Data Preprocessing

Prof. Shibdas Dutta,

Associate Professor,

DCG DATA CORE SYSTEMS INDIA PVT LTD

Kolkata

Data Preprocessing

- Introduction

- Why Data Pre - processing?
- Data Pre - processing Techniques
- Normalization
- Binarization
- Standardization
- Data Labeling
- Label Encoding

Introduction

- Machine Learning algorithms are completely dependent on data because it is the most crucial aspect that makes model training possible.

- On the other hand, if we won’t be able to make sense out of that data, before feeding it to ML algorithms, a machine will be useless.

- In simple words, we always need to feed right data i.e. the data in **correct scale, format and containing meaningful features**, for the problem we want machine to solve.

- This makes data preparation the most important step in ML process. Data preparation may be defined as the procedure that makes our dataset more appropriate for ML process.

Why Data Pre - processing?

- After selecting the raw data for ML training, the most important task is data pre- processing.

- In broad sense, data preprocessing will convert the selected data into a form we can work with or can feed to ML algorithms.

- We always need to preprocess our data so that it can be as per the expectation of machine learning algorithm.

Data Pre - processing Techniques

- We have the following data preprocessing techniques that can be applied on data set to produce data for ML algorithms:
- Scaling:
- Most probably our dataset comprises of the attributes with varying scale, but we cannot provide such data to ML algorithm hence it requires rescaling.
- Data rescaling makes sure that attributes are at same scale.
- Generally, attributes are rescaled into the range of 0 and 1.
- ML algorithms like gradient descent and k-Nearest Neighbors requires scaled data.
- We can rescale the data with the help of MinMaxScaler class of scikit-learn Python library.

Example

- In this example we will rescale the data of Pima Indians Diabetes dataset which we used earlier.

- First, the CSV data will be loaded (as done in the previous chapters) and then with the help of MinMaxScaler class, it will be rescaled in the range of 0 and 1.

- The first few lines of the following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv

from numpy import set_printoptions

from sklearn import preprocessing

path = r'C:\pima-indians-diabetes.csv'
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']

dataframe = read_csv(path, names=names)

array = dataframe.values
```

Now, we can use **`MinMaxScaler`** class to rescale the data in the range of 0 and 1.

```
data_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))

data_rescaled = data_scaler.fit_transform(array)
```

We can also summarize the data for output as per our choice.

Here, we are setting the precision to 1 and showing the first 10 rows in the output.

```
set_printoptions(precision=1)
print ("\nScaled data:\n", data_rescaled[0:10])
```

Output

```
Scaled data:
[[0.4 0.7 0.6 0.4 0.  0.5 0.2 0.5 1. ]


 [0.1 0.4 0.5 0.3 0.  0.4 0.1 0.2 0.  ]
 [0.5 0.9 0.5 0.  0.  0.3 0.3 0.2 1.  ]
 [0.1 0.4 0.5 0.2 0.1 0.4 0.  0.  0.  ]
 [0.  0.7 0.3 0.4 0.2 0.6 0.9 0.2 1.  ]
 [0.3 0.6 0.6 0.  0.  0.4 0.1 0.2 0.  ]
 [0.2 0.4 0.4 0.3 0.1 0.5 0.1 0.1 1.  ]
 [0.6 0.6 0.  0.  0.  0.5 0.  0.1 0.  ]
 [0.1 1.  0.6 0.5 0.6 0.5 0.  0.5 1.  ]
 [0.5 0.6 0.8 0.  0.  0.  0.1 0.6 1. ]]
```

From the above output, all the data got rescaled into the range of 0 and 1.

Normalization

Another useful data preprocessing technique is Normalization.

This is used to rescale each row of data to have a length of 1.

It is mainly useful in Sparse dataset where we have lots of zeros.

We can rescale the data with the help of **`Normalizer`** class of **`scikit-learn`** Python library.

Types of Normalization

In machine learning, there are two types of normalization preprocessing techniques as follows:

1. L1 Normalization
2. L2 Normalization

L1 Normalization

- It may be defined as the normalization technique that modifies the dataset values in a way that in each row the sum of the absolute values will always be up to 1. It is also called Least Absolute Deviations.

Example

In this example, we use L1 Normalize technique to normalize the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded and then with the help of **`Normalizer`** class it will be normalized.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv

from numpy import set_printoptions

from sklearn.preprocessing import Normalizer

path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']

dataframe = read_csv (path, names=names)

array = dataframe.values
```

Now, we can use **`Normalizer`** class with L1 to normalize the data.

```
Data_normalizer = Normalizer(norm='l1').fit(array)

Data_normalized = Data_normalizer.transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 2 and showing the first 3 rows in the output.

```
set_printoptions(precision=2)
print ("\nNormalized data:\n", Data_normalized [0:3])
```

Output

```
 Normalized data:
[[0.020.430.210.1              0.       0.1   0.     0.140.          ]
[0.          0.360.280.12      0.       0.11  0.     0.130.          ]


[0.030.590.210.                0.       0.07  0.     0.10.          ]]
```

L2 Normalization

It may be defined as the normalization technique that modifies the dataset values in a way that in each row the sum of the squares will always be up to 1. It is also called least squares.

Example

In this example, we use L2 Normalization technique to normalize the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded (as done in previous chapters) and then with the help of **`Normalizer`** class it will be normalized.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from numpy import set_printoptions

from sklearn.preprocessing import Normalizer
```

```
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']

dataframe = read_csv (path, names=names)

array = dataframe.values
```

Now, we can use **Normalizer** class with L2 to normalize the data.

```
Data_normalizer = Normalizer(norm='l2').fit(array)

Data_normalized = Data_normalizer.transform(array)
```

We can also summarize the data for output as per our choice.

