RepMLA_Etivity_3_Ciaran_Cox_19170157

August 11, 2021

#**Artificial Intelligence - MSc** CS6501 - MACHINE LEARNING AND APPLICATIONS #**Business Analytics - MSc** ET5003 - MACHINE LEARNING APPLICATIONS ##***Annual Repeat*** ###Instructor: Enrique Naredo

###RepMLA_Etivity-3

[1]:
```
#@title Current Date
Today = '2021-08-04' #@param {type:"date"}
```

[2]:
```
#@markdown ---
#@markdown ### Enter your details here:
Student_ID = "19170157" #@param {type:"string"}
Student_full_name = "Ciaran Cox" #@param {type:"string"}
#@markdown ---
```

[3]:
```
#@title Notebook information
Notebook_type = 'Etivity' #@param ["Example", "Lab", "Practice", "Etivity",␣
↪"Assignment", "Exam"]
Version = 'Final' #@param ["Draft", "Final"] {type:"raw"}
Submission = True #@param {type:"boolean"}
```

1 Etivity-3 Task 1 Fuzzy Systems

1.1 Introduction

Using el notebook RepMLA_3_1.ipynb as a baseline, solve the following task

Antecedents:

- Fuzzy triangular sets for 'service': unacceptable, poor, acceptable, good, amazing
- Fuzzy trapezoidal sets for 'quality': really_bad, bad, decent, great, really_great

Consequents:

- Fuzzy Gaussian sets for 'tip': very_low, low, medium, high, very_high
- Design 5 rules using the antecedent and consequents
- Give 5 examples of usage, for instance, from the notebook; the service as 9.8, and the quality as 6.5

1.2 Dataset

- There is no dataset used for Task 1
- skfuzzy is used to create a dataset to create a set of rules to determine how much to tip based on the service and food quality

1.3 Method

- We create the Antecedents and Consequents are created
- Service and Quality are rated from 0-10 and tip is rated from 0-25
- The rules are set to determine how much of a tip should be given.

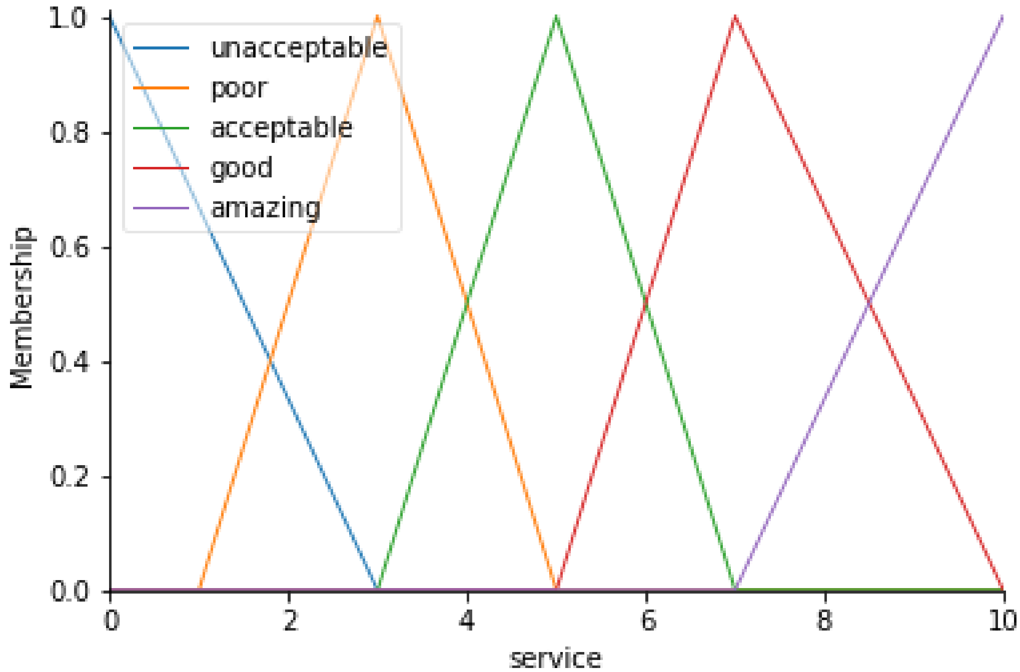
[4]:
```
#! pip install scikit-fuzzy

import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

[5]:
```
# Antecedent/Consequent (Input/Output) variables

#Antecedent 1
service = ctrl.Antecedent(np.arange(0, 10+1, 1), 'service')

#Antecedent 2
quality = ctrl.Antecedent(np.arange(0, 10+1, 1), 'quality')

#Consequent
tip = ctrl.Consequent(np.arange(0, 25+1, 1), 'tip')
```

[6]:
```
# Give service and quality 5 membership functions each
#service.automf(3)
#quality.automf(3)
```

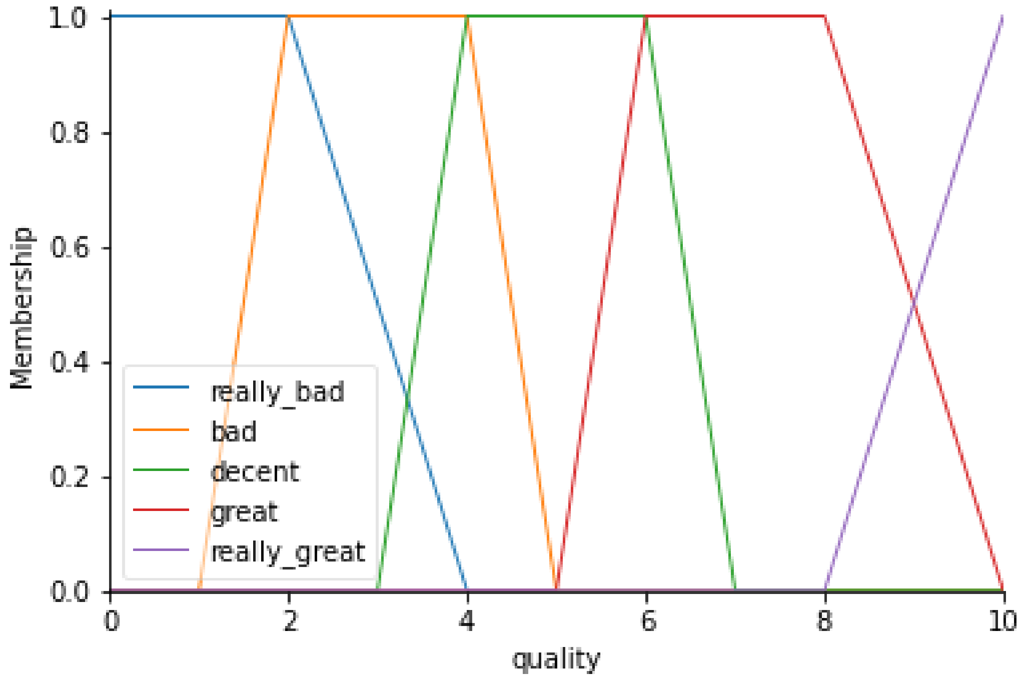
[7]:
```
# unacceptable, poor, acceptable, good, amazing
# 0 - 10
service['unacceptable'] = fuzz.trimf(service.universe, [0,0,3])
service['poor'] = fuzz.trimf(service.universe, [1,3,5])
service['acceptable'] = fuzz.trimf(service.universe, [3,5,7])
service['good'] = fuzz.trimf(service.universe, [5,7,10])
service['amazing'] = fuzz.trimf(service.universe, [7,10,10])
service.view()
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```



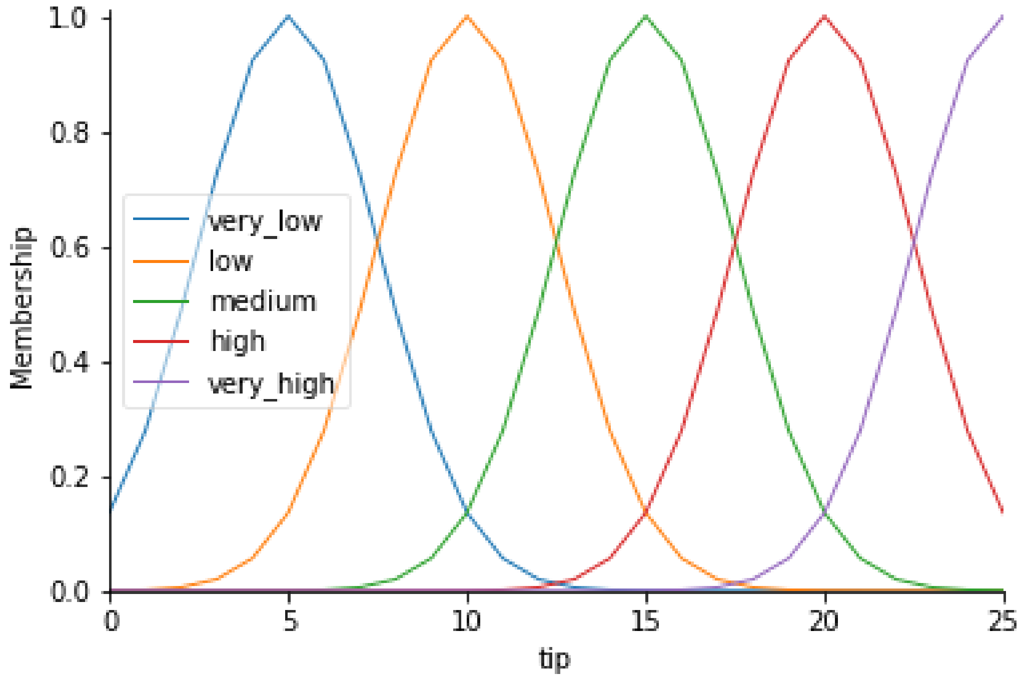
[8]:
```
# unacceptable, poor, acceptable, good, amazing
# 0 - 10
quality['really_bad'] = fuzz.trapmf(quality.universe, [0,0,2,4])
quality['bad'] = fuzz.trapmf(quality.universe, [1,2,4,5])
quality['decent'] = fuzz.trapmf(quality.universe, [3,4,6,7])
quality['great'] = fuzz.trapmf(quality.universe, [5,6,8,10])
quality['really_great'] = fuzz.trapmf(quality.universe, [8,10,10,10])
quality.view()
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```



[9]:
```
# very_low, low, medium, high, very_high
# 5,10,15,20,25

sigma = 2.5

tip['very_low'] = fuzz.gaussmf(tip.universe,5, sigma)
tip['low'] = fuzz.gaussmf(tip.universe,10, sigma)
tip['medium'] = fuzz.gaussmf(tip.universe,15, sigma)
tip['high'] = fuzz.gaussmf(tip.universe,20, sigma)
tip['very_high'] = fuzz.gaussmf(tip.universe,25, sigma)
tip.view()
```

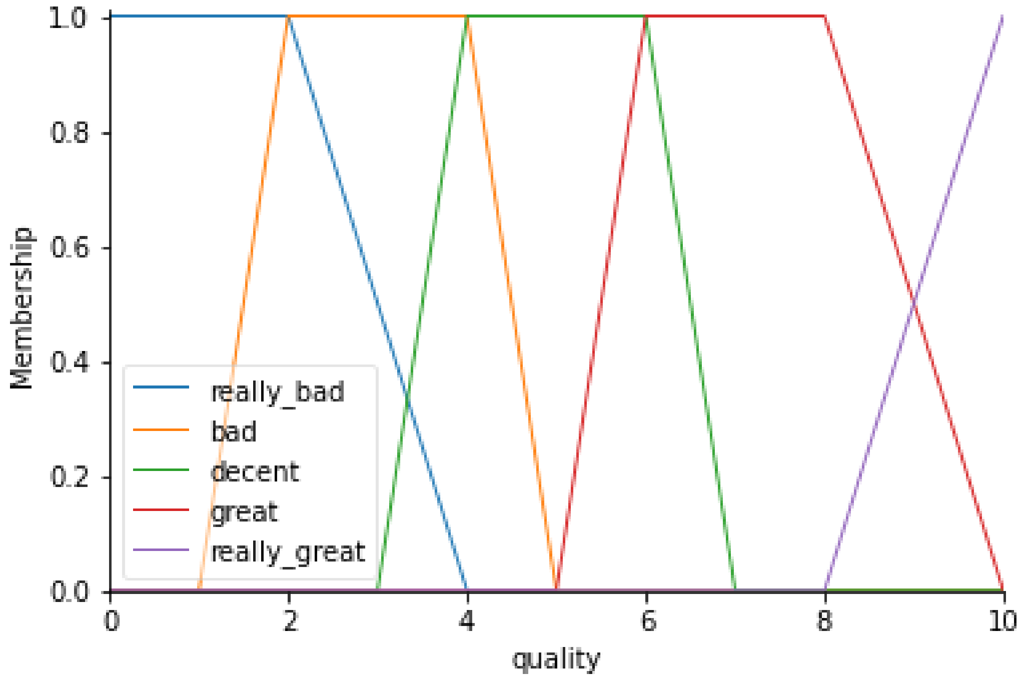
```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```

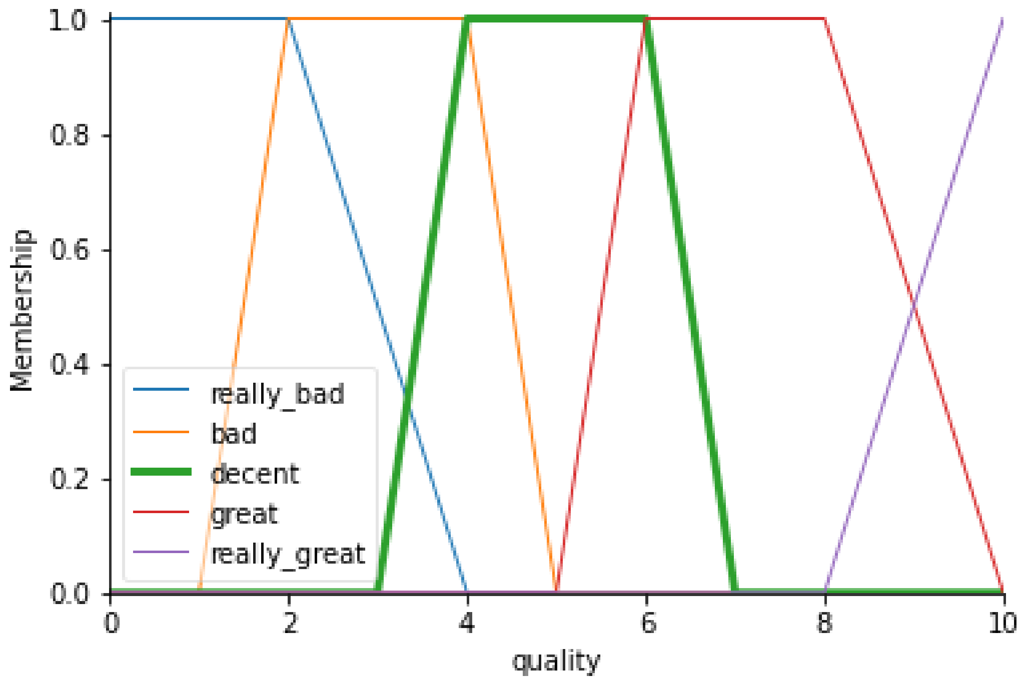


[10]:
```
# # very_low, low, medium, high, very_high
# # 5,10,15,20,25
# tip['very_low'] = fuzz.trimf(tip.universe, [0,0,5])
# tip['low'] = fuzz.trimf(tip.universe, [0,5,10])
# tip['medium'] = fuzz.trimf(tip.universe, [5,10,15])
# tip['high'] = fuzz.trimf(tip.universe, [10,15,20])
# tip['very_high'] = fuzz.trimf(tip.universe, [15,20,25])
```

[11]:
```
quality.view()
quality['decent'].view()
```

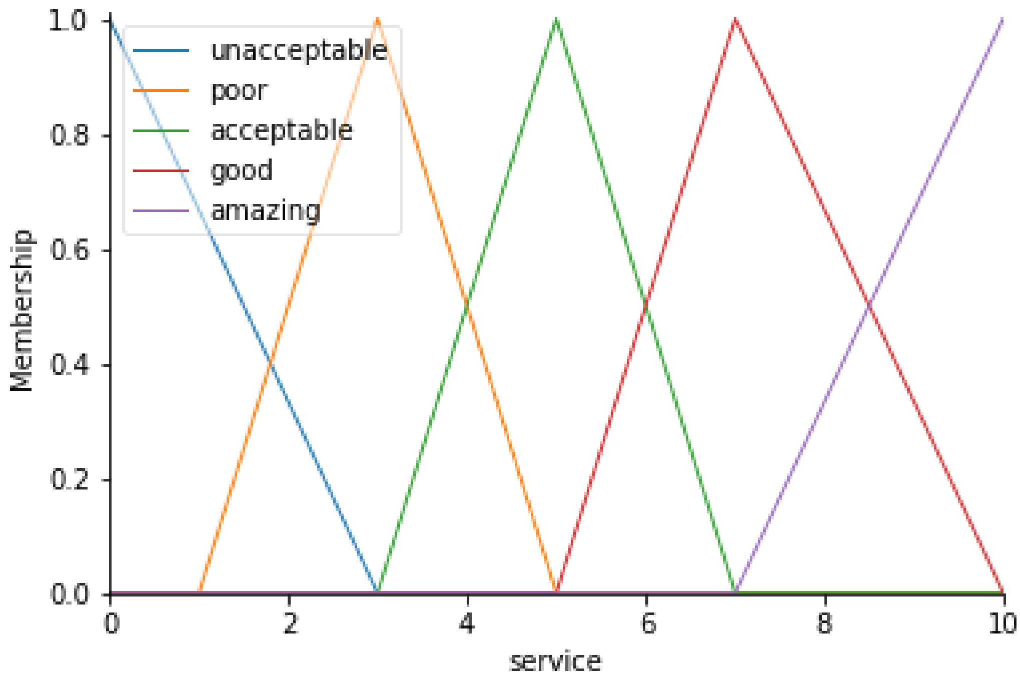
```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\term.py:74: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```

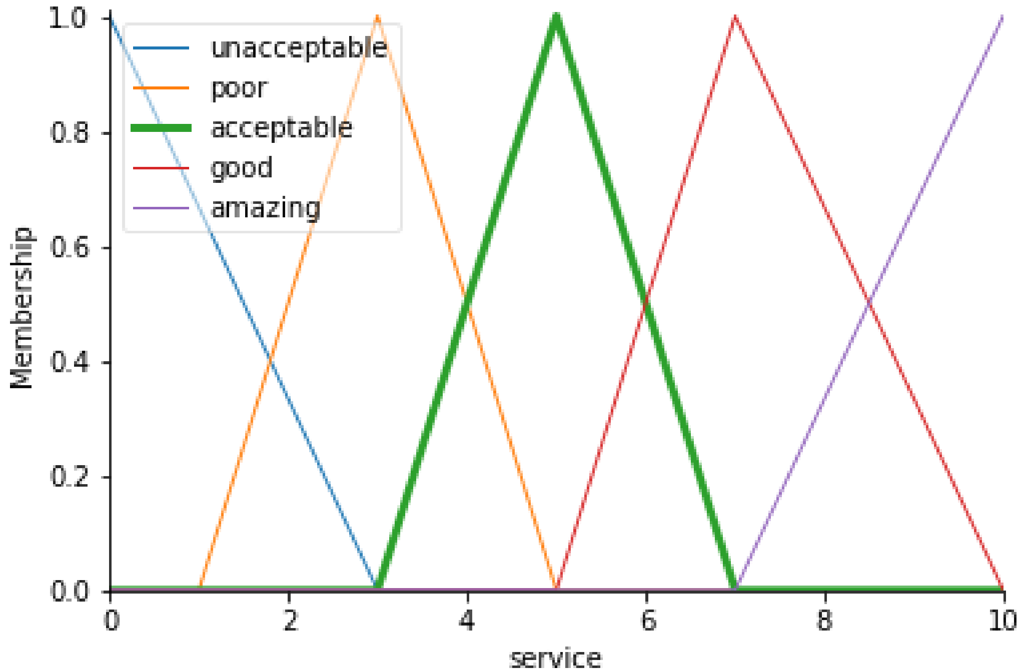




[12]:
```
service.view()
service['acceptable'].view()
```

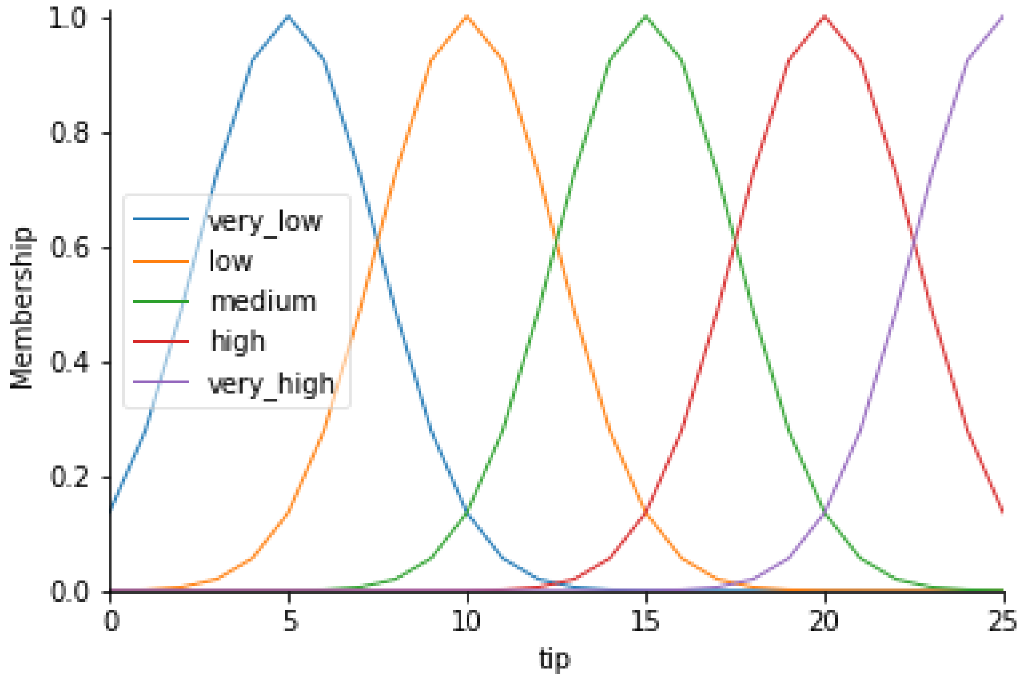
```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\term.py:74: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```



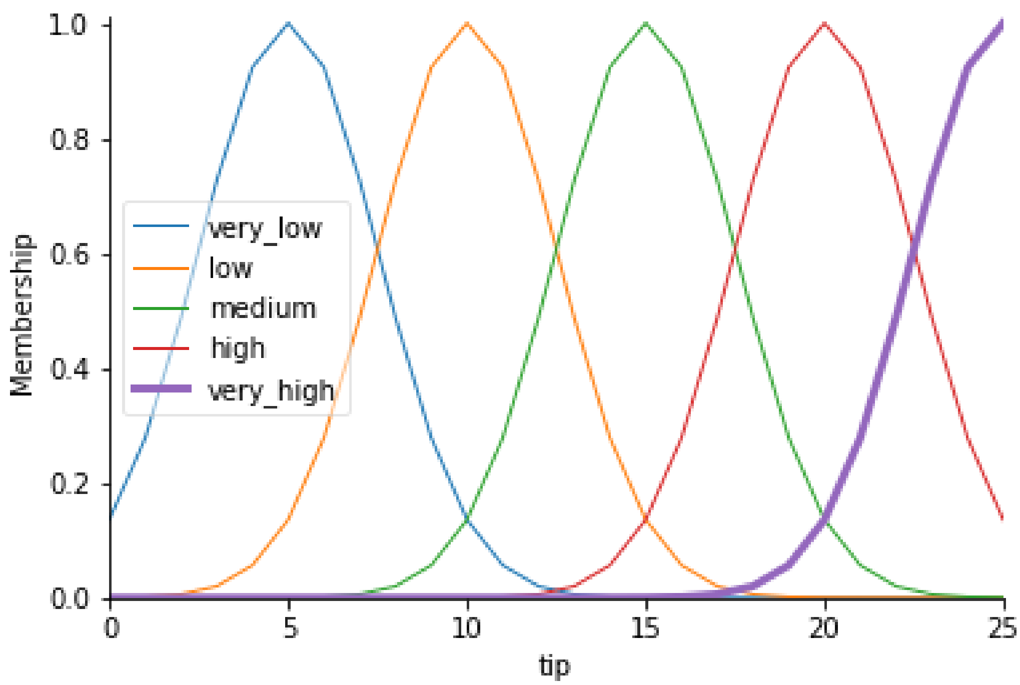


[13]:
```
tip.view()
tip['very_high'].view()
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\term.py:74: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```



1.0
0.8
0.6
0.4
0.2
0.0
Membership
very_low
low
medium
high
very_high
0
5
10
15
20
25
tip



1.0
0.8
0.6
0.4
0.2
0.0
Membership
very_low
low
medium
high
very_high
0
5
10
15
20
25
tip

1.4 Fuzzy Rules

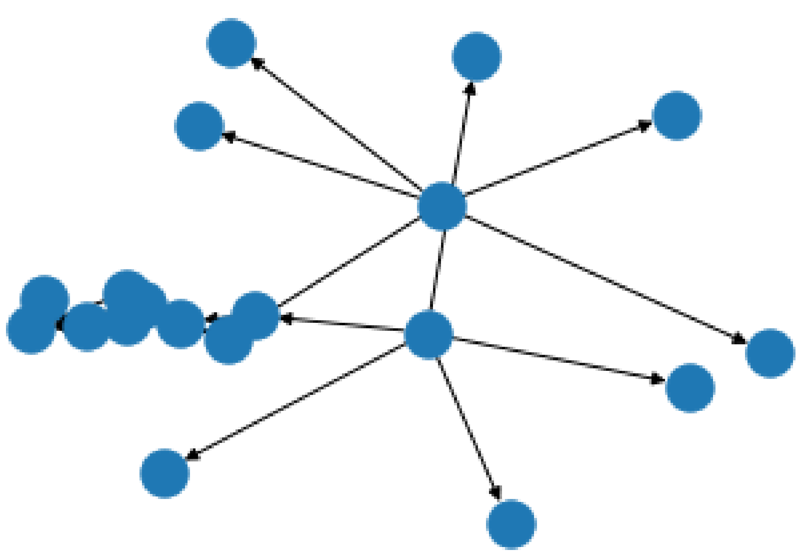
[14]:
```
## 1. If service AND quality is poor tip will be v.low
## 2. If service AND quality is good tip will be v.high
## 3. If quality is mediocre tip will be medium
## 4. If service is average OR decent AND quality is poor OR mediocre tip will␣
↪be low
## 5. If service is mediocre OR average AND quality is mediocre OR decent tip␣
↪will be high

rule_1 = ctrl.Rule(quality['really_bad'] & service['poor'],tip['very_low'])
rule_2 = ctrl.Rule(quality['great'] & service['good'],tip['very_high'])
rule_3 = ctrl.Rule(quality['decent'],tip['medium'])
rule_4 = ctrl.Rule(service['acceptable'] | service['good'] &␣
↪quality['really_bad'] | quality['bad'],tip['low'])
rule_5 = ctrl.Rule(service['poor'] | service['acceptable'] & quality['bad'] |␣
↪quality['decent'],tip['high'])
```

Directed Graphs

[15]:
```
print("Rule 1:")
rule_1.view()
```

