МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский технический университет связи и информатики»
(МТУСИ)

Кафедра
«Сетевые информационные технологии и сервисы»

Лабораторная работа 6
По дисциплине «Интеллектуальные базы данных»
«Использование методов нечеткой логики для классификации объектов по информации о них, хранящейся в интеллектуальной базе данных»

Выполнил студент:
М092301(75) Леонов Н.Н.
Вариант 17
Проверил:
Ст.пр. Фатхулин Т.Д.

Москва 2024

1. Цель работы

Целью данной работы является построение системы, выполняющей классификацию объектов базы данных.

2. Задание

- Выбрать таблицу базы данных с исходными данными;
- Определить входные переменные и нечеткие множества для этих переменных;
- Определить выходную переменную и её нечеткое множество;
- Определить базу правил;
- Реализовать алгоритм нечеткого вывода.

3. Теоретические сведения

Нечеткая логика — это форма многозначной логики, в которой истинные значения переменных могут быть любыми действительными числами от 0 до 1 включительно. Она используется для обработки концепции «частичной истины», где истинное значение может варьироваться между полностью истинным и полностью ложным. В то время как в классической булевой логике истинные значения переменных могут быть только значениями 0 или 1.

Примеры применения нечеткой логики

− наведение телекамер в спортивных трансляциях;

− упрощенное управление роботами;

− диагностика рака;

− управление стиральной машиной;

− распознавание рукописного текста;

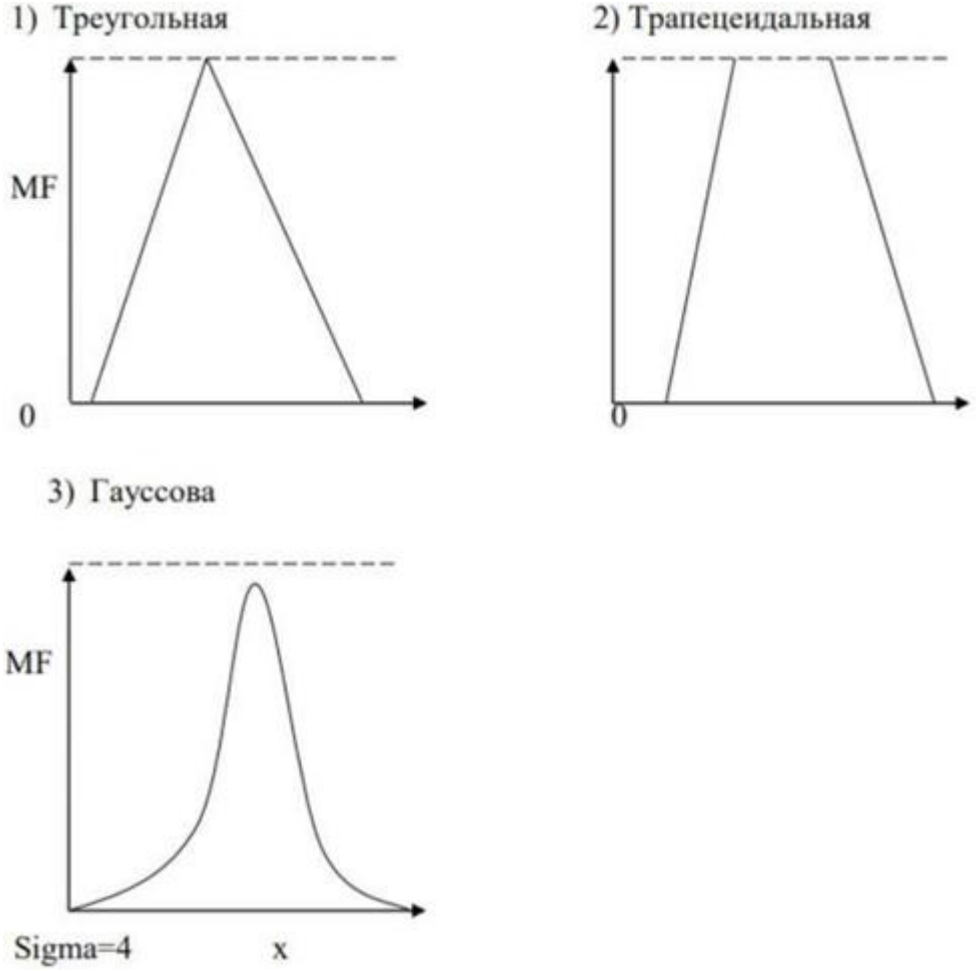
− управление «экономичной» скоростью автомобиля;

Основные понятия нечеткой логики

Функция принадлежности

Данная функция представляет степень принадлежности каждого члена пространства рассуждения к данному нечёткому множеству. Существуют множества

типовых форм для задания функций принадлежности. Наибольшее распространение получили Треугольная, Трапецеидальная и Гауссова функция принадлежности.