Here, we are setting the precision to 2 and showing the first 3 rows in the output.

```
set_printoptions(precision=2)
print ("\nNormalized data:\n", Data_normalized [0:3])
```

Output

```
Normalized data:
 [[0.030.830.4        0.20.    0.190.               0.280.01]
 [0.010.720.56       0.240.    0.220.       0.260.]
 [0.040.920.32    0.     0.    0.120.             0.160.01]]
```

Binarization

As the name suggests, this is the technique with the help of which we can make our data binary.

We can use a binary threshold for making our data binary.

The values above that threshold value will be converted to 1 and below that threshold will be converted to 0.

For example, if we choose threshold value = 0.5, then the dataset value above it will become 1 and below this will become 0.

That is why we can call it **binarizing** the data or **thresholding** the data.

This technique is useful when we have probabilities in our dataset and want to convert them into crisp values.

We can binarize the data with the help of **`Binarizer`** class of **`scikit-learn`** Python library.

Example

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded and then with the help of **`Binarizer`** class it will be converted into binary values i.e. 0 and 1 depending upon the threshold value. We are taking 0.5 as threshold value.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv

from sklearn.preprocessing import Binarizer

path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']

dataframe = read_csv(path, names=names)

array = dataframe.values
```

Now, we can use **`Binarize`** class to convert the data into binary values.

```
binarizer = Binarizer(threshold=0.5).fit(array)

Data_binarized = binarizer.transform(array)
```

Here, we are showing the first 5 rows in the output.

Output

```
Binary data:
[[1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [1. 1. 1. 1. 0. 1. 0. 1.  0.]
 [1. 1. 1. 0. 0. 1. 1. 1.  1.]
 [1. 1. 1. 1. 1. 1. 0. 1.  0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

Standardization

Another useful data preprocessing technique which is basically used to transform the data attributes with a Gaussian distribution.

It differs the mean and SD (Standard Deviation) to a standard Gaussian distribution with a mean of 0 and a SD of 1.

This technique is useful in ML algorithms like linear regression, logistic regression that assumes a Gaussian distribution in input dataset and produce better results with rescaled data.

We can standardize the data (mean = 0 and SD =1) with the help of **`StandardScaler`** class of **`scikit-learn`** Python library.

Example

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier.

First, the CSV data will be loaded and then with the help of **`StandardScaler`** class it will be converted into Gaussian Distribution with mean = 0 and SD = 1.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from sklearn.preprocessing import StandardScaler

from pandas import read_csv

from numpy import set_printoptions

path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']

dataframe = read_csv(path, names=names)

array = dataframe.values
```

Now, we can use **`StandardScaler`** class to rescale the data.

```
data_scaler = StandardScaler().fit(array)

data_rescaled = data_scaler.transform(array)
```

We can also summarize the data for output as per our choice.

Here, we are setting the precision to 2 and showing the first 5 rows in the output.

```
set_printoptions(precision=2)
print ("\nRescaled data:\n", data_rescaled [0:5])
```

Output

```
Rescaled data:
[[ 0.64  0.85  0.15  0.91 -0.69  0.2       0.47  1.43  1.37]
 [-0.84  -1.12 -0.16  0.53   -0.69 -0.68 -0.37  -0.19  -0.73]
  [1.23    1.94 -0.26 -1.29  -0.69  -1.1   0.6   -0.11  1.37]
 [-0.84 -1.     -0.16  0.15    0.12 -0.49 -0.92  -1.04  -0.73]
[-1.14  0.5  -1.5      0.91  0.77  1.41  5.48 -0.02  1.37]]
```

Data Labeling

- We discussed the importance of good data for ML algorithms as well as some techniques to pre-process the data before sending it to ML algorithms.
- One more aspect in this regard is data labeling.
- It is also very important to send the data to ML algorithms having proper labeling.
- For example, in case of classification problems, lot of labels in the form of words, numbers etc. are there on the data.

What is Label Encoding?

Most of the **sklearn** functions expect that the data with number labels rather than word labels.

Hence, we need to convert such labels into number labels.

This process is called label encoding.

We can perform label encoding of data with the help of **LabelEncoder()** function of **scikit-learn** Python library.

Example

In the following example, Python script will perform the label encoding. First, import the required Python libraries as follows:

```
import numpy as np

from sklearn import preprocessing
```

Now, we need to provide the input labels as follows:

```
input_labels = ['red','black','red','green','black','yellow','white']
```

The next line of code will create the label encoder and train it.

```
encoder = preprocessing.LabelEncoder()

encoder.fit(input_labels)
```

The next lines of script will check the performance by encoding the random ordered list:

```
test_labels = ['green','red','black']

encoded_values = encoder.transform(test_labels)

print("\nLabels =", test_labels)

print("Encoded values =", list(encoded_values))

encoded_values = [3,0,4,1]

decoded_list = encoder.inverse_transform(encoded_values)
```

We can get the list of encoded values with the help of following python script:

```
print("\nEncoded values =", encoded_values)

print("\nDecoded labels =", list(decoded_list))
```

Output

```
Labels = ['green', 'red', 'black']

Encoded values = [1, 2, 0]


Encoded values = [3, 0, 4, 1]


Decoded labels = ['white', 'black', 'yellow', 'green']
```

Thank You