Toggle code

Week 9 - Tree-Based Methods

Dr. David Elliott

1. Majority Voting
2. Bagging
3. Random Forests

NOTES

Question: Why not just stick to decision trees?

Answer: Although they are useful for interpretation, they are not as competative for supervised learning methods, so we'll also talk about bagging, random forests, and boosting; which all rely on producing multiple trees and combining them to gain a single prediction. This typically improves accuracy a lot, but at the expense of interpretability.

The goal of ensemble methods is to combine different classifiers into a metaclassifier that has better generalization performance than each individual classifier alone[4].

Most popular ensemble methods use the majority (or plurality) voting principle.

Notes

- If I posed a complex question to thousands of random people and aggregated their answer, in a lot of cases that answer is better than asking a single expert[3].

4. Majority Voting

Majority voting can be done by simply selecting the class label that has been predicted by the majority of the classifiers (more than 50% of votes).

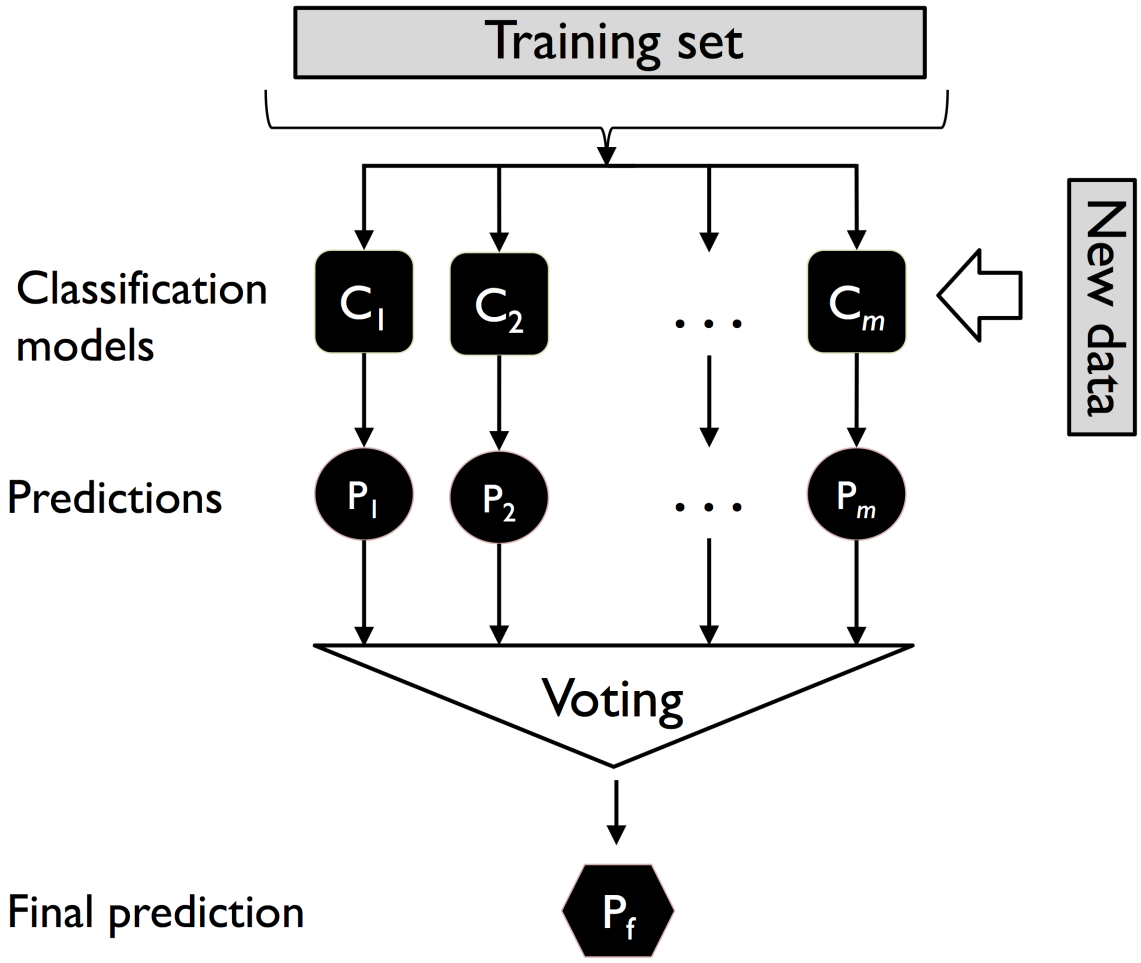
Majority vote refers to binary class decisions but can be generalized to a multi-class setting using *plurality voting*[4].

