Encoding of categorical variables

Source: https://esciencecenter-digital-skills.github.io/scikit-learn-mooc/python_scripts/03_categorical_pipeline.html

In [1]:
```
import pandas as pd
```

In [2]:
```
adult_census = pd.read_csv("../../datasets/adult-census.csv")
adult_census = adult_census.drop(columns="education-num")

target_name = "class"
# we separate the predictors and the target
target = adult_census[target_name]
data = adult_census.drop(columns=[target_name])
```

Categorical variables are often strings -> harder to handle for a computer

In [3]:
```
data["native-country"].value_counts().sort_index()
```

Out[3]:
```
native-country
?                              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: count, dtype: int64
```

In [4]:
```
data.dtypes
```

Out[4]:
```
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
```

Selecting features based on their data type

In [5]:
```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
```

Out[5]:
```
['workclass',
 'education',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native-country']
```

In [6]:
```python
data_categorical = data[categorical_columns]
data_categorical.head()
```

Out[6]:

	workclass	education	marital-status	occupation	relationship	race	sex	native-country
0	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	United-States
1	Private	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male	United-States
2	Local-gov	Assoc-acdm	Married-civ-spouse	Protective-serv	Husband	White	Male	United-States
3	Private	Some-college	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	United-States
4	?	Some-college	Never-married	?	Own-child	White	Female	United-States

In [7]:
```python
print(f"The dataset contains {data_categorical.shape[1]} categorical features")
```

```
The dataset contains 8 categorical features
```

Strategies to encode categorical variables

Encoding ordinal categories

In [8]:
```python
from sklearn.preprocessing import OrdinalEncoder
education_column = data_categorical[["education"]]

encoder = OrdinalEncoder().set_output(transform="pandas")
education_encoded = encoder.fit_transform(education_column)
education_encoded
```

Out[8]:

	education
0	1.0
1	11.0
2	7.0
3	15.0
4	15.0
...	...
48837	7.0
48838	11.0
48839	11.0
48840	11.0
48841	11.0

48842 rows × 1 columns

In [9]:
```python
encoder.categories_
```

Out[9]:
```
[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)]
```

In [10]:
```python
data_encoded = encoder.fit_transform(data_categorical)
data_encoded[:5]
```

Out[10]:

	workclass	education	marital-status	occupation	relationship	race	sex	native-country
0	4.0	1.0	4.0	7.0	3.0	2.0	1.0	39.0
1	4.0	11.0	2.0	5.0	0.0	4.0	1.0	39.0
2	2.0	7.0	2.0	11.0	0.0	4.0	1.0	39.0
3	4.0	15.0	2.0	7.0	0.0	2.0	1.0	39.0
4	0.0	15.0	4.0	0.0	3.0	4.0	0.0	39.0

In [11]:
```
data_encoded["education"].min()
```

Out[11]:
```
np.float64(0.0)
```

In [12]:
```
encoder.categories_
```

Out[12]:
```
[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)]
```

Note

- number of features is unchanged
- each feature has been encoded separetely (numbers re-start at 0 each time)
- careful, 1: it uses a lexigraphical strategy -> see how "10th" comes before "9th"
- careful, 2: assumes values are ordered -> but this can lead you astray.
  - example: cloth sizes: `S, M, L, XL` would be encoded `2,1,0,3`
  - we can pass a `categories` argument to tell the encoded the expected ordering
  - but even then, in many cases it's not clear if all categories are "equally far apart", ie 1,2,3,4, or if it should be something else!
- ignoring this can worsen the statistical models because the ordering has not meaning, and the model picks up noise. In such cases, one-hot encoding might be better

Encoding nominal categories -- assuming no order

In [13]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse_output=False).set_output(transform="pandas")
# encoder = OneHotEncoder(sparse_output=False).set_output(transform="pandas")
education_encoded = encoder.fit_transform(education_column)
education_encoded
```

Out[13]:

	education_10th	education_11th	education_12th	education_1st-4th	education_5th-6th	education_7th-8th	education_9th	education_Assoc-acdm	education_Assoc-voc	education_Bachelors
0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
48837	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
48838	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48839	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48840	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48841	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

48842 rows × 16 columns

In [14]:
```python
print(f"The dataset has {data_categorical.shape[1]} numerical features")
```

```
The dataset has 8 numerical features
```

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

In [16]:
```python
data_encoded[:5]
```

Out[16]:

	workclass_?	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	workclass_State-gov	workclass_Without-pay	education_10th	...
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 102 columns

In [17]:
```python
print(f"The encoded dataset contains {data_encoded.shape[1]} features")
```

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

Observations

- `OrdinalEncoder` outputs ordinal categories -> better for tree-based models, unless categories are truly ordered
- `OneHotEncoder` creates many features, which is not computationally efficient for tree-based models.

Evaluate our predictive pipeline

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

