МИНИСТЕРСТВО НАУК И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ИМЕНИ И.М. ГУБКИНА

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

**ОТЧЕТ**

**Лабораторной работе №5**

«Ранжирование по Сугено»

По дисциплине «Теория принятия решения. Нечеткие модели»

Выполнил работу студент группы АС-21-05

Алешко Альберт Алексеевич

Проверил Манзюк Николай Юрьевич

Москва, 2025

Метод Сугено — это способ принятия решений, когда у нас есть несколько вариантов и каждый из них оценивается по нескольким критериям, но эти оценки нечеткие. То есть вместо точных чисел (например, доход = 100 тыс) могут быть размытые оценки (например, доход = высокий). Метод помогает ранжировать варианты (например, выбрать лучший инвестиционный проект, университет, автомобиль), даже если данные неоднозначны.

Интеграл Сугено — это способ агрегировать множество значений, учитывая их важность (вес), особенно полезен в условиях нечёткости и принятия решений.

Пусть есть множество критериев



, и:

* f(xi) — значение признака (например, нормализованная оценка),
* g(xi) — вес (мера важности) признака.

Интеграл Сугено определяется следующим образом:



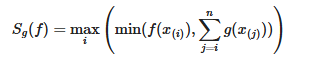
где:

* — элементы, отсортированные по убыванию значений ( f ):



* — сумма весов критериев с оценкой не ниже текущей,
* — отсечённая по важности оценка.

Финальный результат — максимум среди всех таких минимумов:



Такой подход позволяет учесть как силу выраженности признака, так и его значимость.

**Работа метода.**

1. Сбор данных У нас есть несколько объектов (например, проекты, автомобили, университеты). Каждый объект оценивается по нескольким критериям (например, цена, безопасность, комфорт). Оценки часто бывают неточными, поэтому представляются нечеткими числами (например, «низкий», «средний», «высокий»).

2. Дефаззификация (превращаем нечеткие оценки в числа) Чтобы сравнивать оценки, их надо привести к числовому виду. Например, оценку «низкий» можно представить как 0.2, «средний» как 0.5, «высокий» как 0.8. Один из способов — метод центра масс: если у нас есть нечеткое число вида (a, b, c), то его четкая оценка считается как их среднее арифметическое.

3. Применяем интеграл Сугено Этот шаг позволяет учитывать влияние разных критериев. Каждый критерий имеет вес (например, безопасность важнее, чем цена). Формула интеграла: S(f)=maxmin(значение; сумма весов критериев, которые ему соответствуют) По сути, мы рассматриваем, насколько "плохое" значение компенсируется важностью других критериев.

4. Ранжируем объекты После расчёта интеграла Сугено для каждого объекта, выбираем лучший. Чем выше значение интеграла — тем лучше объект.

**Используемые данные**

Для ранжирования были выбраны данные о наличии депрессии у студентов

В наборе данных представлены такие переменные, как:

* Пол
* Возраст
* Уровень образования
* Семейное положение
* Работа
* Среднее время сна
* Диетические привычки
* Уровень стресса
* Уровень физической активности
* Уровень социальной поддержки
* Уровень финансовой стабильности
* Уровень удовлетворенности жизнью
* Уровень депрессии

Листинг кода и результаты:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from scipy.stats import entropy

from sklearn.model\_selection import train\_test\_split

from scipy.optimize import minimize

from sklearn.metrics import f1\_score, classification\_report

class SugenoDepressionAnalyzer:

    def \_\_init\_\_(self, file\_path):

        """Инициализация анализатора с загрузкой данных"""

        self.data = pd.read\_csv(file\_path, index\_col='id')

        self.data = self.data.dropna()

        self.weights = {}

        self.features = [

            'Academic Pressure',

            'Work Pressure',

            'Sleep Duration',

            'Financial Stress',

            'Family History of Mental Illness',

            'Have you ever had suicidal thoughts ?',

            'CGPA',

            'Work/Study Hours'

        ]

    def \_preprocess\_data(self):

        """Нормализация и преобразование данных"""