To predict a class label via simple majority or plurality voting, we start by training m different classifiers ($C_1, \ldots, C_m$), and then combine the predicted labels of each classifier, C_j, selecting the class label, $\hat{y}$, that received the most votes[4].

Notes

- choosing the most commonly voted for class is sometimes called 'hard voting'.
- Scikit-learn has a `VotingClassifier` where multipule classification pipelines can be combined to create a classifier that aggregates predictions.

Figure 22: Majority Voting Classifier



In binary classification (-1,1) we can write the majority vote prediction as[4]:

$$\hat{y} = \text{mode}\{C_1(x), C_2(x), \ldots, C_m(x)\}$$

$$C(x) = \text{sgn}\left[\sum_j^m C_j(x)\right] = \left\{\begin{matrix} 1 \text{ if } \sum_j C_j(x) \geq 0 \\ -1 \text{ otherwise} \end{matrix}\right\}$$

Instead we may want to weight the votes:

$$\hat{y} = \underset{i}{\arg\max} \sum_{j=1}^{m} w_j 1_A(C_j(x) = i)$$

where w_j is a weight associated with a base classifier, C_j ; $\hat{y}$ is the predicted class label of the ensemble; A is the set of unique class labels; 1_A is the characteristic function or indicator function, which returns 1 if the predicted class of the jth classifier matches $C_j(x) = i$.

Particular classifiers return the probability of a predicted class label and this can be used for *"soft voting"* instead of class labels[4]. The modified version of the majority vote for predicting class labels from probabilities can be written as:

$$\hat{y} = \arg \underset{i}{\max} \sum_{j=1}^{m} w_j p_{ij},$$

where p_{ij} is the predicted probability of the jth classifier for class label i.

Soft voting often achives a higher performance than hard voting because highly confident votes are given more weight[3].

NOTES

- Scikit-learn uses the `predict_proba` method to compute class probabilities.
- *"In decision trees, the probabilities are calculated from a frequency vector that is created for each node at training time. The vector collects the frequency values of each class label computed from the class label distribution at that node. Then, the frequencies are normalized so that they sum up to 1...Although the normalised probabilities returned by both the decision tree and k-nearest neighbors classifier may look similar to the probabilities obtained from a logistic regression model, we have to be aware these are actually not derrived from probability mass functions."*[4]

SVM
```
Training ACC Score: 0.99
Validation ACC Score: 0.96
```

LR
```
Training ACC Score: 0.99
Validation ACC Score: 0.96
```

DT
```
Training ACC Score: 1.00
Validation ACC Score: 0.90
```

Hard Majority Voting
```
Training ACC Score: 0.99
Validation ACC Score: 0.96
```

Soft Majority Voting
```
Training ACC Score: 1.00
Validation ACC Score: 0.97
```

5. Bagging

A bagging classifier is an ensemble of base classifiers, each fit on random subsets of a dataset. Their predictions are then pooled or aggregated to form a final prediction.

This reduces *variance* of an estimator so can be a simple way to reduce overfitting and increase prediction accuracy[5].

We could use bagging by taking many separate training sets, B, from the population, building a separate prediction model using each training set, $\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$, and average the resulting predictions[1]:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x).$$

However we generally do not have access to multiple training sets so instead we can bootstrap by taking repeated samples from a (single) training data set to create multiple bootstrapped training data sets, B. We then train our method on the bth bootstrapped training set to get $f^{*b}(x)$, and finally average all the predictions, to obtain[1]:

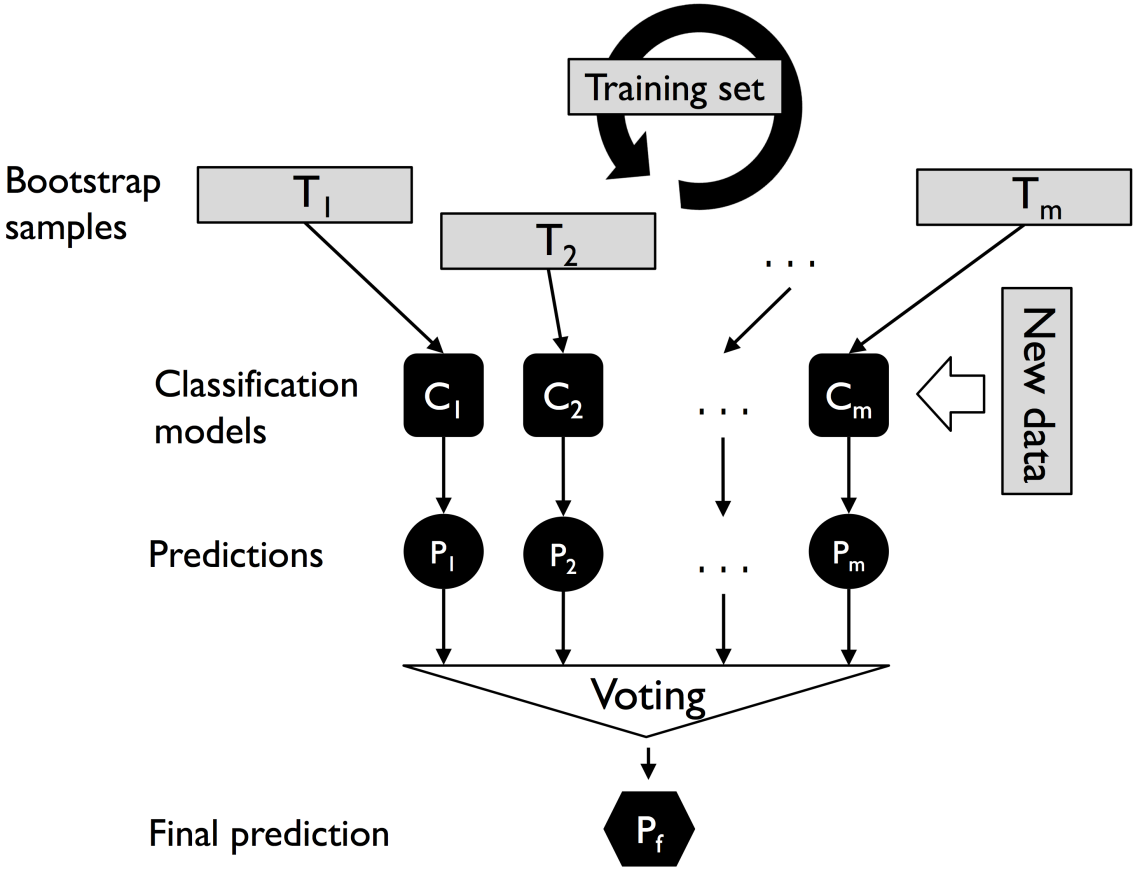
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

Notes

- This is because the variance of the mean $\bar{Z}$ of n independent observations, $Z_1, \ldots, Z_n$, is given by σ^2/n; meaning averaging a set of observations typically reduces variance.
- **High Variance** means if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- **Low Variance** means a procedure will yield similar results if applied repeatedly to distinct data sets.

- Decision trees typically suffer from *high variance*.
- Linear regression tends to have low variance, if the ratio of n to p is moderately large.

Figure 23: Bagging Classifier



Specifically, bagging is when sampling is produced with replacement[6], and without replacement being called *pasting*[7].

Pasting is designed to use smaller sample sizes than the training dataset in cases where the training dataset does not fit into memory[7].

Both bagging and pasting allow training to be sampled several times across multiple predictors, with bagging only allowing several samples for the same predictor [3].

```
Original Data
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
Bagging (0.8 Samples)
[array([9, 2, 9, 7, 7, 8, 3, 2]),
 array([7, 3, 7, 3, 9, 5, 3, 9]),
 array([6, 0, 7, 7, 9, 8, 5, 8]),
 array([0, 5, 7, 4, 1, 3, 6, 2]),
 array([6, 9, 6, 3, 3, 8, 3, 7])]
Pasting (0.8 Samples)
[array([6, 7, 5, 8, 3, 0, 4, 1]),
 array([0, 4, 5, 6, 2, 1, 8, 9]),
 array([5, 3, 9, 4, 8, 2, 1, 7]),
 array([9, 3, 6, 1, 8, 4, 2, 7]),
 array([1, 5, 4, 2, 0, 8, 7, 3])]
```

Averaging methods generally work best when the predictors are as independent as possible, so one way of achiving this is to get diverse classifiers[3].

Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

These trees are grown deep, and are not pruned so each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.

To apply bagging to *decision trees*, we simply:

1. construct B decision trees using B bootstrapped training sets,
2. take a majority vote of the resulting predictions.

