**Федеральное агентство по образованию**Государственное образовательное учреждение высшего профессионального   
образования **«Тихоокеанский Государственный университет»**

Факультет компьютерных и фундаментальных наук

Кафедра ПОВТАС

**Лабораторная работа №3**

по дисциплине: «Архитектура систем ИИ»

на тему: «Задача распознавания образов. Используя ИНС»  
Вариант №4

Выполнил: студент группы ПИИ(м)-21

Забавин А.С.

Проверил: ст. преп. кафедры ПОВТАС

Тормозов В.С.

# Постановка задачи

Для распознавания предъявляется циферблат электронных часов, который состоит из четырех позиций: две позиции для вывода часов и две позиции для вывода минут (чч:мм). Каждая цифра представляется набором пикселов из 9 строк по 6 пикселов в каждой строке. Каждый пиксел может принимать одно из двух значений символов «\*» или «.». Сочетания «\*» и «.» задают одну из 10 цифр.

В идеале на циферблате часов показывается реальное время. Однако в системе часов произошел сбой и некоторые пикселы приняли «неправильные» значения.

Требуется определить, какое же время выводят часы или вывести сообщение о невозможности определения времени.

**Исходные данные**

В текстовом файле “etalon.txt” приведены образцы цифр в виде 10 групп 6 х 8 символов «\*» или «.». Во втором текстовом файле “times.txt” записаны показания с циферблата часов приведены образцы цифр в виде 4 групп 6 х 8 символов «\*» или «.», как с точными значениями, так и поврежденными. При запуске – программа попросит ввести время из файла times.txt, затем сама разобьет ее на соответствующие часам и минутам значения и попытается распознать цифры используя ИНС.

**Выходные данные**

На экран выводится сообщение о времени в форме «чч:мм» или сообщение «[Невозможно определить]» если средняя ошибка распознавания ИНС превышает 50%.

# Краткая теория

Под распознаванием образа понимаются процессы восприятия и познания, связанные с поиском, выделением, идентификацией и классификацией образов на основе имеющихся данных.

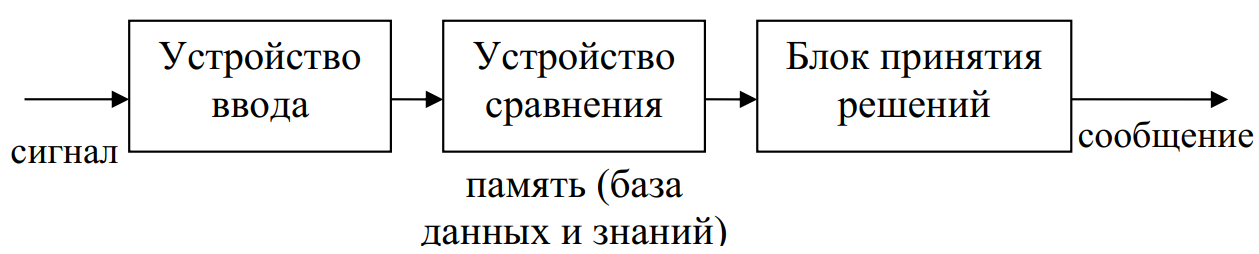


Рис. . Система распознавания

В общем виде систему, проводящую распознавание образов, можно представить как совокупность устройств ввода информации–образа, сравнения с накопленными ранее знаниями и устройства принятия решений (Рис. 1).

Известны следующие типы задач распознавания.

1. Отнесение предъявляемого объекта по его формализованному описанию (образу) к одному из заданных классов.
2. Разбиение множества объектов по их формализованным описаниям к системе непересекающихся классов.
3. Определение набора признаков для построения формализованного описания объекта распознавания; оценка информативности отдельных признаков и их сочетаний.
4. Построение формализованного описания объекта распознавания.

В приведенной на рисунке 1 схеме нашли отражение задачи первого и второго типов данной классификации. С учетом всех четырех типов общая структура системы распознавания выглядит так, как изображено на Рис. 2.

