(*) Download R& setup google collab.

I will use VS code

ctrl+enter ⇒ to run a selected code

~~Varia~~

3 window in R Studio
- (A) Enviroment
- (B) Files
- (C) Code editor

Here we start with machine learning

(*) machine learning processes
- (A) Data Preprocessing
- (B) Modeling
- (C) Evalution

this heiarchy is called as machine learning process.

these three are main core of entire machine learning and we can do anything in ML on basis of these three steps

These above stages can further resolved in another sets

Data Preprocessing
- (I) Import the data
- (II) clean data
- (III) split data in training and test sets

Modelling
- (I) Build a model
- (II) Train a model
- (III) Make Prediction on basis of model

Evaluation
- (I) calculate performance metrics
- (II) Make a verdict about model

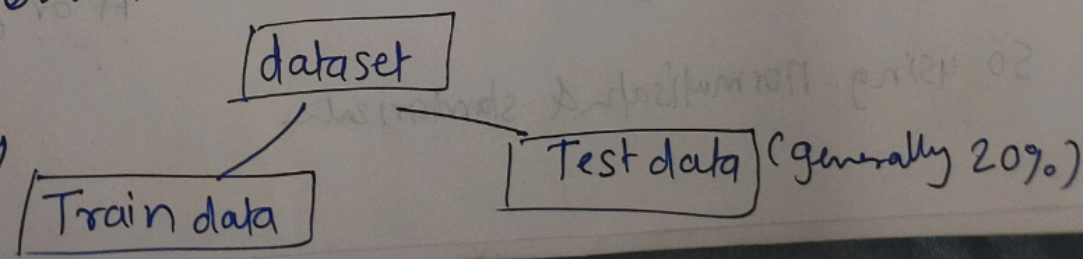
so this is basically a structure each machine learning developer must gone through

→ Evaluation help us to find out about performance of model

Spliting of data into train ~~test~~ data & test data

spliting of data is important in order to find out information about the model

⇒ So Overall dataset is chopped in two pieces



Generally (80%)

(generally 20%)

We use 80% Train dataset to build a model & rest 20% is used to test the trained model.
=) and such like that we can find out predicted values for test set and check it with actual result and by this we will be able to evaluate model.

#Feature scaling (scaling is good & important)

Feature scaling is alway apply to columns of dataset. and you will never apply feature scaling to row.

Feature scaling can be done with:

(A) Normalization

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

above formula will be apply to each value in column & as a end result you will have value betn [0:1]

Range of values : [0:1]

(B) Standardization

$$X' = \frac{X - \mu}{\sigma}$$

above formula will also apply to entire column and. you will end up getting values in Range: [-3 ; +3]

Q) 3 Persons

		Salary	age
1	A	70000	45
	B	60000	44
	C	520000	40

Find out with feature scaling whom with Person B is like more likely A or C

So using normalisation & standarization

Persons	normalised salary	Standardised salary	normalised age	Standardised age
A	1	0.731	1	0.925
B	0.44	-0.052	0.8	0.463
C	0	-0.68	0	-1.4

μ = mean

σ = std. deviation

Variance age = $\frac{2^2+1^2+3^2}{3} = \frac{4+9+1}{3} = \frac{14}{3}$

$\mu_{salary} = \frac{182000}{3} = 60,667.$ $\mu_{age} = \frac{43}{}$ =

Variance Salary $= \sigma^2_{salary} = \frac{\Sigma(X-\mu)^2}{N} = \frac{(70000-60,667)^2}{3}$

$\sigma_{salary} = 12754$

$\sigma_{age} = 2.16$

$= \frac{(70000-60,667)^2+(60000-60667)^2+(52000-60667)^2}{3}$

$\sigma^2_{sal} = 162666667$ $\sigma = 12754$

1

Standa.

