=INDEX=

Questions:-	Remark:-
1. Clustering algorithms for unsupervised classification.	
2. Association algorithms for supervised classification on any dataset.	
03. Developing and implementing Decision Tree model on the dataset.	
04. Bayesian classification on any dataset.	
05. SVM classification on any dataset .	
06. Text Mining algorithms on unstructured dataset .	
07. Plot the cluster data using python visualizations .	

08. Creating & Visualizing Neural Network for the given data. (Use python) .	
09. Recognize optical character using ANN .	
10. Write a program to implement CNN .	
11. Write a program to implement RNN .	
12. Write a program to implement GAN .	
13. Web scraping experiments (by using tools) .	

Que 01. Clustering algorithms for unsupervised classification ?

->

Cluster analysis, or clustering, is an unsupervised machine learning task. It involves automatically discovering natural grouping in data. Unlike supervised learning (like predictive modeling), clustering algorithms only interpret the input data and find natural groups or clusters in feature space

Unsupervised Learning: Clustering Algorithms

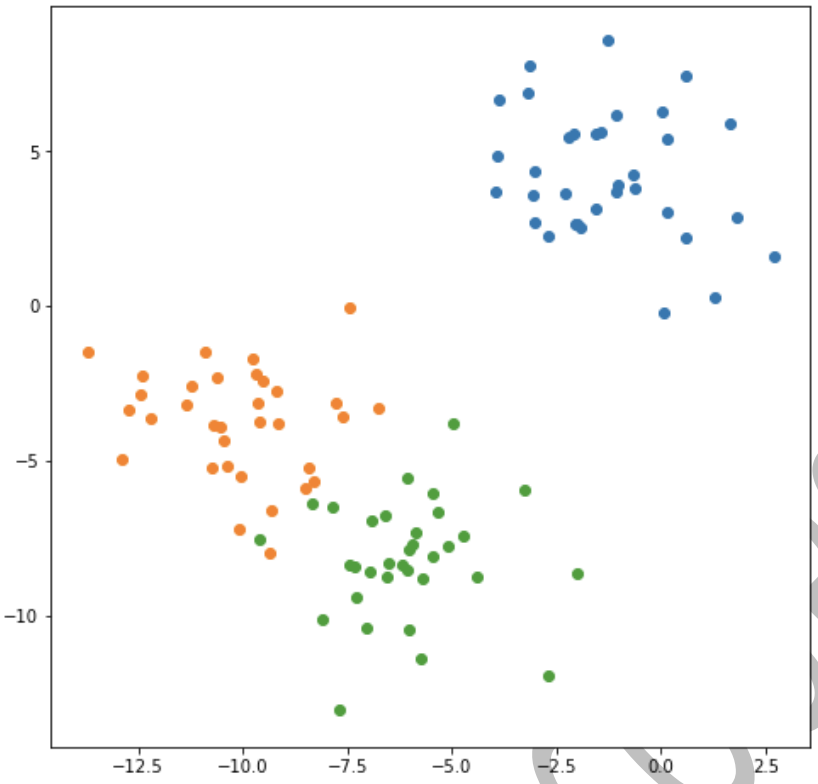
Using K-means and agglomerative clustering algorithms to group data with no labels

Predictive models generally require what is called _labeled' data to train on- that is that the data has some target variable that you have filled out already. Of course, the goal is to use the model on unseen data where you don't know the value of the target variable, but without properly labeled training data, you have no way of validating your model. For this reason, data production is often the hardest part of a data science project. Say, for instance, you want to teach a computer to read handwriting. It's not enough to collect hundreds of pages of written words and scan them in. You also need to label this data, to have an accompanying _correct reading' for each word. With something as complicated as handwriting, you might need hundreds of thousands of training examples and hand labeling that many entries will be laborious.

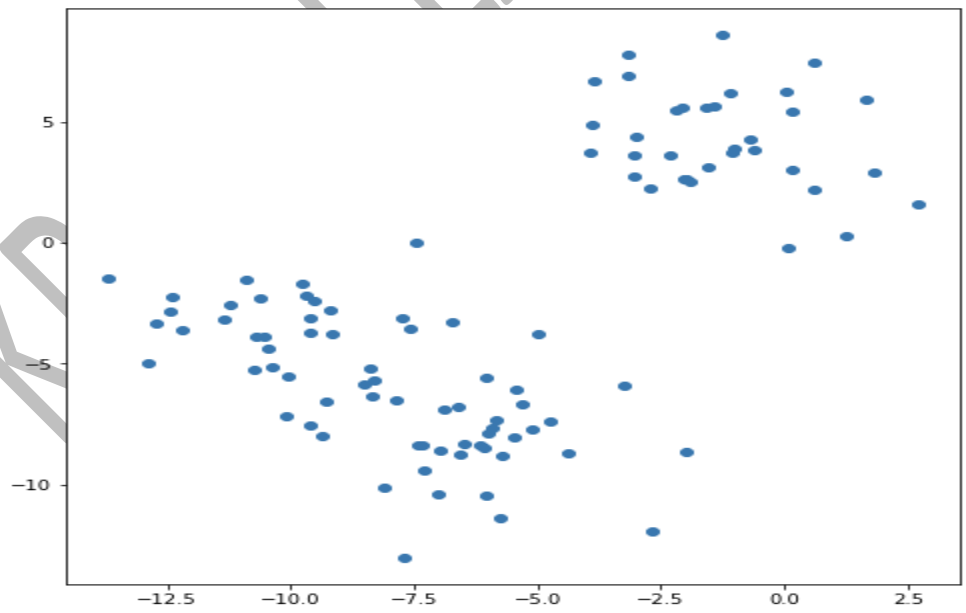
Perhaps if you encounter a dataset without a labeled target variable there's a way to glean some insight out of it without going through the hassle of labeling it. Maybe you simply don't have the time or resources to label the data or maybe there isn't a clear target variable to predict, is there still a way to use the data? This is the world of _unsupervised learning'. One of the more common goals of unsupervised learning is to cluster the data, to find reasonable groupings where the points in each group seem more similar to each other than to those in the other groups. Two of the most common clustering methods are K-means clustering and agglomerative clustering.

K-means clustering

So, you have some data, you don't have any category labels for it, but you're pretty sure you can segment the data into sensible groups, you just don't know what those groups are. Let's generate some data to experiment with:

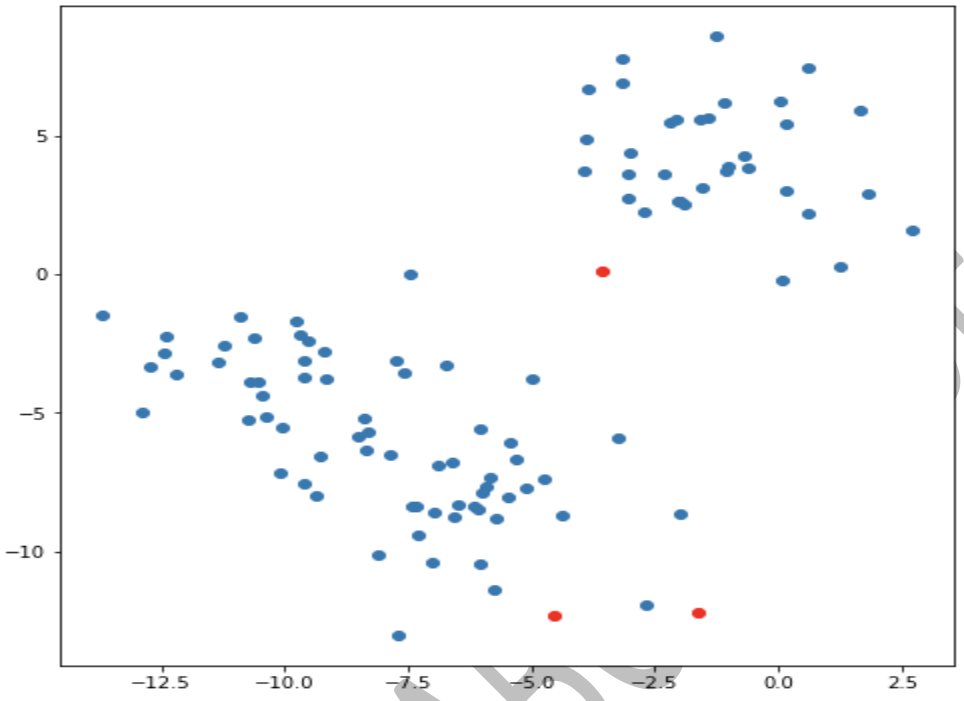


This data was **generated in three clumps, as you can see. For this example, that guarantees that there is a sensible grouping into three clusters, but remember that the point here is to see if you can find that grouping on your own, so let‘s remove the color coding:**

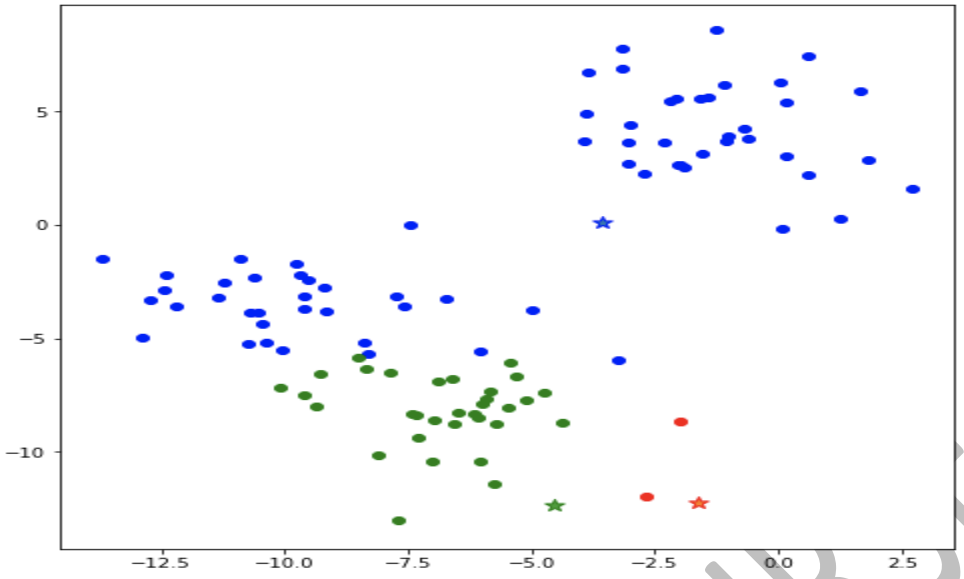


Visually you can probably still see the clusters, at least a little bit, but how do we get the computer to discover the clusters on its own? The first strategy we‘ll discuss is k-means. In the kmeans

