01

November 18, 2021

[13]:
```
import random
import csv
```

[14]:
```
# Dataset

data = []
with open ('enjoysport.csv') as file:
    reader = csv.reader(file)
    for row in reader:
        data.append(row)
print(data)
```

```
[['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']]
```

[15]:
```
# No of attributes

n = len(data[0])-1

hypothesis = data[0].copy()[:-1]
print('Initial hypothesis', hypothesis)
```

```
Initial hypothesis ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
```

[16]:
```
# Find S algorithm

for i in range (0, len(data)):
    if data[i][n] == 'Yes':
        for j in range (0, n):
            if hypothesis[j] != '?' and hypothesis[j] != data[i][j]:
                hypothesis[j] = '?'
    print('Hypothesis after {} iteration {}'.format(i+1, hypothesis))
```

```
Hypothesis after 1 iteration ['sunny', 'warm', 'normal', 'strong', 'warm',
'same']
Hypothesis after 2 iteration ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Hypothesis after 3 iteration ['sunny', 'warm', '?', 'strong', 'warm', 'same']
```

```
Hypothesis after 4 iteration ['sunny', 'warm', '?', 'strong', '?', '?']
```

[17]:
```python
print('Final Hypothesis : ', hypothesis)
```

```
Final Hypothesis :  ['sunny', 'warm', '?', 'strong', '?', '?']
```

02

November 18, 2021

[45]:

```python
import numpy as np
import pandas as pd
import csv
```

[46]:

```python
# Loading dataset

X = []
y = []

with open ('c1.csv') as file:
    reader = csv.reader(file)
    for row in reader:

        # Select every column except last column
        X.append(row[:-1])

        # Select last column
        y.append(row[-1])
```

[47]:

```python
# Candidate Elimination algorithm

def learn (X, y):

    # Number of attributes
    n = len(X[0])

    # Specific hypothesis
    specific = X[0].copy()

    # General hypothesis
    general = [['?' for _ in range(n)] for _ in range(n)]

    for i, h in enumerate(X):

        if y[i] == 'Y':

            for x in range (n):
                if h[x] != specific[x]:
```

```
                    specific[x] = '?'
                    general[x][x] = '?'

        elif y[i] == 'N':

            for x in range (n):
                if h[x] != specific[x]: general[x][x] = specific[x]
                else: general[x][x] = '?'

    # Remove elements from general hypothesis if its equal to [ '?', '?', '?', .
↪..., '?' ]
    indices = [i for i, val in enumerate(general) if val == (['?'] * n)]
    for _ in indices: general.remove(['?'] * n)

    return specific, general
```

[48]:
```
specific, general = learn(X,y)

print('Specific hypothesis', specific, sep='\n')
print()
print('General hypothesis', general, sep='\n')
```

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

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

03

November 18, 2021

[56]:
```python
import math
import csv
import pandas as pd
import numpy as np
```

[57]:
```python
# Load dataset

def load_dataset():
    reader = csv.reader(open('PlayTennis.csv', 'r'))

    data = list(reader)
    header = data.pop(0)

    return data, header
```

[58]:
```python
class Node:

    def __init__ (self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = "" # NULL indicates children exists. Not Null indicates␣
 ↪this is a Leaf Node
```

[59]:
```python
def subtables (data, col, delete):
    dic = {}

    colData = [row[col] for row in data]
    attr = list(set(colData))

    for k in attr: dic[k] = []

    for y in range (len(data)):
        key = data[y][col]
        if delete: del data[y][col]
        dic[key].append(data[y])

    return attr, dic
```

[60]:

```
def entropy (S):
    attr = list(set(S))

    if len(attr) == 1: return 0

    counts = [0] * len(attr)

    for i in range (len(attr)): counts[i] = sum( [1 for x in S if x == attr[i]]␣
↪) / (len(S) * 1.0)

    sums = 0
    for cnt in counts: sums += -1 * cnt * math.log(cnt,2)

    return sums
```

[61]:

```
def compute_gain (data, col):
    attr, dic = subtables (data, col, delete=False)

    total_entropy = entropy([row[-1] for row in data])

    for x in range (len(attr)):
        ratio = len(dic[attr[x]]) / (len(data) * 1.0)

        entro = entropy([ row[-1] for row in dic[attr[x]] ])

        total_entropy -= ratio * entro

    return total_entropy
```

[62]:

```
def build_tree (data, header):
    y = [row[-1] for row in data]

    if len(set(y)) == 1:
        node = Node("")
        node.answer = y[0]
        return node

    n = len(data[0])-1
    gains = [compute_gain(data, col) for col in range(n)]

    split = gains.index(max(gains))
    node = Node(header[split])
    fea = header[:split] + header[split+1:]

    attr, dic = subtables(data, split, delete=True)

    for x in range (len(attr)):
```

```python
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))

    return node
```

