ЛР 2. Реализация систем нечеткого вывода на языке Python

**Цель** лабораторной работы – получение навыков создания систем нечеткого вывода (fuzzy inference system, FIS) средствами языка Python.

**По ЛР 2 оформляется отчет в электронном виде**.

**Общие сведения**

Для работы с FIS рекомендуется применить библиотеку skfuzzy. Для ее инсталяции в среде Google Colab применяется команда:

!pip install -U scikit-fuzzy

– она запускается вначале работы блокнота, потом ее надо закомментировать.

При работе в пакетном менеджере Anaconda команду pip применять запрещено, поэтому используется команда:

!conda install -U scikit-fuzzy

Отметим, что существуют разные библиотеки Python для работы с нечеткими системами, поэтому можно, на свое усмотрение, применить какую-либо из них самостоятельно разобравшись в особенностях синтаксиса. Например, есть библиотека FuzzyWuzzy, с помощью которой можно выполнять нечёткое сравнение строк на основе расстояние Левенштейна (<https://habr.com/ru/articles/491448/> ).

Освоение приемов создания FIS в skfuzzy рассмотрим на примере.

ПРИМЕР. Разработать FIS на основе алгоритма Mamdani, структура которой показана на рисунке 1.

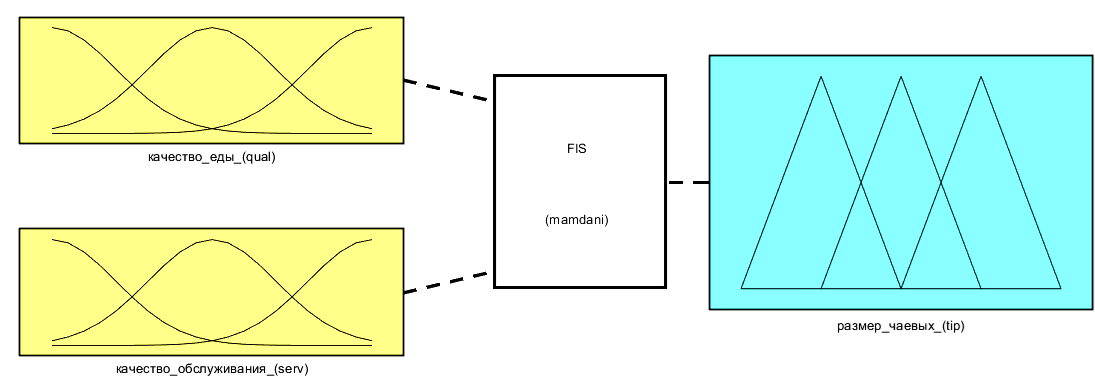


Рисунок 1– Структура FIS примера

Отметим, изображение структуры FIS создано в MatLAB, так как библиотек Python, поддерживающих визуальные инструменты разработки FIS нет – по крайней мере, они пока не встречались.

Пусть, FIS должна содержать, три правила (П), например, такие:

П1: "Если **еда** плохая ИЛИ **обслуживание** плохое, ТО **чаевые** маленькие"

П2: "Если **обслуживание** среднее, ТО **чаевые** средние"

П3: "Если **еда** высокого качества ИЛИ **обслуживание** высокого качества, ТО **чаевые** высокие

В результате работы программы должен выводиться на экран размер чаевых, в % от суммы заказа. Входные переменные прописывать будем непосредственно в коде программы, текст которой представлен в листинге.

Далее следует копировать фрагменты программы (на фрагменты разбивайте самостоятельно) и выполнять их, попутно разбираясь в синтаксисе и смысле проводимых программой действий – если нет Интернета и работа проводиться локально в Anaconda, в противном случае можно воспользоваться ссылкой на [ноутбук с программой](https://colab.research.google.com/drive/1MYe00iMnT3WYuBehZSg_yh2AMV_ADXmP?usp=sharing) и по ней разбираться. После этого выполнить индивидуальное задание, приведенное ниже.