```
Rule 1:
```

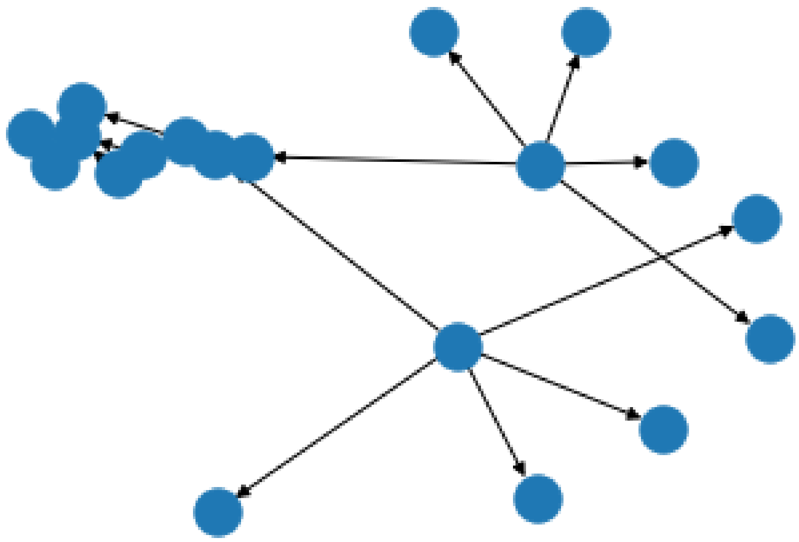
[15]:
```
(<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)
```



[16]:
```
print("Rule 2:")
rule_2.view()
```

Rule 2:

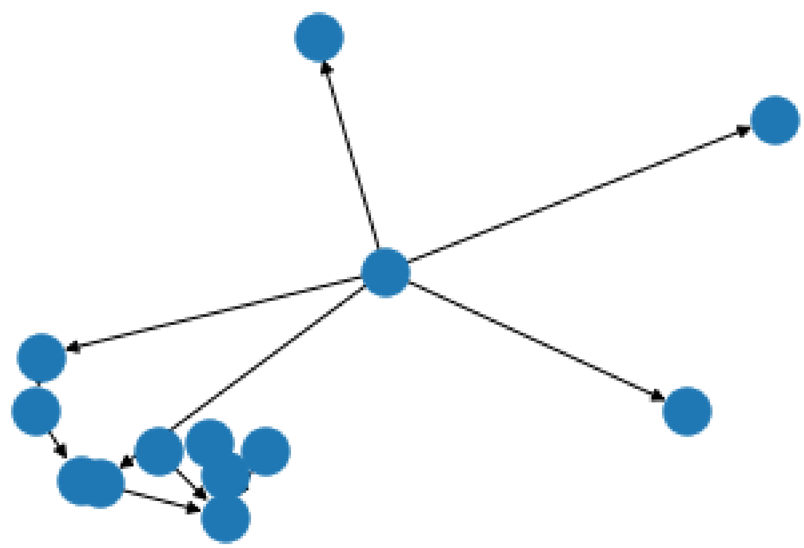
[16]: (<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)



[17]:
```
print("Rule 3:")
rule_3.view()
```

Rule 3:

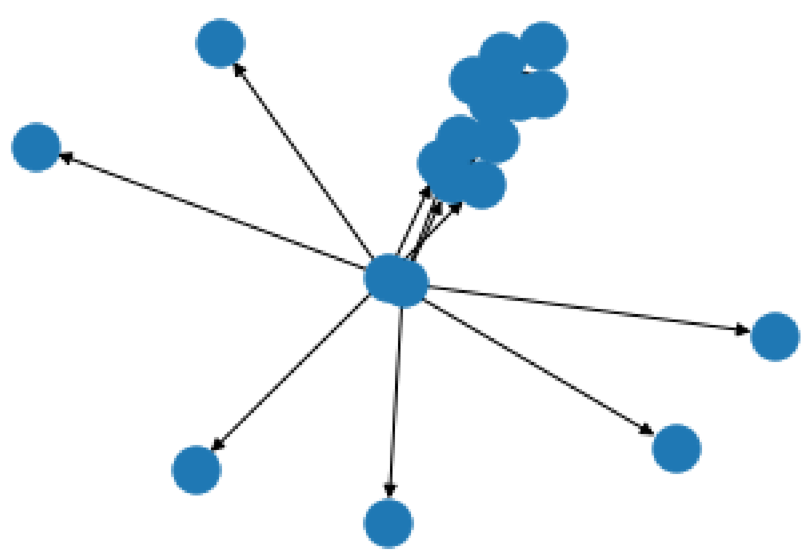
[17]: (<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)



```
[18]: print("Rule 4:")
      rule_4.view()
```

Rule 4:

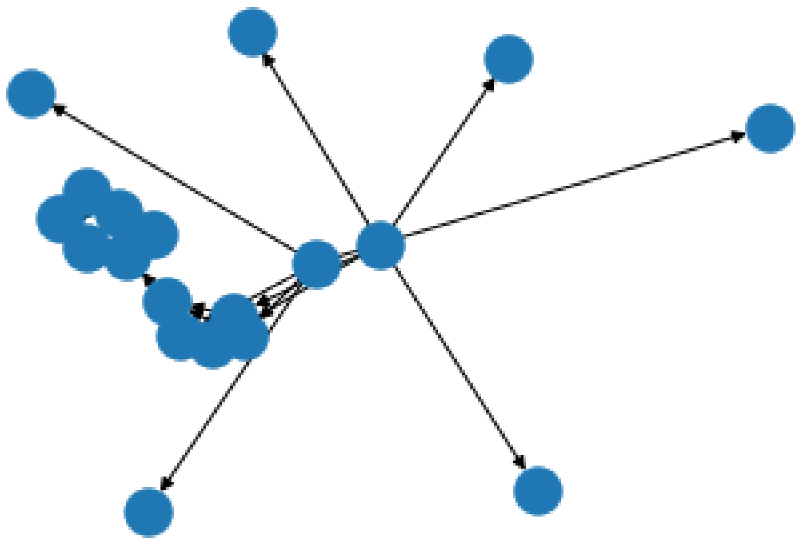
```
[18]: (<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)
```



[19]:
```
print("Rule 5:")
rule_5.view()
```

```
Rule 5:
```

[19]:
```
(<Figure size 432x288 with 1 Axes>, <AxesSubplot:>)
```



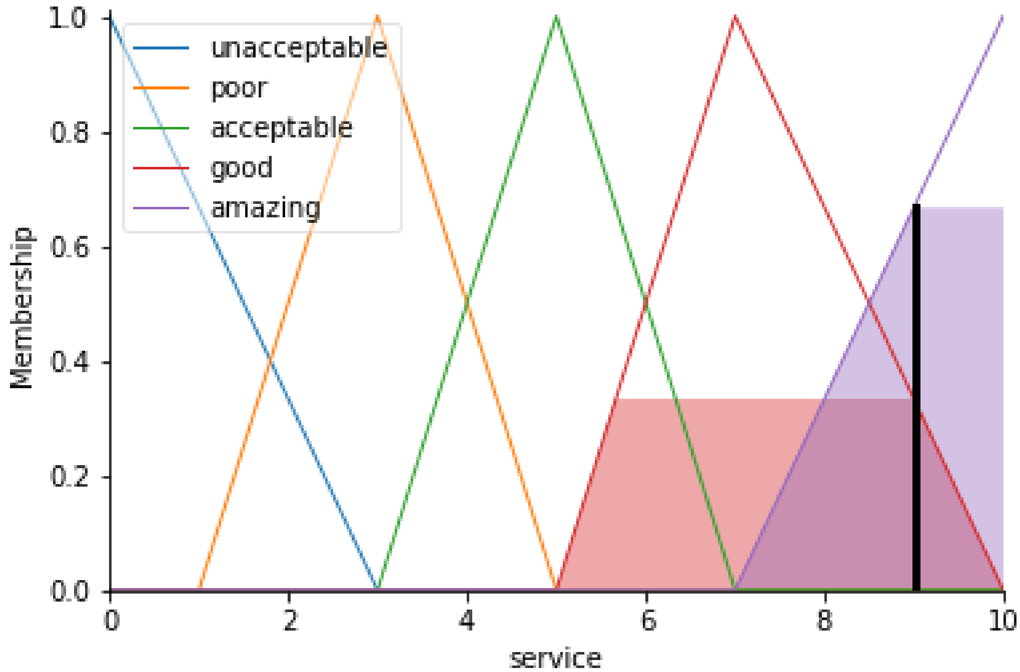
[20]:
```
tipping_ctrl = ctrl.ControlSystem([rule_1,rule_2,rule_3,rule_4,rule_5])
```

[21]:
```
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
```

[22]:
```
tipping.input['quality'] = 9
tipping.input['service'] = 9

tipping.compute()
```

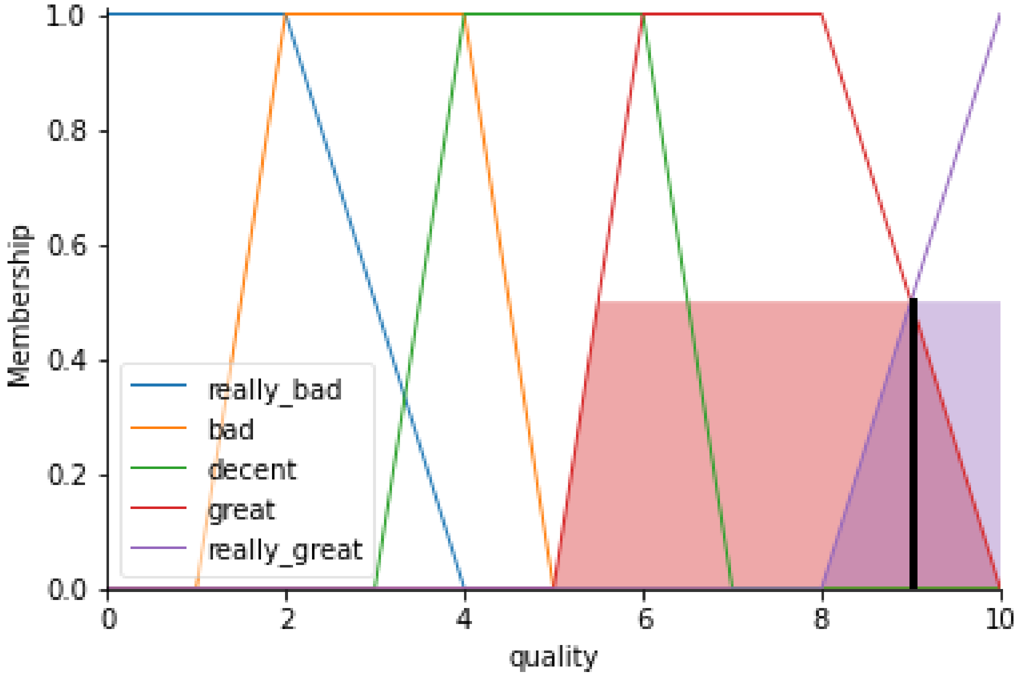
[23]:
```
service.view(sim=tipping)
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```



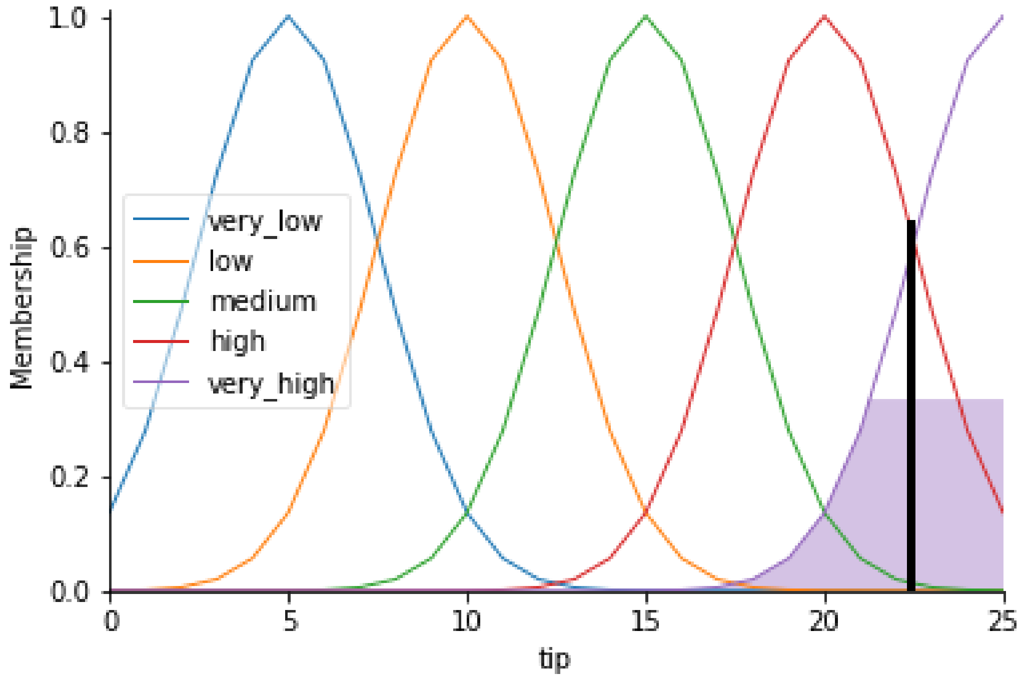
[24]:
```
quality.view(sim=tipping)
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```



[25]:
```
tip.view(sim=tipping)
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\skfuzzy\control\fuzzyvariable.py:122: UserWarning: Matplotlib is
currently using module://ipykernel.pylab.backend_inline, which is a non-GUI
backend, so cannot show the figure.
  fig.show()
```



[26]:
```
print(tipping.output['tip'])
```

```
22.3527832811357
```

1.4.1 Summary

- Fuzzy logic allows for intivitive analysis of complicated systems with a set of rules.
- This process would work well for systems with many, complecated rules that would be difficult to solve manually.
- My setup does not seem to be able to solve for values less than 2,2 (for quality and service) or greater than 9,9.
- This may be due to how the rules are set up.

2 Etivity-3 Task 2 Fuzzy Classification

2.0.1 Introduction

- Using the notebook RepMLA_3_2.ipynb as a baseline, solve the following task
- Consider all the features: ['sepal-length', 'sepal-width', 'petal-length', 'petal-width']
- Perform a binary classification problem considering 'Iris-setosa' , 'Iris-versicolor' 'Iris-versicolor' , 'Iris-virginica' 'Iris-setosa' , 'Iris-virginica'
- Perform a multi-classification problem considering: 'Iris-setosa' , 'Iris-versicolor' , 'Iris-virginica'

2.0.2 Dataset

- The Iris dataset contains information on flowers from 3 different species of flower.

[27]:
```
import pandas as pd
```

[28]:
```
iris_data = 'iris.csv'
```

[29]:
```
df = pd.read_csv(iris_data)
df.head()
```

[29]:

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

[30]:
```
df.columns=['sepal-length', 'sepal-width', 'petal-length', 'petal-width',␣
 ↪'class']
df.head()
```

[30]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

[31]:
```
print(df['class'].unique())
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

2.0.3 Method

- We will create a fuzzy classifier to create a binary classifier for 2 classes
- Another classifier is used for all 3 classes
- The classifiers will be able to determine what kind of flower class there is based on sepal and petal input data.

