1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «СИСТЕМЫ НЕЧЕТКОГО ВЫВОДА»
2. по дисциплине «Основы искусственного интеллекта»
3. Выполнил
4. студент гр. 5151003/00801 В.А. Васильев

<*подпись*>

1. Преподаватель
2. В.М. Крундышев

<*подпись*>

1. Санкт-Петербург
2. 2023

# Теоретические сведения

**Нечеткая логика** (англ. fuzzy logic) – раздел математики, являющийся обобщением классической логики и теории множеств, базирующийся на понятии нечёткого множества, впервые введённого Лотфи Заде в 1965 году как объекта с функцией принадлежности элемента ко множеству, принимающей любые значения на отрезке [0,1] а не только 0 или 1. На основе этого понятия вводятся различные логические операции над нечёткими множествами и формулируется понятие лингвистической переменной, в качестве значений которой выступают нечёткие множества.

Системы нечеткого вывода.

* **нечеткие экспертные системы**:требуется привлечение экспертов для построения функций принадлежности, определения оптимального количества нечетких градаций для входных лингвистических переменных. Субъективный характер экспертных оценок, высокая трудоемкость формирования моделей, а также отсутствие информации о достоверности получаемых решений в случае неполноты исходных данных могут привести к неэффективности формируемых нечетких моделей, недостаточной точности получаемых оценок о состоянии объектов и, как следствие, низкой эффективности практического использования нечетких экспертных систем.
* **адаптивные нейро-нечеткие системы**: параметры функций принадлежности настраиваются автоматически (без участия экспертов) в процессе обучения на экспериментальных данных. Особенность систем данного класса заключается в использовании нейросетевых структур и нечеткой логики для управления сложными динамическими объектами, способных функционировать в условиях неопределенности математического описания объекта управления.

# Формулировка задания

Цель работы – изучение принципов работы систем нечеткого вывода, моделей представления знаний, а также метода нечеткой кластеризации.

# Ход работы

## Система нечеткого вывода

В соответствии с вариантом задания была сформирована база правил с 2 параметрами и 3 диапазонами:

rule1 = ctrl.Rule(intelligence['poor'] | talent['poor'], chance['poor'])  
rule2 = ctrl.Rule(intelligence['average'], chance['average'])  
rule3 = ctrl.Rule(intelligence['good'] | talent['good'], chance['good'])

Для входных параметров *intelligence = 78* и *talent = 1* система нечеткого вывода рассчитала следующий процент вероятности стать ученым:

![A diagram of a graph

Description automatically generated]()

Рисунок 1 – Результаты работы системы нечеткого вывода

Далее был разработан набор данных со 150 записями для работы вычислительной модели:

A screenshot of a cell phone

Description automatically generated

Рисунок 2 – Набор данных Input.csv

## Метрики

Для оценки производительности модели были использованы следующие метрики:

*Среднее значение (Mean):*

* Среднее значение представляет собой сумму всех значений в наборе данных, разделенную на количество этих значений.
* Формула для вычисления среднего значения: Mean = (Σx) / n, где Σx - сумма всех значений, а n - количество значений.
* Среднее значение позволяет определить "центр" данных, указывая на типичное или среднее значение в наборе данных.

*Стандартное отклонение (Standard Deviation):*

* Стандартное отклонение измеряет степень разброса данных от их среднего значения.
* Большое стандартное отклонение указывает на большой разброс данных, тогда как маленькое стандартное отклонение означает, что данные сгруппированы близко к среднему значению.
* Формула для вычисления стандартного отклонения: Std Deviation = √(Σ(x - Mean)² / n), где Σ(x - Mean)² - сумма квадратов разниц между каждым значением и средним значением, а n - количество значений.

*Membership matrix (матрица членства):*

* Это концепция, связанная с теорией нечетких множеств и нечеткой логикой.
* Она используется для описания степени принадлежности элементов к различным нечетким множествам.
* Матрица членства предоставляет информацию о том, насколько каждый элемент "принадлежит" каждому из нечетких множеств.