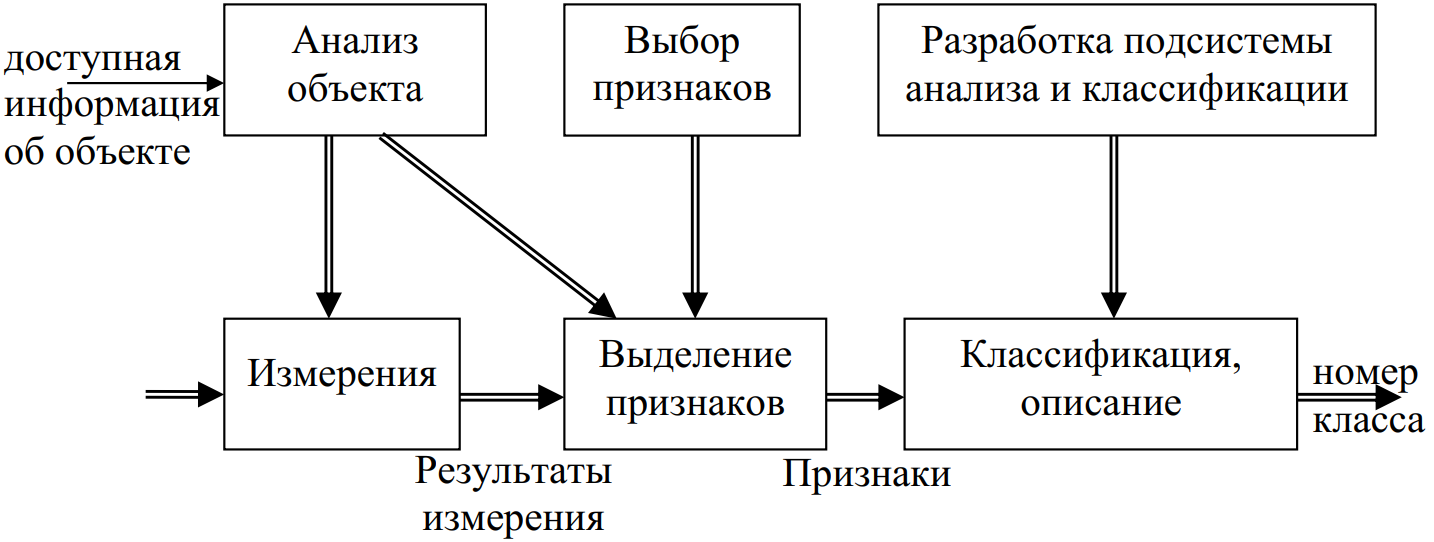


Рис. . Общая структура системы распознавания

В настоящее время разработано большое количество методов (моделей) решения задачи распознавания. Перечислим некоторые из них.

*Р-модель* (*модель разделения*) характеризуется тем, что проводиться граница между классами в пространстве Rn размерности n. При построении информационного вектора исследуется положение объекта относительно данной границы.

*Статистические модели* основаны на использовании аппарата математической статистики. Применяются тогда, когда могут быть определены вероятностные характеристики классов, например, функции распределения.

*П-модели* используют понятие потенциала, определенного для каждой точки пространства. Для определения принадлежности объекта классу вычисляются значения потенциальной функции.

В *моделях, основанных на вычислении оценок*, вычисление близости объектов между собой определяется совпадением их частичных описаний. Вычисляется обобщенная близость объекта классам, по которой и устанавливаются значения для информационного вектора .

На аппарате алгебры логики основаны *Л-модели*.

# Результаты работы

Первым делом, согласно поставленной задачи, было создано 2 файла, отвечающих за ввод «эталонных» и входных «неправильных» значений (см. Листинг 1, Листинг 1).

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*....\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

.....\*

...\*.\*

..\*..\*

.\*...\*

\*....\*

.....\*

.....\*

.....\*

.....\*

..\*\*\*.

.\*...\*

\*...\*.

...\*..

..\*...

.\*....

\*.....

\*.....

\*\*\*\*\*\*

\*\*\*\*\*\*

.....\*

.....\*

.....\*

\*\*\*\*\*\*

.....\*

.....\*

.....\*

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

.....\*

.....\*

.....\*

.....\*

\*\*\*\*\*\*

\*.....

\*.....

\*.....

\*\*\*\*..

....\*.

.....\*

....\*.

\*\*\*\*..

\*\*\*\*\*\*

\*.....

\*.....

\*.....