[63]:
```python
def print_tree (node, level):
    if node.answer != "":
        print("---"*level, node.answer)
        return

    print("---"*level, node.attribute)

    for value, n in node.children:
        print("---"*(level+1), value)
        print_tree(n, level+2)
```

[64]:
```python
data, header = load_dataset()
```

[65]:
```python
node = build_tree(data, header)
print_tree(node, 0)
```

```
 Humidity
--- normal
------ Outlook
--------- sunny
------------ yes
--------- overcast
------------ yes
--------- rain
------------ Wind
--------------- strong
------------------ no
--------------- weak
------------------ yes
--- high
------ Outlook
--------- sunny
------------ Temperature
--------------- hot
------------------ no
--------------- mild
------------------ no
--------- overcast
------------ yes
--------- rain
------------ Wind
--------------- strong
------------------ no
```

```
--------------- weak
------------------ yes
```

04

November 18, 2021

[11]:
```python
import numpy as np
from numpy import random as rand
```

[12]:
```python
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
```

[13]:
```python
def sigmoid (x):
    return 1 / (1 + np.exp(-x))

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

[14]:
```python
epoch = 1000
eta = 0.2

input_neurons = len(X[0])
hidden_neurons = 3
output_neurons = len(y[0])

wh = rand.uniform(size = (input_neurons, hidden_neurons))
bh = rand.uniform(size = (1, hidden_neurons))

wout = rand.uniform(size = (hidden_neurons, output_neurons))
bout = rand.uniform(size = (1, output_neurons))
```

[15]:
```python
for _ in range (epoch):

    # Forward propagation

    h_ip = np.dot(X, wh) + bh
    h_act = sigmoid(h_ip)

    o_ip = np.dot(h_act, wout) + bout
    o_act = sigmoid(o_ip)
```

```python
    # Backward propagation

    e_o = y - o_act
    out_grad = sigmoid_grad(o_act)
    d_output = e_o * out_grad

    e_h = d_output.dot(wout.T)
    hidden_grad = sigmoid_grad(h_act)
    d_hidden = e_h * hidden_grad

    wout += h_act.T.dot(d_output) * eta
    wh += X.T.dot(d_hidden) * eta
```

[16]:
```python
print("Normalised Input", X, sep='\n')
print()
print("Actual Output", y, sep='\n')
print()
print("Predicted Output", o_act, sep='\n')
```

```
Normalised Input
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output
[[0.92]
 [0.86]
 [0.89]]

Predicted Output
[[0.89350942]
 [0.88456829]
 [0.89228046]]
```

05

November 20, 2021

[45]:
```python
import csv, math, random
import statistics as st
```

[46]:
```python
def load_csv (filename):
    reader = csv.reader(open(filename, 'r'))
    data = list(reader)

    for i in range (len(data)):
        data[i] = [float(x) for x in data[i]]

    return data
```

[47]:
```python
def split_data (data, ratio):
    test_size = int(len(data) * ratio)
    train_set = list(data)
    test_set = []

    for _ in range (test_size):
        index = random.randrange(len(train_set))
        test_set.append( train_set.pop(index) )

    return train_set, test_set
```

[48]:
```python
def separate_by_class (data):
    separated = {}
    for x in data:
        if x[-1] not in separated:
            separated[x[-1]] = []
        separated[x[-1]].append(x)
    return separated
```

[49]:
```python
def compute_mean_std (data):
    mean_std = [ (st.mean(attr), st.stdev(attr)) for attr in zip(*data) ]
    del mean_std[-1] # It is for labels
    return mean_std
```

[50]:
```python
def summarize_by_class (data):
    separated = separate_by_class(data)
```

```python
    summary = {}
    # Class and Instance
    for cls, inst in separated.items():
        summary[cls] = compute_mean_std(inst)
    return summary
```

[51]:
```python
def estimate_probability (x, mean, stdev):
    # e ^ ( -err^2 / ( 2 * stdev^2 ) )
    exp = math.exp( -(math.pow(x-mean,2)) / (2 * (math.pow(stdev,2)) ))
    return (1 / (math.sqrt(2*math.pi)) * stdev) * exp
```

[52]:
```python
def calculate_class_probability (summaries, test_vector):
    p = {}
    for cls, summary in summaries.items():
        p[cls] = 1
        for i in range (len(summary)):
            mean, stdev = summary[i]
            x = test_vector[i]
            p[cls] -= estimate_probability(x, mean, stdev)
    return p
```

[53]:
```python
def predict (summaries, test_vector):
    all_p = calculate_class_probability(summaries, test_vector)
    best_label, best_prob = None, -1

    for label, p in all_p.items():
        if best_label is None or p > best_prob:
            best_prob = p
            best_label = label

    return best_label
```

[54]:
```python
def perform_classification (summaries, test_set):
    predictions = []
    for x in test_set:
        prediction = predict(summaries, x)
        predictions.append( prediction )
    return predictions
```

[55]:
```python
def get_accuracy (test_set, predictions):
    correct = 0
    for i in range (len(test_set)):
        if test_set[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0
```

[56]:
```
data = load_csv('diabetes.csv')
print(len(data))
print(len(data[0])-1)
```