Нечеткое множество

Нечеткое множество – совокупность функций принадлежностей, которые составляют это множество.

Нечеткое множество (набор функций принадлежности) выбирается при проектировании системы прогнозирования экспертом в данной предметной области (или самим программистом).

При построении нечеткого множества обязательным условием является пересечение хотя бы двух функций принадлежности, иначе данные функции можно легко привести к четкости, что лишает смысла все дальнейшие вычисления.

Принцип осуществления нечеткого логического вывода Система нечеткого вывода – это процесс получения нечетких заключений о требуемом управлении объектом на основе нечетких условий или предпосылок, представляющих собой информацию о текущем состоянии объекта.

− Фаззификация (введение нечеткости);

Функции принадлежности входных переменных применяются к фактическим значениям этих переменных для определения истинности предпосылки каждого правила

− Логический вывод;

Вычисленное значение предпосылок применяется к заключению каждого правила − Композиция;

Объединение правил

− Дефаззификация

Используется, если на выходе нужно получить конкретное четкое число

Алгоритмы осуществления нечеткого логического вывода:

− Алгоритм Мамдани

− Алгоритм Цукамото

− Алгоритм Сугено

4. Выполнение лабораторной работы

1. Выбор таблицы с исходными данными

На данном этапе необходимо выбрать таблицу, записи в которой необходимо классифицировать. Таблица «Expenses2», представленная на рисунке 1.

	exp2_ID	exp2_name	exp2_limit	exp2_time	exp2_raiting	exp2_descri
1	1	main_exp	8000	10	6	NULL
2	2	add_exp	5000	7	3	NULL
3	3	dep3_exp	29000	30	7	NULL
4	4	dep2_exp	30000	20	8	NULL
5	5	dep1_exp	40000	20	10	NULL

Рисунок 1 - Таблица " Expenses2"

2. Выбор входных переменных и определение необходимых нечетких множеств.

На данном этапе необходимо выбрать переменные, которые непосредственно будут влиять на принадлежность записи к определенному классу. В нашем случае данными значениями будут являться следующие поля:

- «exp2_limit» (Денежный лимит);
- «exp2_time» (Количество дней);
- «exp2_raiting» (Популярность данного вида затрат)

Далее необходимо определить диапазон значений, которые могут принимать данные входные переменные, а также определить нечеткое множество значений и функции принадлежности для каждой из переменной.

Для контрольного примера диапазоны значений являются следующими:

- Денежный лимит – [8000; 40000];
- Время – [7; 30];
- Популярность – [6; 10]

Данные диапазоны сложно использовать, так как они сильно отличаются друг от друга. Поэтому их стоит привести к одному диапазону. В контрольном примере приведем значения всех входных переменных к диапазону [0; 10]:

Денежный лимит:

– Сместим диапазон на 8000 влево, получим: [8000-8000; 40000-8000] = [0; 32000];

– Разделим весь диапазон на 3200, получим: [0/3200; 32000/3200] = [0; 10]

Время:

– Сместим диапазон на 7 влево, получим: [7-7; 30-7] = [0; 23];

– Разделим весь диапазон на 2,3, получим: [0/2,3; 23/2,3] = [0; 10]

Популярность:

– Сместим диапазон на 7 влево, получим: [6-6; 10-6] = [0; 4];

– Разделим весь диапазон на 0,4 получим: [0/0,4; 4/0,4] = [0; 10]

Определим нечеткое множество значений каждой входной переменной:

limit = [Небольшой, Средний, Высокий]

time = [Мало, Средне, Много]

raiting = [Низкий, Средний, Большой]

class = [Плохо, Ниже среднего, Выше среднего, Отлично]

