machine-learning-laboratory-1

June 15, 2023

1 1)a)demonstrate the FIND-S algorithm for finding the most specifichypothesis based on a given set of training data samples. Read the training data from a.CSV file and show the output for test cases.

[1]:

```python
import numpy as np
import pandas as pd
data = pd.read_csv('finds.csv')
def train(concepts, target):
    for i,val in enumerate(target):
        if val == "Yes":
            specific_h = concepts[i]
        break

    for i,h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] == specific_h[x]:
                    pass
            else:
                specific_h[x] = "?"
                return specific_h
#slicing rows and column, : means begining to end of row
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print(train(concepts,target))
```

```
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' '?']
```

2 1)b)Develop an interactive program byCompareing the result by implementing LIST THEN ELIMINATE algorithm

[3]:

```python
import numpy as np
import pandas as pd
data = pd.read_csv('finds.csv')
```

```
def list_then_eliminate(concepts, target):
    positive_examples = concepts[target == 'Yes']
    specific_hypothesis = positive_examples[0].copy()

    for example in positive_examples[1:]:
        for i, attribute in enumerate(example):
            if attribute != specific_hypothesis[i]:
                specific_hypothesis[i] = '?'

                return specific_hypothesis

concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])

print('Specific hypothesis obtained by LIST THEN ELIMINATE algorithm:')
print(list_then_eliminate(concepts, target))
print('\nSpecific hypothesis obtained by the original code:')
print(train(concepts, target))
```

```
Specific hypothesis obtained by LIST THEN ELIMINATE algorithm:
['?' 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Specific hypothesis obtained by the original code:
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' '?']
```

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

[4]:

```
import numpy as np
import pandas as pd
#Loading Data from CSV file
data=pd.DataFrame(data=pd.read_csv("finds.csv"))
#Seperating concept features from Target
concepts=np.array(data.iloc[:,0:-1])
#Isolating target into seperate Dataframe
target=np.array(data.iloc[:,-1])
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))]
    #the learning iterations
    for i, h in enumerate(concepts):
        #checking if the hypothesys has a positive target
        if target[i]=="Yes":
            for x in range(len(specific_h)):
            #Change values in S&G only if values change
                if h[x]!=specific_h[x]:
                    specific_h[x]='?'
                    general_h[x][x]='?'
                #Checking if the hypothesys has a positive target
                if target[i]=="No":
                    for x in range (len(specific_h)):
                        #for negative hypothesys change the value only in G
                        if h[x]!=specific_h[x]:
                            general_h[x][x]='?'
                            #find indices where we have empty row, meaning those that are unchanged
                            indices=[i for i ,val in enumerate(general_h)if val==['?','?','?','?','?','?']]
                            for i in indices:
                                #remove those rows form general_h
                                general_h.remove(['?','?','?','?','?','?'])
                                #return final values
        return specific_h,general_h


s_final, g_final=learn(concepts,target)
print("Final S:",s_final,sep="\n")
print("Final G:",g_final,sep="\n")
data.head()
```

```
Final S:
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Final G:
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?', '?']]
```

```
[4]:    Eg    Sky Temperature Humidity    Wind Water Forecast Enjoyable
     0   1  Sunny        Warm   Normal  Strong  Warm     Same       Yes
     1   2  Sunny        Warm     High  Strong  Warm     Same       Yes
     2   3  Rainy        Cold     High  Strong  Warm     Same       Yes
     3   4  Sunny        Warm     High  Strong  Warm     Same       Yes
```

4 3)Demonstrate Pre processing (Data Cleaning, Integration and Transformation) activity on suitable data: For example: Identify and Delete Rows that Contain Duplicate Data by considering an appropriate dataset. Identify and Delete Columns That Contain a Single Value by considering an appropriate dataset.