```
768
8
```

[57]:
```
split_ratio = 0.2
train_set, test_set = split_data(data, split_ratio)
print(len(train_set))
print(len(test_set))
```

```
615
153
```

[58]:
```
summaries = summarize_by_class(data)
print(summaries)
```

```
{1.0: [(4.865671641791045, 3.741239044041554), (141.25746268656715,
31.939622058007203), (70.82462686567165, 21.491811650604127),
(22.16417910447761, 17.679711400465692), (100.33582089552239,
138.68912473153495), (35.14253731343283, 7.262967242346375), (0.5505,
0.37235448355461087), (37.06716417910448, 10.968253652367915)], 0.0: [(3.298,
3.0171845826218893), (109.98, 26.14119975535359), (68.184, 18.063075413305828),
(19.664, 14.889947113744233), (68.792, 98.86528929231788), (30.3042,
7.689855011650115), (0.429734, 0.29908530435741093), (31.19,
11.66765479163115)]}
```

[59]:
```
predictions = perform_classification(summaries, test_set)
accuracy = get_accuracy(test_set, predictions)
print('Accuracy =', accuracy)
```

```
Accuracy = 71.24183006535948
```

06

November 20, 2021

[26]:
```
import pandas as pd
from sklearn.model_selection import train_test_split as split
from sklearn.feature_extraction.text import CountVectorizer as CV
from sklearn.naive_bayes import MultinomialNB as MNB
from sklearn import metrics
```

[27]:
```
data = pd.read_csv('data6.csv', names=['message', 'result'])
data.shape
```

[27]: (8, 2)

[28]:
```
data['target'] = data.result.map({'pos':1, 'neg':0})
X = data['message']
y = data['target']
```

[29]:
```
x_train, x_test, y_train, y_test = split(X, y)
print(x_train.shape)
print(x_test.shape)
```

```
(6,)
(2,)
```

[30]:
```
vect = CV()

x_train_dtm = vect.fit_transform(x_train)
x_test_dtm = vect.transform(x_test)

print('Features extracted using CV', x_train_dtm.shape[1])
```

```
Features extracted using CV 29
```

[31]:
```
df = pd.DataFrame(x_train_dtm.toarray(), columns=vect.get_feature_names_out())
print(df)
```

```
   about  amazing  an  bad  beers  enemy  feel  fun  good  great  …  these  \
0      1        0   0    0      1      0     1    0     1      0  …      1
1      0        0   0    1      0      0     0    0     0      0  …      0
2      0        1   1    0      0      0     0    0     0      0  …      0
3      0        0   0    0      0      1     0    0     0      0  …      0
4      0        0   0    0      0      0     0    1     1      0  …      0
```

```
5       0         0    0     0       0       0      0     0      0       1  …       0

   this  to  today  tomorrow  very  we  went  what  will
0     0   0      0         0     1   0     0     0     0
1     0   1      0         0     0   0     0     0     0
2     1   0      0         0     0   0     0     0     0
3     0   1      1         0     0   0     1     0     0
4     0   0      0         1     0   1     0     0     1
5     0   0      0         0     0   0     0     1     0

[6 rows x 29 columns]
```

[32]:

```python
clf = MNB().fit(x_train_dtm, y_train)
```

[33]:

```python
predicted = clf.predict(x_test_dtm)

for x, y in zip(x_test, predicted):
    print(x, y)
```

```
I love this sandwich 1
This is my best work 1
```

[34]:

```python
print('Accuracy Metrics\n')
print('Accuracy', metrics.accuracy_score(y_test, predicted))
print('Recall', metrics.recall_score(y_test, predicted))
print('Precision', metrics.precision_score(y_test, predicted))
```

```
Accuracy Metrics

Accuracy 1.0
Recall 1.0
Precision 1.0
```

[35]:

```python
print('Confusion Matrix\n')
print(metrics.confusion_matrix(y_test, predicted))
```

```
Confusion Matrix

[[2]]
```

08

November 20, 2021

[28]:
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture as GM
```