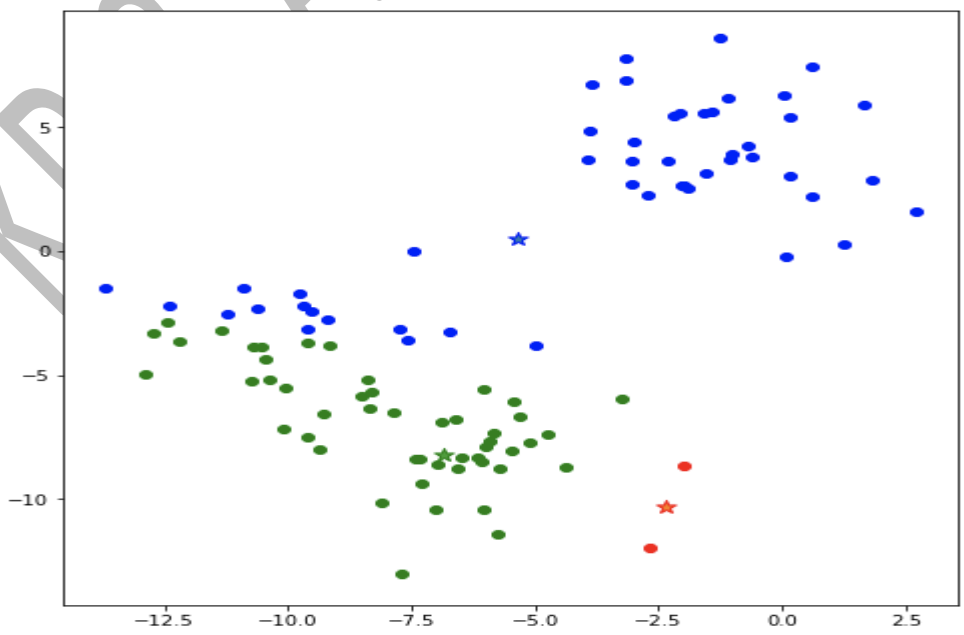
strategy, you first specify how many clusters you're looking for, in this case we'll say 3, and the computer starts by placing that many new points randomly on the graph:



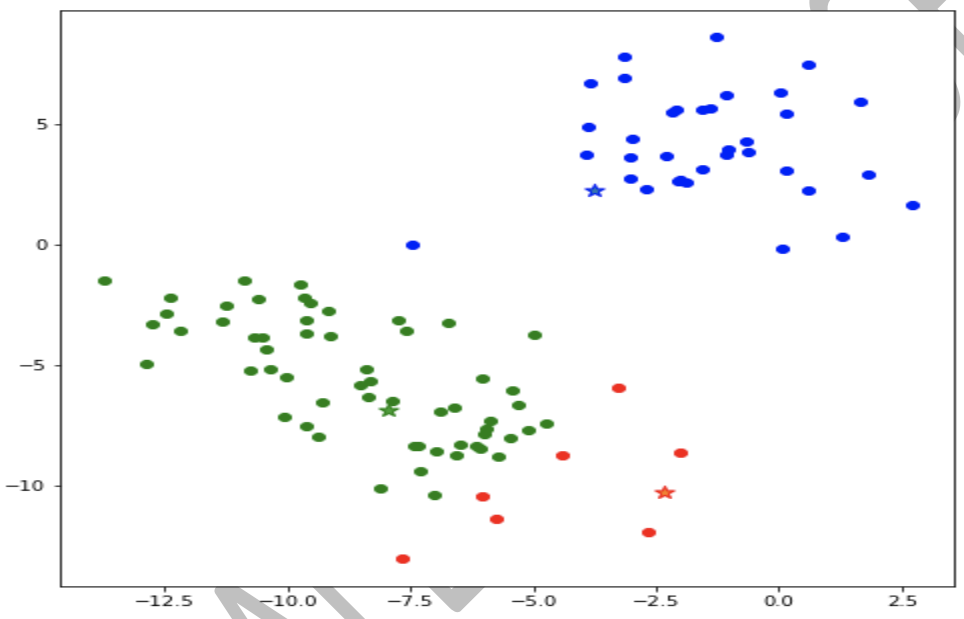
These new points are called _centroids' — they will be the centers of our clusters. Why place them randomly? Well, there isn't really a way for the computer to tell ahead of time where would be a sensible place to put the cluster centers, so the strategy is to place them randomly and then step by step move them to improve the clustering. To decide how to move the centroids, you first assign each of your points to the nearest centroid to it:



I‘ve made the centroids star shaped and then color coded the points according to which centroid is closest. You can see that the red and green centroids are near each other and towards the edge of the data, this doesn‘t seem to make a lot of sense — for one thing the red cluster only has two points in it. So we‘ll try moving these centroids someplace more central. We move each centroid to the center of the cluster, that is the average location within the cluster. Once we‘ve moved the centroids, we reassign the points to the nearest centroid. Here‘s what the new assignments look like one step into the process:



The red centroid did not move far, it still only has two points in its cluster. The green centroid, however, took a big step towards the center of the cluster. When it did that, it captured a bunch of points from the blue cluster. As a result, when we repeat this process, the blue centroid will move further into the top corner, leaving the bottom to the green and the red:



Que 02. Association algorithms for supervised classification on any dataset ?

->

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc. Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:

Association rule learning can be divided into three types of algorithms:

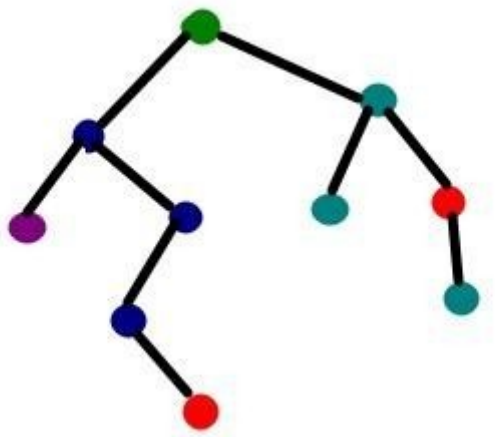
1. **Apriori**
2. **Eclat**
3. **F-P Growth Algorithm**

Que 03. Developing and implementing Decision Tree model on the dataset ?

->

Prerequisites: Decision Tree, DecisionTreeClassifier, sklearn, numpy, pandas

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



In this article, We are going to implement a Decision tree algorithm on the Balance Scale Weight & Distance Database presented on the UCI.

Data-set Description :

Title : Balance Scale Weight & Distance Database

Number of Instances : 625 (49 balanced, 288 left, 288 right)

Number of Attributes : 4 (numeric) +
class name = 5 Attribute
Information:

Class Name (Target variable): 3

L [balance scale tip to the left]

B [balance scale be balanced]

R [balance scale tip to the right]

Left-Weight: 5 (1, 2, 3, 4, 5)

Left-Distance: 5 (1, 2, 3, 4, 5)

Right-Weight: 5 (1,
2, 3, 4, 5) Right-
Distance: 5 (1, 2, 3,
4, 5) Missing
Attribute Values:
None Class
Distribution:

46.08 percent are L

07.84 percent are B

46.08 percent are R

You can find more details of the dataset here.

Used Python Packages:

sklearn :

In python, sklearn is a machine learning package which include a lot of ML algorithms.

Here, we are using some of its modules like train_test_split, DecisionTreeClassifier and accuracy_score.

NumPy :

It is a numeric python module which provides fast maths functions for calculations. It is used to read data in numpy arrays and for manipulation purpose.

Pandas :

Used to read and write different files.

Data manipulation can be done easily with dataframes.

Installation of the packages :

In Python, sklearn is the package which contains all the required packages to implement Machine learning algorithm. You can install the sklearn package by following the commands given below.

using pip :

pip install -U scikit-learn

Before using the above command make sure you have scipy and numpy packages installed.

If you don‘t have pip. You can install it using

python
get-
pip.py
using
conda :

conda install scikit-learn

Assumptions we make while using Decision tree :

At the beginning, we consider the whole training set as the root.

Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.

On the basis of attribute values records are distributed recursively.

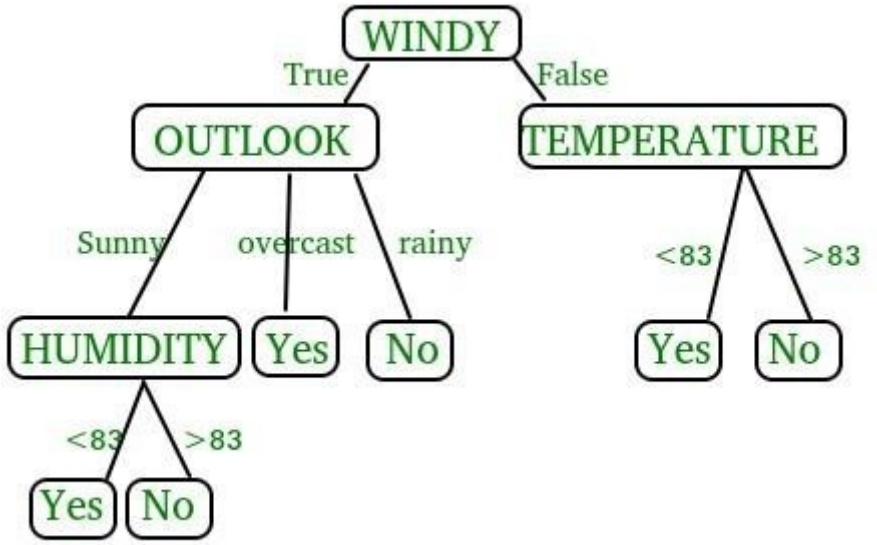
We use statistical methods for ordering attributes as root or internal node.

Pseudocode :

Find the best attribute and place it on the root node of the tree.

Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.

Find leaf nodes in all branches by repeating 1 and 2 on each subset.



While implementing the decision tree we will go through the following two phases:

Building Phase

Preprocess the dataset.

Split the dataset from train and test using Python sklearn package. Train the classifier. Operational Phase Make predictions.
Calculate the accuracy.

Data Import :

To import and manipulate the data we are using the pandas package provided in python.

Here, we are using a URL which is directly fetching the dataset from the UCI site no need to download the dataset. When you try to run this code on your system make sure the system should have an active Internet connection.

As the dataset is separated by —,|| so we have to pass the sep parameter‘s value as —,||.

Another thing is notice is that the dataset doesn‘t contain the header so we will pass the Header parameter‘s value as none. If we will not pass the header parameter then it will consider the first line of the dataset as the header.

Data Slicing :

Before training the model we have to split the dataset into the training and testing dataset. To split the dataset for training and testing we are using the sklearn module train_test_split First of all we have to separate the target variable from the attributes in the dataset.

X **= balance_data.values[:, 1:5]**
Y **= balance_data.values[:,0]**

Above are the lines from the code which separate the dataset. The variable X contains the attributes while the variable Y contains the target variable of the dataset.

Next step is to split the dataset for training and testing purpose.