[32]:
```
# Convert the dataset into 3 binary classification problems:
## 1.'Iris-setosa' , 'Iris-versicolor'
## 2.'Iris-versicolor' , 'Iris-virginica'
## 3.'Iris-setosa' , 'Iris-virginica'
df_1 = df[~(df['class']=='Iris-virginica')]
df_2 = df[~(df['class']=='Iris-setosa')]
df_3 = df[~(df['class']=='Iris-versicolor')]
df_all = df
```

```
print(df_1['class'].unique())
print(df_2['class'].unique())
print(df_3['class'].unique())
print(df_all['class'].unique())
```

```
['Iris-setosa' 'Iris-versicolor']
['Iris-versicolor' 'Iris-virginica']
['Iris-setosa' 'Iris-virginica']
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

[33]:
```
# Assigning binary values to each of the new datasets
# Dataset 1
# class 0
df_1.replace(to_replace='Iris-setosa', value=0, inplace=True)
# class 1
df_1.replace(to_replace='Iris-versicolor', value=1, inplace=True)

# Dataset 2
# class 0
df_2.replace(to_replace='Iris-versicolor', value=0, inplace=True)
# class 1
df_2.replace(to_replace='Iris-virginica', value=1, inplace=True)

# Dataset 3
# class 0
df_3.replace(to_replace='Iris-setosa', value=0, inplace=True)
# class 1
df_3.replace(to_replace='Iris-virginica', value=1, inplace=True)

# Dataset all
# class 0
df_all.replace(to_replace='Iris-setosa', value=0, inplace=True)
# class 1
df_all.replace(to_replace='Iris-versicolor', value=1, inplace=True)
# class 2
df_all.replace(to_replace='Iris-virginica', value=2, inplace=True)
```

```
C:\Users\Ciaran\anaconda3\envs\Repeat\lib\site-
packages\pandas\core\frame.py:5233: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().replace(
```

[34]:
```
print(df_1.head())
print(df_2.head())
```

```
print(df_3.head())
print(df_all.head())
```

```
   sepal-length  sepal-width  petal-length  petal-width  class
0           4.9          3.0           1.4          0.2      0
1           4.7          3.2           1.3          0.2      0
2           4.6          3.1           1.5          0.2      0
3           5.0          3.6           1.4          0.2      0
4           5.4          3.9           1.7          0.4      0
    sepal-length  sepal-width  petal-length  petal-width  class
49           7.0          3.2           4.7          1.4      0
50           6.4          3.2           4.5          1.5      0
51           6.9          3.1           4.9          1.5      0
52           5.5          2.3           4.0          1.3      0
53           6.5          2.8           4.6          1.5      0
   sepal-length  sepal-width  petal-length  petal-width  class
0           4.9          3.0           1.4          0.2      0
1           4.7          3.2           1.3          0.2      0
2           4.6          3.1           1.5          0.2      0
3           5.0          3.6           1.4          0.2      0
4           5.4          3.9           1.7          0.4      0
   sepal-length  sepal-width  petal-length  petal-width  class
0           4.9          3.0           1.4          0.2      0
1           4.7          3.2           1.3          0.2      0
2           4.6          3.1           1.5          0.2      0
3           5.0          3.6           1.4          0.2      0
4           5.4          3.9           1.7          0.4      0
```

Random Sampling

[35]:
```
df_1_s = df_1.sample(frac=1)
print(df_1_s.head())

df_2_s = df_2.sample(frac=1)
print(df_2_s.head())

df_3_s = df_3.sample(frac=1)
print(df_3_s.head())

df_all_s = df_all.sample(frac=1)
print(df_all_s.head())
```

```
    sepal-length  sepal-width  petal-length  petal-width  class
16           5.1          3.5           1.4          0.3      0
68           5.6          2.5           3.9          1.1      1
4            5.4          3.9           1.7          0.4      0
82           6.0          2.7           5.1          1.6      1
56           4.9          2.4           3.3          1.0      1
```

```
     sepal-length  sepal-width  petal-length  petal-width  class
72            6.1          2.8           4.7          1.2      0
131           6.4          2.8           5.6          2.2      1
90            6.1          3.0           4.6          1.4      0
75            6.8          2.8           4.8          1.4      0
68            5.6          2.5           3.9          1.1      0
     sepal-length  sepal-width  petal-length  petal-width  class
106           7.3          2.9           6.3          1.8      1
9             5.4          3.7           1.5          0.2      0
102           6.3          2.9           5.6          1.8      1
46            4.6          3.2           1.4          0.2      0
109           6.5          3.2           5.1          2.0      1
     sepal-length  sepal-width  petal-length  petal-width  class
34            5.0          3.2           1.2          0.2      0
15            5.4          3.9           1.3          0.4      0
86            6.3          2.3           4.4          1.3      1
53            6.5          2.8           4.6          1.5      1
126           6.1          3.0           4.9          1.8      2
```

Normalization

[36]:
```python
feature_names = df.columns[0:4]
feature_values_1 = df_1_s[feature_names].values
feature_values_2 = df_2_s[feature_names].values
feature_values_3 = df_3_s[feature_names].values
feature_values_all = df_all_s[feature_names].values

print(feature_values_1[0:9])
print(feature_values_2[0:9])
print(feature_values_3[0:9])
print(feature_values_all[0:9])
```

```
[[5.1 3.5 1.4 0.3]
 [5.6 2.5 3.9 1.1]
 [5.4 3.9 1.7 0.4]
 [6.  2.7 5.1 1.6]
 [4.9 2.4 3.3 1. ]
 [5.  3.6 1.4 0.2]
 [5.4 3.  4.5 1.5]
 [5.7 3.  4.2 1.2]
 [4.5 2.3 1.3 0.3]]
[[6.1 2.8 4.7 1.2]
 [6.4 2.8 5.6 2.2]
 [6.1 3.  4.6 1.4]
 [6.8 2.8 4.8 1.4]
 [5.6 2.5 3.9 1.1]
 [5.8 2.6 4.  1.2]
 [4.9 2.5 4.5 1.7]
```

```
 [6.  2.9 4.5 1.5]
 [6.2 2.2 4.5 1.5]]
[[7.3 2.9 6.3 1.8]
 [5.4 3.7 1.5 0.2]
 [6.3 2.9 5.6 1.8]
 [4.6 3.2 1.4 0.2]
 [6.5 3.2 5.1 2. ]
 [4.4 2.9 1.4 0.2]
 [7.4 2.8 6.1 1.9]
 [5.3 3.7 1.5 0.2]
 [7.1 3.  5.9 2.1]]
[[5.  3.2 1.2 0.2]
 [5.4 3.9 1.3 0.4]
 [6.3 2.3 4.4 1.3]
 [6.5 2.8 4.6 1.5]
 [6.1 3.  4.9 1.8]
 [4.9 2.5 4.5 1.7]
 [6.5 3.  5.8 2.2]
 [6.3 2.9 5.6 1.8]
 [5.8 2.7 4.1 1. ]]
```

[37]:

```
# min/max values

df_1_min = np.min(feature_values_1, axis=0)
df_1_max = np.max(feature_values_1, axis=0)

df_2_min = np.min(feature_values_2, axis=0)
df_2_max = np.max(feature_values_2, axis=0)

df_3_min = np.min(feature_values_3, axis=0)
df_3_max = np.max(feature_values_3, axis=0)

df_all_min = np.min(feature_values_all, axis=0)
df_all_max = np.max(feature_values_all, axis=0)

print("Min:",df_1_min, "Max:",df_1_max)
print("Min:",df_2_min, "Max:",df_2_max)
print("Min:",df_3_min, "Max:",df_3_max)
print("Min:",df_all_min, "Max:",df_all_max)
```

```
Min: [4.3 2.  1.  0.1] Max: [7.  4.4 5.1 1.8]
Min: [4.9 2.  3.  1. ] Max: [7.9 3.8 6.9 2.5]
Min: [4.3 2.2 1.  0.1] Max: [7.9 4.4 6.9 2.5]
Min: [4.3 2.  1.  0.1] Max: [7.9 4.4 6.9 2.5]
```

[38]:

```
# Normalizing Features
feature_norm_1 = (feature_values_1 - df_1_min) / (df_1_max - df_1_min)
```

```
feature_norm_2 = (feature_values_2 - df_2_min) / (df_2_max - df_2_min)
feature_norm_3 = (feature_values_3 - df_3_min) / (df_3_max - df_3_min)
feature_norm_all = (feature_values_all - df_all_min) / (df_all_max - df_all_min)
```

2.0.4 Training and test sets

[39]:
```
# Split the train/test data 0.7/0.3
training_perc = 0.7

# Number of items in the datasets
data_size_1 = len(feature_norm_1)
data_size_2 = len(feature_norm_2)
data_size_3 = len(feature_norm_3)
data_size_all = len(feature_norm_all)

#feature norm
train_size_1 = round(data_size_1*training_perc)
test_size_1 = data_size_1 - train_size_1

train_size_2 = round(data_size_2*training_perc)
test_size_2 = data_size_2 - train_size_2

train_size_3 = round(data_size_3*training_perc)
test_size_3 = data_size_3 - train_size_3

train_size_all = round(data_size_all*training_perc)
test_size_all = data_size_all - train_size_all

print('Entire dataset size: ', data_size_1)
print('training set size: ', train_size_1)
print('test set size: ', test_size_1)

print('Entire dataset size: ', data_size_2)
print('training set size: ', train_size_2)
print('test set size: ', test_size_2)

print('Entire dataset size: ', data_size_3)
print('training set size: ', train_size_3)
print('test set size: ', test_size_3)

print('Entire dataset size: ', data_size_all)
print('training set size: ', train_size_all)
print('test set size: ', test_size_all)
```

```
Entire dataset size:  99
training set size:  69
test set size:  30
```

```
Entire dataset size:  100
training set size:  70
test set size:  30
Entire dataset size:  99
training set size:  69
test set size:  30
Entire dataset size:  149
training set size:  104
test set size:  45
```

[40]:

```
# Class labels
class_label_1 = df_1_s['class'].values
class_label_2 = df_2_s['class'].values
class_label_3 = df_3_s['class'].values
class_label_all = df_all_s['class'].values

#train sets
X_train_1 = feature_norm_1[0:train_size_1]
y_train_1 = class_label_1[0:train_size_1]
X_train_2 = feature_norm_2[0:train_size_2]
y_train_2 = class_label_2[0:train_size_2]
X_train_3 = feature_norm_3[0:train_size_3]
y_train_3 = class_label_3[0:train_size_3]
X_train_all = feature_norm_all[0:train_size_all]
y_train_all = class_label_all[0:train_size_all]

#test sets
X_test_1 = feature_norm_1[train_size_1:]
y_test_1 = class_label_1[train_size_1:]
X_test_2 = feature_norm_2[train_size_2:]
y_test_2 = class_label_2[train_size_2:]
X_test_3 = feature_norm_3[train_size_3:]
y_test_3 = class_label_3[train_size_3:]
X_test_all = feature_norm_all[train_size_all:]
y_test_all = class_label_all[train_size_all:]
```

[41]:

```
class Animator:
        '''
        An animator class only for animating 2D hyperboxes
        '''

        def __init__(self, box_history, train_patterns, classes, frame_rate,␣
 ↪exp_bound, sensitivity,
                                 filename='fuzzy_animation', verbose=True):
                # TODO: Customizable parameters
```

```
                assert len(box_history) == len(train_patterns), '{}␣
↪(box-history) != {} (train_patterns)'.format(len(box_history),␣
↪len(train_patterns))
                assert len(train_patterns[0][0]) == 2, 'Only 2D points are␣
↪allowed.'

                self.fig = plt.figure()
                self.fig.set_dpi(100)
                self.fig.set_size_inches(7, 6.5)
                self.fig.suptitle('Fuzzy min-max classifier')
                if filename == '':
                        filename = 'fuzzy_animation'
                self.filename = filename + '.mp4'
                self.box_history = box_history
                self.train_patterns = train_patterns
                self.classes = classes
                self.verbose = verbose

                self.frames = np.ravel(np.array([[i]*frame_rate for i in␣
↪range(len(box_history))]))
                self.total = len(box_history)

                self.ax = plt.axes(xlim=(0, 1), ylim=(0, 1))
                self.ax.set_title('  = {} and   = {}'.format(exp_bound,␣
↪sensitivity))
                self.rectangles = []
                self.scatters = []
                self.colormap = [np.array([255, 0, 0]), np.array([0, 0, 255])]␣
↪+ [self.__get_random_color(color) for i in range(len(np.unique(classes)) -␣
↪2)]

                for i in range((len(train_patterns))):
                        x, y = train_patterns[i]
                        y = int(y)
                        if y == 0:
                                self.scatters.append(plt.scatter(-1, -1,␣
↪c=tuple(self.colormap[y] / 255)))

                        else:
                                self.scatters.append(plt.scatter(-1, -1,␣
↪c=tuple(self.colormap[y] / 255)))

                for _class in classes:
                        if _class == 0:
                                self.rectangles.append(plt.Rectangle((0, 0), 0,␣
↪0, fill=False, color='r'))
```

```
                else:
                    self.rectangles.append(plt.Rectangle((0, 0), 0,␣
↪0, fill=False, color='b'))

            if self.verbose:
                print('{:<20}: {:<10}'.format('Total Boxes', len(self.
↪rectangles)))
                print('{:<20}: {:<10}'.format('Points to plot',␣
↪len(self.scatters)))


    def __get_random_color(self):
        r = lambda: random.randint(0,255)
        return np.array([r(), r(), r()])


    def box_to_rect(self, box):
        vj, wj = box
        height = wj[1] - vj[1]
        width = wj[0] - vj[0]
        return tuple(vj), width, height


    def init(self):
        for i in self.rectangles:
            self.ax.add_patch(i)

        return tuple(self.rectangles) + tuple(self.scatters)


    def _animate(self, i):
        hyperboxes = self.box_history[i]
        # Plot training point
        x, y = self.train_patterns[i]
        self.scatters[i].set_offsets(tuple(x))
        for box in range(len(hyperboxes)):
            base, width, height = self.box_to_rect(hyperboxes[box])
            self.rectangles[box].set_xy(base)
            if width == 0:
                width = 0.02
            if height == 0:
                height = 0.02

            self.rectangles[box].set_width(width)
            self.rectangles[box].set_height(height)

        if self.verbose:
```

```
                print('{:<20}: {}/{}'.format('Animating frame', i+1,␣
↪self.total), end='\r')

            return tuple(self.rectangles) + tuple(self.scatters)


    def animate(self):
        '''
        Main function to start animation
        '''

        anim = animation.FuncAnimation(self.fig, self._animate,
                                          init_func = self.
↪init,
                                          frames = self.frames,
                                          interval = 20,
                                          blit = True)

        anim.save(self.filename, fps=30,
                  extra_args=['-vcodec', 'h264',
                              '-pix_fmt',␣
↪'yuv420p'])

        if self.verbose:
            print('Animation complete! Video saved at {}'.format(os.
↪path.join(os.getcwd(), self.filename)))
```

[42]:
```
class FuzzyMMC:

    def __init__(self, sensitivity=1, exp_bound=1, animate=False):
        '''
        Constructor for FuzzyMMC class
        '''
        self.sensitivity = sensitivity
        self.hyperboxes = None
        self.isanimate = animate
        self.classes = np.array([])
        self.exp_bound = exp_bound

        if self.animate:
            self.box_history = []
            self.train_patterns = []


    def membership(self, pattern):
        '''
        Calculates membership values a pattern
```

