Machine Learning 03
Preprocessing - Categorical to number conversions

Claudio Sartori

1 Preprocessing: transform categorical data

In `scikit-learn` the classifiers require numeric data. The library makes available a set of preprocessing fuctions which help the transformation. This exercise proposes two types of transformations:

- `OneHotEncoder` for purely categorical columns: if the column has **V** distinct values it is substituted by **V** binary columns where in each row only the bit corrosponding to the original value is true
- `OrdinalEncoder` for ordinal columns: the original **V** values are mapped into the **0..V-1** range

The additional function `ColumnTransformer` allows to apply the different transformations to the appropriate columns with a single statement.

1.0.1 To do:

- import the appropriate names
- set the random state
- import the data set with the appropriate column names
- inspect the content and the data types
- read carefully the `.names` file of the data set, to understand which are the ordinal and categorical data
- data cleaning
  - the **ordinal transformer** generates a mapping from strings to numbers according to the lexicographic sorting of the strings; in this particular case, the strings indicate numeric subranges, and ranges with one digit constitute exceptions '5-9' happens to be after '20-25'
  - it is necessary to transform '5-9' into '05-09', and the same for other similar cases
  - a way to do this is to prepare dictionaries for the translation and use the `.map` function
- prepare the lists of the ordinal, categorical and numeric columns
- prepare the preprocessor
- split the cleaned data into the X and y part
- fit_transform the preprocessor and generate the transformed data set
- split the transformed data set into train and test
- use the same method used for the exercise of 19/11 to test several classifiers

`http://scikit-`

```
learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html
@author: scikit-learn.org and Claudio Sartori
```

[2]:

```
                  Class    age menopause tumor-size inv-nodes node-caps  \
0  no-recurrence-events  30-39   premeno      30-34       0-2        no
1  no-recurrence-events  40-49   premeno      20-24       0-2        no
2  no-recurrence-events  40-49   premeno      20-24       0-2        no
3  no-recurrence-events  60-69      ge40      15-19       0-2        no
4  no-recurrence-events  40-49   premeno        0-4       0-2        no

   deg-malig breast breast-quad irradiat
0          3   left    left_low       no
1          2  right    right_up       no
2          2   left    left_low       no
3          2  right     left_up       no
4          2  right   right_low       no
```

Show the types of the columns

```
Class          object
age            object
menopause      object
tumor-size     object
inv-nodes      object
node-caps      object
deg-malig       int64
breast         object
breast-quad    object
irradiat       object
dtype: object
```

Clean the column `tumor-size`

[4]:

```
{'30-34': '30-34',
 '20-24': '20-24',
 '15-19': '15-19',
 '0-4': '0-4',
 '25-29': '25-29',
 '50-54': '50-54',
 '10-14': '10-14',
 '40-44': '40-44',
 '35-39': '35-39',
 '5-9': '5-9',
 '45-49': '45-49'}
```

Clean the column `inv-nodes`

Inspect the data

```
[10]:                  Class    age menopause tumor-size inv-nodes node-caps  \
      0  no-recurrence-events  30-39   premeno      30-34     00-02        no
      1  no-recurrence-events  40-49   premeno      20-24     00-02        no
      2  no-recurrence-events  40-49   premeno      20-24     00-02        no
      3  no-recurrence-events  60-69      ge40      15-19     00-02        no
      4  no-recurrence-events  40-49   premeno      00-04     00-02        no

         deg-malig breast breast-quad irradiat
      0          3   left    left_low       no
      1          2  right    right_up       no
      2          2   left    left_low       no
      3          2  right     left_up       no
      4          2  right   right_low       no
```

Prepare the lists of numeric features, ordinal features, categorical features

```
The non-numeric features are:
['Class' 'age' 'menopause' 'tumor-size' 'inv-nodes' 'node-caps' 'breast'
 'breast-quad' 'irradiat']

The numeric features are:
['deg-malig']

The ordinal features are:
['age', 'tumor-size', 'inv-nodes']

The categorical features are:
['menopause', 'irradiat', 'breast', 'node-caps', 'breast-quad']
```

Prepare the transformer

Split X and y and check the shapes

```
The labels are:
['no-recurrence-events' 'recurrence-events']
```

```
[18]: (286, 9)
```