Для простоты используем одно нечеткое множество для каждой входной переменной. Данное множество является совокупностью Гауссовских функций принадлежности и представлена на рисунке 2. Функция Гаусса задается следующим уравнением:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

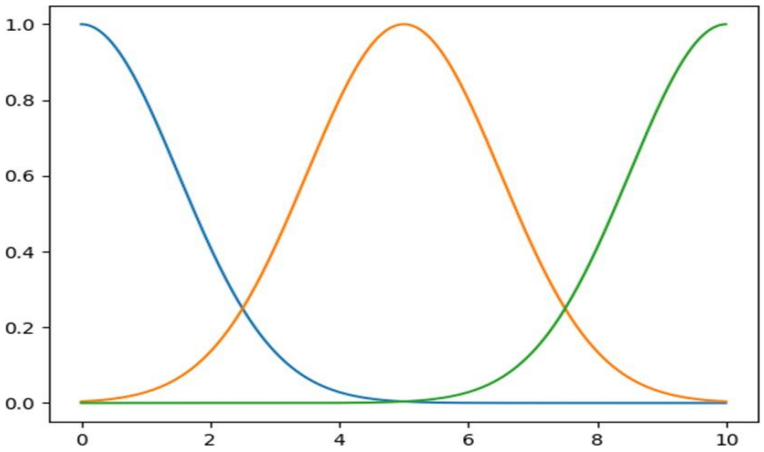


Рисунок 2 - нечеткое множество для входной переменной

Данные графики строятся из следующих уравнений:

– Синий график: $f(x) = e - x$ 2 4.5

– Оранжевый график: $f(x) = e - (x-5)$ 2 4.5

– Зеленый график: $f(x) = e - (x-10)\ 2\ 4.5\ 3$.

Определение выходной переменной и ее функции принадлежности

В качестве выходной переменной выступает класс, к которому система отнесет определенную запись в таблице. Данная переменная принимает следующее нечеткое множество значений: [Плохо, ниже среднего, выше среднего, отлично]. Нечеткое множество, определяющее данную переменную, также представляет из себя набор Гауссовских функций и представлено на рисунке 3