Notes

- Diverse classifiers increase the chance they each make different types of errors which in combination will improve the overall accuracy[3].
- In practice, *bagging* tends to work best with complex models[5], so although bagging is useful when applied to regression methods, they are particularly useful for decision trees[1].
- *boosting* will generally work better with weak models.
- To apply bagging to **regression trees**, we simply:
    1. construct B regression trees using B bootstrapped training sets,
    2. average the resulting predictions.

Out-of-Bag Error Estimation

Using a bagged model means we can get a validation/test error without using cross-validation.

As each tree does not use all observations for training, we can predict the remaining observations.

With a sufficiently large amount of bags, OOB error is virtually equivalent to leave-one-out cross-validation error, which is convenient when performing bagging on large data sets for which cross-validation would be computationally onerous[1].

```
Bagged OOB Accuracy: 0.96 (1.75 secs)
Leave-one-out Cross-validation Accuracy: 0.96 (57.25 secs)
```

Variable Importance Measures

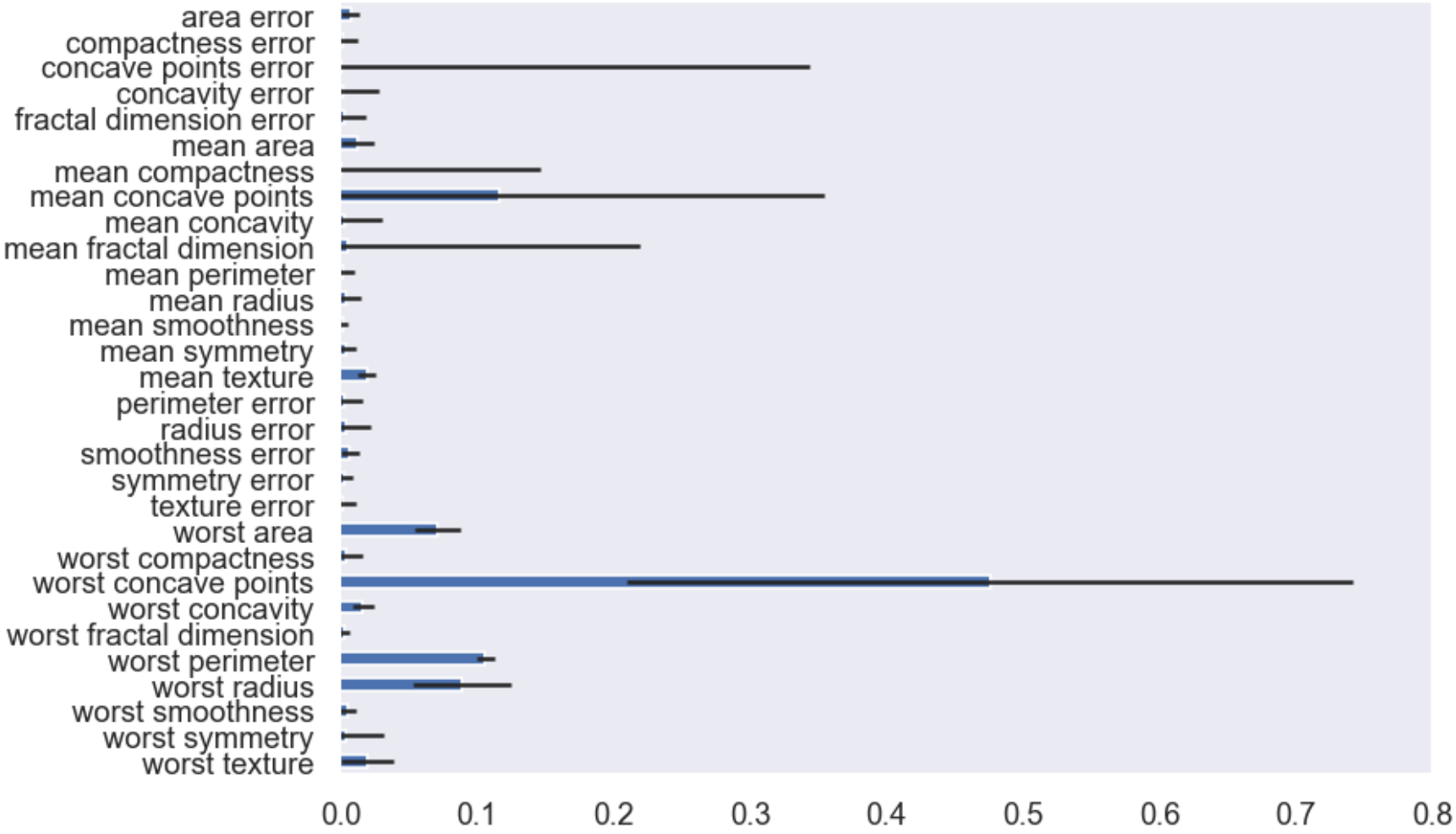
Bagging improves prediction accuracy at the expense of interpretability.

