Machine Learning Laboratory Record Book

1 Find S Algorithm

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

[1]:
```
import pandas as pd
df = pd.read_csv('Datasets/EnjoySport.csv')
df = df.drop(['slno'], axis=1)
column_length = df.shape[1]
df.head()
```

[1]:

	sky	airTemp	humidity	wind	water	forecast	enjoySport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

[2]:
```
h = ['0'] * (column_length - 1)
hp = []
hn = []
```

[3]:
```
for training_example in df.values:
    if training_example[-1] != 'no':
        hp.append(list(training_example))
    else:
        hn.append(list(training_example))
```

[4]:
```
for i in range(len(hp)):
    for j in range(column_length - 1):
        if (h[j] == '0'):
            h[j] = hp[i][j]
        if (h[j] != hp[i][j]):
            h[j] = '?'
        else:
            h[j] = hp[i][j]
```

[5]:
```
print(f'Positive Hypotheses:\n{hp}')
print(f'Negative Hypotheses:\n{hn}')
print(f'Maximally Specific Hypothesis:\n{h}')
```

```
Positive Hypotheses:
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm',
```

```
'high', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong',
'cool', 'change', 'yes']]
Negative Hypotheses:
[['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']]
Maximally Specific Hypothesis:
['sunny', 'warm', '?', 'strong', '?', '?']
```

Dataset

EnjoySport.csv

sky	airTemp	humidity	wind	water	forecast	enjoySport
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

2 Candidate Elimination Algorithm

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

[1]:
```
import pandas as pd
df = pd.read_csv('Datasets/EnjoySport.csv')
df = df.drop(['slno'], axis=1)
concepts = df.values[:,:-1]
target = df.values[:,-1]
df.head()
```

[1]:

	sky	airTemp	humidity	wind	water	forecast	enjoySport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

[2]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in␣
 ↪range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    indices = [i for i, val in enumerate(general_h) if val == ['?','?','?','?
 ↪','?','?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h, general_h
```

[3]:
```
s_final, g_final = learn(concepts, target)
print(f"Final S: {s_final}")
print(f"Final G: {g_final}")
```

```
Final S: ['sunny' 'warm' '?' 'strong' '?' '?']
Final G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Dataset

EnjoySport.csv

sky	airTemp	humidity	wind	water	forecast	enjoySport
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

3 Decision Tree based on ID3

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

[1]:
```
def infoGain(P, N):
    import math
    return -P / (P + N) * math.log2(P / (P + N)) - N / (P + N) * math.log2(N /␣
 ↪(P + N))
```

[2]:
```
def insertNode(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:
        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node: 'None'}
    return tree
```

[3]:
```
def insertConcept(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:
        tree[addTo] = Node
    return tree
```

[4]:
```
def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:
        return tree

    countC = {}
    for cVal in conceptVals:
        dataCC = data[data[concept] == cVal]
        countC[cVal] = dataCC.shape[0]

    if countC[conceptVals[0]] == 0:
        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree
    if countC[conceptVals[1]] == 0:
        tree = insertConcept(tree, addTo, conceptVals[0])
        return tree

    ClassEntropy = infoGain(countC[conceptVals[0]], countC[conceptVals[1]])
```

```
    Attr = {}
    for a in AttributeList:
        Attr[a] = list(set(data[a]))

    AttrCount = {}
    EntropyAttr = {}
    for att in Attr:
        for vals in Attr[att]:
            for c in conceptVals:
                iData = data[data[att] == vals]
                dataAtt = iData[iData[concept] == c]
                AttrCount[c] = dataAtt.shape[0]
            TotalInfo = AttrCount[conceptVals[0]] + AttrCount[conceptVals[1]]
            if AttrCount[conceptVals[0]] == 0 or AttrCount[conceptVals[1]] == 0:
                InfoGain = 0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[0]],␣
↪AttrCount[conceptVals[1]])

            if att not in EntropyAttr:
                EntropyAttr[att] = (TotalInfo / Total) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + (TotalInfo / Total) *␣
↪InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]

    Node = max(Gain, key=Gain.get)

    tree = insertNode(tree, addTo, Node)
    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)
        newData = data[data[Node] == nD].drop(Node, axis=1)
        AttributeList = list(newData)[:-1]
        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree,␣
↪nD)
    return tree
```

[5]:
```
import pandas as pd
def main():
    data = pd.read_csv('Datasets/PlayTennis.csv')
    AttributeList = list(data)[:-1]
    concept = str(list(data)[-1])
    conceptVals = list(set(data[concept]))
```

```
    tree = getNextNode(data, AttributeList, concept, conceptVals, {'root':␣
↪'None'}, 'root')
    return tree
```

