DATA PREPROCESSING

Prepared By,
Heinek Halwin Roshan

Introduction

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. **Data preprocessing** is a proven method of resolving such issues

Normalization and Standardization

Normalization rescales the values into a range of [0,1]. This might be useful in some cases where all parameters need to have the same positive scale. However, the outliers from the data set are lost.

$$Xchanged = X - Xmin \,/\, Xmax - Xmin$$

Standardization rescales data to have a mean (μ) of 0 and standard deviation (σ) of 1 (unit variance).

$$Xchanged = X - \mu \,/\, \sigma$$

For most applications standardization is recommended.

Data Preprocessing Example

- **Get the Dataset**

Dataset:

Name	Country	Age	Salary	Purchased
John	France	44	72000	No
Wick	Spain	27		Yes
Shawn	Germany	30	54000	No
Mendez	Germany		30000	Yes
Casey	France	48	61000	Yes
Neistat	France	41		No
Mike	Spain	29		Yes
Shinoda	Germany		71000	Yes

From the above dataset, we can see that it is a dataset regarding purchase of a particular product.

First of all, we can ignore the column “Name” from our dataset, as the name of the individual does not contribute to the purchase decision of the product. So, the remaining dataset is:

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27		Yes
Germany	30	54000	No
Germany		30000	Yes
France	48	61000	Yes
France	41		No
Spain	29		Yes
Germany		71000	Yes

- **Dependent and Independent Variables**

From the above dataset, the columns ‘Country’, ‘Age’ and ‘Salary’ are the **independent variables**; since their values will not fluctuate if there is a change in the other values of the same row.

The column ‘Purchased’ is the **dependent variable**, since the purchase decision will be influenced by the other independent variables.

Now, since we have identified the dependent variable and independent variable from our dataset. We have to split the dataset accordingly.

Code:

```
>>> # First import your necessary packages
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>>
>>> # Now read the dataset
>>> Dataset = pd.read_csv(‘Data.csv’)
>>>
>>> # Now split the dataset into Dependent variables and Independent variables
>>> X = Dataset.iloc[:, :-1].values       # Contains the independent variables
>>> Y = Dataset.iloc[:, 3].values          #Contains the dependent variables
```

Code explanation:

[: , :-1] or [: , 3] – Here, the first parameter is to specify the number of lines to read and the second specifies the indexes from the dataset. So by leaving the first parameter empty, python will read all the lines from the csv.

[: , :-1] - Here, we have used negative indexing, so python will read the dataset, till the second last column, i.e, from Country till Salary.

[: , 3] - Here, we have used positive indexing, so python will take the fourth column, i.e. Purchased.

- **Missing Data**

In any dataset, there can be missing values. This can either be seen as a blank space or “NaN” in the dataset. Having missing data in the dataset, will be problematic for the algorithm we are trying to develop or the visualisation we are trying to achieve.

We can tackle this problem in two ways:

1) First method is to remove the entire row from the dataset,
2) Second method is to fit a value in the missing places.

The first step is not recommended since there might be any vital information that we are removing from the dataset. So, we recommend using the second method.

Since, we have made the decision to use this method, lets see how we can do this using our python code:

Code:

```
>>> # Import the necessary packages
>>> from sklearn.preprocessing import Imputer
>>>
>>> # Fitting missing values
>>> imputer = Imputer( missing_values = ’NaN’, strategy = ‘mean’, axis = 0 )
>>> imputer = imputer.fit( X[ : , 1:3])
>>> X[ : , 1:3] = imputer.transform(X[ : , 1:3 ])
```

Code explanation:

imputer = Imputer(missing_values = ’NaN’, strategy = ‘mean’, axis = 0)

Here, *missing_values* is to specify what value to look for, we have specified “NaN”.
strategy is to specify which strategy to use to fit missing values; we are using mean to fit the values. Other viable options are median, most_frequent
axis is to specify along which axis to make the mean value of,
axis = 0 means along the column
axis = 1 means along the row

imputer = imputer.fit(X[: , 1:3])

This is to specify which all columns we should apply mean to

X[: , 1:3] = imputer.transform(X[: , 1:3])

This is to overwrite and transform only the columns which have values missing.

