Q1. Write a program to find union and intersection of a fuzzy set.

Union :

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Union of them, then for every member of A and B, Y will be:

```
degree_of_membership(Y)= max(degree_of_membership(A),
                             degree_of_membership(B))
```

Code :

```
A = dict()
B = dict()
Y = dict()

A = { "a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = { "a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print( 'The First Fuzzy Set is :' , A )
print( 'The Second Fuzzy Set is :' ,  B )

for A_key, B_key in zip(A, B):
    A_value = A[A_key]
    B_value = B[B_key]

 if A_value > B_value:
    Y[A_key] = A_value
 else:
     Y[B_key] = B_value

print('Fuzzy Set Union is :', Y )
```

Output:

```
The First Fuzzy Set is: {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is: {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union is: {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
```

Intersection :

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Intersection of them, then for every member of A and B, Y will be:

```
degree_of_membership(Y)= min(degree_of_membership(A),
                             degree_of_membership(B))
```

Code:

```
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)


for A_key, B_key in zip(A, B):
 A_value = A[A_key]
 B_value = B[B_key]

 if A_value < B_value:
 Y[A_key] = A_value
 else:
 Y[B_key] = B_value
print('Fuzzy Set Intersection is :', Y)
```

Output:

```
The First Fuzzy Set is: {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is: {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Intersection is: {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd':
0.5}
```

Q2. Write a program to find complement and difference of a fuzzy set.

Complement :

Consider a Fuzzy Sets denoted by A, then let's consider Y be the Complement of it, then for every member of A, Y will be:

```
degree_of_membership(Y)= 1 - degree_of_membership(A)
```

Implementation

```
A = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

print('The Fuzzy Set is :', A)

for A_key in A:
 Y[A_key] = 1 - A[A_key]

print('Fuzzy Set Complement is :', Y)
```

Output

```
The Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
Fuzzy Set Complement is : {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
```

Difference:

Consider 2 Fuzzy Sets denoted by A and B, then let's consider Y be the Intersection of them, then for every member of A and B, Y will be:

```
degree_of_membership(Y)= min(degree_of_membership(A), 1-
degree_of_membership(B))
```

Implementation

```
A = dict()
B = dict()
Y = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

print('The First Fuzzy Set is :', A)
print('The Second Fuzzy Set is :', B)

for A_key, B_key in zip(A, B):
 A_value = A[A_key]
 B_value = B[B_key]
 B_value = 1 - B_value

 if A_value < B_value:
 Y[A_key] = A_value
 else:
 Y[B_key] = B_value

print('Fuzzy Set Difference is :', Y)
```

Output

```
The First Fuzzy Set is: {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}
The Second Fuzzy Set is: {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}
Fuzzy Set Difference is: {"a": 0.1, "b": 0.1, "c": 0.6, "d": 0.5}
```

Q3. Write a Program to find Core, Support and crossover points of a fuzzy set.

Implementation

```
import numpy as np

def find_core(fuzzy_set):
 core = []
 for i, value in enumerate(fuzzy_set):
 if value == 1.0:
 core.append(i)
 return core

def find_support(fuzzy_set):
 support = []
 for i, value in enumerate(fuzzy_set):
 if value > 0.0:
 support.append(i)
 return support

def find_crossover_points(fuzzy_set):
 crossover_points = []
 for i in range(1, len(fuzzy_set)):
 if fuzzy_set[i] == 0.5:
 crossover_points.append(i)
 return crossover_points


fuzzy_set = np.array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2, 0.0])


core = find_core(fuzzy_set)
print("Core:", core)


support = find_support(fuzzy_set)
print("Support:", support)


crossover_points = find_crossover_points(fuzzy_set)
print("Crossover points:", crossover_points)
```

Output

```
Core: [5 6] Support: [0 1 2 3 4 5 6 7 8 9 10] Crossover points: [4 7]
```

Q4. Write a Program to find if the given fuzzy set is symmetric or not.

Code

Implementation

```
import numpy as np

def is_symmetric(fuzzy_set):
 if len(fuzzy_set) % 2 == 0:
 midpoint = int(len(fuzzy_set) / 2)
 left_half = fuzzy_set[:midpoint]
 right_half = fuzzy_set[midpoint:]
 reversed_right_half = right_half[::-1]
 return np.array_equal(left_half, reversed_right_half)
 else:
 midpoint = int(len(fuzzy_set) / 2)
 left_half = fuzzy_set[:midpoint]
 right_half = fuzzy_set[midpoint + 1:]
 reversed_right_half = right_half[::-1]
 return np.array_equal(left_half, reversed_right_half)

fuzzy_set = np.array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2, 0.0])

if is_symmetric(fuzzy_set):
 print("The fuzzy set is symmetric.")
else:
 print("The fuzzy set is not symmetric.")
```

Code

```
The fuzzy set is Symmetric.
```

Q5. Write a Program to create fuzzy relation by Cartesian product of any two fuzzy sets.

