

parameter_tuning_manual

May 29, 2021

1 Set and get hyperparameters in scikit-learn

This notebook shows how one can get and set the value of a hyperparameter in a scikit-learn estimator. We recall that hyperparameters refer to the parameter that will control the learning process.

They should not be confused with the fitted parameters, resulting from the training. These fitted parameters are recognizable in scikit-learn because they are spelled with a final underscore `_`, for instance `model.coef_`.

We will start by loading the adult census dataset and only use the numerical feature.

```
[1]: import pandas as pd

adult_census = pd.read_csv("../datasets/adult-census.csv")

target_name = "class"
numerical_columns = [
    "age", "capital-gain", "capital-loss", "hours-per-week"]

target = adult_census[target_name]
data = adult_census[numerical_columns]
```

Our data is only numerical.

```
[2]: data.head()
```

```
[2]:
```

	age	capital-gain	capital-loss	hours-per-week
0	25	0	0	40
1	38	0	0	50
2	28	0	0	40
3	44	7688	0	40
4	18	0	0	30

Let's create a simple predictive model made of a scaler followed by a logistic regression classifier.

As mentioned in previous notebooks, many models, including linear ones, work better if all features have a similar scaling. For this purpose, we use a `StandardScaler`, which transforms the data by rescaling features.

```
[3]: from sklearn.pipeline import Pipeline
      from sklearn.preprocessing import StandardScaler
      from sklearn.linear_model import LogisticRegression

      model = Pipeline(steps=[
          ("preprocessor", StandardScaler()),
          ("classifier", LogisticRegression())
      ])
```

We can evaluate the statistical performance of the model via cross-validation.

```
[4]: from sklearn.model_selection import cross_validate

      cv_results = cross_validate(model, data, target)
      scores = cv_results["test_score"]
      print(f"Accuracy score via cross-validation:\n"
            f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

Accuracy score via cross-validation:
0.800 +/- 0.003

We created a model with the default `C` value that is equal to 1. If we wanted to use a different `C` parameter we could have done so when we created the `LogisticRegression` object with something like `LogisticRegression(C=1e-3)`.

We can also change the parameter of a model after it has been created with the `set_params` method, which is available for all scikit-learn estimators. For example, we can set `C=1e-3`, fit and evaluate the model:

```
[5]: model.set_params(classifier__C=1e-3)
      cv_results = cross_validate(model, data, target)
      scores = cv_results["test_score"]
      print(f"Accuracy score via cross-validation:\n"
            f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

Accuracy score via cross-validation:
0.787 +/- 0.002

When the model of interest is a `Pipeline`, the parameter names are of the form `<model_name>__<parameter_name>` (note the double underscore in the middle). In our case, `classifier` comes from the `Pipeline` definition and `C` is the parameter name of `LogisticRegression`.

In general, you can use the `get_params` method on scikit-learn models to list all the parameters with their values. For example, if you want to get all the parameter names, you can use:

```
[6]: for parameter in model.get_params():
      print(parameter)
```

memory

```

steps
verbose
preprocessor
classifier
preprocessor__copy
preprocessor__with_mean
preprocessor__with_std
classifier__C
classifier__class_weight
classifier__dual
classifier__fit_intercept
classifier__intercept_scaling
classifier__l1_ratio
classifier__max_iter
classifier__multi_class
classifier__n_jobs
classifier__penalty
classifier__random_state
classifier__solver
classifier__tol
classifier__verbose
classifier__warm_start

```

`.get_params()` returns a dict whose keys are the parameter names and whose values are the parameter values. If you want to get the value of a single parameter, for example `classifier__C`, you can use:

```
[7]: model.get_params()['classifier__C']
```

```
[7]: 0.001
```

We can systematically vary the value of `C` to see if there is an optimal value.

```
[8]: for C in [1e-3, 1e-2, 1e-1, 1, 10]:
    model.set_params(classifier__C=C)
    cv_results = cross_validate(model, data, target)
    scores = cv_results["test_score"]
    print(f"Accuracy score via cross-validation with C={C}:\n"
          f"{scores.mean():.3f} +/- {scores.std():.3f}")

```

Accuracy score via cross-validation with `C=0.001`:

0.787 +/- 0.002

Accuracy score via cross-validation with `C=0.01`:

0.799 +/- 0.003

Accuracy score via cross-validation with `C=0.1`:

0.800 +/- 0.003

Accuracy score via cross-validation with `C=1`:

0.800 +/- 0.003

Accuracy score via cross-validation with `C=10`:

0.800 +/- 0.003

We can see that as long as C is high enough, the model seems to perform well.

What we did here is very manual: it involves scanning the values for C and picking the best one manually. In the next lesson, we will see how to do this automatically.

Warning

When we evaluate a family of models on test data and pick the best performer, we can not trust the corresponding prediction accuracy, and we need to apply the selected model to new data. Indeed, the test data has been used to select the model, and it is thus no longer independent from this model.

In this notebook we have seen:

- how to use `get_params` and `set_params` to get the parameters of a model and set them.