[6]:
```
tree = main()['root']
```

[7]:
```
df = pd.read_csv('Datasets/PlayTennis.csv')
def test(tree, d):
    for k in tree:
        for v in tree[k]:
            if (d[k] == v and isinstance(tree[k][v], dict)):
                test(tree[k][v], d)
            elif (d[k] == v):
                print("Classification: " + tree[k][v])
```

[8]:
```
print(tree)
```

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

[9]:
```
test(tree, df.loc[0, :])
```

```
Classification: No
```

Dataset

PlaySport.csv

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

4 Artificial Neural Network with Backpropogation Algorithm

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

[1]:
```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X / np.amax(X, axis=0)
y = y / 100
```

[2]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)
```

[3]:
```
epoch = 1000
learning_rate = 0.6
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
```

[4]:
```
wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh = np.random.uniform(size=(1, hiddenlayer_neurons))
wo = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))
bo = np.random.uniform(size=(1, output_neurons))
```

[5]:
```
for i in range(epoch):
    #Forward Propogation
    net_h = np.dot(X, wh) + bh
    sigma_h = sigmoid(net_h)
    net_o = np.dot(sigma_h, wo) + bo
    output = sigmoid(net_o)
    #Backpropagation
    deltaK = (y - output) * derivatives_sigmoid(output)
    deltaH = deltaK.dot(wo.T) * derivatives_sigmoid(sigma_h)
    wo = wo + sigma_h.T.dot(deltaK) * learning_rate
    wh = wh + X.T.dot(deltaH) * learning_rate
```

[6]:
```
print(f"Input: \n {X}")
print(f"Actual Output: \n{y}")
print(f"Predicted Output: \n{output}")
```

```
Input:
 [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
```

```
 [0.89]]
Predicted Output:
[[0.89451639]
 [0.87924211]
 [0.89579247]]
```

5 Naive Bayes Classifier from Scratch

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

[1]:
```
def probAttr(data, attr, val):
    Total = data.shape[0]
    cnt = len(data[data[attr] == val])
    return cnt, cnt / Total
```

[2]:
```
def train(data, Attr, conceptVals, concept):
    conceptProbs = {}
    countConcept = {}
    for cVal in conceptVals:
        countConcept[cVal], conceptProbs[cVal] = probAttr(data, concept, cVal)

    AttrConcept = {}
    probability_list = {}
    for att in Attr:
        probability_list[att] = {}
        AttrConcept[att] = {}
        for val in Attr[att]:
            AttrConcept[att][val] = {}
            a, probability_list[att][val] = probAttr(data, att, val)
            for cVal in conceptVals:
                dataTemp = data[data[att] == val]
                AttrConcept[att][val][cVal] = len(dataTemp[dataTemp[concept] ==␣
 ↪cVal]) / countConcept[cVal]

    print(f"P(A) : {conceptProbs}\n")
    print(f"P(X/A) : {AttrConcept}\n")
    print(f"P(X) : {probability_list}\n")
    return conceptProbs, AttrConcept, probability_list
```

[3]:
```
def test(examples, Attr, concept_list, conceptProbs, AttrConcept,␣
 ↪probability_list):
    misclassification_count = 0
    Total = len(examples)
    for ex in examples:
        px = {}
        for a in Attr:
            for x in ex:
                for c in concept_list:
                    if x in AttrConcept[a]:
                        if c not in px:
                            px[c] = conceptProbs[c] * AttrConcept[a][x][c] /␣
 ↪probability_list[a][x]
                        else:
```

```
                            px[c] = px[c] * AttrConcept[a][x][c] /␣
 ↪probability_list[a][x]
        print(px)
        classification = max(px, key=px.get)
        print(f"Classification : {classification} Expected : {ex[-1]}")
        if (classification != ex[-1]):
            misclassification_count += 1
    misclassification_rate = misclassification_count * 100 / Total
    accuracy = 100 - misclassification_rate
    print(f"Misclassification Count={misclassification_count}")
    print(f"Misclassification Rate={misclassification_rate}%")
    print(f"Accuracy={accuracy}%")
```

[4]:
```
import pandas as pd
df = pd.read_csv('Datasets/PlayTennis.csv')
concept = str(list(df)[-1])
concept_list = set(df[concept])
Attr = {}

for a in df.columns[:-1]:
    Attr[a] = set(df[a])
    print(f"{a}: {Attr[a]}")

conceptProbs, AttrConcept, probability_list = train(df, Attr, concept_list,␣
 ↪concept)

examples = pd.read_csv('Datasets/PlayTennis.csv')
test(examples.values, Attr, concept_list, conceptProbs, AttrConcept,␣
 ↪probability_list)
```

