Bisesh Shrestha

Student ID: **219570044**

BSc (Hons) Computer Systems Engineering

International School of Management and Technology (ISMT), Nepal

TUTORIAL TASKS FROM WEEK EXERCISES

I have uploaded all my Lab Exercise and Task Sheet to my eportfolio which can be viewed through the link below,

https://canvas.sunderland.ac.uk/eportfolios/10091?verifier=xLMGe6HDXgG007CwBnjVNWmIHYVxONb99D7uSnww

Week 1

Task Sheet

Task 1: Computerised Agents

In groups of 3-5 you should perform the simple task of stacking 3 objects (e.g., books) in which each object is rotated 180 degrees from the preceding one. You will have to be imaginative because we must perform this remotely, but it is the process of breaking down the task and communicating ideas that needs to be considered and captured for this task.

1. Each person in the group takes one role: left eye, right eye, left arm, right arm, brain.
2. Note that the eyes cannot think, the brain and arms cannot see, and the arms are also deaf! This means that the eyes must describe the scene they see, the brain makes decisions on this information and passes commands to the hands.

3. After 15 minutes break from this task and consider:
    a. What issues and problems arose and what would you do differently next time?
    Some potential issues and problems that may have arisen during the task include misunderstandings or miscommunications between the different roles, difficulty in accurately describing the objects or their positions, and difficulty in coordinating the movements of the arms. To address these issues, the group could consider the following:

    Establishing clearer communication protocols, such as using specific language or terminology to describe the objects and their positions

Providing more detailed descriptions of the objects and their positions, including their size, shape, colour, and distance from the eyes

Using visual aids or diagrams to help communicate the desired positions and orientations of the objects

Practicing the task beforehand to improve coordination and communication between the different roles

By addressing these issues and problems, the group can improve their performance and efficiency in future tasks.

b. What would the main components of a very simple "agent" be (e.g. an ant which lived in a box around which food was scattered and which only needed to eat enough to stay alive)?

The main components of a very simple "agent" such as an ant living in a box with food scattered around it would include:

Sensors: These are devices that allow the agent to gather information about its environment. In the case of an ant, the sensors might include sensors to detect the presence and location of food.

Actuators: These are devices that allow the agent to interact with its environment. In the case of an ant, the actuators might include muscles or other mechanisms to move the ant towards the food.

Control system: This is a component that processes the information from the sensors and activates the actuators in a way that allows the agent to achieve its goals. In the case of an ant, the control system might include algorithms or rules that dictate how the ant should move towards the food.

Memory: This is a component that allows the agent to store and retrieve information about its past experiences. In the case of an ant, the memory might include information about the location and availability of food sources.

By incorporating these components, the agent can gather information about its environment, make decisions based on that information, and interact with its environment in order to achieve its goals.

c. What problems might there be with such a simple agent and what would need to be added to this agent to make it more intelligent.

Some problems that may arise with such a simple agent include:

Lack of adaptability: A simple agent may struggle to adapt to changing environments or to learn from past experiences, which can limit its ability to solve new or complex problems.

Limited intelligence: A simple agent may only be able to perform a narrow range of tasks, and may not have the ability to think, reason, or learn in the same way that a human does.

To make the agent more intelligent, additional components such as learning mechanisms or more advanced control systems could be added. These components could allow the agent to learn from past experiences and adapt to new situations, which would enable it to perform a wider range of tasks and solve more complex problems.

d. What do we mean by intelligence and artificial intelligence?

Intelligence refers to the ability of an individual or a system to think, reason, and learn. It involves the ability to process and analyze information, make decisions based on that information, and adapt to new situations.

Artificial intelligence, or AI, refers to the development of computer systems and software that are able to perform tasks that would normally require human intelligence. This includes tasks such as visual perception, speech recognition, decision-making, and language translation.

There are different types of artificial intelligence, including weak AI, which is designed to perform a specific task, and strong AI, which is designed to have general intelligence and the ability to perform a wide range of tasks. There are also various techniques used in AI, including supervised learning, unsupervised learning, and reinforcement learning.

4. Produce a definition of AI with your group

Artificial intelligence is the development of computer systems and software that are able to perform tasks that would normally require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. AI involves the use of algorithms, machine learning techniques, and other advanced technologies to enable computers and software to process and analyze information, make decisions based on that information, and adapt to new situations. AI can be classified as either weak or strong, depending on its ability to perform a wide range of tasks, and there are various techniques used in AI, including supervised learning, unsupervised learning, and reinforcement learning. The goal of AI is to enable computers and software to perform tasks that would normally require human intelligence and to improve the efficiency and effectiveness of these tasks.

Task 2: Latest Developments in AI

Today we looked at a selection of the latest developments in Artificial Intelligence. Please select one of these developments to look into in more detail, or source your own idea.

1. You are to produce a one or two page brief which will discuss the possible techniques being used to deliver solutions in your chosen application area, its impact on society and any associated ethical implications (if there are no ethical implications please discuss why)

One development in Artificial Intelligence that has gained significant attention in recent years is the use of machine learning algorithms to analyze and interpret large datasets. Machine learning algorithms are designed to find patterns and relationships within data and use this information to make predictions or decisions. These algorithms have been applied to a wide range of applications, including healthcare, finance, and marketing.

One potential technique for delivering solutions in this application area is the use of supervised learning algorithms, which are trained on a labeled dataset in order to learn how to classify or predict outcomes. For example, a machine learning algorithm might be trained on a dataset of medical records in order to predict the likelihood of a patient developing a particular disease.

The impact of machine learning algorithms on society has been significant. These algorithms have the potential to improve decision-making and efficiency in a wide range of industries, and have been used to predict and prevent a variety of problems, such as fraudulent activity, credit defaults, and patient outcomes. However, there are also ethical considerations associated with the use of machine learning algorithms, such as the potential for bias in the training data to be reflected in the algorithm's decisions, and the need to protect personal data and privacy.

In order to address these ethical concerns, it is important for organizations using machine learning algorithms to carefully consider the implications of their use, and to put in place measures to ensure that they are used ethically and responsibly. This may include ensuring that the training data is representative and free from bias, and implementing robust data protection measures to protect personal data and privacy.

Week2

Task 1: Relationships

Scenario: You want to automate the dynamic clothing of your characters in the action game you are creating, and your game world is modelled on our own ecosystem.

