**Bagging**

Welcome back! In this video, you'll be introduced to an ensemble method known as Bootstrap aggregation or Bagging.

**2. Ensemble Methods**

In the last chapter, you learned that the Voting Classifier is an ensemble of models that are fit to the same training set using different algorithms. You also saw that the final predictions were obtained by majority voting. In Bagging, the ensemble is formed by models that use the same training algorithm. However, these models are not trained on the entire training set. Instead, each model is trained on a different subset of the data.

**3. Bagging**

In fact, bagging stands for bootstrap aggregation. Its name refers to the fact that it uses a technique known as the bootstrap. Overall, Bagging has the effect of reducing the variance of individual models in the ensemble.

**4. Bootstrap**

Let's first try to understand what the bootstrap method is. Consider the case where you have 3 balls labeled A, B, and C. A bootstrap sample is a sample drawn from this with replacement. By replacement, we mean that any ball can be drawn many times. For example, in the first bootstrap sample shown in the diagram here, B was drawn 3 times in a raw. In the second bootstrap sample, A was drawn two times while B was drawn once, and so on. You may now ask how bootstraping can help us produce an ensemble.

**5. Bagging: Training**

In fact, in the training phase, bagging consists of drawing N different bootstrap samples from the training set. As shown in the diagram here, each of these bootstrap samples are then used to train N models that use the same algorithm .

**6. Bagging: Prediction**

When a new instance is fed to the different models forming the bagging ensemble, each model outputs its prediction. The meta model collects these predictions and outputs a final prediction depending on the nature of the problem.

**7. Bagging: Classification & Regression**

In classification, the final prediction is obtained by majority voting. The corresponding classifier in scikit-learn is BaggingClassifier. In regression, the final prediction is the average of the predictions made by the individual models forming the ensemble. The corresponding regressor in scikit-learn is BaggingRegressor.

**8. Bagging Classifier in sklearn (Breast-Cancer dataset)**

Great! Now that you understand how Bagging works, let's train a BaggingClassifier in scikit-learn on the breast cancer dataset. Note that the dataset is already loaded. First import BaggingClassifier, DecisionTreeClassifier, accuracy\_score and train\_test\_split and then split the data into 70%-train and 30%-test as shown here.

**9. Bagging Classifier in sklearn (Breast-Cancer dataset)**

Now, instantiate a classification tree dt with the parameters max\_depth set to 4 and min\_samples\_leaf set to 0-dot-16. You can then instantiate a BaggingClassifier bc that consists of 300 classification trees dt. This can be done by setting the parameters base\_estimator to dt and n\_estimators to 300. In addition, set the paramter n\_jobs to -1 so that all CPU cores are used in computation. Once you are done, fit bc to the training set, predict the test set labels and finally, evaluate the test set accuracy. The output shows that a BaggingClassifier achieves a test set accuracy of 93-dot-6%. Training the classification tree dt, which is the base estimator here, to the same training set would lead to a test set accuracy of 88-dot-9%. The result highlights how bagging outperforms the base estimator dt.

**Out Of Bag Evaluation**

You will now learn about Out-of-bag evaluation.

**2. Bagging**

Recall that in bagging, some instances may be sampled several times for one model. On the other hand, other instance may not be sampled at all.

**3. Out Of Bag (OOB) instances**

On average, for each model, 63% of the training instances are sampled. The remaining 37% that are not sampled constitute what is known as the Out-of-bag or OOB instances. Since OOB instances are not seen by a model during training, these can be used to estimate the performance of the ensemble without the need for cross-validation. This technique is known as OOB-evaluation.

**4. OOB Evaluation**

To understand OOB-evaluation more concretely, take a look at this diagram. Here, for each model, the bootstrap instances are shown in blue while the OOB-instances are shown in red. Each of the N models constituting the ensemble is then trained on its corresponding bootstrap samples and evaluated on the OOB instances. This leads to the obtainment of N OOB scores labeled OOB1 to OOBN. The OOB-score of the bagging ensemble is evaluated as the average of these N OOB scores as shown by the formula on top.

**5. OOB Evaluation in sklearn (Breast Cancer Dataset)**

Alright! Now it's time to see OOB-evaluation in action. Again, we'll be classifying cancerous cells as malignant or benign from the breast cancer dataset which is already loaded. After importing BaggingClassifier, DecisionTreeClassifier, accuracy\_score and train\_test\_split, split the dataset in a stratified way into 70%-train and 30%-test by setting the parameter stratify to y.