**Листинг** – Программа реализации FIS

!pip install -U scikit-fuzzy  # в первый запуск разкоментировать и нсталировать эту библиотеку

import numpy as np

import skfuzzy as fuzz

import matplotlib.pyplot as plt

#Генерация переменных окружения:

x\_qual = np.arange(0, 11, 1)  # качество еды, в диапазоне от 0 до 11 с шагом 1

x\_serv = np.arange(0, 11, 1)  # качество обслуживания

x\_tip  = np.arange(0, 26, 1)  # размер чаевых в % от суммы заказа

# \_\_\_ Проектирование функций принадлежности (membership function, MF) \_\_\_

# для качества еды

qual\_lo = fuzz.trimf(x\_qual, [0, 0, 5])  # "низкое качество еды".  Первое число - начало основания.

qual\_md = fuzz.trimf(x\_qual, [0, 5, 10]) # "среднее качество еды". Второе - координата вершины треугольнка.

qual\_hi = fuzz.trimf(x\_qual, [5, 10, 10])# "высокое качество еды". Третье - правая граница основания.

 # для качества обслуживания

serv\_lo = fuzz.trimf(x\_serv, [0, 0, 5])

serv\_md = fuzz.trimf(x\_serv, [0, 5, 10])

serv\_hi = fuzz.trimf(x\_serv, [5, 10, 10])

# для размера чаевых

tip\_lo = fuzz.trimf(x\_tip, [0, 0, 13])

tip\_md = fuzz.trimf(x\_tip, [0, 13, 25])

tip\_hi = fuzz.trimf(x\_tip, [13, 25, 25])

# Визуализация функций принадлежности

fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))

ax0.plot(x\_qual, qual\_lo, 'b', linewidth=1.5, label='Bad')

ax0.plot(x\_qual, qual\_md, 'g', linewidth=1.5, label='Decent')

ax0.plot(x\_qual, qual\_hi, 'r', linewidth=1.5, label='Great')

ax0.set\_title('Food quality')

ax0.legend()

ax1.plot(x\_serv, serv\_lo, 'b', linewidth=1.5, label='Poor')

ax1.plot(x\_serv, serv\_md, 'g', linewidth=1.5, label='Acceptable')

ax1.plot(x\_serv, serv\_hi, 'r', linewidth=1.5, label='Amazing')

ax1.set\_title('Service quality')

ax1.legend()

ax2.plot(x\_tip, tip\_lo, 'b', linewidth=1.5, label='Low')

ax2.plot(x\_tip, tip\_md, 'g', linewidth=1.5, label='Medium')

ax2.plot(x\_tip, tip\_hi, 'r', linewidth=1.5, label='High')

ax2.set\_title('Tip amount')

ax2.legend()

# Какой будет уровень (level) чаевые при следующих входных данных: качество еды qual=6.5, обслуживание serv=9.8

# Для этого используем fuzz.interp\_membership

# Сначала сразу вычислим значения MF для заданных величин входных переменных

qual\_level\_lo = fuzz.interp\_membership(x\_qual, qual\_lo, 6.5)

qual\_level\_md = fuzz.interp\_membership(x\_qual, qual\_md, 6.5)

qual\_level\_hi = fuzz.interp\_membership(x\_qual, qual\_hi, 6.5)

serv\_level\_lo = fuzz.interp\_membership(x\_serv, serv\_lo, 9.8)

serv\_level\_md = fuzz.interp\_membership(x\_serv, serv\_md, 9.8)

serv\_level\_hi = fuzz.interp\_membership(x\_serv, serv\_hi, 9.8)

# Теперь формируем правила и применяем их.

# Правило 1 (П1): "Если еда плохая ИЛИ обслуживание плохое, ТО чаевые маленькие"

# ИЛИ реализуется как взятие максимального из двух значений

active\_rule1 = np.fmax(qual\_level\_lo, serv\_level\_lo)  # фазификация для П1

tip\_activation\_lo = np.fmin(active\_rule1, tip\_lo)     # импликация для П1 - срезали MF tip\_lo на уровне active\_rule1

# Правило 2 (П2): "Если обслуживание среднее, ТО чаевые средние" - еда здесь не учитывается, мы так решили

tip\_activation\_md = np.fmin(serv\_level\_md, tip\_md)  # срезали MF tip\_md на уровне serv\_level\_md

# Правило 3 (П3): "Если обслуживание высокого качества ИЛИ еда высокого качества, ТО чаевые высокие

active\_rule3 = np.fmax(qual\_level\_hi, serv\_level\_hi)

tip\_activation\_hi = np.fmin(active\_rule3, tip\_hi) # срезали MF tip\_hiна уровне active\_rule3

tip0 = np.zeros\_like(x\_tip)  # создали шаблон- масив 0 того же размера, что и x\_tip - см. строка 10

# Визуализация срабатывания праил

fig, ax0 = plt.subplots(figsize=(8, 3))

# закрасить пространство между  y1=tip0 и  y2=tip\_activation\_lo по координатам x=x\_tip

# для П1

ax0.fill\_between(x\_tip, tip0, tip\_activation\_lo, facecolor='b', alpha=0.7)

ax0.plot(x\_tip, tip\_lo, 'b', linewidth=0.5, linestyle='--', )

# для П2

ax0.fill\_between(x\_tip, tip0, tip\_activation\_md, facecolor='g', alpha=0.7)

ax0.plot(x\_tip, tip\_md, 'g', linewidth=0.5, linestyle='--')

# для П3

ax0.fill\_between(x\_tip, tip0, tip\_activation\_hi, facecolor='r', alpha=0.7)

ax0.plot(x\_tip, tip\_hi, 'r', linewidth=0.5, linestyle='--')

ax0.set\_title('Усеченные функции активации "чаевых" - выходы правил')

# Отключим верхнюю/правую рамки графиков

for ax in (ax0,):

    ax.spines['top'].set\_visible(False)

    ax.spines['right'].set\_visible(False)

    ax.get\_xaxis().tick\_bottom()

    ax.get\_yaxis().tick\_left()

plt.tight\_layout()

# КОМПОЗИЦИЯ - объединяем все результирующие MF для чаевых (см. предыдущий график) вместе

aggregated = np.fmax(tip\_activation\_lo,

                     np.fmax(tip\_activation\_md, tip\_activation\_hi))

# ДЕФАЗИФИКАЦИЯ - центроидным методом вычисляем "четкое" значения для размера чаевых в %

tip = fuzz.defuzz(x\_tip, aggregated, 'centroid')

tip\_activation = fuzz.interp\_membership(x\_tip, aggregated, tip)  # for plot

print('Размер чаевых рекомендуется, в % от суммы заказа: ',tip)

# Визуализация композиции

fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.plot(x\_tip, tip\_lo, 'b', linewidth=0.5, linestyle='--', )

ax0.plot(x\_tip, tip\_md, 'g', linewidth=0.5, linestyle='--')

ax0.plot(x\_tip, tip\_hi, 'r', linewidth=0.5, linestyle='--')

ax0.fill\_between(x\_tip, tip0, aggregated, facecolor='Orange', alpha=0.7)

ax0.plot([tip, tip], [0, tip\_activation], 'k', linewidth=1.5, alpha=0.9)

ax0.set\_title('Aggregated membership and result (line)')

# Отключим верхнюю/правую рамки графиков

for ax in (ax0,):

    ax.spines['top'].set\_visible(False)

    ax.spines['right'].set\_visible(False)

    ax.get\_xaxis().tick\_bottom()

    ax.get\_yaxis().tick\_left()

plt.tight\_layout()