```
Out[18]: native-country
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
Laos                             23
Outlying-US(Guam-USVI-etc)       23
Yugoslavia                       23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: count, dtype: int64
```

Encoder will fail if the sample is in the test set but not in the train set (because the encoder learns on the train set).

Options to deal with small categories

- use kwarg `categories` in the `fit` method to define all categories
- `handle_unkown` ="ignore" -> all one-hot encoded columns for this record will be 0
- adjust `min_frequency` -> collapse rarest categories into one single feature.

We will only use the second option in the live coding

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

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

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

```
Out[20]: {'fit_time': array([0.12081909, 0.1092937 , 0.11319757, 0.11659431, 0.10908055]),
 'score_time': array([0.01381397, 0.01384926, 0.01356745, 0.01423931, 0.01363182]),
 'test_score': array([0.83232675, 0.83570478, 0.82831695, 0.83292383, 0.83497133])}
```

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

```
The accuracy is: 0.833 +- 0.003
```

Exercise: The impact of using integer encoding for with logistic regression (groups of 2, 15min) [Flavio]

Goal: understand the impact of arbitrary integer encoding for categorical variables with linear classification such as logistic regression.

We keep using the `adult_census` data set already loaded in the code before. Recall that `target` contains the variable we want to predict and `data` contains the features.

If you need to re-load the data, you can do it as follows:

```
import pandas as pd


adult_census = pd.read_csv("../datasets/adult-census.csv")
target_name = "class"
target = adult_census[target_name]
data = adult_census.drop(columns=[target_name, "education-num"])
```

Q0 Select columns containing strings

Use `sklearn.compose.make_column_selector` to automatically select columns containing strings that correspond to categorical features in our dataset.

Q1 Build a scikit-learn pipeline composed of an `OrdinalEncoder` and a `LogisticRegression` classifier

You'll need the following, already loaded modules:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import LogisticRegression
```

Because OrdinalEncoder can raise errors if it sees an unknown category at prediction time, you can set the handle_unknown="use_encoded_value" and unknown_value parameters. You can refer to the scikit-learn documentation for more details regarding these parameters.

Q2 Evaluate the model with cross-validation.

You'll need the following, already loaded modules:

```
from sklearn.model_selection import cross_validate
```

Q3 Repeat the previous steps using an `OneHotEncoder` instead of an `OrdinalEncoder`

You'll need the following, already loaded modules:

```
from sklearn.preprocessing import OneHotEncoder
```

Solution

In [22]:
```
import pandas as pd

adult_census = pd.read_csv("../../datasets/adult-census.csv")
target_name = "class"
target = adult_census[target_name]
data = adult_census.drop(columns=[target_name, "education-num"])
```

Q0

In [23]:
```
from sklearn.compose import make_column_selector
selector = make_column_selector(dtype_include=object)
categorical_columns = selector(data)
data_categorical = data[categorical_columns]
```

Q1: build pipeline

In [24]:
```
model = make_pipeline(
    OrdinalEncoder(handle_unknown="use_encoded_value", unknown_value=-1),
    LogisticRegression(max_iter=500)
)
```

Q2: Cross-validate

In [25]:
```
cv_results = cross_validate(model, data_categorical, target)
```

In [26]:
```
scores = cv_results["test_score"]
print(
    "The mean cross-validation accuracy is: "
    f"{scores.mean():.3f} ± {scores.std():.3f}"
)
```

```
The mean cross-validation accuracy is: 0.755 ± 0.002
```

This performs even worse than using the most frequent class -- which is often a good baseline to compare our models against

In [27]:
```
from sklearn.dummy import DummyClassifier

cv_results = cross_validate(
    DummyClassifier(strategy="most_frequent"), data_categorical, target
)
scores = cv_results["test_score"]
print(
    "The mean cross-validation accuracy is: "
    f"{scores.mean():.3f} ± {scores.std():.3f}"
)
```

```
The mean cross-validation accuracy is: 0.761 ± 0.000
```

Q3: OneHotEncoder

In [28]:
```
model = make_pipeline(
    OneHotEncoder(handle_unknown="ignore"),
    LogisticRegression(max_iter=500)
)
cv_results = cross_validate(model, data_categorical, target)
scores = cv_results["test_score"]
print(
    "The mean cross-validation accuracy is: "
    f"{scores.mean():.3f} ± {scores.std():.3f}"
)
```

```
The mean cross-validation accuracy is: 0.833 ± 0.003
```

In [ ]:

In [ ]:

In [ ]: