1 : Implement and demonstrate the FIND-S algorithm

```
import random
import csv
```

```
attributes = [
                ['Sunny','Rainy'  ],
                ['Warm','Cold'    ],
                ['Normal','High'  ],
                ['Strong','Weak'  ],
                ['Warm','Cool'    ],
                ['Same','Change'  ]
              ]
```

```
data_list = []

with open('ws.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        data_list.append(row)
```

```
num_attributes = len(attributes)
hypothesis = ['0'] * num_attributes

print("The initial value of hypothesis:", end='\n'*3)
print(hypothesis)
```

```
The initial value of hypothesis:


['0', '0', '0', '0', '0', '0']
```

```
# Comparing with First Training Example ( Assigning )

*first_sample, output = data_list[0]
hypothesis = first_sample[:]            # Deep copy
```

```python
# Comparing with Remaining Training Examples of Given Data Set

print("Find S: Finding a Maximally Specific Hypothesis", end='\n'*3)

outer_index = 1

for *data, output in data_list:

    if output == 'Yes':
            for index, attribute in enumerate(data):
                if attribute != hypothesis[index]:
                    hypothesis[index] = '?'



    print("For Training Example No : {0} the hypothesis is {1} ".format( outer_index, hypothesis) )
    outer_index += 1
```

```
Find S: Finding a Maximally Specific Hypothesis


For Training Example No : 1 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training Example No : 2 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training Example No : 3 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training Example No : 4 the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```python
print(" The Maximally Specific Hypothesis for a given Training Examples:", end='\n'*2)
print(hypothesis)
```

```
 The Maximally Specific Hypothesis for a given Training Examples:

['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```python
# PS: Dataset for clarity
print(open('ws.csv').read())
```

```
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

```python

```

2 : Implement and demonstrate the Candidate-Elimination algorithm

```
import csv
```

```
data_list = []

with open('data.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        data_list.append(row)
```

```
*first_sample, output = data_list[0]
num_attributes = len(first_sample)

S = ['0'] * num_attributes
G = ['?'] * num_attributes

print("The Initial value of hypothesis", end='\n\n')
print ("The most specific hypothesis S0 : ", S)
print ("The most general hypothesis G0 : ", G)
```

```
The Initial value of hypothesis

The most specific hypothesis S0 :  ['0', '0', '0', '0', '0', '0']
The most general hypothesis G0 :  ['?', '?', '?', '?', '?', '?']
```

```
# Comparing with First Training Example ( Assigning )

S = first_sample[:]
```

```
# Comparing with Remaining Training Examples of Given Data Set

print("Candidate Elimination algorithm  Hypotheses Version Space Computation", end='\n\n')

general_hypothesis_space = []
outer_index = 1


for *data, output in data_list:

    if output == 'Y':
        for index, attribute in enumerate(data):
            if attribute != S[index]:
                S[index] = '?'

        for general_hypothesis in general_hypothesis_space:
            for index, attribute in enumerate(general_hypothesis):
                 if attribute not in {'?', S[index] }:
                    general_hypothesis_space.remove(general_hypothesis)
                                                    #remove it if it's not matching with the specific hypothesis


    elif output == 'N':
        for index, attribute in enumerate(data):
             if S[index] not in  {'?', attribute}:    # if not  matching with the specific Hypothesis take it seperately and store it
                G[index] = S[index]
                general_hypothesis_space.append(G)    # this is the version space to store all Hypotheses
                G = ['?'] * num_attributes            # resetting


    #-----------------------------------------printing section-----------------------------
    print()
    print("for training example no : {0}, S{0}: ".format(outer_index), S)

    if ( len(general_hypothesis_space) == 0 ):
        print("for training example no : {0}, G{0}: ".format(outer_index), G)
    else:
        print("for training example no : {0}, G{0}: ".format(outer_index), general_hypothesis_space)
    print('-' * 90)
    #------------------------------------------------------------------------------------------

    outer_index += 1
```

```
Candidate Elimination algorithm  Hypotheses Version Space Computation


for training example no : 1, S1:  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
for training example no : 1, G1:  ['?', '?', '?', '?', '?', '?']
------------------------------------------------------------------------------------------

for training example no : 2, S2:  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
for training example no : 2, G2:  ['?', '?', '?', '?', '?', '?']
------------------------------------------------------------------------------------------

for training example no : 3, S3:  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
for training example no : 3, G3:  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', 'Same']]
------------------------------------------------------------------------------------------

for training example no : 4, S4:  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
for training example no : 4, G4:  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
------------------------------------------------------------------------------------------
```

```
print("Specific hypothesis : ", S)
print()
print("General hypothesis : ", general_hypothesis_space)
```

```
Specific hypothesis :  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

General hypothesis :  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

```python
def get_version_space(S, G):
    version_space = [S]       # Intialize the version space list and append S (Specific hypothesis)

    for i in range(len(S)):
        for general_hypothesis in G:
            if general_hypothesis[i] != S[i]:                   # Iterate over Specific and General Hypotheses to build
                temp_hypothesis = list(general_hypothesis)     # version space
                temp_hypothesis[i] = S[i]
                if temp_hypothesis not in version_space:
                    version_space.append(temp_hypothesis)

    version_space.extend(G)                                    # Finally put hypotheses that exist in G

    return version_space

print("Version Space : ")
get_version_space(S, general_hypothesis_space)
```

```
Version Space :

[['Sunny', 'Warm', '?', 'Strong', '?', '?'],
 ['Sunny', 'Warm', '?', '?', '?', '?'],
 ['Sunny', '?', '?', 'Strong', '?', '?'],
 ['?', 'Warm', '?', 'Strong', '?', '?'],
 ['Sunny', '?', '?', '?', '?', '?'],
 ['?', 'Warm', '?', '?', '?', '?']]
```

```python
# PS: Dataset for clarity
print(open('data.csv').read())
```

```
Sunny,Warm,Normal,Strong,Warm,Same,Y
Sunny,Warm,High,Strong,Warm,Same,Y
Rainy,Cold,High,Strong,Warm,Change,N
Sunny,Warm,High,Strong,Cool,Change,Y
```

```
```