```
            Returns an ndarray of membership values of all hyperboxes
            '''
            min_pts = self.hyperboxes[:, 0, :]
            max_pts = self.hyperboxes[:, 1, :]

            a = np.maximum(0, (1 - np.maximum(0, (self.sensitivity * np.
↪minimum(1, pattern - max_pts)))))
            b = np.maximum(0, (1 - np.maximum(0, (self.sensitivity * np.
↪minimum(1, min_pts - pattern)))))

            return np.sum(a + b, axis=1) / (2 * len(pattern))


        def overlap_contract(self, index):
            '''
            Check if any classwise dissimilar hyperboxes overlap
            '''
            contracted = False
            for test_box in range(len(self.hyperboxes)):

                    if self.classes[test_box] == self.classes[index]:
                            # Ignore same class hyperbox overlap
                            continue

                    expanded_box = self.hyperboxes[index]
                    box = self.hyperboxes[test_box]

                    ## TODO: Refactor for vectorization
                    vj, wj = expanded_box
                    vk, wk = box

                    delta_new = delta_old = 1
                    min_overlap_index = -1
                    for i in range(len(vj)):
                            if vj[i] < vk[i] < wj[i] < wk[i]:
                                    delta_new = min(delta_old, wj[i] -␣
↪vk[i])

                            elif vk[i] < vj[i] < wk[i] < wj[i]:
                                    delta_new = min(delta_old, wk[i] -␣
↪vj[i])

                            elif vj[i] < vk[i] < wk[i] < wj[i]:
                                    delta_new = min(delta_old, min(wj[i] -␣
↪vk[i], wk[i] - vj[i]))
```

```
                    elif vk[i] < vj[i] < wj[i] < wk[i]:
                        delta_new = min(delta_old, min(wj[i] -
↪vk[i], wk[i] - vj[i]))

                    if delta_old - delta_new > 0:
                        min_overlap_index = i
                        delta_old = delta_new

                if min_overlap_index >= 0:
                    i = min_overlap_index
                    # We need to contract the expanded box
                    if vj[i] < vk[i] < wj[i] < wk[i]:
                        vk[i] = wj[i] = (vk[i] + wj[i])/2

                    elif vk[i] < vj[i] < wk[i] < wj[i]:
                        vj[i] = wk[i] = (vj[i] + wk[i])/2

                    elif vj[i] < vk[i] < wk[i] < wj[i]:
                        if (wj[i] - vk[i]) > (wk[i] - vj[i]):
                            vj[i] = wk[i]

                        else:
                            wj[i] = vk[i]

                    elif vk[i] < vj[i] < wj[i] < wk[i]:
                        if (wk[i] - vj[i]) > (wj[i] - vk[i]):
                            vk[i] = wj[i]

                        else:
                            wk[i] = vj[i]

                    self.hyperboxes[test_box] = np.array([vk, wk])
                    self.hyperboxes[index] = np.array([vj, wj])
                    contracted = True

        return contracted



    def train_pattern(self, X, Y):
        '''
        Main function that trains a fuzzy min max classifier
        Note:
        Y is a one-hot encoded target variable
        '''
        target = Y
```

```
                if target not in self.classes:

                        # Create a new hyberbox
                        if self.hyperboxes is not None:
                                self.hyperboxes = np.vstack((self.hyperboxes,␣
↪np.array([[X, X]])))
                                self.classes = np.hstack((self.classes, np.
↪array([target])))

                        else:
                                self.hyperboxes = np.array([[X, X]])
                                self.classes = np.array([target])

                        if self.isanimate:
                                self.box_history.append(np.copy(self.
↪hyperboxes))
                                self.train_patterns.append((X, Y))
                else:

                        memberships = self.membership(X)
                        memberships[np.where(self.classes != target)] = 0
                        memberships = sorted(list(enumerate(memberships)),␣
↪key=lambda x: x[1], reverse=True)

                        # Expand the most suitable hyperbox
                        count = 0
                        while True:
                                index = memberships[count][0]
                                min_new = np.minimum(self.hyperboxes[index, 0, :
↪], X)
                                max_new = np.maximum(self.hyperboxes[index, 1, :
↪], X)

                                if self.exp_bound * len(np.unique(self.
↪classes)) >= np.sum(max_new - min_new):
                                        self.hyperboxes[index, 0] = min_new
                                        self.hyperboxes[index, 1] = max_new
                                        break
                                else:
                                        count += 1

                                if count == len(memberships):
                                        self.hyperboxes = np.vstack((self.
↪hyperboxes, np.array([[X, X]])))
                                        self.classes = np.hstack((self.classes,␣
↪np.array([target])))
```

```
                                        index = len(self.hyperboxes) - 1
                                        break

                        # Overlap test
                        if self.isanimate:
                                self.box_history.append(np.copy(self.
↪hyperboxes))
                                self.train_patterns.append((X, Y))

                        contracted = self.overlap_contract(index)

                        if self.isanimate and contracted:
                                self.box_history.append(np.copy(self.
↪hyperboxes))
                                self.train_patterns.append((X, Y))


        def fit(self, X, Y):
                '''
                Wrapper for train_pattern
                '''
                for x, y in zip(X, Y):
                        self.train_pattern(x, y)


        def predict(self, X):
                '''
                Predict the class of the pattern X
                '''
                classes = np.unique(self.classes)
                results = []
                memberships = self.membership(X)
                max_prediction = 0
                pred_class = 0
                for _class in classes:
                        mask = np.zeros((len(self.hyperboxes),))
                        mask[np.where(self.classes == _class)] = 1
                        p = memberships * mask
                        prediction, class_index = np.max(p), np.argmax(p)
                        if prediction > max_prediction:
                                max_prediction = prediction
                                pred_class = class_index

                return max_prediction, self.classes[pred_class]


        def score(self, X, Y):
```

```
        '''
        Scores the classifier
        '''
        count = 0
        for x, y in zip(X, Y):
                _, pred = self.predict(x)
                if y == pred:
                        count += 1

        return count / len(Y)


def animate(self, frame_rate=10, filename='', verbose=True):
        '''
        To make a video of the classifier training.
        NOTE: Only possible when working with 2 dimensional patterns
        '''
        if self.isanimate:
                animator = Animator(box_history=self.box_history,
                                                        train_patterns=self.
↪train_patterns,
                                                        classes=self.
↪classes,
                                                        frame_rate=frame_rate,
                                                        exp_bound=self.
↪exp_bound,
                                                        sensitivity=self.
↪sensitivity,
                                                        filename=filename,
                                                        verbose=verbose)

                animator.animate()

                return animator.filename

        else:
                raise Exception('No animation data was collected!␣
↪Create a fuzzy classifier instance with animate=True')
```

Fuzzy Classifier

[43]:
```
clf1 = FuzzyMMC(sensitivity=1, exp_bound=0.1, animate=True)
clf2 = FuzzyMMC(sensitivity=1, exp_bound=0.1, animate=True)
clf3 = FuzzyMMC(sensitivity=1, exp_bound=0.1, animate=True)
clf_all = FuzzyMMC(sensitivity=1, exp_bound=0.1, animate=True)
```

[44]:
```
clf1.fit(X_train_1, y_train_1)
clf2.fit(X_train_2, y_train_2)
clf3.fit(X_train_3, y_train_3)
clf_all.fit(X_train_all, y_train_all)
```

[45]:
```
clf1.score(X_test_1, y_test_1)
```

[45]: 1.0

[46]:
```
clf2.score(X_test_2, y_test_2)
```

[46]: 0.9333333333333333

[47]:
```
clf3.score(X_test_3, y_test_3)
```

[47]: 1.0

[48]:
```
clf_all.score(X_test_all, y_test_all)
```

[48]: 0.9555555555555556

2.0.5 Summary

- Interestingly, the classifier performs best when using 2 class labels. Iris-setosa & Iris-versicolor and Iris-setosa & Iris-virginica
- Using all the class labels did not perform as well.
- I was not able to get the animations working as I got an error.

3 Etivity-3 Task 3 Fuzzy C-means clustering

3.0.1 Introduction

- Using el notebook RepMLA_3_3.ipynb as a baseline, solve the following task
- Design 3 clustering problems using 500 data points and use the fuzzy partition coefficient (FPC) from 2 to 15 clusters.
- These are the problems
  - Clustering problem 1 with 4 clusters
  - Clustering problem 2 with 6 clusters
  - Clustering problem 3 with 8 clusters

3.0.2 Dataset

- The dataset is created by setting the position of the cluster centres.
- The datapoints are generated in clusters around the centres.

3.0.3 Method

- Fuzzy logic is used to cluster multi-dimensional data. This gives each point a percentage membership value to each cluster centre.
- This is done with skfuzzy.cmeans

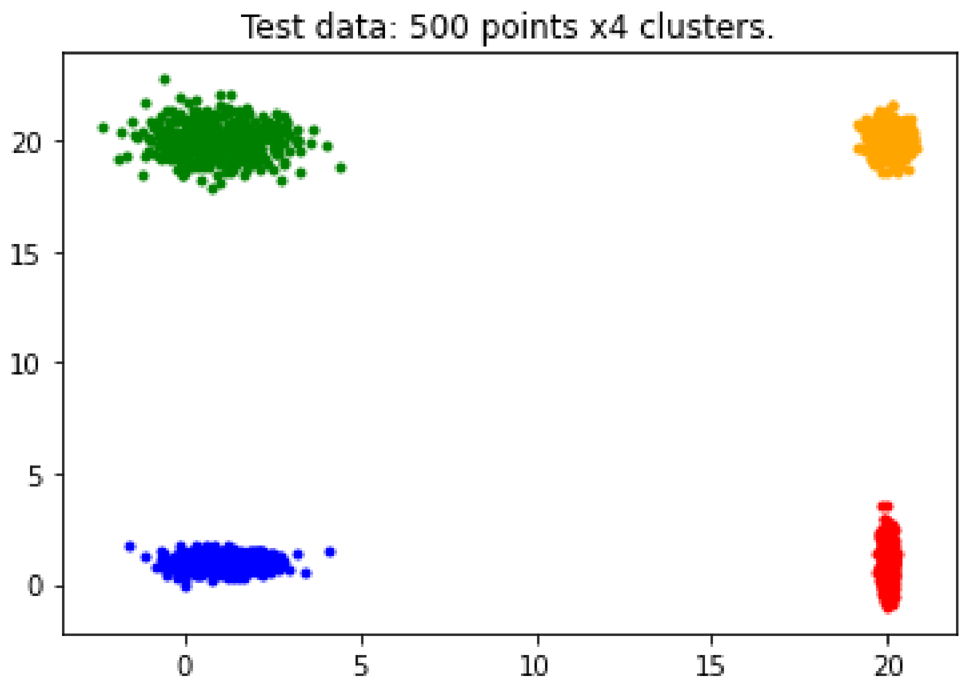
[49]:
```
#!pip install scikit-fuzzy
from __future__ import division, print_function
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz
```

[50]:
```
colors = ['b', 'orange', 'g', 'r', 'c', 'm', 'y', 'k', 'Brown', 'ForestGreen']

# Define the cluster centers
centers = [[1, 1],
           [20, 20],
           [1, 20],
           [20, 1],
           [10, 15],
           [10, 5],
           [15, 10],
           [4, 10]]

# Define three cluster sigmas in x and y, respectively
sigmas = [[0.8, 0.3],
          [0.3, 0.5],
          [1.1, 0.7],
          [0.1, 0.8],
          [0.6, 0.7],
          [2.0, 0.5],
          [0.3, 1.3],
          [1.5, 0.9]]
```

[51]:
```
# Set number of clusters
n = 4

# Generate test data
np.random.seed(42)  # Set seed for reproducibility
xpts_1 = np.zeros(1)
ypts_1 = np.zeros(1)
labels_1 = np.zeros(1)
for i, ((xmu, ymu), (xsigma, ysigma)) in enumerate(zip(centers[:n], sigmas[:
↪n])):
    xpts_1 = np.hstack((xpts_1, np.random.standard_normal(500) * xsigma + xmu))
    ypts_1 = np.hstack((ypts_1, np.random.standard_normal(500) * ysigma + ymu))
    labels_1 = np.hstack((labels_1, np.ones(500) * i))
```

```python
# Visualize the test data
fig0, ax0 = plt.subplots()
for label in range(n):
    ax0.plot(xpts_1[labels_1 == label], ypts_1[labels_1 == label], '.',
             color=colors[label])
ax0.set_title('Test data: 500 points x4 clusters.')
```

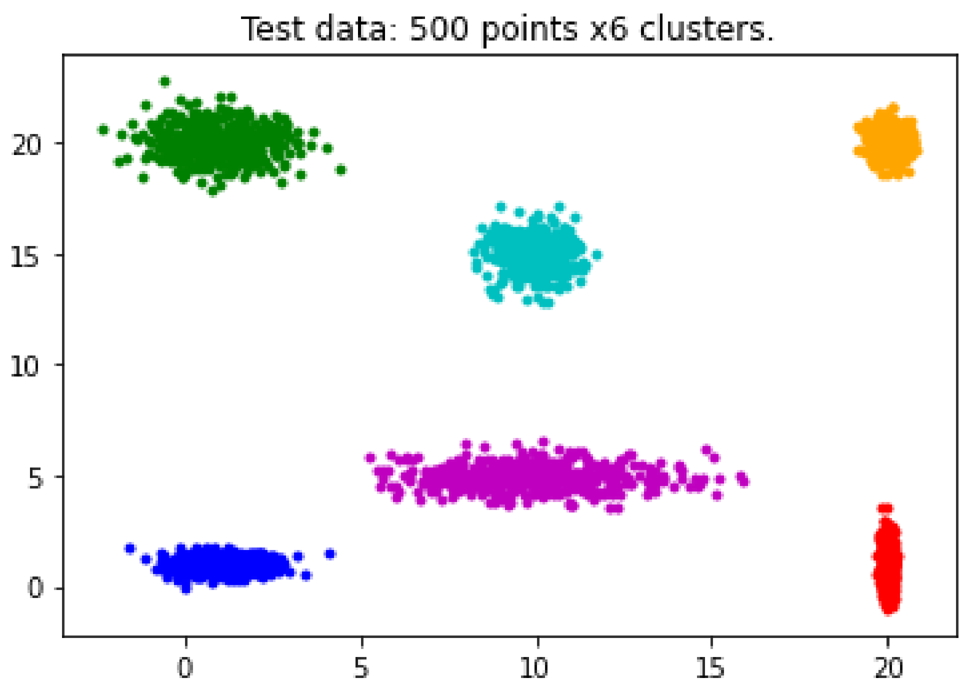
[51]: Text(0.5, 1.0, 'Test data: 500 points x4 clusters.')



[52]:
```python
# Set number of clusters
n = 6