## Вычислительная модель

## Была разработана модель на основе C-средних и разработанном наборе данных «Вероятность стать ученым».

*Метод "Fuzzy C-Means" (FCM)* – это алгоритм нечеткой кластеризации, который используется для разделения набора данных на нечеткие кластеры.

Преимущество метода Fuzzy C-Means заключается в его способности учитывать нечеткость и неопределенность в данных. Он может использоваться в различных областях, включая обработку изображений, биоинформатику, паттерн-распознавание и так далее. Тем не менее, важно правильно настраивать параметры, такие как количество кластеров и индекс размытости, для достижения хороших результатов.

В результате работы получим:

![A graph with colored circles and dots

Description automatically generated]()  
Рисунок 3 – Финальная нечеткая кластеризация данных

А также оценку вычислительной модели:

mean= 87.06666666666666

Std dev= 3.358571124749333

Accuracy = 88.0

# Выводы

В ходе лабораторной работы была освоена работа с системами нечеткого вывода, а также вычислительными моделями, основанными на методе С-средних. Также были изучены такие метрики оценки, как Mean и Standard Deviation, разработан собственный набор данных «Вероятность стать ученым».

# Приложение А

Система нечеткого вывода

import numpy as np  
import skfuzzy as fuzz  
from skfuzzy import control as ctrl  
  
intelligence = ctrl.Antecedent(np.arange(70, 161, 1), 'intelligence')  
talent = ctrl.Antecedent(np.arange(0, 11, 1), 'talent')  
chance = ctrl.Consequent(np.arange(0, 101, 1), "chance")  
  
intelligence.automf(3)  
talent.automf(3)  
  
chance['poor'] = fuzz.trimf(chance.universe, [0, 0, 50])  
chance['average'] = fuzz.trimf(chance.universe, [0, 50, 100])  
chance['good'] = fuzz.trimf(chance.universe, [50, 100, 100])  
  
intelligence.view()  
talent.view()  
chance.view()  
  
rule1 = ctrl.Rule(intelligence['poor'] | talent['poor'], chance['poor'])  
rule2 = ctrl.Rule(intelligence['average'], chance['average'])  
rule3 = ctrl.Rule(intelligence['good'] | talent['good'], chance['good'])  
  
chance\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])  
probabilities = ctrl.ControlSystemSimulation(chance\_ctrl)  
  
probabilities.input["intelligence"] = 78  
probabilities.input["talent"] = 1  
  
probabilities.compute()  
  
print(probabilities.output['chance'])  
  
chance.view(sim=probabilities)

# Приложение Б

Вычислительная модель на основе метода C-средних

# The Algorithm\*\*\*\*  
# Fuzzy c-means (FCM) is a method of clustering which allows one  
# piece of data to belong to two or more clusters. This method (developed by  
# Dunn in 1973 and improved by Bezdek in 1981) is frequently used in pattern  
# recognition.  
  
# # Loading modules and training data  
import pandas as pd  
import numpy as np  
import random  
import operator  
import math  
import matplotlib.pyplot as plt  
from scipy.stats import multivariate\_normal  
  
df\_full = pd.read\_csv("Input.csv")  
df\_full.head()  
df\_full = df\_full.drop(['Id'], axis=1)  
df\_full.shape  
columns = list(df\_full.columns)  
features = columns[:len(columns)-1]  
class\_labels = list(df\_full[columns[-1]])  
df = df\_full[features]  
  
# # Defining parameters  
# Number of Clusters  
k = 3  
# Maximum number of iterations  
MAX\_ITER = 100  
# Number of data points  
n = len(df)  
# Fuzzy parameter  
m = 1.7 #Select a value greater than 1 else it will be knn  
# # Scatter Plots  
plt.figure(figsize=(10,10))  
plt.scatter(list(df.iloc[:,0]), list(df.iloc[:,1]), marker='o')  
plt.axis('equal')  
plt.xlabel('intelligence', fontsize=16)  
plt.ylabel('talent', fontsize=16)  
plt.title('Chance Plot', fontsize=22)  
plt.grid()  
plt.show()  
  
# # Calculating accuracy  
def accuracy(cluster\_labels, class\_labels):  
 correct\_pred = 0  
 #print(cluster\_labels)  
 low = max(set(labels[0:50]), key=labels[0:50].count)  
 medium = max(set(labels[50:100]), key=labels[50:100].count)  
 high = max(set(labels[100:]), key=labels[100:].count)  
 for i in range(len(df)):  
 if cluster\_labels[i] == low and class\_labels[i] == 'poor;;':  
 correct\_pred = correct\_pred + 1  
 if cluster\_labels[i] == medium and class\_labels[i] == 'average;;' and medium!=low:  
 correct\_pred = correct\_pred + 1  
 if cluster\_labels[i] == high and class\_labels[i] == 'good;;' and high!=low and high!=medium:  
 correct\_pred = correct\_pred + 1  
 accuracy = (correct\_pred/len(df))\*100  
 return accuracy  
  
# # Initialize membership matrix  
def initializeMembershipMatrix():  
 membership\_mat = []  
 for i in range(n):  
 random\_num\_list = [random.random() for i in range(k)]  
 summation = sum(random\_num\_list)  
 temp\_list = [x/summation for x in random\_num\_list]  
 flag = temp\_list.index(max(temp\_list))  
 for j in range(0,len(temp\_list)):  
 if(j == flag):  
 temp\_list[j] = 1  
 else:  
 temp\_list[j] = 0  
 membership\_mat.append(temp\_list)  
 return membership\_mat  
  
membership\_mat = initializeMembershipMatrix()  
  
# # Calculating Cluster Center  
def calculateClusterCenter(membership\_mat):  
 cluster\_mem\_val = list(zip(\*membership\_mat))  
 cluster\_centers = []  
 for j in range(k):  
 x = list(cluster\_mem\_val[j])  
 xraised = [p \*\* m for p in x]  
 denominator = sum(xraised)  
 temp\_num = []  
 for i in range(n):  
 data\_point = list(df.iloc[i])  
 prod = [xraised[i] \* val for val in data\_point]  
 temp\_num.append(prod)  
 numerator = map(sum, list(zip(\*temp\_num)))  
 center = [z / denominator for z in numerator]  
 cluster\_centers.append(center)  
 return cluster\_centers  
  
calculateClusterCenter(membership\_mat)  
  
# # Updating Membership Value  
def updateMembershipValue(membership\_mat, cluster\_centers):  
 p = float(2/(m-1))  
 for i in range(n):  
 x = list(df.iloc[i])  
 distances = [np.linalg.norm(np.array(list(map(operator.sub, x, cluster\_centers[j])))) for j in range(k)]  
 for j in range(k):  
 den = sum([math.pow(float(distances[j]/distances[c]), p) for c in range(k)])  
 membership\_mat[i][j] = float(1/den)  
 return membership\_mat  
  
# # Getting the clusters  
def getClusters(membership\_mat):  
 cluster\_labels = list()  
 for i in range(n):  
 max\_val, idx = max((val, idx) for (idx, val) in enumerate(membership\_mat[i]))  
 cluster\_labels.append(idx)  
 return cluster\_labels  
  
# \*\*Below are three different initializations. When the  
# initialization is at the origin all points converge into one cluster and  
# for the other 2 cases we get the clusters as we have initialized before(3  
# in this code\*\*)\*\*  
  