[5]:

```python
import pandas as pd

# Sample dataset 1: contains duplicate rows
data1 = {'A': [1, 2, 2, 3, 4, 5, 5],
         'B': [5, 6, 6, 7, 8, 9, 9]}

# Sample dataset 2: contains a column with a single value
data2 = {'C': [10, 10, 10, 10, 10],
         'D': [11, 12, 13, 14, 15]}

# Creating dataframes from the datasets
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Identifying and deleting duplicate rows in df1
print("Before removing duplicates in dataset 1:")
print(df1)
df1.drop_duplicates(inplace=True)
print("\nAfter removing duplicates in dataset 1:")
print(df1)

# Identifying and deleting columns with a single value in df2
print("\nBefore removing single value columns in dataset 2:")
print(df2)
cols_to_remove = [col for col in df2.columns if df2[col].nunique() <= 1]
df2.drop(cols_to_remove, axis=1, inplace=True)
print("\nAfter removing single value columns in dataset 2:")
print(df2)
```

```
Before removing duplicates in dataset 1:
   A  B
0  1  5
1  2  6
2  2  6
3  3  7
4  4  8
5  5  9
6  5  9

After removing duplicates in dataset 1:
   A  B
0  1  5
1  2  6
3  3  7
4  4  8
5  5  9

Before removing single value columns in dataset 2:
    C   D
0  10  11
1  10  12
2  10  13
3  10  14
4  10  15

After removing single value columns in dataset 2:
    D
0  11
1  12
2  13
3  14
4  15
```

5 4)Demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge toclassify a new sample.

[7]:

```python
import csv
import math
import random

#Function tells which class has more entries in given data-set
def majorClass(attributes, data, target):
    freq = {}
    index = attributes.index(target)
    for tuple in data:
        if tuple[index] in freq:
            freq[tuple[index]] += 1
        else:
            freq[tuple[index]] = 1
    max = 0
    major = ""
    for key in freq.keys():
        if freq[key]>max:
            max = freq[key]
            major = key
    return major

# Calculates the entropy of the data given the target attribute
def entropy(attributes, data, targetAttr):
    freq = {}
    dataEntropy = 0.0
    i = 0
    for entry in attributes:
        if (targetAttr == entry):
            break
        i = i + 1

    i = i - 1
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0
    for freq in freq.values():
        dataEntropy += (-freq/len(data)) * math.log(freq/len(data), 2)

    return dataEntropy

# Calculates the information gain (reduction in entropy) in the data when a particular attribute is chosen for splitting the data.
def info_gain(attributes, data, attr, targetAttr):
    freq = {}
    subsetEntropy = 0.0
    i = attributes.index(attr)
    for entry in data:
        if entry[i] in freq:
            freq[entry[i]] += 1.0
        else:
            freq[entry[i]] = 1.0

    for val in freq.keys():
        valProb = freq[val] / sum(freq.values())
        dataSubset = [entry for entry in data if entry[i] == val]
        subsetEntropy += valProb * entropy(attributes, dataSubset, targetAttr)

    return (entropy(attributes, data, targetAttr) - subsetEntropy)

# This function chooses the attribute among the remaining attributes which has the maximum information gain.
def attr_choose(data, attributes, target):
    best = attributes[0]
    maxGain = 0;
    for attr in attributes:
        newGain = info_gain(attributes, data, attr, target)
        if newGain>maxGain:
            maxGain = newGain
            best = attr

    return best

# This function will get unique values for that particular attribute from the given data
def get_values(data, attributes, attr):
    index = attributes.index(attr)
    values = []
    for entry in data:
        if entry[index] not in values:
            values.append(entry[index])

    return values

# This function will get all the rows of the data where the chosen "best" attribute has a value "val"
def get_data(data, attributes, best, val):
    new_data = [[]]
    index = attributes.index(best)
    for entry in data:
        if (entry[index] == val):
            newEntry = []
            for i in range(0,len(entry)):
                if(i != index):
                    newEntry.append(entry[i])
            new_data.append(newEntry)

    new_data.remove([])
    return new_data

def build_tree(data, attributes, target):
    data = data[:]
    vals = [record[attributes.index(target)] for record in data]
    default = majorClass(attributes, data, target)
    if not data or (len(attributes) - 1) <= 0:
        return default
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        best = attr_choose(data, attributes, target)
        tree = {best: {}}
        for val in get_values(data, attributes, best):
            new_data = get_data(data, attributes, best, val)
            newAttr = attributes[:]
            newAttr.remove(best)
            subtree = build_tree(new_data, newAttr, target)
            tree[best][val] = subtree
        return tree


def execute_decision_tree():
    data = []
    #load file
    with open("weather.csv") as tsv:
        for line in csv.reader(tsv):
            data.append(tuple(line))
    print("Number of records:", len(data))

    #set attributes
    attributes = ['outlook', 'temperature', 'humidity', 'wind', 'play']
    target = attributes[-1]

    #set training data
    acc = []
    training_set = [x for i, x in enumerate(data)]
    tree = build_tree(training_set, attributes, target)
    print(tree)

    #execute algorithm on test data
    results = []
    test_set = [('rainy', 'mild', 'high', 'strong')]
    for entry in test_set:
        tempDict = tree.copy()
        result = ""
        while isinstance(tempDict, dict):
            child = []
            nodeVal = next(iter(tempDict))
            child = tempDict[next(iter(tempDict))].keys()
            tempDict = tempDict[next(iter(tempDict))]
            index = attributes.index(nodeVal)
            value = entry[index]

            if value in tempDict.keys():
                result = tempDict[value]
                tempDict = tempDict[value]
            else:
                result = "Null"
                break
        if result != "Null":
            results.append(result == entry[-1])

        print(result)


if __name__ == "__main__":
    execute_decision_tree()
```

```
Number of records: 15
{'outlook': {'id': 'wind', '1': 'weak', '2': 'strong', '3': 'weak', '4': 'weak',
'5': 'weak', '6': 'weak', '7': 'strong', '8': 'weak', '9': 'weak', '10': 'weak',
'11': 'strong', '12': 'strong', '13': 'weak', '14rainy': 'no'}}
Null
```