3 Decision tree based ID3, build and classify for appropriate dataset

Import Play Tennis Data

```
from math import log2
from collections import Counter

import pandas as pd
dataframe = pd.read_csv('PlayTennis.csv')
```

```
#piping out the dataframe
dataframe
```

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong
7	No	Sunny	Mild	High	Weak
8	Yes	Sunny	Cool	Normal	Weak
9	Yes	Rain	Mild	Normal	Weak
10	Yes	Sunny	Mild	Normal	Strong
11	Yes	Overcast	Mild	High	Strong
12	Yes	Overcast	Hot	Normal	Weak
13	No	Rain	Mild	High	Strong

Entropy of the Training Data Set

```
def entropy(subframe):

    counter = Counter(subframe)
    num_instances = len(subframe)

    entropy = 0
    for value in counter.values():
        probability = value/num_instances
        entropy = entropy   +  ( -probability * log2(probability) )   # Entropy,  -p*log2*p

    return entropy
```

Information Gain of Attributes

```
total_samples = len(dataframe.index)

# helper function in finding aggregates

def probability(samples):
        return len(samples)/total_samples
```

```
def information_gain(dataframe, split_attribute, target_attribute):

    # Split and Aggregate Data by possible vals of Attribute:

    frame = dataframe.groupby(split_attribute)
    aggregate = frame.agg(  [entropy, probability] )

    aggregate = aggregate[target_attribute]      # since interest is only on target


    new_entropy = sum( aggregate['entropy'] * aggregate['probability'] )
    old_entropy = entropy(dataframe[target_attribute])
    return old_entropy - new_entropy
```

ID3 Algorithm

```python
def id3(dataframe, target_attribute, attributes_list, default_class=None):



    counter = Counter(dataframe[target_attribute])              # Atrribute class of YES/NO (binary)


    ''' First check: Is this split of the dataset homogeneous ?   '''

    if len(counter) == 1:
        return next(iter(counter))
                                                             # counter is a dictionary, iterate the dictionary, and release the first key


        ''' Second check: Is this split of the dataset empty ?  '''

    elif dataframe.empty or (not attributes_list):
        return default_class                                 # Return None for Empty Data Set


        ''' Otherwise: '''
    else:

        # Get Default Value for next recursive call of this function :

        default_class = max(counter)

 #-----------------------------------------------------------------------------------

        # Compute the Information Gain of all the attributes:

        gains = []
        for attribute in attributes_list:
            gains.append(
                            [ information_gain(dataframe, attribute, target_attribute), attribute ]
                        )

                                                              #appending attribute, will be easy in extracting attribute with maxgain


#-----------------------------------------------------------------------------------

        # Choose Best Attribute to split on:
        best_attribute = max(gains)[1]                       # 1 because it's attribute's index

        # Create an empty tree, with best attribute as a node
        tree = { best_attribute : {} }

        attributes_list.remove(best_attribute)               # since interest is on remaining attributes

 #-----------------------------------------------------------------------------------

        # Split dataset
        #       On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which are the result of the recursive call

        for attribute_value, data_subset in dataframe.groupby(best_attribute):
            subtree = id3(data_subset, target_attribute, attributes_list, default_class)

            tree[best_attribute][attribute_value] = subtree


        return tree
```

Predicting Attributes

```python
# Get the column headers in dateframe

attributes_list = list(dataframe.columns)
attributes_list.remove('PlayTennis')                          #     since it is not an attribute

print(attribute_list)
```

```
['Outlook', 'Temperature', 'Humidity', 'Wind']
```

Tree Construction

```python
# Run Algorithm:


tree = id3(dataframe, 'PlayTennis', attributes_list.copy() )    #.copy since attributes are removed in recursions


# pretty print

from pprint import  pprint
pprint(tree)
```

```
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Classification Accuracy

```python
#classifying function

def classify(instance, tree, default=None):


    attribute = next(iter(tree))                     # Outlook/Humidity/Wind

    subtree =  tree[attribute]

    if instance[attribute] in subtree:               # Value of the attributs in  set of Tree keys
        result = subtree[ instance[attribute] ]

        if isinstance(result, dict):                 # if it is a tree(dictionary), go deeper
            return classify(instance, result)
        else:
            return result                            # this is a label
    else:
        return default
```

```python
# creating a new column "predicted" in dataframe by applying "classify" along rows

dataframe['predicted'] = dataframe.apply(classify, axis=1, args=(tree,'No') )


# Accuracy, no_of_correct_predictions / total_samples

accuracy =  sum( dataframe['PlayTennis'] == dataframe['predicted'] ) / len(dataframe.index)
print('Accuracy is: ', accuracy )


dataframe
```

```
Accuracy is:  1.0
```

	PlayTennis	Outlook	Temperature	Humidity	Wind	predicted
0	No	Sunny	Hot	High	Weak	No
1	No	Sunny	Hot	High	Strong	No
2	Yes	Overcast	Hot	High	Weak	Yes
3	Yes	Rain	Mild	High	Weak	Yes
4	Yes	Rain	Cool	Normal	Weak	Yes
5	No	Rain	Cool	Normal	Strong	No
6	Yes	Overcast	Cool	Normal	Strong	Yes
7	No	Sunny	Mild	High	Weak	No
8	Yes	Sunny	Cool	Normal	Weak	Yes
9	Yes	Rain	Mild	Normal	Weak	Yes
10	Yes	Sunny	Mild	Normal	Strong	Yes
11	Yes	Overcast	Mild	High	Strong	Yes
12	Yes	Overcast	Hot	Normal	Weak	Yes
13	No	Rain	Mild	High	Strong	No

Classification Accuracy: Training/Testing Set