1. In groups of around 4 work together to create an IF THEN frame for clothes for seasons and location.

IF THEN frame for clothes for seasons and location:

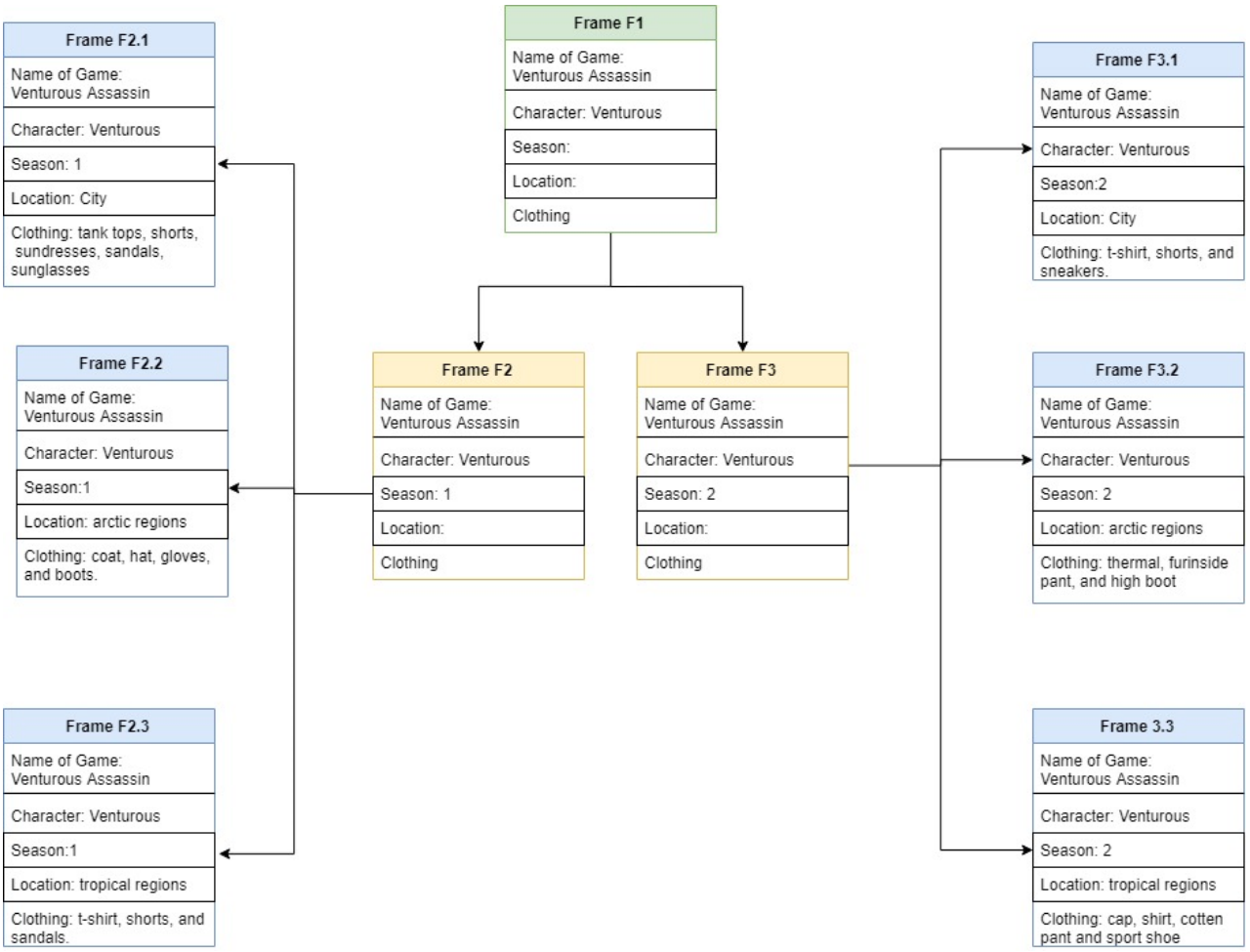


Fig 1: IF THEN frame diagrammatic representation for clothes based on Season and Location

Task 2: Semantics

Scenario: You need to model a semantic net to contribute to a machine vision project that wants to automate the sorting of wildlife images.

1. In your same groups create a semantic net that clarifies the difference between animals and birds.

Here is an example of a semantic net that clarifies the difference between animals and birds:

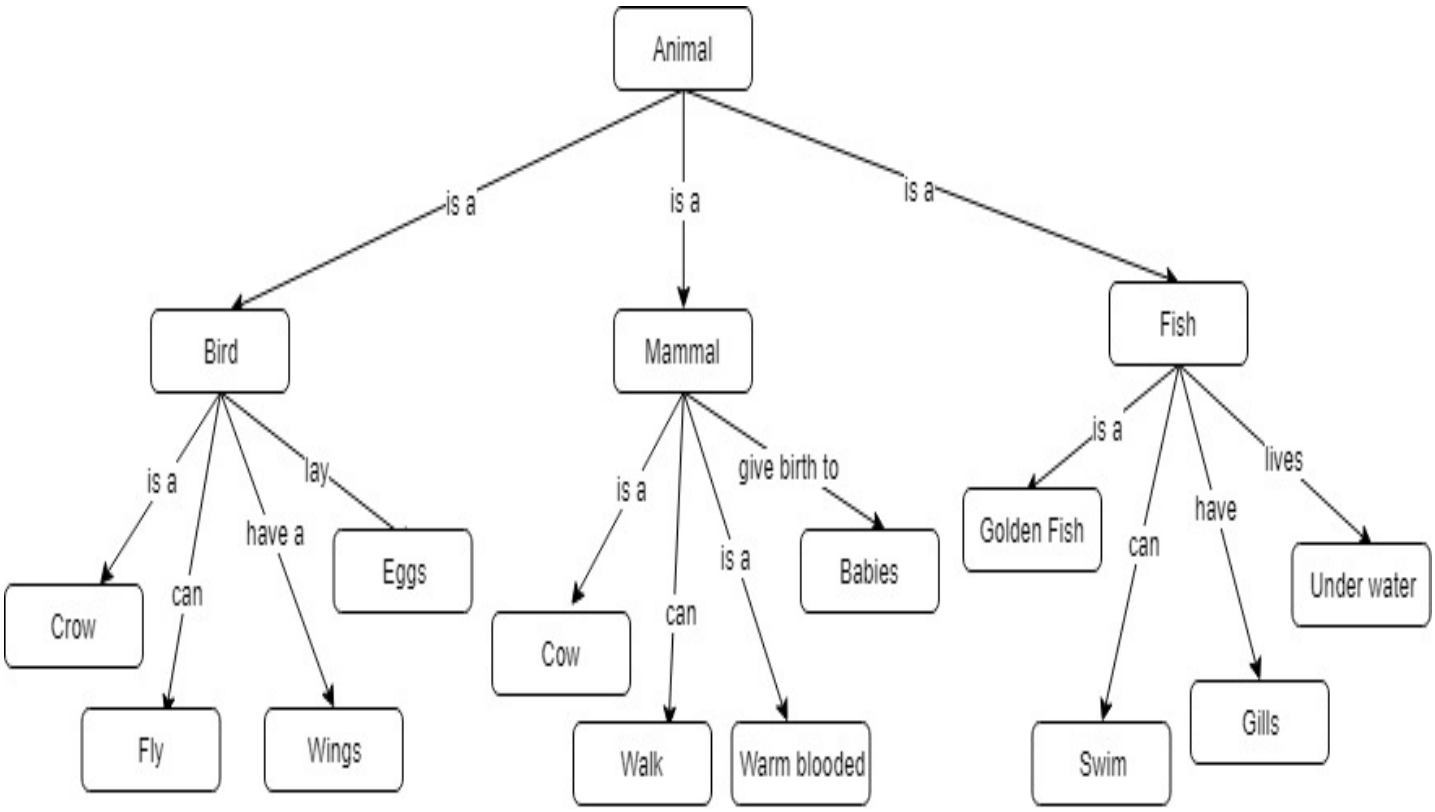


Fig 2: Semantic Net diagram for differentiating animals and birds

Lab Exercise

In [1]:
```
import json
from kanren import Relation, facts, run, eq, membero, var, conde
```

