11/02/22

- Till now we have done many models which supervised learning.
- Now, lets start unsupervised learning. which comes under a concept called clustering.
- and we have K-means clustering, and many more.

- Clustering:-

For explaning clustering in real world sir has created a demo of web server configuration and when user hits that page that server always stores every details in logs.

i.e; at

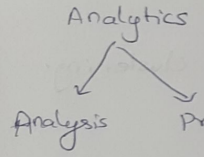
vi /var/log/httpd/access_log

- Now, here this file became dataset which stores all information of requests came to that server.

- Every request become a record in that file.

- we can do Analytics on this data

ie;



- In that log file we have data so, we have predictor(x) and target(y)

So, In Supervised Training/Learning we have x and y ie; at first y is there to identify/learn/train the model.

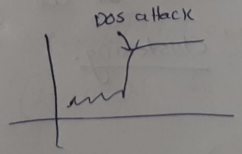
- But in some scenario's we don't know what to find ie; y.

But we need/have data and explore what all we can get from that data. and by this we can achieve some thing.

- So, Lets assume when we hit that server 1000 times then we can understand that some script is trying to hit that is Dos attack (Dienal of Service).

which will lead to loss for the company. So, from that log file we can identify patterns like this.

- If we have a graph of number of requests.



- When there a change in pattern (or) has this type of pattern (or) signature then we have many which sends mails (or) message which says it is Dos attack.

- But If we don't have pattern (or) signature. we are getting that type of pattern for the first time (ie Zero-day) then it is called as unsupervised Learning.

ie; we don't have "y" for specific "x", and we cannot train before to the model by not having any pattern.

- Splunk is one of the tool which identifies newer patterns.

- So, for this Identification ~~of~~ we use one of the thing is Clustering.

- When we get data we need to first Identify the pattern then after we can do other things like classification (or) regression models can be used based on pattern.

- For Clustering sir used one example:-

```
import pandas as pd

dataset = Pd.read_csv('Example.csv')

dataset.info()
```

- In this dataset we have two columns ie; Satisfied and loyality.

The user 1 (or) record 1 is satisfied with that much amount by our product and he is loyal to the product is explained by that loyality field. (It might be % (or) rating out of 10)

- So, Categorize all similar users at one place is called cluster.

ie; who are loyal and satisfied by the product we keep in one cluster and others to another cluster like that we divide.

So, for this we use K-means clustering Algorithm.

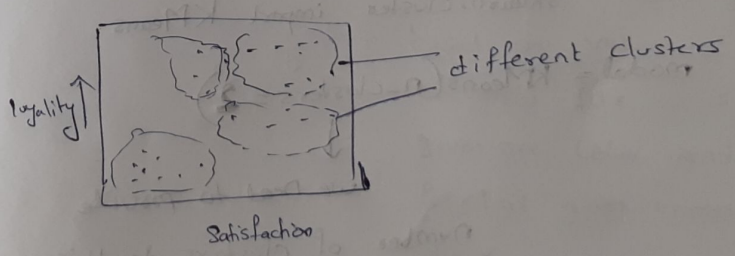
K means number of clusters. and it is constant.

- For Identifying this All the users/records we can do by maths (or) visual way/ graphical way (EDA) ↓ Explority Data Analysis

```
import matplotlib.pyplot as plt
sat = dataset['Satisfaction']
loy = dataset['Loyality']
plt.scatter(sat, loy)
plt.xlabel('satisfaction')
plt.ylabel('loyality')
```

In this graph we can identify many clusters.

- we can have, 1 to n clusters.
- In future sir will say how many clusters is better. so, that we get patterns and Informations from that data/clusters.
- hypothesis is equal to creating model.



21/02/22

K-means Clustering

- yestarday we saw a dataset in EDA (or) graphical way and identified manually different clusters.
- But few data points (users in that dataset) are very difficult to identify in which cluster they belong to.
- So, to make (or) identify to which cluster they belong we use a mathematical model (or) Algorithm called K-means.

