Data Preprocessing and ML Modelling Template Notebook

Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
```

Dataset Intro

A dummy dataset made for the purpose of this exapmle notebook. It has four columns in total. The three independent data columns are 'Country', 'Age' and 'Salary'. The independent column is titled as 'Purchased'. This dummy dataset is about house purchases.

Importing the dataset

In [2]:
```
# Remove 'Data.(csv/json/xslx)' with your own csv/xslx/json filename. I have added templates for reading other formats as well.

dataset = pd.read_csv('Data.csv') #For reading csv files and storing them as pandas data frame

#dataset = pd.read_excel('Data.xslx') #For reading excel files and storing them as pandas data frame
#datasets = pd.read_json('Data.json') #For reading json files and storing them as pandas data frame
```

For more in depth information about different pandas read functions click here.

In [3]:
```
# Seperating data frame into X and y. X and y are both now numpy arrays.

X = dataset.iloc[:, :-1].values #Leaves out the last column and stores the rest in variable 'X'
y = dataset.iloc[:, -1].values #Stores the last column and stores them in variable 'y'
```

For more examples and to read up on iloc functions, click here.

In [4]:
```
X
```

Out[4]:
```
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)
```

In [5]:
```
y
```

Out[5]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Taking care of missing data

As seen by the printout of code cell 14, there are few missing values. There are several ways to handle this.

1) Imputing Missing Data - Simple Imputer

SimpleImputer is a scikit-learn class which is helpful in handling the missing data in the predictive model dataset. It replaces the NaN values with a specified placeholder.

It is implemented by the use of the SimpleImputer() method which takes the following arguments :

- **missing_values:** The missing_values placeholder which has to be imputed. By default is NaN
- **stategy:** The data which will replace the NaN values from the dataset. The strategy argument can take the values:
  - 'mean'(default) -- Can only be used with numeric data
  - 'median' -- Can only be used with numeric data
  - 'most_frequent' -- Can be used with numeric or string data
  - 'constant' -- Can be used with numeric or string data
- **fill_value:** The constant value to be given to the NaN data using the constant strategy.

I have provided examples with all the possible strategies.

For more information and reading up on it, click here.

In [6]:
```
from sklearn.impute import SimpleImputer
```

In [7]:
```
# a) Imputing Mean values
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# b) Imputing Median values
#imputer = SimpleImputer(missing_values=np.nan, strategy='median')

# c) Imputing Most frequent values
#imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

# d) Imputing Constant values
#imputer = SimpleImputer(missing_values=np.nan, strategy='constant')
```

For this example, I have chosen to go with 'mean' strategy.

In [8]:
```
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

In [9]:
```
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]]
```

2) Deleting missing data

In [10]:
```
# a) Drop entire row where atleast one element is missing.
#dataset.dropna()

# b) Drop the columns where at least one element is missing.
#dataset.dropna(axis='columns')

# c) Drop the rows where all elements are missing.
#dataset.dropna(how='all')

# d) Keep only the rows with at least 2 non-NA values.
#dataset.dropna(thresh=2)
```

Deleting missing values, is not a good practice in most cases and should be avoided because it caan lead the data to become biased in some instances.

Encoding categorical data

The dataset I am using has one column 'Country' as categorical column.

Encoding the Independent Variable

What is One Hot Encoding?

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. For example you have a dataframe like this:

Country	Salary Level
Spain	2
Germany	1
Germany	1
Spain	2

After one hot encoding it will look like this

Spain	Germany	Salary Level
1	0	2
0	1	1
0	1	1
1	0	2

What really happens is, if you look at row 1 in 'Country' column it states 'Spain' and in row 2 it states 'Germany'. In hot encoding it creates as many columns as the number of categories. In this case two columns and according to their exsistance they are marked on each row.

NOTE: 0 indicates non existent while 1 indicates existent.

In [11]:
```
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))
```

In [12]:
```
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]]
```

Encoding the Dependent Variable

What is Label Encoder ?

