1.Data Preprocessing Refresher

May 19, 2021

1 Import the necessary libraries

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2 Import the dataset

[2]:
```
df = pd.read_csv("Data.csv")
df
```

[2]:

	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

3 Basic Data Exploration

[3]:
```
#Get a rough feel of the data
df.describe()
```

[3]:

	Age	Salary
count	9.000000	9.000000
mean	38.777778	63777.777778
std	7.693793	12265.579662
min	27.000000	48000.000000
25%	35.000000	54000.000000
50%	38.000000	61000.000000
75%	44.000000	72000.000000

```
max     50.000000  83000.000000
```

[4]:
```
#Check the size of dataset
df.shape
```

[4]:
```
(10, 4)
```

[5]:
```
#Check the data types of the objects
df.dtypes
```

[5]:
```
Country       object
Age          float64
Salary       float64
Purchased     object
dtype: object
```

[6]:
```
#Check for any null values
df.isnull()
```

[6]:

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

[7]:
```
#Check for total number of null values present
df.isnull().sum()
```

[7]:
```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

4 Seperating into dependant and independant variables

Finally, we define both our independant (features) variables and the dependant variables. In this case the first three columns are the features and the last column is the dependant variable.

[8]:
```
#Defining the independant and dependant variables
X = df.iloc[:,:-1].values
```

```
X #Defining the independant variables
```

[8]:
```
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)
```

[9]:
```
y= df.iloc[:,-1].values #Defining the depedant variable
y
```

[9]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Note :

loc gets rows (and/or columns) with particular labels.

iloc gets rows (and/or columns) at integer locations.

5 Handling missing values

We specify the missing values and the strategy. In this case the strategy is that the missing values will be replaced by the mean

[10]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan,strategy='mean')
```

Now we need to transform and fit our data for the imputation. Select only the numerical columns

[11]:
```
imputer.fit(X[:,1:3])  #This will select the second and third column
X[:,1:3] = imputer.transform(X[:,1:3]) #This will replace the missing values␣
↪with the mean for the columns 2 and 3
X
```

[11]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
```

```
       ['France', 37.0, 67000.0]], dtype=object)
```

Encoding the independant Variables

In machine learning, all the data fed in must be numerical. Categorical variables can be represented by numbers through encoding and thus, encoding is important.

[12]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

[13]:
```
ct = ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[0])],remainder␣
 ↪= 'passthrough') #passthrough will let you keep the remaining columns i.e.␣
 ↪Age and Salary as well
X=ct.fit_transform(X)
X
```

[13]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

[14]:
```
#Convert to numpy array
X = np.array(X)
X
```

[14]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

6 Ecoding the dependant Variable

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

[15]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

7 Splitting the dataset into Training and Test

We do the splitting of train and test set before feature scaling. This is because, the test set is always hidden and shouldnt be leaked into the training set while training the model. In feature scaling we perform either normalization or standardization i.e. converting normal distrubtion to standard normal distribution. If feature scaling was done prior to splitting dataset , this could leak the standard deviation and mean of all the values including the ones of the test set. This is called information leakage.

[16]:
```
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

[17]:
```
X_train
```

[17]:
```
array([[0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

[18]:
```
X_test
```

[18]:
```
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0]], dtype=object)
```

[19]:
```
y_train
```

[19]:
```
array([1, 1, 1, 0, 1, 0, 0, 1])
```

[20]:
```
y_test
```

[20]:
```
array([0, 0])
```

8 Feature Scaling

The goal of feature scaling is to make all the values in the similar range Note : Dont apply feature scaling on the one hot encoding dummy variables

[21]:
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
X_train[:,3:] = sc.fit_transform(X_train[:,3:])
X_test[:,3:] =sc.transform(X_test[:,3:])
```

[22]:
```
X_train
```

[22]:
```
array([[0.0, 1.0, 0.0, 0.11473097566202424, -0.017053551664523183],
       [1.0, 0.0, 0.0, 1.2948210110428438, 1.3179959215010502],
       [0.0, 0.0, 1.0, -1.8029153318318076, -1.400827458157308],
       [0.0, 1.0, 0.0, 1.5898435198880487, 1.6688118414569675],
       [1.0, 0.0, 0.0, -0.622825296450988, -0.5237876582675149],
       [0.0, 0.0, 1.0, -0.06556055752115642, -1.0500115382013908],
       [0.0, 0.0, 1.0, -0.18029153318318064, -0.26067571830057706],
       [1.0, 0.0, 0.0, -0.3278027876057831, 0.2655481616332987]],
      dtype=object)
```

[23]:
```
X_test
```

[23]:
```
array([[1.0, 0.0, 0.0, 0.704775993352434, 0.7040680615781951],
       [0.0, 1.0, 0.0, -1.3603815685640002, -0.8746035782234322]],
      dtype=object)
```

Note : Why fit_transform() on train set and transform() on test set ? fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data. Here, the model built by us will learn the mean and variance of the features of the training set. These learned parameters are then used to scale our test data. So what actually is happening here! The fit method is calculating the mean and variance of each of the features present in our data. The transform method is transforming all the features using the respective mean and variance. Now, we want scaling to be applied to our test data too and at the same time do not want to be biased with our model. We want our test data to be a completely new and a surprise set for our model. The transform method helps us in this case.

transform() Using the transform method we can use the same mean and variance as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test data.

Thus, this prevents information leakage.

[ ]:
```
```