# # Fuzzy C-Means with cluster centres at the origin  
def fuzzyCMeansClustering(): #First Iteration with centers at 0  
 # Membership Matrix  
 membership\_mat = initializeMembershipMatrix()  
 curr = 0  
 acc=[]  
 cent\_temp = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]  
 while curr < MAX\_ITER:  
 if (curr == 0):  
 cluster\_centers = cent\_temp  
 print("Cluster Centers:")  
 print(np.array(cluster\_centers))  
 else:  
 cluster\_centers = calculateClusterCenter(membership\_mat)  
 # cluster\_centers = calculateClusterCenter(membership\_mat)  
 membership\_mat = updateMembershipValue(membership\_mat, cluster\_centers)  
 cluster\_labels = getClusters(membership\_mat)  
 acc.append(cluster\_labels)  
 curr += 1  
 print("---------------------------")  
 print("Membership Matrix:")  
 print(np.array(membership\_mat))  
 return cluster\_labels, cluster\_centers, acc  
  
# # Fuzzy C-Means with with cluster centers at random locations  
# within a multi-variate Gaussian distribution with zero-mean and unit-  
# variance.  
  
def fuzzyCMeansClustering(): #Second Iteration Multivariate Gaussian  
 # Membership Matrix  
 membership\_mat = initializeMembershipMatrix()  
 curr = 0  
 acc=[]  
 mean = [0, 0]  
 cov = [[1, 0], [0, 1]]  
 lis1,cent\_temp=[],[]  
 for i in range(0,k):  
 Z = list(np.random.multivariate\_normal(mean, cov))  
 Z1 = list(np.random.multivariate\_normal(mean, cov))  
 lis1 = Z+Z1  
 cent\_temp.append(lis1)  
 while curr < MAX\_ITER:  
 if(curr == 0):  
 cluster\_centers = cent\_temp  
 print("Cluster Centers:")  
 print(np.array(cluster\_centers))  
 else:  
 cluster\_centers = calculateClusterCenter(membership\_mat)  
 #cluster\_centers = calculateClusterCenter(membership\_mat)  
 membership\_mat = updateMembershipValue(membership\_mat, cluster\_centers)  
 cluster\_labels = getClusters(membership\_mat)  
 acc.append(cluster\_labels)  
 curr += 1  
 print("---------------------------")  
 print("Membership Matrix:")  
 print(np.array(membership\_mat))  
 return cluster\_labels, cluster\_centers, acc  
  
# # Fuzzy C-Means with cluster centers at random vectors chosen from  
# the data.  
def fuzzyCMeansClustering(): #Third iteration Random vectors from data  
 # Membership Matrix  
 membership\_mat = initializeMembershipMatrix()  
 curr = 0  
 acc=[]  
 while curr < MAX\_ITER:  
 cluster\_centers = calculateClusterCenter(membership\_mat)  
 membership\_mat = updateMembershipValue(membership\_mat, cluster\_centers)  
 cluster\_labels = getClusters(membership\_mat)  
 acc.append(cluster\_labels)  
 if(curr == 0):  
 print("Cluster Centers:")  
 print(np.array(cluster\_centers))  
 curr += 1  
 print("---------------------------")  
 print("Partition matrix:")  
 print(np.array(membership\_mat))  
 #return cluster\_labels, cluster\_centers  
 return cluster\_labels, cluster\_centers, acc  
  
# # Calculating the Accuracy  
labels, centers, acc = fuzzyCMeansClustering()  
a = accuracy(labels, class\_labels)  
acc\_lis = []  
for i in range(0,len(acc)):  
 val = accuracy(acc[i], class\_labels)  
 acc\_lis.append(val)  
acc\_lis = np.array(acc\_lis) #calculating accuracy and std deviation 100 times  
print("mean=",np.mean(acc\_lis))  
print("Std dev=",np.std(acc\_lis))  
print("Accuracy = " + str(round(a, 2)))  
print("Cluster center vectors:") #final cluster centers  
print(np.array(centers))  
  
