ex2-data-preprocessing

August 5, 2024

1 Importing Libraries

We begin by importing the necessary libraries for numerical operations and data manipulation.

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

2 Loading the Dataset

Here, we load the dataset from a CSV file and separate the features (independent variables) from the target (dependent variable).

[2]:
```
dataset = pd.read_csv("datasets/ShopSellData.csv")
```

3 Splitting the inputs and outputs

`dataset.iloc` can be used to index into rows and columns using integers.

The first parameter is rows and the second parameter is columns.

`dataset.iloc[:, :-1]` indexes all rows and columns from 0 to before last, thus excluding the last one.

if column names are known, then we can also use `dataset.loc` to perform indexing using only column names.

It is notable that using column names will index inclusively (last column is included in result unlike integers).

[3]:
```
x = dataset.loc[:, :"Salary"].values
```

[4]:
```
x
```

[4]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
```

```
        ['France', 48.0, 79000.0],
        ['Germany', 50.0, 83000.0],
        ['France', 37.0, 67000.0]], dtype=object)
```

[5]:
```
dataset.iloc[:, :-1]
```

[5]:
```
    Country   Age   Salary
0    France  44.0  72000.0
1     Spain  27.0  48000.0
2   Germany  30.0  54000.0
3     Spain  38.0  61000.0
4   Germany  40.0      NaN
5    France  35.0  58000.0
6     Spain   NaN  52000.0
7    France  48.0  79000.0
8   Germany  50.0  83000.0
9    France  37.0  67000.0
```

[6]:
```
y = dataset.iloc[:, 3].values
```

[7]:
```
dataset.iloc[:, 3]
```

[7]:
```
0     No
1    Yes
2     No
3     No
4    Yes
5    Yes
6     No
7    Yes
8     No
9    Yes
Name: Purchased, dtype: object
```

4 Viewing the Dataset

We display the entire dataset and a quick overview of the first two rows to understand its structure.

[8]:
```
dataset
```

[8]:
```
    Country   Age   Salary Purchased
0    France  44.0  72000.0        No
1     Spain  27.0  48000.0       Yes
2   Germany  30.0  54000.0        No
3     Spain  38.0  61000.0        No
4   Germany  40.0      NaN       Yes
5    France  35.0  58000.0       Yes
```

```
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

[9]:
```
dataset.head()
```

[9]:
```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
```

[10]:
```
dataset.head(2)
```

[10]:
```
  Country   Age   Salary Purchased
0  France  44.0  72000.0        No
1   Spain  27.0  48000.0       Yes
```

[11]:
```
dataset.tail(2)
```

[11]:
```
   Country   Age   Salary Purchased
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

5 Label Encoding

Label Encoding is used to convert categorical data into numeric form. Here, we encode the 'Country' column.

`LabelEncoder` is a class we import from scikit-learn (imported as `sklearn`) library

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

[13]:
```
label_encode_x = LabelEncoder()
```

[14]:
```
x[:, 0] = label_encode_x.fit_transform(x[:, 0])
```

[15]:
```
x
```

[15]:
```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, nan],
       [0, 35.0, 58000.0],
```

```
       [2, nan, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

6 One Hot Encoding

One Hot Encoding is used to create dummy variables for categorical data. This step ensures that the encoded categorical data does not imply any ordinal relationship.

[16]:
```
from sklearn.preprocessing import OneHotEncoder
```

[17]:
```
onehotencoder = OneHotEncoder()
```

[18]:
```
onehotencoder.fit_transform(x[:, 0].reshape(-1, 1)).toarray()
```

[18]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

7 Encoding the Target Variable

Similar to the feature encoding, we encode the target variable 'Purchased' to convert it into numeric form.

[19]:
```
label_encode_y = LabelEncoder()
```

[20]:
```
y = label_encode_y.fit_transform(y)
```

[21]:
```
y
```

[21]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

8 Splitting the Dataset

We split the dataset into training and test sets. This allows us to train our model on one set of data and test it on another to evaluate its performance.

[22]:
```
from sklearn.model_selection import train_test_split
```

[23]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=0)
```

[24]:
```
x_train
```

[24]:
```
array([[1, 40.0, nan],
       [0, 37.0, 67000.0],
       [2, 27.0, 48000.0],
       [2, nan, 52000.0],
       [0, 48.0, 79000.0],
       [2, 38.0, 61000.0],
       [0, 44.0, 72000.0],
       [0, 35.0, 58000.0]], dtype=object)
```

[25]:
```
x_test
```

[25]:
```
array([[1, 30.0, 54000.0],
       [1, 50.0, 83000.0]], dtype=object)
```

[26]:
```
y_train
```

[26]:
```
array([1, 1, 1, 0, 1, 0, 0, 1])
```

[27]:
```
y_test
```

[27]:
```
array([0, 0])
```

9 Feature Scaling

Feature scaling is performed to standardize the range of independent variables. It ensures that each feature contributes equally to the model.

[28]:
```
from sklearn.preprocessing import StandardScaler
```

[29]:
```
sc_x = StandardScaler()
```

[30]:
```
x_train_scaled = sc_x.fit_transform(x_train)
```

[31]:
```
x_test_scaled = sc_x.transform(x_test)
```

[32]:
```
x_train_scaled
```

[32]:
```
array([[ 0.13483997,  0.25315802,         nan],
       [-0.94387981, -0.23014365,  0.44897083],
       [ 1.21355975, -1.84114924, -1.41706417],
       [ 1.21355975,         nan, -1.0242147 ],
       [-0.94387981,  1.54196248,  1.62751925],
```

```
       [ 1.21355975, -0.0690431 , -0.14030338],
       [-0.94387981,  0.89756025,  0.94003267],
       [-0.94387981, -0.55234477, -0.43494049]])
```

[33]:
```
x_test_scaled
```

[33]:
```
array([[ 0.13483997, -1.35784756, -0.82778996],
       [ 0.13483997,  1.8641636 ,  2.02036872]])
```