However we can use the feature importance method as previously discussed, to get an overall summary of the importance of each predictor across trees.

NOTES

- Bagging can be used with most classifiers, although you can only assess feature importances if a method provides a `feature_importances_`.
- *"Recall that one of the advantages of decision trees is the attractive and easily interpreted diagram that results... However, when we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure."*[1]

Figure 24: Average (and Standard Devision) Decision Tree Feature Importances in a Bagged Classifier



6. Random Forests

Random forests are among the most widely used machine learning algorithm[8].

Random forests are essentially bagged tree classifiers, but decorrelate the trees by using a random sample of features each time a split in a tree is considered. The random forest algorithm can therefore be summarized in four steps[4]:

1. Draw a random bootstrap sample of size n.
2. Grow a decision tree from the bootstrap sample. At each node:

   a. Randomly select d features without replacement (typically the square root of the total number of predictors).

   b. Split the node using the feature that provides the best split according to the objective function.

3. Repeat the steps above k times.

4. Aggregate the prediction by each tree to assign the class label by majority vote.

NOTES

- Random forests fit decision trees on different bootstrap samples, and for each decision tree, select a random subset of features at each node to decide upon the optimal split. The feature subset to consider at each node is a hyperparameter that we can tune[8].
- Instead of using majority vote, as was done in the origional publication[9], in Sklearn the `RandomForestClassifier` averages the probabilistic prediction.
- Remember we cannot use `graphviz` or `sklearn.tree.plot_tree` on the whole forest as we did for the trees, as each tree is built differently.
- if a random forest is built using all features, then this is simply bagging.
- you can also bootstrap features in the `BaggingClassifier` using `bootstrap_features=True`

By not allowing the model to use the majority of the available predictors, we ensure the bagged trees look different from each other.

If there is a particularly strong set of predictors in the data, then without randomly selecting features, the bagged trees will look quite similar to each other and predictions will be highly correlated.

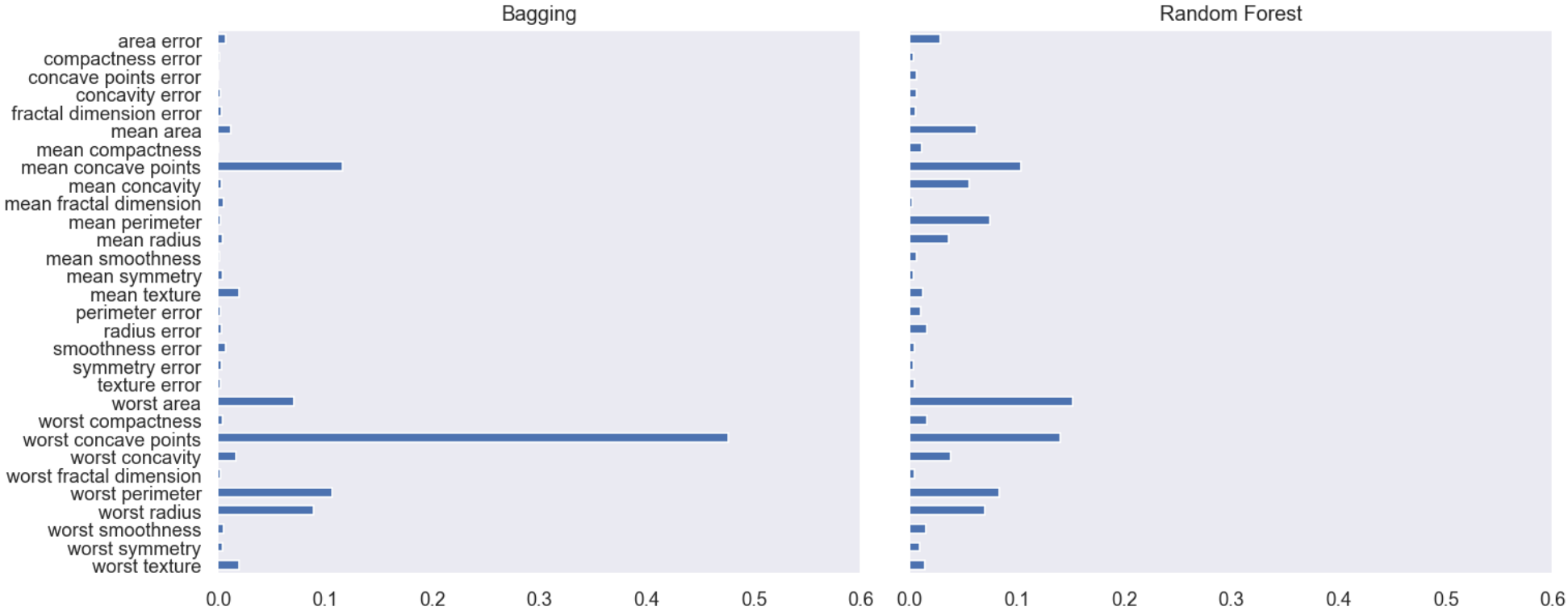
Averaging highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities[1].

