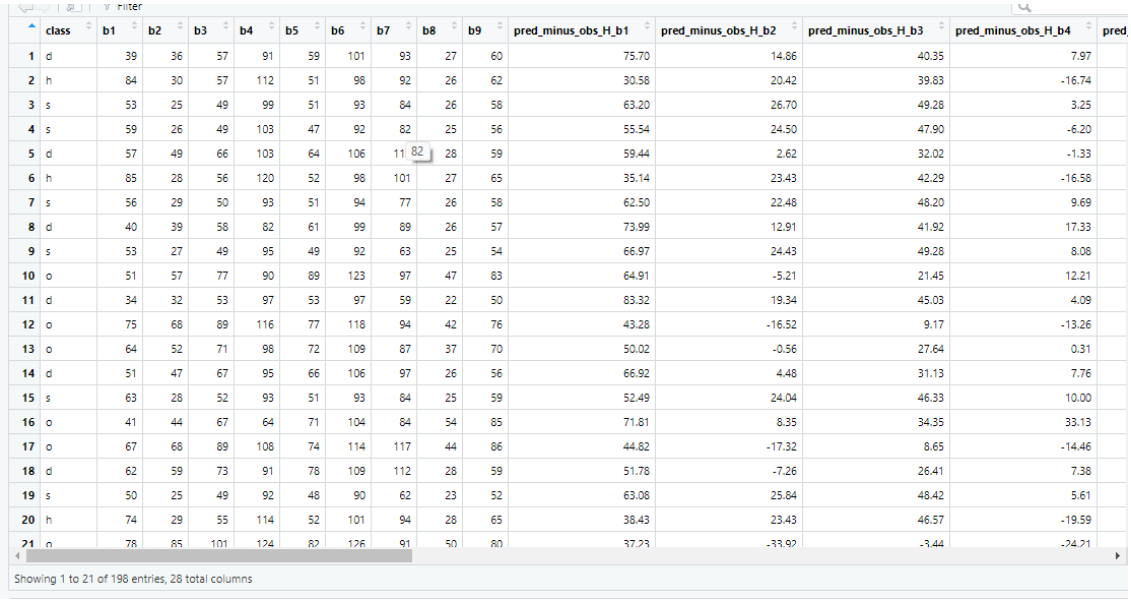
Using training and testing splits with cross-validation, I will find test error rates for random forest and boosting methods and compare them to other baseline classification methods like LDA, Naïve Bayes, Single Tree, and Multinomial Regression.

Exploratory Data Analysis and Data Sources

The dataset comes from the UC Irvine Machine Learning Repository (1) and includes 523 total observations, each with 28 features. The original data and associated paper (2) from 2012 are by Johnson, et al.

There is 1 response variable, class, and there are 27 predictor variables. The class variable is categorical with four possible values so this is a multiclass problem without ordinality, where possible classes for different forest cover types are “d, h, s, or o,” which stand for deciduous mixed forest, Hinoki forest, Sugi forest, or other non-forest land, respectively. A small portion of the dataset is shown below.

As downloaded, the data was pre-split into training and testing datasets, though the training dataset only contained 198 observations while the testing dataset contained 325 observations. Because of the unusual split between training and testing datasets, I decided to merge them and create my own splits as needed.



	class	b1	b2	b3	b4	b5	b6	b7	b8	b9	pred_minus_obs_H_b1	pred_minus_obs_H_b2	pred_minus_obs_H_b3	pred_minus_obs_H_b4	pred
1	d	39	36	57	91	59	101	93	27	60	75.70	14.86	40.35	7.97	
2	h	84	30	57	112	51	98	92	26	62	30.58	20.42	39.83	-16.74	
3	s	53	25	49	99	51	93	84	26	58	63.20	26.70	49.28	3.25	
4	s	59	26	49	103	47	92	82	25	56	55.54	24.50	47.90	-6.20	
5	d	57	49	66	103	64	106	11	28	59	59.44	2.62	32.02	-1.33	
6	h	85	28	56	120	52	98	101	27	65	35.14	23.43	42.29	-16.58	
7	s	56	29	50	93	51	94	77	26	58	62.50	22.48	48.20	9.69	
8	d	40	39	58	82	61	99	89	26	57	73.99	12.91	41.92	17.33	
9	s	53	27	49	95	49	92	63	25	54	66.97	24.43	49.28	8.08	
10	o	51	57	77	90	89	123	97	47	83	64.91	-5.21	21.45	12.21	
11	d	34	32	53	97	53	97	59	22	50	83.32	19.34	45.03	4.09	
12	o	75	68	89	116	77	118	94	42	76	43.28	-16.52	9.17	-13.26	
13	o	64	52	71	98	72	109	87	37	70	50.02	-0.56	27.64	0.31	
14	d	51	47	67	95	66	106	97	26	56	66.92	4.48	31.13	7.76	
15	s	63	28	52	93	51	93	84	25	59	52.49	24.04	46.33	10.00	
16	o	41	44	67	64	71	104	84	54	85	71.81	8.35	34.35	33.13	
17	o	67	68	89	108	74	114	117	44	86	44.82	-17.32	8.65	-14.46	
18	d	62	59	73	91	78	109	112	28	59	51.78	-7.26	26.41	7.38	
19	s	50	25	49	92	48	90	62	23	52	63.08	25.84	48.42	5.61	
20	h	74	29	55	114	52	101	94	28	65	38.43	23.43	46.57	-19.59	
21	o	78	85	101	124	82	126	91	50	80	37.23	-33.92	-3.44	-24.21	