Encode target labels with value between 0 and n_classes-1.

This transformer should be used to encode target values, i.e. y, and not the input X.

In [13]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

In [14]:
```
print(y)
```

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

Splitting the dataset into the Training set and Test set

In [15]:
```
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)
```

In [16]:
```
print(X_train)
```

```
[[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]]
```

In [17]:
```
print(X_test)
```

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

In [18]:
```
print(y_train)
```

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

In [19]:
```
print(y_test)
```

```
[0 1]
```

Feature Scaling

What is StandardScaler? Standardize features by removing the mean and scaling to unit variance

The standard score of a sample x is calculated as:

$$z = \frac{(x - u)}{s}$$

- **x** is the sample
- **u** is the mean of the training samples or zero if with_mean=False
- **s** is the standard deviation of the training samples or one if with_std=False.

In [20]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
```

In [21]:
```
print(X_train)
```

```
[[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]]
```

In [22]:
```
print(X_test)
```

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

Let's see what ML Algo suits best - LazyPredict

I am going to use **'lazypredict'** for this part as it helps build a lot of basic models without much code and helps understand which models works better without any parameter tuning. It supports supervised learning (Classification and Regression).

NOTE:

- It only works for python version ≥ 3.6
- It's built on top of various other libraries so if you don't have those libraries in the system,
- If you encounter 'ModuleError' so interpret the error properly and install the required libraries.

For more information about LazyPredict, please visit here

In [23]:
```
# Done to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

Lazy Classifier

In [24]:
```
from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split

clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X_train, X_test, y_train, y_test)
models
```

```
100%|██████████| 30/30 [00:03<00:00,  8.06it/s]
```

Out[24]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LinearSVC	1.00	1.00	1.00	1.00	0.03
PassiveAggressiveClassifier	1.00	1.00	1.00	1.00	0.02
LogisticRegression	1.00	1.00	1.00	1.00	0.04
LabelSpreading	1.00	1.00	1.00	1.00	0.03
LabelPropagation	1.00	1.00	1.00	1.00	0.03
RandomForestClassifier	1.00	1.00	1.00	1.00	0.37
GaussianNB	1.00	1.00	1.00	1.00	0.03
NearestCentroid	1.00	1.00	1.00	1.00	0.04
CategoricalNB	1.00	1.00	1.00	1.00	0.04
BernoulliNB	1.00	1.00	1.00	1.00	0.04
RidgeClassifierCV	0.50	0.50	0.50	0.33	0.03
SGDClassifier	0.50	0.50	0.50	0.33	0.02
SVC	0.50	0.50	0.50	0.33	0.02
XGBClassifier	0.50	0.50	0.50	0.33	2.01
QuadraticDiscriminantAnalysis	0.50	0.50	0.50	0.33	0.04
Perceptron	0.50	0.50	0.50	0.33	0.02
RidgeClassifier	0.50	0.50	0.50	0.33	0.02
AdaBoostClassifier	0.50	0.50	0.50	0.33	0.22
NuSVC	0.50	0.50	0.50	0.33	0.03
BaggingClassifier	0.50	0.50	0.50	0.33	0.08
LinearDiscriminantAnalysis	0.50	0.50	0.50	0.33	0.04
KNeighborsClassifier	0.50	0.50	0.50	0.33	0.03
ExtraTreesClassifier	0.50	0.50	0.50	0.33	0.28
ExtraTreeClassifier	0.50	0.50	0.50	0.33	0.03
DecisionTreeClassifier	0.50	0.50	0.50	0.33	0.03
CheckingClassifier	0.50	0.50	0.50	0.33	0.03
LGBMClassifier	0.50	0.50	0.50	0.33	0.05
DummyClassifier	0.00	0.00	0.00	0.00	0.04

Now, seeing the results for different models I can choose the ones that show the most promise and build a fine tuned version of them.

Hope you enjoyed reading this and learned something new ! Feel free to provide any feedback and connect with me.

Thanks !

Perpared by Asad Mahmood.