6 5) Demonstrate the working of the Random forest algorithm. Use an appropriate data set for building and apply this knowledge toclassify a new sample.

[4]:

```
pip install random-forest-mc
```

```
Collecting random-forest-mcNote: you may need to restart the kernel to use
updated packages.

  Downloading random_forest_mc-1.0.3-py3-none-any.whl (13 kB)
Requirement already satisfied: pandas>=1.3 in c:\users\hi\anaconda3\lib\site-
packages (from random-forest-mc) (1.5.3)
Requirement already satisfied: tqdm>=4.60 in c:\users\hi\anaconda3\lib\site-
packages (from random-forest-mc) (4.64.1)
Requirement already satisfied: numpy>=1.20 in c:\users\hi\anaconda3\lib\site-
packages (from random-forest-mc) (1.23.5)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\users\hi\anaconda3\lib\site-packages (from pandas>=1.3->random-forest-mc)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\hi\anaconda3\lib\site-
packages (from pandas>=1.3->random-forest-mc) (2022.7)
Requirement already satisfied: colorama in c:\users\hi\anaconda3\lib\site-
packages (from tqdm>=4.60->random-forest-mc) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\hi\anaconda3\lib\site-
packages (from python-dateutil>=2.8.1->pandas>=1.3->random-forest-mc) (1.16.0)
Installing collected packages: random-forest-mc
Successfully installed random-forest-mc-1.0.3
```

[9]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load irs dataset
iris = load_iris()
# Create feature and target amays
X=iris.data

y = iris.target

#Splitinto training and test set
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,␣
 ↪random_state=42)

# Create a Gaussian Classifier
clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets y_pred=clf predict(X_test)
clf.fit(X_train,y_train)

# Perform prediction on the test data
y_pred = clf.predict(X_test)

# Check the accuracy of the model
print("Accuracy:", accuracy_score(y_test, y_pred))

# Classify anew sample
new_sample = [(3, 5, 4, 2)]

new_pred = clf.predict(new_sample)
print("Predicted class of new sample:" , new_pred)
```

```
Accuracy: 1.0
Predicted class of new sample: [2]
```

7 6)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]:

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score



# load the training dataset from the .CSV file
data = pd.read_csv("finds.csv")
```

```python


# split the data into features and target variables
X = data.iloc[:, :-1]
y = data.iloc[:, -1]


# encode categorical variables if necessary
X = pd.get_dummies(X)

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)


# create a Naive Bayes classifier object
nb = GaussianNB()

# train the classifier on the training data
nb.fit(X_train, y_train)


# make predictions on the testing data
y_pred = nb.predict(X_test)


# evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

8 7)Assuming a set of documents that need to be classified, use the naive Bayesian Classifier model to perform this task. Calculate the accuracy, precision, and recall for your data set

[1]:

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.datasets import fetch_20newsgroups


# load the dataset
newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

