Foundations of Data Science Using Python

Session 4: Data Pre-processing using Python

Machine Learning – Preparing Data

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 Machine Learning 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 Machine Learning training, the most important task is data pre-processing. In broad sense, data pre-processing will convert the selected data into a form we can work with or can feed to Machine Learning algorithms. We always need to pre-process our data so that it can be as per the expectation of machine learning algorithm. Machine Learning algorithms don't work so well with processing raw data. Before we can feed such data to Machine Learning algorithm, we must pre-process it. In other words, we must apply some transformations on it. With data pre-processing we convert raw data into a clean data set.

Data Pre-processing Techniques

We have the following data pre-processing techniques that can be applied on data set to produce data for Machine Learning algorithms:

1. Rescaling Data
2. Standardizing Data
3. Normalizing Data
4. Binarizing Data
5. Mean Removal
6. One Hot Encoding
7. Label Encoding

Some Machine Learning models need information to be in a specified format. For instance, the Random Forest Algorithm does not take null values. To pre-process data, we will use the library scikit-learn or sklearn.

Rescaling Data

Most probably our dataset comprises of the attributes with varying scale, but we cannot provide such data to Machine Learning 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. Machine Learning 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 Wine Quality dataset. First, the CSV data will be loaded and then with the help of MinMaxScaler class, it will be rescaled in the range of 0 and 1.

Implementation in Python

Data-Preprocessing.py

```
#for rescaling the data
import pandas, scipy, numpy
from sklearn.preprocessing import MinMaxScaler

#code for rescaling the data
df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
array=df.values
#Separating data into input and output components
x=array[:,0:8]
y=array[:,8]
scaler=MinMaxScaler(feature_range=(0,1))
rescaledX=scaler.fit_transform(x)
numpy.set_printoptions(precision=3) #Setting precision for the output
print(rescaledX[0:5,:])
```

Input: The winequality-red.csv file

Output: Values scaled between 0 and 1.

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
[[0.248 0.397 0.    0.068 0.107 0.141 0.099 0.568]
 [0.283 0.521 0.    0.116 0.144 0.338 0.216 0.494]
 [0.283 0.438 0.04  0.096 0.134 0.197 0.17  0.509]
 [0.584 0.11  0.56  0.068 0.105 0.225 0.191 0.582]
 [0.248 0.397 0.    0.068 0.107 0.141 0.099 0.568]]
```

Rescaling the data is useful for neural networks, optimization algorithms and those that use distance measures like k-nearest neighbors and weight inputs like regression.

Standardizing Data

Standardizing basically used to transform the data attributes with a Gaussian distribution. It differs the mean and standard deviation to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. With standardizing, we can take attributes with a Gaussian distribution and different means and standard deviations and transform them into a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. This technique is useful in Machine Learning 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 standard deviation = 1) with the help of StandardScaler class of scikit-learn Python library.

Example

In this example, we will rescale the data of Wine Quality 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 standard deviation = 1.

Implementation in Python

Data-Preprocessing.py

```
#for standadizing the data
import pandas, scipy, numpy
from sklearn.preprocessing import StandardScaler

df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
array=df.values
#Separating data into input and output components
x=array[:,0:8]
y=array[:,8]

#code for standadizing the data
scaler=StandardScaler().fit(x)
rescaledX=scaler.transform(x)
print(rescaledX[0:5,:])
```

Input: The winequality-red.csv file

Output: Data transformed into a standard Gaussian distribution with mean 0, SD 1

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
[[-0.52835961  0.96187667 -1.39147228 -0.45321841 -0.24370669 -0.46619252
  -0.37913269  0.55827446]
 [-0.29854743  1.96744245 -1.39147228  0.04341614  0.2238752   0.87263823
   0.62436323  0.02826077]
 [-0.29854743  1.29706527 -1.18607043 -0.16942723  0.09635286 -0.08366945
   0.22904665  0.13426351]
 [ 1.65485608 -1.38444349  1.4841536  -0.45321841 -0.26496041  0.10759209
   0.41150046  0.6642772 ]
 [-0.52835961  0.96187667 -1.39147228 -0.45321841 -0.24370669 -0.46619252
  -0.37913269  0.55827446]]
```

Normalizing Data

It is a 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 as Least Squares (L2 Normalization). In this task, we rescale each observation to a length 1 (a unit norm). For this we use Normalizer class of scikit-learn Python library.

Example:

In this example, we use Normalize technique to normalize the data of Wine Quality dataset which we used earlier. First, the CSV data will be loaded and then with the help of Normalizer class it will be normalized.

Implementation in Python

```
Data-Preprocessing.py
'''
#for normalizing the data
import pandas, scipy, numpy
from sklearn.preprocessing import Normalizer
df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
array=df.values
#Separating data into input and output components
x=array[:,0:8]
y=array[:,8]

#code for normalizing the data
scaler=Normalizer().fit(x)
normalizedX=scaler.transform(x)
print(normalizedX[0:5,:])
```

Output:

