Introduction to Random Forests: Takeaways

Syntax

- Instantiating the RandomForestClassifier:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=5, random_state=1, min_samples_leaf=2)
```

- Predicting the probability that a given class is correct for a row:

```
RandomForestClassifier.predict_proba()
```

- Computing the area under the curve:

```
print(roc_auc_score(test["high_income"], predictions))
```

- Introducing variation through bagging:

```
bag_proportion = .6
predictions = []
for i in range(tree_count):
    bag = train.sample(frac=bag_proportion, replace=True, random_state=i)
    clf = DecisionTreeClassifier(random_state=1, min_samples_leaf=2)
    clf.fit(bag[columns], bag["high_income"])
    predictions.append(clf.predict_proba(test[columns])[:,1])
combined = numpy.sum(predictions, axis=0) / 10
rounded = numpy.round(combined)
```

Concepts

- The random forest algorithm is a powerful tool to reduce overfitting in decision trees.
- Random forest is an ensemble algorithm that combines the predictions of multiple decision trees to create a more accurate final prediction.
- There are many methods to get from the output of multiple models to a final vector of predictions. One method is majority voting. In majority voting, each decision tree classifier gets a "vote" and the most commonly voted value for each row "wins."
- To get ensemble predictions, use the `predict_proba` method on the classifiers to generate probabilities, take the mean of the probabilities for each row, and then round the result.
- The more dissimilar the models we use to construct an ensemble are, the stronger their combined predictions will be. For example, ensembling a decision tree and a logistic regression model will result in stronger predictions than ensembling two decision trees with similar parameters. However, ensembling similar models will result in a negligible boost in the accuracy of the model.
- Variation in the random forest will ensure each decision tree is constructed slightly differently and will make different predictions as a result. Bagging and random forest subsets are two main ways to introduce variation in a random forest.
- With bagging, we train each tree on a random sample of the data or "bag". When doing this, we perform sampling with replacement, which means that each row may appear in the "bag"

multiple times. With random forest subsets, however, only a constrained set of features that is selected randomly will be used to introduce variation into the trees.

- `RandomForestClassifier` has an `n_estimators` parameter that allows you to indicate how many trees to build. While adding more trees usually improves accuracy, it also increases the overall time the model takes to train. The class also includes the `bootstrap` parameter which defaults to `True`. "Bootstrap aggregation" is another name for bagging.
- `RandomForestClassifier` has a similar interface to `DecisionTreeClassifier` and we can use the `fit()` and `predict()` methods to train and make predictions.
- The main strengths of a random forest are:
    - Very accurate predictions: Random forests achieve near state-of-the-art performance on many machine learning tasks.
    - Resistance to overfitting: Due to their construction, random forests are fairly resistant to overfitting.
- The main weaknesses of using a random forest are:
    - They're difficult to interpret: Since we've averaging the results of many trees, it can be hard to figure out why a random forest is making predictions the way it is.
    - They take longer to create: Making two trees takes twice as long as making one, making three trees takes three times as long, and so on.

Resources

- Majority Voting
- RandomForestClassifier documentation