Preprocessing: transform categorical data

by Claudio Sartori

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.

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

In [1]:

```python
"""
http://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html
@author: scikit-learn.org and Claudio Sartori
"""
import warnings
warnings.filterwarnings('ignore') # uncomment this line to suppress warnings

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer

from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

print(__doc__) # print information included in the triple quotes at the beginning

random_state = 42
```

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

In [2]:
```
# url = 'diagnosis.data'
# names = ['Temp', 'Nau', 'Lum', 'Uri', 'Mic', 'Bur', 'd1', 'd2']
# sep = "\t"
url = 'breast-cancer.data'
names = ['Class','age','menopause','tumor-size','inv-nodes',
         'node-caps','deg-malig','breast','breast-quad','irradiat']
sep = ","

df = pd.read_csv(url, names = names, sep=sep)
df.head()
```

Out[2]:

	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	no-recurrence-events	30-39	premeno	30-34	0-2	no	3	left	left_low	no
1	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	right_up	no
2	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	left	left_low	no
3	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	right	left_up	no
4	no-recurrence-events	40-49	premeno	0-4	0-2	no	2	right	right_low	no

Show the types of the columns

In [3]:
```
print(df.dtypes)
```

```
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`

In [4]:
```
tumor_size_dict = dict(zip(list(df['tumor-size'].unique()),list(df['tumor-size'].unique())))
tumor_size_dict
```

Out[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'}
```

In [5]:
```
tumor_size_dict['0-4'] = '00-04'
tumor_size_dict['5-9'] = '05-09'
```

In [6]:
```
df['tumor-size'] = df['tumor-size'].map(tumor_size_dict)
```

Clean the column `inv-nodes`

In [7]:
```
inv_nodes_dict = dict(zip(list(df['inv-nodes'].unique()),list(df['inv-nodes'].unique())))
```

In [8]:
```
inv_nodes_dict['0-2']  = '00-02'
inv_nodes_dict['3-5']  = '03-05'
inv_nodes_dict['6-8']  = '06-08'
inv_nodes_dict['9-11'] = '09-11'
```

In [9]:
```
df['inv-nodes'] = df['inv-nodes'].map(inv_nodes_dict)
```

Inspect the data

In [10]:
```
df.head()
```

Out[10]:

	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	no-recurrence-events	30-39	premeno	30-34	00-02	no	3	left	left_low	no
1	no-recurrence-events	40-49	premeno	20-24	00-02	no	2	right	right_up	no
2	no-recurrence-events	40-49	premeno	20-24	00-02	no	2	left	left_low	no
3	no-recurrence-events	60-69	ge40	15-19	00-02	no	2	right	left_up	no
4	no-recurrence-events	40-49	premeno	00-04	00-02	no	2	right	right_low	no

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

In [11]:
```

```

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

In [12]:
```

```

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

In [13]:
```

```

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

In [14]:
```

```

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

Prepare the transformer

In [15]:
```
# transf_dtype = np.float64
transf_dtype = np.int32

categorical_transformer = OneHotEncoder(handle_unknown='ignore', sparse = False, dtype = transf_dtype)
ordinal_transformer = OrdinalEncoder(dtype = transf_dtype)
preprocessor = ColumnTransformer(
    transformers = [('cat', categorical_transformer, categorical_features),
                    ('ord', ordinal_transformer, ordinal_features)
                   ],
                    remainder = 'passthrough'
    )
```

Split X and y and check the shapes

In [16]:

In [17]:

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

In [18]:

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

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

In [19]:

Out[19]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('cat',
                                 OneHotEncoder(dtype=<class 'numpy.int32'>,
                                               handle_unknown='ignore',
                                               sparse=False),
                                 ['irradiat', 'breast-quad', 'menopause',
                                  'node-caps', 'breast']),
                                ('ord',
                                 OrdinalEncoder(dtype=<class 'numpy.int32'>),
                                 ['age', 'tumor-size', 'inv-nodes'])])
```

In [20]:

```
{'cat': OneHotEncoder(dtype=<class 'numpy.int32'>, handle_unknown='ignore',
              sparse=False), 'ord': OrdinalEncoder(dtype=<class 'numpy.int32'>), 'remainder': 'passthrough'}
```

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

In [21]:

In [22]:

Out[22]:

```
(286, 20)
```

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

In [23]:

In [24]:

Out[24]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	
count	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.00(
mean	0.762238	0.237762	0.003497	0.073427	0.384615	0.339161	0.083916	0.115385	0.451049	0.024476	0.524476	0.027972	0.776224	0.195
std	0.426459	0.426459	0.059131	0.261293	0.487357	0.474254	0.277748	0.320046	0.498470	0.154791	0.500276	0.165182	0.417504	0.397
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00(
25%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.00(
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	0.00(
75%	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	0.00(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00(

In [25]:

Out[25]:

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

In [26]:

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

Train/test split

In [27]:

Classification and test

In [28]:

In [29]:

In [30]:

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

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

{'max_depth': 2}

Grid scores on train set:

0.567 (+/-0.073) for {'max_depth': 1}
0.601 (+/-0.119) for {'max_depth': 2}
0.590 (+/-0.127) for {'max_depth': 3}
0.590 (+/-0.164) for {'max_depth': 4}
0.547 (+/-0.109) for {'max_depth': 5}
0.551 (+/-0.076) for {'max_depth': 6}
0.589 (+/-0.184) for {'max_depth': 7}
0.564 (+/-0.194) for {'max_depth': 8}
0.560 (+/-0.219) for {'max_depth': 9}
```