```
Outlook: {'Rain', 'Overcast', 'Sunny'}
Temperature: {'Cool', 'Mild', 'Hot'}
Humidity: {'Normal', 'High'}
Wind: {'Weak', 'Strong'}
P(A) : {'Yes': 0.6428571428571429, 'No': 0.35714285714285715}

P(X/A) : {'Outlook': {'Rain': {'Yes': 0.3333333333333333, 'No': 0.4},
'Overcast': {'Yes': 0.4444444444444444, 'No': 0.0}, 'Sunny': {'Yes':
0.2222222222222222, 'No': 0.6}}, 'Temperature': {'Cool': {'Yes':
0.3333333333333333, 'No': 0.2}, 'Mild': {'Yes': 0.4444444444444444, 'No': 0.4},
'Hot': {'Yes': 0.2222222222222222, 'No': 0.4}}, 'Humidity': {'Normal': {'Yes':
0.6666666666666666, 'No': 0.2}, 'High': {'Yes': 0.3333333333333333, 'No': 0.8}},
'Wind': {'Weak': {'Yes': 0.6666666666666666, 'No': 0.4}, 'Strong': {'Yes':
0.3333333333333333, 'No': 0.6}}}

P(X) : {'Outlook': {'Rain': 0.35714285714285715, 'Overcast': 0.2857142857142857,
'Sunny': 0.35714285714285715}, 'Temperature': {'Cool': 0.2857142857142857,
'Mild': 0.42857142857142855, 'Hot': 0.2857142857142857}, 'Humidity': {'Normal':
```

```
0.5, 'High': 0.5}, 'Wind': {'Weak': 0.5714285714285714, 'Strong':
0.42857142857142855}}

{'Yes': 0.2419753086419753, 'No': 0.9408000000000002}
Classification : No Expected : No
{'Yes': 0.16131687242798354, 'No': 1.8816000000000002}
Classification : No Expected : No
{'Yes': 0.6049382716049383, 'No': 0.0}
Classification : Yes Expected : Yes
{'Yes': 0.4839506172839506, 'No': 0.4181333333333335}
Classification : Yes Expected : Yes
{'Yes': 1.0888888888888888, 'No': 0.07840000000000004}
Classification : Yes Expected : Yes
{'Yes': 0.7259259259259259, 'No': 0.15680000000000005}
Classification : Yes Expected : No
{'Yes': 1.2098765432098766, 'No': 0.0}
Classification : Yes Expected : Yes
{'Yes': 0.3226337448559671, 'No': 0.6272000000000001}
Classification : No Expected : No
{'Yes': 0.7259259259259256, 'No': 0.11760000000000002}
Classification : Yes Expected : Yes
{'Yes': 0.9679012345679012, 'No': 0.10453333333333338}
Classification : Yes Expected : Yes
{'Yes': 0.43017832647462273, 'No': 0.31360000000000005}
Classification : Yes Expected : Yes
{'Yes': 0.5377229080932785, 'No': 0.0}
Classification : Yes Expected : Yes
{'Yes': 1.2098765432098766, 'No': 0.0}
Classification : Yes Expected : Yes
{'Yes': 0.3226337448559671, 'No': 0.8362666666666669}
Classification : No Expected : No
Misclassification Count=1
Misclassification Rate=7.142857142857143%
Accuracy=92.85714285714286%
```

Dataset

PlaySport.csv

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

6 Naive Bayes Classifier using an API

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

[1]:
```
import pandas as pd

msg = pd.read_csv('Datasets/document.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
```

```
Total Instances of Dataset:  18
```

[2]:
```
X = msg.message
y = msg.labelnum
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)
count_v = CountVectorizer()
X_train_dm = count_v.fit_transform(X_train)
X_test_dm = count_v.transform(X_test)
```

[3]:
```
df = pd.DataFrame(X_train_dm.toarray(), columns=count_v.get_feature_names())
```

[4]:
```
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB()
clf.fit(X_train_dm, y_train)
pred = clf.predict(X_test_dm)
for doc, p in zip(X_train, pred):
    p = 'pos' if p == 1 else 'neg'
    print(f"{doc} -> {p}")
```

```
He is my sworn enemy -> pos
I do not like the taste of this juice -> neg
I am sick and tired of this place -> pos
This is an awesome place -> neg
My boss is horrible -> neg
```

[5]:
```
from sklearn.metrics import (accuracy_score, confusion_matrix, precision_score,␣
 ↪recall_score)

print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(y_test, pred))
print('Recall: ', recall_score(y_test, pred))
print('Precision: ', precision_score(y_test, pred))
print('Confusion Matrix: \n', confusion_matrix(y_test, pred))
```