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters = 2)
```

↓

we need to provide number of clusters to this Algorithm.

```
model.fit(dataset)
pred = model.fit_predict(dataset)
```

↓

This will give you output ie; each record in any of the 2 clusters as we provided 2 clusters ie; 0 (or) 1.

Now, we can see the dataset and say in which cluster that record/user belong to.

Lets see in graph ie;

```
plt.scatter(sat, loy, c=pred)
```

↓

Color map based on values of "pred" for 0 is given one color and for 1 it gives another color.

- In the dataset we have negative & positive value for different features.

So, the K-mean clustering model has given more importance to the positive values

∴ So, to remove this we need to do Pre processing ie; feature scaling.

ie; bringing all features to one scale only. (we have standard scalar)

- Not only here but any model if we have numbers as data for different features (or) fields it is good practise to do feature scaling.

- So, before training (or) Creating model do this :-

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data_scaled = sc.fit_transform(dataset)
```

data_scaled: after transform dataset.

dataset: This is original data set

use this dataset in Kmeans

```
model = KMeans(n_clusters = 2)
model.fit(data_scaled)
```

→ newly created dataset

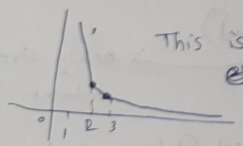
```
pred = model.fit_Predict(data_scaled)
plt.scatter(Satyloy, C=pred)
```

- now, graph will be different as we did scaling for both features.

- Try changing different number of clusters.

- To identify how many number of clusters are good is identified based on a method called WCSS (within-cluster sum of square)

- After having different clusters where the difference is coming and becoming constant till that change we can have that many clusters



This is also called as elbow method.

The sudden change in interia graph there we can say that point is best for number of clusters.

- model.inertia_

So, we see many inertia values and when we see sudden change that point is the best number of clusters.

- Now, lets create a code for different clusters, different inertias.

```
wcss = []
for i in range(1, 11):        # we are seeing 1 to 10 clusters
    model = KMeans(n_clusters = i)
    model.fit(data_scaled)
    w = model.inertia_
    wcss.append(w)
```

Now, if we see wcss value

```
plt.plot(range(1,11), wcss, marker = 'o')
```

here, we can see after 4 clusters it is constant.

25/2/22

KNN:- (K-Nearest Neighbours)

- Mostly All models (or) ML models are Trained and got weights (or) parameters and then we predict (or) test the model.

But KNN is a type of Algorithm where without Training (or) without parameters we can use it.

i.e; Non-parametric Algorithm

So, without training (or) without parameters the model can predict the data (output)

- So, Now, it doesn't learn (or) train we can say this KNN model as Lazy learner model/Algo

- while predicting it does some calculation and gives output so, it will be slower compared with other models.

- But in model creating this KNN will be faster.

- For this sir has took one dataset to show:-

```
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
dataset.info()
```

- This dataset is a company does advertisement and now, A person with a Age & salary purchased that product (or) not is given. in this dataset. '0' means doesn't purchase and vice versa for 1.

- KNN is mostly used for classification but even we can use for regression also.

```
X=dataset[['Age','Estimated Salary']]
Y=dataset[['Purchased']]
```

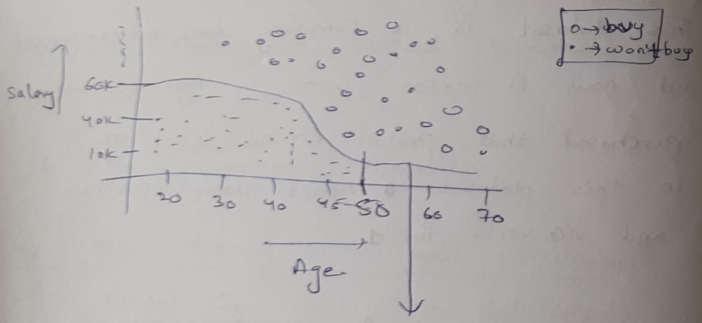
Lets see a graph using Seaborn

import seaborn as sns

sns.scatterplot(data=dataset, x='Age', y='Estimated Salary', hue='Purchased')

↓ To Categorize

- Now, we did EDA (Exploratory Data Analysis) and it shows like.