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
[[2.02389758e-01 1.91449771e-02 0.00000000e+00 5.19649378e-02
  2.07859751e-03 3.00849640e-01 9.29898888e-01 2.72897974e-02]
 [1.08339837e-01 1.22229560e-02 0.00000000e+00 3.61132790e-02
  1.36119283e-03 3.47243068e-01 9.30611421e-01 1.38452756e-02]
 [1.37698515e-01 1.34167784e-02 7.06146229e-04 4.06034082e-02
  1.62413633e-03 2.64804836e-01 9.53297409e-01 1.76006948e-02]
 [1.76658709e-01 4.41646772e-03 8.83293543e-03 2.99688881e-02
  1.18298242e-03 2.68142683e-01 9.46385939e-01 1.57415528e-02]
 [2.02389758e-01 1.91449771e-02 0.00000000e+00 5.19649378e-02
  2.07859751e-03 3.00849640e-01 9.29898888e-01 2.72897974e-02]]
```

Binarizing Data

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 Wine Quality 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. The threshold value considered in the example is 0.5.

Implementation in Python

```
Data-Preprocessing.py
#for binarizing the data
import pandas, scipy, numpy
from sklearn.preprocessing import Binarizer

df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
array=df.values
#Separating data into input and output components
x=array[:,0:8]
y=array[:,8]

#code for binarizing the data
binarizer=Binarizer(threshold=0.5).fit(x)
binaryX=binarizer.transform(x)
print(binaryX[0:5,:])
```

Output:

This marks 0 over all values equal to or less than 0.5, and marks 1 over the rest.

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
[[1. 1. 0. 1. 0. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1.]
 [1. 0. 1. 1. 0. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1.]]

F:\PYTHON\MyPythonPrograms>
```

Mean removal

It involves removing the mean from each feature so that it is centered on zero. Mean removal helps in removing any bias from the features.

Implementation in Python

```
Data-Preprocessing.py
65  #for mean removal
66  import pandas, scipy, numpy
67  from sklearn.preprocessing import scale
68
69  df=pandas.read_csv( 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv ',sep=';')
70  array=df.values
71  #Separating data into input and output components
72  x=array[:,0:8]
73  y=array[:,8]
74
75  #code for mean removal
76  data_standardized=scale(df)
77  print(data_standardized.mean(axis=0))
78  print(data_standardized.std(axis=0))
```

Output:

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
[ 3.55493551e-16  1.73303106e-16 -8.88733878e-17 -1.24422743e-16
  3.91042906e-16 -6.22113715e-17  4.44366939e-17  2.36403212e-14
  2.86172309e-15  6.75437748e-16  1.06648065e-16  8.88733878e-17]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

One Hot Encoding

It may be required to deal with numerical values that are few and scattered, and you may not need to store these values. In such situations you can use One Hot Encoding technique. If the number of distinct values is **k**, it will transform the feature into a k-dimensional vector where only one value is 1 and all other values are 0.

Implementation in Python

```
Data-Preprocessing.py
81  #for One Hot Encoding
82  from sklearn.preprocessing import OneHotEncoder
83
84  #code for One Hot Encoding
85  encoder=OneHotEncoder()
86  encoder.fit([[0,2,1,2],
87  [1,3,5,3],
88  [2,3,2,12],
89  [1,2,4,3]
90  ])
91  print(encoder)
92  encoder_vector = encoder.transform([[2,3,5,3]]).toarray()
93  print(encoder_vector)
```

Output:

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,
              handle_unknown='error', sparse=True)
[[0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0.]]
```

In the example above, let us consider the third feature in each feature vector. The values are 1, 5, 2, and 4. There are four separate values here, which means the one-hot encoded vector will be of length 4. If we want to encode the value 5, it will be a vector [0, 1, 0, 0]. Only one value can be 1 in this vector. The second element is 1, which indicates that the value is 5.

Label Encoding

In supervised learning, we mostly come across a variety of labels which can be in the form of numbers or words. If they are numbers, then they can be used directly by the algorithm. However, many times, labels need to be in readable form. Hence, the training data is usually labelled with words. Label encoding refers to changing the word labels into numbers so that the algorithms can understand how to work on them. Let us understand in detail how to perform label encoding.

Implementation in Python

Data-Preprocessing.py

```
#Label Encoding
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford', 'bmw']
label_encoder.fit(input_classes)
print("\nClass mapping:  ")
for i, item in enumerate(label_encoder.classes_):
    print(item,'-->', i)
```

Output:

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py

Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3

F:\PYTHON\MyPythonPrograms>
```

As shown in the above output, the words have been changed into indexed numbers starting with 0. Now, when we deal with a set of labels, we can transform them as follows:

```
105  labels = ['toyota', 'ford', 'suzuki']
106  encoded_labels = label_encoder.transform(labels)
107  print("\nLabels =", labels)
108  print("Encoded labels =", list(encoded_labels))
```

Upon adding the above lines of code, after the execution, the following output can be observed.

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py

Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3

Labels = ['toyota', 'ford', 'suzuki']
Encoded labels = [3, 1, 2]

F:\PYTHON\MyPythonPrograms>
```

Label encoding is efficient than manually maintaining mapping between words and numbers. You can check by transforming numbers back to word labels as shown in the code below:

```
110  encoded_labels = [3, 2, 0, 2, 1]
111  decoded_labels = label_encoder.inverse_transform(encoded_labels)
112  print("\nEncoded labels =", encoded_labels)
113  print("Decoded labels =", list(decoded_labels))
```

Upon adding the above lines of code, after the execution, the following output can be observed.

```
F:\PYTHON\MyPythonPrograms>py Data-Preprocessing.py

Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3

Labels = ['toyota', 'ford', 'suzuki']
Encoded labels = [3, 1, 2]

Encoded labels = [3, 2, 0, 2, 1]
Decoded labels = ['toyota', 'suzuki', 'bmw', 'suzuki', 'ford']
```

From the output we can notice that the mapping is preserved perfectly.