In [2]:
```
# check if 'x' is the parent of 'y'
def parent(x,y):
    return conde([father(x,y)],[mother(x,y)])

# check if 'x' is grand parent of 'y'
def grandparent(x,y):
    temp = var()
    return conde((parent(x,temp),parent(temp,y)))

# check for sibling relationship between 'a' and 'b'
def sibling(x,y):
    temp = var()
    return conde((parent(temp,x),parent(temp,y)))

# check if 'x' is the uncle of 'y'
def uncle(x,y):
    temp = var()
    return conde((father(temp,x),grandparent(temp,y)))
```

In [3]:
```
# check if 'x' is the cousin of 'y'
def cousin(x,y):
    temp = var()
    return conde((father(temp,x),uncle(temp,y)))
```

In [7]:
```
# check if 'x' is the aunty of 'y'
def aunty(x,y):
    temp = var()
    return conde((mother(x,temp),cousin(temp,y)))
```

In [12]:
```
if __name__=='__main__':
    father=Relation()
    mother=Relation()
    with open('relationships.json') as f:
        d=json.loads(f.read())

    for item in d['father']:
        facts(father, (list(item.keys())[0], list(item.values())[0]))

    for item in d['mother']:
        facts(mother, (list(item.keys())[0],list(item.values())[0]))

    x=var()
```

In [13]:
```
# John's children
name='John'
output=run(0,x,father(name,x))
print("\nList of "+name+"'s children:")
for item in output:
    print(item)

# William's mother
name='William'
output=run(0,x,mother(x,name))[0]
print("\n"+name+"'s mother: \n"+output)

# Adam's parents
name='Adam'
output=run(0,x,parent(x,name))
print("\nList of "+name+"'s parents: ")
for item in output:
    print(item)

# Wayne's parents
name="Wayne"
output=run(0,x,parent(x,name))
print("\nList of "+name+"'s parents:")
for item in output:
    print(item)

# Wayne's grandparents
name="Wayne"
output=run(0,x,grandparent(x,name))
print("\nList of "+name+"'s grandparents:")
for item in output:
    print(item)

# Megan's grandchildren
name="Megan"
output=run(0,x,grandparent(name,x))
print("\nList of "+name+"'s grandchildren:")
for item in output:
    print(item)
```

```
List of John's children:
Adam
David
William

William's mother:
Megan

List of Adam's parents:
John
Megan

List of Wayne's parents:
David

List of Wayne's grandparents:
John
Megan

List of Megan's grandchildren:
Chris
Sophia
Peter
Stephanie
Neil
Wayne
Julie
Tiffany
```

In [14]:
```
# David's siblings
name='David'
output=run(0,x,sibling(x,name))
siblings=[x for x in output if x!=name]
print("\nList of "+name+"'s siblings:")
for item in siblings:
    print(item)

# Tiffany's uncles
name = 'Tiffany'
name_father = run(0,x,father(x,name))[0]
output = run(0,x,uncle(x,name))
output = [x for x in output if x!=name_father]
print("\nList of "+ name + "'s uncles:")
for item in output:
    print(item)

# All spouses
a,b,c = var(),var(),var()
output = run(0,(a,b),father(a,c), mother(b,c))
print("\nList of all spouses: ")
for item in output:
    print('Husband:', item[0],'<==>Wife',item[1])
```

```
List of David's siblings:
Adam
William

List of Tiffany's uncles:
Adam
William

List of all spouses:
Husband: William <==>Wife Emma
Husband: Adam <==>Wife Lily
Husband: David <==>Wife Olivia
Husband: John <==>Wife Megan
```

In [15]:
```
# Extension works
# Chris's cousin
name='Chris'
output=run(0,x,cousin(x,name))
siblings=run(0,x,sibling(x,name))
cousins = [x for x in output if x not in siblings]
print("\nList of "+name+"'s cousin:")
for item in cousins:
    print(item)
```

```
List of Chris's cousin:
Sophia
Peter
Neil
Wayne
Julie
```

In [16]:
```
# Chris's aunty
name='Chris'
output=run(0,x,aunty(x,name))
mom=run(0,x,mother(x,name))[0]
auntys = [x for x in output if x!= mom]
print("\nList of "+name+"'s aunty:")
for item in set(auntys):
    print(item)
```

```
List of Chris's aunty:
Lily
Olivia
```

Exercise2:

In [6]:
```
from kanren import Relation, facts, run, eq, membero, var, conde
```

In [7]:
```
adjacent =Relation()
coastal = Relation()
```

In [8]:
```
file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'
```

In [9]:
```
# Read the file containing coastal states
with open(file_coastal,'r') as f:
    line = f.read()
    coastal_states = line.split(',')

# Add info to the facts base
for state in coastal_states:
    facts(coastal,state)

# read the file containing the adjacent states
with open(file_adjacent,'r') as f:
    adjlist = [line.strip().split(',') for line in f if line and line[0].isalpha()]

# Add info to the fact base
for L in adjlist:
    head, tail = L[0],L[1:]
    for state in tail:
        facts(adjacent,head,tail)
```

In [5]:
```
# initialize the variables
x =var()
y = var()
```

In [12]:
```
# is Nevada adjacent to Louisiana?
output = run(0,x,adjacent('Nevada','Louisiana'))
print('\nIs Nevada adjacent to Louisiana?')
print('Yes' if len(output) else 'No')

# States adjacent to Oregon
output = run(0,x,adjacent('Oregon',x))
print('\nList of states adjacent to Oregon:')
for item in output:
    print(item)

# States adjacent to Mississippi that are coastal
output = run(0,x,adjacent('Mississippi',x),coastal(x))
for item in output:
    print(item)

# List of 'n' states that border a coastal state
output =
```

```
Is Nevada adjacent to Louisiana?
No
```

In [6]:
```
n = int(input("Enter total number to calculate average:"))
sum = 0
for i in range(n):
    x = float(input("Enter a number:"))
    sum+=x
print("Average: ",sum/n)
```

```
Enter total number to calculate average.3
Enter a number2
Enter a number4
Enter a number6
Average:  4.0
```

Week 3

Task 1: Search Algorithms

1. Do some background research on the search algorithms and methods we have looked at today and produce a scenario where each one might be used and a set of pros and cons for each of them

1. Search Techniques:

Scenario: You have a maze and you want to find the shortest path from the start to the end.

Pros:

Simple and easy to implement

Can be effective for small or well-structured search spaces

Cons:

May not be efficient for large or complex search spaces

May get stuck in infinite loops or miss the optimal solution

2. Heuristic Search:

Scenario: You are playing a game of chess and want to find the best move to make based on your current position and the position of your opponent's pieces.

Pros:

Can find the optimal solution faster than other search algorithms

Can handle large and complex search spaces more efficiently

Cons:

May not be guaranteed to find the optimal solution

Can be computationally expensive if the heuristic function is not designed well

3. Different Algorithms for Searching:

Scenario: You have a large database of information and you want to find a specific record as quickly as possible.

Pros:

Can find the desired record quickly and efficiently

Can handle large datasets effectively

Cons:

May not be suitable for search spaces that are not well-structured or have a large number of possible solutions

Can be computationally expensive for large datasets

4. Genetic Algorithms:

Scenario: You want to optimize a complex system, such as a supply chain or a manufacturing process, by finding the best combination of variables and parameters.