X_train, X_test, y_train, y_test = train_test_split(

X, Y, test_size = 0.3, random_state = 100)

Above line split the dataset for training and testing. As we are splitting the dataset in a ratio of 70:30 between training and testing so we are pass test_size parameter's value as 0.3.

random_state variable is a pseudo-random number generator state used for random sampling.

Terms used in code :

Gini index and information gain both of these methods are used to select from the n attributes of

the dataset which attribute would be placed at the root node or the internal node.

Gini index:

$$\text{Gini Index} = 1 - \sum_j p_j^2$$

\text { Gini Index }=1-\sum_{j}{ }_{\mathrm{j}}^{2}

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.

It means an attribute with lower gini index should be preferred.

Sklearn supports —gini|| criteria for Gini Index and by default, it takes —gini|| value. Entropy:

if a random variable x can take N different value , the i'value

x_{i} with probability

$\mathrm{p}\left(\mathrm{x}_{\mathrm{ii}}\right)$,we can associate the following entropy with x : $$ H(x)=-\sum_{i=1}^{N} p\left(x_{i}\right) \log _{2} p\left(x_{i}\right) $$

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content. Information Gain

Definition: Suppose S is a set of instances, A is an attribute,

S_{v} is the subset of s with $\mathrm{A}=\mathrm{v}$ and Values(A) is the set of all possible of A,then

The entropy typically changes when we use a node in a decision tree to partition the training

instances into smaller subsets. Information gain is a measure of this change in entropy.

Sklearn supports —entropy|| criteria for Information Gain and if we want to use Information Gain method in sklearn then we have to mention it explicitly.

Accuracy score

Accuracy score is used to calculate the accuracy of the trained classifier.

Confusion Matrix

Confusion Matrix is used to understand the trained classifier behavior over the test dataset or validate dataset.

Below is the python code for the decision tree.

```
# Run this program on
your local python #
interpreter, provided
you have installed #
the required libraries.

# Importing the required
packages import numpy as np
import pandas as pd from
sklearn.metrics import
confusion_matrix from
sklearn.cross_validation
import train_test_split from
sklearn.tree import
DecisionTreeClassifier from
sklearn.metrics import
accuracy_score
```

```
from sklearn.metrics import classification_report


# Function
importing
Dataset def
importdata():
balance_data
= pd.read_csv(
'https://archive.ics.uci.edu/
ml/machine-learning-'+
'databases/balance-
scale/balance-scale.data',
sep= ',', header = None)


  # Printing the dataswet
shape   print ("Dataset
Length: ",
len(balance_data))
  print ("Dataset Shape: ", balance_data.shape)


  #   Printing   the
dataset  obseravtions
print       ("Dataset:
",balance_data.head(
))                 return
balance_data

# Function to
split the dataset
def
```

```
splitdataset(bal
ance_data):

    # Separating the target variable
X = balance_data.values[:, 1:5]
Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.3, random_state = 100)

    return X, Y, X_train, X_test, y_train, y_test

# Function to perform
training with giniIndex. def
train_using_gini(X_train,
X_test, y_train):

    # Creating the classifier
object     clf_gini =
DecisionTreeClassifier(criterion
= "gini",
        random_state = 100,max_depth=3,
min_samples_leaf=5)

    #
Performing
training
clf_gini.fit(X_t
```

```
rain,   y_train)
return clf_gini

# Function to perform
training with entropy. def
tarin_using_entropy(X_train
, X_test, y_train):

  # Decision tree with
entropy clf_entropy =
DecisionTreeClassifier(
criterion = "entropy",
random_state = 100,
        max_depth = 3, min_samples_leaf = 5)

  # Performing
training
clf_entropy.fit(X
_train,   y_train)
return
clf_entropy

# Function to
make predictions
def
prediction(X_test
, clf_object):
```

```
    # Predicton on
test with giniIndex
y_pred =
clf_object.predict(X
_test)
print("Predicted
values:")
print(y_pred)
    return y_pred


# Function to
calculate accuracy
def
cal_accuracy(y_te
st, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))

#
Driv
er
code
```

```
def
main
():


   #
Building
Phase
data =
importda
ta()

   X, Y, X_train, X_test, y_train, y_test =
splitdataset(data)    clf_gini =
train_using_gini(X_train, X_test, y_train)

   clf_entropy = tarin_using_entropy(X_train, X_test, y_train)


   # Operational
Phase
print("Results
Using Gini Index:")


   # Prediction using gini
y_pred_gini                =
prediction(X_test,
clf_gini)
cal_accuracy(y_test,
y_pred_gini)
print("Results        Using
Entropy:")    # Prediction
using                entropy
y_pred_entropy          =
```

```
prediction(X_test,
clf_entropy)
   cal_accuracy(y_test, y_pred_entropy)


#       Calling
main
function    if
__name__=
="__main__
":        main()
Data
Infomation:


Dataset Length:  625

Dataset Shape:  (625, 5)

Dataset:    0  1  2  3  4

0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results
Using Gini
Index:


Predicted values:

['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
```

'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'

'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L'

'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'L'

'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'

'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'

'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'

'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]

[0 67 18]

[0 19 71]]

Accuracy :

73.4042553191

Report : precision

recall f1-score

support B 0.00

0.00 0.00 13

L 0.73 0.79

0.76 85 R

0.74 0.79

0.76 90

avg/total 0.68 0.73 0.71 188

Results Using Entropy:

Predicted values:

['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]
[0 63 22]
[0 20 70]]

Accuracy : 70.7446808511

Report :

	precision	recall	f1-score	support	B
	0.00	0.00	0.00	13	

L 0.71 0.74
0.72 85 R
0.71 0.78 0.74
90

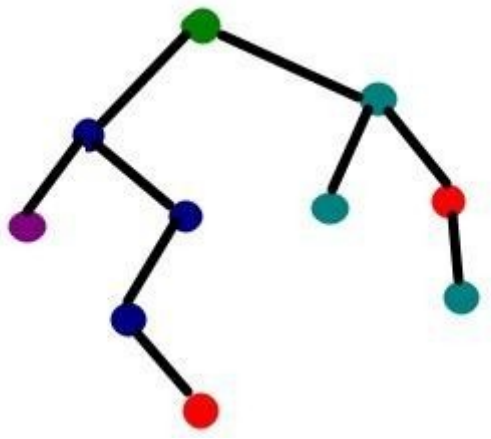
avg / total 0.66 0.71 0.68 188

Que 04. Bayesian classification on any dataset ?

->

Prerequisites: Decision Tree, DecisionTreeClassifier, sklearn, numpy, pandas

Decision Tree is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



In this article, We are going to implement a Decision tree algorithm on the Balance Scale Weight & Distance Database presented on the UCI.

Data-set Description :

Title : Balance Scale Weight & Distance Database

Number of Instances : 625 (49 balanced, 288 left, 288 right)
Number of Attributes : 4 (numeric) + class name = 5 Attribute Information:

Class Name (Target variable): 3

L [balance scale tip to the left]

B [balance scale be balanced]

R [balance scale tip to the right]

Left-Weight: 5 (1, 2, 3, 4, 5)

Left-Distance: 5 (1, 2, 3, 4, 5)

Right-Weight: 5 (1, 2, 3, 4, 5) Right-Distance: 5 (1, 2, 3, 4, 5) Missing Attribute Values: None Class Distribution:

46.08 percent are L

07.84 percent are B

46.08 percent are R

You can find more details of the dataset here.

Used Python Packages:

sklearn :

In python, sklearn is a machine learning package which include a lot of ML algorithms.

Here, we are using some of its modules like train_test_split, DecisionTreeClassifier and accuracy_score.

NumPy :

It is a numeric python module which provides fast maths functions for calculations. It is used to read data in numpy arrays and for manipulation purpose.

Pandas :

Used to read and write different files.

Data manipulation can be done easily with dataframes.

Installation of the packages :

In Python, sklearn is the package which contains all the required packages to implement Machine learning algorithm. You can install the sklearn package by following the commands given below.

using pip :

pip install -U scikit-learn

Before using the above command make sure you have scipy and numpy packages installed.

If you don‘t have pip. You can install it using

python
get-
pip.py

using
conda :

conda install scikit-learn

Assumptions we make while using Decision tree :

At the beginning, we consider the whole training set as the root.

Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.

On the basis of attribute values records are distributed recursively.

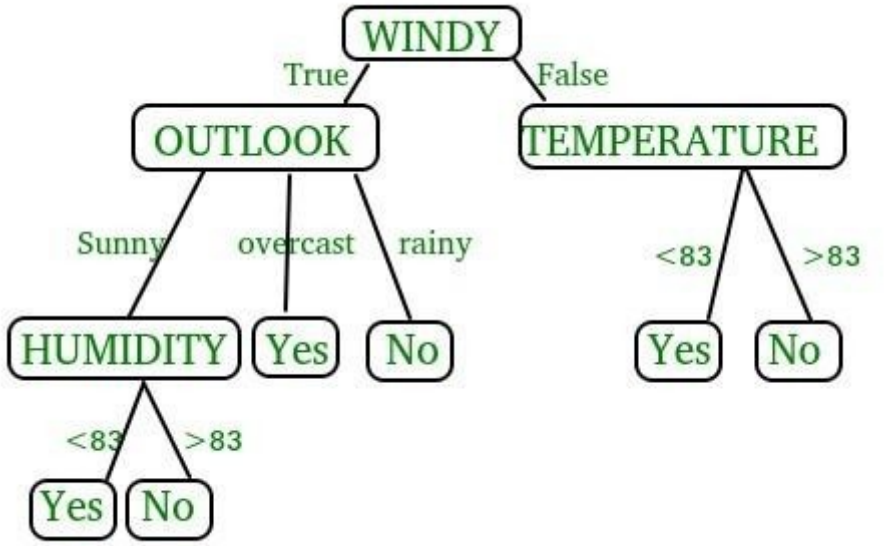
We use statistical methods for ordering attributes as root or internal node.

Pseudocode :

Find the best attribute and place it on the root node of the tree.

Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.

Find leaf nodes in all branches by repeating 1 and 2 on each subset.



While implementing the decision tree we will go through the following two phases:

Building Phase

Preprocess the dataset.

Split the dataset from train and test using Python sklearn package. Train the classifier. Operational Phase Make predictions.
Calculate the accuracy.

Data Import :

To import and manipulate the data we are using the pandas package provided in python.

Here, we are using a URL which is directly fetching the dataset from the UCI site no need to download the dataset. When you try to run this code on your system make sure the system should have an active Internet connection.

As the dataset is separated by —,|| so we have to pass the sep parameter‘s value as —,||.

Another thing is notice is that the dataset doesn‘t contain the header so we will pass the Header parameter‘s value as none. If we will not pass the header parameter then it will consider the first line of the dataset as the header.