Fit the preprocessor with X and check the parameters printing the `.named_transformers_` attribute

```
[19]: ColumnTransformer(n_jobs=None, remainder='passthrough', sparse_threshold=0.3,
                        transformer_weights=None,
                        transformers=[('cat',
                                       OneHotEncoder(categorical_features=None,
                                                     categories=None, drop=None,
                                                     dtype=<class 'numpy.int32'>,
                                                     handle_unknown='ignore',
                                                     n_values=None, sparse=False),
                                       ['menopause', 'irradiat', 'breast',
                                        'node-caps', 'breast-quad']),
```

```
                                         ('ord',
                                          OrdinalEncoder(categories='auto',
                                                         dtype=<class 'numpy.int32'>),
                                          ['age', 'tumor-size', 'inv-nodes'])],
                         verbose=False)

{'cat': OneHotEncoder(categorical_features=None, categories=None, drop=None,
              dtype=<class 'numpy.int32'>, handle_unknown='ignore',
              n_values=None, sparse=False), 'ord':
OrdinalEncoder(categories='auto', dtype=<class 'numpy.int32'>), 'remainder':
'passthrough'}
```

Fit-transform X and store the result in X_p, check the shape

[22]:
```
(286, 20)
```

For ease of inspection transform `X_p` into a data frame `df_p` and inspect it

[24]:

	0	1	2	3	4	5
count	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000
mean	0.451049	0.024476	0.524476	0.762238	0.237762	0.531469
std	0.498470	0.154791	0.500276	0.426459	0.426459	0.499883
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000
75%	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	6	7	8	9	10	11
count	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000
mean	0.468531	0.027972	0.776224	0.195804	0.003497	0.073427
std	0.499883	0.165182	0.417504	0.397514	0.059131	0.261293
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	12	13	14	15	16	17
count	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000
mean	0.384615	0.339161	0.083916	0.115385	2.664336	4.881119
std	0.487357	0.474254	0.277748	0.320046	1.011818	2.105930
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	2.000000	4.000000
50%	0.000000	0.000000	0.000000	0.000000	3.000000	5.000000
75%	1.000000	1.000000	0.000000	0.000000	3.000000	6.000000
max	1.000000	1.000000	1.000000	1.000000	5.000000	10.000000

	18	19
count	286.000000	286.000000
mean	0.517483	2.048951
std	1.110417	0.738217
min	0.000000	1.000000
25%	0.000000	2.000000
50%	0.000000	2.000000
75%	1.000000	3.000000
max	6.000000	3.000000

[25]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	6	0	3
1	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	1	2	4	0	2
2	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	2	4	0	2
3	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	4	3	0	2
4	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	0	2	0	0	2

The columns in the transformed dataset are generated according to the order you see printing the preprocessor after fitting, therefore the last four columns correspond to `'age'`, `'tumor-size'`, `'inv-nodes'`, `'deg-malig'`.

In order to inspect if the translation and check if the mapping is as expected, compare the sorted values of df['tumor-size'] and df_p[17], e.g. comparing the index sequences

```
The number of index discordances between 'tumor-size' and '17' is 0
```

Train/test split

Classification and test

```
========================================
# Tuning hyper-parameters for recall_macro

----------------------------------------
Trying model Decision Tree
Best parameters set found on train set:

{'max_depth': 14}

Grid scores on train set:

0.567 (+/-0.086) for {'max_depth': 1}
0.610 (+/-0.115) for {'max_depth': 2}
0.583 (+/-0.127) for {'max_depth': 3}
0.551 (+/-0.082) for {'max_depth': 4}
0.574 (+/-0.148) for {'max_depth': 5}
0.574 (+/-0.138) for {'max_depth': 6}
0.597 (+/-0.148) for {'max_depth': 7}
0.591 (+/-0.226) for {'max_depth': 8}
0.567 (+/-0.223) for {'max_depth': 9}
```

```
0.576 (+/-0.285) for {'max_depth': 10}
0.577 (+/-0.168) for {'max_depth': 11}
0.552 (+/-0.166) for {'max_depth': 12}
0.564 (+/-0.163) for {'max_depth': 13}
0.620 (+/-0.187) for {'max_depth': 14}
0.565 (+/-0.142) for {'max_depth': 15}
0.573 (+/-0.089) for {'max_depth': 16}
0.608 (+/-0.121) for {'max_depth': 17}
0.571 (+/-0.154) for {'max_depth': 18}
0.576 (+/-0.177) for {'max_depth': 19}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

                     precision    recall  f1-score   support

no-recurrence-events       0.72      0.84      0.77        49
   recurrence-events       0.47      0.30      0.37        23

            accuracy                           0.67        72
           macro avg       0.59      0.57      0.57        72
        weighted avg       0.64      0.67      0.64        72

[[41  8]
 [16  7]]

----------------------------------------
Trying model Gaussian Naive Bayes
Best parameters set found on train set:

{'var_smoothing': 0.01}

Grid scores on train set:

0.500 (+/-0.000) for {'var_smoothing': 10}
0.506 (+/-0.049) for {'var_smoothing': 1}
0.593 (+/-0.115) for {'var_smoothing': 0.1}
0.629 (+/-0.134) for {'var_smoothing': 0.01}
0.627 (+/-0.125) for {'var_smoothing': 0.001}
0.624 (+/-0.121) for {'var_smoothing': 0.0001}
0.611 (+/-0.076) for {'var_smoothing': 1e-05}
0.601 (+/-0.092) for {'var_smoothing': 1e-06}
0.591 (+/-0.094) for {'var_smoothing': 1e-07}
0.577 (+/-0.124) for {'var_smoothing': 1e-08}
0.556 (+/-0.142) for {'var_smoothing': 1e-09}
0.551 (+/-0.135) for {'var_smoothing': 1e-10}
```