Implementation

```
import numpy as np

def create_fuzzy_relation(fuzzy_set1, fuzzy_set2):
 fuzzy_relation = np.zeros((len(fuzzy_set1), len(fuzzy_set2)))
 for i, value1 in enumerate(fuzzy_set1):
 for j, value2 in enumerate(fuzzy_set2):
 fuzzy_relation[i, j] = min(value1, value2)
 return fuzzy_relation

fuzzy_set1 = np.array([0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2, 0.0])
fuzzy_set2 = np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0])

fuzzy_relation = create_fuzzy_relation(fuzzy_set1, fuzzy_set2)

print("Fuzzy relation:")
print(fuzzy_relation)
```

```
Fuzzy relation: [[0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0]
[0.0 0.1 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.0]
[0.0 0.1 0.2 0.3 0.4 0.4 0.4 0.3 0.2 0.1 0.0]
[0.0 0.1 0.2 0.3 0.4 0.5 0.4 0.3 0.2 0.1 0.0]
                 [0.0 0.1 0.2 0.3 0.4 0.5 0.4 0.3 0.2 0.1 0.0]
                 [0.0 0.1 0.2 0.3 0.4 0.5 0.4 0.3 0.2 0.1 0.0]
                 [0.0 0.1 0.2 0.3 0.4 0.5 0.4 0.3 0.2 0.1 0.0]
                 [0.0 0.1 0.2 0.3 0.4 0.4 0.4 0.3 0.2 0.1 0.0]
                 [0.0 0.1 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.0]
                [0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0]]
```

Q6. Write a program to perform max-min composition on any two fuzzy relations.

Implementation

```
import numpy as np

def max_min_composition(matrix1, matrix2):

 if matrix1.shape[1] != matrix2.shape[0]:
 raise ValueError("Matrices must have compatible dimensions")

 result_matrix = np.zeros((matrix1.shape[0], matrix2.shape[1]))

 for i in range(matrix1.shape[0]):
 for j in range(matrix2.shape[1]):
 maximum = -np.inf
 for k in range(matrix1.shape[1]):
 maximum = max(maximum, min(matrix1[i, k], matrix2[k, j]))
 result_matrix[i, j] = maximum

 return result_matrix

matrix1 = np.array([[0.5, 0.3, 0.2], [0.8, 0.6, 0.4], [0.9, 1.0, 0.7]])
matrix2 = np.array([[0.7, 0.5, 0.3], [0.6, 0.4, 0.2], [0.5, 0.3, 0.1]])

result_matrix = max_min_composition(matrix1, matrix2)

print("Resulting matrix:")
print(result_matrix)
```

```
Resulting matrix: [[0.6 0.4 0.2]
                   [0.7 0.5 0.3]
                   [0.7 0.6 0.4]]
```

Q7. Reema And Sameer submitted the CS Assignments Reema.txt and Sameer.txt, respectively. Sameer has the nature of copying Reema's assignment. Use Neural Network Algorithm to detect the percentage of plagiarism in Sameer's assignment.

Implementation

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

doc_1 = "Data is the oil of the digital economy"
doc_2 = "Data is a new oil"
data = [doc_1, doc_2]

count_vectorizer = CountVectorizer()
vector_matrix = count_vectorizer.fit_transform(data)
tokens = count_vectorizer.get_feature_names_out()

def create_dataframe(matrix, tokens):
doc_names = [f'doc_{i+1}' for i, _ in enumerate(matrix)]
df = pd.DataFrame(data=matrix.toarray(), index=doc_names, columns=tokens)
return df

doc_names = [f'doc_{i+1}' for i in range(len(data))]
term_document_matrix = create_dataframe(vector_matrix, tokens)

cosine_similarity_matrix = cosine_similarity(vector_matrix)
cosine_similarity_df = pd.DataFrame(cosine_similarity_matrix, index=doc_names,
columns=doc_names)

print ("Term-Document Matrix:")
print (term_document_matrix)
print ("\nCosine Similarity Matrix:")
print (cosine_similarity_df)

create_dataframe(cosine_similarity_matrix,['doc_1','doc_2'])
```

```
Term-Document Matrix:
          data digital economy is new of oil the
doc_1     1       1         1      1   0   1   1  2
doc_2     1       0         0      1   1   0   1  0
Cosine Similarity Matrix:
         doc_1     doc_2
doc_1 1.000000 0.474342
doc_2 0.474342 1.000000
```

8. Write a program to implement multi-layer perceptron model for social media dataset.

Implementation

