03_categorical_pipeline

May 29, 2021

1 Encoding of categorical variables

In this notebook, we will present typical ways of dealing with **categorical variables** by encoding them, namely **ordinal encoding** and **one-hot encoding**.

Let's first load the entire adult dataset containing both numerical and categorical data.

[1]:
```
import pandas as pd

adult_census = pd.read_csv("../datasets/adult-census.csv")
# drop the duplicated column `"education-num"` as stated in the first notebook
adult_census = adult_census.drop(columns="education-num")

target_name = "class"
target = adult_census[target_name]

data = adult_census.drop(columns=[target_name])
```

1.1 Identify categorical variables

As we saw in the previous section, a numerical variable is a quantity represented by a real or integer number. These variables can be naturally handled by machine learning algorithms that are typically composed of a sequence of arithmetic instructions such as additions and multiplications.

In contrast, categorical variables have discrete values, typically represented by string labels (but not only) taken from a finite list of possible choices. For instance, the variable `native-country` in our dataset is a categorical variable because it encodes the data using a finite list of possible countries (along with the `?` symbol when this information is missing):

[2]:
```
data["native-country"].value_counts().sort_index()
```

[2]:
```
 ?                              857
 Cambodia                        28
 Canada                         182
 China                          122
 Columbia                        85
 Cuba                           138
 Dominican-Republic             103
 Ecuador                         45
```

```
El-Salvador                     155
England                         127
France                           38
Germany                         206
Greece                           49
Guatemala                        88
Haiti                            75
Holand-Netherlands                1
Honduras                         20
Hong                             30
Hungary                          19
India                           151
Iran                             59
Ireland                          37
Italy                           105
Jamaica                         106
Japan                            92
Laos                             23
Mexico                          951
Nicaragua                        49
Outlying-US(Guam-USVI-etc)       23
Peru                             46
Philippines                     295
Poland                           87
Portugal                         67
Puerto-Rico                     184
Scotland                         21
South                           115
Taiwan                           65
Thailand                         30
Trinadad&Tobago                  27
United-States                 43832
Vietnam                          86
Yugoslavia                       23
Name: native-country, dtype: int64
```

How can we easily recognize categorical columns among the dataset? Part of the answer lies in the columns' data type:

[3]:
```
data.dtypes
```

[3]:
```
age                  int64
workclass           object
education           object
marital-status      object
occupation          object
relationship        object
```

```
race              object
sex               object
capital-gain       int64
capital-loss       int64
hours-per-week     int64
native-country    object
dtype: object
```

If we look at the "native-country" column, we observe its data type is object, meaning it contains string values.

1.2 Select features based on their data type

In the previous notebook, we manually defined the numerical columns. We could do a similar approach. Instead, we will use the scikit-learn helper function make_column_selector, which allows us to select columns based on their data type. We will illustrate how to use this helper.

[4]:
```python
from sklearn.compose import make_column_selector as selector

categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(data)
categorical_columns
```

[4]:
```
['workclass',
 'education',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native-country']
```

Here, we created the selector by passing the data type to include; we then passed the input dataset to the selector object, which returned a list of column names that have the requested data type. We can now filter out the unwanted columns:

[5]:
```python
data_categorical = data[categorical_columns]
data_categorical.head()
```

[5]:
```
    workclass      education       marital-status          occupation  \
0     Private           11th        Never-married   Machine-op-inspct
1     Private        HS-grad   Married-civ-spouse     Farming-fishing
2   Local-gov     Assoc-acdm   Married-civ-spouse     Protective-serv
3     Private   Some-college   Married-civ-spouse   Machine-op-inspct
4           ?   Some-college        Never-married                   ?

  relationship    race     sex  native-country
0    Own-child   Black    Male   United-States
```

```
1      Husband   White     Male   United-States
2      Husband   White     Male   United-States
3      Husband   Black     Male   United-States
4    Own-child   White   Female   United-States
```

[6]:
```
print(f"The dataset is composed of {data_categorical.shape[1]} features")
```

```
The dataset is composed of 8 features
```

In the remainder of this section, we will present different strategies to encode categorical data into numerical data which can be used by a machine-learning algorithm.