Showing 1 to 21 of 198 entries, 28 total columns

One thing I noticed while exploring the data is that the classes are unbalanced when training and testing data is combined (as shown below).

	Count
d	159
h	86
s	195
o	83

Classes were reasonably well balanced if the training set was used on its own, however. Regardless, in order to use the combined data, I oversampled the “h” and “o” classes in order to have similar counts across the four labels. This gave a dataset that was much better balanced. My final dataset had the below class counts.

	Count
d	159
h	172
s	195
o	166

Proposed Methodology

My methodology begins with downloading the data and merging the provided training and testing datasets into one larger dataset. This was done because of the unusual train/test split as noted in the previous section. Next, I had to oversample some classes to ensure my data was reasonably well balanced. For my analysis, when splitting this merged and oversampled data, I split randomly so I had 80% training data and 20% testing data when building my models.

For boosting, I used cross-validation with 10-fold CV. I built my model with 5000 boosting iterations and selected the number of iterations with the maximum accuracy as my selected model (see Appendix for additional discussion on selected metric). I used that model to predict test dataset classifications and took the class with the highest probability as the predicted class.

For the other models (Multi-class Logistic Regression, LDA, Naïve Bayes, Single Tree, and Random Forest), I used 100 cross-validation iterations and for each iteration I randomly split the data into training and testing datasets. For each iteration, I built each model with the training data and found the error rate for the test data. The overall test error rate for each model was calculated as the average of all 100 individual test error rates.

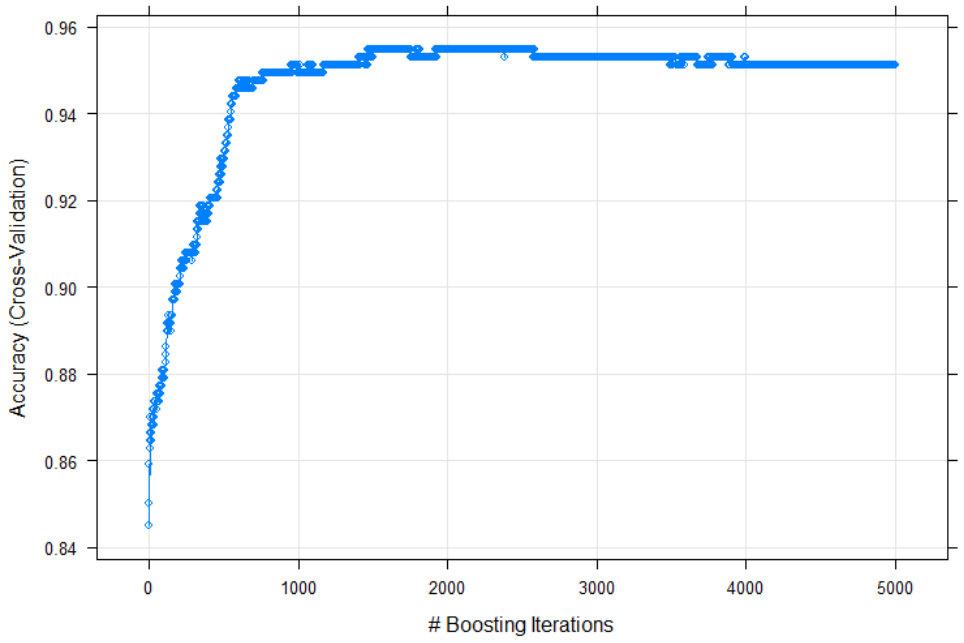
Using cross-validation provides a much more robust and accurate overall test error rate since there is no concern that any given random split may have provided an outlier result where the test error rate was unusually high or low.

