Random forests

Paulo S. A. Sousa

2023-05-29

Abstract

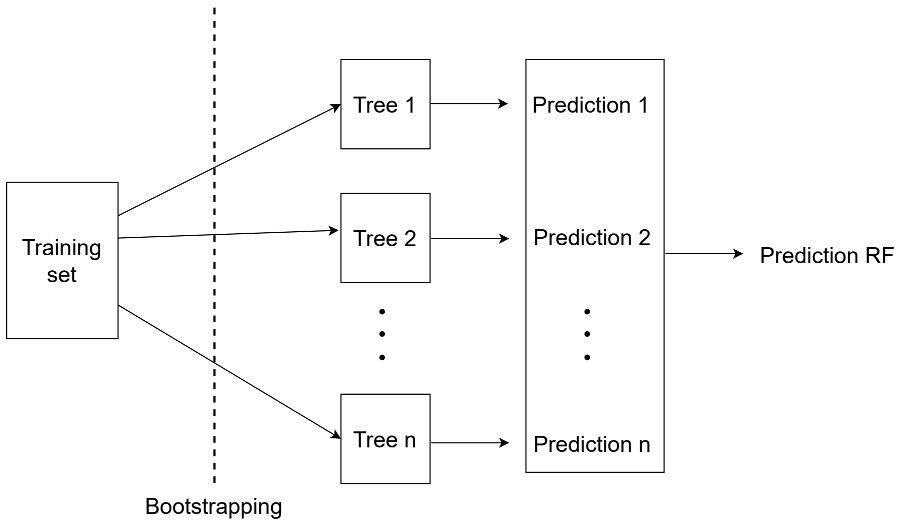
Random forests are introduced in the context of classification and regression.

1 Introduction

One of the most productive ideas in machine learning has been the idea of the creation of a meta-model, by using an ensemble of classic models. This simple idea has led to powerful models. Random forest is a prominent example: It uses an ensemble of trees and is very strong, especially in classification problems.

2 Algorithm

The algorithm of random forests is schematized in the following figure:



It works by constructing a multitude of decision trees during the training phase and outputs the mode of the classes (classification) or the mean prediction (regression) of individual trees. The key concept behind random forests is that the combination of multiple trees reduces overfitting and improves the overall accuracy and stability of the model.

Random Forests use a technique called bootstrapping to create several different datasets from the original dataset. It does this by sampling with replacement, meaning some data points might be repeated in a single bootstrap dataset, while others might be excluded.

For each bootstrap dataset, a decision tree is created. During the construction of each decision tree, a random subset of predictors is selected at each node to determine the best split. This random feature selection introduces diversity among the trees and helps in reducing overfitting.

Once all the decision trees have been constructed, the final prediction is made by aggregating the predictions of each tree. In the case of classification, this is usually done by taking a majority vote of the classes predicted by individual trees. In the case of regression, it's done by averaging the predicted values.

2.1 Hyperparameters

As already discussed, the optimal values for the hyperparameters can be determined by grid search and cross-validation. In the case of random forests, the hyperparameters, beyond `ccp_alpha` (the tree complexity parameter), are:

- `n_estimators:` This hyperparameter controls the number of decision trees to be used in the forest. Increasing the number of trees can improve the performance of the model, but also increases the computational cost.
- `max_depth:` This hyperparameter controls the maximum depth of each decision tree in the forest. Increasing the depth can improve the performance of the model, but also increases the risk of overfitting.
- `min_samples_split:` This hyperparameter controls the minimum number of samples required to split an internal node. Increasing this parameter can help prevent overfitting.
- `min_samples_leaf:` This hyperparameter controls the minimum number of samples required to be at a leaf node. Increasing this parameter can help prevent overfitting.
- `max_features:` This hyperparameter controls the maximum number of features to consider when looking for the best split. Increasing this parameter can improve the performance of the model, but also increases the risk of overfitting.
- `bootstrap:` This hyperparameter controls whether or not to sample the dataset with replacement. Setting this parameter to `True` (the default value) means that the algorithm will randomly sample the data with replacement while setting it to `False` means that the algorithm will use the entire dataset for each tree.

3 Implementation in Python

To use random forests in Python, we need to import `RandomForestClassifier` and `RandomForestRegressor` from `sklearn`:

```python
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.ensemble import RandomForestRegressor
```

For imbalanced classification problems, we can use the parameter `class_weight='balanced'`, likewise as we did in the case of classification trees.