Pros:

Can find good solutions quickly and efficiently

Can handle large and complex search spaces effectively

Cons:

May not be guaranteed to find the optimal solution

Can be computationally expensive if the fitness function is not designed well

5. Planning:

Scenario: You want to plan a route for a self-driving car to take from one location to another, considering factors such as traffic, road conditions, and fuel efficiency.

Pros:

Can find a logical and consistent plan that takes into account multiple constraints and objectives

Can handle complex and dynamic search spaces effectively

Cons:

May be computationally expensive for large or complex search spaces

May not be able to find a solution if there are too many constraints or conflicting objectives.

Week 4

Task 1: Adaptive drill for language learning

This task will look at Bayes Probability in action:

1. Group up with 3 other people in your group
2. Nominate one person to identify 10 words in a language not native to the group (e.g. Latin)
3. That person should then say the words to the group and the other group members should write the words down.

Original word	Bisesh Shrestha	Sanam Bogati	Nischal Pradhan	Hitam Lama
Abutor	Abutor	aboutor	Abutor	Abutor
bibo	**bibo**	**bibo**	**bibo**	**bibo**

calco	**calco**	**calco**	**calco**	**calco**
deficio	difeco	**deficio**	**dificio**	**deficio**
expeto	**expeto**	**expeto**	**expeto**	**expeto**
fere	**fere**	**fere**	Firi	feri
grassor	**grassor**	**grassor**	**grassor**	**grassor**
hic	**hic**	**hic**	**hic**	**hik**
ilico	**ilico**	**ilico**	Ileco	**ilico**
jugis	**jugis**	**jugis**	**juges**	**juges**
Error	1 word	1 word	4 words	3 Words

4. Once this is done pool everyone's answers and analyse them:
    a. Categorise errors
        i. Look at the answers provided by each group member, and identify any errors that were made.
        ii. Categorize the errors into different types, such as mispronunciation, spelling mistakes, or misunderstanding of the word's meaning.
    b. Look at similarities
        i. Look for common errors made by multiple group members. These may indicate a pattern or trend in the group's understanding of the language.
        ii. Look for differences in the errors made by different group members. These may indicate individual strengths and weaknesses in the group's understanding of the language.
    c. Work our error frequencies
        i. Calculate the frequency of each type of error made by each group member.
        ii. Compare the frequency of errors made by each group member to identify any trends or patterns.

5. Jot down some of your thoughts about the exercise and how this relates to some of the things we have covered today.

Week 5

In [6]:
```
import nltk
import numpy as np
import random
import string # to process standard python strings

from nltk.stem import WordNetLemmatizer
```

In [7]:
```
# nltk.download()
```

In [8]:
```
f = open('chatbot.txt','r',errors = 'ignore')
raw = f.read()
raw = raw.lower() # converts to lowercase
nltk.download('punkt') # first-time use only
nltk.download('wordnet') #first-time use only

sent_tokens = nltk.sent_tokenize(raw) # converts to list of sentences
word_tokens = nltk.word_tokenize(raw) # converts to list of words
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ACER\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ACER\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [9]:
```
sent_tokens[:2]
```

Out[9]:
```
['a chatbot or chatterbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, in lieu of providing direct contact with a live human agent.',
 '[1][2] designed to convincingly simulate the way a human would behave as a conversational partner, chatbot systems typically require continuous tuning and testing, and many in production remain unable to adequately converse, while none of them can pass the standard turing test.']
```

In [10]:
```
word_tokens[:2]
```

Out[10]:
```
['a', 'chatbot']
```

In [11]:
```
lemmer = nltk.stem.WordNetLemmatizer()
# WordNet is a semantically-oriented dictionary of English included in NLTK

def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct),None) for punct in string.punctuation)

def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

In [12]:
```
# Keyword matching
greeting_inputs = ("hello","hi","greetings","sup","what's up", "hey","namaste","namaskar")
greeting_responses = ("hi","hey","*nods*","hi there","hello", "I am glad! You are talking to me","namaste","namaskar")

def greeting(sentence):
#     for word in sentence.split(): # for what's up greeting
    if sentence.lower() in greeting_inputs:
            return random.choice(greeting_responses)
```

In [13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

In [14]:
```
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer = LemNormalize,stop_words = 'english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1],tfidf)
    idx = vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]

    if(req_tfidf == 0):
        robo_reponse = robo_response+" I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

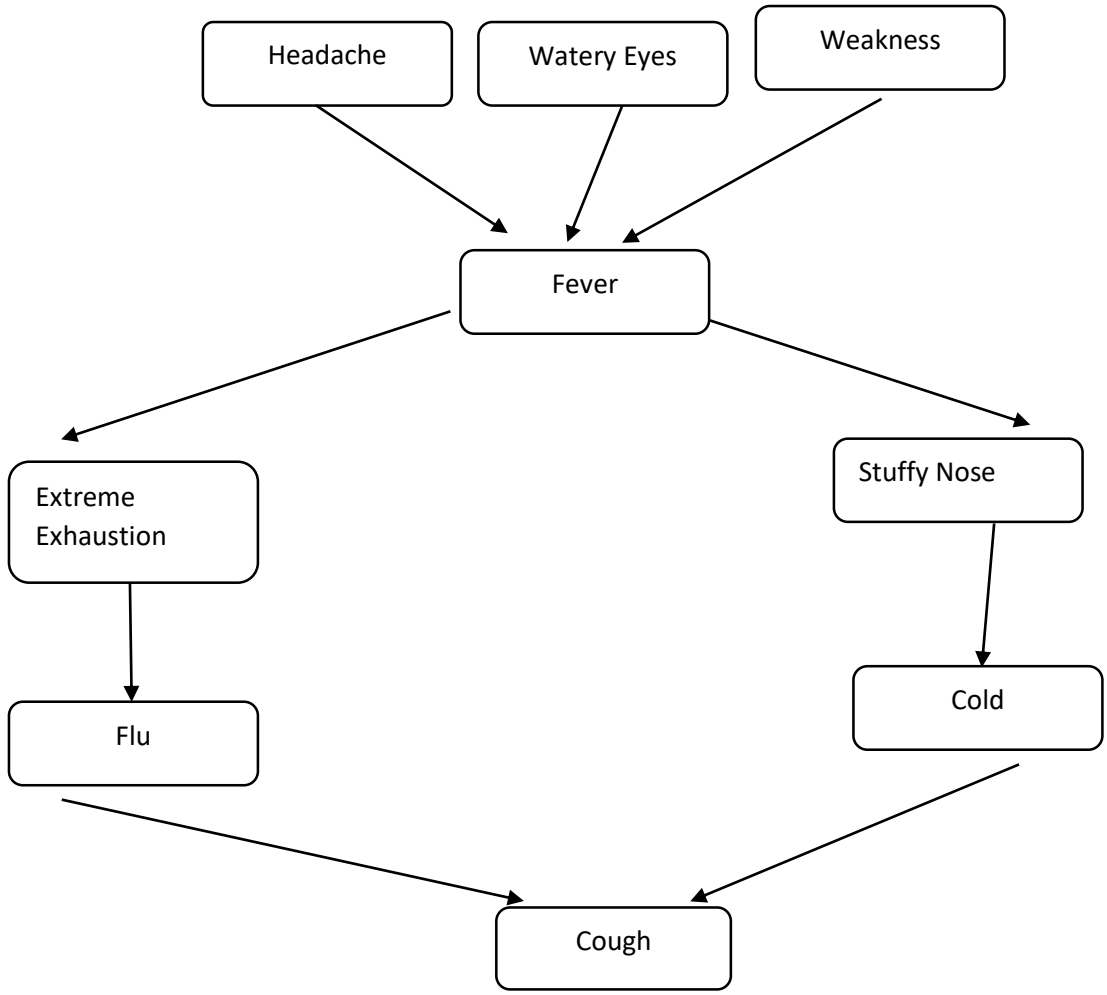
In [ ]:
```
flag = True
print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")

while(flag):
    user_response = input()
    user_response = user_response.lower()
    if(user_response!='bye'):
        if(user_response == 'thanks' or user_response == 'thank you'):
            flag = False
            print("ROBO: You are welcome...")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response)+"! How may I help you?")
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag = False
        print("ROBO: Bye! Take care...")
```