```

```python
# vectorize the documents
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)

# target labels
y_train = newsgroups_train.target
y_test = newsgroups_test.target

# create a Naive Bayes Classifier model
nb_classifier = MultinomialNB()
# train the model on the training data
nb_classifier.fit(X_train, y_train)

# make predictions on the testing data
y_pred = nb_classifier.predict(X_test)

# calculate the accuracy, precision, and recall of the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

# print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[1], line 8
      4 from sklearn.datasets import fetch_20newsgroups
      7 # load the dataset
----> 8 newsgroups_train = fetch_20newsgroups(subset='train')
      9 newsgroups_test = fetch_20newsgroups(subset='test')
     11 # vectorize the documents

File ~\anaconda3\lib\site-packages\sklearn\datasets\_twenty_newsgroups.py:269,␣
 ↪in fetch_20newsgroups(data_home, subset, categories, shuffle, random_state,␣
 ↪remove, download_if_missing, return_X_y)
    267 if download_if_missing:
    268     logger.info("Downloading 20news dataset. This may take a few minutes.
 ↪")
--> 269     cache = _download_20newsgroups(
    270         target_dir=twenty_home, cache_path=cache_path
    271     )
    272 else:
    273     raise IOError("20Newsgroups dataset not found")
```

```
File ~\anaconda3\lib\site-packages\sklearn\datasets\_twenty_newsgroups.py:83, i␣
 ↪_download_20newsgroups(target_dir, cache_path)
     78 os.remove(archive_path)
     80 # Store a zipped pickle
     81 cache = dict(
     82     train=load_files(train_path, encoding="latin1"),
---> 83     test=load_files(test_path, encoding="latin1"),
     84 )
     85 compressed_content = codecs.encode(pickle.dumps(cache), "zlib_codec")
     86 with open(cache_path, "wb") as f:

File ~\anaconda3\lib\site-packages\sklearn\datasets\_base.py:257, in␣
 ↪load_files(container_path, description, categories, load_content, shuffle,␣
 ↪encoding, decode_error, random_state, allowed_extensions)
    255 data = []
    256 for filename in filenames:
--> 257     data.append(Path(filename).read_bytes())
    258 if encoding is not None:
    259     data = [d.decode(encoding, decode_error) for d in data]

File ~\anaconda3\lib\pathlib.py:1126, in Path.read_bytes(self)
   1122 def read_bytes(self):
   1123     """
   1124     Open the file in bytes mode, read it, and close the file.
   1125     """
-> 1126     with self.open(mode='rb') as f:
   1127         return f.read()

File ~\anaconda3\lib\pathlib.py:1119, in Path.open(self, mode, buffering,␣
 ↪encoding, errors, newline)
   1117 if "b" not in mode:
   1118     encoding = io.text_encoding(encoding)
-> 1119 return self._accessor.open(self, mode, buffering, encoding, errors,
   1120                            newline)

KeyboardInterrupt:
```

9 8)Construct aBayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

[4]:

```
from bayespy.nodes import Categorical,Mixture
from bayespy.inference import VB
import numpy as np
```

```
return np.take([p_false, p_true], [[FALSE, TRUE], [TRUE, TRUE]], axis=0)

asia = Categorical([0.5, 0.5])
tuberculosis = Mixture(asia, Categorical, [[0.99, 0.01], [0.8, 0.2]])
smoking = Categorical([0.5, 0.5])
lung = Mixture(smoking, Categorical, [[0.98, 0.02], [0.25, 0.75]])
bronchitis = Mixture(smoking, Categorical, [[0.97, 0.03], [0.08, 0.92]])
xray = Mixture(tuberculosis, Mixture, lung, Categorical, or ([0.96, 0.04], [0.
 ↪115, 0.885]))

dyspnea = Mixture(bronchitis, Mixture, tuberculosis, Mixture, lung,␣
 ↪Categorical, or([0.6,0.4],[0.18,0.82]), or([0.11,0.89],[0.04,0.96]))
tuberculosis.observe(TRUE)
smoking.observe(FALSE)
bronchitis.observe(TRUE)
Q=VB(dyspnea,xray,bronchitis,lung,smoking,tuberculosis,asia)
Q.update(repeat=100)
print("P(asia):",asia.get_moments()[0][TRUE])
print("P(tuberculosis):",tuberculosis.get_moments()[0][TRUE])
print("P(smoking):",smoking.get_moments()[0][TRUE])
print("P(lung):",lung.get_moments()[0][TRUE])
print("P(bronchitis):",bronchitis.get_moments()[0][TRUE])
print("P(xray):",xray.get_moments()[0][TRUE])
print("P(dyspnea):",dyspnea.get_moments()[0][TRUE])
```

```
  Cell In[4], line 12
    xray = Mixture(tuberculosis, Mixture, lung, Categorical, or ([0.96, 0.04],␣
 ↪[0.115, 0.885]))
                                                             ^
SyntaxError: invalid syntax
```