```python
                                                         #.copy() to ignore a warning
training_data = dataframe.iloc[:-4].copy()                # all but last four instances and column headers
test_data  = dataframe.iloc[-4:].copy()                   # just the last four



train_tree = id3(training_data, 'PlayTennis', attributes_list.copy() )    #Note : .copy

test_data['predicted2'] = test_data.apply(                                # <---- test_data source
                                          classify,
                                          axis=1,
                                          args=(train_tree, 'Yes') )       # <---- train_data tree

accuracy =  sum( test_data['PlayTennis'] == test_data['predicted2'] ) / len(test_data.index)
print('Accuracy is over test data : ', accuracy )
```

```
Accuracy is over test data :  0.75
```

```python
#P.S : Dataset
for line in open('PlayTennis.csv').readlines():
    print("{0:>10},{1:>10},{2:>10},{3:>10}".format(*line.strip().split(',')))
```

```
PlayTennis,   Outlook,Temperature,  Humidity
        No,     Sunny,       Hot,      High
        No,     Sunny,       Hot,      High
       Yes,  Overcast,       Hot,      High
       Yes,      Rain,      Mild,      High
       Yes,      Rain,      Cool,    Normal
        No,      Rain,      Cool,    Normal
       Yes,  Overcast,      Cool,    Normal
        No,     Sunny,      Mild,      High
       Yes,     Sunny,      Cool,    Normal
       Yes,      Rain,      Mild,    Normal
       Yes,     Sunny,      Mild,    Normal
       Yes,  Overcast,      Mild,      High
       Yes,  Overcast,       Hot,    Normal
        No,      Rain,      Mild,      High
```

4 : Artificial Neural Network, Backpropagation and testing

```
import numpy as np

'''
    each sample in dataset has
'''
        # [ hours_of_sleeping, hours_of_studying,  marks_obtained_in_a_test ]


dataset = [  [2, 9, 92],
             [1, 5, 86],
             [3, 6, 89]
          ]
```

Preparing inputs and outputs

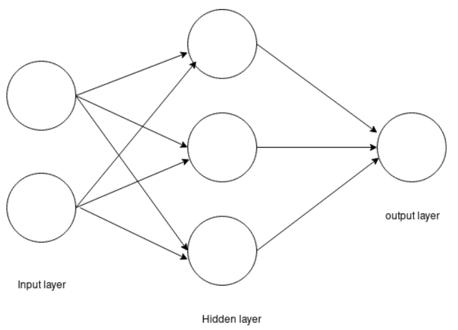
```
# Forming inputs and outputs for the Neural Network

inputs  = [ sample[:-1]      for sample in dataset ]
outputs = [ [ sample[-1] ]   for sample in dataset ]

            # Notice : extra big brackets for generating column array, highly critical if it is ignored


# converting normal arrays to numpy float arrays,

inputs = np.array(inputs, dtype=float)
outputs =  np.array(outputs, dtype=float)


# Normalizing by max divide, The inputs and outputs shoud be in range [0, 1]

inputs = inputs/np.amax(inputs, axis=0)     # input has 2 columns, dividing each column with maximum of respective column
outputs = outputs/100                       # since max test score is 100
```

Defining necessary functions

```
#activation function

def sigmoid(s):

    return 1/(1 + np.exp( -s ) )
```

```
#dervative of activation function which is necessary for backtracking

def sigmoid_derivative(s):

        return s * (1 - s)
```

static single hidden layered Neural Net class without biases



```python
class Neural_Network(object):


    def __init__(self, inputs, outputs):

                   # Initialization
        self.inputs = inputs
        self.outputs = outputs

        #-------------------------------------------------------

                   # Parameters
        self.input_size  = len(inputs[0])
        self.output_size = 1
        self.hidden_size = 3                   # number of neurons in hidden layer

        #-------------------------------------------------------

                                               # random weights for first layer and second layer'


        self.weights1 = np.random.randn(self.input_size, self.hidden_size)        # (3x2) weight matrix from input to hidden layer
        self.weights2 = np.random.randn(self.hidden_size, self.output_size)       # (3x1) weight matrix from hidden to output layer


    def forward(self, inputs):

        ''' forward propogation  '''


    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

        self.z1 = np.dot(inputs, self.weights1)        # dot product of input layer and first set of 3x2 weights
        self.z2 = sigmoid(self.z1)                     # activation function applied on previous output

    #+-------------------------------------------------------------------------------------------------------+#

        self.z3 = np.dot(self.z2, self.weights2)       # dot product of hidden layer and second set of 3x1 weights
        obtained_output = sigmoid(self.z3)             # final activation function

    #++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

        return obtained_output


    def backward(self, obtained_output):

        ''' backward propgate through the network    '''


                # error in output
        output_error = self.outputs - obtained_output

                # applying derivative of sigmoid to obtained outputs
        output_delta = output_error * sigmoid_derivative(obtained_output)

        '''--------------------------------------------------------------'''

                 # z2 error:  hidden layer weights contribribution to output error
        z2_error = output_delta.dot( self.weights2.transpose()    )

                 # applying derivative of sigmoid to z2 error
        z2_delta = z2_error * sigmoid_derivative(self.z2)


        '''--------------------------------------------------------------'''

        #----------------------------------------------------#
        #   updating weights, wj -> wj + n * delta(wj)       #
        #                                n, learning rate    #
        #----------------------------------------------------#

                  # adjusting first set (input --> hidden) weights
        self.weights1 = self.weights1 +   (0.5 * self.inputs.T.dot(z2_delta) )     #.T stands for transpose

                  # adjusting second set (hidden --> output) weights
        self.weights2 = self.weights2 +   (0.5 * self.z2.T.dot(output_delta) )


    def train(self):