```
ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!
hello
ROBO: hi! How may I help you?
```

Week6

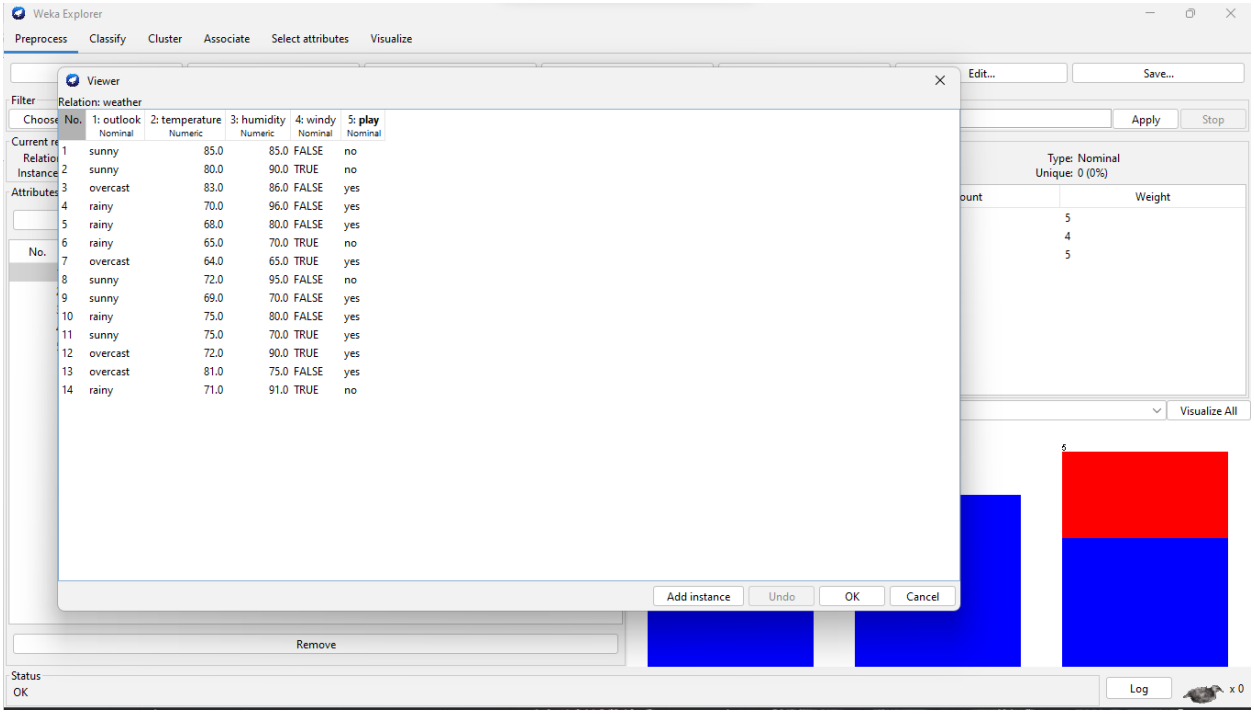
Task : Reasoning Task and Using Weka on a Different Dataset

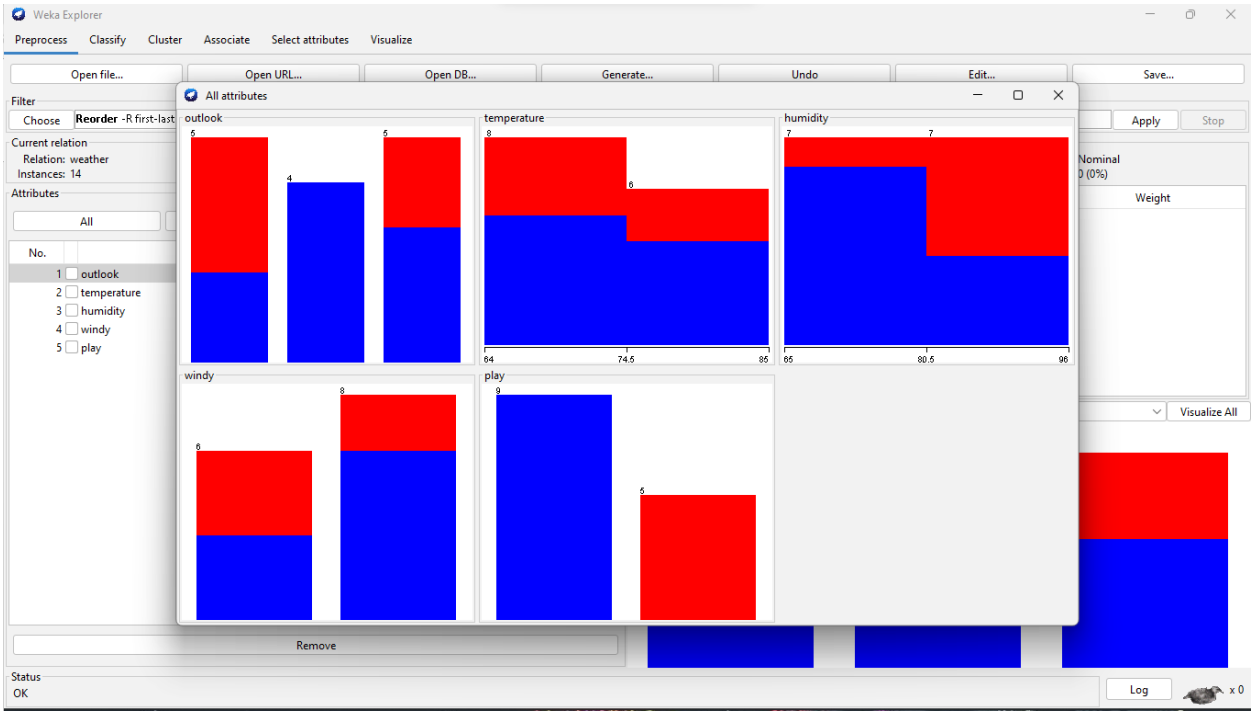
1. Research the symptoms for a cold and the flu and draw a simple JPD to help with diagnosis

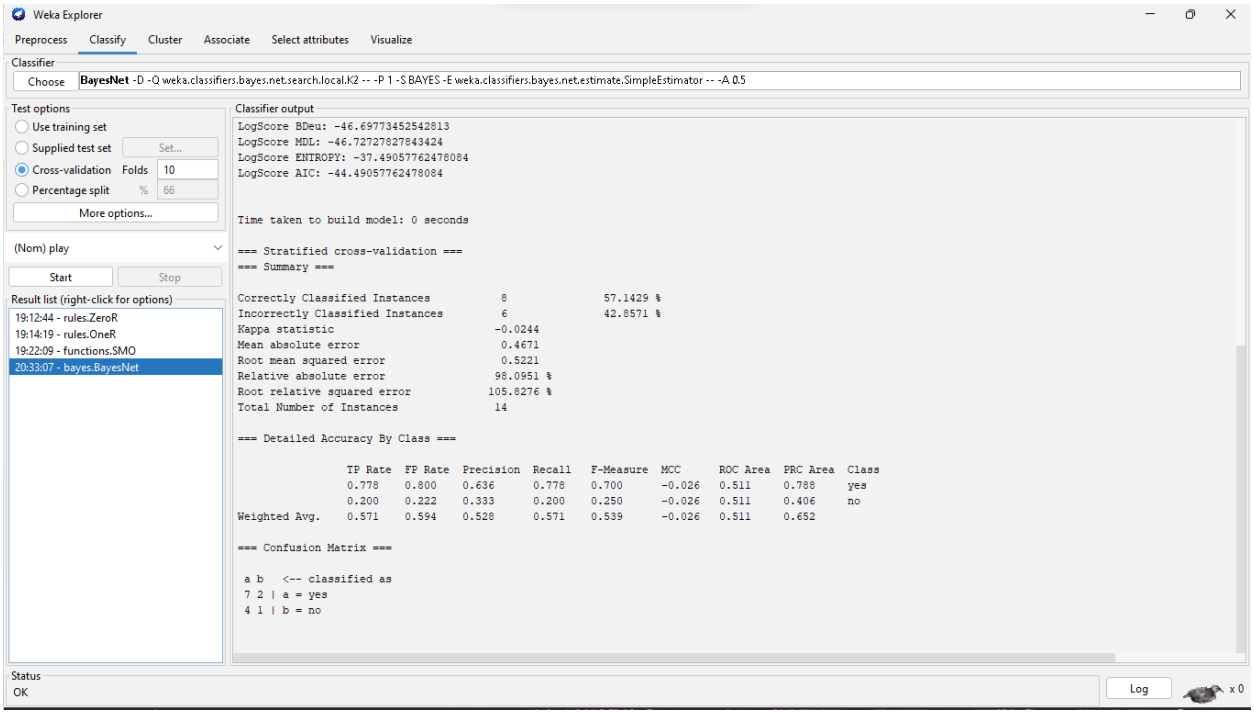


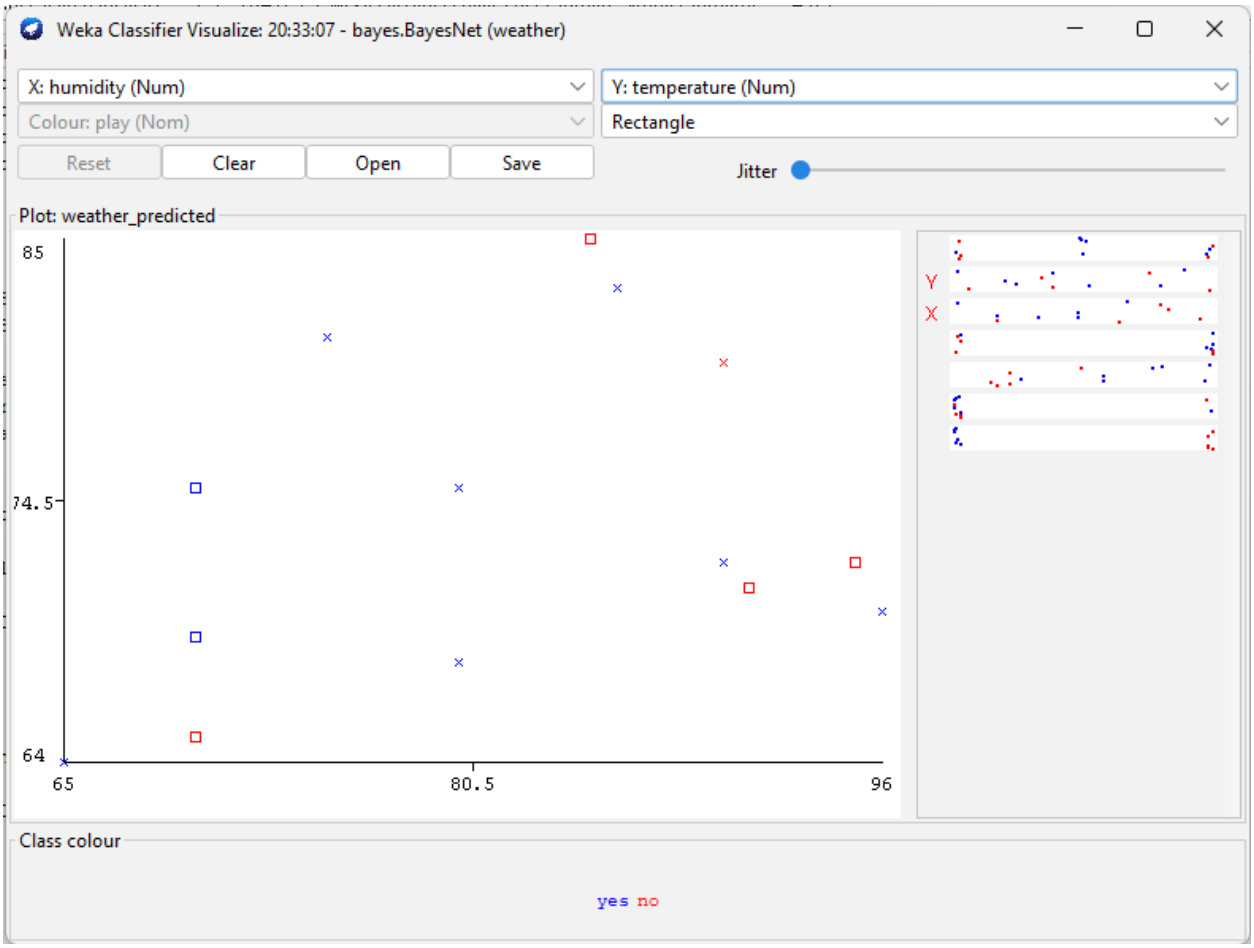
WEKA Machine Learning: Naive Bayes

Data: weather.numeric




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Reorder -R first-last
Apply
Stop
Current relation
Relation: weather
Instances: 14
Attributes
All
No.
1 outlook
2 temperature
3 humidity
4 windy
5 play
Nominal
0 (0%)
Weight
Visualize All
Remove
Status
OK
Log
x 0
All attributes
outlook
temperature
humidity
windy
play


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
BayesNet -D -Q weka.classifiers.bayes.net.search.local.K2 -- -P 1 -S BAYES -E weka.classifiers.bayes.net.estimate.SimpleEstimator -- -A 0.5
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) play
Start
Stop
Result list (right-click for options)
19:12:44 - rules.ZeroR
19:14:19 - rules.OneR
19:22:09 - functions.SMO
20:33:07 - bayes.BayesNet
Classifier output
LogScore BDeu: -46.69773452542813
LogScore MDL: -46.72727827843424
LogScore ENTROPY: -37.49057762478084
LogScore AIC: -44.49057762478084
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 8 57.1429 %
Incorrectly Classified Instances 6 42.8571 %
Kappa statistic -0.0244
Mean absolute error 0.4671
Root mean squared error 0.5221
Relative absolute error 98.0951 %
Root relative squared error 105.8276 %
Total Number of Instances 14
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.778 0.800 0.636 0.778 0.700 -0.026 0.511 0.788 yes
0.200 0.222 0.333 0.200 0.250 -0.026 0.511 0.406 no
Weighted Avg. 0.571 0.594 0.528 0.571 0.539 -0.026 0.511 0.652
=== Confusion Matrix ===
a b <-- classified as
7 2 | a = yes
4 1 | b = no
Status
OK
Log
x 0



Week 7

Task: Machine Learning Types

1. In groups of 4 to 6, think about each of the 3 types of machine learning, then do some research to find 2 examples of practical applications for each type.
2. List your applications and tell us what they do and if possible, how they do it.

1. Supervised learning:

i. Spam filter: A spam filter is a practical application of supervised learning that is used to identify and classify spam emails. The spam filter is trained on a labelled dataset of spam and non-spam emails, and uses this training data to learn the characteristics of spam emails. Once trained, the spam filter can then classify new emails as spam or non-spam based on the features it has learned.

ii. Credit fraud detection: Credit fraud detection is another practical application of supervised learning that is used to identify fraudulent transactions in a credit card dataset. The fraud detection system is trained on a labelled dataset of fraudulent and non-fraudulent transactions, and uses this training data to learn the characteristics of fraudulent transactions. Once trained, the fraud detection system can then classify new transactions as fraudulent or non-fraudulent based on the features it has learned.

2. Unsupervised learning:

i. Anomaly detection: Anomaly detection is a practical application of unsupervised learning that is used to identify unusual or anomalous patterns in data. Anomaly detection algorithms are trained on a dataset of normal patterns, and use this training data to learn what is considered normal behaviour. Once trained, the anomaly detection algorithm can then identify patterns in new data that are significantly different from the normal patterns, and flag these patterns as potential anomalies.

ii. Clustering: Clustering is another practical application of unsupervised learning that is used to group data points into clusters based on their similarity. Clustering algorithms are trained on a dataset of data points, and use this training data to learn how to group similar data points together. Once trained, the clustering algorithm can then group new data points into clusters based on their similarity to the data points in the training dataset.

3. Reinforcement learning:

i. Game playing: Game playing is a practical application of reinforcement learning that is used to train artificial intelligence (AI) agents to play games such as chess or Go. In reinforcement learning, an AI agent learns to play a game by receiving rewards or punishments for its actions, and adjusting its behaviour accordingly. The AI agent learns to maximize its rewards over time by exploring different actions and adapting to the game's rules and objectives.

ii. Robotics: Robotics is another practical application of reinforcement learning that is used to train robots to perform tasks in real-world environments. In reinforcement learning, a robot learns

to perform a task by receiving rewards or punishments for its actions, and adjusting its behaviour accordingly. The robot learns to maximize its rewards over time by exploring different actions and adapting to the task's requirements and constraints.

Week 9

Task: Clustering Research

3. In groups of 4 to 6, identify 2 machine learning clustering algorithms
4. List the best use for each of the algorithms you have found

Here are two machine learning clustering algorithms and their best uses:

1. K-means clustering: K-means clustering is an iterative algorithm that is used to group data points into clusters based on their similarity. It is best used for datasets with a small number of distinct clusters and relatively homogeneous data points within each cluster. K-means clustering is a fast and simple algorithm that is easy to implement and can handle large datasets efficiently.

2. Hierarchical clustering: Hierarchical clustering is an algorithm that is used to group data points into clusters based on their similarity. It is best used for datasets with a large number of data points, or for datasets with complex relationships between data points. Hierarchical clustering generates a tree-like structure (a dendrogram) that shows the relationships between data points and the clusters they belong to. This can be useful for visualizing the structure of the dataset and understanding the relationships between data points.

Some possible uses for these algorithms include:

1. K-means clustering:

i. Customer segmentation: K-means clustering can be used to group customers into different segments based on their characteristics, such as age, income, or purchasing habits. This can be useful for targeted marketing and personalized recommendations.

ii. Data compression: K-means clustering can be used to reduce the size of a dataset by representing data points as the centroids of their clusters, rather than storing each data point individually. This can be useful for reducing the storage requirements of a dataset and improving the performance of machine learning algorithms.