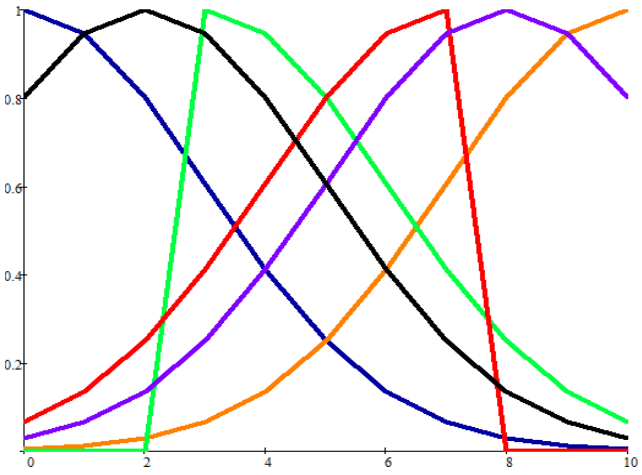


Рисунок 3 - нечеткое множество для выходной переменной

Программный код:

```
from getpass import getpass
from mysql.connector import connect, Error
import numpy as np

class Fuzzy:
# Конструктор класса
    def __init__(self, inp: str, inFuncs: list, outFuncs: list, outRevFuncs: list) -> None:
        self. inp = inp # Вводная строка
        self. rules = list() # Выходной список
        self. inFuncs = inFuncs.copy() # Список функций, входящих в ФП входной переменной
        self. outFuncs = outFuncs.copy() # Список функций, входящих в ФП выходной переменной
        self. outRevFuncs = outRevFuncs.copy() # Список обратных функций (f(y)), входящих в ФП
выходной переменной
        self. queryResult = list() # Результат запроса
        self. inData = list() # Преобразованные данные на вход алгоритма
        # Преобразование строки базы правил в список для дальнейшей работы
    def rulesStringToList(self) -> list:
    # Разбиение входной строки по символам переноса строки
        self. rules = self.__inp.split("\n")
        # Преобразование каждой строки в список чисел
        for i in range(len(self.__rules)):
            self. rules[i] = [int(x) for x in self. rules[i]]
    # Генерация понятной базы правил
    def generateRules(self) -> None:
```

```
        """
            limit = [Небольшой, Средний, Высокий]
            time = [Мало, Средне, Много]
            raiting = [Низкий, Средний, Большой]
            class = [Плохо, Ниже среднего, Выше среднего, Отлично]
        """

        t = list()
        i = 0
        for element in self.__rules:
            t.clear()
            i += 1
            t.append(["Небольшой", "Средний", "Высокий"][element[0] - 1])
            t.append(["Мало", "Средне", "Много"][element[1] - 1])
            t.append(["Низкий", "Средний", "Большой"][element[2] - 1])
            t.append(["Плохо", "Ниже среднего", "Выше среднего", "Отлично"][element[3] - 1])
            print(f"ПРАВИЛО №{i}: ЕСЛИ limit=({t[0]}) И time=({t[1]}) И raiting=({t[2]}) ТОГДА
class=({t[3]})")
    # Получение данных из БД
    def getDBData(self):
        try:
            # Подключение к БД
            with connect(
                host="localhost",
                user=input("Введите имя пользователя БД (default=root): "),
                password=getpass("Пароль пользователя: "),
                database="zp"
            ) as connection:
            # Выполняемый запрос
                dbQuery = "SELECT * FROM salaries"
            # Преобразование результата запроса в список
                with connection.cursor() as cursor:
                    cursor.execute(dbQuery)
                    for db in cursor:
                        self. queryResult.append(db)
        except Error as e:
            print(e)
    # Преобразование входных данных перед использованием
    def preProcess(self) -> None:
        for element in self.__queryResult:
            limit = (float(element[4]) - 8000) / 3200
            time = (float(element[5]) - 7) / 2.3
            raiting = (float(element[6]) - 6) / 0.4
            # Проверка что данные не выходят за интервал [0; 10]
            if not 0 <= limit <= 10 or not 0 <= time <= 10 or not 0 <= raiting <= 10:
                print([limit, time, raiting])
                print("Ошибка генерации входных значений")
                return self. inData.append([ (float(element[4]) - 8000) / 3200, (float(element[5]) - 7) / 2.3,
                    (float(element[6]) - 6) / 0.4])
    # Выполнение основного алгоритма
    def process(self) -> None:
        # Если на этапе преобразования данных произошла ошибка, то длина списков будет разной
        if len(self.__queryResult) != len(self.__inData):
            print("На этапе генерации входных данных произошла ошибка")
            return
```

```
        i = 0
        print("\nРезультат выполнения алгоритма:\n")
        for inData in self. inData:
            upper = 0
            lower = 0
            funcs = self. inFuncs
            outFuncs = self. outFuncs
            outRevFuncs = self. outRevFuncs
            for rule in self. rules:
                fuzzy = [funcs[rule[i] - 1](inData[i]) for i in range(len(inData))]
                minFuzzy = min(fuzzy)
                z0 = outRevFuncs[rule[-1] - 1](minFuzzy)
                if z0 > 10:
                    z0 = 10
                if z0 < 0:
                    z0 = 0
                upper += (minFuzzy*z0)
                lower += minFuzzy

            x = upper / lower
            temp = [f(x) for f in outFuncs]
            answer = max(temp)
            print(f'[{self. queryResult[i][1]} {self. queryResult[i][2]}
            {self. queryResult[i][3]}] - {["Плохо", "Ниже среднего", "Выше среднего",
"Отлично"][temp.index(answer)]}' )
            i += 1

    def run(self):
        self.generateRules()    # Вывод баз правил
        self.rulesStringToList()    # Преобразование правил из строки список
        self.getDBData()    # Получение данных из БД
        self.preProcess()   # Преобразование данных
        self.process()

    # Гауссовское (нормальное) распределение
gauss = lambda k, x, phi, sigma: k / (sigma*np.sqrt(2*np.pi)) * np.exp(- 1/2* np.power((x-phi)/sigma, 2))

# Обратная функция Гаусса (f(y)). "Left" и "Right" потому, что прямая y = k пересекает функцию
дважды,
# Поэтому учитывается либо левый кусок функции, либо правый

revGaussLeft = lambda k, y, phi, sigma: -np.sqrt(-2 * np.power(sigma, 2)*
np.log(y*sigma*np.sqrt(2*np.pi)/k)) + phi

revGaussRight = lambda k, y, phi, sigma: np.sqrt(-2 * np.power(sigma, 2)*
np.log(y*sigma*np.sqrt(2*np.pi)/k)) + phi

# Входное нечеткое множество
inputM = lambda x: gauss(1 / gauss(1, 0, 0, 1.5), x, 0, 1.5)
inputS = lambda x: gauss(1 / gauss(1, 0, 0, 1.5), x, 5, 1.5)
inputD = lambda x: gauss(1 / gauss(1, 0, 0, 1.5), x, 10, 1.5)

# Выходное нечеткое множество
output1 = lambda x: gauss(1 / gauss(1, 0, 0, 3), x, 0, 3)
output2 = lambda x: gauss(1 / gauss(1, 0, 0, 3), x, 2, 3)
```

```
output5 =  lambda  x:  gauss(1 /  gauss(1,   0,  0,  3), x, 8, 3)
output6 =  lambda  x:  gauss(1 /  gauss(1,   0,  0,  3), x, 10, 3)
output7 =  lambda  x:  gauss(1 /  gauss(1,   0,  0,  2.2), x, 3, 2.2) if x >= 3 else 0
output8 =  lambda  x:  gauss(1 /  gauss(1,   0,  0,  2.2), x, 7, 2.2) if x <= 7 else 0

# Обратное выходное нечеткое множество
revOutput1 = lambda y: revGaussRight(1 / gauss(1, 0, 0, 3), y, 0, 3)
revOutput2 = lambda y: revGaussRight(1 / gauss(1, 0, 0, 3), y, 2, 3)
revOutput5 = lambda y: revGaussRight(1 / gauss(1, 0, 0, 3), y, 8, 3)
revOutput6 = lambda y: revGaussLeft(1 / gauss(1, 0, 0, 3), y, 10, 3)
revOutput7 = lambda y: revGaussRight(1 / gauss(1, 0, 0, 2.2), y, 3, 2.2)
revOutput8 = lambda y: revGaussLeft(1 / gauss(1, 0, 0, 2.2), y, 7, 2.2)

      # База правил
inp = """1111
1123
1134
2111
2123
2134
2212
2223
2234
3111
3123
3134
3211
3222
16
3234
3312
3322
3333"""
inFuncs = [inputM, inputS, inputD]
outFuncs = [output1, output2, output7,output8, output5, output6]
outRevFuncs = [revOutput1, revOutput2,revOutput7,revOutput8, revOutput5,
revOutput6]
# Создание объекта основного класса
payload = Fuzzy(inp, inFuncs, outFuncs, outRevFuncs)
# Запуск
payload.run()
```