Notes

- *"The final feature importance, at the Random Forest level, is it's average over all the trees. The sum of the feature's importance value on each trees is calculated and divided by the total number of trees"* https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e
- Using notation from the previous notebook:

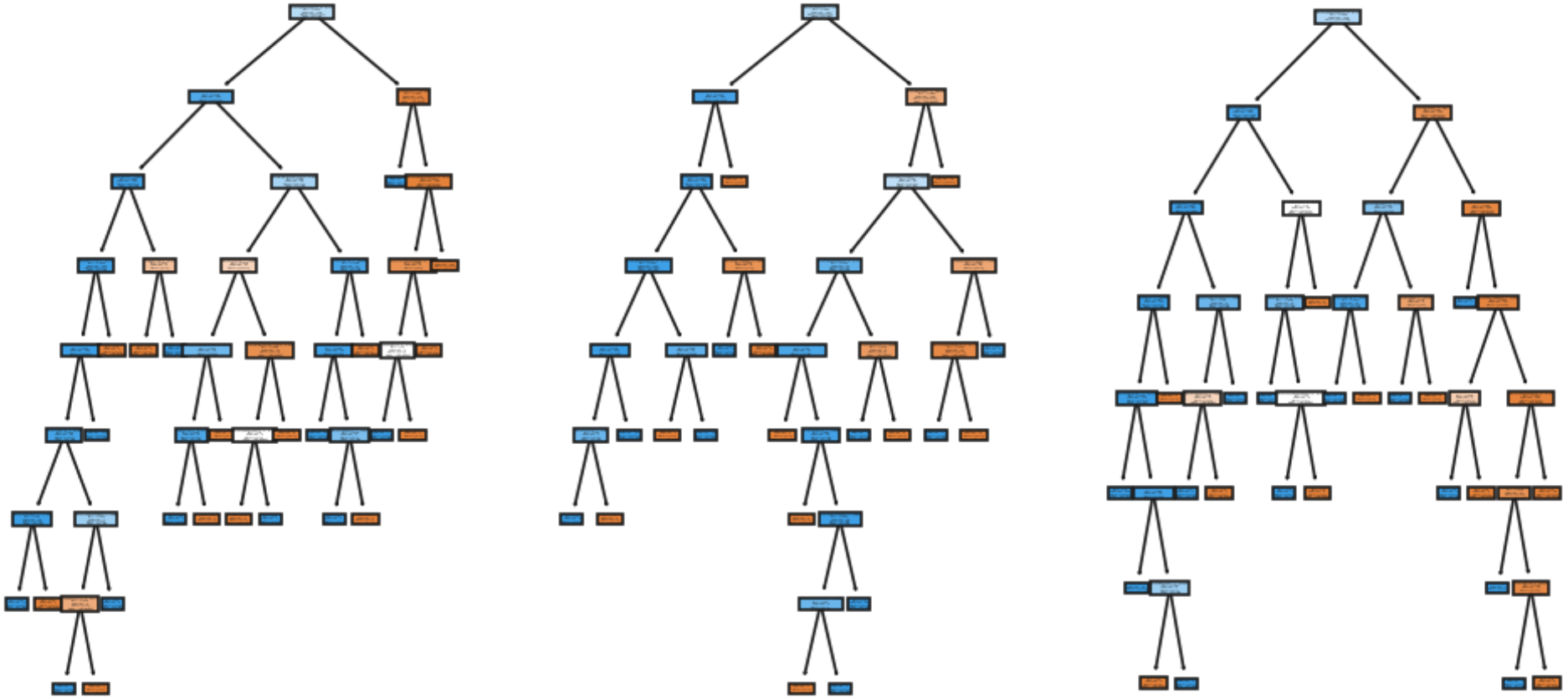
$$RFfi_i = \frac{\sum_j^m normfi_{i,j}}{m}$$