1.3 Strategies to encode categories

1.3.1 Encoding ordinal categories

The most intuitive strategy is to encode each category with a different number. The `OrdinalEncoder` will transform the data in such manner. We will start by encoding a single column to understand how the encoding works.

[7]:
```
from sklearn.preprocessing import OrdinalEncoder

education_column = data_categorical[["education"]]

encoder = OrdinalEncoder()
education_encoded = encoder.fit_transform(education_column)
education_encoded
```

[7]:
```
array([[ 1.],
       [11.],
       [ 7.],
       …,
       [11.],
       [11.],
       [11.]])
```

We see that each category in `"education"` has been replaced by a numeric value. We could check the mapping between the categories and the numerical values by checking the fitted attribute `categories_`.

[8]:
```
encoder.categories_
```

[8]:
```
[array([' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',
        ' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors', ' Doctorate',
        ' HS-grad', ' Masters', ' Preschool', ' Prof-school',
        ' Some-college'], dtype=object)]
```

Now, we can check the encoding applied on all categorical features.

[9]:
```python
data_encoded = encoder.fit_transform(data_categorical)
data_encoded[:5]
```

[9]:
```
array([[ 4.,  1.,  4.,  7.,  3.,  2.,  1., 39.],
       [ 4., 11.,  2.,  5.,  0.,  4.,  1., 39.],
       [ 2.,  7.,  2., 11.,  0.,  4.,  1., 39.],
       [ 4., 15.,  2.,  7.,  0.,  2.,  1., 39.],
       [ 0., 15.,  4.,  0.,  3.,  4.,  0., 39.]])
```

[10]:
```python
encoder.categories_
```

[10]:
```
[array([' ?', ' Federal-gov', ' Local-gov', ' Never-worked', ' Private',
        ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'],
       dtype=object),
 array([' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',
        ' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors', ' Doctorate',
        ' HS-grad', ' Masters', ' Preschool', ' Prof-school',
        ' Some-college'], dtype=object),
 array([' Divorced', ' Married-AF-spouse', ' Married-civ-spouse',
        ' Married-spouse-absent', ' Never-married', ' Separated',
        ' Widowed'], dtype=object),
 array([' ?', ' Adm-clerical', ' Armed-Forces', ' Craft-repair',
        ' Exec-managerial', ' Farming-fishing', ' Handlers-cleaners',
        ' Machine-op-inspct', ' Other-service', ' Priv-house-serv',
        ' Prof-specialty', ' Protective-serv', ' Sales', ' Tech-support',
        ' Transport-moving'], dtype=object),
 array([' Husband', ' Not-in-family', ' Other-relative', ' Own-child',
        ' Unmarried', ' Wife'], dtype=object),
 array([' Amer-Indian-Eskimo', ' Asian-Pac-Islander', ' Black', ' Other',
        ' White'], dtype=object),
 array([' Female', ' Male'], dtype=object),
 array([' ?', ' Cambodia', ' Canada', ' China', ' Columbia', ' Cuba',
        ' Dominican-Republic', ' Ecuador', ' El-Salvador', ' England',
        ' France', ' Germany', ' Greece', ' Guatemala', ' Haiti',
        ' Holand-Netherlands', ' Honduras', ' Hong', ' Hungary', ' India',
        ' Iran', ' Ireland', ' Italy', ' Jamaica', ' Japan', ' Laos',
        ' Mexico', ' Nicaragua', ' Outlying-US(Guam-USVI-etc)', ' Peru',
        ' Philippines', ' Poland', ' Portugal', ' Puerto-Rico',
        ' Scotland', ' South', ' Taiwan', ' Thailand', ' Trinadad&Tobago',
        ' United-States', ' Vietnam', ' Yugoslavia'], dtype=object)]
```

[11]:
```python
print(
    f"The dataset encoded contains {data_encoded.shape[1]} features")
```

```
The dataset encoded contains 8 features
```

We see that the categories have been encoded for each feature (column) independently. We also note that the number of features before and after the encoding is the same.

However, be careful when applying this encoding strategy: using this integer representation leads downstream predictive models to assume that the values are ordered ($0 < 1 < 2 < 3$… for instance).