```
Accuracy Metrics:

Accuracy:  0.8
Recall:  1.0
Precision:  0.5
Confusion Matrix:
 [[3 1]
 [0 1]]
```

Dataset

document.csv

sentence	target
I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

7 Bayesian Network using an API

Write a program to construct aBayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

[1]:
```
import sys
from urllib.request import urlopen

import numpy as np
import pandas as pd
```

[2]:
```
heartDisease_df = pd.read_csv('Datasets/heartDisease.csv')
```

[3]:
```
heartDisease_df.drop(['ca', 'slope', 'thal', 'oldpeak'], axis=1, inplace=True)
```

[4]:
```
heartDisease_df.replace('?', np.nan, inplace=True)
```

[6]:
```
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel

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

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

[7]:
```
print(model.get_cpds('age'))
print(model.get_cpds('chol'))
print(model.get_cpds('sex'))
model.get_independencies()
```

```
+---------+------------+
| age(28) | 0.00383142 |
+---------+------------+
| age(29) | 0.00383142 |
+---------+------------+
| age(30) | 0.00383142 |
+---------+------------+
| age(31) | 0.00766284 |
+---------+------------+
.
.
.

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

[8]:
```
from pgmpy.inference import VariableElimination

HeartDisease_infer = VariableElimination(model)
```

[10]:
```
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
```

```
+---------------+--------------------+
| heartdisease  |  phi(heartdisease) |
+===============+====================+
| heartdisease_0 |             0.6333 |
+---------------+--------------------+
| heartdisease_1 |             0.3667 |
+---------------+--------------------+
```

[11]:
```
q = HeartDisease_infer.query(variables=['heartdisease'],evidence={'chol': 100})
print(q['heartdisease'])
```

```
+---------------+--------------------+
| heartdisease  |  phi(heartdisease) |
+===============+====================+
| heartdisease_0 |             1.0000 |
+---------------+--------------------+
| heartdisease_1 |             0.0000 |
+---------------+--------------------+
```

Dataset

8 K-Means and E-M Clustering Algorithms using API

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.mixture import GaussianMixture
```

[2]:
```
iris = load_iris()
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df['target'] = iris['target']
```

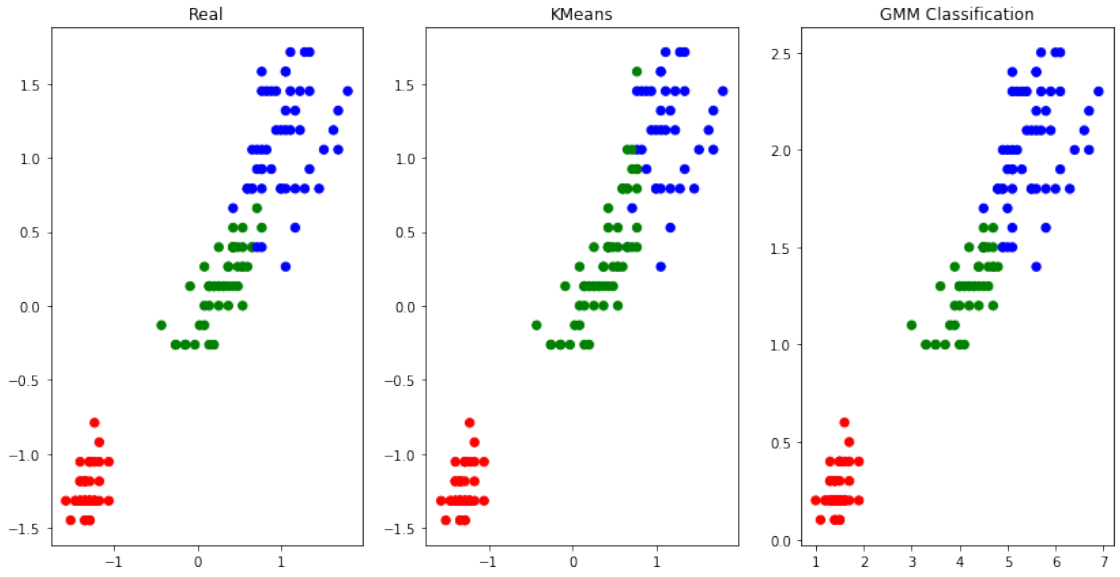
[3]:
```
X = df.iloc[:, :-1]
y = df['target']
```