This like is called best-fit line which differentiate who will buy and who will not buy

This line is called Non-linear Best fit line

mostly we won't get linear Best fit line

- So, how good we create that Non-linear line and are are non-linear model then it will be best model based on that line.

- Now, for predicting if a person comes based on Age and Salary we place in that graph. and predict how many points ie, people near to you (or) nearest neighbour to that person/point.

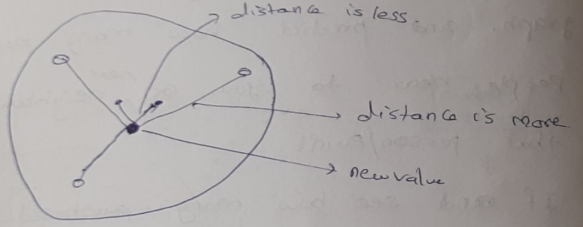
If and see how many purchased if many purchased neighbours are there then there is a chance that new person also buy's.

So, we call it as K Nearest neighbour ie; how many nearest neighbour are there similar thing is alone by that new person.

mostly we go for odd Number for K values ie; that many neighbours are compared.

- The Algorithm uses Euclidean Distance formula to check the distance to find the nearest neighbour.

- Based on K-value it find all the points near it and find distance near to the new value.



- so, this is why the prediction is time taking to find the distance of all points.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(X,y,
                                 test_size=0.25, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
```
↓ for this see before class

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
```
↓ underscore

```
from sklearn.neighbours import KNeighborsClassifier
model =
KNeighborsClassifier(n_neighbors=5)
```
→ we can change this number based on accuracy.

```
model.fit(x_train, y_train)
```
↓

This fit doesn't mean model is trained but it just store the data because it is a lazy learner. (model is not trained)

```
model.predict([[30, 20,000]])
```
↓ age ↓ salary

This doesn't give right way because we used scaled value so, to Predict first transform this values also.

- sc.transform([[20, 30000]])

↓

use same object as it uses same formula how it scaled for dataset

- after getting output copy paste it

- model.predict([[copied values]])

- Now, do it for test data

y_pred =

- model.predict(sc.transform(X_test))

↓

test data gets transformed and gets predicted.

we can compare this output ^ with

- y_test

- we can use confusion matrix to compare the outputs.

- from sklearn.metrics import confusion_matrix

- confusion_matrix(y_test, y_pred)

- This gives matrix.

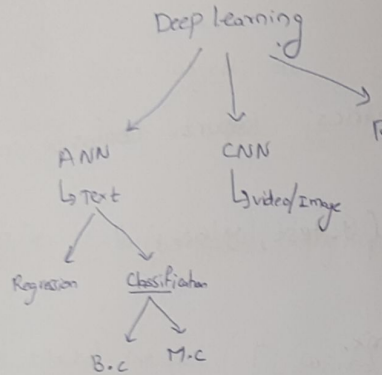
- If you need accuracy score.

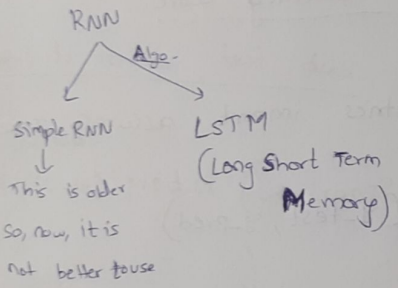
- from sklearn.metrics import accuracy_score

- accuracy_score(y_test, y_pred)

- Sir has shown "KNN Code" file with entire code. try it.