[2]:

```
!pip install bayespy
```

```
Collecting bayespy
  Downloading bayespy-0.5.26.tar.gz (401 kB)
     ---------------------------------------- 401.7/401.7 kB 4.2 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: numpy>=1.10.0 in c:\users\hi\anaconda3\lib\site-
packages (from bayespy) (1.23.5)
Requirement already satisfied: scipy>=0.13.0 in c:\users\hi\anaconda3\lib\site-
packages (from bayespy) (1.10.0)
Requirement already satisfied: h5py in c:\users\hi\anaconda3\lib\site-packages
(from bayespy) (3.7.0)
```

```
Building wheels for collected packages: bayespy
  Building wheel for bayespy (setup.py): started
  Building wheel for bayespy (setup.py): finished with status 'done'
  Created wheel for bayespy: filename=bayespy-0.5.26-py3-none-any.whl
size=377662
sha256=3c9333cea1e02437233cf5a9682a9969794700f9a0970922ea71713dccd0bba4
  Stored in directory: c:\users\hi\appdata\local\pip\cache\wheels\17\b2\f9\55d4f
52e600e891f31b66b776199e078c4a94de2c127ab3c4a
Successfully built bayespy
Installing collected packages: bayespy
Successfully installed bayespy-0.5.26
```

[1]:

```python
from bayespy.nodes import Categorical, Mixture
from bayespy.inference import VB
import numpy as np

TRUE = 1
FALSE = 0

p_false = 0.0  # Replace with the actual probability values
p_true = 1.0  # Replace with the actual probability values

# Define the conditional probability tables
asia = Categorical([0.5, 0.5])
tuberculosis = Mixture(asia, Categorical, [np.array([0.99, 0.01]), np.array([0.
 ↪8, 0.2])])
smoking = Categorical([0.5, 0.5])
lung = Mixture(smoking, Categorical, [np.array([0.98, 0.02]), np.array([0.25, 0.
 ↪75])])
bronchitis = Mixture(smoking, Categorical, [np.array([0.97, 0.03]), np.array([0.
 ↪08, 0.92])])
#xray = Mixture(tuberculosis, Mixture, lung, Categorical, np.array([0.96, 0.
 ↪04]), np.array([0.115, 0.885]))
xray = Mixture(tuberculosis, Mixture(lung, Categorical, np.array([0.96, 0.
 ↪04])), Categorical, np.array([0.115, 0.885]))

dyspnea = Mixture(bronchitis, Mixture, tuberculosis, Mixture, lung,␣
 ↪Categorical, np.array([0.6, 0.4]), np.array([0.18, 0.82]), np.array([0.11, 0.
 ↪89]), np.array([0.04, 0.96]))

# Observations
tuberculosis.observe(TRUE)
smoking.observe(FALSE)
bronchitis.observe(TRUE)

# Perform variational Bayesian inference
Q = VB(dyspnea, xray, bronchitis, lung, smoking, tuberculosis, asia)
```

```python
Q.update(repeat=100)

# Print the probabilities
print("P(asia):", asia.get_moments()[0][TRUE])
print("P(tuberculosis):", tuberculosis.get_moments()[0][TRUE])
print("P(smoking):", smoking.get_moments()[0][TRUE])
print("P(lung):", lung.get_moments()[0][TRUE])
print("P(bronchitis):", bronchitis.get_moments()[0][TRUE])
print("P(xray):", xray.get_moments()[0][TRUE])
print("P(dyspnea):", dyspnea.get_moments()[0][TRUE])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[1], line 18
     16 bronchitis = Mixture(smoking, Categorical, [np.array([0.97, 0.03]), np.
  ↪array([0.08, 0.92])])
     17 #xray = Mixture(tuberculosis, Mixture, lung, Categorical, np.array([0.
  ↪96, 0.04]), np.array([0.115, 0.885]))
---> 18 xray = Mixture(tuberculosis,␣
  ↪Mixture(lung, Categorical, np.array([0.96, 0.04])), Categorical, np.array([0.
  ↪115, 0.885]))
     20 dyspnea = Mixture(bronchitis, Mixture, tuberculosis, Mixture, lung,␣
  ↪Categorical, np.array([0.6, 0.4]), np.array([0.18, 0.82]), np.array([0.11, 0.