В качестве результата необходимо получить соответствие строки в выбранной таблице и её класса, который выдал алгоритм. Пример результата выполнения алгоритма представлен на рисунке 4.

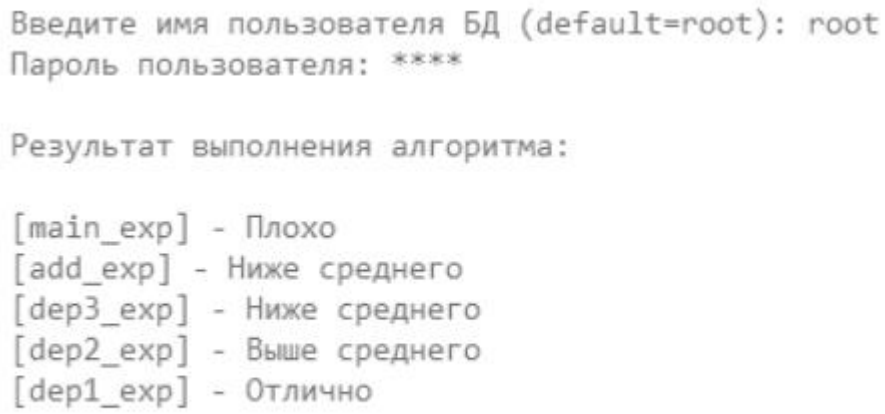


Рисунок 4 - Результат выполнения алгоритма

5. Выводы

В ходе лабораторной работы были использованы методов нечеткой логики для классификации объектов по информации о них, хранящейся в интеллектуальной базе данных.