$X'_{A\,sala} = \frac{70000-60667}{12754}$. $X_{A\,age} = \frac{2}{2.16}$

→ we don't want to find out difference on basis of substraction only we want more optimised technique in our case it is normalisation & standariadn

⇒ In each column value share same property

⇒ In row there can be different value for different property.

By these values we can find out with which value the similar

Preprocessing In Python

Python

as main stages IN ML

Ⓐ Data Preprocessing
Ⓑ Modeling
Ⓒ Evaluation & optimisation

so we can do it all in R or in python but here we are going to do it with Python

⇒ data is not uniform also not normalised
So it contains regulaties like empty space or wrong value in order to remove them from system we use ML.

#Independent variable : what you have
#dependent variable : Variable which we want to Predict.

#Tasks we have to done in preprocessing

Ⓐ Taking care of missing data
Ⓑ Removing irregularities & redundancy
Ⓒ Encoding data (mean verbal data to numbers)
Ⓓ Splitting data in Training and Test
Ⓔ Feature scaling (normalising) standardizing data)

Imp we divide the dataset in preprocessing state itself.

If you preprocess data in very well manner it become easy to improve efficiency of model

#Start with data preprocessing

Ⓐ Importing libraries / Packages

use keywords like a import, from, as.

Ex Import pandas as pd
from matplotlib import pyplot as plt

⇒
Such like that we can code to use the functions inside the package which can help us in ML & further steps

⑥ importing dataset

for importing file should be at appropiate position of file system we mostly use pandas to import data set it has functions like read_csv(), read_excel, read_gbq & the type of file you want

Ex: data = pd.read_csv("file name with extension").

pd.read_csv() give return in a dataframe format

you can use ⇒ Print (type(data)) to check datatype of data

#Important

you also have to create
- (A) Matrix of features
- (B) dependent variable vector

(A) ~~vectors are row~~ (A) Features are columns other than last

~~The final data~~ should be on

⇒ last column provide info about verdict mean final result which is dependent variable (which we have to predict)

→ All other feature other than final col are independent var.

so we have to create 2 entities

(A) dataset without last column (matrix of feature)

(B) only last column (dependent variable vector)

In ML on basis of matrix of feature we have to predict dependent variable value

Ex: df.columns["name"] ⇒ this will give all data for a ~~colo~~ feature "name"

So for matrix of feature ~~yo~~

~~mof =~~

you can use iloc function where you can specify rows an colum which you want

df.iloc[rowstart:rowend, colstart:colend].

so with iloc it become a easy

so

we want all rows for

Ⓐ matrix of features → all col except last

mof = data.iloc[: , :-1]; → this will work regardless size of ~~df~~ data

↑ all rows

~~duv = df~~ Ⓑ for dependent variable ~~vector~~

duv = data.iloc[: , -1 :];

↑ only last col,

now mof = mof.values & duv = ~~data~~ duv.values

this will convert [mof / duv] datatype to the ~~array list~~ numpyarray

↑

using values is not neccesary as we can use dataframe ~~or it~~ also dataframe can easily converted to ~~easily~~

by ~~data = list(df)~~ .

[data = np.array(df)] → numpyarray

values also removes ~~a~~ title of each column like blah blah

⇒ which help to focus on data

mostly the matrix of features and dependent variable vector directly used so

[X = mof] & [y = duv] and with these you can create your ~~model~~.

Ⓒ Taking care of missing data

missing data is a Redundancy in ~~dataset~~ so it is important to remove it

NaN ⇒ Null value

Nan is show in python when there is a missing value in dataset

⇒ Removing redundancy is Important in order to make model work fine

⊙ How to handle redundant data

i] deleting redundant data.

this is good for large data, but it is not a great as if you have a large data with large redundantness so that is a problem.

ii] Actually Replacing missing value with average

this is good idea that to fill a empty entry just ~~remove~~ add that entry with a value of average of all from that col (not row obviously)

So how to do it.