Data Slicing :

Before training the model we have to split the dataset into the training and testing dataset. To split the dataset for training and testing we are using the sklearn module train_test_split First of all we have to separate the target variable from the attributes in the dataset.

Z = balance_data.values[:, 1:5]

AA = balance_data.values[:,0]

Above are the lines from the code which separate the dataset. The variable X contains the attributes while the variable Y contains the target variable of the dataset.

Next step is to split the dataset for training and testing purpose.

X_train, X_test, y_train, y_test = train_test_split(

X, Y, test_size = 0.3, random_state = 100)

Above line split the dataset for training and testing. As we are splitting the dataset in a ratio of 70:30 between training and testing so we are pass test_size parameter‘s value as 0.3.

random_state variable is a pseudo-random number generator state used for random sampling.

Terms used in code :

Gini index and information gain both of these methods are used to select from the n attributes of

the dataset which attribute would be placed at the root node or the internal node.

Gini index:

$$\text{Gini Index} = 1 - \sum_j p_j^2$$

\text { Gini Index }=1-\sum_{j}{ }_{\mathrm{j}}^{2}

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.

It means an attribute with lower gini index should be preferred.

Sklearn supports —gini|| criteria for Gini Index and by default, it takes —gini|| value. Entropy:

if a random variable x can take N different value , the i'value

x_{i} with probability

$\mathrm{p}\left(\mathrm{x}_{\mathrm{ii}}\right)$,we can associate the following entropy with x : $$ H(x)=-\sum_{i=1}^{N} p\left(x_{i}\right) \log _{2} p\left(x_{i}\right) $$

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy the more the information content. Information Gain

Definition: Suppose S is a set of instances, A is an attribute,

S_{v} is the subset of s with $\mathrm{A}=\mathrm{v}$ and Values(A) is the set of all possible of A,then

The entropy typically changes when we use a node in a decision tree to partition the training

instances into smaller subsets. Information gain is a measure of this change in entropy.

Sklearn supports —entropy|| criteria for Information Gain and if we want to use Information Gain method in sklearn then we have to mention it explicitly.

Accuracy score

Accuracy score is used to calculate the accuracy of the trained classifier.

Confusion Matrix

Confusion Matrix is used to understand the trained classifier behavior over the test dataset or validate dataset.

Below is the python code for the decision tree.

```
# Run this program on
your local python #
interpreter, provided
```

```
you  have  installed  #
the required libraries.

# Importing the required
packages import numpy as np
import pandas as pd from
sklearn.metrics import
confusion_matrix from
sklearn.cross_validation
import train_test_split from
sklearn.tree import
DecisionTreeClassifier from
sklearn.metrics import
accuracy_score
from sklearn.metrics import classification_report

# Function
importing
Dataset def
importdata():
balance_data
= pd.read_csv(
'https://archive.ics.uci.edu/
ml/machine-learning-'+
'databases/balance-
scale/balance-scale.data',
sep= ',', header = None)

   # Printing the dataswet
shape     print ("Dataset
```

```
Length: ",
len(balance_data))
   print ("Dataset Shape: ", balance_data.shape)

   # Printing the
dataset obseravtions
print ("Dataset:
",balance_data.head(
)) return
balance_data

# Function to
split the dataset
def
splitdataset(bal
ance_data):

   # Separating the target variable
Z = balance_data.values[:, 1:5]
AA    = balance_data.values[:, 0]

   # Splitting the dataset into train and test
   X_train, X_test, y_train, y_test = train_test_split(
   X, Y, test_size = 0.3, random_state = 100)

   return X, Y, X_train, X_test, y_train, y_test

# Function to perform
training with giniIndex. def
```

```
train_using_gini(X_train,
X_test, y_train):


   # Creating the classifier
object     clf_gini =
DecisionTreeClassifier(criterion
= "gini",
         random_state = 100,max_depth=3,
min_samples_leaf=5)


   #
Performing
training
clf_gini.fit(X_t
rain,    y_train)
return clf_gini

# Function to perform
training with entropy. def
tarin_using_entropy(X_train
, X_test, y_train):

  #    Decision    tree    with
entropy          clf_entropy    =
DecisionTreeClassifier(
criterion      =      "entropy",
random_state = 100,
         max_depth = 3, min_samples_leaf = 5)
```

```
    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
```

```
    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))

#
Driv
er
code
def
main
():

    #
Building
Phase
data =
importda
ta()
    X, Y, X_train, X_test, y_train, y_test =
splitdataset(data)     clf_gini =
train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
```

```
    # Operational
Phase
print("Results
Using Gini Index:")


   #    Prediction    using    gini
y_pred_gini =  prediction(X_test,
clf_gini)        cal_accuracy(y_test,
y_pred_gini)        print("Results
Using Entropy:")    # Prediction
using entropy    y_pred_entropy
= prediction(X_test, clf_entropy)
   cal_accuracy(y_test, y_pred_entropy)



#   Calling   main
function          if
__name__=="__
main__":
main()        Data
Infomation:


Dataset Length:  625

Dataset Shape:  (625, 5)

Dataset:    0  1  2  3  4

5  B  1  1  1  1
6  R  1  1  1  2
7  R  1  1  1  3
8  R  1  1  1  4
9  R  1  1  1  5
Results Using
Gini Index:


Predicted values:

['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'

 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
```

'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L'
'R' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L'
'R'

'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'

'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'

'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'

'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

Confusion Matrix: [[0 6 7]

[0 67 18]

[0 19 71]]

Accuracy : 73.4042553191
Report : precision recall
f1-score support B 0.00
0.00 0.00 13
L 0.73 0.79 0.76
85 R 0.74 0.79
0.76 90

avg/total 0.68 0.73 0.71 188

Results Using Entropy:

Predicted values:

['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'

'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'

'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'

'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'

'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'

'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'

'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'

```
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'

 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'

 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']


Confusion Matrix:  [[ 0  6  7]

                    [ 0 63 22]

                    [ 0 20 70]]

Accuracy :  70.7446808511

Report :

        precision    recall  f1-score
support    B       0.00      0.00      0.00
13

   L       0.71      0.74      0.72
85     R       0.71      0.78
0.74       90

avg / total 0.66      0.71      0.68      188
```

Que 05. Bayesian classification on any dataset ?

->

This article discusses the theory behind the Naive Bayes classifiers and their implementation.

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

To start with, let us consider a dataset.

Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each

tuple classifies the conditions as fit(—Yes‖) or unfit(—No‖) for playing golf.

Here is a tabular representation of our dataset.

	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

The dataset is divided into two parts, namely, feature matrix and the response vector.

Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of dependent features. In above dataset, features are _Outlook‘, _Temperature‘, _Humidity‘ and _Windy‘. Response vector contains the value of class variable(prediction or output) for each row of feature matrix. In above dataset, the class variable name is _Play golf‘.
Assumption:

The fundamental Naive Bayes assumption is that each feature makes an:

independent
equal
contribution
to the
outcome.

With relation to our dataset, this concept can be understood as:

We assume that no pair of features are dependent. For example, the temperature being _Hot‘ has nothing to do with the humidity or the outlook being _Rainy‘ has no effect on the winds. Hence, the features are assumed to be independent.

Secondly, each feature is given the same weight(or importance). For example, knowing only temperature and humidity alone can‘t predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing equally to the outcome.

Note: The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Now, before moving to the formula for Naive Bayes, it is important to know about Bayes‘ theorem.

Bayes‘ Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

where A and B are events and P(B) ≠ 0.

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).

P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen. Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = \frac{P(X|y)\,P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector (of size n) where:

$$X = (x_1,x_2,x_3,.....,x_n)$$

Just to clear, an example of a feature vector and corresponding class variable can be: (refer 1st row of dataset)

X = (Rainy,
Hot, High,
False) y = No

So basically, P(y|X) here means, the probability of —Not playing golf‖ given that the weather conditions are —Rainy outlook‖, —Temperature is hot‖, —high humidity‖ and —no wind‖.

Naive assumption

Now, its time to put a naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts.

Now, if any two events A and B are independent, then,

P(A,B) = P(A)P(B)

Hence, we reach to the result:

P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}

which can be expressed as:

P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n}P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}

Now, as the denominator remains constant for a given input, we can remove that term:

P(y|x_1,...,x_n)\propto P(y)\prod_{i=1}^{n}P(x_i|y)

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable y and pick up the output with maximum probability. This can be expressed mathematically as:

y = argmax_{y} P(y)\prod_{i=1}^{n}P(x_i|y)

So, finally, we are left with the task of calculating P(y) and P(xi | y).

Please note that P(y) is also called class probability and P(xi | y) is called conditional probability.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y).

Let us try to apply the above formula manually on our weather dataset. For this, we need to do some precomputations on our dataset.

We need to find P(xi | yj) for each xi in X and yj in y. All these calculations have been demonstrated in the tables below:

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

table23

So, in the figure above, we have calculated P(xi | yj) for each xi in X and yj in y manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e P(temp. = cool | play golf = Yes) = 3/9.

Also, we need to find class probabilities (P(y)) which has been calculated in the table 5. For example, P(play golf = Yes) = 9/14.

So now, we are done with our pre-computations and the classifier is ready!

Let us test it on a new set of features (let us call it today):

**today = (Sunny, Hot,
Normal, False) So,
probability of playing
golf is given by:**

$$P(Yes \mid today) = \frac{P(Sunny\ Outlook|Yes)P(Hot\ Temperature|Yes)P(Normal\ Humidity|Yes)P(No\ Wind|Yes)P(Yes)}{P(today)}$$

and probability to not play golf is given by:

$$P(No \mid today) = \frac{P(Sunny\ Outlook|No)P(Hot\ Temperature|No)P(Normal\ Humidity|No)P(No\ Wind|No)P(No)}{P(today)}$$

Since, P(today) is common in both probabilities, we can ignore P(today) and find proportional probabilities as:

$$P(Yes \mid today) \propto \frac{2}{9}.\frac{2}{9}.\frac{6}{9}.\frac{6}{9}.\frac{9}{14} \approx 0.0141$$

and

$$P(No \mid today) \propto \frac{3}{5}.\frac{2}{5}.\frac{1}{5}.\frac{2}{5}.\frac{5}{14} \approx 0.0068$$