02/03/22





- data is in the format Series then we use RNN

Ex:- Time Series

Days	usage
1- monday	70%
2- Tuesday	20%
⋮	⋮
7 - Sunday	60%

- So, Based on previous day we are going to predict next day usage.

- Ill'y we can do for previous week, previous month (or) previous qvater (or) year- - - -

- we don't use any formula like $y = b + wx$ and identify y using x and weight but instead we use previous data.

- we have many examples i:e; NLP (natural Language Processing)

and Google translator, Keyboard suggestion, like this there are many examples based on previous data they give suggestions (or) prediction.

- Even sir explained when we have image and we do object identification and gives captions like (i:e; Gautam, Tree - - -).

- So, for RNN we are going to use LSTM Algorithm.

Code :-

- import pandas as pd

Time Series Data
↳ stock price data set

- dataset = pd.read_csv('Google_Stock_Price_Train.csv')
- X = dataset['Open']
- X.plot() ⟶ This method is from Pandas.

In this the every record has index has date.

- Here we need to find the window size that is after how many days the graph is going to repeat.

04/03/22

RNN :

one example :

values → 1 2 4 7 11

years → 1st 2nd 3rd 4th 5th

x	y
1st	1
2nd	2
3rd	4
4th	7
5th	11
6th	16

To find the 6th year we just used Previous value of y.

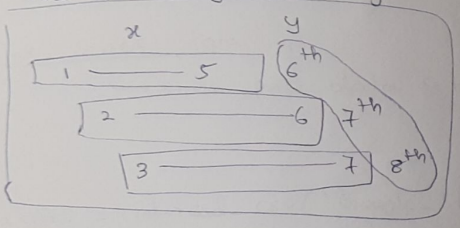
In regression we used x for prediction of y but here we used y to predict y.

So, This is how RNN work.

- For prediction y as 16 we used 5 previous y values to identify Pattern.

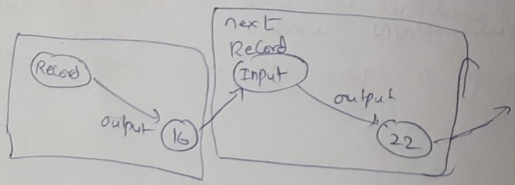
So, this is called as window size.

- III'd we use next 5 values to predict the next value

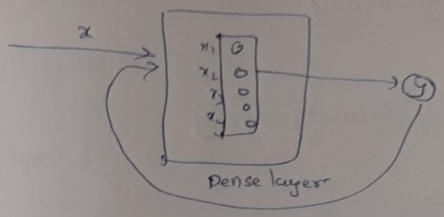


x_1	x_2	x_3	x_4	x_5	y
1	2	4	7	11	16
2	4	7	11	16	22
4	7	11	16	22	?

ie; previous Record Output becomes Input to Next Record.



we can think like a dense layers.



here the output is given as input to the next record. we can see in above layer.

So, we can sending output again into the layer that's why we say as Recurrent neural network (RNN)

- Here always we need to remember the output so, that it can send back again into layer (or) network.

for this we use Gate (like Logical Gates)

This stores for long term so that's why we call it as LSTM (long short term memory)

Continuation of yesterday code.

```
training_set = dataset['open']
training_set.shape
```

It look similar to CNN when we have Image and we convert into DataSet and feed into model.

Here also we take "open" looks like array and convert into a dataSet and feed in Model called as RNN.

```
training_set[0:60]
```

This all 60 records become $x_1, x_2 ---- x_{60}$

and,

```
training_set[61]
```

This will be come y_1

- for repeating we can use for loop

```
x_train = []
y_train = []
for i in range(60, 1258):
    x_train.append(training_set[i-60:i])
    y_train.append(training_set[i])
```

(total records of dataset.)

