cross_validation_validation_curve

May 29, 2021

1 Overfit-generalization-underfit

In the previous notebook, we presented the general cross-validation framework and how it helps us quantify the training and testing errors as well as their fluctuations.

In this notebook, we will put these two errors into perspective and show how they can help us know if our model generalizes, overfit, or underfit.

Let's first load the data and create the same model as in the previous notebook.

[1]:
```
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing(as_frame=True)
data, target = housing.data, housing.target
target *= 100  # rescale the target in k$
```

Note

If you want a deeper overview regarding this dataset, you can refer to the Appendix - Datasets description section at the end of this MOOC.

[2]:
```
from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor()
```

1.1 Overfitting vs. underfitting

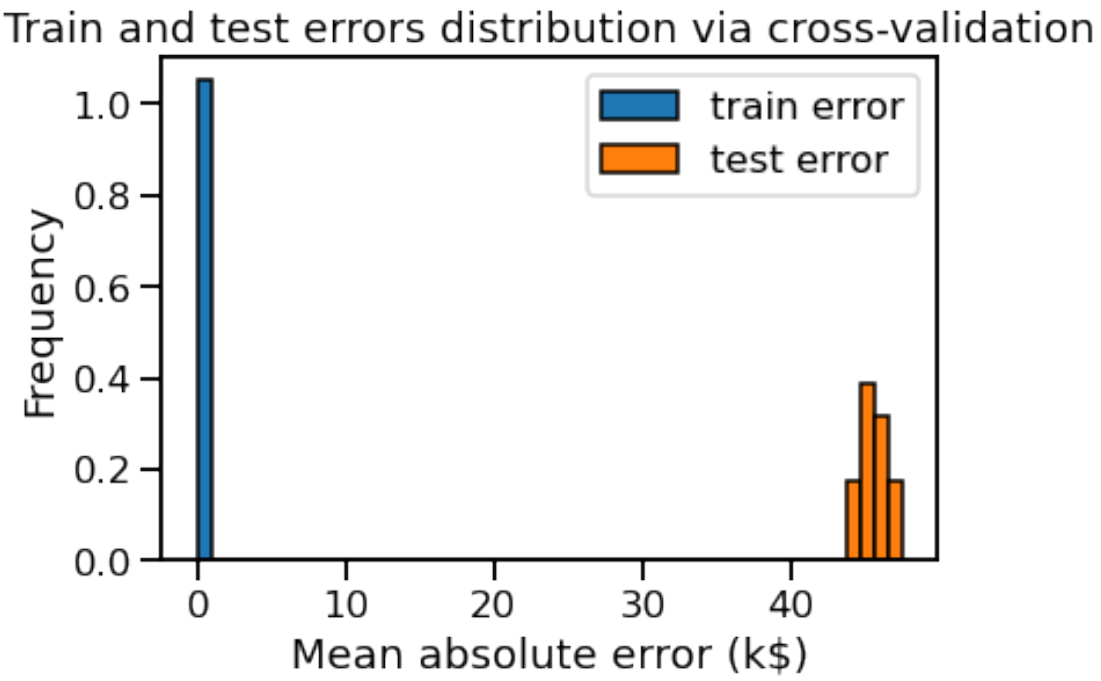
To better understand the statistical performance of our model and maybe find insights on how to improve it, we will compare the testing error with the training error. Thus, we need to compute the error on the training set, which is possible using the `cross_validate` function.

[3]:
```
import pandas as pd
from sklearn.model_selection import cross_validate, ShuffleSplit

cv = ShuffleSplit(n_splits=30, test_size=0.2)
cv_results = cross_validate(regressor, data, target,
                            cv=cv, scoring="neg_mean_absolute_error",
                            return_train_score=True, n_jobs=2)
cv_results = pd.DataFrame(cv_results)
```

The cross-validation used the negative mean absolute error. We transform the negative mean absolute error into a positive mean absolute error.

[4]:
```
scores = pd.DataFrame()
scores[["train error", "test error"]] = -cv_results[
    ["train_score", "test_score"]]
```

[5]:
```
import matplotlib.pyplot as plt

scores.plot.hist(bins=50, edgecolor="black", density=True)
plt.xlabel("Mean absolute error (k$)")
_ = plt.title("Train and test errors distribution via cross-validation")
```



By plotting the distribution of the training and testing errors, we get information about whether our model is over-fitting, under-fitting (or both at the same time).