[29]:
```
iris = datasets.load_iris()
```

[30]:
```
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Target']
```

[31]:
```
model = KMeans(n_clusters=3)
```

[32]:
```
model.fit(X)
```

[32]:
```
KMeans(n_clusters=3)
```

[33]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

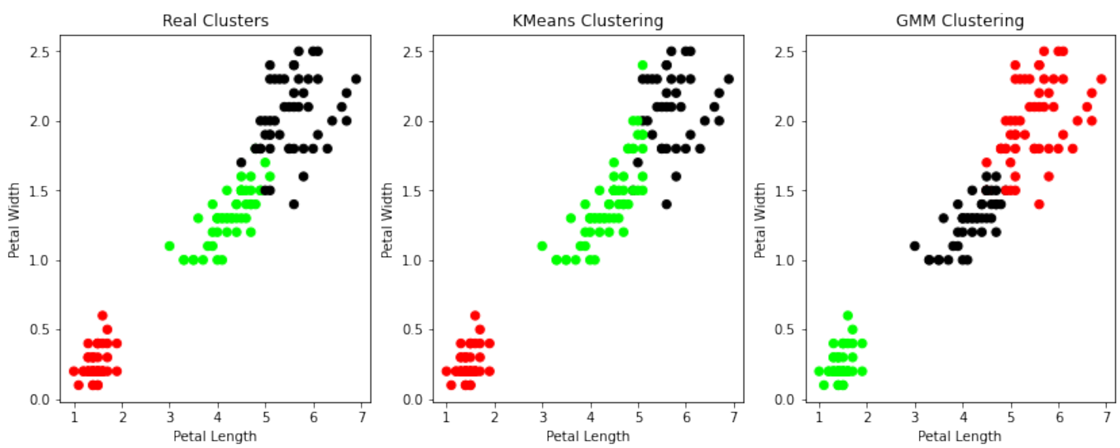
[34]:
```
gmm = GM(n_components=3)
```

[35]:
```
gmm.fit(xs)
gmm_y = gmm.predict(xs)
```

[36]:
```
plt.figure(figsize=(14,5))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1,3,1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Target], s=40)
plt.title('Real Clusters')
```

```python
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(1,3,2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('KMeans Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(1,3,3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

[36]: Text(0, 0.5, 'Petal Width')



[37]:
```python
print('Observation: GMM using EM algorithm based clustering matched the true␣
↪labels more closely than the Kmeans.')
```

```
Observation: GMM using EM algorithm based clustering matched the true labels
more closely than the Kmeans.
```

09

November 20, 2021

[13]:
```python
from sklearn.model_selection import train_test_split as split
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn import datasets
```

[14]:
```python
iris = datasets.load_iris()
```

[15]:
```python
x_train, x_test, y_train, y_test = split(iris.data, iris.target, test_size=0.2)
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(120, 4) (30, 4)
(120,) (30,)
```

[16]:
```python
print('Target names', iris.target_names)
```

```
Target names ['setosa' 'versicolor' 'virginica']
```

[17]:
```python
classifier = KNN(n_neighbors=1)
```

[18]:
```python
classifier.fit(x_train, y_train)
```

[18]:
```
KNeighborsClassifier(n_neighbors=1)
```

[19]:
```python
y_pred = classifier.predict(x_test)
```

[20]:
```python
print('Accuracy', classifier.score(x_test, y_test))
```

```
Accuracy 0.9333333333333333
```

10

November 20, 2021

[19]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

[20]:
```
def kernel (point, x_mat, k):
    m, n = np.shape(x_mat)
    weights = np.mat(np.eye(m))
    for j in range (m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff * diff.T / (-2.0 * k**2))
    return weights
```

[21]:
```
def local_weight (point, x_mat, y_mat, k):
    weight = kernel(point, x_mat, k)
    return (X.T * (weight*X)).I * (X.T * (weight * y_mat.T))
```

[22]:
```
def local_weight_regression (x_mat, y_mat, k):
    m, n = np.shape(x_mat)
    y_pred = np.zeros(m)
    for i in range (m):
        y_pred[i] = x_mat[i] * local_weight(x_mat[i], x_mat, y_mat, k)
    return y_pred
```

[23]:
```
def graph_plot (X, y_pred):
    sort_index = X[:,1].argsort(0)
    x_sort = X[sort_index][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill, tip, color='green')
    ax.plot(x_sort[:,1], y_pred[sort_index], color='red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show()
```

[24]:
```
data = pd.read_csv('data10_tips.csv')

bill = np.array(data.total_bill)
tip = np.array(data.tip)
```

[25]:
```
m_bill = np.mat(bill)
m_tip = np.mat(tip)
```

[26]:
```
m = np.shape(m_bill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T, m_bill.T))
```

[27]:
```
y_pred = local_weight_regression(X, m_tip, 3)
```

[28]:
```
graph_plot(X, y_pred)
```