Out-Of-Bag Error in Random Forests

Tom Blain

Data Science Portfolio A - Section II

Abstract

In this paper we will introduce decision trees and random forests, with the goal of exploring how we can use out-of-bag (OOB) samples as a computationally efficient alternative to cross validation (CV) methods. We experiment by generating a standard classification dataset, and building a random forest classifier with varying values of max_features (mtry). We conclude that out-of-bag error may not be a good choice for tuning the mtry parameter since OOB error seems to become more biased than CV when the parameter is small, but more investigation will have do be done with different varying datasets.

1 Motivation

Decision trees, random forests, and boosting, will often be go to reliable methods for facing a regression or classification task. On a surface level, the algorithm is straightforward, explainable, and interpretable, however it is of importance to be interested in the properties of bootstrapping methods that play a large part in random forests. The course notes introduce these models and explain how you might go about implementing them, but we would like to ask the questions "Why is a random forest random?" and "How can we get the best results from our models without overfitting?".

To extend the course notes, we can take an experimental approach to better see the direct consequences of bootstrapping, which can then be applied to implementations of the methods. Since struggling with gauging accuracy of model performance in our recent DST project, we might be interested in investigating how the bootstrapping element of random forests allows us to use OOB error as an alternative to something such as K-fold CV. Kaggle competitions might have trained me to overfit, so CV methods are vital in accurate assessments of model performance, and parameter optimisation. Another issue with using a Grid Search in the last project with k-fold CV was the computational cost which forced me into changing the entirely of my XGBoost model to LightGBM so it would run faster - OOB error requires much less computation than K-fold. It is vital as a data scientist to be able to work with larger and larger datasets in the future where an algorithm will struggle with speed.

The paper, 'On the overestimation of random forest's out-of-bag error' provides an experimental approach to diving into issues relating to out-of-bag error in varied conditions relating to parameter choices and datasets. Reading this paper satisfied the gap in my knowledge for my initial questions since it helps provide justification for the possibility of straying away from the standard technique of K-fold CV, in certain situations.

2 Introduction

Tree-based learning methods, such as decision trees and random forests, are known for their simplicity and interpretability in terms of the results they produce for classification and regression tasks. Despite this, they have proven to be among the most powerful and successful algorithms in the field of machine learning, particularly in competitive environments such as Kaggle competitions. It is therefore of importance to study how we can ensure high performance with techniques such as Out-of-Bag (OOB) error and Cross Validation (CV) which can provide a robust estimate of the model performance.

2.1 Decision Trees

The decision tree algorithm is a powerful tool that can be applied to both regression and classification problems in machine learning. In regression models, the goal is to predict a continuous target, such as a price or temperature. In contrast, for a classification problem, the goal is to predict the class or category to which an observation belongs, represented by a discrete variable.

A decision tree works by making splits on the features of the data, in order to divide it into smaller and more homogeneous groups. The algorithm starts at the root node and for each non-leaf node, it evaluates all the features and chooses the one that provides the most information gain, based on a chosen impurity measure. Then it splits the data into two or more subsets, depending on the chosen feature. This process is repeated recursively, until a stopping criterion is met, such as reaching a maximum depth or a minimum number of samples per leaf [1].

The final result is a tree structure, where each internal node represents a feature, each branch represents a split, and each leaf node represents a prediction of the target variable.

Decision trees can be effective as a starting point for many problems and can be used to get some insight about the relationships in the data. But generally, they are not powerful enough to achieve high accuracy on complex problems. Ensemble methods such as random forests can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation [2].

2.2 Random Forests

Random forests are an ensemble of decision trees, where each tree is trained on a different subset of the data, also known as bootstrapping. By averaging the predictions of these multiple trees, the random forest is able to produce a more accurate and reliable prediction than any individual tree alone. This helps to reduce overfitting and improve the overall accuracy of the model. Additionally, random forests can handle high dimensional data with categorical and numerical variables, and it can also estimate feature importance. The many decision trees in a random forest allow high performance on non-linear relationships in the data by using different combinations of features and different split points for those features [3].