# Generate test data
np.random.seed(42)  # Set seed for reproducibility
xpts_2 = np.zeros(1)
ypts_2 = np.zeros(1)
labels_2 = np.zeros(1)
for i, ((xmu, ymu), (xsigma, ysigma)) in enumerate(zip(centers[:n], sigmas[:
 ↪n])):
    xpts_2 = np.hstack((xpts_2, np.random.standard_normal(500) * xsigma + xmu))
    ypts_2 = np.hstack((ypts_2, np.random.standard_normal(500) * ysigma + ymu))
    labels_2 = np.hstack((labels_2, np.ones(500) * i))

# Visualize the test data
```

```
fig0, ax0 = plt.subplots()
for label in range(n):
    ax0.plot(xpts_2[labels_2 == label], ypts_2[labels_2 == label], '.',
             color=colors[label])
ax0.set_title('Test data: 500 points x6 clusters.')
```

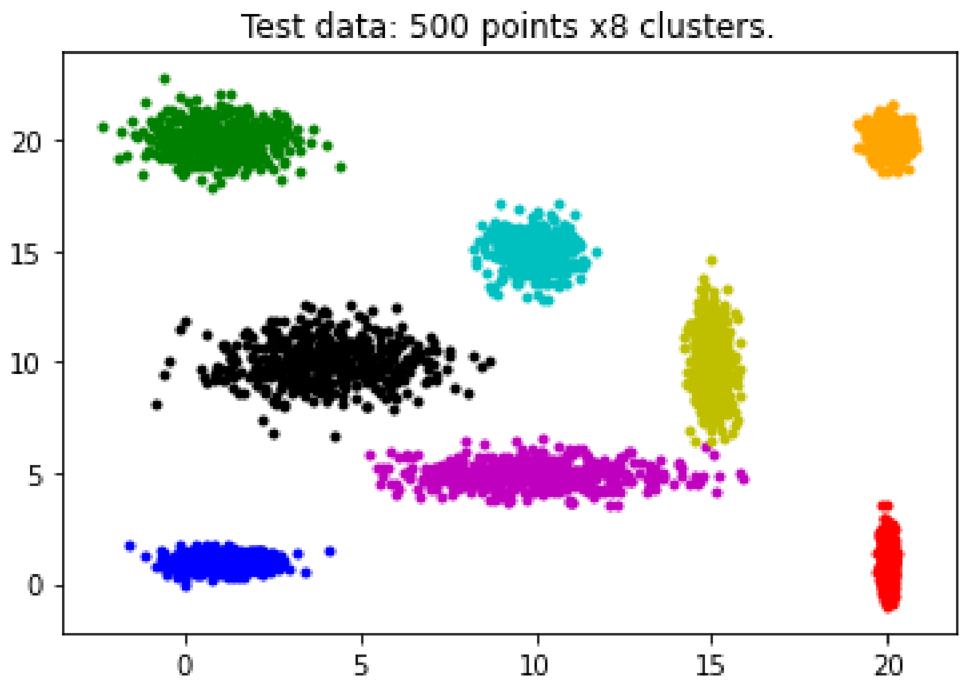
[52]: Text(0.5, 1.0, 'Test data: 500 points x6 clusters.')



[53]:
```
# Set number of clusters
n = 8

# Generate test data
np.random.seed(42)  # Set seed for reproducibility
xpts_3 = np.zeros(1)
ypts_3 = np.zeros(1)
labels_3 = np.zeros(1)
for i, ((xmu, ymu), (xsigma, ysigma)) in enumerate(zip(centers[:n], sigmas[:
↪n])):
    xpts_3 = np.hstack((xpts_3, np.random.standard_normal(500) * xsigma + xmu))
    ypts_3 = np.hstack((ypts_3, np.random.standard_normal(500) * ysigma + ymu))
    labels_3 = np.hstack((labels_3, np.ones(500) * i))

# Visualize the test data
fig0, ax0 = plt.subplots()
```

```
for label in range(n):
    ax0.plot(xpts_3[labels_3 == label], ypts_3[labels_3 == label], '.',
             color=colors[label])
ax0.set_title('Test data: 500 points x8 clusters.')
```

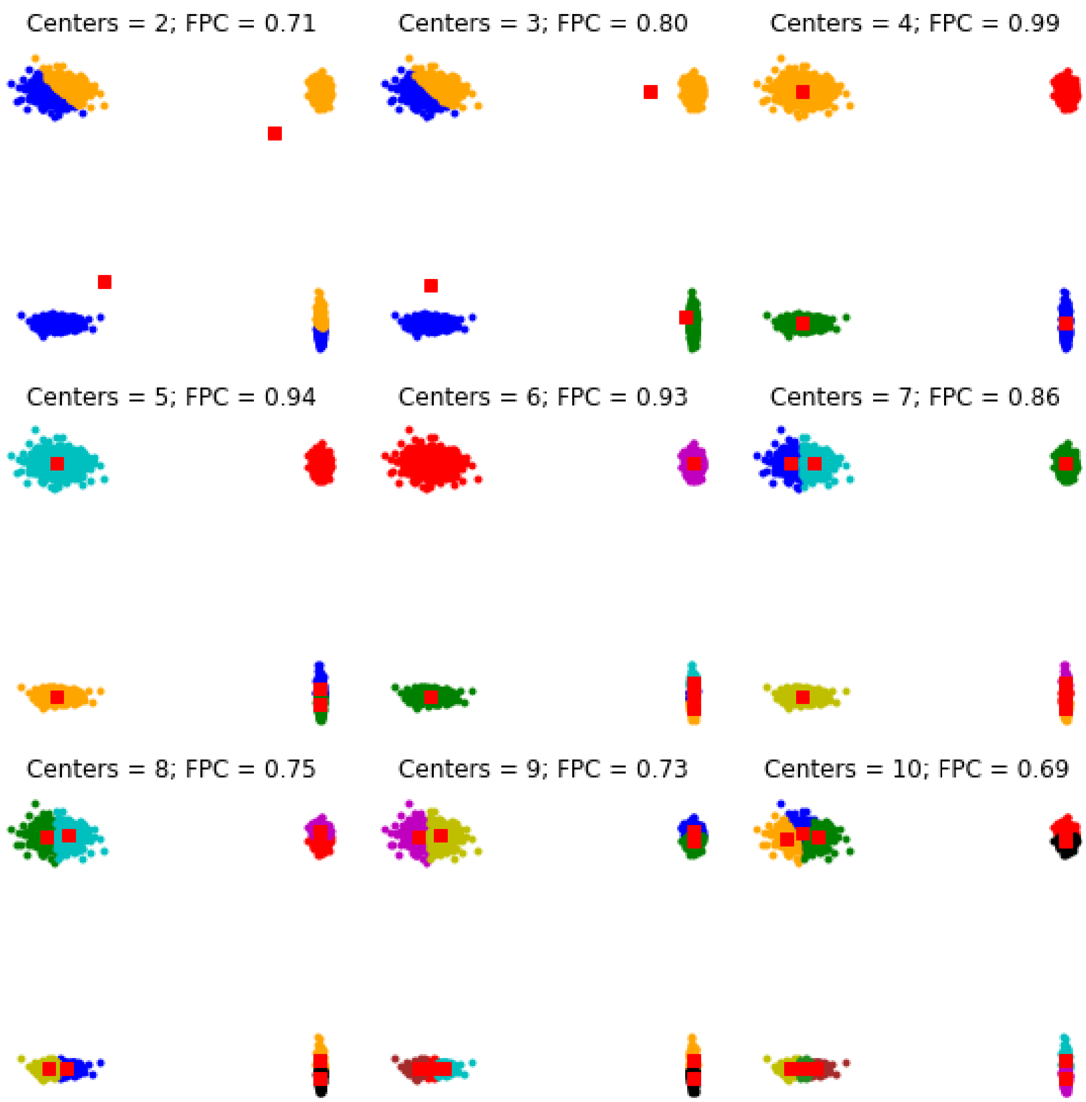
[53]: Text(0.5, 1.0, 'Test data: 500 points x8 clusters.')



3.0.4 Clustering

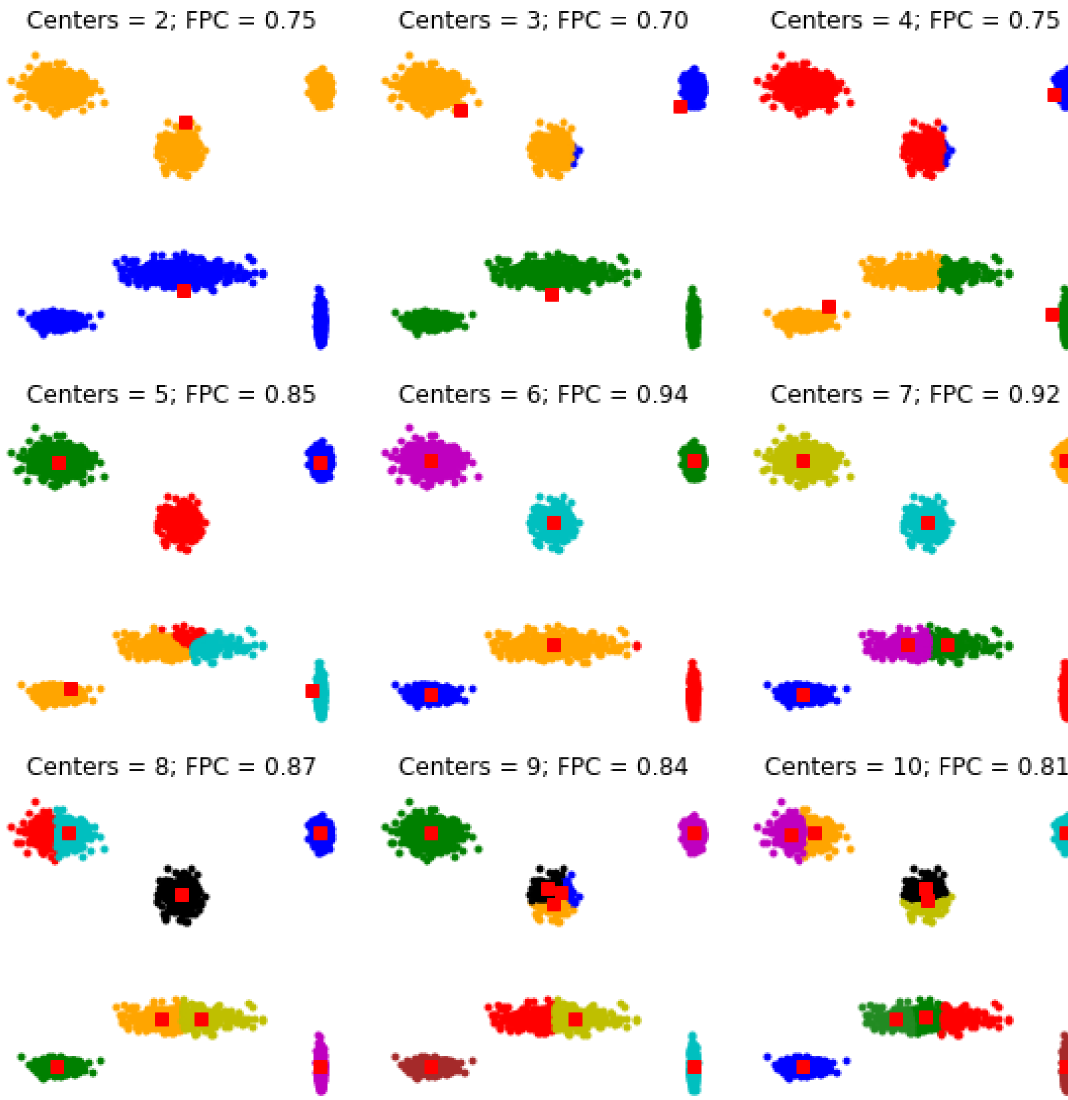
[54]:
```
# Set up the loop and plot
fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))
alldata_1 = np.vstack((xpts_1, ypts_1))
fpcs_1 = []
for ncenters, ax in enumerate(axes1.reshape(-1), 2):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        alldata_1, ncenters, 2, error=0.005, maxiter=1000, init=None)

    # Store fpc values for later
    fpcs_1.append(fpc)

    # Plot assigned clusters, for each data point in training set
    cluster_membership = np.argmax(u, axis=0)
    for j in range(ncenters):
        ax.plot(xpts_1[cluster_membership == j],
```

```
                ypts_1[cluster_membership == j], '.', color=colors[j])

    # Mark the center of each fuzzy cluster
    for pt in cntr:
        ax.plot(pt[0], pt[1], 'rs')

    ax.set_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))
    ax.axis('off')

fig1.tight_layout()
```



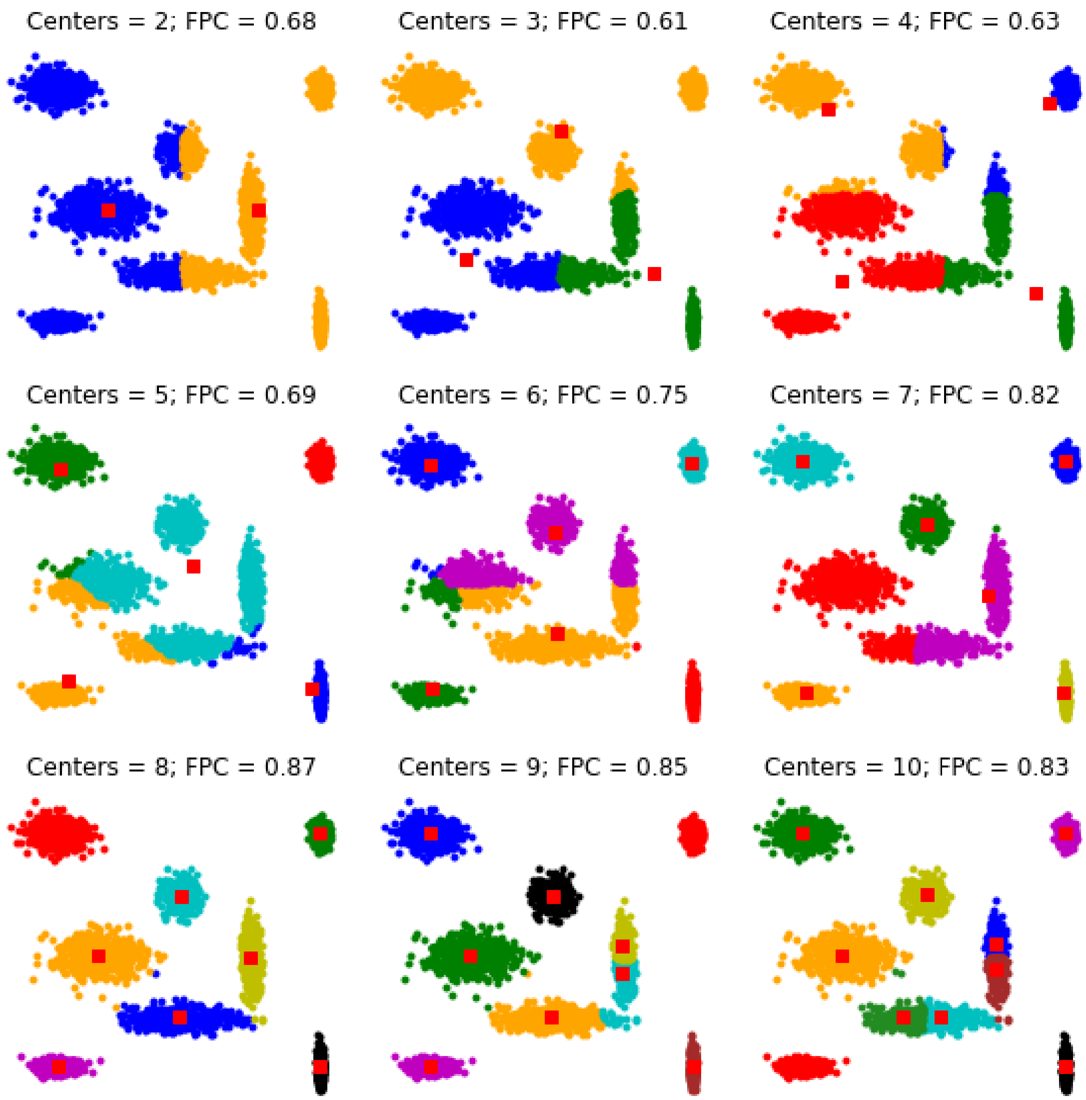
[55]:

```
# Set up the loop and plot
fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))
alldata_2 = np.vstack((xpts_2, ypts_2))
fpcs_2 = []

for ncenters, ax in enumerate(axes1.reshape(-1), 2):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        alldata_2, ncenters, 2, error=0.005, maxiter=1000, init=None)

    # Store fpc values for later
    fpcs_2.append(fpc)

    # Plot assigned clusters, for each data point in training set
    cluster_membership = np.argmax(u, axis=0)
    for j in range(ncenters):
        ax.plot(xpts_2[cluster_membership == j],
                ypts_2[cluster_membership == j], '.', color=colors[j])

    # Mark the center of each fuzzy cluster
    for pt in cntr:
        ax.plot(pt[0], pt[1], 'rs')

    ax.set_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))
    ax.axis('off')

fig1.tight_layout()
```



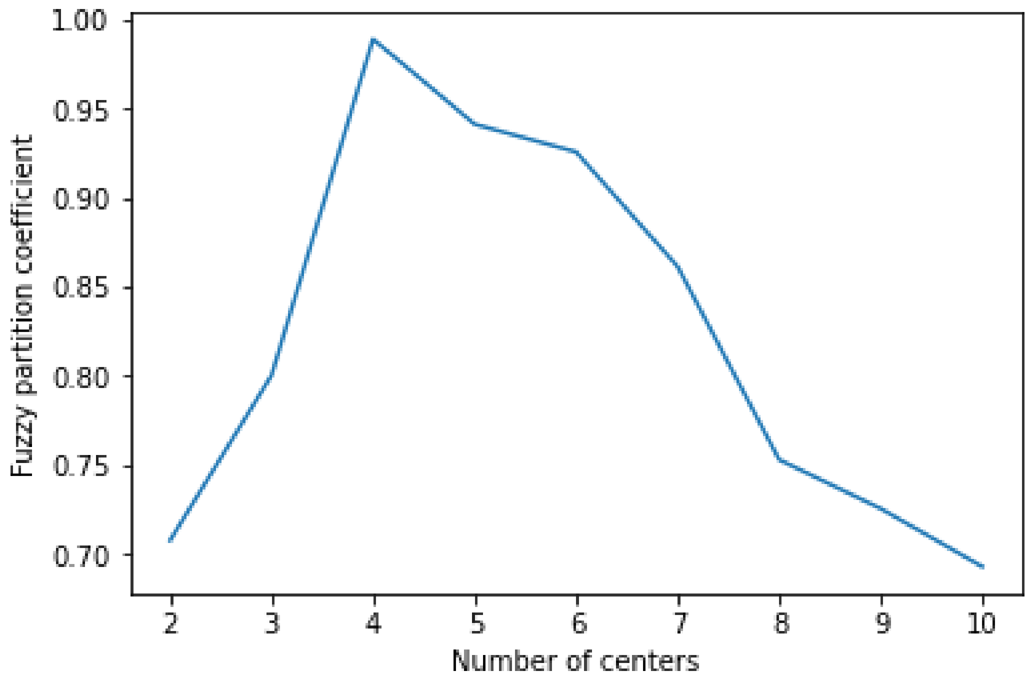
[56]:
```
# Set up the loop and plot
fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))
alldata_3 = np.vstack((xpts_3, ypts_3))
fpcs_3 = []

for ncenters, ax in enumerate(axes1.reshape(-1), 2):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        alldata_3, ncenters, 2, error=0.005, maxiter=1000, init=None)

    # Store fpc values for later
    fpcs_3.append(fpc)

    # Plot assigned clusters, for each data point in training set
```

```
    cluster_membership = np.argmax(u, axis=0)
    for j in range(ncenters):
        ax.plot(xpts_3[cluster_membership == j],
                ypts_3[cluster_membership == j], '.', color=colors[j])

    # Mark the center of each fuzzy cluster
    for pt in cntr:
        ax.plot(pt[0], pt[1], 'rs')

    ax.set_title('Centers = {0}; FPC = {1:.2f}'.format(ncenters, fpc))
    ax.axis('off')

fig1.tight_layout()
```



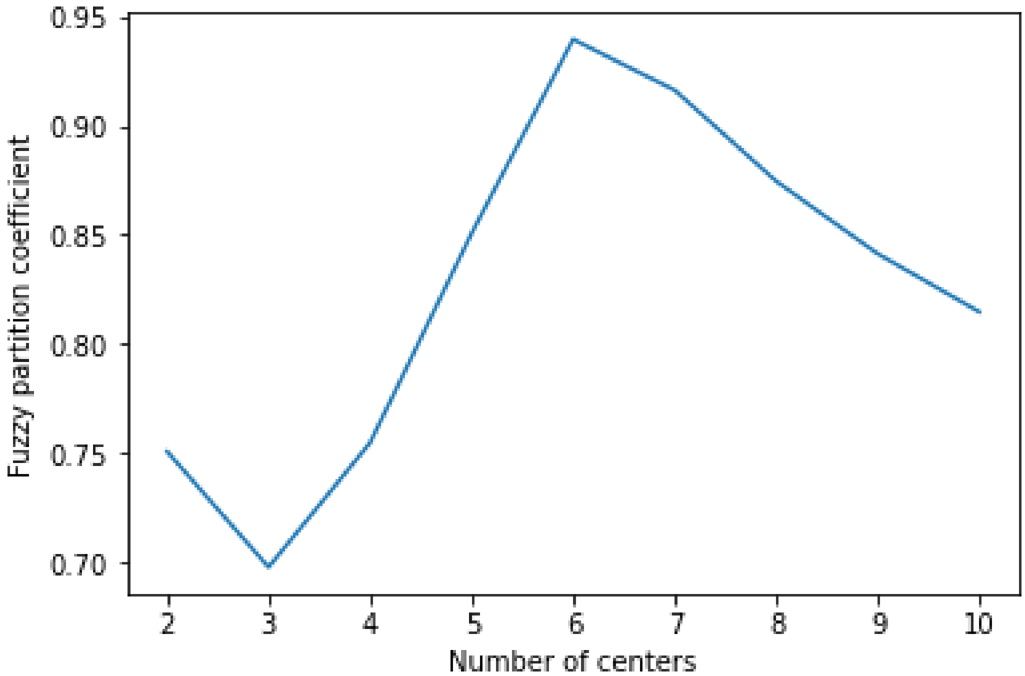
```
[57]: fig2, ax2 = plt.subplots()
ax2.plot(np.r_[2:11], fpcs_1)
ax2.set_xlabel("Number of centers")
ax2.set_ylabel("Fuzzy partition coefficient")
```

```
[57]: Text(0, 0.5, 'Fuzzy partition coefficient')
```



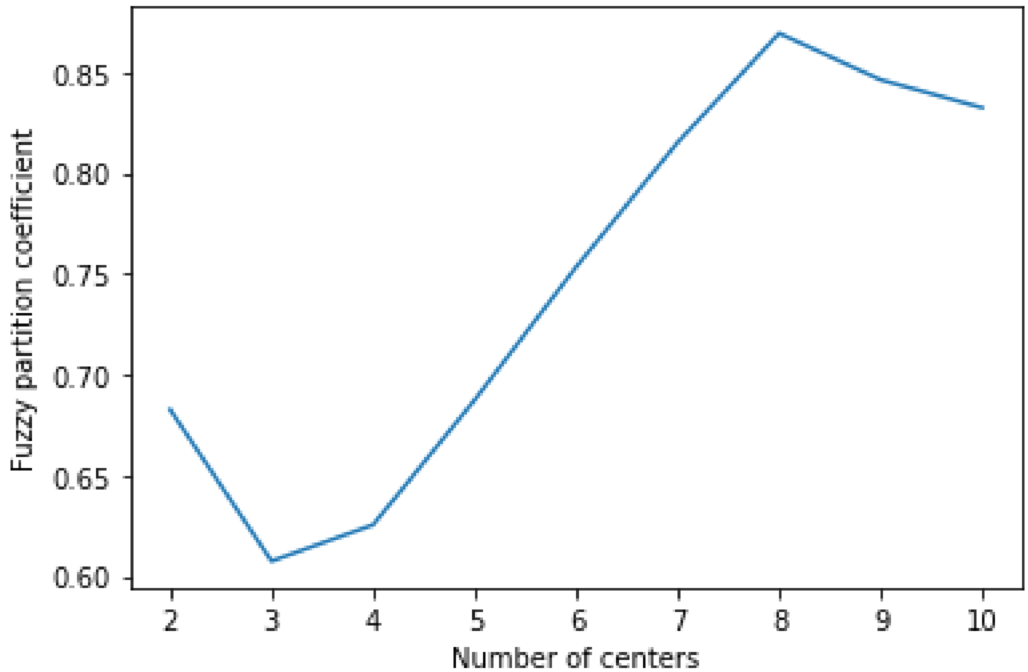

```
[58]: fig2, ax2 = plt.subplots()
ax2.plot(np.r_[2:11], fpcs_2)
ax2.set_xlabel("Number of centers")
ax2.set_ylabel("Fuzzy partition coefficient")
```

```
[58]: Text(0, 0.5, 'Fuzzy partition coefficient')
```



[59]:
```
fig2, ax2 = plt.subplots()
ax2.plot(np.r_[2:11], fpcs_3)
ax2.set_xlabel("Number of centers")
ax2.set_ylabel("Fuzzy partition coefficient")
```

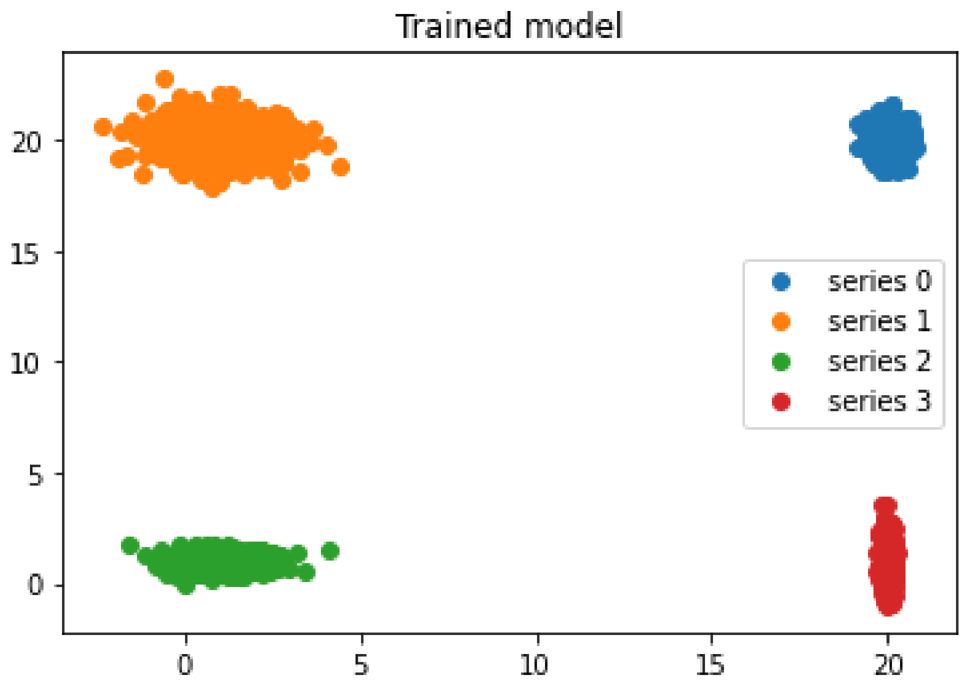
[59]: Text(0, 0.5, 'Fuzzy partition coefficient')



3.0.5 Building the models

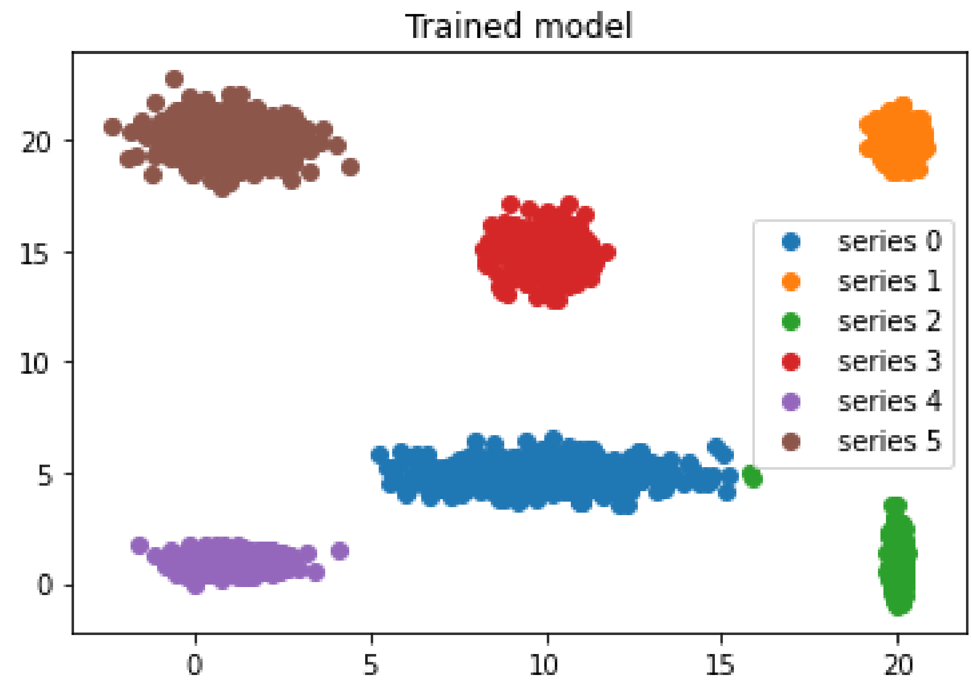
[60]:
```
# Regenerate fuzzy model with 3 cluster centers - note that center ordering
# is random in this clustering algorithm, so the centers may change places
cntr, u_orig, _, _, _, _, _ = fuzz.cluster.cmeans(
    alldata_1, 4, 2, error=0.005, maxiter=1000)