- Lets Convert this list to numpy array

```
import numpy as np
x_train_array = np.array(x_train)
y_train_array = np.array(y_train)
x_train_array.shape
y_train_array.shape
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
model = Sequential()

model.add ( LSTM(
            units = 60,
            return_sequences = True,
            input_shape = (60, )
            )
          )
```

- here we get error saying it should be 3D format, but we have in 2D format.

09/03/22

In yesterday's code sir did few modifications.

ie; instead of dataset sir changed to.

```
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv',
                            index_col = 0)
```

and new dataset for testing.

```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv',
                           index_col = 0)
```

- In previous model change variable of dataset to dataset_train

- we are going to predict open field from train dataset and we are going to compare with the test dataset

- `training_set.plot()` ⟶ train dataset graph
- `real_stock_price = dataset_test['open']`
- `real_stock_price.plot()` ⟶ test dataset graph

- As we know scaling of dataset will be increasing the accuracy of model.

- `from sklearn.preprocessing import MinMaxScaler`
- `sc = MinMaxScaler(feature_range=(0,1))`
  ↓
  we are scaling in between range of 0 to 1 but we can do scaling like before but this will be much more easier.

- Before transform lets convert data into 2D. we can do like [instead of reshape]

- `training_set = dataset_train[['Open']]`
- `real_stock_price = dataset_test[['Open']]`
  ↓
  while taking we directly convert into 2D

now, lets convert/transform

- `training_set_scaled = sc.fit_transform(training_set)`

- Now, similar to yesterday code we can see that for every 60 days [quater!] our data gets repeated.

```
X_train = []
y_train = []

for i in range(60, 1258):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])
```

because it is 2D

```
import numpy as np
x_train_array = np.array(x_train)
y_train_array = np.array(y_train)

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
model = Sequential()
model.add( LSTM(
        units = 60,
        return_sequences = True,
        input_shape = (60, 1)
    ) )
```

↓ it look like we make data into 3D but it is not

```
model.layers
model.summary()
```

```
model.add ( LSTM (
        units = 30,
        return_sequences = True,
    ) )
```

← This is for sending this layer output to next layer

```
model.add ( LSTM ( units = 30))
```

↓ This is last layer so, no need to send output to next layer.

```
model.add ( Dense ( units=1))
```

↓ By default the Dense function uses linear function/model/Activation function.

This is the output layer

```
model.layers
model.compile (optimizer = 'adam', loss = 'mean_squared_error')
model.fit(x_train, y_train_array)
```

- model.get_weights

- now, model is trained and lets predict and we can compare both the values from test data.

And we can compare using graphs also.

16/03/22

- Algorithm which can solve both Regression and Classification Problems Support vector (SV).

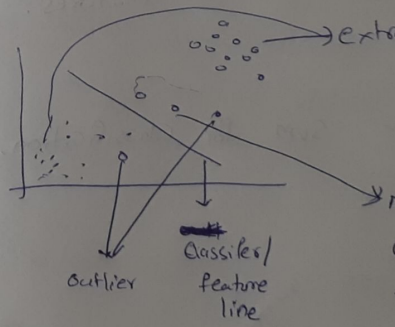
→ Support vector Regression (SVR) used for Regression

→ support vector machine (SVM) used for Classification

- Mostly used for SVM, which give more accuracy

- Previously the model use to Compare with the data points which are similar to the object.

Ex:- apple and orange.



- All the min. features of apple are close to orange if it is no. then it is not apple.

- If all the min. features of orange are close to orange then the object is orange.

- This is how SVM works.

- The Classifer line is dependent on extreme features. in NN.

- But in support vector the classifer line is dependent on min. features.

- Mostly we use SVM for Classification.

