**Tuning a CART's hyperparameters**

To obtain a better performance, the hyperparameters of a machine learning should be tuned.

**2. Hyperparameters**

Machine learning models are characterized by parameters and hyperparameters. Parameters are learned from data through training; examples of parameters include the split-feature and the split-point of a node in a CART. Hyperparameters are not learned from data; they should be set prior to training. Examples of hyperparameters include the maximum-depth and the splitting-criterion of a CART.

**3. What is hyperparameter tuning?**

Hyperparameter tuning consists of searching for the set of optimal hyperparameters for the learning algorithm. The solution involves finding the set of optimal hyperparameters yielding an optimal model. The optimal model yields an optimal score. The score function measures the agreement between true labels and a model's predictions. In sklearn, it defaults to accuracy for classifiers and r-squared for regressors. A model's generalization performance is evaluated using cross-validation.

**4. Why tune hyperparameters?**

A legitimate question that you may ask is: why bother tuning hyperparameters? Well, in scikit-learn, a model's default hyperparameters are not optimal for all problems. Hyperparameters should be tuned to obtain the best model performance.

**5. Approaches to hyperparameter tuning**

Now there are many approaches for hyperparameter tuning including: grid-search, random-search, and so on. In this course, we'll only be exploring the method of grid-search.

**6. Grid search cross validation**

In grid-search cross-validation, first you manually set a grid of discrete hyperparameter values. Then, you pick a metric for scoring model performance and you search exhaustively through the grid. For each set of hyperparameters, you evaluate each model's score. The optimal hyperparameters are those for which the model achieves the best cross-validation score. Note that grid-search suffers from the curse of dimensionality, i-dot-e-dot, the bigger the grid, the longer it takes to find the solution.

**7. Grid search cross validation: example**

Let's walk through a concrete example to understand this procedure. Consider the case of a CART where you search through the two-dimensional hyperparameter grid shown here. The dimensions correspond to the CART's maximum-depth and the minimum-percentage of samples per leaf. For each combination of hyperparameters, the cross-validation score is evaluated using k-fold CV for example. Finally, the optimal hyperparameters correspond to the model achieving the best cross-validation score.

**8. Inspecting the hyperparameters of a CART in sklearn**

Let's now see how we can inspect the hyperparameters of a CART in scikit-learn. You can first instantiate a DecisionTreeClassifier dt as shown here.

**9. Inspecting the hyperparameters of a CART in sklearn**

Then, call dt's -dot-get\_params() method. This prints out a dictionary where the keys are the hyperparameter names. In the following, we'll only be optimizing max\_depth, max\_features and min\_samples\_leaf. Note that max\_features is the number of features to consider when looking for the best split. When it's a float, it is interpreted as a percentage. You can learn more about these hyperparameters by consulting scikit-learn's documentation.

**10. Grid search CV in sklearn (Breast Cancer dataset)**

Let's now tune dt on the wisconsin breast cancer dataset which is already loaded and split into 80%-train and 20%-test. First, import GridSearchCV from sklearn-dot-model\_selection. Then, define a dictionary called params\_dt containing the names of the hyperparameters to tune as keys and lists of hyperparameter-values as values. Once done, instantiate a GridSearchCV object grid\_dt by passing dt as an estimator and params\_dt as param\_grid. Also set scoring to accuracy and cv to 10 in order to use 10-fold stratified cross-validation for model evaluation. Finally, fit grid\_dt to the training set.

**11. Extracting the best hyperparameters**

After training grid\_dt, the best set of hyperparameter-values can be extracted from the attribute -dot-best\_params\_ of grid\_dt. Also, the best cross validation accuracy can be accessed through grid\_dt's -dot-best\_score\_ attribute.

**12. Extracting the best estimator**

Similarly, the best-model can be extracted using the -dot-best\_estimator attribute. Note that this model is fitted on the whole training set because the refit parameter of GridSearchCV is set to True by default. Finally, you can evaluate this model's test set accuracy using the score method. The result is about 94-dot-7% while the score of an untuned CART is of 93%.

**Tuning an RF's Hyperparameters**

Let's now turn to a case where we tune the hyperparameters of Random Forests which is an ensemble method.

**2. Random Forests Hyperparameters**

In addition to the hyperparameters of the CARTs forming random forests, the ensemble itself is characterized by other hyperparameters such as the number of estimators, whether it uses bootstraping or not and so on.

**3. Tuning is expensive**

As a note, hyperparameter tuning is computationally expensive and may sometimes lead only to very slight improvement of a model's performance. For this reason, it is desired to weigh the impact of tuning on the pipeline of your data analysis project as a whole in order to understand if it is worth pursuing.

**4. Inspecting RF Hyperparameters in sklearn**

To inspect the hyperparameters of a RandomForestRegressor, first, import RandomForestRegressor from sklearn.ensemble and then instantiate a RandomForestRegressor rf as shown here.

**5. Inspecting RF Hyperparameters in sklearn**

The hyperparameters of rf along with their default values can be accessed by calling rf's dot-get\_params() method. In the following, we'll be optimizing n\_estimators, max\_depth, min\_samples\_leaf and max\_features. You can learn more about these hyperparameters by consulting scikit-learn's documentation.

**6. GridSearchCV in sklearn (auto dataset)**

We'll perform grid-search cross-validation on the auto-dataset which is already loaded and split into 80%-train and 20%-test. First import mean\_squared\_error as MSE from sklearn.metrics and GridSearchCV from sklearn.model\_selection. Then, define a dictionary called params\_rf containing the grid of hyperparameters. Finally, instantiate a GridSearchCV object called grid\_rf and pass the parameters rf as estimator, params\_rf as param\_grid. Also set cv to 3 to perform 3-fold cross-validation. In addition, set scoring to neg\_mean\_squared\_error in order to use negative mean squared error as a metric. Note that the parameter verbose controls verbosity; the higher its value, the more messages are printed during fitting.

**7. Searching for the best hyperparameters**

You can now fit grid\_rf to the training set as shown here. The output shows messages related to grid fitting as well as the obtained optimal model.

**8. Extracting the best hyperparameters**

You can extract rf's best hyperparameters by getting the attribute best\_params\_ from grid\_rf. The results are shown here.

**9. Evaluating the best model performance**

You can also extract the best model from rf. This enables you to predict the test set labels and evaluate the test-set RMSE. The output shows a result of 3-dot-89. If you would have trained an untuned model, the RMSE would be 3-dot-98