In a random forest, bootstrapping is used to create multiple training sets for building the decision trees. The process is as follows: For each tree in the forest, a new training set is created by randomly selecting observations from the original training set with replacement. This means that some observations may be repeated in the new training set, while others may not be included at all. The data not included is known as the out-of-bag samples. This results in different training sets for each tree, and therefore different decision trees. This technique, known as bootstrap aggregating or bagging, helps to reduce the variance of the final model and improve its overall performance.

Random forests also introduce randomness in the process of feature selection for each split, known as random subspaces method, which allows the decision trees in the forest to consider different subsets of features at each split, and this increased diversity of features will help the model to capture non-linear relationships. max_features (or mtry) is an important parameter choice since it will have a strong affect on the randomness in the model by limiting the amount of features we consider in each tree.

3 Bias-Variance

Bias refers to the difference between the average prediction of a model and the true value of the target variable. A model with high bias tends to make the same prediction for all data points, regardless of the input. A model with high bias is said to be underfitting the data, which means it is not capturing the underlying patterns in the data.

Variance refers to the variability of the model's predictions for different sets of training data. A model with high variance is sensitive to the noise in the training data, and it tends to make predictions that are far from the true value of the target variable. A model with high variance is said to be overfitting the data, which means it is capturing the noise in the data rather than the underlying patterns. [4]

In decision tree models, increasing the depth of the tree increases the variance and reduces the bias. So, by controlling the depth of the tree we can balance the trade-off between bias and variance and control the complexity of the model. Techniques like pruning, which removes the tree branches, also helps to control the complexity of the model and balance the trade-off between bias and variance.

Decision trees are ideal candidates for ensemble methods since a simple tree will have low bias and high variance, making them likely to benefit from the averaging and randomness process.

3.1 Bagging

Earlier when investigating bootstrapping, we found that the bootstrap mean is approximately a posterior average. Bagging further exploits this connection.

Bagging works by randomly selecting N samples from the dataset D, with replacement, to create a new dataset D_i (also called bootstrap sample). This process is repeated multiple times to create multiple datasets $D_1, D_2, ..., D_B$, where B is the number of bootstrap samples.

Then, we train the model f on each of these bootstrap samples and obtain multiple versions of the model $f_1, f_2, ..., f_B$.

For a regression problem, we average the predictions of all the versions of the model. For a new input x, the final prediction of bagging is given by:

$\hat{y} = \frac{1}{B} \sum_{i=1}^{B} f_i(x)$

where $\hat{y}$ is the final prediction and $f_i(x)$ is the prediction of the i-th version of the model.

For classification, we take a majority vote from the bagging predictions.

3.2 Decorrelated trees

For Random forests, we also introduce randomness in the feature selection process. When building each decision tree, instead of considering all features, a random subset of features is chosen at each split. This helps to reduce the correlation between the trees and improve the overall performance of the model.

Let D be the original training set. For each $i \in 1, 2, ..., n$, a new training set D_i is created by randomly sampling D with replacement. When building each decision tree T_i, a random subset of features F_i is chosen at each split instead of considering all features. The final prediction $\hat{y}$ is made by averaging the predictions of all the trees for regression or taking a majority vote for classification.

When the trees in a random forest are highly correlated, they tend to make similar predictions, which can lead to overfitting and poor generalization to new data. By decorrelating the trees, we ensure that each tree is making slightly different predictions, which helps to reduce the overall variance of the model. This results in a more stable and accurate model [5].

Consequently, random forests are a more powerful method than bagging since it is less likely to overfit.

4 Out Of Bag Error

With ensemble methods that construct models on bootstrap samples, like Bagging or Random Forests, we have the possibility of using the left-out samples to form estimates of important statistics.

For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which zi did not appear.

Out-of-bag estimates constitute a computationally efficient alternative to K-fold cross-validation, since they are not used in model training and will only require n iterations of the random forest, in contract to K-fold CV which will require Kn

While out-of-bag estimates constitute an helpful tool, their benefits should however be put in balance with the potential decrease of accuracy that the use of bootstrap replicates may induce. As shown experimentally in [6], bootstrapping is in fact rarely crucial for random forests to obtain good accuracy. On the contrary, not using bootstrap samples usually yield better results.

