PART 1 - Data Preprocessing.

#workflow

Data Preprocessing
- import data
- clean data -* important step IRL.
- split into train & test sets.

Modelling
- Build Model
- Train Model
- Make Predictions.

Evaluation
- Calculate performance matrix
- Make verdict.

Train-Test split.

usually 80 / 20 percent

80 → Training

20 → Used trained model on test set that has known results to compare with models result.

Feature-Scaling

Remember (always applied to columns & never on rows).

① Normalization $x' = \dfrac{x - x_{min}}{x_{max} - x_{min}}$

[Most values b/w [0, 1]]

where x is the value of column

x_{min} is lowest & x_{max} is highest

② Standardization $x' = \dfrac{x - \mu}{\sigma}$ where μ is mean/average & σ is std. deviation

[Most values except [+3, -3] extremes or outliers.]

- with unscaled features, the lower valued/unit column become irrelevant & make small difference to large scale values.

Practical.

allows to import a library or function or any module.

(1) import libraries
   (1) numpy - works with arrays for inputs
   (2) matplotlib - for charts/graph/visualizations.
   (3) pandas - import dataset / create matrix of features & dependent variable vector.

- Libraries - ensumble of module containing ~~modules~~ functions & clasess.

- import numpy as np
  alias.
  we can call with np ~~too~~ now.
- import matplotlib.pyplot as plt.
  interested in this module.
- import pandas as pd.

(2) Import dataset
   (1) dataset = pd.read_csv(filename.extension).
   - create a dataframe with all values in data set.

   (2) create 2 new entities
      (1) matrix of features & (2) dependent variable vector

* features (independent variables) - the column with which we are going to predict dependent variable.
* dependent variables :- last column of dataset that need to be predicted

fullform - index location

feature of pandas, allows us to take index of columns/rows we wanna extract.

* x = dataset.iloc [:, :-1].values

takes all rows (in python it means a range), without (LB or UB means all).

Here [:, : -1]

Row all

column (all except last).

so,

* y = dataset.iloc [:, -1]

③ Handling Missing Values.

Method I: Just delete the row of missing value.
- works good for large datasets with low % of missing data.

Method II: Replace missing value by average of all data value/ in column.

* Scikitlearn - has lot of tools & preprocessing tools for this task.

* we use a module called impute in scikitlearn & ~~a class~~ import the simpleImputer from it.

value to replace

imputer = simpleImputer (missing_value = np.nan, strategy = mean)

to be replaced with.

- Now, this object class has functions that will be used to find missing values then input it.

upper bound is excluded in python (3-1) = 2 cols.

- imputer.fit (x[:, 1:3])

expects all the column with numeric of x

- imputer.transform(x[:, 1:3]) ← returns new updated matrix of features

value

So to change original as it have feature matrix
- x [: ,1:3] = imputer.transform(x[:,1:3]).

④ Encoding Categorical Data.

* It will be difficult to co-relate /compute these features with category String data, Thus we need to turn these categories to numbers.

Idea ~~Method~~ I: Encode France to 0
Spain to 1
Germany to 2

- However model may interpret as order/numeric order & interpret as the order matters.

Better To-Do: ~~Turn~~ Turn to one hot encoding i.e. turn the category into total columns with total no. of distinct category.

So for France spain & germany we ge.

~~for~~ France	1	0	0
Spain	0	1	0
germany	0	0	1

① Encode InDependent Variable
- For this we use 2 classes
  ① column transformer class from composed module of scikit
  ② one hot encoder class from preprocess module of scikit

① ct = ColumnTransformer(transformers, remainder)

what kind of transformation on which indexes

to specify we are doing encoding transformations

class name to encode to

columns that won't be applied for transformations.

* transformers = [('encoder', oneHotEncoder(), [0])] → indexes to transform/encode

codename to say we wanna keep columns as it is & wont apply transformation.

remainders = 'passthrough'

If we dont apply the remaining columns wont be added.