Code :-

Sir used Social_Network_Ads.csv dataset

```
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
dataset.columns
X = dataset[['Age', 'Estimated Salary']]
Y = dataset['Purchased']
from sklearn.model_selection import
                        train_test_split
X_train, X_test, y_train, y_test
        = train_test_split(X, Y, test_size=0.25,
                        random_state=42)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

→ same scale is used as X_train

```
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

30/3/22

Unsupervised Learning

↳ Dimention Reduction

↳ Principle Component Analysis (PCA)

- In Any model we will be identifying weights that is which feature has more weight.
  [features here are after feature selection and eleminated unwanted features]
- If a model has 2000 (or) more features it might not give accurate results some time and time taking process.
  Cause In this 2000 many features might have less weights i.e; which contribute to the model.

These are called as Curse of Dimensionality as many features increases the graph plotting all the dimensions also increases.

- But here we need to do Feature extraction.
- So, For reducing the Features (or) Dimensionality we can use PCA.

Code :-

```
import pandas as pd
dataset = pd.read_csv('wine.csv')
```

↓ multi classification example.

```
X = dataset[[ 'All Columns']]
y = dataset ['Customer_Segment']
from Sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
    = train_test_split(X, y, test_size=0.20,
                       random_state=42)
from Sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

#4 Now, Sir showed both normal model without feature / dimension reduction. and with feature reduction.

```
from Sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
from Sklearn.metrics import confusion_matrix,
                            accuracy_score
y_pred = model.predict(X_test)
confusion_matrix(y_test, y_pred)
```

* Here it shows for multi classification as it has 3 Categories. [rest all same]

```
accuracy_score(y_test, y_pred)
```

- Before this training model it better to do PCA

```
from sklearn.decomposition import PCA
mypca = PCA(n_components=2)
```

↓ Number of components (Dimensions/features) gets reduced to 2.

```
x_train
mypca.fit_transform(x_train)
```

↳ It mostly combines multiple features which are nearly same. By this we reduced features but information is not lost.

- Identifying similarities with other features is also called as clustering that's why it comes under unsupervised learning.

```
x_test=
mypca.transform(x_test)
```

↳ same model is used for test data

- Now, write the same code and test the model.

- After using PCA mostly reduce time and increase accuracy.

05/04/22

LDA :-

- If you want to reduce dimentionality with multi classification then PCA might not perform so well.
- For this case we can use LDA. (Linear Discriminant Analysis)
- In LDA it takes all features and compare with the output (y).
  So, it is supervised learning.
- In previous example we have transformed only x_train data as we focus on the x features as it is unsupervised learning But here we use both.
- Sir has explained in previous code and done few modifications. instead of PCA sir used LDA.

Code :-

```
from  SKlearn.discriminant_analysis
                    import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components = 2)

x_train = lda.fit_transform(x_train, y_train)

x_test = lda.transform(x_test)
```

↳ we just use x_test as we use previous x_train model & weights for x_test.

---

few models for Classifications

→ Logistic Regression
→ Neural Network where we use sigmod function for output
→ SVM (Support vector machine)
→ KNN
→ Naive Bayes
→ Random forest

Gradient Boosting Algorithms

Gradient → slope (or) weight.

for this one of the best Algorithm is

- extreme Gradient Boosting Algorithm (XGBoost)

- This automatically does hyper tuning parameters.
- auto feature selection is also done.
- By this the accuracy is more compared with other algorithms. (SVM, KNN)
- Sir Create code using SVC by removing that model.
- ## pip install xgboost

After installing write the code by removing SVC() model

```
from xgboost import XGBClassifer
model = XGBClassifier()
model.fit(x_train, y_train)
```

- This Algorithm is best in Big data.

12/04/22

- After we create model and we move to production then the model might not give more accuracy when compared with test data.

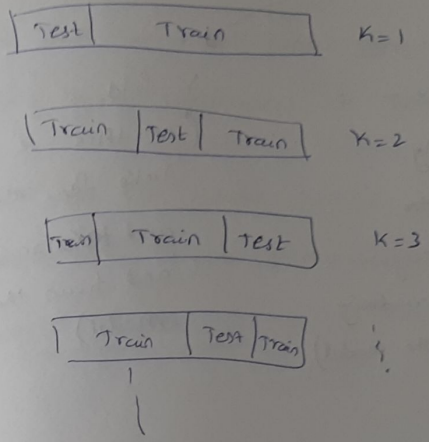
This happens based on Dataset split.

- To remove this issue we always split data using random_state. which every time we run the model it takes random records and train the model.

- So mostly its better to train the model again and again.

- For this we have Cross validation (cv) which will find which random state will be more accurate and the model is create.

For this,



Like this it takes different parts in data set

This is called as K-fold Cross validation