        ''' training  the network'''

        obtained_output = self.forward(self.inputs)
        self.backward(obtained_output)
```

Creating a neural net and training it

```
net = Neural_Network(inputs, outputs)


for i in range(20):                   #  the more you train, the  less error you obtain untill it overfits

    loss =  np.mean( np.square(outputs - net.forward(inputs)))      # mean sum squared loss
    print ("Epoch-> ", i, " Loss:", loss)
    net.train()
```

```
Epoch->  0  Loss: 0.07274521673240818
Epoch->  1  Loss: 0.0585782542249422
Epoch->  2  Loss: 0.047897611723800394
Epoch->  3  Loss: 0.039725730089891075
Epoch->  4  Loss: 0.03337521083036012
Epoch->  5  Loss: 0.028364553631714123
Epoch->  6  Loss: 0.0243542932035502
Epoch->  7  Loss: 0.021102324867346034
Epoch->  8  Loss: 0.018433588767724624
Epoch->  9  Loss: 0.01621966047452225
Epoch->  10  Loss: 0.014364967246628403
Epoch->  11  Loss: 0.01279739659922404
Epoch->  12  Loss: 0.011461822793107186
Epoch->  13  Loss: 0.01031558692703971
Epoch->  14  Loss: 0.009325299066303581
Epoch->  15  Loss: 0.008464545890300382
Epoch->  16  Loss: 0.007712226436721825
Epoch->  17  Loss: 0.007051329051034897
Epoch->  18  Loss: 0.006468022117193933
Epoch->  19  Loss: 0.005950970628860717
```

Prediction

```
test = np.array( [ [2, 5], [1, 9], [2, 3] ], dtype=float)    # same as preparing inputs
test = test/np.amax(test, axis=0)

print("Input: ",test, sep='\n\n')                             # Normalized input
```

```
Input:

[[1.         0.55555556]
 [0.5        1.        ]
 [1.         0.33333333]]
```

```
output_of_test =  net.forward(test) * 100

print("Predicted Output", output_of_test, sep='\n\n')        # predicted marks for each sample
```

```
Predicted Output

[[86.19906654]
 [79.93228323]
 [86.8421938 ]]
```

```

```

5: Naïve Bayes classifier for a sample data and compute the accuracy

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

store the feature matrix (x) and response vector (y)

```
x = iris_dataset.data
y = iris_dataset.target
```

spliting 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)
```

training the model on training set

```
from sklearn.naive_bayes import GaussianNB
gnb  = GaussianNB()

gnb.fit(x_train,y_train)
```

making prediction on the testing set

```
y_pred = gnb.predict(x_test)
```

comparing actual response values(y_test) with prediction response values (y_pred)

```
from sklearn import metrics

percent =  round( metrics.accuracy_score(y_test,y_pred) * 100, 2)

print("Gaussian Naive Bayes model accuracy is {0}%".format(percent) )
```

```
Gaussian Naive Bayes model accuracy is 97.37%
```

```

```

6: classify documents by bayesian classifier model

load dataset

```
from sklearn.datasets import fetch_20newsgroups

docs_train = fetch_20newsgroups(subset='train')

attributes = docs_train.target_names
```

import necessary modules

```
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.feature_extraction.text import TfidfTransformer

from sklearn.naive_bayes import MultinomialNB
```

create a pipeline for workflow

```
from sklearn.pipeline import Pipeline

text_clf = Pipeline([  ('vect',  CountVectorizer()   ),
                       ('tfidf', TfidfTransformer()  ),
                       ('clf',   MultinomialNB()     ),
                    ])

text_clf.fit( docs_train.data, docs_train.target )
```

make predictions over test dataset

```
docs_test = fetch_20newsgroups( subset='test' )

predicted = text_clf.predict(docs_test.data)
```

print accuracy

```
from sklearn import metrics

percent =  round( metrics.accuracy_score( docs_test.target, predicted) * 100,  2)



print("Accuracy is {0}% ".format(percent))
```

```
Accuracy is 77.39%
```

classify a few sentences

```
statements = ['computers are incredible machines', 'microsoft and windows',  'India and christianity']

classifications = [ attributes[value] for value in text_clf.predict(statements) ]

print(classifications)
```

```
['comp.sys.mac.hardware', 'comp.os.ms-windows.misc', 'soc.religion.christian']
```

```
# just in case

# from sklearn import metrics
# print(metrics.classification_report(docs_test.target, predicted) )
```

```

```

7 : Construct a Bayesian Network to demonstrate the diagnosis of heart patients using standard Heart Disease

```
# Install if the module doesn't exist
! pip install pgmpy
```

```
Requirement already satisfied: pgmpy in /usr/local/lib/python3.7/dist-packages (0.1.9)
```