By default, `OrdinalEncoder` uses a lexicographical strategy to map string category labels to integers. This strategy is arbitrary and often meaningless. For instance, suppose the dataset has a categorical variable named `"size"` with categories such as "S", "M", "L", "XL". We would like the integer representation to respect the meaning of the sizes by mapping them to increasing integers such as `0, 1, 2, 3`. However, the lexicographical strategy used by default would map the labels "S", "M", "L", "XL" to 2, 1, 0, 3, by following the alphabetical order.

The `OrdinalEncoder` class accepts a `categories` constructor argument to pass categories in the expected ordering explicitly. You can find more information in the scikit-learn documentation if needed.

If a categorical variable does not carry any meaningful order information then this encoding might be misleading to downstream statistical models and you might consider using one-hot encoding instead (see below).

1.3.2 Encoding nominal categories (without assuming any order)

`OneHotEncoder` is an alternative encoder that prevents the downstream models to make a false assumption about the ordering of categories. For a given feature, it will create as many new columns as there are possible categories. For a given sample, the value of the column corresponding to the category will be set to `1` while all the columns of the other categories will be set to `0`.

We will start by encoding a single feature (e.g. `"education"`) to illustrate how the encoding works.

[12]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)
education_encoded = encoder.fit_transform(education_column)
education_encoded
```

[12]:
```
array([[0., 1., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.]])
```

Note

sparse=False is used in the OneHotEncoder for didactic purposes, namely easier visualization of the data.

Sparse matrices are efficient data structures when most of your matrix elements are zero. They won't be covered in details in this course. If you want more details about them, you can look at this.

We see that encoding a single feature will give a NumPy array full of zeros and ones. We can get a better understanding using the associated feature names resulting from the transformation.

[13]:
```python
feature_names = encoder.get_feature_names(input_features=["education"])
education_encoded = pd.DataFrame(education_encoded, columns=feature_names)
education_encoded
```

[13]:
```
       education_ 10th  education_ 11th  education_ 12th  education_ 1st-4th  \
0                  0.0              1.0              0.0                 0.0
1                  0.0              0.0              0.0                 0.0
2                  0.0              0.0              0.0                 0.0
3                  0.0              0.0              0.0                 0.0
4                  0.0              0.0              0.0                 0.0
…                 …               …               …                  …
48837              0.0              0.0              0.0                 0.0
48838              0.0              0.0              0.0                 0.0
48839              0.0              0.0              0.0                 0.0
48840              0.0              0.0              0.0                 0.0
48841              0.0              0.0              0.0                 0.0

       education_ 5th-6th  education_ 7th-8th  education_ 9th  \
0                     0.0                 0.0             0.0
1                     0.0                 0.0             0.0
2                     0.0                 0.0             0.0
3                     0.0                 0.0             0.0
4                     0.0                 0.0             0.0
…                    …                  …              …
48837                 0.0                 0.0             0.0
48838                 0.0                 0.0             0.0
48839                 0.0                 0.0             0.0
48840                 0.0                 0.0             0.0
48841                 0.0                 0.0             0.0

       education_ Assoc-acdm  education_ Assoc-voc  education_ Bachelors  \
0                        0.0                   0.0                   0.0
1                        0.0                   0.0                   0.0
2                        1.0                   0.0                   0.0
3                        0.0                   0.0                   0.0
4                        0.0                   0.0                   0.0
…                       …                    …                    …
48837                    1.0                   0.0                   0.0
48838                    0.0                   0.0                   0.0
48839                    0.0                   0.0                   0.0
48840                    0.0                   0.0                   0.0
48841                    0.0                   0.0                   0.0

       education_ Doctorate  education_ HS-grad  education_ Masters  \
0                       0.0                 0.0                 0.0
1                       0.0                 1.0                 0.0
2                       0.0                 0.0                 0.0
```