Here, we observe a **small training error** (actually zero), meaning that the model is **not under-fitting**: it is flexible enough to capture any variations present in the training set.

However the **significantly larger testing error** tells us that the model is **over-fitting**: the model has memorized many variations of the training set that could be considered "noisy" because they do not generalize to help us make good prediction on the test set.

1.2 Validation curve

Some model hyperparameters are usually the key to go from a model that underfits to a model that overfits, hopefully going through a region were we can get a good balance between the two.

We can acquire knowledge by plotting a curve called the validation curve. This curve can also be applied to the above experiment and varies the value of a hyperparameter.

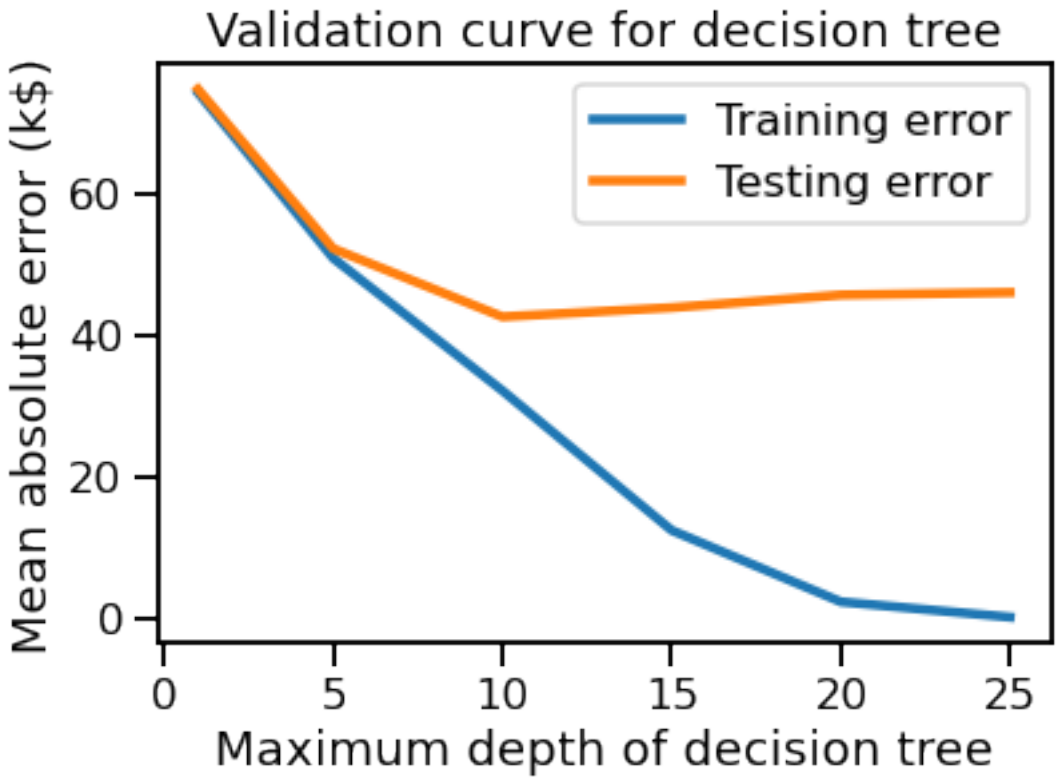
For the decision tree, the `max_depth` parameter is used to control the tradeoff between under-fitting and over-fitting.

[6]:
```
%%time
from sklearn.model_selection import validation_curve

max_depth = [1, 5, 10, 15, 20, 25]
train_scores, test_scores = validation_curve(
    regressor, data, target, param_name="max_depth", param_range=max_depth,
    cv=cv, scoring="neg_mean_absolute_error", n_jobs=2)
train_errors, test_errors = -train_scores, -test_scores
```

```
CPU times: user 166 ms, sys: 31.1 ms, total: 197 ms
Wall time: 9.7 s
```

Now that we collected the results, we will show the validation curve by plotting the training and testing errors (as well as their deviations).