Importing Heart Disease Data Set and Customizing

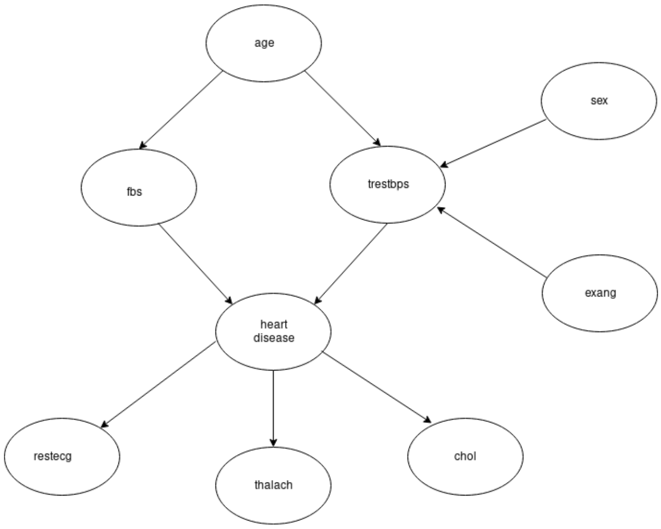
```
import pandas as pd
from urllib.request import urlopen

#data_url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.hungarian.data'
data_url = 'https://tinyurl.com/processed-hungarian-data'


#names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'heartdis
ease']

names = urlopen('https://tinyurl.com/names-csv').read().decode().split(',')   # need a live connection

data = urlopen(data_url)
heart_disease = pd.read_csv(data, names = names)                              # gets Cleveland data

heart_disease.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
0	28	1	2	130	132	0	2	185	0	0.0	?	?	?	0
1	29	1	2	120	243	0	0	160	0	0.0	?	?	?	0
2	29	1	2	140	?	0	0	170	0	0.0	?	?	?	0
3	30	0	1	170	237	0	1	170	0	0.0	?	?	6	0
4	31	0	2	100	219	0	1	150	0	0.0	?	?	?	0

Dropping columns which are more non numeric

```
heart_disease.drop(['ca', 'slope', 'thal', 'oldpeak'], axis=1)

# also replacing '?' with numpy's NaN ( not a number)

import numpy

heart_disease = heart_disease.replace('?', numpy.NaN)
```

Modeling Heart Disease Data




```
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

associations_list =  [
                       ('age', 'trestbps'),       ('age', 'fbs'),             ('sex', 'trestbps'),
                       ('exang', 'trestbps'),     ('trestbps','heartdisease'),('fbs','heartdisease'),
                       ('heartdisease','restecg'),('heartdisease','thalach'), ('heartdisease','chol')
                    ]
model = BayesianModel(associations_list)

# Learing CPDs using Maximum Likelihood Estimators
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
```

```python
print(model.get_cpds('age'))
```

```
+---------+---+
| age(28) | 0 |
+---------+---+
| age(29) | 0 |
+---------+---+
| age(30) | 0 |
+---------+---+
| age(31) | 0 |
+---------+---+
| age(32) | 0 |
+---------+---+
| age(33) | 0 |
+---------+---+
| age(34) | 0 |
+---------+---+
| age(35) | 0 |
+---------+---+
| age(36) | 0 |
+---------+---+
| age(37) | 0 |
+---------+---+
| age(38) | 0 |
+---------+---+
| age(39) | 0 |
+---------+---+
| age(40) | 0 |
+---------+---+
| age(41) | 0 |
+---------+---+
| age(42) | 0 |
+---------+---+
| age(43) | 0 |
+---------+---+
| age(44) | 0 |
+---------+---+
| age(45) | 0 |
+---------+---+
| age(46) | 0 |
+---------+---+
| age(47) | 1 |
+---------+---+
| age(48) | 0 |
+---------+---+
| age(49) | 0 |
+---------+---+
| age(50) | 0 |
+---------+---+
| age(51) | 0 |
+---------+---+
| age(52) | 0 |
+---------+---+
| age(53) | 0 |
+---------+---+
| age(54) | 0 |
+---------+---+
| age(55) | 0 |
+---------+---+
| age(56) | 0 |
+---------+---+
| age(57) | 0 |
+---------+---+
| age(58) | 0 |
+---------+---+
| age(59) | 0 |
+---------+---+
| age(60) | 0 |
+---------+---+
| age(61) | 0 |
+---------+---+
| age(62) | 0 |
+---------+---+
| age(63) | 0 |
+---------+---+
| age(65) | 0 |
+---------+---+
| age(66) | 0 |
```

```python
print(model.get_cpds('sex'))
```

```
+--------+---+
| sex(0) | 0 |
+--------+---+
| sex(1) | 1 |
+--------+---+
```

