TASK

Supervised Learning – Random Forests

Visit our website

Introduction

WELCOME TO THE SUPERVISED LEARNING – RANDOM FORESTS TASK!

This task introduces ensemble methods. The predictive power of trees can be amplified greatly with ensemble methods. We also introduce bootstrapping, bagging and boosting, and show you how these methods can be applied in simple Python code.

ENSEMBLES

Decision trees are easy to understand, apply, interpret, and visualise. But they are not very **robust**. In this context, trees that are not robust means that small perturbations in the training data can give rise to substantially different predictions at test time. Another way to express this problem is to say that the predictions of decision trees have a very high **variance**. As discussed before, we want our models to capture general patterns, not be so dependent on the data they have trained on, that a bit of noise or a different sample changes predictions entirely.

Ensemble techniques tackle this problem by treating a tree's predictions as votes towards labels. Rather than trying to create one classifier that makes perfect predictions, ensemble methods aggregate the predictions of multiple classifiers into a single, improved prediction. Aside from random forests, ensemble methods can and do get applied to methods other than decision trees, but trees can benefit in particular due to how flexible they are.

BOOTSTRAPPING

The principle behind ensemble methods is that different classifiers identify different patterns. Some of these are noise and others are useful. Training many different classifiers and combining the result will help identify and get rid of some noise and retain only the best patterns. The most straightforward way to leverage this fact would be to train multiple models on the same training data and take an average, but this is not very effective. One step up in sophistication is bootstrapping.

In bootstrapping, samples are drawn from a data set repeatedly. If the training portion on a dataset is size N, the samples are each a size smaller than N. The samples are drawn with replacement, meaning an instance can occur in more than one sample. The collection of samples is called the bootstrap data set.

Below is a function that implements sampling with replacement. The function selects rows from the data randomly until the sample size has been reached.

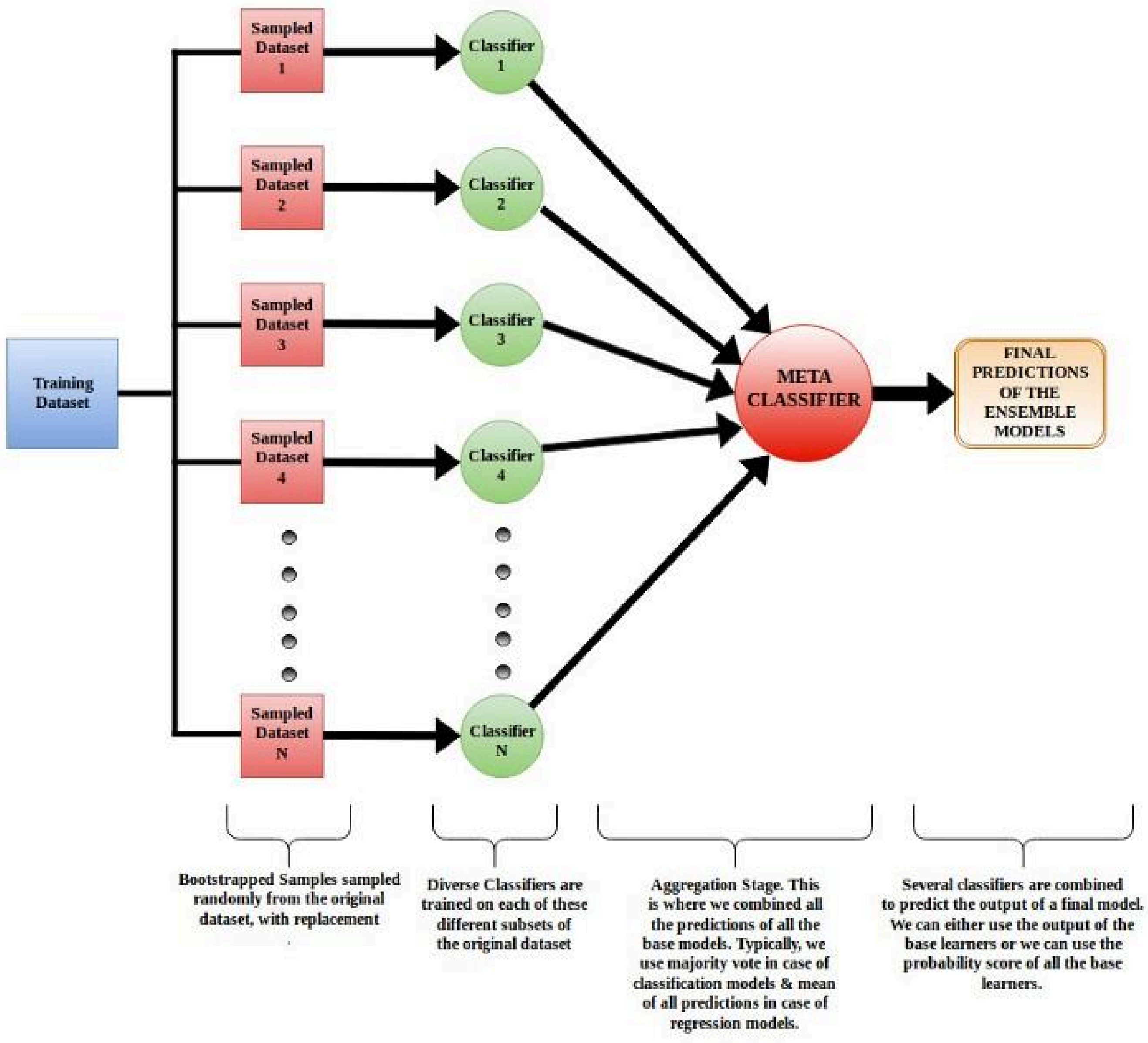
```python
def subsample(dataset, ratio=0.7):

    sample = list()
    n_sample = round(len(dataset) * ratio)
    while len(sample) < n_sample:
        index = randrange(len(dataset))
        sample.append(dataset[index])

    return sample
```