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

	precision	recall	f1-score	support
no-recurrence-events	0.73	0.88	0.80	49
recurrence-events	0.54	0.30	0.39	23
accuracy			0.69	72
macro avg	0.63	0.59	0.59	72
weighted avg	0.67	0.69	0.67	72

```
[[43  6]
 [16  7]]
```

Trying model Linear Perceptron
Best parameters set found on train set:

{'early_stopping': True}

Grid scores on train set:

0.564 (+/-0.111) for {'early_stopping': True}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

	precision	recall	f1-score	support
no-recurrence-events	1.00	0.14	0.25	49
recurrence-events	0.35	1.00	0.52	23
accuracy			0.42	72
macro avg	0.68	0.57	0.39	72
weighted avg	0.79	0.42	0.34	72

```
[[ 7 42]
 [ 0 23]]
```

Trying model Support Vector
Best parameters set found on train set:

```
{'C': 10, 'kernel': 'linear'}

Grid scores on train set:

0.500 (+/-0.000) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.495 (+/-0.048) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.549 (+/-0.064) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.495 (+/-0.048) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.574 (+/-0.122) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.554 (+/-0.074) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.582 (+/-0.091) for {'C': 1, 'kernel': 'linear'}
0.599 (+/-0.159) for {'C': 10, 'kernel': 'linear'}
0.599 (+/-0.159) for {'C': 100, 'kernel': 'linear'}
0.599 (+/-0.159) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

                      precision    recall  f1-score   support

no-recurrence-events       0.73      0.92      0.81        49
   recurrence-events       0.60      0.26      0.36        23

            accuracy                           0.71        72
           macro avg       0.66      0.59      0.59        72
        weighted avg       0.69      0.71      0.67        72

[[45  4]
 [17  6]]

----------------------------------------
Trying model K Nearest Neighbor
Best parameters set found on train set:

{'metric': 'manhattan', 'n_neighbors': 7}

Grid scores on train set:

0.567 (+/-0.088) for {'metric': 'euclidean', 'n_neighbors': 1}
0.524 (+/-0.037) for {'metric': 'euclidean', 'n_neighbors': 2}
0.554 (+/-0.190) for {'metric': 'euclidean', 'n_neighbors': 3}
0.542 (+/-0.104) for {'metric': 'euclidean', 'n_neighbors': 4}
0.548 (+/-0.115) for {'metric': 'euclidean', 'n_neighbors': 5}
```

```
0.502 (+/-0.075) for {'metric': 'euclidean', 'n_neighbors': 6}
0.555 (+/-0.096) for {'metric': 'euclidean', 'n_neighbors': 7}
0.523 (+/-0.080) for {'metric': 'euclidean', 'n_neighbors': 8}
0.521 (+/-0.091) for {'metric': 'euclidean', 'n_neighbors': 9}
0.524 (+/-0.079) for {'metric': 'euclidean', 'n_neighbors': 10}
0.578 (+/-0.077) for {'metric': 'manhattan', 'n_neighbors': 1}
0.554 (+/-0.063) for {'metric': 'manhattan', 'n_neighbors': 2}
0.554 (+/-0.170) for {'metric': 'manhattan', 'n_neighbors': 3}
0.570 (+/-0.086) for {'metric': 'manhattan', 'n_neighbors': 4}
0.562 (+/-0.052) for {'metric': 'manhattan', 'n_neighbors': 5}
0.553 (+/-0.097) for {'metric': 'manhattan', 'n_neighbors': 6}
0.584 (+/-0.124) for {'metric': 'manhattan', 'n_neighbors': 7}
0.566 (+/-0.158) for {'metric': 'manhattan', 'n_neighbors': 8}
0.560 (+/-0.161) for {'metric': 'manhattan', 'n_neighbors': 9}
0.561 (+/-0.095) for {'metric': 'manhattan', 'n_neighbors': 10}