5 Experimentation

In this paper, we wish to investigate the OOB error to find out if this is a reliable alternative to cross validation techniques for random forests. A general conclusion will require extensive case by case analysis, however, a conclusion reached by [7] seems to suggest that a smaller mtry paramter (maximum features) for random forests may increase the bias in our OOB errors. Mtry parameter selection is essential for ensuring randomness in the random forest and hence preventing overfitting. This section will follow along with the part of the experimentation conducted in the paper [7] to replicate the results and draw a conclusion for a typical use case in random forests for classification tasks.

5.1 Generalisation

We have chosen to try and verify these results hold in a simulated setting which attempts to mimic a typical scenario where a random forest would be applicable. Hence, for this experiment, we generate a binary classification dataset.

```
# Generate complex classification dataset
X, y = make_classification(n_samples=1000, n_features=20,
    n_informative=15, n_classes=2,
    n_clusters_per_class=2, random_state=42)

# Initialize Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, oob_score=True, max_features=m)
```

5.2 Limitations

The dataset attempts to replicate a common scenario in classification problems with 1000 data points, and 20 features. The classes are set as balanced, and more experimentation may need to be done to verify these results on situations with class imbalance. More experimentation would also have to take place to verify any results on datasets with fewer data, or a much larger/smaller amount of informative features included.

5.3 Results

The RF classifier is set with default parameter choices apart from the max_features, since this is the variable we are investigating in this experiment.

The code then fits the rf, and calculates OOB error, Stratified OOB error, CV error, and Stratified CV error. We repeat this process 1000 times generating a new dataset for each, and plot the results. The code used is available at https://gist.github.com/TBlainUoB/d0b97442063a8daf888c44b411b364d9

1

5.4 Conclusion

The outcome of our experiment follows along with the original results cited in the paper. We see a larger amount of bias on the OOB errors at a small value of mtry. As the value of mtry increases, the error becomes similar to that of the CV error. Note here we are comparing the non-stratified errors together and the stratified errors together.

The experiment shows that it may be worth considering if you should use the OOB error in a similar scenario to that tested here. The changes in bias when varying the mtry parameter may mean that OOB error would not be a good choice when tuning the mtry parameter. The OOB error is computationally faster to compute, but the increased bias may be significant in your own modelling application if using this as an alternative to CV. It is also interesting to see that the stratified CV performs best in all tests, however, choosing to use stratified approaches often depends on the case and the data you are working on.

6 Future Investigation

More experimentation needs to be done measuring OOB error with different parameter selections and datasets. Some open research questions include how we can accurately estimate OOB error with the sensitivity to the number of trees and subsampling ratio. In the future we would like to extend this investigation to other common conditions where you will implement a random forest model, since the computational efficiency of OOB error would be powerful in situations