**6. OOB Evaluation in sklearn (Breast Cancer Dataset)**

Now, first instantiate a classification tree dt with a maximum-depth of 4 and a minimum percentage of samples per leaf equal to 16%. Then instantiate a BaggingClassifier called bc that consists of 300 classification trees. This can be done by setting the parameters n\_estimators to 300 and base\_estimator to dt. Importantly, set the parameter oob\_score to True in order to evaluate the OOB-accuracy of bc after training. Note that in scikit-learn, the OOB-score corresponds to the accuracy for classifiers and the r-squared score for regressors. Now fit bc to the training set and predict the test set labels.

**7. OOB Evaluation in sklearn (Breast Cancer Dataset)**

Assign the test set accuracy to test\_accuracy. Finally, evaluate the OOB-accuracy of bc by extracting the attribute oob\_score\_ from the trained instance; assign the result to oob\_accuracy and print out the results. The test-set accuracy is about 93.6% and the OOB-accuracy is about 92.5%. The two obtained accuracies are pretty close though not exactly equal. These results highlight how OOB-evaluation can be an efficient technique to obtain a performance estimate of a bagged-ensemble on unseen data without performing cross-validation.

**Random Forests**

You will now learn about another ensemble learning method known as Random Forests.

**2. Bagging**

Recall that in bagging the base estimator could be any model including a decision tree, logistic regression or even a neural network. Each estimator is trained on a distinct bootstrap sample drawn from the training set using all available features.

**3. Further Diversity with Random Forests**

Random Forests is an ensemble method that uses a decision tree as a base estimator. In Random Forests, each estimator is trained on a different bootstrap sample having the same size as the training set. Random forests introduces further randomization than bagging when training each of the base estimators. When each tree is trained, only d features can be sampled at each node without replacement, where d is a number smaller than the total number of features.

**4. Random Forests: Training**

The diagram here shows the training procedure for random forests. Notice how each tree forming the ensemble is trained on a different bootstrap sample from the training set. In addition, when a tree is trained, at each node, only d features are sampled from all features without replacement. The node is then split using the sampled feature that maximizes information gain. In scikit-learn d defaults to the square-root of the number of features. For example, if there are 100 features, only 10 features are sampled at each node.

**5. Random Forests: Prediction**

Once trained, predictions can be made on new instances. When a new instance is fed to the different base estimators, each of them outputs a prediction. The predictions are then collected by the random forests meta-classifier and a final prediction is made depending on the nature of the problem.

**6. Random Forests: Classification & Regression**

For classification, the final prediction is made by majority voting. The corresponding scikit-learn class is RandomForestClassifier. For regression, the final prediction is the average of all the labels predicted by the base estimators. The corresponding scikit-learn class is RandomForestRegressor. In general, Random Forests achieves a lower variance than individual trees.

**7. Random Forests Regressor in sklearn (auto dataset)**

Alright, now it's time to put all this into practice. Here, you'll train a random forests regressor to the auto-dataset which you were introduced to in previous chapters. Note that the dataset is already loaded. After importing RandomForestRegressor, train\_test\_split and mean\_squared\_error as MSE, split the dataset into 70%-train and 30%-test as shown here.

**8. Random Forests Regressor in sklearn (auto dataset)**

Then instantiate a RandomForestRegressor consisting of 400 regression trees. This can be done by setting n\_estimators to 400. In addition, set min\_samples\_leaf to 0-dot-12 so that each leaf contains at least 12% of the data used in training. You can now fit rf to the training set and predict the test set labels. Finally, print the test set RMSE. The result shows that rf achieves a test set RMSE of 3-dot-98; this error is smaller than that achieved by a single regression tree which is 4-dot-43.

**9. Feature Importance**

When a tree based method is trained, the predictive power of a feature or its importance can be assessed. In scikit-learn, feature importance is assessed by measuring how much the tree nodes use a particular feature to reduce impurity. Note that the importance of a feature is expressed as a percentage indicating the weight of that feature in training and prediction. Once you train a tree-based model in scikit-learn, the features importances can be accessed by extracting the feature\_importance\_ attribute from the model.

**10. Feature Importance in sklearn**

To visualize the importance of features as assessed by rf, you can create a pandas series of the features importances as shown here and then sort this series and make a horiztonal-barplot.

**11. Feature Importance in sklearn**

The results show that, according to rf, displ, size, weight and hp are the most predictive features.