2. Hierarchical clustering:

i. Gene expression analysis: Hierarchical clustering can be used to group genes based on their expression patterns, and identify genes that are co-expressed or differentially expressed. This can be useful for understanding the relationships between genes and their functions.

ii. Document clustering: Hierarchical clustering can be used to group documents based on their content and identify topics or themes within the documents. This can be useful for organizing and categorizing a large collection of documents.

Week 10

Task: Neural Networks

1. In For your e-Portfolio – find an interesting application of an Artificial Neural Network, then compile a short info sheet on the following:
    a. What it is being used for?
    b. Any success/failure stats you can find
    c. What type of ANN it is and any methods it employs?
    d. Why you think it is interesting?

One interesting application of an Artificial Neural Network (ANN) is the use of deep learning for object recognition in self-driving cars.

a. What it is being used for:

Deep learning neural networks are used in self-driving cars to enable the car to recognize and classify different objects in its environment, such as pedestrians, other vehicles, traffic signs, and obstacles. This is an important safety feature that allows the car to make informed decisions about how to navigate the roads and avoid collisions.

b. Success/failure stats:

Deep learning neural networks have achieved impressive results in object recognition tasks, with some systems achieving near-human levels of accuracy. However, the performance of these systems can be affected by factors such as the quality and diversity of the training data, the complexity of the environment, and the robustness of the algorithms. In general, deep learning neural networks have shown strong performance in object recognition tasks and have been successfully applied in a variety of self-driving car applications.

c. Type of ANN and methods:

The deep learning neural network used for object recognition in self-driving cars is a type of Convolutional Neural Network (CNN). CNNs are particularly well-suited for image recognition tasks, as they are able to learn and recognize patterns and features in images using convolutional layers that process the image in a hierarchical manner. CNNs may also employ other techniques such as pooling, activation functions, and dropout to improve their performance and prevent over fitting.

d. Why it is interesting:

The use of deep learning neural networks for object recognition in self-driving cars is an interesting application of ANNs because it demonstrates the potential for these systems to perform complex tasks that require a high level of perception and decision-making. It also highlights the importance of robust and

reliable algorithms in safety-critical applications, and the need to continue improving and testing these systems to ensure their reliability and performance.

Lab Exercise

In [2]:
```
# https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6
import numpy as np
```

In [1]:
```
class NeuralNetwork:
    def __init__(self, x, y):
        self.input      = x
        self.weights1   = np.random.rand(self.input.shape[1],4)
        self.weights2   = np.random.rand(4,1)
        self.y          = y
        self.output     = np.zeros(self.y.shape)

