data_processing

September 13, 2020

1 Machine learning

1.0.1 Part 1: Data processing

We are going to start studying some of the fundamental libraries that we need: * numpy * matplotlib * pandas

We have to import them:

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

To import our datasets (typically a .csv file) we can use:

[2]:
```
dataset = pd.read_csv('Data.csv')
```

Data.csv in this case is a four column and 10 row file, with csv extension. The first 3 columns are our **features** (country, age and salary) and the last one is our **dependent variable** (purchase).

We have to create a python variable for our features and another for our dependent variable:

[3]:
```
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:,-1].values
```

We can print our variables:

[4]:
```
print(X)
print(Y)
```

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

Note, that in **X** there is some NaN (not a number) values. Sometimes our dataset has some missing data. We should fix that.

To solve this we can use the library *Scikit-learn*, one of the most useful data science libraries for *python*.

[5]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])
print(X)
```

```
[['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]]
```

Now the problem is fixed. This function calculates the mean of the no NaN values and gives that value to the NaN values (we calculate as means as columns with numerical data are). It is recommendable to do this for every column of numerical data that our dataset has. In data science problems is typically that datasets are formed with thousands and millions of rows of data, and maybe we do not have enough time to revise if there are empty places in out dataset.

We have another important problem to face. In our dataset we maybe have *categorical data*, namely data that are not a number, but we need number type data. To solve this problem is called *encode categorical data*.

We can solve this creating a vector with dimension dim=N, where N is the number of different names that are in our categorical data. Then we give to every of these different names a vector with one and different component. For example, in our case in the first column there are three options: France, Germany, Spain. So:

- France = (1,0,0)
- Germany = (0,1,0)
- Spain = (0,0,1)

If we want to code that we have to use the following function:

[6]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct =␣
 ↪ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[0])],remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
```

```
[[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]]
```

We have to do the same if our dependent variable is categorical, but now we only have to possible outputs, YES or NO, so it is sufficent with only give 1 to YES and 0 to NO:

[7]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(Y)
print(Y)
```

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

We have to split to dataset in order to create a training set and a test set. Training set will train our model and we will check if this model works properly with the test set

It is recommendable that training set contains 80% of the data of our dataset, and the test set the same.

To code this:

[8]:
```
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,random_state=1)
print(X_train)
print(X_test)
print(Y_train)
print(Y_test)
```

```
[[0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [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 35.0 58000.0]]
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
[0 1 0 0 1 1 0 1]
[0 1]
```

Finally we have to do a *feature scaling*. This is because we do not want that any data would be so

big that dominate over the other. We can solve this with different methodes: * **Standarisation:** the values will be between -3 and 3 approximatetly. Always is useful. We can calculate with:

$$x_{stand} = \frac{x - mean(x)}{standard\ deviation(x)}$$

* **Normalisation:** the values will be between 0 and 1 approximatetly. Is useful when our data follows normal distribution. We can calculate with:

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

We will always use standarisation method.

[9]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

But we should be carefull and do not apply to categorical data that we encoded, because it will be nosense.

[10]:
```
X_train [:,3:] = sc.fit_transform(X_train[:,3:])
X_test [:,3:] = sc.transform(X_test[:,3:])
print(X_train)
print(X_test)
```

```
[[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425]
 [0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372]
 [1.0 0.0 0.0 0.566708506533324 0.633562432710455]
 [0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867]
 [0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582]
 [1.0 0.0 0.0 1.1475343068237058 1.232653363453549]
 [0.0 1.0 0.0 1.4379472069688968 1.5749910381638885]
 [1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]
[[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727]
 [1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]]
```