[4]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)
X_scaled_array = scaler.transform(X)
X_scaled = pd.DataFrame(X_scaled_array, columns = X.columns)
```

[8]:
```
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'green', 'blue'])
#REAL PLOT
plt.subplot(1, 3, 1)
plt.scatter(X_scaled['petal length (cm)'], X_scaled['petal width (cm)'],␣
 ↪c=colormap[y], s=40)
plt.title('Real')
#K-PLOT
plt.subplot(1, 3, 2)
model = KMeans(n_clusters=3, random_state=0)
pred_y = model.fit_predict(X)
pred_y = np.choose(pred_y, [1, 0, 2]).astype(np.int64)
plt.scatter(X_scaled['petal length (cm)'], X_scaled['petal width (cm)'],␣
 ↪c=colormap[pred_y], s=40)
plt.title('KMeans')
#GMM PLOT
gmm = GaussianMixture(n_components=3, max_iter=200)
y_cluster_gmm = gmm.fit_predict(X_scaled)
y_cluster_gmm = np.choose(y_cluster_gmm, [2, 0, 1]).astype(np.int64)
plt.subplot(1, 3, 3)
```

```
plt.scatter(X['petal length (cm)'], X['petal width (cm)'],␣
 ↪c=colormap[y_cluster_gmm], s=40)
plt.title('GMM Classification')
```

[8]: `Text(0.5, 1.0, 'GMM Classification')`



9 KNN Algorithm using API

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

[1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
```

[2]:
```
dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split(dataset["data"],␣
 ↪dataset["target"], random_state=0)
```

[3]:
```
kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(X_train, y_train)
```

[3]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

[4]:
```
prediction = kn.predict(X_test)
confusion_matrix(y_test, prediction)
```

[4]:
```
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  0,  9]])
```

10 Locally Weighted Regression

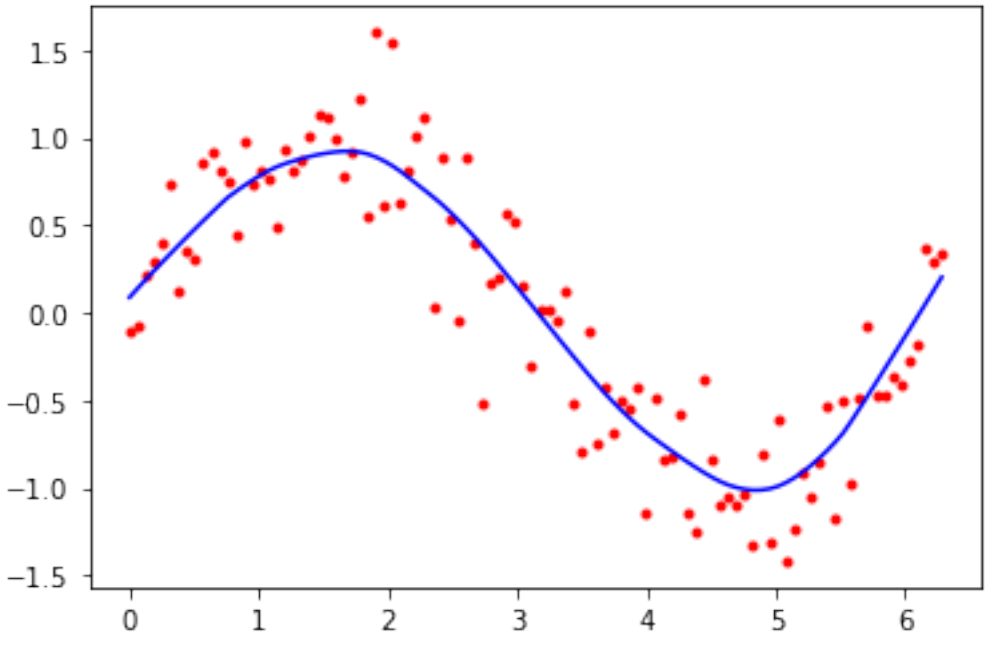
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

[1]:
```
from math import ceil
import math
import numpy as np
from scipy import linalg
```

[2]:
```
def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w**3)**3
    y_estimate = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights),
                           np.sum(weights * x)],
                          [np.sum(weights * x),
                           np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            y_estimate[i] = beta[0] + beta[1] * x[i]

        residuals = y - y_estimate
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta**2)**2

    return y_estimate
```

[4]:
```
import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f = 0.25
iterations = 3
y_estimate = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt
plt.plot(x, y, "r.")
plt.plot(x, y_estimate, "b-")
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7fa062eb0a90>]
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5
6