Figure 25: Average Feature Importances for Bagged Trees and a Random Forest



Extra

As a further example, we could look at the trees in our forest directly and see they are quite different.



Extra[8]

Earlier random decision forests by Tin Kam Ho[10] used the *"random subspace method"*, where each tree got a random subset of features.

> “The essence of the method is to build multiple trees in randomly selected subspaces of the feature space.”

However, a few years later, Leo Breiman[11] described the procedure of selecting different subsets of features for each node; so each tree was given the full set of features. This formulation is now the typical *"random forest"* algorithm

> ". . . random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on."

Hyperparameters

Important parameters to adjust are[13]:

- `n_estimators` : Larger is generally better as averaging more trees will typically yield a more robust ensemble by reducing overfitting.
- `max_features` : Determines how random each tree is so smaller number of features at each split reduces overfitting.
- `max_samples` : Sample size of the bootstrap sample, also reduces overfitting.
- pre-pruning options (`max_depth` , `max_leaf_nodes`): more important for single trees, but can improve performance, reduce space, and time requirements.

Notes

- Typically you want to use as many estimators as you have time and memory for training.
  - Start with $p \times 10$ trees and adjust as necessary[14].
- A good rule of thumb for `max_features` default values are `max_features=sqrt(n_features)` for classification and `max_features=n_features` for regression[13].
  - Start with a few evenly spaced values across the range 2–p[14].
- _"Segal (2004) showed that if your data has many noisy predictors and higher [`max_features`] values are performing best, then performance may improve by increasing node size (i.e., decreasing tree depth and complexity). Moreover, if computation time is a concern then you can often decrease run time substantially by increasing the node size and have only marginal impacts to your error estimate..."_[14]
- small bootstrap samples do tend to produce worse models[4]
- For a more thorough discussion of forest hyperparameters, see: Probst, Philipp, Bernd Bischl, and Anne-Laure Boulesteix. 2018. "Tunability: Importance of Hyperparameters of Machine Learning Algorithms." arXiv Preprint arXiv:1802.09596.

Well in this case theres not much impact on run time, but there often is (see https://bradleyboehmke.github.io/HOML/random-forest.html)

Extremely Randomized Trees[12]

As averaging methods work best when the predictors are as independent as possible[5], we may specifically want our trees in our ensemble to be more independent.

Instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.

Notes

- An extratree is similar to a tree classifier except it more randomized and therefore produces more complex trees.
- When used in an ensemble, this usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias[5].
- As we can see how just one tree looks, this is a very complex model!

Extra

Lets look at decision boundaries created by the different trees to demonstrate generalisation performance.

Associated Exercises

Now might be a good time to try exercises 7-10.

References

1. James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning. Vol. 112. New York: springer, 2013.
2. Gorman KB, Williams TD, Fraser WR (2014). Ecological sexual dimorphism and environmental variability within a community of Antarctic penguins (genus Pygoscelis). PLoS ONE 9(3):e90081. https://doi.org/10.1371/journal.pone.0090081
3. Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
4. Raschka, Sebastian, and Vahid Mirjalili. Python Machine Learning, 2nd Ed. Packt Publishing, 2017.

5. https://scikit-learn.org/stable/modules/ensemble.html
6. Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.
7. Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. Machine learning, 36(1-2), 85-103.
8. https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L07/07-ensembles__notes.pdf
9. L. Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001
10. Tin Kam Ho. “Random decision forests”. In: Document analysis and recognition, 1995., proceedings of the third international conference on. Vol. 1. IEEE. 1995, pp. 278–282.
11. Leo Breiman. “Random forests”. In: Machine learning 45.1 (2001), pp. 5–32.
12. Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63(1), 3-42.
13. Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
14. https://bradleyboehmke.github.io/HOML/random-forest.html