**Индивидуальное задание** №**1. Разработка простой FIS.** Разработать FIS, структура которой показана на рисунке 2.

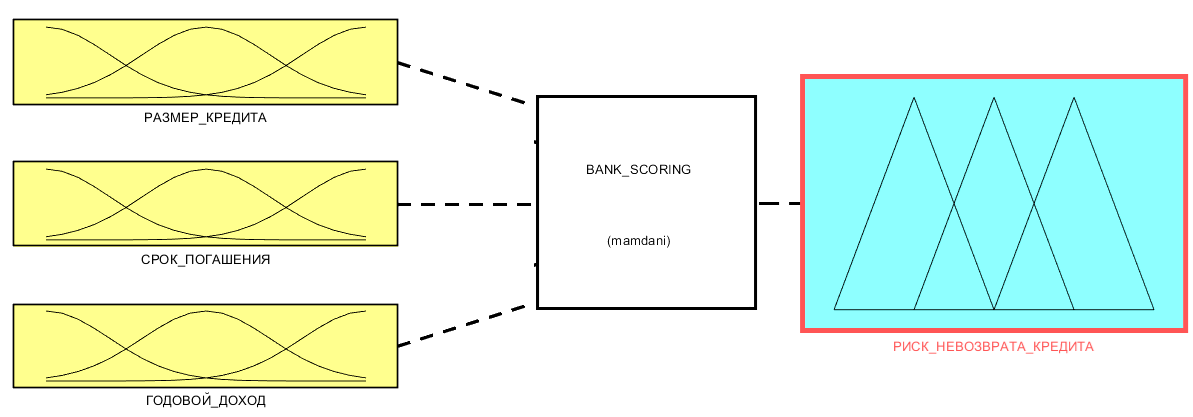


Рисунок 2 – Структура FIS индивидуального задания №1

Предполагается, что FIS реализует процедуру банковского скоринга – оценки риска невозврата кредита клиентом в зависимости от размера кредита, срока погашения и годового дохода клиента. Взять следующие диапазоны значений переменных:

– размера кредита от 10 000 до 10 000 000 млн. руб.;

– срока погашения от 1 года до 30 лет;

– годовой дохода клиента от 150 000 до 6 000 000 млн. руб.;

– риска невозврата кредита от 0 до 1.

В базе правил FIS, имеющей название BANK\_SCORING должно содержаться 4-е правила:

П1: "Если **размер кредита** большой И **срок погашения** большой И **годовой доход** большой, ТО **риск невозврата кредита** маленький"

П2: "Если **размер кредита** большой И **годовой доход** малый, ТО **риск невозврата кредита** большой"

П3: "Если **размер кредита** малый И **срок погашения** большой И **годовой доход** малый, ТО **риск невозврата кредита** маленький"

П4: "Если **размер кредита** средний И **срок погашения** средний И **годовой доход** малый, ТО **риск невозврата кредита** средний "

Используя FIS оценить риск невозврата кредита и заполнить таблицу при следующих значениях входных переменных:

|  |  |  |  |
| --- | --- | --- | --- |
| Размер кредита,  тыс. руб. | Срок погашения,  год | Годовой доход  тыс. руб. | Риск невозврата кредита |
| 21 | 3 | 190 |  |
| 560 | 9 | 330 |  |
| 2000 | 15 | 2000 |  |
| 4300 | 21 | 3100 |  |
| 9500 | 29 | 5600 |  |

Сделать вывод о корректности полученных результатов соотнеся их со здравым смыслом.

**Индивидуальное задание №2. Разработка дерева FIS**

При разработке FIS, в которых надо учитывать много параметров, применяются иерархические нечеткие системы (деревья FIS), состоящие из нескольких FIS. Это позволяет разбить трудоемкий процесс формирования правил, содержащих перебор множества сочетаний входных переменных для одной большой FIS, на несколько процессов создания правил для небольших, частных FIS, что эффективнее с точки минимизации комбинаций входных переменных, а также проще для понимания.

В задании требуется разработать FIS, структура которой показана на рисунке 3. Самостоятельно придумать диапазоны переменных и 4 правила (по 2 для FIS\_1 и FIS\_2) и рассчитать оценку вероятности невозврата кредита для какой-либо комбинации входных данных.



Рисунок 3 – Структура FIS индивидуального задания № 3

**Индивидуальное задание № 3. Работа с БД в Python.**

Разработать в среде Access базу данных, состоящую из одной таблицы:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID клиента | Фамилия клиента | Размер кредита,  тыс. руб. | Срок погашения,  год | Годовой доход  тыс. руб. |
| 1 | Петров | 21 | 3 | 190 |
| 2 | Кудалезов | 560 | 9 | 330 |
| 3 | Ахтыгадов | 2000 | 15 | 2000 |
| 4 | Ухкакаева | 4300 | 21 | 3100 |
| 5 | Угораздилов | 9500 | 29 | 5600 |

Используя разработанную FIS BANK\_SCORING оценить риск невозврата кредита указанных клиентов.

СУБД можно взять любую, на свое усмотрение и самостоятельно реализовать задание. Рекомендуется SQLite или другая «легковесная» система.

**Методические указания.** Для выполнения задания необходимо обеспечить связь БД с средой разработки программы Python. Материал ниже взят из [статьи](https://external.software/archives/12582).

Одним из ключевых преимуществ Access является простота применения и возможность создания и управления базами данных без необходимости использования серверных решений. Предполагается, что Access установлен на компьютере.

Для работы с базами данных Access в Python чаще всего используется библиотека pyodbc. Она является универсальным интерфейсом для взаимодействия с различными ODBC совместимыми базами данных.

Для установки pyodbc, выполните следующую команду:

!pip install pyodbc

***Подключение к базе данных Access***

Чтобы начать работу с базой данных Access, сначала необходимо установить соединение. В следующем примере мы покажем, как подключиться к базе данных Access:

import pyodbc

def connect\_to\_access\_db(db\_file: str) -> pyodbc.Connection:

    """Устанавливает соединение с базой данных Access."""

    connection\_string = rf'DRIVER={{Microsoft Access Driver (\*.mdb, \*.accdb)}};DBQ={db\_file};'

    return pyodbc.connect(connection\_string)

conn = connect\_to\_access\_db('database.accdb')

Этот код предоставляет функцию connect\_to\_access\_db, которая принимает путь к файлу базы данных Access и возвращает объект соединения.

***Выполнение SQL-запросов***

Различные типы SQL-запросов можно выполнять через установленное соединение. Приведем примеры выполнения INSERT, UPDATE, DELETE и SELECT запросов.

**INSERT**

def execute\_query(conn: pyodbc.Connection, query: str) -> None:

    """Выполняет SQL-запрос."""

    cursor = conn.cursor()

    cursor.execute(query)

    conn.commit()

    cursor.close()

execute\_query(conn, 'INSERT INTO users (name, age) VALUES ("John", 30)')

**UPDATE**

execute\_query(conn, 'UPDATE users SET age = 31 WHERE name = "John"')

**DELETE**

execute\_query(conn, 'DELETE FROM users WHERE name = "John"')

***Чтение данных из базы***

def fetch\_all\_users(conn: pyodbc.Connection) -> list:

    """Извлекает всех пользователей из базы."""

    cursor = conn.cursor()

    cursor.execute('SELECT \* FROM users')

    rows = cursor.fetchall()

    cursor.close()

    return rows

users = fetch\_all\_users(conn)

for user in users:

    print(user)

Этот пример показывает, как извлекать данные из таблицы users и выводить их.

***Обработка ошибок***

При работе с БД всегда важно учитывать возможные ошибки. Использование try-except поможет обработать исключительные ситуации.

try:

    conn = connect\_to\_access\_db('non\_existent\_database.accdb')

except pyodbc.Error as e:

    print(f'Ошибка соединения: {e}')

***Закрытие соединения***

Не забывайте закрывать соединение с базой данных после завершения работы. Это освобождает ресурсы и предотвращает потенциальные утечки памяти.

def close\_connection(conn: pyodbc.Connection) -> None:

    """Закрывает соединение с базой данных."""

    conn.close()

close\_connection(conn)

***Примеры интеграции с другими библиотеками***

Python предоставляет мощные инструменты для работы с данными. Например, библиотека pandas отлично дополняет базовые возможности pyodbc для анализа данных.

import pandas as pd

def access\_to\_dataframe(conn: pyodbc.Connection, query: str) -> pd.DataFrame:

    """Возвращает DataFrame из результата SQL-запроса."""

    return pd.read\_sql(query, conn)

df = access\_to\_dataframe(conn, 'SELECT \* FROM users')

print(df)

С помощью pandas можно легко преобразовать результаты SQL-запросов в DataFrame (специальные структуры данных, применяемые в data science, машинном обучении, научных вычислениях и многих других областях, связанных с активным использованием данных.) и проводить над ними аналитические операции.

Использование рассмотренных возможностей в реальных проектах поможет создавать более гибкие и эффективные решения для хранения и управления данными.

**Отчет по лабораторной работе** должен содержать:

– результаты индивидуальных заданий;

– выводы исходя из цели лабораторной работы и выполненных пунктов задания.