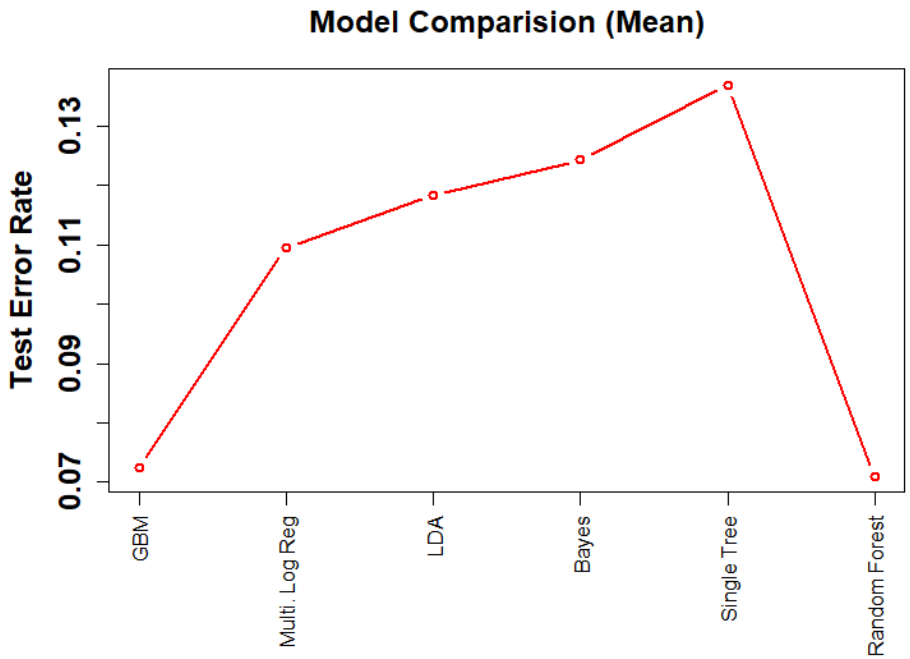
Analysis and Results

For the boosting algorithm (GBM), the accuracy versus number of boosting iterations is provided in the first plot on the following page. As shown, accuracy improves as boosting iterations are increased before largely leveling off. Using the model with the number of boosting iterations that provided the best accuracy, I predicted the classifications for the test data and found the test error rate.

The second plot shows the different test error rates for the different models. For the given dataset, the random forest model provided the smallest testing error. The second-best error rate was the boosting model, followed by Multi-Class Logistic Regression and the remainder of the models as shown in the following table.

	Test Error
GBM	0.0725
Multi-Class Logistic Reg	0.1096
LDA	0.1184
Naïve Bayes	0.1243
Single Tree	0.1370
Random Forest	0.0709





Conclusion

In conclusion, random forest provided the best model with the lowest test error rate. Boosting was very similar to random forest in providing a small test error rate, but it took the longest to run, so there may be a computational expense that exceeds that of other models. While it performed similarly to random forest, if a large dataset was being analyzed it's possible that it might be too computationally expensive. As such, it makes sense to use random forest rather than boosting with this data.

Possible future work includes extending this type of analysis to other datasets with additional cover types, or possibly combining all the forest types (deciduous, Hinoki, and Sugi) and predicting a combined forest class against a single non-forest class. This would be a simpler binary response problem where an observation was forest or non-forest. The analysis could also be redone using the provided training and testing datasets as they were downloaded, ignoring the fact that the testing dataset had more observations than the testing set.

Lessons I have Learned

Some lessons I have learned are how powerful boosting and random forest can be. I also learned how to perform a multi-class analysis, which extended my knowledge beyond a simple binary response problem. I also learned about the Kappa performance metric (as discussed in the Appendix), though it was unclear if it is beneficial to use it.

Most importantly, it was reinforced in my mind that having balanced or relatively balanced classes is important when building models to perform classifications. Initially, I didn't balance my classes after I combined the training and testing datasets. This resulted in models that were less accurate. It was after I realized that the classes were imbalanced that I corrected this problem and redid my models and predictions.

Appendix

Additional Technical Discussion

Accuracy was used as the metric for the boosting model, although there are other options.

I also tried to use "Kappa" as the metric for the boosting model, and I ended up with the same index (i.e., the same number of boosting iterations) for the max Kappa value as I did for the maximum Accuracy value. As such, the same model was selected, and my test error rate didn't change from that presented in the Results section.

However, before I balanced the classes, when I used Kappa, I found a test error rate of 14.29% which was more in-line with Naïve Bayes, Multiclass logistic regression, and single tree models (again, before balancing classes). This was a higher test error rate than found with Accuracy, which points out that at least in some cases, particularly when classes are unbalanced, Kappa and Alpha may return different values and subsequently select different models.

Ultimately, choosing Accuracy or Kappa as the performance metric didn't matter since they gave the same result once I balanced the dataset. However, according to some articles (3), using Kappa may introduce problems and should be avoided. For that reason, I was planning to use accuracy as my metric for the boosting algorithm even if the results were different.

Bibliography and Credits

1. UCI Machine Learning Repository: Forest Type Mapping Data Set, https://archive.ics.uci.edu/ml/datasets/Forest+type+mapping, Accessed 21 Mar 2022

2. Johnson, B., Tateishi, R., Xie, Z., 2012. Using geographically-weighted variables for image classification. Remote Sensing Letters, 3 (6), 491-499.

3. Delgado R, Tibau X-A (2019) Why Cohen's Kappa should be avoided as performance measure in classification. PLoS ONE 14(9): e0222916. https://doi.org/10.1371/journal.pone.0222916