Now, since

$$P(Yes \mid today) + P(No \mid today) = 1$$

These numbers can be converted into a probability by making the sum equal to 1 (normalization):

$$P(Yes \mid today) = \frac{0.0141}{0.0141 + 0.0068} = 0.67$$

and

$$P(No \mid today) = \frac{0.0068}{0.0141 + 0.0068} = 0.33$$

Since

$$P(Yes \mid today) > P(No \mid today)$$

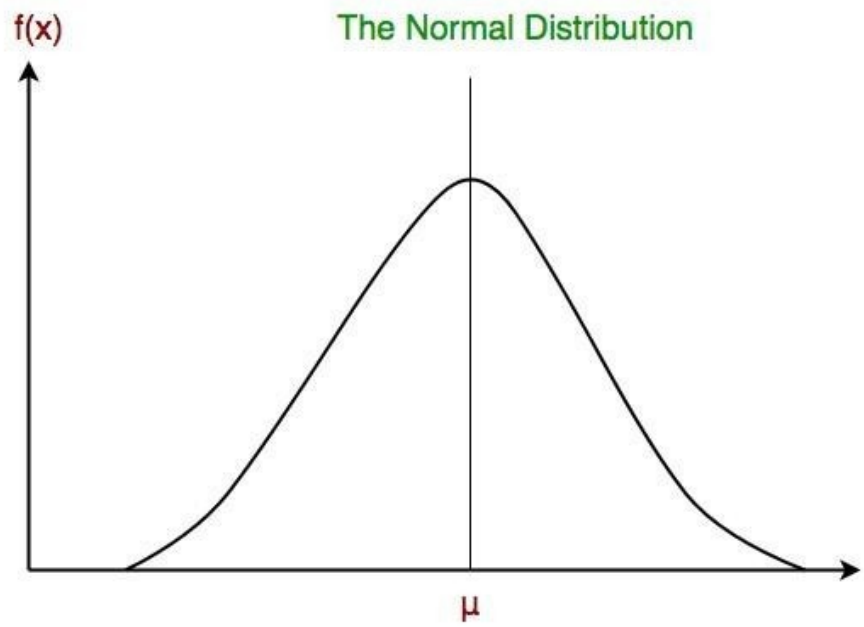
So, prediction that golf would be played is _Yes‘.

The method that we discussed above is applicable for discrete data. In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi | y).

Now, we discuss one of such classifiers here.

Gaussian Naive Bayes classifier

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:



normal

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma _{y}^{2} }} exp \left (- \frac{(x_i-\mu _{y})^2}{2\sigma _{y}^{2}} \right)$$

Now, we look at an implementation of Gaussian Naive Bayes classifier using scikit-learn.

```
# load the iris dataset

from sklearn.datasets
import load_iris iris =
load_iris()


# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target


# splitting X and y into training
and testing sets from
sklearn.model_selection
import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)


# training the model on
training set from
sklearn.naive_bayes
import GaussianNB gnb =
GaussianNB()
gnb.fit(X_train, y_train)


# making predictions on
the testing set y_pred =
gnb.predict(X_test)
```

 # comparing actual response values (y_test) with predicted response values (y_pred) from sklearn import metrics print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100) Output:

Gaussian Naive Bayes model accuracy(in %): 95.0

Que 06. SVM classification on any dataset ?

->

Support Vector Machines (SVMs in short) are machine learning algorithms that are used for classification and regression purposes. SVMs are one of the powerful machine learning algorithms for classification, regression and outlier detection purposes. An SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier.

The original SVM algorithm was developed by Vladimir N Vapnik and Alexey Ya. Chervonenkis in 1963. At that time, the algorithm was in early stages. The only possibility is to draw hyperplanes for linear classifier. In 1992, Bernhard E. Boser, Isabelle M Guyon and Vladimir N Vapnik suggested a way to create non-linear classifiers by applying the kernel trick to maximum-margin hyperplanes. The current standard was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.

SVMs can be used for linear classification purposes. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using the kernel trick. It enable us to implicitly map the inputs into high dimensional feature spaces.

2. Support Vector Machines intuition

Table of Contents

Now, we should be familiar with some SVM terminology.

Hyperplane

A hyperplane is a decision boundary which separates between given set of data points having different class labels. The SVM classifier separates data points using a hyperplane with the maximum amount of margin. This hyperplane is known as the `maximum margin hyperplane` and the linear classifier it defines is known as the `maximum margin classifier`.

Support Vectors

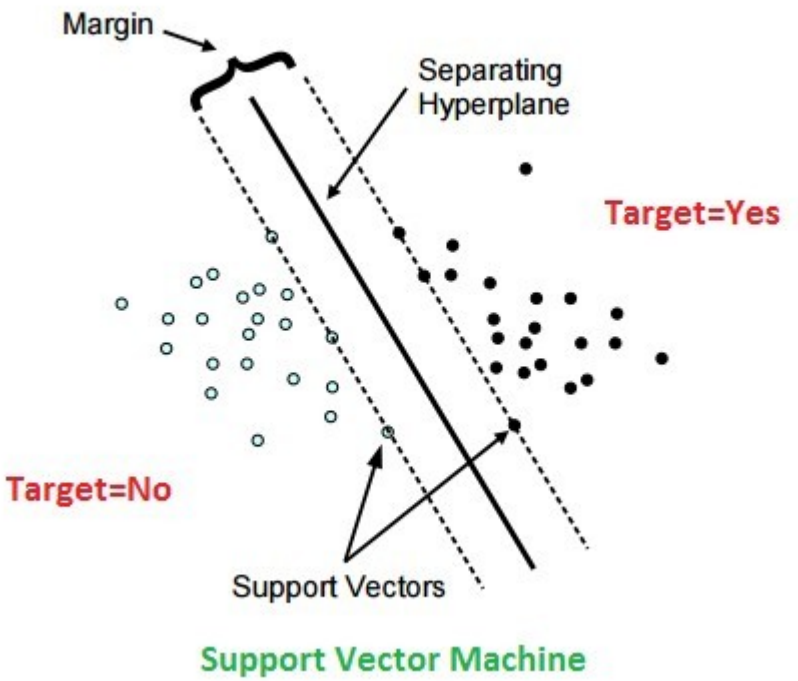
Support vectors are the sample data points, which are closest to the hyperplane. These data points will define the separating line or hyperplane better by calculating margins.

Margin

A margin is a separation gap between the two lines on the closest data points. It is calculated as the perpendicular distance from the line to support vectors or closest data points. In SVMs, we try to maximize this separation gap so that we get maximum margin.

The following diagram illustrates these concepts visually.

Margin in SVM



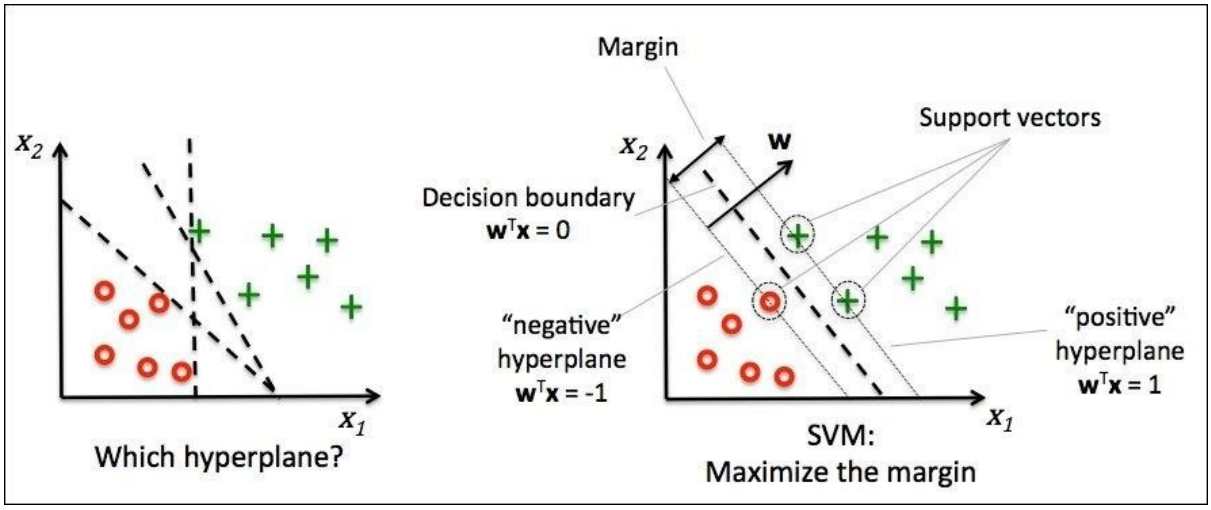
SVM Under the hood

In SVMs, our main objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum margin hyperplane in the following 2 step process –

1. Generate hyperplanes which segregates the classes in the best possible way. There are many hyperplanes that might classify the data. We should look for the best hyperplane that represents the largest separation, or margin, between the two classes.
2. So, we choose the hyperplane so that distance from it to the support vectors on each side is maximized. If such a hyperplane exists, it is known as the maximum margin hyperplane and the linear classifier it defines is known as a maximum margin classifier.

The following diagram illustrates the concept of maximum margin and maximum margin hyperplane in a clear manner.

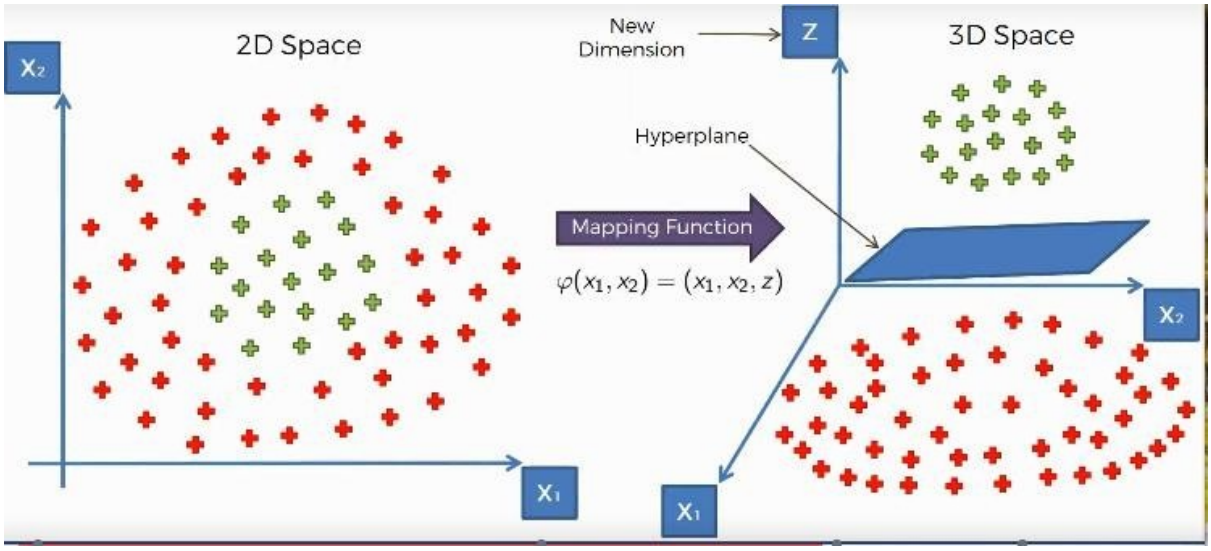
Maximum margin hyperplane



Problem with dispersed datasets

Sometimes, the sample data points are so dispersed that it is not possible to separate them using a linear hyperplane. In such a situation, SVMs uses a `kernel trick` to transform the input space to a higher dimensional space as shown in the diagram below. It uses a mapping function to transform the 2-D input space into the 3-D input space. Now, we can easily segregate the data points using linear separation.

