Solution for Exercise M1.05

Contents

The goal of this exercise is to evaluate the impact of feature preprocessing on a pipeline that uses a decision-tree-based classifier instead of a logistic regression.

- The first question is to empirically evaluate whether scaling numerical features is helpful or not;
- The second question is to evaluate whether it is empirically better (both from a computational and a statistical perspective) to use integer coded or one-hot encoded categories.

```python
import pandas as pd

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

```python
target_name = "class"
target = adult_census[target_name]
data = adult_census.drop(columns=[target_name, "education-num"])
```

As in the previous notebooks, we use the utility `make_column_selector` to select only columns with a specific data type. Besides, we list in advance all categories for the categorical columns.

```python
from sklearn.compose import make_column_selector as selector

numerical_columns_selector = selector(dtype_exclude=object)
categorical_columns_selector = selector(dtype_include=object)
numerical_columns = numerical_columns_selector(data)
categorical_columns = categorical_columns_selector(data)
```

Reference pipeline (no numerical scaling and integer-coded categories)

First let’s time the pipeline we used in the main notebook to serve as a reference:

```python
import time

from sklearn.model_selection import cross_validate
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import HistGradientBoostingClassifier

categorical_preprocessor = OrdinalEncoder(handle_unknown="use_encoded_value",
                                          unknown_value=-1)
preprocessor = ColumnTransformer([
    ('categorical', categorical_preprocessor, categorical_columns)],
    remainder="passthrough")

model = make_pipeline(preprocessor, HistGradientBoostingClassifier())

start = time.time()
cv_results = cross_validate(model, data, target)
elapsed_time = time.time() - start

scores = cv_results["test_score"]

print("The mean cross-validation accuracy is: "
      f"{scores.mean():.3f} ± {scores.std():.3f} "
      f"with a fitting time of {elapsed_time:.3f}")
```

```
The mean cross-validation accuracy is: 0.873 ± 0.003 with a fitting time of 4.974
```

Scaling numerical features

Let's write a similar pipeline that also scales the numerical features using `StandardScaler` (or similar):

```python
# solution
import time

from sklearn.preprocessing import StandardScaler

preprocessor = ColumnTransformer([
    ('numerical', StandardScaler(), numerical_columns),
    ('categorical', OrdinalEncoder(handle_unknown="use_encoded_value",
                                   unknown_value=-1),
     categorical_columns)])

model = make_pipeline(preprocessor, HistGradientBoostingClassifier())

start = time.time()
cv_results = cross_validate(model, data, target)
elapsed_time = time.time() - start

scores = cv_results["test_score"]

print("The mean cross-validation accuracy is: "
      f"{scores.mean():.3f} ± {scores.std():.3f} "
      f"with a fitting time of {elapsed_time:.3f}")
```

```
The mean cross-validation accuracy is: 0.873 ± 0.003 with a fitting time of 5.030
```

Analysis

We can observe that both the accuracy and the training time are approximately the same as the reference pipeline (any time difference you might observe is not significant).

Scaling numerical features is indeed useless for most decision tree models in general and for `HistGradientBoostingClassifier` in particular.

One-hot encoding of categorical variables

We observed that integer coding of categorical variables can be very detrimental for linear models. However, it does not seem to be the case for `HistGradientBoostingClassifier` models, as the cross-validation score of the reference pipeline with `OrdinalEncoder` is reasonably good.

Let’s see if we can get an even better accuracy with `OneHotEncoder`.

Hint: `HistGradientBoostingClassifier` does not yet support sparse input data. You might want to use `OneHotEncoder(handle_unknown="ignore", sparse=False)` to force the use of a dense representation as a workaround.

```python
# solution
import time

from sklearn.preprocessing import OneHotEncoder

categorical_preprocessor = OneHotEncoder(handle_unknown="ignore", sparse=False)
preprocessor = ColumnTransformer([
    ('one-hot-encoder', categorical_preprocessor, categorical_columns)],
    remainder="passthrough")

model = make_pipeline(preprocessor, HistGradientBoostingClassifier())

start = time.time()
cv_results = cross_validate(model, data, target)
elapsed_time = time.time() - start

scores = cv_results["test_score"]

print("The mean cross-validation accuracy is: "
      f"{scores.mean():.3f} ± {scores.std():.3f} "
      f"with a fitting time of {elapsed_time:.3f}")
```

```
The mean cross-validation accuracy is: 0.873 ± 0.002 with a fitting time of 17.584
```

Analysis

From an accuracy point of view, the result is almost exactly the same. The reason is that `HistGradientBoostingClassifier` is expressive and robust enough to deal with misleading ordering of integer coded categories (which was not the case for linear models).

However from a computation point of view, the training time is much longer: this is caused by the fact that `OneHotEncoder` generates approximately 10 times more features than `OrdinalEncoder`.

Note that the current implementation `HistGradientBoostingClassifier` is still incomplete, and once sparse representation are handled correctly, training time might improve with such kinds of encodings.

The main take away message is that arbitrary integer coding of categories is perfectly fine for `HistGradientBoostingClassifier` and yields fast training times.

Important

Which encoder should I use?

	Meaningful order	Non-meaningful order
Tree-based model	`OrdinalEncoder`	`OrdinalEncoder`
Linear model	`OrdinalEncoder` with caution	`OneHotEncoder`

- `OneHotEncoder`: will always do something meaningful, but can be unnecessary slow with trees.
- `OrdinalEncoder`: can be detrimental for linear models unless your category has a meaningful order and you make sure that `OrdinalEncoder` respects this order. Trees can deal with `OrdinalEncoder` fine as long as they are deep enough.