```
model.get_independencies()
```

```
(age _|_ exang, sex)
(age _|_ exang | sex)
(age _|_ exang, sex | fbs)
(age _|_ sex | exang)
(age _|_ thalach, chol, restecg | heartdisease)
(age _|_ exang | fbs, sex)
(age _|_ thalach, chol, restecg | heartdisease, sex)
(age _|_ thalach, restecg | chol, heartdisease)
(age _|_ chol, thalach | restecg, heartdisease)
(age _|_ sex | fbs, exang)
(age _|_ thalach, chol, restecg, heartdisease | trestbps, fbs)
(age _|_ thalach, chol, restecg | fbs, heartdisease)
(age _|_ chol, restecg | thalach, heartdisease)
(age _|_ thalach, chol, restecg | exang, heartdisease)
(age _|_ thalach, chol, restecg | trestbps, heartdisease)
(age _|_ thalach, restecg | chol, heartdisease, sex)
(age _|_ chol, thalach | restecg, heartdisease, sex)
(age _|_ thalach, chol, restecg, heartdisease | trestbps, fbs, sex)
(age _|_ thalach, chol, restecg | fbs, heartdisease, sex)
(age _|_ chol, restecg | thalach, heartdisease, sex)
(age _|_ thalach, chol, restecg | heartdisease, exang, sex)
(age _|_ thalach, chol, restecg | trestbps, heartdisease, sex)
(age _|_ thalach | chol, restecg, heartdisease)
(age _|_ thalach, restecg, heartdisease | fbs, chol, trestbps)
(age _|_ thalach, restecg | fbs, chol, heartdisease)
(age _|_ restecg | chol, thalach, heartdisease)
(age _|_ thalach, restecg | chol, exang, heartdisease)
(age _|_ thalach, restecg | trestbps, chol, heartdisease)
(age _|_ chol, thalach, heartdisease | trestbps, fbs, restecg)
(age _|_ chol, thalach | fbs, restecg, heartdisease)
(age _|_ chol | thalach, restecg, heartdisease)
(age _|_ chol, thalach | restecg, exang, heartdisease)
(age _|_ chol, thalach | trestbps, restecg, heartdisease)
(age _|_ chol, restecg, heartdisease | trestbps, fbs, thalach)
(age _|_ chol, restecg | fbs, thalach, heartdisease)
(age _|_ thalach, chol, restecg, heartdisease | trestbps, fbs, exang)
(age _|_ thalach, chol, restecg | fbs, exang, heartdisease)
(age _|_ thalach, chol, restecg | trestbps, fbs, heartdisease)
(age _|_ chol, restecg | thalach, exang, heartdisease)
(age _|_ chol, restecg | trestbps, thalach, heartdisease)
(age _|_ thalach, chol, restecg | trestbps, exang, heartdisease)
(age _|_ thalach | chol, restecg, heartdisease, sex)
(age _|_ restecg, thalach, heartdisease | fbs, chol, trestbps, sex)
(age _|_ restecg, thalach | fbs, chol, heartdisease, sex)
(age _|_ restecg | chol, thalach, heartdisease, sex)
(age _|_ restecg, thalach | chol, heartdisease, exang, sex)
(age _|_ restecg, thalach | trestbps, chol, heartdisease, sex)
(age _|_ chol, thalach, heartdisease | trestbps, fbs, restecg, sex)
(age _|_ chol, thalach | fbs, restecg, heartdisease, sex)
(age _|_ chol | thalach, restecg, heartdisease, sex)
(age _|_ chol, thalach | heartdisease, restecg, exang, sex)
(age _|_ chol, thalach | trestbps, restecg, heartdisease, sex)
(age _|_ chol, restecg, heartdisease | trestbps, fbs, thalach, sex)
(age _|_ chol, restecg | fbs, thalach, heartdisease, sex)
(age _|_ restecg, chol, thalach, heartdisease | trestbps, fbs, exang, sex)
(age _|_ restecg, chol, thalach | fbs, heartdisease, exang, sex)
(age _|_ restecg, chol, thalach | trestbps, fbs, heartdisease, sex)
(age _|_ chol, restecg | heartdisease, thalach, exang, sex)
(age _|_ chol, restecg | trestbps, thalach, heartdisease, sex)
(age _|_ restecg, chol, thalach | trestbps, heartdisease, exang, sex)
(age _|_ thalach, heartdisease | fbs, chol, restecg, trestbps)
(age _|_ thalach | fbs, chol, restecg, heartdisease)
(age _|_ thalach | chol, restecg, exang, heartdisease)
(age _|_ thalach | trestbps, chol, restecg, heartdisease)
(age _|_ restecg, heartdisease | fbs, chol, thalach, trestbps)
(age _|_ restecg | fbs, chol, thalach, heartdisease)
(age _|_ restecg, thalach, heartdisease | fbs, chol, trestbps, exang)
(age _|_ restecg, thalach | fbs, chol, exang, heartdisease)
(age _|_ restecg, thalach | fbs, chol, trestbps, heartdisease)
(age _|_ restecg | chol, thalach, exang, heartdisease)
(age _|_ restecg | trestbps, chol, thalach, heartdisease)
(age _|_ restecg, thalach | trestbps, chol, exang, heartdisease)
(age _|_ chol, heartdisease | thalach, trestbps, fbs, restecg)
(age _|_ chol | thalach, fbs, restecg, heartdisease)
(age _|_ chol, thalach, heartdisease | trestbps, fbs, restecg, exang)
(age _|_ chol, thalach | fbs, restecg, exang, heartdisease)
```

Inferencing with Bayesian Network

```python
# Doing exact inference using Variable Elimination
from pgmpy.inference import VariableElimination
heart_disease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
query = heart_disease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(query)
```

```
Finding Elimination Order: : 100%|██████████| 7/7 [00:00<00:00, 1932.99it/s]
Eliminating: restecg: 100%|██████████| 7/7 [00:00<00:00, 219.51it/s]
```

```
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.4919 |
+-----------------+---------------------+
| heartdisease(1) |              0.5081 |
+-----------------+---------------------+
```

8 : EM, K-Means algorithm to cluster a set of data and comparison

K Means

load a sample dataset

```
from sklearn import datasets
iris_dataset = datasets.load_iris()


iris_dataset.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
import pandas

inputs = pandas.DataFrame(iris_dataset.data)

# Assigning column names to inputs
inputs.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width' ]

outputs = pandas.DataFrame(iris_dataset.target)

# Assigning column names to outputs
outputs.columns = ['targets' ]
```

fit the model on inputs

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)    # 3 centroids


model.fit(inputs)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

plotter helper function

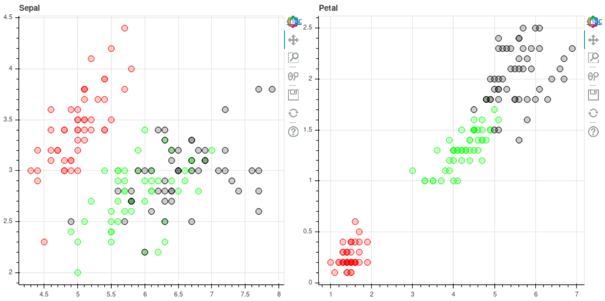
```
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import row
output_notebook()


def custom_plotter( first_list, second_list, labels,  colormap_outputs):

    colormap =   ['red', 'lime', 'black']

    first_label, second_label = labels
    first_output, second_output = colormap_outputs



    colors = [ colormap[x] for x in  first_output]

    figure1 = figure(width=500, plot_height=500, title = first_label)
    figure1.circle( *first_list, color=colors, fill_alpha=0.2, size=10)
    plot = figure1

    empty = None
    if  second_list is not empty:

        colors = [ colormap[x] for x in  second_output]

        figure2 = figure(width=500, plot_height=500, title = second_label)
        figure2.circle( *second_list, color=colors, fill_alpha=0.2, size=10)

        plot = row(figure1, figure2)

    show(plot)
```

BokehJS 1.4.0 successfully loaded.