Kernel trick - transformation of input space to higher dimensional space



3. Kernel trick

Table of Contents

In practice, SVM algorithm is implemented using a `kernel`. It uses a technique called the `kernel trick`. In simple words, a `kernel` is just a function that maps the data to a higher dimension where data is separable. A kernel transforms a low-dimensional input data space into a higher dimensional space. So, it converts non-linear separable problems to linear separable problems by adding more dimensions to it. Thus, the kernel trick helps us to build a more accurate classifier. Hence, it is useful in non-linear separation problems.

We can define a kernel function as follows-

Kernel function

$$K\left(\overline{x}\right) = \begin{matrix} 1 & \text{if} \left\|\overline{x}\right\| \le 1 \\ 0 & \text{otherwise} \end{matrix}$$

In the context of SVMs, there are 4 popular kernels – `Linear kernel`,`Polynomial kernel`,`Radial Basis Function (RBF) kernel` (also called Gaussian kernel) and `Sigmoid kernel`. These are described below -

3.1 Linear kernel

In linear kernel, the kernel function takes the form of a linear function as follows-

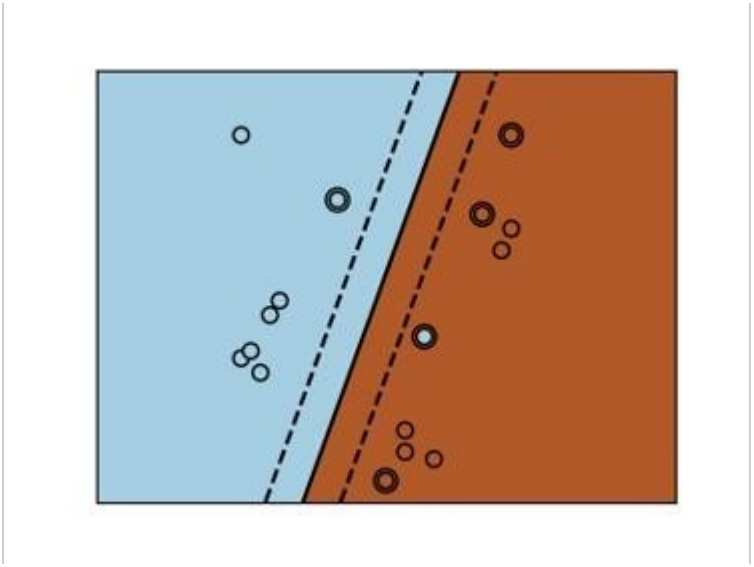
linear kernel : K(xi , xj) = xiT xj

Linear kernel is used when the data is linearly separable. It means that data can be separated using a single line. It is one of the most common kernels to be used. It is mostly used when there are large number of features in a dataset. Linear kernel is often used for text classification purposes.

Training with a linear kernel is usually faster, because we only need to optimize the C regularization parameter. When training with other kernels, we also need to optimize the γ parameter. So, performing a grid search will usually take more time.

Linear kernel can be visualized with the following figure.

Linear Kernel



3.2 Polynomial Kernel

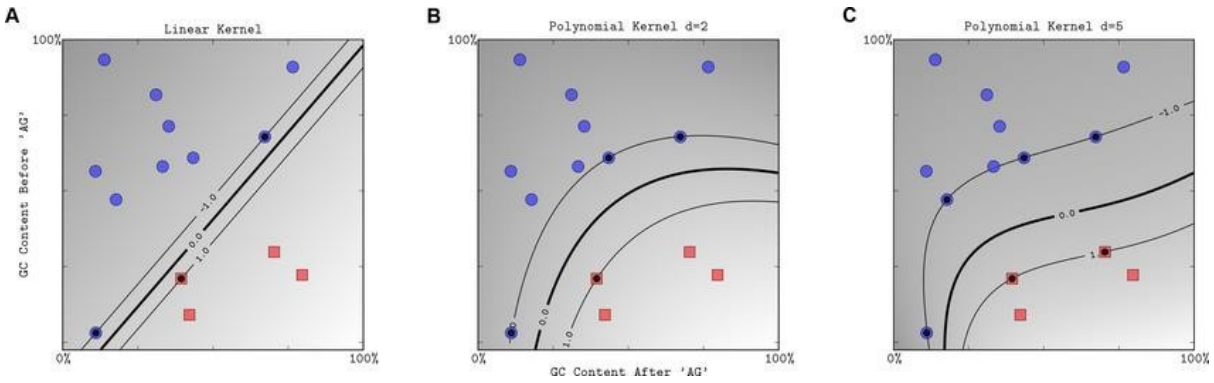
Polynomial kernel represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables. The polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of the input samples.

For degree-d polynomials, the polynomial kernel is defined as follows –

Polynomial kernel : $K(x_i , x_j) = (\gamma x_i^T x_j + r)^d , \gamma > 0$

Polynomial kernel is very popular in Natural Language Processing. The most common degree is d = 2 (quadratic), since larger degrees tend to overfit on NLP problems. It can be visualized with the following diagram.

Polynomial Kernel



3.3 Radial Basis Function Kernel

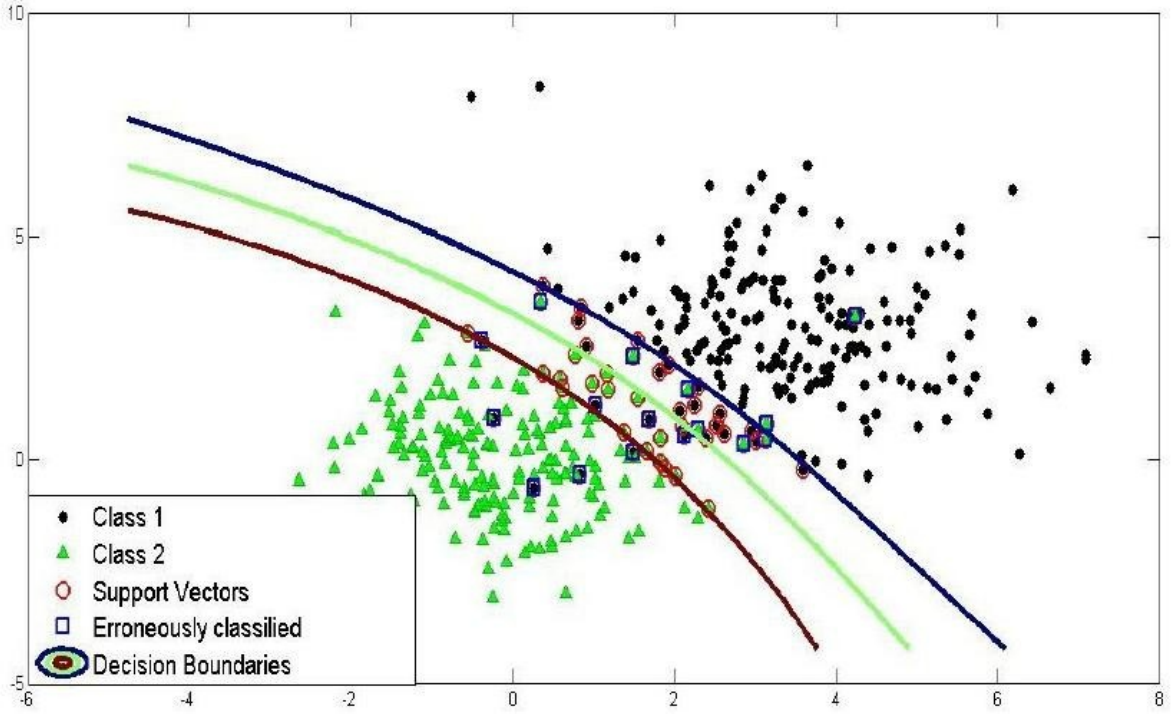
Radial basis function kernel is a general purpose kernel. It is used when we have no prior knowledge about the data. The RBF kernel on two samples x and y is defined by the following equation –

Radial Basis Function kernel

$$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

The following diagram demonstrates the SVM classification with rbf kernel.

SVM Classification with rbf kernel



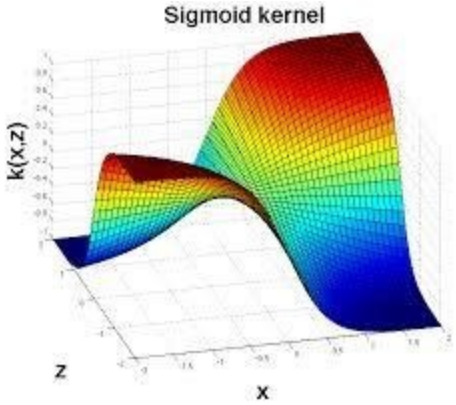
3.4 Sigmoid kernel

Sigmoid kernel has its origin in neural networks. We can use it as the proxy for neural networks. Sigmoid kernel is given by the following equation –

sigmoid kernel : k (x, y) = tanh(αxTy + c)

Sigmoid kernel can be visualized with the following diagram-

Sigmoid kernel



4. SVM Scikit-Learn libraries

Table of Contents

Scikit-Learn provides useful libraries to implement Support Vector Machine algorithm on a dataset. There are many libraries that can help us to implement SVM smoothly. We just need to call the library with parameters that suit to our needs. In this project, I am dealing with a classification task. So, I will mention the Scikit-Learn libraries for SVM classification purposes.

First, there is a LinearSVC() classifier. As the name suggests, this classifier uses only linear kernel. In LinearSVC() classifier, we don't pass the value of kernel since it is used only for linear classification purposes.

Scikit-Learn provides two other classifiers
- SVC() and NuSVC() which are used for classification purposes. These classifiers are mostly similar with some difference in parameters. NuSVC() is similar to SVC() but uses a parameter to control the number of support vectors. We pass the values of kernel, gamma and C along with other parameters. By default kernel parameter uses rbf as its value but we can pass values like poly, linear, sigmoid or callable function.

5. Dataset description

Table of Contents

I have used the Predicting a Pulsar Star dataset for this project.

Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter. Classification algorithms in particular are being adopted, which treat the data sets as binary classification problems. Here the legitimate pulsar examples form minority positive class and spurious examples form the majority negative class.

The data set shared here contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples. Each row lists the variables first, and the class label is the final entry. The class labels used are 0 (negative) and 1 (positive).

Attribute Information:

Each candidate is described by 8 continuous variables, and a single class variable. The first four are simple statistics obtained from the integrated pulse profile. The remaining four variables are similarly obtained from the DM-SNR curve . These are summarised below:

1. Mean of the integrated profile.
2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.
9. Class

6. Import libraries

Table of Contents

I will start off by importing the required Python libraries.