[7]:
```
plt.plot(max_depth, train_errors.mean(axis=1), label="Training error")
plt.plot(max_depth, test_errors.mean(axis=1), label="Testing error")
plt.legend()

plt.xlabel("Maximum depth of decision tree")
plt.ylabel("Mean absolute error (k$)")
_ = plt.title("Validation curve for decision tree")
```



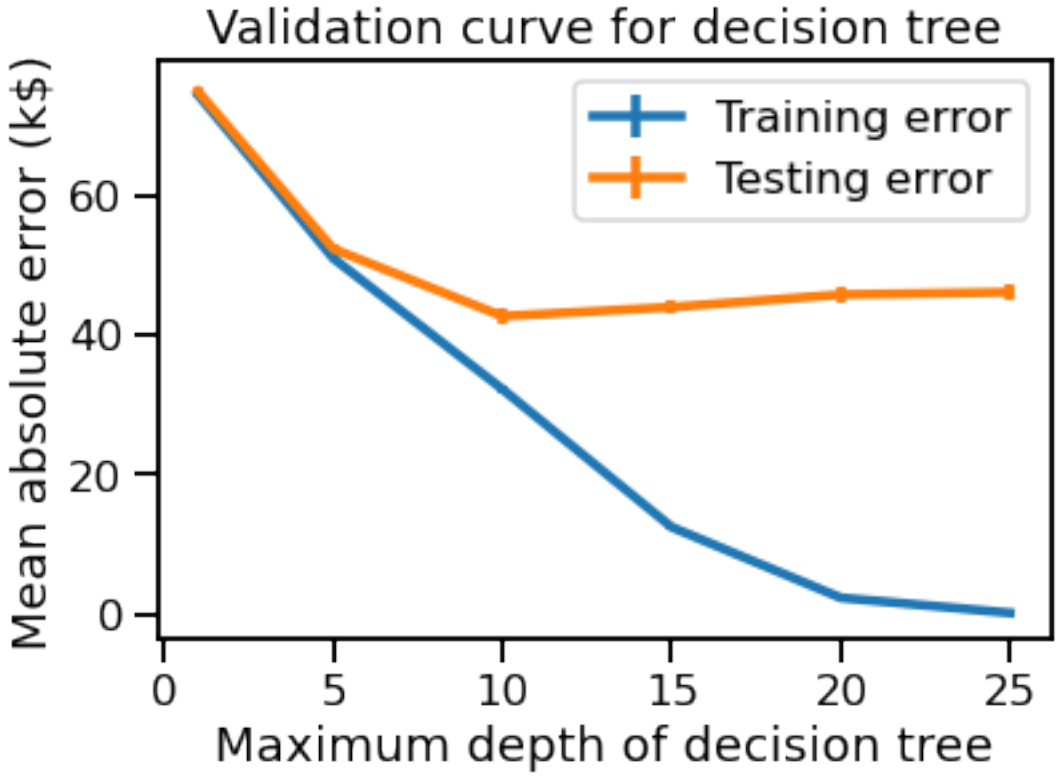
The validation curve can be divided into three areas:

- For `max_depth < 10`, the decision tree underfits. The training error and therefore the testing error are both high. The model is too constrained and cannot capture much of the variability of the target variable.
- The region around `max_depth = 10` corresponds to the parameter for which the decision tree generalizes the best. It is flexible enough to capture a fraction of the variability of the target that generalizes, while not memorizing all of the noise in the target.
- For `max_depth > 10`, the decision tree overfits. The training error becomes very small, while the testing error increases. In this region, the models create decisions specifically for noisy samples harming its ability to generalize to test data.

Note that for `max_depth = 10`, the model overfits a bit as there is a gap between the training error and the testing error. It can also potentially underfit also a bit at the same time, because the training error is still far from zero (more than 30 k\$), meaning that the model might still be too constrained to model interesting parts of the data. However, the testing error is minimal, and this is what really matters. This is the best compromise we could reach by just tuning this parameter.

Be aware that looking at the mean errors is quite limiting. We should also look at the standard deviation to assess the dispersion of the score. We can repeat the same plot as before but this time, we will add some information to show the standard deviation of the errors as well.

[8]:
```
plt.errorbar(max_depth, train_errors.mean(axis=1),
             yerr=train_errors.std(axis=1), label='Training error')
plt.errorbar(max_depth, test_errors.mean(axis=1),
             yerr=test_errors.std(axis=1), label='Testing error')
plt.legend()

plt.xlabel("Maximum depth of decision tree")
plt.ylabel("Mean absolute error (k$)")
_ = plt.title("Validation curve for decision tree")
```



We were lucky that the variance of the errors was small compared to their respective values, and therefore the conclusions above are quite clear. This is not necessarily always the case.

1.3 Summary:

In this notebook, we saw:

- how to identify whether a model is generalizing, overfitting, or underfitting;
- how to check influence of a hyperparameter on the tradeoff underfit/overfit.