  ↪89]), np.array([0.04, 0.96]))
     22 # Observations

File ~\anaconda3\lib\site-packages\bayespy\inference\vmp\nodes\mixture.py:431,␣
  ↪in Mixture.__init__(self, z, node_class, cluster_plate, *params, **kwargs)
    429 def __init__(self, z, node_class, *params, cluster_plate=-1, **kwargs):
    430     self.cluster_plate = cluster_plate
--> 431     super().__init__(z, node_class, *params, cluster_plate=cluster_plate,
    432                      **kwargs)

File ~\anaconda3\lib\site-packages\bayespy\inference\vmp\nodes\expfamily.py:82,␣
  ↪in useconstructor.<locals>.constructor_decorator(self, *args, **kwargs)
     75 def constructor_decorator(self, *args, **kwargs):
     76     if (self.dims is None or
     77         self._distribution is None or
     78         self._moments is None or
     79         self._parent_moments is None):
     81         (args, kwargs, dims, plates, dist, stats, pstats) = \
---> 82           self._constructor(*args, **kwargs)
     84         self.dims = dims
     85         self._distribution = dist

File ~\anaconda3\lib\site-packages\bayespy\inference\vmp\nodes\mixture.py:449,␣
  ↪in Mixture._constructor(cls, z, node_class, cluster_plate, *args, **kwargs)
```

```
    447 # Check that at least one of the parents has the cluster plate axis
    448 if len(mixture_plates) < abs(cluster_plate):
--> 449     raise ValueError("The mixed distribution does not have a plates "
    450                      "axis for the cluster plate axis")
    452 # Resolve the number of clusters
    453 mixture_plates = list(mixture_plates)

ValueError: The mixed distribution does not have a plates axis for the cluster
  ↪plate axis
```

10 9)Demonstrate the working of EM algorithm to cluster a set of data stored in a .CSV file.

[13]:

```python
import pandas as pd
import numpy as np
from sklearn.mixture import GaussianMixture

# Load the data from the CSV file
data = pd.read_csv('data.csv')

# Convert the data to a numpy array
X = data.values

# Specify the number of clusters
num_clusters = 3

# Create an instance of Gaussian Mixture Model
gmm = GaussianMixture(n_components=num_clusters)

# Fit the GMM to the data
gmm.fit(X)

# Predict the cluster labels for the data points
labels = gmm.predict(X)

# Print the cluster labels
print('Cluster Labels:')
print(labels)
```

```
Cluster Labels:
[1 1 1 1 0 0 2]
```

```
C:\Users\HI\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
```

```
  warnings.warn(
```

11 10)Demonstrate the working of SVM classifier for a suitable data set

[10]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load the dataset
cancer = load_breast_cancer()

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,␣
 ↪test_size=0.2, random_state=42)
from sklearn.svm import SVC

# create an SVM classifier
svm = SVC(kernel='linear', C=1)

# train the classifier
svm.fit(X_train, y_train)
```

[10]:
```
SVC(C=1, kernel='linear')
```

[11]:
```
from sklearn.metrics import accuracy_score, classification_report

# make predictions on the testing data
y_pred = svm.predict(X_test)

# calculate the accuracy and print the classification report of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

report = classification_report(y_test, y_pred)
print("Classification report:\n", report)
```

```
Accuracy: 0.956140350877193
Classification report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

[ ]:

```
```