```
'''                          old
def custom_plotter( first_list, second_list, labels,  colormap_outputs):

    # Note: List unpacking is done at scatter method

    first_label, second_label = labels
    first_output, second_output = colormap_outputs

    import matplotlib.pyplot as plt

    plt.figure(figsize=(14,7))    # better if figure is resized


    # Create a colormap  of 3 colors for clusters ( centroids )
    # If you do not pass colormap, plot will be in uni color
    import numpy
    colormap = numpy.array(['red', 'lime', 'black'])



    # ploting first_list

    plt.subplot(1, 2, 1)     # creates a portion in a figure   # Note : 1,2 is common
    plt.scatter( *first_list, c = colormap[first_output] )
    plt.title( first_label)
                             # give a title to the figure

    empty = None
    if  second_list is not empty:    # for plotting single figure ( essential in later section )


        # similarly ploting second list
        plt.subplot(1, 2, 2)
        plt.scatter( *second_list,  c = colormap[second_output] )
        plt.title( second_label )
'''
```

Plotting based on the outputs from the dataset

```
figure1_inputs = [ inputs.sepal_length, inputs.sepal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]
labels = ['Sepal', 'Petal']
color_mapper = [outputs.targets, outputs.targets]



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```

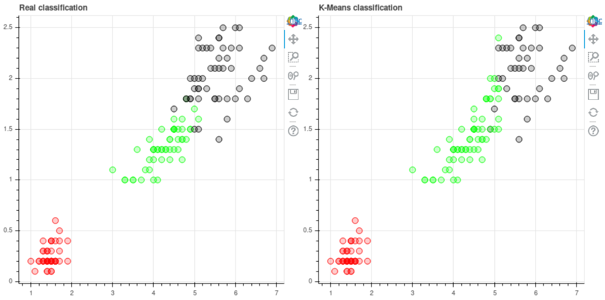


assigning predictions of a model to a equivalent variable

```
predictions = model.labels_
```

Plotting based on the predictions from the model

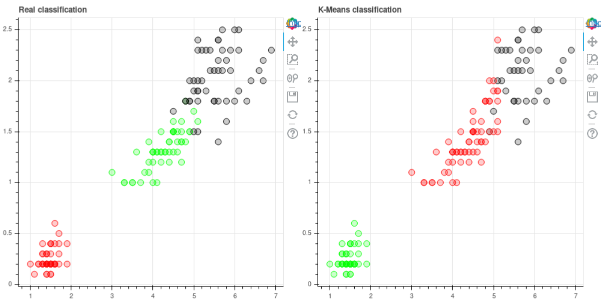
```
figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]  #same inputs
labels  = ['Real classification', 'K-Means classification']
color_mapper = [outputs.targets, predictions]                 # predictions



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```



flip predictions (Interchanging 1 and 0) and re plotting (This can be ignored)

```python
import numpy
flipped_predictions = numpy.choose(predictions, [1, 0, 2]).astype(numpy.int64)

print('predictions \n', predictions, end='\n'*3)
print('flipped predictions \n', flipped_predictions, end='\n'*3)



figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = [ inputs.petal_length, inputs.petal_width ]
labels  = ['Real classification', 'K-Means classification']
color_mapper = [outputs.targets, flipped_predictions]         # flipped predictions


# color will  be inverted
custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```

```
predictions 
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]


flipped predictions 
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



Accuracy measure of K-means model

```python
from sklearn import  metrics

percent = metrics.accuracy_score(outputs, predictions) * 100

print('Accuracy of K Means : {0:.2f}%'.format(percent))
```

```
Accuracy of K Means : 89.33%
```

GMM Gausian Mixture Model

prepare a dataframe to fit

```python
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(inputs)

# pandas is already imported without aliasing, hence not pd

scaled_data = scaler.transform(inputs)
dataframe = pandas.DataFrame(scaled_data, columns = inputs.columns)


# Inspecting dataframe
dataframe.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width
29	-1.385353	0.328414	-1.226552	-1.315444
70	0.068662	0.328414	0.592246	0.790671
142	-0.052506	-0.822570	0.762758	0.922303
76	1.159173	-0.592373	0.592246	0.264142
16	-0.537178	1.939791	-1.397064	-1.052180

fit dataframe to gmm (gausian_mixture_model)

```python
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture( n_components=3 )      # Note : parameter is not cluster
gmm.fit(dataframe)
```

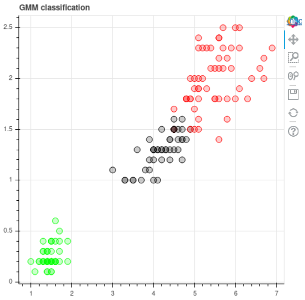
```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
        means_init=None, n_components=3, n_init=1, precisions_init=None,
        random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
        verbose_interval=10, warm_start=False, weights_init=None)
```

obtaining predections from gausian_mixture_model

```python
gmm_predictions = gmm.predict(dataframe)
```

plotting based on predictions from gausian_mixture_model

```
figure1_inputs = [ inputs.petal_length, inputs.petal_width ]
figure2_inputs = None
labels  = ['GMM classification', None]
color_mapper = [gmm_predictions, None]         # gmm_predictions



custom_plotter( figure1_inputs, figure2_inputs, labels, color_mapper)
```



Accuracy measure of gausian mixture model

```
percent = metrics.accuracy_score(outputs, gmm_predictions) * 100

# No idea why it is 30
print('Accuracy of Gausian Mixture model: {0:.2f}%'.format(percent))
```

Accuracy of Gausian Mixture model: 0.00%

9 : K-Nearest Neighbour and classify iris dataset

load iris dataset

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

targets = iris_dataset.target_names
print(targets)
```

```
['setosa' 'versicolor' 'virginica']
```

split the dataset for testing and training