```
3                    0.0              0.0              0.0
4                    0.0              0.0              0.0
…                    …                …                …
48837                0.0              0.0              0.0
48838                0.0              1.0              0.0
48839                0.0              1.0              0.0
48840                0.0              1.0              0.0
48841                0.0              1.0              0.0

       education_ Preschool  education_ Prof-school  education_ Some-college
0                    0.0                    0.0                     0.0
1                    0.0                    0.0                     0.0
2                    0.0                    0.0                     0.0
3                    0.0                    0.0                     1.0
4                    0.0                    0.0                     1.0
…                    …                      …                       …
48837                0.0                    0.0                     0.0
48838                0.0                    0.0                     0.0
48839                0.0                    0.0                     0.0
48840                0.0                    0.0                     0.0
48841                0.0                    0.0                     0.0

[48842 rows x 16 columns]
```

As we can see, each category (unique value) became a column; the encoding returned, for each sample, a 1 to specify which category it belongs to.

Let's apply this encoding on the full dataset.

[14]:
```python
print(
    f"The dataset is composed of {data_categorical.shape[1]} features")
data_categorical.head()
```

```
The dataset is composed of 8 features
```

[14]:
```
   workclass      education       marital-status          occupation  \
0    Private           11th        Never-married   Machine-op-inspct
1    Private        HS-grad   Married-civ-spouse     Farming-fishing
2  Local-gov     Assoc-acdm   Married-civ-spouse     Protective-serv
3    Private   Some-college   Married-civ-spouse   Machine-op-inspct
4          ?   Some-college        Never-married                   ?

  relationship   race     sex  native-country
0    Own-child  Black    Male   United-States
1      Husband  White    Male   United-States
2      Husband  White    Male   United-States
3      Husband  Black    Male   United-States
4    Own-child  White  Female   United-States
```

[15]:
```python
data_encoded = encoder.fit_transform(data_categorical)
data_encoded[:5]
```

[15]:
```
array([[0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
        0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0.,
        1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1., 0., 0.]])
```

[16]:
```python
print(
    f"The encoded dataset contains {data_encoded.shape[1]} features")
```

```
The encoded dataset contains 102 features
```

Let's wrap this NumPy array in a dataframe with informative column names as provided by the encoder object:

[17]:
```
columns_encoded = encoder.get_feature_names(data_categorical.columns)
pd.DataFrame(data_encoded, columns=columns_encoded).head()
```

[17]:
```
   workclass_ ?  workclass_ Federal-gov  workclass_ Local-gov  \
0           0.0                     0.0                   0.0
1           0.0                     0.0                   0.0
2           0.0                     0.0                   1.0
3           0.0                     0.0                   0.0
4           1.0                     0.0                   0.0

   workclass_ Never-worked  workclass_ Private  workclass_ Self-emp-inc  \
0                      0.0                 1.0                      0.0
1                      0.0                 1.0                      0.0
2                      0.0                 0.0                      0.0
3                      0.0                 1.0                      0.0
4                      0.0                 0.0                      0.0

   workclass_ Self-emp-not-inc  workclass_ State-gov  workclass_ Without-pay  \
0                          0.0                   0.0                     0.0
1                          0.0                   0.0                     0.0
2                          0.0                   0.0                     0.0
3                          0.0                   0.0                     0.0
4                          0.0                   0.0                     0.0

   education_ 10th  …  native-country_ Portugal  \
0              0.0  …                       0.0
1              0.0  …                       0.0
2              0.0  …                       0.0
3              0.0  …                       0.0
4              0.0  …                       0.0

   native-country_ Puerto-Rico  native-country_ Scotland  \
0                          0.0                       0.0
1                          0.0                       0.0
2                          0.0                       0.0
3                          0.0                       0.0
4                          0.0                       0.0

   native-country_ South  native-country_ Taiwan  native-country_ Thailand  \
0                    0.0                     0.0                       0.0
1                    0.0                     0.0                       0.0
2                    0.0                     0.0                       0.0
3                    0.0                     0.0                       0.0
4                    0.0                     0.0                       0.0

   native-country_ Trinadad&Tobago  native-country_ United-States  \
0                              0.0                            1.0
```

```
1                                0.0                                1.0
2                                0.0                                1.0
3                                0.0                                1.0
4                                0.0                                1.0

   native-country_ Vietnam  native-country_ Yugoslavia
0                      0.0                         0.0
1                      0.0                         0.0
2                      0.0                         0.0
3                      0.0                         0.0
4                      0.0                         0.0