0.490 (+/-0.152) for {'metric': 'chebyshev', 'n_neighbors': 1}
0.521 (+/-0.128) for {'metric': 'chebyshev', 'n_neighbors': 2}
0.575 (+/-0.146) for {'metric': 'chebyshev', 'n_neighbors': 3}
0.539 (+/-0.090) for {'metric': 'chebyshev', 'n_neighbors': 4}
0.576 (+/-0.095) for {'metric': 'chebyshev', 'n_neighbors': 5}
0.518 (+/-0.087) for {'metric': 'chebyshev', 'n_neighbors': 6}
0.531 (+/-0.100) for {'metric': 'chebyshev', 'n_neighbors': 7}
0.518 (+/-0.068) for {'metric': 'chebyshev', 'n_neighbors': 8}
0.520 (+/-0.086) for {'metric': 'chebyshev', 'n_neighbors': 9}
0.539 (+/-0.070) for {'metric': 'chebyshev', 'n_neighbors': 10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

                     precision    recall  f1-score   support

no-recurrence-events       0.69      0.92      0.79        49
   recurrence-events       0.43      0.13      0.20        23

            accuracy                           0.67        72
           macro avg       0.56      0.52      0.49        72
        weighted avg       0.61      0.67      0.60        72

[[45  4]
 [20  3]]

----------------------------------------
Trying model Random Forest
Best parameters set found on train set:

{'max_depth': 8}
```

```
Grid scores on train set:

0.533 (+/-0.082) for {'max_depth': 1}
0.604 (+/-0.076) for {'max_depth': 2}
0.587 (+/-0.130) for {'max_depth': 3}
0.595 (+/-0.111) for {'max_depth': 4}
0.599 (+/-0.194) for {'max_depth': 5}
0.590 (+/-0.115) for {'max_depth': 6}
0.580 (+/-0.117) for {'max_depth': 7}
0.614 (+/-0.110) for {'max_depth': 8}
0.560 (+/-0.087) for {'max_depth': 9}
0.600 (+/-0.112) for {'max_depth': 10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

                      precision    recall  f1-score   support

no-recurrence-events       0.73      0.96      0.83        49
   recurrence-events       0.75      0.26      0.39        23

            accuracy                           0.74        72
           macro avg       0.74      0.61      0.61        72
        weighted avg       0.74      0.74      0.69        72

[[47  2]
 [17  6]]

----------------------------------------
Trying model Adaboost
Best parameters set found on train set:

{'learning_rate': 0.01}

Grid scores on train set:

0.586 (+/-0.146) for {'learning_rate': 1.0}
0.620 (+/-0.102) for {'learning_rate': 0.1}
0.643 (+/-0.161) for {'learning_rate': 0.01}
0.567 (+/-0.086) for {'learning_rate': 0.001}
0.567 (+/-0.086) for {'learning_rate': 0.0001}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
```

The scores are computed on the full test set.

```
                      precision    recall  f1-score   support

no-recurrence-events       0.72      0.98      0.83        49
   recurrence-events       0.80      0.17      0.29        23

            accuracy                           0.72        72
           macro avg       0.76      0.58      0.56        72
        weighted avg       0.74      0.72      0.65        72

[[48  1]
 [19  4]]

Summary of results for recall_macro
Estimator
Decision Tree            - score: 0.62%
Gaussian Naive Bayes     - score: 0.63%
Linear Perceptron        - score: 0.56%
Support Vector           - score:  0.6%
K Nearest Neighbor       - score: 0.58%
Random Forest            - score: 0.61%
Adaboost                 - score: 0.64%
```