# # Plotting the data  
sepal\_df = df\_full.iloc[:,0:2]  
sepal\_df = np.array(sepal\_df)  
  
m1 = random.choice(sepal\_df)  
m2 = random.choice(sepal\_df)  
m3 = random.choice(sepal\_df)  
cov1 = np.cov(np.transpose(sepal\_df))  
cov2 = np.cov(np.transpose(sepal\_df))  
cov3 = np.cov(np.transpose(sepal\_df))  
x1 = np.linspace(70,150,150)  
x2 = np.linspace(0,11,150)  
X, Y = np.meshgrid(x1,x2)  
Z1 = multivariate\_normal(m1, cov1)  
Z2 = multivariate\_normal(m2, cov2)  
Z3 = multivariate\_normal(m3, cov3)  
pos = np.empty(X.shape + (2,))  
pos[:, :, 0] = X; pos[:, :, 1] = Y  
plt.figure(figsize=(10,10))  
plt.scatter(sepal\_df[:,0], sepal\_df[:,1], marker='o')  
plt.contour(X, Y, Z1.pdf(pos), colors="r" ,alpha = 0.5)  
plt.contour(X, Y, Z2.pdf(pos), colors="b" ,alpha = 0.5)  
plt.contour(X, Y, Z3.pdf(pos), colors="g" ,alpha = 0.5)  
plt.axis('equal')  
plt.xlabel('intelligence', fontsize=16)  
plt.ylabel('talent', fontsize=16)  
plt.title('Initial Random Clusters (Chance Evaluation)', fontsize=22)  
plt.grid()  
plt.show()  
  
#finding mode  
low = max(set(labels[0:50]), key=labels[0:50].count)  
medium = max(set(labels[50:100]), key=labels[50:100].count)  
high = max(set(labels[100:]), key=labels[100:].count)  
  
#features  
s\_mean\_clus1 = np.array([centers[low][0],centers[low][1]])  
s\_mean\_clus2 = np.array([centers[medium][0],centers[medium][1]])  
s\_mean\_clus3 = np.array([centers[high][0],centers[high][1]])  
  
values = np.array(labels) #label  
  
#search all 3 chance level  
searchval\_low = low  
searchval\_medium = medium  
searchval\_high = high  
  
#index of all 3 chance level  
ii\_low = np.where(values == searchval\_low)[0]  
ii\_medium = np.where(values == searchval\_medium)[0]  
ii\_high = np.where(values == searchval\_high)[0]  
ind\_low = list(ii\_low)  
ind\_medium = list(ii\_medium)  
ind\_high = list(ii\_high)  
  
sepal\_df = df\_full.iloc[:,0:2]  
  
low\_df = sepal\_df[sepal\_df.index.isin(ind\_low)]  
medium\_df = sepal\_df[sepal\_df.index.isin(ind\_medium)]  
high\_df = sepal\_df[sepal\_df.index.isin(ind\_high)]  
  
cov\_low = np.cov(np.transpose(np.array(low\_df)))  
cov\_medium = np.cov(np.transpose(np.array(medium\_df)))  
cov\_high = np.cov(np.transpose(np.array(high\_df)))  
  
sepal\_df = np.array(sepal\_df)  
  
x1 = np.linspace(70,160,150)  
x2 = np.linspace(0,11,150)  
X, Y = np.meshgrid(x1,x2)  
  
Z1 = multivariate\_normal(s\_mean\_clus1, cov\_low)  
Z2 = multivariate\_normal(s\_mean\_clus2, cov\_medium)  
Z3 = multivariate\_normal(s\_mean\_clus3, cov\_high)  
  
pos = np.empty(X.shape + (2,))  
pos[:, :, 0] = X; pos[:, :, 1] = Y  
  
plt.figure(figsize=(10,10))  
plt.scatter(sepal\_df[:,0], sepal\_df[:,1], marker='o')  
plt.contour(X, Y, Z1.pdf(pos), colors="r" ,alpha = 0.5)  
plt.contour(X, Y, Z2.pdf(pos), colors="b" ,alpha = 0.5)  
plt.contour(X, Y, Z3.pdf(pos), colors="g" ,alpha = 0.5)  
plt.axis('equal')  
plt.xlabel('intelligence', fontsize=16)  
plt.ylabel('talent', fontsize=16)  
plt.title('Final Clusters (Chance Evaluation)', fontsize=22)  
plt.grid()  
plt.show()