[5 rows x 102 columns]
```

Look at how the "workclass" variable of the 3 first records has been encoded and compare this to the original string representation.

The number of features after the encoding is more than 10 times larger than in the original data because some variables such as `occupation` and `native-country` have many possible categories.

1.3.3 Choosing an encoding strategy

Choosing an encoding strategy will depend on the underlying models and the type of categories (i.e. ordinal vs. nominal).

Indeed, using an `OrdinaleEncoder` will output ordinal categories. It means that there is an order in the resulting categories (e.g. `0 > 1 > 2`). The impact of violating this ordering assumption is really dependent on the downstream models. Linear models will be impacted by misordered categories while tree-based models will not be.

Thus, in general `OneHotEncoder` is the encoding strategy used when the downstream models are **linear models** while `OrdinalEncoder` is used with **tree-based models**.

You still can use an `OrdinalEncoder` with linear models but you need to be sure that: - the original categories (before encoding) have an ordering; - the encoded categories follow the same ordering than the original categories. The next exercise highlight the issue of misusing `OrdinalEncoder` with a linear model.

Also, there is no need to use an `OneHotEncoder` even if the original categories do not have an given order with tree-based model. It will be the purpose of the final exercise of this sequence.

1.4 Evaluate our predictive pipeline

We can now integrate this encoder inside a machine learning pipeline like we did with numerical data: let's train a linear classifier on the encoded data and check the statistical performance of this machine learning pipeline using cross-validation.

Before we create the pipeline, we have to linger on the `native-country`. Let's recall some statistics regarding this column.

[18]:
```
data["native-country"].value_counts()
```

[18]:
```
United-States                 43832
Mexico                          951
?                               857
Philippines                     295
Germany                         206
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Guatemala                        88
Poland                           87
Vietnam                          86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Nicaragua                        49
Greece                           49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Thailand                         30
Hong                             30
Cambodia                         28
Trinadad&Tobago                  27
Outlying-US(Guam-USVI-etc)       23
Laos                             23
Yugoslavia                       23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: native-country, dtype: int64
```

We see that the `Holand-Netherlands` category is occurring rarely. This will be a problem during cross-validation: if the sample ends up in the test set during splitting then the classifier would not have seen the category during training and will not be able to encode it.

In scikit-learn, there are two solutions to bypass this issue:

- list all the possible categories and provide it to the encoder via the keyword argument `categories`;
- use the parameter `handle_unknown`.

Here, we will use the latter solution for simplicity.

Tip

Be aware the OrdinalEncoder exposes as well a parameter handle_unknown. It can be set to use_encoded_value and by setting unknown_value to handle rare categories. You are going to use these parameters in the next exercise.

We can now create our machine learning pipeline.

[19]:
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

model = make_pipeline(
    OneHotEncoder(handle_unknown="ignore"), LogisticRegression(max_iter=500)
)
```

Note

Here, we need to increase the maximum number of iterations to obtain a fully converged LogisticRegression and silence a ConvergenceWarning. Contrary to the numerical features, the one-hot encoded categorical features are all on the same scale (values are 0 or 1), so they would not benefit from scaling. In this case, increasing max_iter is the right thing to do.

Finally, we can check the model's statistical performance only using the categorical columns.

[20]:
```
from sklearn.model_selection import cross_validate
cv_results = cross_validate(model, data_categorical, target)
cv_results
```

[20]:
```
{'fit_time': array([0.85367131, 0.86852932, 0.80280256, 0.86758685,
0.84853792]),
 'score_time': array([0.02705097, 0.02723289, 0.02719402, 0.02859902,
0.02724504]),
 'test_score': array([0.83222438, 0.83560242, 0.82872645, 0.83312858,
0.83466421])}
```

[21]:
```
scores = cv_results["test_score"]
print(f"The accuracy is: {scores.mean():.3f} +/- {scores.std():.3f}")
```

```
The accuracy is: 0.833 +/- 0.002
```

As you can see, this representation of the categorical variables is slightly more predictive of the revenue than the numerical variables that we used previously.

In this notebook we have: * seen two common strategies for encoding categorical features: **ordinal encoding** and **one-hot encoding**; * used a **pipeline** to use a **one-hot encoder** before fitting a logistic regression.