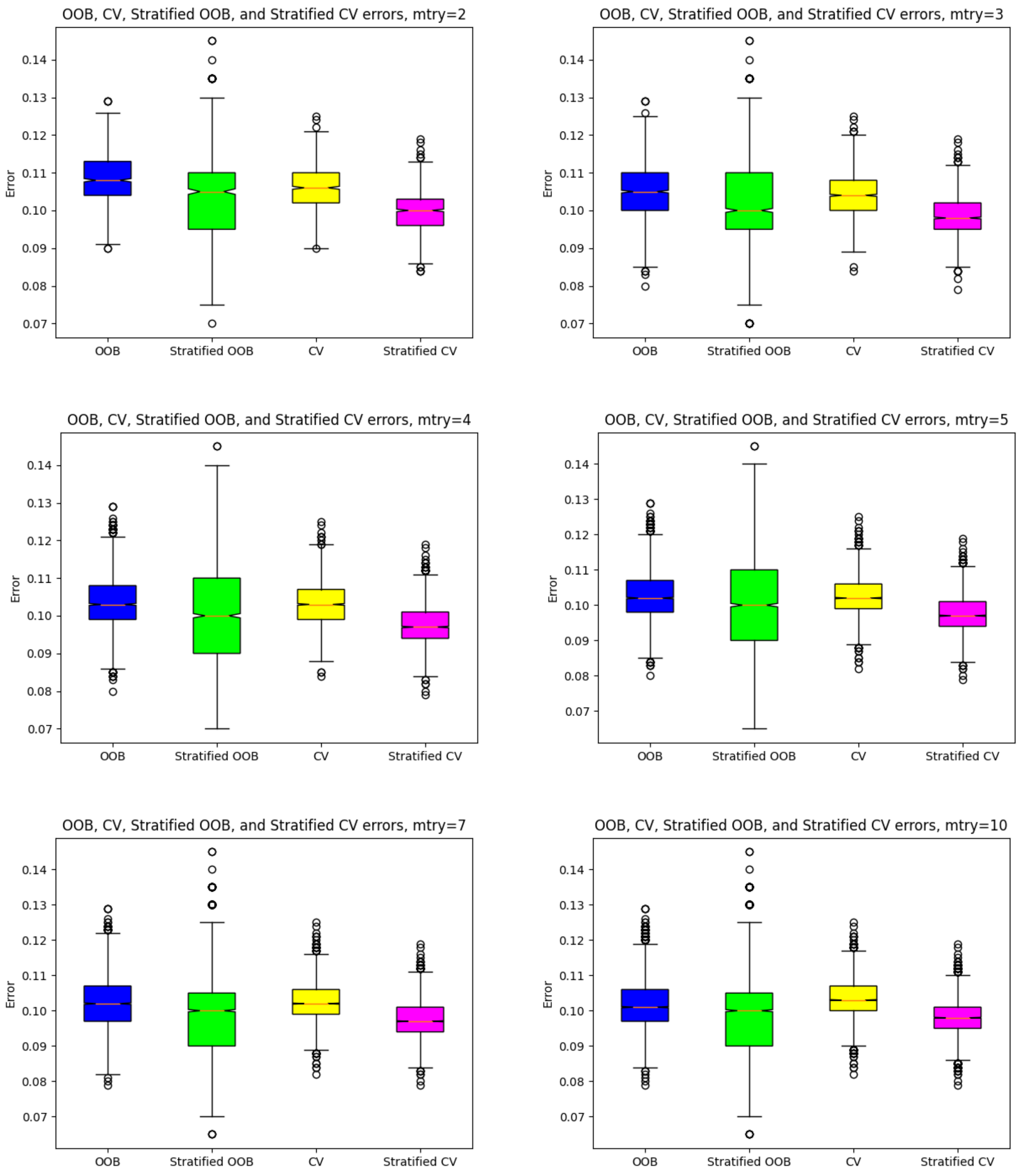

Figure 1: Errors at different values of mtry

where computational memory or computational cost has strong importance. Furthermore, we should investigate if a theoretical link can be made which to solidify the experimental results in this paper.

7 Reflection

Investigating this area was worthwhile to allow me to have a better understanding of random forests and the consequences of bootstrapped samples for training. By first studying the recommended further reading, I began by learning about the basic decision tree model, and built strong fundamental insights for the strengths and weaknesses of the design. This naturally led to better understanding how random forests can improve on this model in almost every way apart from possibly interpretability and speed/memory. I next studied bias-variance since this is one of the most important elements of data science when building a model. I have never been sure how to best evaluate model performance, for both training and final evaluation. Having a strong grasp on bias-variance will allow me to create better models in the future, especially for powerful ensemble methods where it is essential to layer your model in a way which best minimises this.

The background then led into out-of-bag samples. Many mathematical results can start to be applied here, however, I did not feel this would be the most beneficial route to go down to further my own understanding and confidence applying the knowledge - I like the experimental approach of the base paper and felt that building a model in a typical use case I would encounter would allow me to prove the results for myself. Quoting the original paper, ”Prior to our work, little had been known about the bias of the OOB error, and the OOB error is still frequently used for error estimation in classification settings”, It therefore felt worthwhile to replicate the results.

The investigation was insightful about the care and depth required to fully utalise the power of random forests. The first paper helped me to answer some of my questions on bootstrapping, and this paper allowed me to take that knowledge further in a primary use of bootstrapping within data science. I am satisfied that I will be able to come back to this paper in the future and reuse the code for some further investigation if required. It has already been the case that I have encountered problems with speed when using random forest algorithms and this could be an area where OOB samples help. I have learnt that boosting might be superior to random forests in a lot of cases, however, we could find uses for random forests when imputing missing values, and this can also be an area where speed can matter.

References

[1] S. Kotsiantis, “Decision trees: a recent overview,” *Artif Intell Rev 39, 261–283 (2013)*. [Online]. Available: https://towardsmachinelearning.org/decision-tree-algorithm/

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani, *Tree-Based Methods*. New York, NY: Springer New York, 2013, pp. 303–335. [Online]. Available: https://doi.org/10.1007/978-1-4614-7138-7_8

[3] R. J. McAlexander and L. Mentch, “Predictive inference with random forests: A new perspective on classical analyses,” *Research & Politics*, vol. 7, no. 1, p. 2053168020905487, 2020. [Online]. Available: https://doi.org/10.1177/2053168020905487

[4] S. Fortmann-Roe, “Understanding the bias-variance tradeoff,” 2012. [Online]. Available: https://scott.fortmann-roe.com/docs/BiasVariance.html

[5] stackexchange, “Why is tree correlation a problem when working with bagging?” [Online]. Available: https://stats.stackexchange.com/questions/295868/why-is-tree-correlation-a-problem-when-working-with-bagging

[6] G. Louppe and P. Geurts, “Ensembles on random patches,” in *ECML/PKDD*, 2012.

[7] S. Janitza and R. Hornung, “On the overestimation of random forest’s out-of-bag error,” *PLOS ONE*, vol. 13, pp. 1–31, 08 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0201904

A Parameters

A.1 Decision Trees

Maximum depth: This parameter controls the maximum depth of the tree. A deeper tree will have more splits, resulting in a more complex model with more branches. A shallower tree will have fewer splits, resulting in a simpler model with fewer branches.

Minimum number of samples per leaf: This parameter controls the minimum number of samples that are required to be at a leaf node. A higher value will result in a tree with fewer leaf nodes, and it will be less complex.

Minimum number of samples per split: This parameter controls the minimum number of samples that are required to split an internal node. A higher value will result in a tree with fewer splits, and it will be less complex.

Maximum number of features: This parameter controls the maximum number of features that are considered for each split. A higher value will result in a tree that considers more features for each split, and it will be more complex.

Criterion: This parameter controls the impurity measure that is used to evaluate the quality of a split. The most common criterion are Gini impurity and information gain.

Splitter: This parameter controls the strategy used to choose the split point of a feature. The most common strategies are best and random.

A.2 Random Forests

A random forest has a few additional parameters that a decision tree does not have:

n_estimators: This parameter controls the number of decision trees in the forest. Increasing the number of trees will generally improve the performance of the model, but at the cost of increased computational time and memory usage.

max_features: This parameter controls the number of features that are considered when splitting a node in the decision tree. Setting a smaller value will reduce the correlation between the trees in the forest and decrease the chance of overfitting.

random_state: This parameter is used to control the randomness in the training process. It is used to seed the random number generator that is used to create the bootstrapped training sets and select random subsets of features at each split.

oob_score: This parameter controls whether the out-of-bag (OOB) accuracy is calculated during training. The OOB accuracy is a measure of the model's performance on unseen data, and can be used to estimate the generalization error of the model.

max_samples: This parameter controls the size of the bootstrapped dataset for each tree, which can be used to control the trade-off between the number of trees and the size of the dataset.

n_jobs: This parameter controls the number of CPU cores used for parallel computing. This can be used to speed up the training process.

warm_start: This parameter allows to reuse previous tree(s) when fitting new random forest.

Additionally, random forests also have a feature_importances_ attribute, which gives the importance of each feature in the decision making process.