Machine Learning: Data Preprocessing

Data Preprocessing. ***Source: Google Images***

It will be fun if we relate this with our real life. Remember the task of washing and then chopping raw vegetables before cooking a dish. It is somewhat similar to this task. Data Preprocessing is a way of converting raw form of data to a much more usable form. It looks similar right? The only difference between the two task is that one helps to cook tasty dish and other a Machine Learning model for prediction of classes of new observations. Indeed, the step of Data Preprocessing step would not sound interesting for most of the people but, it is the very first important step because every time you build a model you always have Data Preprocessing phase to work on. The step of Preprocessing should be done in such a way that the model can be trained in right way because the efficiency of the model is dependent on this step.

Steps Involved in Data Preprocessing: This step is completely dependent on some preprocessing tools:

1. **Importing Libraries:** We need to import basically 3 libraries(NumPy, Matplotlib, pandas) in the beginning to start the phase of preprocessing. But, what a library is? A library is symbol of module which contains functions and classes through which you can perform actions and operations.

i)*NumPy*- You need this library to work with array. As future model will require arrays for implementation.

ii)M*atplotlib*- You need this library to create some beautiful charts.

iii)*pandas*- This library is not only helpful to import the data set but, also create matrix of features and the dependent variable vectors.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

Importing Libraries.

2. **Importing The Dataset:** Here comes the steps of importing the data set, which is one of the important phase.

```
1 dataset = pd.read_csv('Data.csv')
```

Importing Dataset.

The dataset variable here acts as a data frame of rows and columns similar to the data set(Data.csv).

i) *Creating Matrix of Features and Dependent Variable Vector:*

```
[11]  1 X=dataset.iloc[:, :-1].values
      2 print(X)

[['India' 44.0 72000.0]
 ['China' 27.0 48000.0]
 ['America' 30.0 54000.0]
 ['India' 38.0 61000.0]
 ['America' 40.0 nan]
 ['India' 35.0 58000.0]
 ['China' nan 52000.0]
 ['India' 48.0 79000.0]
 ['America' 50.0 83000.0]
 ['India' 37.0 67000.0]]

  1 Y = dataset.iloc[:, -1].values
  2 print(Y)

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Features of Matrices.

All we need to know is to play with the indexes of Matrix and we can easily create these two feature matrices. X here acts as independent variable matrix and Y as dependent variable vector. These two feature matrices will be the first need of the furthers steps of preprocessing.

3. **Take Care of Missing Data:** Do look out the above picture and the rows containing value NaN which signify that the values are missing and it can affect the learning of the model.

```
[11]  1 from sklearn.impute import SimpleImputer
      2 imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
      3 imputer.fit(X[:, 1:3])
      4 X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
  1 print(X)
```

```
[['India' 44.0 72000.0]
 ['China' 27.0 48000.0]
 ['America' 30.0 54000.0]
 ['India' 38.0 61000.0]
 ['America' 40.0 63777.77777777778]
 ['India' 35.0 58000.0]
 ['China' 38.77777777777778 52000.0]
 ['India' 48.0 79000.0]
 ['America' 50.0 83000.0]
 ['India' 37.0 67000.0]]
```

Missing Values are Replaced With Mean Values.

It could be done in many ways like Deleting/Ignoring the rows with missing values. But, it is beneficial when the number of rows with missing values are at most 1%.

The other ways are replacing the missing values with median or mean value of that column.

But, the most preferred is replacing with average sum value(**Mean**) of the column. It is done with the help of best data science library scikit-learn.

4. **Encoding Categorical Data:** The above feature matrices contains the column with string values which is tough to handle and work with. As the computers works well with binary values. So, we will convert string values into binary values.

i) *Encoding The Independent Variables:*

```
[13]  1 from sklearn.compose import ColumnTransformer
      2 from sklearn.preprocessing import OneHotEncoder
      3 ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
      4 X = np.array(ct.fit_transform(X))

      1 print(X)

[[0.0 0.0 1.0 44.0 72000.0]
 [0.0 1.0 0.0 27.0 48000.0]
 [1.0 0.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [1.0 0.0 0.0 40.0 63777.77777777778]
 [0.0 0.0 1.0 35.0 58000.0]
 [0.0 1.0 0.0 38.77777777777778 52000.0]
 [0.0 0.0 1.0 48.0 79000.0]
 [1.0 0.0 0.0 50.0 83000.0]
 [0.0 0.0 1.0 37.0 67000.0]]
```

Encoding of Independent variables.

We have three categories of data so, the encoded values are:

“India as 0, 0, 0", “China as 0, 0, 1”, and “America as 1, 0, 0”.

ii) *Encoding The Dependent Variables:*

```
[15]  1 from sklearn.preprocessing import LabelEncoder
      2 le = LabelEncoder()
      3 y = le.fit_transform(y)

      1 print(y)

[0 1 0 0 1 1 0 1 0 1]
```

Encoding of Dependent variables.

Here, we have two categories of values so, the encoded values are:

“No as 0” and “Yes as 1”

5. **Splitting of Dataset into Training set and Test set:** In this step we work on generalization. The purpose of splitting to avoid overfitting. **Overfitting** happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. It optimizes the training accuracy. We need our model to predict well the outcome of new observation(Test set), which is similar to training set.

```
[17]  1 from sklearn.model_selection import train_test_split
      2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)

[18]  1 print(X_train)

[[0.0 1.0 0.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 40.0 63777.77777777778]
 [0.0 0.0 1.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 27.0 48000.0]
 [0.0 0.0 1.0 48.0 79000.0]
 [1.0 0.0 0.0 50.0 83000.0]
 [0.0 0.0 1.0 35.0 58000.0]]

[21]  1 print(X_test)

[[1.0 0.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 37.0 67000.0]]

[22]  1 print(y_train)

[0 1 0 0 1 1 0 1]

      1 print(y_test)

[0 1]
```

Splitting of Dataset into 80% Training set and 20% Test set.

6. **Feature Scaling:**

```
[24]  1 from sklearn.preprocessing import StandardScaler
      2 sc = StandardScaler()
      3 X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
      4 X_test[:, 3:] = sc.transform(X_test[:, 3:])

[25]  1 print(X_train)

[[0.0 1.0 0.0 -0.19159184384578545 -1.0781259408412425]
 [1.0 0.0 0.0 -0.014117293757057777 -0.07013167641635372]
 [0.0 0.0 1.0 0.566708506533324 0.633562432710455]
 [0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867]
 [0.0 1.0 0.0 -1.9018011447007988 -1.420463615551582]
 [0.0 0.0 1.0 1.1475343068237058 1.232653363453549]
 [1.0 0.0 0.0 1.4379472069688968 1.5749910381638885]
 [0.0 0.0 1.0 -0.7401495441200351 -0.5646194287757332]]

      1 print(X_test)

[[1.0 0.0 0.0 -1.4661817944830124 -0.9069571034860727]
 [0.0 0.0 1.0 -0.44973664397484414 0.2056403393225306]]
```

Feature Scaling is important to bring the variety of data in the same scale. It involves the step of normalising the data in the particular range.

The two most commonly used scaling techniques are Normalization and Standardization. But, we prefer normalization technique.

Conclusion: *The above mentioned steps are really important to work with as the data preprocessing is one of the standing pillar of the Model building process. I hope the mentioned details will help many of the beginners to develop the understanding of data preprocessing.*