    def feedforward(self):
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.output = sigmoid(np.dot(self.layer1, self.weights2))

    def backprop(self):
        # application of the chain rule to find derivative of the loss function with respect to weights2 and weights1
        d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) * sigmoid_derivative(self.output)))
        d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) * sigmoid_derivative(self.output), self.weights2.T)

        # update the weights with the derivative (slope) of the loss function
        self.weights1 += d_weights1
        self.weights2 += d_weights2
```

In [1]:
```
#  https://towardsdatascience.com/creating-neural-networks-from-scratch-in-python-6f02b5dd911
import numpy as np
```

In [2]:
```
# Activation Functions
def tanh(x):
    return np.tanh(x)

def d_tanh(x):
    return 1 - np.square(np.tanh(x))

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def d_sigmoid(x):
    return (1 - sigmoid(x)) * sigmoid(x)

# Loss Functions
def logloss(y, a):
    return -(y*np.log(a) + (1-y)*np.log(1-a))

def d_logloss(y, a):
    return (a - y)/(a*(1 - a))


# The Layer class
class Layer:

    activationFunctions = {
        'tanh': (tanh, d_tanh),
        'sigmoid': (sigmoid, d_sigmoid)
    }

    learning_rate = 0.1

    def __init__(self, inputs, neurons, activation):
        self.W = np.random.randn(neurons, inputs)
        self.b = np.zeros((neurons, 1))
        self.act, self.d_act = self.activationFunctions.get(activation)

    def feedforward(self, A_prev):
        self.A_prev = A_prev
        self.Z = np.dot(self.W, self.A_prev) + self.b
        self.A = self.act(self.Z)
        return self.A

    def backprop(self, dA):
        dZ = np.multiply(self.d_act(self.Z), dA)
        dW = 1/dZ.shape[1] * np.dot(dZ, self.A_prev.T)
        db = 1/dZ.shape[1] * np.sum(dZ, axis=1, keepdims=True)
        dA_prev = np.dot(self.W.T, dZ)

        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return dA_prev
```

In [3]:
```
# Training the network
# x_train = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # dim x m
# y_train = np.array([[0, 1, 1, 0]]) # 1 x m

x_train = np.array([[0, 0, 1, 1], [0, 1, 0, 1], [1, 1, 1, 1]]) # dim x m
y_train = np.array([[0, 1, 1, 0]]) # 1 x m

m = 4
epochs = 1500

# layers = [Layer(2, 3, 'tanh'), Layer(3, 1, 'sigmoid')] # original
layers = [Layer(3, 4, 'tanh'), Layer(4, 1, 'sigmoid')]
costs = [] # to plot graph

for epoch in range(epochs):
    # Feedforward
    A = x_train
    for layer in layers:
        A = layer.feedforward(A)

    # Calulate cost to plot graph
    cost = 1/m * np.sum(logloss(y_train, A))
    costs.append(cost)

    # Backpropagation
    dA = d_logloss(y_train, A)
    for layer in reversed(layers):
        dA = layer.backprop(dA)


# Making predictions
A = x_train
for layer in layers:
    A = layer.feedforward(A)
print(A)
```

```
[[0.02627813 0.97483457 0.9828563  0.02832823]]
```

Week 11

Task: What else are we doing with machine vision?

1. Research the very latest applications of Machine Vision from a Machine Learning perspective and document information about one interesting application.

One of the latest applications of machine vision from a machine learning perspective is the use of deep learning for automatic image captioning.

Automatic image captioning is a machine learning task that involves generating a natural language description of an image based on its content. This is a challenging task that requires the machine learning model to understand the visual content of the image and to generate a coherent and descriptive caption that accurately reflects the contents of the image.

Deep learning has shown promising results in the field of automatic image captioning, with some systems achieving near-human levels of performance. This is due to the ability of deep learning models to learn

and extract high-level features from images, and to generate natural language descriptions based on these features.

One example of an interesting application of automatic image captioning is in the field of assistive technology for people with visual impairments. By generating natural language descriptions of images, automatic image captioning can help people with visual impairments to better understand and interact with their environment, and to access visual information that would otherwise be difficult or impossible for them to see.

Some examples of the types of images that automatic image captioning systems can be used to describe include photographs, diagrams, maps, and charts. These systems can be trained on large datasets of labelled images and captions, and can then be used to generate captions for new images.

Overall, the use of deep learning for automatic image captioning is an interesting application of machine vision that has the potential to significantly improve the accessibility of visual information and to enhance the quality of life for people with visual impairments.

Lab Exercise

In [1]:
```
# univariate cnn example
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [2]:
```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

In [3]:
```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

In [4]:
```
# fit model
model.fit(X, y, epochs=1000, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[[103.96351]]
```

In [5]:
```
y
```

Out[5]:
```
array([40, 50, 60, 70, 80, 90])
```

In [7]:
```
# multivariate cnn example
#  multiple series input
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [8]:
```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [9]:
```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
```

In [10]:
```
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
```

In [11]:
```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

In [12]:
```
# fit model
model.fit(X, y, epochs=1000, verbose=0)
# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[[207.43318]]
```

In [13]:
```
# multivariate output 1d cnn example
from numpy import array
from numpy import hstack
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [15]:
```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [16]:
```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# separate output
y1 = y[:, 0].reshape((y.shape[0], 1))
y2 = y[:, 1].reshape((y.shape[0], 1))
y3 = y[:, 2].reshape((y.shape[0], 1))
```

In [17]:
```
# define model
visible = Input(shape=(n_steps, n_features))
cnn = Conv1D(filters=64, kernel_size=2, activation='relu')(visible)
cnn = MaxPooling1D(pool_size=2)(cnn)
cnn = Flatten()(cnn)
cnn = Dense(50, activation='relu')(cnn)
# define output 1
output1 = Dense(1)(cnn)
# define output 2
output2 = Dense(1)(cnn)
# define output 3
output3 = Dense(1)(cnn)
# tie together
model = Model(inputs=visible, outputs=[output1, output2, output3])
model.compile(optimizer='adam', loss='mse')
```

In [18]:
```
# fit model
model.fit(X, [y1,y2,y3], epochs=2000, verbose=0)
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[array([[100.10101]], dtype=float32), array([[105.145805]], dtype=float32), array([[205.17572]], dtype=float32)]
```

In [ ]:
```
```

In [19]:
```
# multivariate output 1d cnn example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [20]:
```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [21]:
```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
```

In [22]:
```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
```

In [23]:
```
# fit model
model.fit(X, y, epochs=3000, verbose=0)
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000023316A82700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
[[100.85259 106.43061 207.58759]]
```

In [24]:
```
#  Multistep CNN
# univariate multi-step vector-output 1d cnn example
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [25]:
```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [26]:
```
# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# split into samples
X, y = split_sequence(raw_seq, n_steps_in, n_steps_out)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

In [27]:
```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse')
```

In [28]:
```
# fit model
model.fit(X, y, epochs=2000, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000023316A82E50> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
[[102.84141 114.66837]]
```

In [29]:
```
# multivariate output multi-step 1d cnn example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

In [30]:
```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

In [31]:
```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# convert into input/output
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
# flatten output
n_output = y.shape[1] * y.shape[2]
y = y.reshape((y.shape[0], n_output))
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
```

In [32]:
```
# define model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(n_output))
model.compile(optimizer='adam', loss='mse')
```

In [33]:
```
# fit model
model.fit(X, y, epochs=7000, verbose=0)
# demonstrate prediction
x_input = array([[60, 65, 125], [70, 75, 145], [80, 85, 165]])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

```
[[ 90.02107  95.10845 185.14993 100.13571 105.15818 205.19562]]
```