In [1]:

```
# This Python 3 environment comes with many helpful
analytics libraries installed
# It is defined by the kaggle/python docker image:
https://github.com/kaggle/docker-python
```

```
# For example, here's several helpful packages to l
oad in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
(e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualiz
ation
import seaborn as sns # for statistical data visual
ization
%matplotlib inline

# Input data files are available in the "../input/"
directory.
# For example, running this (by clicking run or pre
ssing Shift+Enter) will list all files under the in
put directory

import os
for dirname, _, filenames in os.walk('/kaggle/input
'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory ar
e saved as output.
```

```
/kaggle/input/predicting-a-pulsar-star/pulsar_stars
.csv
```

In [2]:

```
import warnings

warnings.filterwarnings('ignore')
```

7.Import dataset

Table of Contents

In [3]:

```
data = '/kaggle/input/predicting-a-pulsar-star/puls
ar_stars.csv'

df = pd.read_csv(data)
```

8. Exploratory data analysis

Table of Contents

Now, I will explore the data to gain insights about the data.

In [4]:

```
# view dimensions of dataset

df.shape
```

Out[4]:

```
(17898, 9)
```

We can see that there are 17898 instances and 9 variables in the data set.

In [5]:

```
# let's preview the dataset

df.head()
```

Out[5]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

Que 07. Text Mining algorithms on unstructured dataset ?

->

First we must understand what unstructured data is, and how we can shape and mold it to fit our business purposes. Unstructured data is everything that touches your business and can be measured. Sometimes, it may already include structure - receipts, payroll, consumer opinion polls, for example.

More often than not the text data you want to use for analysis will not have structure.

Think about things like a giant comment thread on social media, customer surveys with short answer boxes, emails from customers, reviews on third party websites... or anything else your business might care to keep tabs on.

In short, unstructured data is just about everything we can find out in the world. While the examples above limit our analysis to only textual mediums, a dynamic 21st century enterprise cannot afford to keep such blinders on.

Create structured data from unstructured data

Of course boundaries are needed, and the first step to generating powerful insights is to give structure to big, unstructured datasets. This means converting the dataset from whatever medium it exists in its original form (video, audio, image, text) into text.

does this in a variety of ways including by identifying logos in images and videos, but also objects (person, dog, tie, cars, etc.) and scenes (airplane, bar, mountain etc.). And also in audio through speech-to-text conversion. This allows for a broad variety of text documents to incorporate into your data analysis.

When using this process happens behind the scenes and is seamless, appearing before you in seconds. The other alternative is to learn programming languages like R and Python and integrate them into your own machine learning algorithm.

Summarization is one natural language processing and

visualization technique you can apply. This helps you to understand the scope of your dataset and can immediately speed up your time to insight from text analysis: if using a customer email dataset

Categorization is another natural language processing technique that can accomplish similar goals. Sticking with the email dataset

Clustering is the most visually striking machine learning technique (in my humble opinion), and again the goal here is to generate quick insight. By clustering similarly themed data points in a visualization (*1, below*), the analyst is able to quickly see and exploit connections between different themes and ideas that may not have any similarity at first glance.

Que 08. Plot the cluster data using python visualizations ?

->

```
#Call required libraries
import time                    # To time processes
import warnings                # To suppress warnings

import numpy as np             # Data manipulation
import pandas as pd            # Dataframe manipulatio
import matplotlib.pyplot as plt                    # For graphics
import seaborn as sns
import plotly.plotly as py #For World Map
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)

from sklearn.preprocessing import StandardScaler  # For scaling dataset
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagati
on #For clustering
from sklearn.mixture import GaussianMixture #For GMM clustering

import os                      # For os related operations
import sys                     # For data size
```

```
wh = pd.read_csv("../input/2017.csv") #Read the dataset
wh.describe()
```

```
print("Dimension of dataset: wh.shape")
wh.dtypes
```

Basic Visualization

```
wh1 = wh[['Happiness.Score','Economy..GDP.per.Capita.','Family','Health..Life.
Expectancy.', 'Freedom',
          'Generosity','Trust..Government.Corruption.','Dystopia.Residual']] #
Subsetting the data
cor = wh1.corr() #Calculate the correlation of the above variables
sns.heatmap(cor, square = True) #Plot the correlation as heat map
```


```
#Ref: https://plot.ly/python/choropleth-maps/
data = dict(type = 'choropleth',
            locations = wh['Country'],
            locationmode = 'country names',
            z = wh['Happiness.Score'],
            text = wh['Country'],
            colorbar = {'title':'Happiness'})
layout = dict(title = 'Happiness Index 2017',
              geo = dict(showframe = False,
                         projection = {'type': 'Mercator'}))
choromap3 = go.Figure(data = [data], layout=layout)
iplot(choromap3)
```

(1) k-means clustering
(2) Agglomerative Clustering
(3) Affinity Propagation
(4) Guassian Mixture Modelling

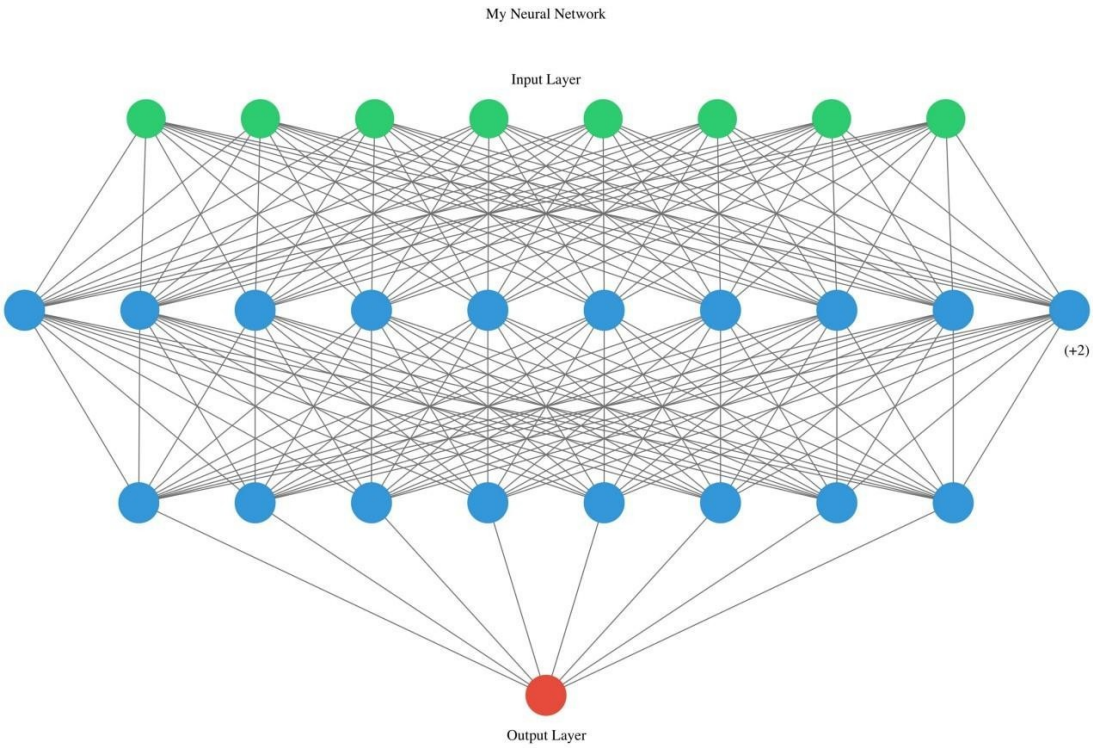
Que 09. Creating & Visualizing Neural Network for the given data. (Use python) ?

->

index.py

```
# Create your first MLP in Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Output:-



Que 10. Recognize optical character using ANN ?

->

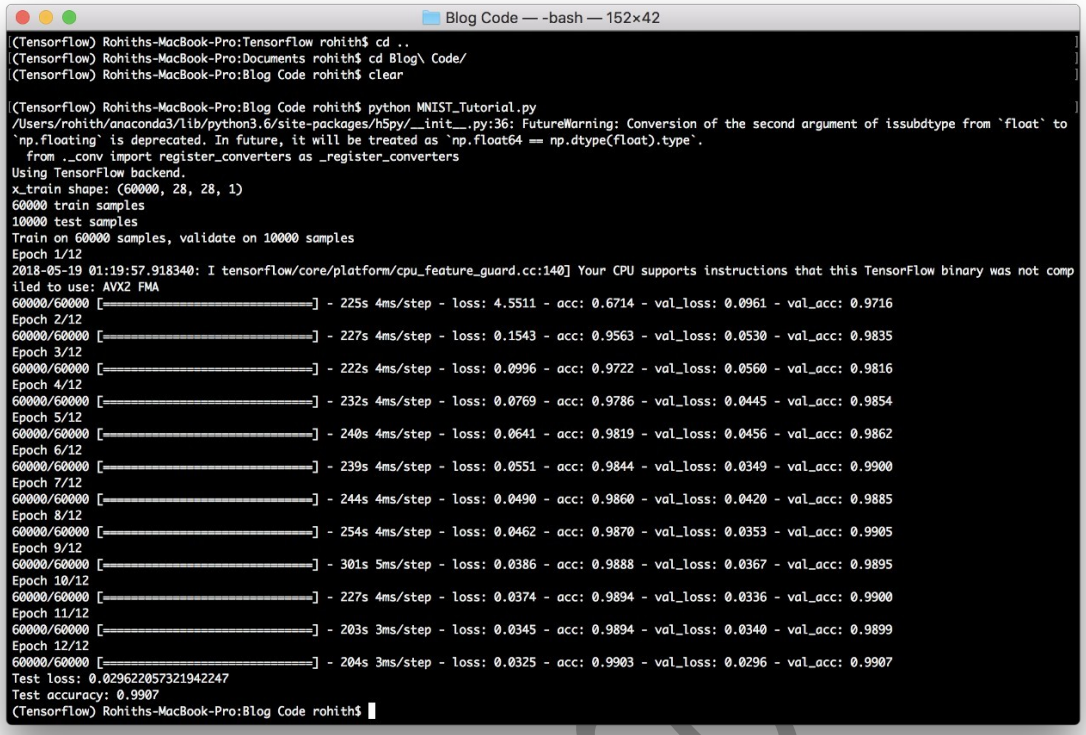
Que 11. Write a program to implement CNN ?

->

```
import
keras
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten
        from keras.layers import Conv2D, MaxPooling2D
        import numpy as np


batch_size
= 128
            num_classes = 10
            epochs = 12

            # input image dimensions
            img_rows, img_cols = 28, 28

            # the data, split between train and test sets
            (x_train, y_train), (x_test, y_test) = mnist.load_data()

            x_train = x_train.reshape(60000,28,28,1)
            x_test = x_test.reshape(10000,28,28,1)

            print('x_train shape:', x_train.shape)
            print(x_train.shape[0], 'train samples')
            print(x_test.shape[0], 'test samples')