For this we are going to use scikit Impute

~~So how~~ which rectify nan values & refill it ~~wi~~ according to info provided

code like: ~~from scikit with import Impute~~

from ~~scikit~~ Sklearn .impute import simpleImputer

↑ It is a class

So create instance (object)

Imputer = simpleImputer(); we don't have to use new keyword here like we use in java

↑ simple Impute take an

(Sklearn)

so

```
Imputer = simpleImputer(missing_values = np.nan,
                        strategy = 'mean')
```

this mean we have to enter which type of missing value you want to rectify & which strategy you want to use

and that will remove that value and provide new set

so you created object so now apply it on dataset, matrix of feature and the dependent variable vector.

So now we have connect dataframe & for that we have to use fit & transform.

So fit can only apply to numerical columns so you have to specify range of dataset with col—

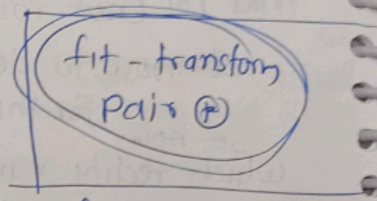
Imputer.fit(X[:, A:B]); Imputer.transform(X[:, A:B])

this is depend on data

.fit will connect imputer object with data and transform will make changes

→ or you can write function for it.

the fit will get value to modify and then use transform to make it do



→ same parameters should be entered in fit & transform

↑

the fit transform method returns a value & which will be new, X.

X[:, A:B] = imputer.transform(X[:, A:B]).

↑ we only want change some specific area

so thus you can remove ~~that~~ missing value

Here you have to write this code appropriately

whole code

```
#from sklearn.Impute import simpleImpute
#create object                                    Object
Impute = simpleImputer(missing_value = np.nan,
                       strategy = "mean");
#now sticking object with dataset & modify it
Impute.fit(X[rowrange, colrange])
                    ↓ same value to be inserted
~~X[ Impute~~
X[rowrange, colrange] = Impute.transform(X[rowrange, colrange])
```

↑

So, thus how you can remove missing values

© #Encode categorical data MLM ∃ machine learning model

So If some column has some set of repeated entries, it become difficult to for MLM so we can encode such data in some encoding Ex our data → dependable

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	yes
Gemany	30	54000	No
Spain	38	61000	NO
Gemany	40	63778.8	yes
France	35	58K	yes
Spain	38.77	52K	NO
france	48	79K	yes
Germans	50	83K	No
France	37	67K	yes

In purchased No of yes Repeating

So in country

France
Spain
Germany

repeating multiple time

Independent

So while encoding categorical data it can be of

(I) dependent variable

(II) Independent variable

And for both we have different way to approach.

(#) For independent

so approach can be 0: France, 1: spain, 2: Germany

But this can lead model to some discontinuities so in order to avoid it all we can do is to create

we are going to use one hot encoding and it is very helpfull in categorical variable like for drink Such like Hot.

- xtra small small, medium, large, extra Large

(#) the first dependent can be binary output but first we learn for independent [scikit-learn ⇒ sklearn]

Onehot encoding from scikit ~~learn~~ learn

(#)

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
#Create object both class
    ct = ColumnTransforme ( Transformer, remainder)
```

~~ct = Colum transfo (Tranformers=[('encoder', OneHotEncoder)~~

```
ct= ColumnTransform ( transformer = [('encoder',
OneHotEncoder(), [0]]] , remainder = "passthrough")
```

we just have to use code nothing else

What should we do for remaining set

Braces

Which column you want to apply transformation

no. of different category = new no. of different cols

3 cols as only have spain, Germany, italy

now use that object and connect to it

the columnTransforme has fit transform who do both fit & transform at once.