- **Categorical Variables**

Columns which have values other than numbers, i.e. either strings or characters can be called categorical variables.

Any machine learning algorithm is basically a mathematical equation and if we substitute a variable in this equation with a string value, even humans will find it diffcult to find the right answer, but if the value was a numerical value, we can find out the answer. So, we have to **encode** these categorical variables into numerical values.

From our dataset, we have to encode the Country column to numerical values.

Code:

```
>>> from sklearn.preprocessing import LabelEncoder
>>> labelencoder_x = LabelEncoder()
>>> X[ :, 0] = labelencoder_x.fit_transform(X[:, 0])
```

Code Explanation:

labelencoder_x = LabelEncoder()

Here, we are initialising an object *labelencoder_x* of the class *LabelEncoder.*

X[:, 0] = labelencoder_x.fit_transform(X[:, 0])

Here, we will encode the countries into numerical values and be overwritten to the first column.

Output:

The below output is just to show u what to expect after encoding. It is not the actual output of the above dataset after encoding.

Country (Actual)	Country (Encoded)
France	0
Spain	2
Germany	1

The problem with this encoding is that, in the above result. We got the values 0, 1 and 2. The algorithm will assume 1 is greater than 0 and 2 is greater than 1, this is not the case, since the values are representing countries. So, it will assume Spain is greater than Germany and France and this assumption will case problems with our dataset. So to prevent this, we will use **dummy encoding**.

Dummy Encoding

Country	France	Germany	Spain
France	1	0	0
Spain	0	0	1
Germany	0	1	0

Here, from the above table we can see that we converted the values 0,1,2 to 0 and 1. Basically, we created columns for each country and inserted 1 in the column where the country is present and 0 where the country is not present.

Code:

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> onehotencoder = OneHotEncoder(categorical_features = [0])
>>> X = onehotencoder.fit_transform(X).toarray()
```

Code Explanation:

onehotencoder = OneHotEncoder(categorical_features = [0])

Here, in the attribute 'categorical_features', we have to specify which column we want to dummy encode.

X = onehotencoder.fit_transform(X).toarray()

Here, we are overwriting the data with the new dummy encoded values.

- **Splitting the data into training data and test data**

Now, that we have standardized our data. We have to split this data into two:

- Training Set
- Testing Set

Training set is used to train the model and Testing set is used to test the model.

Code:

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.2,
    random_state=0)
```

Code Explanation:

Here, we are splitting our two arrays X and Y for training and testing. The variable X_train will hold the training dataset for X and the variable X_test will hold the testing dataset for X. Similarly for Y.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=0)

Here, as the parameters of the function, we have passed our arrays *X* and *Y* and *test_size* and *random_state*.

The *'test_size'* specifies how much data we are allotting for the testing dataset. Here we have specified 0.2, which means 20% of the data will be given for testing. So the remaining 80% of the data will be given for training.

The *'random_state'* specifies how much randomness there is in choosing the test data and training data.

- **Feature Scaling**

We have to do feature scaling, to ensure that no variable(column) will dominate the other variables (columns), when we calculate the Euclidean Distance. So, the ratio values will be very big and we cannot get accurate results from it. So we use feature scaling, to convert all the values to a range of -1 to +1.

Code:

```
>>> from sklearn.preprocessing import StandardScaler
>>> sc_X = StandardScaler()
>>> X_train = sc_X.fit_transform(X_train)
>>> X_test = sc_X.fit_transform(X_test)
```

Code Explanation:

sc_X = StandardScaler()

Initializing an object 'sc_X' of class 'StandardScaler'

X_train = sc_X.fit_transform(X_train)

Here, we are feature scaling the values in X_train and overwriting it to X_train.

X_test = sc_X.fit_transform(X_test)

Here, we are feature scaling the values in X_test and overwriting it to X_test.

Conclusion

Now, the dataset is completely standardized and is now ready to be used in your machine learning algorithms.