            # convert class vectors to binary class matrices
            y_train = keras.utils.to_categorical(y_train, num_classes)
            y_test = keras.utils.to_categorical(y_test, num_classes)


model =
Sequential()
              model.add(Conv2D(32, kernel_size=(3, 3),
                               activation='relu',
                               input_shape=(28,28,1)))
              model.add(Conv2D(64, (3, 3), activation='relu'))
              model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
                model.add(Dropout(0.25))
                model.add(Flatten())
                model.add(Dense(128, activation='relu'))
                model.add(Dropout(0.5))
                model.add(Dense(num_classes, activation='softmax'))



model.compile(loss=keras.losses.categorical_crossen
tropy,
                                                    optimizer=keras.optimizers.Adadelt
                                                    a(),

                                                    metrics=['accuracy'])

                                                    model.fit(x_train, y_train,
                                                              batch_size=batch_size,
                                                              epochs=epochs,
                                                              verbose=1,
                                                              validation_data=(x_test,
                                                    y_test))
                                                    score = model.evaluate(x_test,
                                                    y_test, verbose=0)
                                                    print('Test loss:', score[0])
                                                    print('Test accuracy:', score[1])
```

Output:-



Que 12. Write a program to implement RNN ?

->

```
from
keras.models
import
Sequential
```

```
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding

model = Sequential()

# Embedding layer
model.add(
    Embedding(input_dim=num_words,
              input_length = training_length,
              output_dim=100,
              weights=[embedding_matrix],
              trainable=False,
              mask_zero=True))

# Masking layer for pre-trained embeddings
model.add(Masking(mask_value=0.0))
```

```
# Recurrent layer
model.add(LSTM(64, return_sequences=False,
               dropout=0.1, recurrent_dropout=0.1))

# Fully connected layer
model.add(Dense(64, activation='relu'))

# Dropout for regularization
model.add(Dropout(0.5))

# Output layer
model.add(Dense(num_words, activation='softmax'))

# Compile the model
model.compile(
    optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```


Output:-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 50, 100)           1175500
_________________________________________________________________
masking_3 (Masking)          (None, 50, 100)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                42240
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 11755)             764075
=================================================================
Total params: 1,985,975
Trainable params: 810,475
Non-trainable params: 1,175,500
_________________________________________________________________
```

->

Que 12. Write a program to implement GAN ?

```
# importing the necessary libraries and the MNIST dataset
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data")

# defining functions for the two networks.
# Both the networks have two hidden layers
# and an output layer which are densely or
# fully connected layers defining the
# Generator network function
def generator(z, reuse = None):
    with tf.variable_scope('gen', reuse = reuse):
        hidden1 = tf.layers.dense(inputs = z, units = 128,
                                  activation = tf.nn.leaky_relu)

        hidden2 = tf.layers.dense(inputs = hidden1,
           units = 128, activation = tf.nn.leaky_relu)

        output = tf.layers.dense(inputs = hidden2,
             units = 784, activation = tf.nn.tanh)

        return output

# defining the Discriminator network function
def discriminator(X, reuse = None):
    with tf.variable_scope('dis', reuse = reuse):
        hidden1 = tf.layers.dense(inputs = X, units = 128,
                                  activation = tf.nn.leaky_relu)

        hidden2 = tf.layers.dense(inputs = hidden1,
               units = 128, activation = tf.nn.leaky_relu)

        logits = tf.layers.dense(hidden2, units = 1)
        output = tf.sigmoid(logits)

        return output, logits

# creating placeholders for the outputs
tf.reset_default_graph()

real_images = tf.placeholder(tf.float32, shape =[None, 784])
z = tf.placeholder(tf.float32, shape =[None, 100])
```

```
G = generator(z)
D_output_real, D_logits_real = discriminator(real_images)
D_output_fake, D_logits_fake = discriminator(G, reuse = True)

# defining the loss function
def loss_func(logits_in, labels_in):
    return tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
                          logits = logits_in, labels = labels_in))

 # Smoothing for generalization
D_real_loss = loss_func(D_logits_real, tf.ones_like(D_logits_real)*0.9)
D_fake_loss = loss_func(D_logits_fake, tf.zeros_like(D_logits_real))
D_loss = D_real_loss + D_fake_loss

G_loss = loss_func(D_logits_fake, tf.ones_like(D_logits_fake))

# defining the learning rate, batch size,
# number of epochs and using the Adam optimizer
lr = 0.001 # learning rate

# Do this when multiple networks
# interact with each other

# returns all variables created(the two
# variable scopes) and makes trainable true
tvars = tf.trainable_variables()
d_vars =[var for var in tvars if 'dis' in var.name]
g_vars =[var for var in tvars if 'gen' in var.name]

D_trainer = tf.train.AdamOptimizer(lr).minimize(D_loss, var_list =
d_vars)
G_trainer = tf.train.AdamOptimizer(lr).minimize(G_loss, var_list =
g_vars)

batch_size = 100 # batch size
epochs = 500 # number of epochs. The higher the better the result
init = tf.global_variables_initializer()

# creating a session to train the networks
samples =[] # generator examples

with tf.Session() as sess:
    sess.run(init)
    for epoch in range(epochs):
        num_batches = mnist.train.num_examples//batch_size

        for i in range(num_batches):
            batch = mnist.train.next_batch(batch_size)
```

```
            batch_images = batch[0].reshape((batch_size, 784))
            batch_images = batch_images * 2-1
            batch_z = np.random.uniform(-1, 1, size =(batch_size, 100))
            _= sess.run(D_trainer, feed_dict ={real_images:batch_images,
z:batch_z})
            _= sess.run(G_trainer, feed_dict ={z:batch_z})

        print("on epoch{}".format(epoch))

        sample_z = np.random.uniform(-1, 1, size =(1, 100))
        gen_sample = sess.run(generator(z, reuse = True),
                              feed_dict ={z:sample_z})

        samples.append(gen_sample)

# result after 0th epoch
plt.imshow(samples[0].reshape(28, 28))

# result after 499th epoch
plt.imshow(samples[49].reshape(28, 28))
```

Output:-

Que 13. Web scraping experiments (by using tools) ?

->

Top 8 Web Scraping Tools

Choosing the ideal Web Scraping Tool that perfectly meets your business requirements can be a challenging task, especially when there's a large variety of Web Scraping Tools available in the market. To simplify your search, here is a comprehensive list of 8 Best Web Scraping Tools that you can choose from:

- ParseHub
- Scrapy
- OctoParse
- Scraper API

- Mozenda
- Webhose.io
- Content Grabber
- Common Crawl

1. ParseHub

Target Audience

ParseHub is an incredibly powerful and elegant tool that allows you to build web scrapers without having to write a single line of code. It is therefore as simple as simply selecting the data you need. ParseHub is targeted at pretty much anyone that wishes to play around with data. This could be anyone from analysts and data scientists to journalists.

Key Features of ParseHub

- Clean Text and HTML before downloading data.
- Simple to use graphical interface.
- ParseHub allows you to collect and store data on servers automatically.
- Automatic IP rotation.
- Scraping behind logic walls allowed.
- Provides Desktop Clients for Windows, Mac OS, Linux.
- Data is exported in JSON or Excel Format.
- Can extract data from tables and maps.

2. Scrapy

Target Audience

Scrapy is a Web Scraping library used by python developers to build scalable web crawlers. It is a complete web crawling framework that handles all the functionalities that make building web crawlers difficult such as proxy middleware, querying requests among many others.

Key Features of Scrapy

- Open Source Tool.
- Extremely well documented.
- Easily Extensible.
- Portable Python.
- Deployment is simple and reliable.

- Middleware modules are available for the integration of useful tools

3. OctoParse

Target Audience

OctoParse has a target audience similar to ParseHub, catering to people who want to scrape data without having to write a single line of code, while still having control over the full process with their highly intuitive user interface.

Key Features of OctoParse

- Site Parser and hosted solution for users who want to run scrapers in the cloud.
- Point and click screen scraper allowing you to scrape behind login forms, fill in forms, render javascript, scroll through the infinite scroll, and many more.
- Anonymous Web Data Scraping to avoid being banned.

4. Scraper API

Target Audience

Scraper API is designed for designers building web scrapers. It handles browsers, proxies, and CAPTCHAs which means that raw HTML from any website can be obtained through a simple API call.

Key Features of Scraper API

- Helps you render Javascript.
- Easy to integrate.
- Geolocated Rotating Proxies.
- Great Speed and reliability to build scalable web scrapers.
- Special pools of proxies for E-commerce price scraping, search engine scraping, social media scraping, etc.

5. Mozenda

Target Audience

Mozenda caters to enterprises looking for a cloud-based self serve Web Scraping platform. Having scraped over 7 billion pages, Mozenda boasts enterprise customers all over the world.

Key Features of Mozenda

- Offers point and click interface to create Web Scraping events in no time.
- Request blocking features and job sequencer to harvest web data in real-time.
- Best customer support and in-class account management.
- Collection and publishing of data to preferred BI tools or databases possible.
- Provide both phone and email support to all the customers.
- Highly scalable platform.
- Allows On-premise Hosting.

6. Webhose.io

Target Audience

Webhose.io is best recommended for platforms or services that are on the lookout for a completely developed web scraper and data supplier for content marketing, sharing, etc. The cost offered by the platform happens to be quite affordable for growing companies.

Key Features of Webhose.io

- Content Indexing is fairly fast.
- A dedicated support team that is highly reliable.
- Easy Integration with different solutions.
- Easy to use APIs providing full control for language and source selection.
- Simple and intuitive interface design allowing you to perform all tasks in a much simpler and practical way.
- Get structured, machine-readable data sets in JSON and XML formats.
- Allows access to historical feeds dating as far back as 10 years.
- Provides access to a massive repository of data feeds without having to bother about paying extra fees.
- An advanced feature allows you to conduct granular analysis on datasets you want to feed.

7. Content Grabber

Target Audience

Content Grabber is a cloud-based Web Scraping Tool that helps businesses of all sizes with data extraction.

Key Features of Content Grabber

- Web data extraction is faster compared to a lot of its competitors.
- Allows you to build web apps with the dedicated API allowing you to execute web data directly from your website.
- You can schedule it to scrape information from the web automatically.
- Offers a wide variety of formats for the extracted data like CSV, JSON, etc.

8. Common Crawl

Target Audience

Common Crawl was developed for anyone wishing to explore and analyze data and uncover meaningful insights from it.

Key Features of Common Crawl

- Open Datasets of raw web page data and text extractions.
- Support for non-code based usage cases.
- Provides resources for educators teaching data analysis.

END !...