```
import tensorflow as tf
from tensorflow import keras


(x_train, y_train), (x_test, y_test) =

model = keras.Sequential([
 keras.layers.Dense(64, activation='relu', input_shape=(x_train.shape[1],)),
 keras.layers.Dense(32, activation='relu'),
 keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Q9. Write a program to implement Recurrent Neural Network for Health care dataset.

Implementation

```
import tensorflow as tf
from tensorflow import keras

(x_train, y_train), (x_test, y_test) = ...

model = keras.Sequential([
 keras.layers.Embedding(input_dim=max_vocab_size, output_dim=embedding_dim,
input_length=max_length),
 keras.layers.LSTM(64, return_sequences=True),
 keras.layers.LSTM(32),
 keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
 loss='binary_crossentropy',
 metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Example

```
new_patient_data = ...

new_patient_data_preprocessed = preprocess(new_patient_data)

new_patient_data_input = tf.expand_dims(new_patient_data_preprocessed, axis=0)

prediction = model.predict(new_patient_data_input)

if prediction > 0.5:
 print ('Predicted health outcome: Positive')
else:
 print('Predicted health outcome: Negative')
```

Q10. Write a program to solve Travelling Salesman Problem using Genetic Algorithm.

Implementation

```
import numpy as np
import random

def generate_random_route(number_of_cities):
 route_list = list(range(number_of_cities))
 random.shuffle(route_list)
 return route_list

def calculate_distance(route, distance_matrix):
 total_distance = 0
 for i in range(len(route)):
 next_city = route[(i + 1) % len(route)]
 current_city = route[i]
 distance = distance_matrix[current_city, next_city]
 total_distance += distance
 return total_distance

def selection(population, fitness_values, elite_size):
 selected_population = []
 sorted_fitness_indices = np.argsort(fitness_values)

 for i in range(elite_size):
 selected_population.append(population[sorted_fitness_indices[-i - 1]])

 for _ in range(len(population) - elite_size):
 random_fitness_index = np.random.choice(len(population), p=fitness_values /
np.sum(fitness_values))
 selected_population.append(population[random_fitness_index])

 return selected_population


def crossover(selected_population, crossover_rate):
 offspring_population = []
 for i in range(0, len(selected_population), 2):
 parent1 = selected_population[i]
 parent2 = selected_population[i + 1]

 crossover_point = random.randint(1, len(parent1) - 1)

 offspring1 = parent1[:crossover_point] + parent2[crossover_point:]
 offspring2 = parent2[:crossover_point] + parent1[crossover_point:]
```

```
 if random.random() < crossover_rate:
 offspring1, offspring2 = offspring2, offspring1

 offspring_population.append(offspring1)
 offspring_population.append(offspring2)

 return offspring_population

def mutation (offspring_population, mutation_rate):
 mutated_population = []
 for offspring in offspring_population:
 mutated_offspring = offspring.copy()

 mutation_points = random.sample(range(len(offspring)), int (mutation_rate *
len(offspring)))

 for mutation_point in mutation_points:
 swap_index = random.randint(0, len(offspring) - 1)
 while swap_index == mutation_point:
 swap_index = random.randint(0, len(offspring) - 1)

 mutated_offspring[mutation_point], mutated_offspring[swap_index] =
mutated_offspring[swap_index], mutated_offspring[mutation_point]

 mutated_population.append(mutated_offspring)

 return mutated_population

def genetic_algorithm(distance_matrix, population_size, elite_size, crossover_rate,
mutation_rate, max_generations):

 population = [generate_random_route(len(distance_matrix)) for _ in
range(population_size)]

 for generation in range(max_generations):
 fitness_values = []
 for route in population:
 fitness_values.append(1 / calculate_distance(route, distance_matrix))
 selected_population = selection(population, fitness_values, elite_size)
 offspring_population = crossover(selected_population, crossover_rate)
 mutated_population = mutation(offspring_population, mutation_rate)
 population = selected_population + mutated_population
 best_route = population[np.argmax(fitness_values)]
 best_distance = 1 / np.max(fitness_values)

 return best_route, best_distance
```