- with this class, we can fit and transform at once with function

x = ct.fit_transform(x)

input to transform.

Machine learning use numpy array so we need to convert it to np array

we call x = np.array(ct.fit_transform(x))

② Encode Dependent variable.

- we use labelencoder class from preprocessing from scikit

lbl = LabelEncoder()

only one column with text value, so its obvious what needs to be performed

y = lbl.fit_transform(y).

doesnt need to be numpy array since its single dependent var array.

⑤ Splitting Dataset into Train & Test set.

※ We apply feature scaling after splitting data.

because we only use train set for ML model while test set represents real world data. ~~& shouldn't be altered.~~ ~~It just needs to be fed to model & get o/p.~~

- Also when applying feature scaling we use mean & standard deviat that might leak some info to train set which isn't supposed to happen. The ML model will get info on test set & may fit itself for it.
- We scale test later after splitting & scaling train set.

~~For~~ For this will use sklearn module that has model_selection class with train_test_split fn.

- We split into 4. 2 feature matrices & 2 depend variables.

expects first feature mat

x_train, x_test, y_train, y_test = train_test_split(x, y

next expects dep.var

~2 more args~ train_size=0.8 or test_size=0.2 , random_state=1)

int type seed for random value.

⑥ Feature Scaling: (Not necessary for all models).

works well all the time.

2 techniques ① standardization (with mean(μ) & deviation(σ)).
→ (+3, -3).

② Normalization: (with x_{min} & x_{max}).
→ (0, 1)

works/recommended if we have normal distribution among most of our features.
(recommended for specific situations).

- For this step, we use same scikit module.
from sklearn.preprocessing import StandardScaler

class.

DATE
PAGE NO.

sc = StandardScaler() ← automatically applies standardisation.

- Dummy variables → variable columns after encoding categorical data.

- We dont need to apply standardisation to dummy variables since they are already b/w (-3,3) or (0,1).

① First we fit our scaler on training sets.
so x_train[:,3:] = sc.fit_transform(x_train[:,3:]).

everything after & with 3rd column

② Next we fit x-test.
- we only apply transform method here since this data is like new data received IRL by model.
- The features of test set need to be scaled by the same scaler used on training set.
- we donot calculate a new fit. we got that from fit_transform earlier
x_test[:,3:] = sc.transform(x_test[:,3:])

SUMMARY:

① All Libraries for Preprocessing

- import pandas as pd [General]
- " numpy as np
- from sklearn.impute import SimpleImputer (class) (for Missing values)
- " sklearn.compose " ColumnTransformer (class) (for encoding categorical data)
- " sklearn.preprocessing " OneHotEncoder (class)
- " " " LabelEncoder (class) (for encoding label/dependent data)
- " sklearn.model_selection " train_test_split (function) (for splitting)
- " sklearn.preprocessing " StandardScaler (class) (for feature Scaling).

② To import dataset (and assign dep. & indep. variable).

```
dataset = pd.read_csv('./loadfilename')
x = dataset.iloc[:, :-1].values
y =   "      [:, 1].values
```

③ Handle Missing data

```
imp = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imp.fit(x[:, 1:3])
x[:, 1:3] = imputer.transform(x[:, 1:3])
```

[→ first fit to compute → & then tranform to replace]

④ Encoding Categorical Data.

① Indep. variable.

```
colTr = ColumnTransformer(transformers = [('encoding', OneHotEncoder(),
                          [0])], remainder = 'passthrough')
x = np.array(colTr.fit_transform(x))
```

② Dep. variable.

```
lbEnc = LabelEncoder()
y = lbEnc.fit_transform(y)
```

⑤ Split Data

```
x_tr, x_tst, y_tr, y_tst = train_test_split(x, y, test_size = 0.2,
                                            random_state = 0)
```

⑥ Scaling (only when necessary)

```
sclr = StandardScaler()
x_train[:, 3:] = sclr.fit_transform(x_train[:, 3:])
y_train[:, 3:] = sclr.transform(x_test[:, 3:])
```