Out-of-bag error estimation

To estimate the test error of a bagged model (an ensemble model created through bootstrap sampling, where multiple models are trained on different subsets of the data. It aims to capture diverse variations in the data set for improved predictive performance) without using a test set, we make use of the fact that each sample is a subset of the data. If each sample is a subset of size $n < N$, then there are $N - n$ instances the model trained on that sample has not seen. This unseen set can be used to estimate the error of the models.



Training Dataset
Sampled Dataset 1
Sampled Dataset 2
Sampled Dataset 3
Sampled Dataset 4
Sampled Dataset N
Classifier 1
Classifier 2
Classifier 3
Classifier 4
Classifier N
META CLASSIFIER
FINAL PREDICTIONS OF THE ENSEMBLE MODELS
Bootstrapped Samples sampled randomly from the original dataset, with replacement
Diverse Classifiers are trained on each of these different subsets of the original dataset
Aggregation Stage. This is where we combined all the predictions of all the base models. Typically, we use majority vote in case of classification models & mean of all predictions in case of regression models.
Several classifiers are combined to predict the output of a final model. We can either use the output of the base learners or we can use the probability score of all the base learners.

THE DIFFERENT STAGES OF A BAGGING ALGORITHM

BAGGING

The next step is to aggregate predictions. Bagging (short for bootstrap aggregation) refers to the step in which the predictions of models fitted to the sample are combined into a final prediction. In essence, all of the models vote on a prediction as the final outcome of a classification. In regression, the results of all models are averaged.

The performance of the ensemble is then tested on some held-out data.

Random forests

Random forests provide an improvement over bagged trees by decorrelating the trees. As with regular bagged trees, a number of decision trees are built using bootstrapped training samples. However, in regular bagging, it is possible that one variable that is a particularly strong variable for predicting splitting, which is likely to be used early on by each tree, results in similar trees.

With random forests, this behaviour is prevented by only allowing one of a random subset of independent variables to be considered as a split candidate from the full set of variables. This causes the trees to be more dissimilar, as the strong splitting candidate variable is not always present in the random subset, hence we can think of this process as decorrelating the trees. Averaging many uncorrelated quantities leads to a larger reduction in variance than averaging many correlated quantities. This makes the average of the resulting trees less variable and thus more reliable.

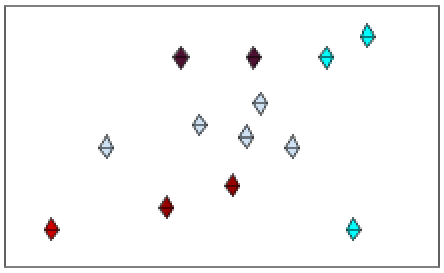
BOOSTING

Bagging lets models run in parallel on different samples without communicating with each other. An alternative approach is to run models on different samples but to do so sequentially in such a way that models later in the sequence have some information about how previous models fared at the task. Boosting attempts this by incorporating the residuals (another name for the error) of earlier models into the data provided to later models. In principle, this causes each model to correct the mistakes of a previous model.

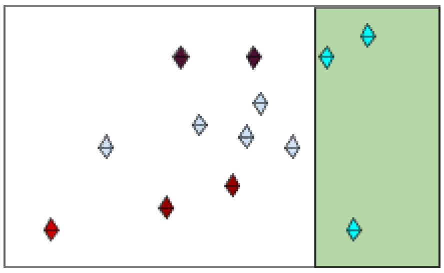
The steps for boosting are as follows:

1. A sample is drawn from a data set.

2. An initial model is trained on this sample, with each instance having equal weighting.

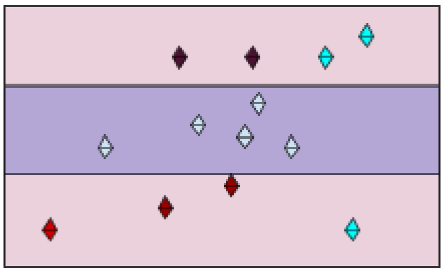


3. The initial model generates predictions for the whole data set.

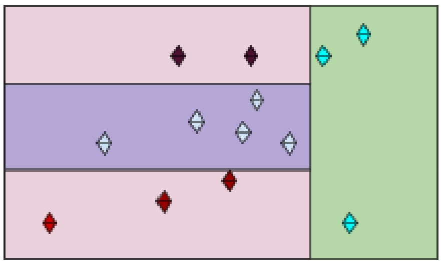


4. The error of the initial model is computed and the weighting of all incorrectly predicted instances is increased.

5. A second model is trained on a sample of this weighted version of the data set.



6. In the end, all models are weighted and the mean or majority of all models' predictions are taken.



How this approach works is that each model focuses on the errors of the previous model. The subsequent model may make new mistakes that the first one did not, as a result of the weighting, but that is okay. Each model is a specialist on some subset of the data: in itself quite weak, but a "boost" to the performance of the ensemble.

As a result, boosting algorithms avoid a scenario where particularly difficult instances are misclassified by every single model. Furthermore, if a vanilla or bagged model performs rather poorly, a boosted model may improve performance.

On the downside, the outcome of boosting depends heavily on the order in which the models were placed, and there are no methods for determining which order is best besides trial and error. Secondly, unlike bagging, boosting does not avoid overfitting, it only lowers classifier variance.

Instructions

Read the Jupyter Notebook in this task's folder before attempting the compulsory task.

Practical Task

Follow these steps:

- In this task, we will continue with the **Decision_Trees.ipynb** Notebook created in the previous task.
- Create a *bagged*, *random forest*, and *boosted tree* for the Titanic data set in the same way that you created a regular classification tree.
- From the random forest model, determine which of the features is the one that contributes the most to predicting whether a passenger survives or not.
- Pick one of these methods, and tune the parameters *n_estimators* and *max_depth*.
- Report the accuracy of all models and report which model performed the best, including the values for *n_estimators* and *max_depth* that the best model had.