```
from sklearn.model_selection import train_test_split

data_store = train_test_split( iris_dataset.data, iris_dataset.target)

training_inputs, testing_inputs, training_outputs, testing_outputs = data_store
```

fit training data to K-nearest_neighbour model

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(training_inputs, training_outputs)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=1, p=2,
           weights='uniform')
```

find predictions on test data

```
for test_input, test_output in zip(testing_inputs, testing_outputs):

    prediction_index = kn.predict([test_input])                            #test_input is 1d, enclose it for passing as 2d, ! Important

    predicted_output = targets[prediction_index][0]   # 0 since its a list having single element
    actual_output =  targets[test_output]              #test output is an index

    # {0:>10} stands for {index_of_element_in_format_parameters : right_justified by 10 characters}

    formatted_string =  '{0:>10} --> {1:>10} | {2:>10} --> {3:>10}'.format('actual output', actual_output, 'predcited output', predicted_output)
    print(formatted_string)
```

```
actual output -->     setosa | predcited output -->     setosa
actual output -->     setosa | predcited output -->     setosa
actual output -->  virginica | predcited output -->  virginica
actual output -->     setosa | predcited output -->     setosa
actual output -->     setosa | predcited output -->     setosa
actual output -->     setosa | predcited output -->     setosa
actual output -->  virginica | predcited output -->  virginica
actual output --> versicolor | predcited output --> versicolor
actual output -->     setosa | predcited output -->     setosa
actual output --> versicolor | predcited output --> versicolor
actual output -->     setosa | predcited output -->     setosa
actual output --> versicolor | predcited output --> versicolor
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output --> versicolor | predcited output --> versicolor
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output -->     setosa | predcited output -->     setosa
actual output --> versicolor | predcited output --> versicolor
actual output -->  virginica | predcited output -->  virginica
actual output -->  virginica | predcited output -->  virginica
actual output -->     setosa | predcited output -->     setosa
actual output -->  virginica | predcited output -->  virginica
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output --> versicolor | predcited output --> versicolor
actual output -->  virginica | predcited output -->  virginica
actual output -->     setosa | predcited output -->     setosa
actual output -->  virginica | predcited output -->  virginica
actual output -->     setosa | predcited output -->     setosa
```

10: Locally Weighted Regression algorithm in order to fit data points

Algorithm of "Locally Weighted Regression"

1. Read the Given data sample to X and the curve to Y

2. Set the value for smoothening parameter say τ (tau)

1. Set the bias point of interest set X_0 which is a subset of X

1. Determine the weight matrix using :

$$w(x, x_0) = e^{\dfrac{(x - x_0)^2}{-2\tau^2}}$$

1. Determine the value of model term parameter $\hat{\beta}$ using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W Y$$

2. Prediction = $X_0 * \hat{\beta}$

a) Read the given data sample to X and the curve to Y

```
import numpy as np
from math import pi


number_of_datapoints = 1000


# our data samples are  f(x) = sin(x),  x ∈ [0, 2π ]

X = np.linspace(0, 2 * pi, number_of_datapoints)                           # generating a evenly space thousand datapoints in range 0 to 2π


# output  = sin(x) + noise, because output will be a narrow curve without noise

Y = np.sin(X)   +   0.1 *  np.random.randn( number_of_datapoints )         # generating the output
```

b) Set the value for smoothening parameter say τ

```
tau = 10
```

c) Set the bias point of interest set X_0 which is a subset of X

```
X0 = X[200]  # any point you like
```

d) Determine the weight matrix using :

$$w(x, x_0) = e^{\dfrac{(x - x_0)^2}{-2\tau^2}}$$

```python
def get_weights(X0, tau):


    squared_difference = (X - X0) ** 2

    denominator =  -2 * (tau ** 2)

    W = np.exp( squared_difference / denominator)

    return W
```

e) Determine the value of model term parameter $\hat{\beta}$ using :

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W Y$$

```python
def calc_beta(W, inputs, outputs):

    X = inputs.copy()    # since we are modifying X else it is dangerous
    Y = outputs

   #-------------------------------------------

    # variable set up, for matrix multiplication
    X = np.c_[np.ones(len(X)), X]                      # essential

   # +-----------------------------------------

    XTW = X.T * W                                      # X_Transpose * W

    XTWX_inverse = np.linalg.pinv( XTW  @ X)

    beta =  XTWX_inverse @ XTW @ Y                    # as per the equation

    return beta
```

f) Prediction = $X_0 * \hat{\beta}$

```python
def predict(beta, X0):


    # variable set up, for matrix multiplication
    X0 = np.r_[1, X0]

    return beta @ X0
```

Putting things together

```python
def local_weighted_regression(X0, tau):



    W = get_weights(X0,tau)



    global X,Y         # accessing X and Y which are generated in step 1

    beta = calc_beta(W, X, Y)

    prediction = predict(beta, X0)

    return prediction
```

Create a domain, and a helper function for plotting

```python
domain =  np.linspace(0, 2*pi, num=300)  # same as X but only few points


def plotter(tau):

    # get all predictions
    predictions = [ local_weighted_regression(X0, tau) for X0 in domain]

    plot = figure(width=400, height=400,title = f'tau={tau}')    #f-string title (python 3.5+)

    plot.scatter(X, Y, alpha=.3)                                 #plot datapoints

    plot.line(domain, predictions, line_width=2, color='red')    #plot the regression line

    return plot
```

Plot for different values of tau

```python
# essential imports  and setup

from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
output_notebook()


first_row_plots = [plotter(10), plotter(1)]

second_row_plots = [plotter(0.1), plotter(0.01)]

grid = gridplot([ first_row_plots, second_row_plots])
show(grid)
```

BokehJS 1.4.0 successfully loaded.