```
from sklearn.model_selection import cross_val_score

accu =
cross_val_score(estimator = classifier, x = x_train,
                                  ↓
                              model name
                y = y_train,
                cv = 10)
                  ↓
              Cross validation

accu.mean() * 100
```

↓

Mostly we consider Average accuracy from all 10 results

Types of cross validation

- Leave one out CV
  (In this we only take 1 record for testing and change that record randomly and train the model)
- K-fold
  (In this we take few records and ~~take~~ randomly and train the model).

- Mostly we use K-fold CV.

13/4/22

- Till now we have taken (or) did manual selection of hyper parameters. to ~~Tune~~ model.
- Lets see a model which will automatically selects the hyper parameters i.e;

  Grid Search CV.

- For few models like SVC( )
  They have many parameters we kept manually but to automate we can use Grid search CV.
- After Creating model.

~~#~~ from sklearn.model_selection import GridSearchCV

- Parameter grid is like a array which has different key value pair in which ~~we~~ ~~can~~ model will take those value.

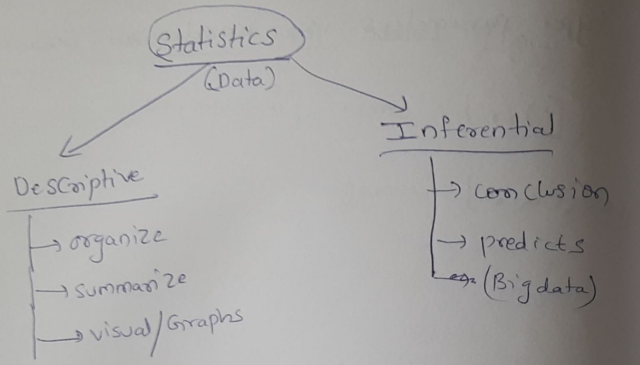
```
p = [
    {'C': [1, 2, 10, 20, 100, 1000], 'kernel': ['linear']},
    {'C': [1, 2, 50, 100, 2000], 'kernel': ['rbf'],
                                 'gamma': [0.1, 0.2, 0.3]}
]
```

- `grid_model = GridSearchCV(estimator=model, param_grid=p, scoring='accuracy', cv=10)`
- `grid_final = grid_model.fit(x_train, y_train)`
- `grid_final.best_score_`
- `grid_final.best_params_`

Shows best hyper parameters.

Another model for Automate this finding of hyper parameteres is RandomizedSearchCV.

20/04/22



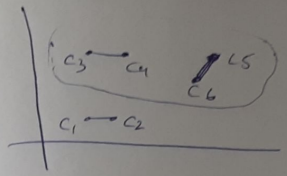
Clustering (unsupervised)

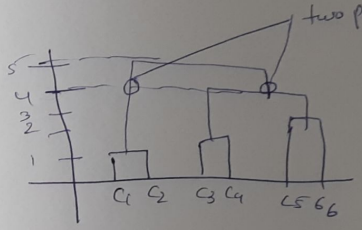
↳ Hierarical clustering (Agglomerative Clustering)

↳ we need to Identify no. of clusters.

- To identify no. of clusters we are going to use a graph called Dendogram.

- Sir showed example of customers. graph.





- having comparisions with different groups.
- From this we can identify the number of clusters.

Ex:-

Sample Code :-

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
dend = shc.dendrogram(shc.linkage(data_scaled,
                                  method="ward"))
```

* For Complete Code See Sir's drive.