```
X= ct.fit-transform(X)
```

→ do change to.

this is not numpy array or

so X= np.array(X)

X= np.array (ct. fit-transform(X));

And thus how you can convert a independent Categorys Encoding

It divide create multiple column so If Coun is germ vector will be [0|0|1] & such three colum

no apply for categorical transformation with dependant variable

we can transform dependable into 1 & 0 and for that we have label encoder so for a string it convert into number

means: [yes = 1 || no = 0]

② we have sklearn.[preprocessing] import labelEncoder

↑ Here encoder are stored

and sklearn.compose contains transforming.

Code do same like create tupple and use columnTransf

Code

⇒ ↓ label encoder do not need

from sklearn.compose import columnTransform as CTR

from sklearn.preproces import labelencoder

Code ⇒ LE = labelencoder()

and le.fitTransform(y);

LE & columntran has fit-transform

So as we splitted X, & y thus how it make us ease

So use scikit for labelencoding, oneHotEncoder ColumnTransform and also Impute transform

Fit & transform different

MOF y

X ⇒ should be in numpy array

y ⇒ should be ∅ should not be no much issue

(D) Splitting data in Training and testing dataset

So firstly feature scaling is very helpfull for normalisation & Standardization of data so there is debate that when to use feature scaling before splitting dataset or after splitting

(*) Splitting first and then feature scaling.

Feature scaling is used to scale data in some range inorder to avoid one specific data to overoccupy all ~~thing~~ logic in model

(IMP)

⇒ we want to keep Test set unknown to training set so inorder to hide attributes of Test set like mean, variance, std. deviation we have to first do splitting and then go for feature scaling diffrently for train & test set

⊙ Basically to avoid info leakage from training set

(*) Splitting data

We are going to use scikit learn inorder to split data

So we are going to make 4 sets.

2 for matrix of feature (X) & 2 for dependent ~~de~~ variable vector (y)

That are

X train
X test

y train
y test

⇒ And all of them are used in building future machine learning model

So

```
from sklearn.mode_selection import train_test_split.
```

So we get 4 outputs

matrix of feature

dependat variable matrix

```
X_train , X_test , y_train , y_test
= train_test_split( X, y, test_size = 0.2, random_state = 1 )
```

Parameters → size

So

Train-test-split (take matrix take dependent of feature ' dVariable vector, test-size ~~0.2~~ =0.2 ↑ random-state =1)

0.2 = 20%

this will do uniform picking ⇒ this will pick entries randomly

optional can skip

⇒ so just use the test-train-split & you will get four dataset as output.

⇒ If you not provide test-size it will split in

70% Train 30% Test

⇒ It should be in mind ~~while~~ while dealing with it.

Xtrain and ytrain are related and ~~Xtest~~ Xtest and ytest are related.

Feature scaling

Feature scaling used to avoid ~~to~~ overweighting some values over another

So It is not mainly used in all cases but very usefull in some of them

Approach

Ⓐ standardisation ⇒ good most of the time

$$X_{stand} = \frac{X - mean(X)}{standard\ deviation(X)}$$

Range ⇒ -3 to 3

Ⓑ Normalisation ↳ good for specific

$$X_{Norm} = \frac{X - Xmin}{Xmax - Xmin}$$

Range ⇒ 0 to 1

⇒ If your data has normal distribution normalisation will be very good thing to do

⇒ standardization work good in all time

#Code scikit

we use standard scalar class

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```
↑ do not need parameter

dummy values like values in so we don't have to apply standardization bcz as value are already betn -3 to 3

0	0	1
1	0	0
0	1	0

→ not good to undergo standardization so we have to exclude it from data

⇒ only apply feature scaling for numerical value which are high in value like in thousand and avoid to apply it for value in ⊖ -3 to 3

so

whom to apply

only apply standardization to ~~test and~~ xtest and Xtrain

so

```
xTrain[range] = sc.fit_transform(xTrain[range])
```

~~xt~~

and

```
xtest = sc.transform(xTrain[range])
```

for xtest

you can't use fit transform so just use same scalar to transform ↑ don't fit the xtest will [range]

sc that will lead to information leak