        # Бинарное кодирование

        self.data['Family History of Mental Illness'] = self.data['Family History of Mental Illness'].map(

            {'Yes': 1, 'No': 0})

        self.data['Have you ever had suicidal thoughts ?'] = self.data['Have you ever had suicidal thoughts ?'].map(

            {'Yes': 1, 'No': 0})

        # Преобразование категориальных данных о сне

        sleep\_map = {

            'Less than 5 hours': 0.2,

            '5-6 hours': 0.5,

            '6-7 hours': 0.8,

            'More than 7 hours': 1

        }

        self.data['Sleep Duration'] = self.data['Sleep Duration'].map(sleep\_map)

        # Нормализация числовых признаков

        scaler = MinMaxScaler()

        numeric\_features = ['Academic Pressure', 'Work Pressure', 'Financial Stress', 'CGPA', 'Work/Study Hours']

        self.data[numeric\_features] = scaler.fit\_transform(self.data[numeric\_features])

        # Инвертируем CGPA (чем выше CGPA, тем ниже риск депрессии)

        self.data['CGPA'] = 1 - self.data['CGPA']

    def \_calculate\_weights(self):

        """Автоматический расчет весов на основе энтропии"""

        normalized = self.data[self.features].apply(lambda x: x/x.sum(), axis=0)

        k = 1/np.log(len(self.data))

        e\_j = normalized.apply(lambda x: -k\*(x\*np.log(x)).sum())

        d\_j = 1 - e\_j

        self.weights = {feature: weight for feature, weight in zip(self.features, d\_j/d\_j.sum())}

        # Ручная корректировка весов для новых признаков

        self.weights['CGPA'] \*= 0.7  # Умеренное влияние

        self.weights['Work/Study Hours'] \*= 0.6  # Среднее влияние

        # Нормализуем веса, чтобы сумма была равна 1

        total\_weight = sum(self.weights.values())

        self.weights = {k: v/total\_weight for k, v in self.weights.items()}

    def \_sugeno\_integral(self, row):

        """Вычисление интеграла Сугено для одной записи"""

        sorted\_scores = sorted([(f, row[f]) for f in self.features], key=lambda x: x[1])

        cumulative\_weight = 0

        sugeno\_values = []

        for feature, score in sorted\_scores:

            cumulative\_weight += self.weights[feature]

            sugeno\_values.append(min(score, cumulative\_weight))

        return max(sugeno\_values)

    def analyze(self):

        """Основной рабочий процесс"""

        self.\_preprocess\_data()

        self.\_calculate\_weights()

        print("Рассчитанные веса критериев:")

        for feature, weight in self.weights.items():

            print(f"{feature}: {weight:.3f}")

        self.data['Sugeno\_Score'] = self.data.apply(self.\_sugeno\_integral, axis=1)

        self.ranked\_data = self.data.sort\_values('Sugeno\_Score', ascending=False)

        self.\_visualize\_results()

        self.\_statistical\_analysis()

    def \_visualize\_results(self):

        """Графическое представление результатов"""

        plt.figure(figsize=(16, 12))

        # Распределение оценок

        plt.subplot(2, 2, 1)

        plt.hist(self.data['Sugeno\_Score'], bins=20, color='skyblue', edgecolor='black')

        plt.title('Распределение оценок депрессии', fontsize=12)

        plt.xlabel('Sugeno Score')

        plt.ylabel('Количество студентов')

        # Матрица корреляций

        plt.subplot(2, 2, 2)

        corr\_matrix = self.data[self.features + ['Sugeno\_Score']].corr()

        plt.imshow(corr\_matrix, cmap='coolwarm', vmin=-1, vmax=1)

        plt.colorbar()

        #plt.xticks(range(len(corr\_matrix.columns)), corr\_matrix.columns, rotation=45)

        plt.yticks(range(len(corr\_matrix.columns)), corr\_matrix.columns)

        plt.title('Матрица корреляций', fontsize=12)

        # Влияние факторов

        plt.subplot(2, 2, 3)

        feature\_impacts = self.data[self.features].mean() \* pd.Series(self.weights)

        feature\_impacts.sort\_values().plot(kind='barh', color='teal')

        plt.title('Вклад факторов в оценку депрессии', fontsize=12)

        plt.xlabel('Взвешенное влияние')

        plt.tight\_layout()

        plt.show()

    def \_statistical\_analysis(self):

        """Анализ статистических показателей"""

        print("\nСтатистика по оценкам:")

        print(self.data['Sugeno\_Score'].describe())

        if 'Depression' in self.data.columns:

            correlation = self.data[['Sugeno\_Score', 'Depression']].corr().iloc[0,1]

            print(f"\nКорреляция с исходными данными: {correlation:.3f}")

            # Анализ точности прогноза

            threshold = 0.5

            predicted = (self.data['Sugeno\_Score'] > threshold).astype(int)

            actual = self.data['Depression']

            accuracy = (predicted == actual).mean()

            print(f"\nТочность прогноза (при threshold={threshold}): {accuracy:.2%}")

        high\_risk = self.data[self.data['Sugeno\_Score'] > 0.7]

        print(f"\nСтудентов с высоким риском: {len(high\_risk)}")

        print("Распределение по полу:")

        print(high\_risk['Gender'].value\_counts())

    def get\_high\_risk\_students(self, threshold=0.7):

        """Получение списка студентов группы риска"""

        return self.ranked\_data[self.ranked\_data['Sugeno\_Score'] > threshold][

            ['Gender', 'Age', 'Sugeno\_Score', 'CGPA', 'Work/Study Hours', 'Depression'] + self.features]

    def get\_all\_students(self):

        return self.ranked\_data[['Gender', 'Age', 'Sugeno\_Score', 'Depression']]

analyzer = SugenoDepressionAnalyzer('Student Depression Dataset.csv')

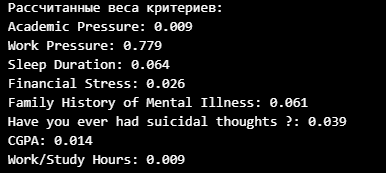
analyzer.analyze()

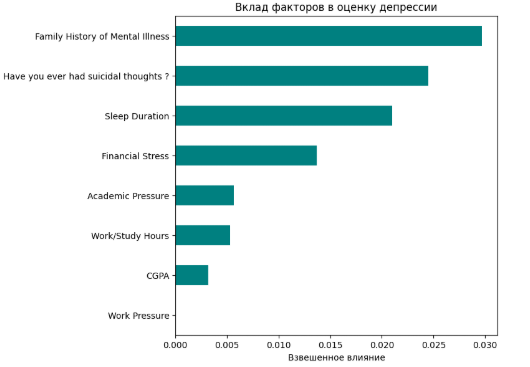
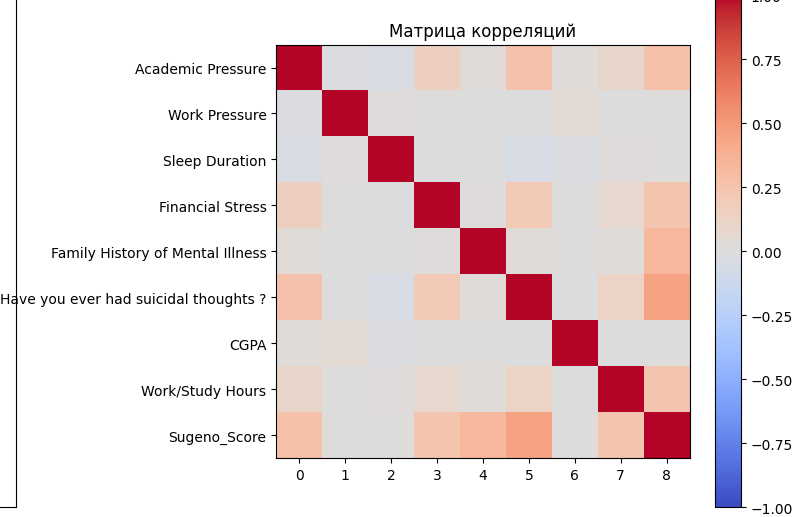
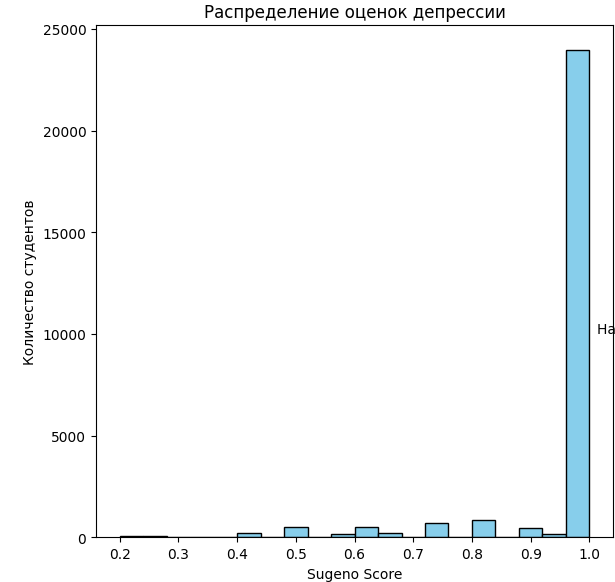
# Получение и вывод студентов группы риска

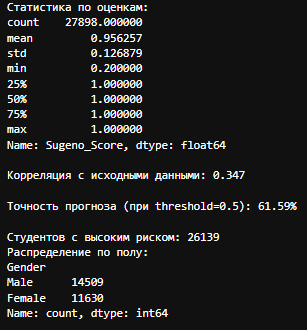
high\_risk = analyzer.get\_high\_risk\_students()

print("\nСписок студентов с высоким риском:")

high\_risk.head(10)







Разделение выборки на тренировочную и тестовую для определения наиболее реалистичного порогового значения:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(studs.drop('Depression', axis=1), studs['Depression'], test\_size=0.2, random\_state=42)

def calculat\_f1(threshold):

    preds = (X\_train['Sugeno\_Score'] > threshold[0]).astype(np.int32)

    return f1\_score(y\_train, preds)

length = 1000

thresh\_values = np.linspace(0.9, 0.96, length)[..., None]

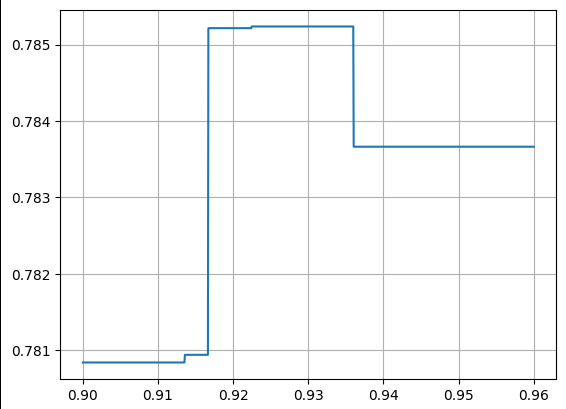
f1\_values = np.zeros(length)

for idx, thresh in enumerate(thresh\_values):

    f1\_values[idx] = calculat\_f1(thresh)

plt.plot(thresh\_values, f1\_values)

plt.grid(True)



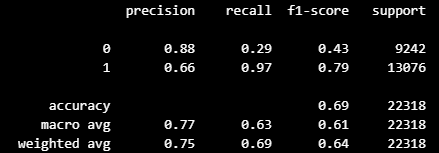
Тем самым пороговое значение равно 0.93 и точность при нём 0.69

threshold = 0.93

y\_pred\_train = (X\_train['Sugeno\_Score'] > threshold).astype(np.int32)

y\_pred\_test = (X\_test['Sugeno\_Score'] > threshold).astype(np.int32)

print(classification\_report(y\_train, y\_pred\_train))



print(classification\_report(y\_test, y\_pred\_test))