# Show 3-cluster model
fig2, ax2 = plt.subplots()
ax2.set_title('Trained model')
for j in range(4):
    ax2.plot(alldata_1[0, u_orig.argmax(axis=0) == j],
             alldata_1[1, u_orig.argmax(axis=0) == j], 'o',
             label='series ' + str(j))
ax2.legend()
```

[60]: <matplotlib.legend.Legend at 0x208c30560a0>



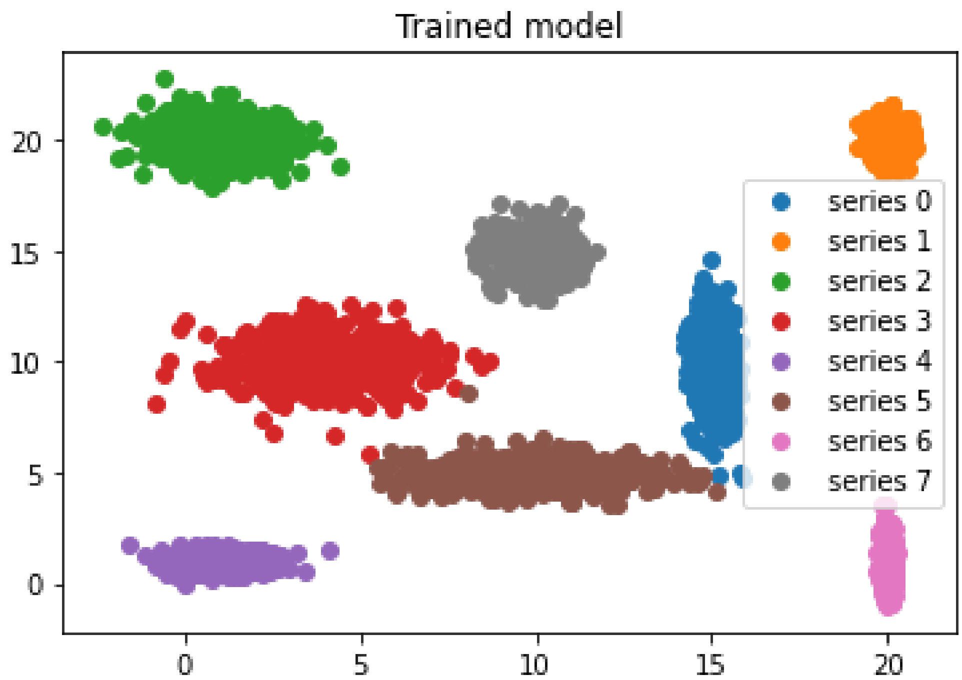
[61]:
```
# Regenerate fuzzy model with 3 cluster centers - note that center ordering
# is random in this clustering algorithm, so the centers may change places
cntr, u_orig, _, _, _, _, _ = fuzz.cluster.cmeans(
    alldata_2, 6, 2, error=0.005, maxiter=1000)

# Show 3-cluster model
fig2, ax2 = plt.subplots()
ax2.set_title('Trained model')
for j in range(6):
    ax2.plot(alldata_2[0, u_orig.argmax(axis=0) == j],
             alldata_2[1, u_orig.argmax(axis=0) == j], 'o',
             label='series ' + str(j))
ax2.legend()
```

[61]: <matplotlib.legend.Legend at 0x208c4559eb0>

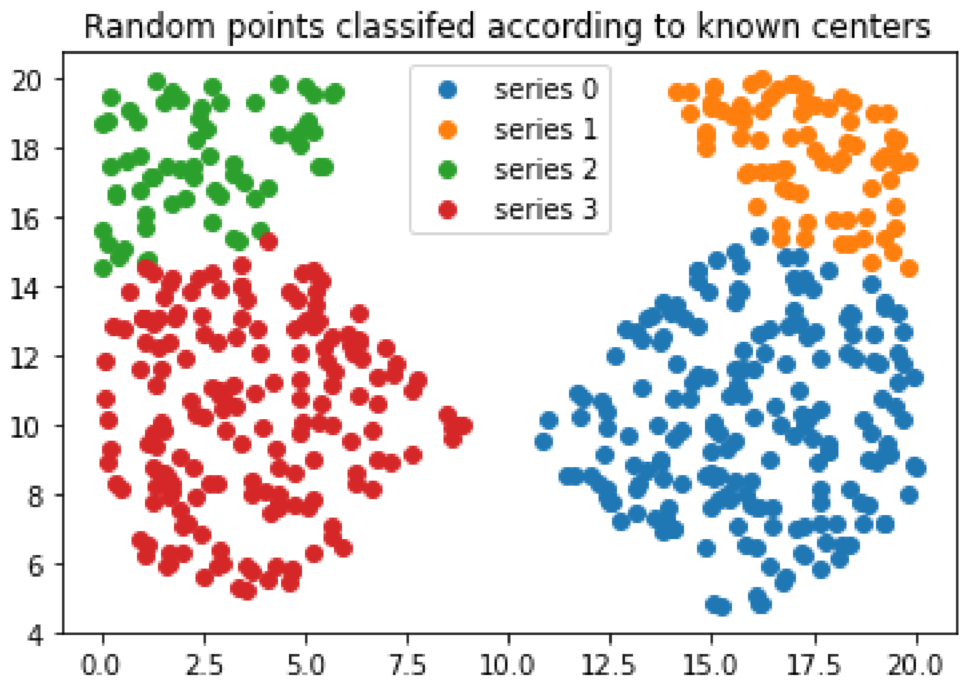


[62]:
```
# Regenerate fuzzy model with 3 cluster centers - note that center ordering
# is random in this clustering algorithm, so the centers may change places
cntr, u_orig, _, _, _, _, _ = fuzz.cluster.cmeans(
    alldata_3, 8, 2, error=0.005, maxiter=1000)

# Show 3-cluster model
fig2, ax2 = plt.subplots()
ax2.set_title('Trained model')
for j in range(8):
    ax2.plot(alldata_3[0, u_orig.argmax(axis=0) == j],
             alldata_3[1, u_orig.argmax(axis=0) == j], 'o',
             label='series ' + str(j))
ax2.legend()
```

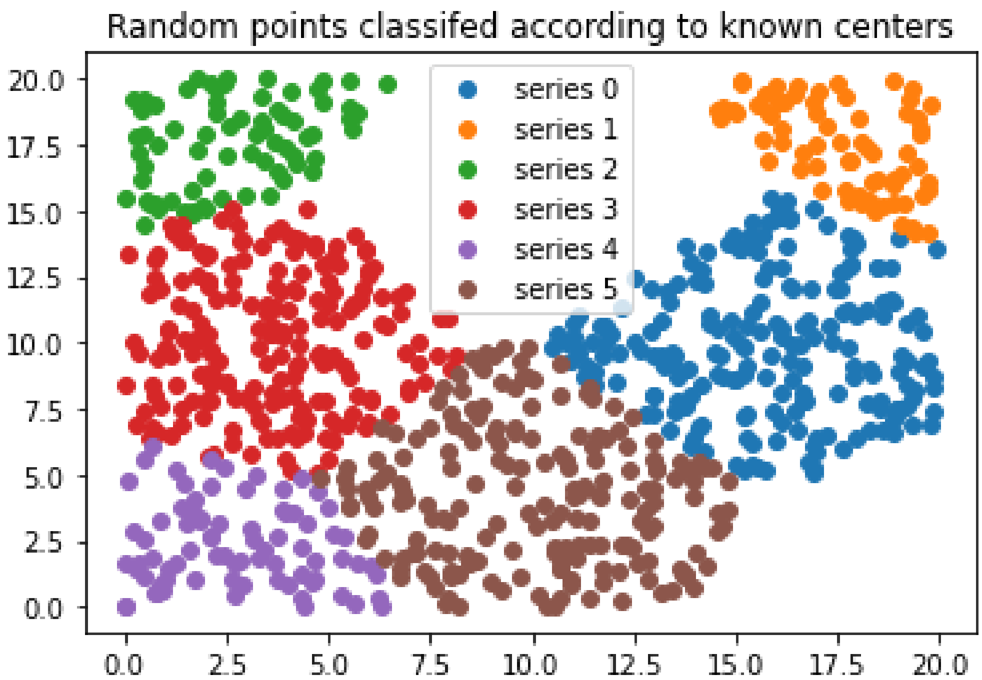
[62]: <matplotlib.legend.Legend at 0x208c2ba7f70>



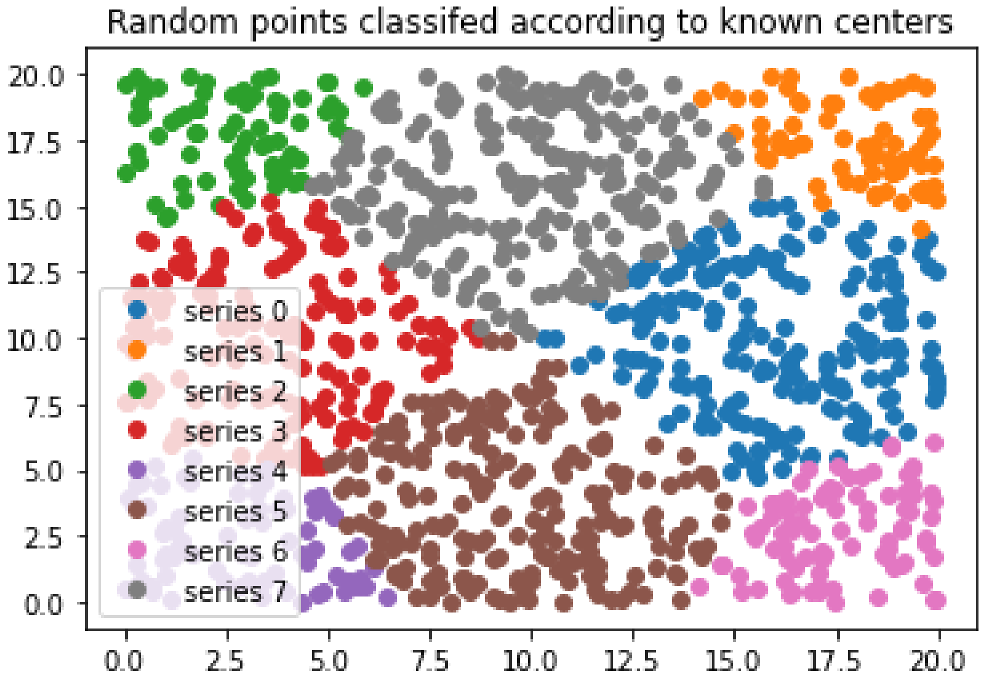
[63]:
```
# Generate uniformly sampled data spread across the range [0, 20] in x and y
newdata_1 = np.random.uniform(0, 1, (1100, 2)) * 20

# Predict new cluster membership with `cmeans_predict` as well as
# `cntr` from the 3-cluster model
u, u0, d, jm, p, fpc = fuzz.cluster.cmeans_predict(
    newdata_1.T, cntr, 2, error=0.005, maxiter=1000)

# Plot the classified uniform data. Note for visualization the maximum
# membership value has been taken at each point (i.e. these are hardened,
# not fuzzy results visualized) but the full fuzzy result is the output
# from cmeans_predict.
cluster_membership = np.argmax(u, axis=0)  # Hardening for visualization

fig3, ax3 = plt.subplots()
ax3.set_title('Random points classifed according to known centers')
for j in range(4):
    ax3.plot(newdata_1[cluster_membership == j, 0],
             newdata_1[cluster_membership == j, 1], 'o',
             label='series ' + str(j))
ax3.legend()

plt.show()
```



[64]:
```
# Generate uniformly sampled data spread across the range [0, 20] in x and y
newdata_2 = np.random.uniform(0, 1, (1100, 2)) * 20

# Predict new cluster membership with `cmeans_predict` as well as
# `cntr` from the 3-cluster model
u, u0, d, jm, p, fpc = fuzz.cluster.cmeans_predict(
    newdata_2.T, cntr, 2, error=0.005, maxiter=1000)

# Plot the classified uniform data. Note for visualization the maximum
# membership value has been taken at each point (i.e. these are hardened,
# not fuzzy results visualized) but the full fuzzy result is the output
# from cmeans_predict.
cluster_membership = np.argmax(u, axis=0)  # Hardening for visualization

fig3, ax3 = plt.subplots()
ax3.set_title('Random points classifed according to known centers')
for j in range(6):
    ax3.plot(newdata_2[cluster_membership == j, 0],
             newdata_2[cluster_membership == j, 1], 'o',
             label='series ' + str(j))
ax3.legend()

plt.show()
```



[65]:
```
# Generate uniformly sampled data spread across the range [0, 20] in x and y
newdata_3 = np.random.uniform(0, 1, (1100, 2)) * 20

# Predict new cluster membership with `cmeans_predict` as well as
# `cntr` from the 3-cluster model
u, u0, d, jm, p, fpc = fuzz.cluster.cmeans_predict(
    newdata_3.T, cntr, 2, error=0.005, maxiter=1000)

# Plot the classified uniform data. Note for visualization the maximum
# membership value has been taken at each point (i.e. these are hardened,
# not fuzzy results visualized) but the full fuzzy result is the output
# from cmeans_predict.
cluster_membership = np.argmax(u, axis=0)  # Hardening for visualization

fig3, ax3 = plt.subplots()
ax3.set_title('Random points classifed according to known centers')
for j in range(8):
    ax3.plot(newdata_3[cluster_membership == j, 0],
             newdata_3[cluster_membership == j, 1], 'o',
             label='series ' + str(j))
ax3.legend()

plt.show()
```



3.1 Summary

- The classifier was very accurate in determining the individual clusters.
- 1. was easy as the cluster were isolated in each corner, but others wer more difficult