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

\*\*\*\*\*\*

.....\*

.....\*

.....\*

..\*\*\*\*

.....\*

.....\*

.....\*

.....\*

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

\*\*\*\*\*\*

\*....\*

\*....\*

\*....\*

\*\*\*\*\*\*

.....\*

.....\*

.....\*

.....\*

Листинг . Эталонные значения (etalon.txt)

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

\*....\* \*....\* \*....\* \*....\*

.....\* \*....\* \*....\* \*....\*

\*....\* \*....\* + \*....\* \*.....

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* + .....\* \*....\*

\*....\* \*....\* \*....\* \*....\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

.....\* .....\* .....\* .....\*

...\*.\* ...\*.\* ...\*.\* ...\*.\*

..\*..\* ..\*..\* ..\*..\* ..\*..\*

.\*...\* .\*...\* + .\*...\* .\*...\*

\*....\* \*....\* \*....\* \*....\*

.....\* .....\* .....\* .....\*

.....\* .....\* + .....\* .....\*

.....\* .....\* .....\* .....\*

.....\* .....\* .....\* .....\*

..\*.\*. ...... ...... ......

.\*...\* .\*...\* .\*...\* .\*...\*

\*...\*. \*...\*. \*...\*. \*...\*.

...\*.. ...\*.. + ...\*.. ...\*..

...... ...... ..\*... ..\*...

.\*.... ...\*.. .\*.... ......

...... \*.\*... + \*..... \*..\*..

\*..... \*..... \*..... \*.....

\*\*.\*.\* .\*\*.\*\* \*\*.\*.\* \*\*.\*.\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

.....\* .....\* .....\* .....\*

.....\* .....\* .....\* .....\*

.....\* .....\* + .....\* .....\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

.....\* .....\* .....\* .....\*

.....\* .....\* + .....\* .....\*

.....\* .....\* .....\* .....\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* + \*....\* \*....\*

\*\*\*\*\*\* \*\*.\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

.....\* .....\* .....\* .....\*

.....\* .....\* + .....\* .....\*

.....\* .....\* .....\* .....\*

.....\* .....\* .....\* .....\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

\*..... \*..... \*..... \*.....

\*..... \*..... \*..... \*.....

\*..... \*..... + \*..... \*.....

\*\*\*\*.. \*\*\*\*.. \*\*\*\*.. \*\*\*\*..

....\*. ....\*. ....\*. ....\*.

.....\* .....\* + .....\* .....\*

....\*. ....\*. ....\*. ....\*.

\*\*\*\*.. \*\*\*\*.. \*\*\*\*.. \*\*\*\*..

\*\*\*\*\*\* \*\*\*\*\*\* .\*\*.\*\* \*\*\*\*\*\*

\*..... \*..... \*..... \*.....

\*..... \*..... \*..... \*.....

\*..... \*..... + \*..... \*.....

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

\*....\* \*....\* \*....\* \*....\*

\*....\* \*....\* + \*....\* \*....\*

\*....\* \*....\* \*....\* \*....\*

\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*

Листинг 2. Поврежденные значения показаний времени (times.txt)

Средствами языка python были написана программа, код программы приведен на Листинге 3 (некоторые части сокращены «. . . . . . .», полный код программы в приложенном файле main.py).

class **Classificator**:

*"""*

*Классификатор блоков пикселей (символов), записанных в текстовых файлах*

*При определении класс подгружает эталоны и определяет производные значения*

*"""*

# Init Class

. . . . . . .

ETALON\_PIXEL\_DATA = defaultdict(list)

MAX\_HAMM\_DIST = 0

MATRIX\_MAX\_X = 0

MATRIX\_MAX\_Y = 0

. . . . . . .

# Other methods

*@staticmethod*

def **quartering**(digitsimb: list):

*"""*

*Поделитть пиксели на четверти*

**:param** *digitsimb: список вида:*

*[*

*'\*\*\*\*\*\*\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*\*\*\*\*\*\*',*

*]*

*"""*

. . . . . . .

return quadrants

*@staticmethod*

def **bin\_vectorize\_block**(digitblock: list):

*"""*

*С помощью numpy представить список "пикселей" как бинарный вектор*

**:param** *digitblock: список строк пикселей типаЖ*

*[*

*'\*\*\*\*',*

*'\*..',*

*'\*..',*

*]*

*"""*

return np.array(list(map(lambda x: 1 if x == *'\*'* else 0, itertools.chain.from\_iterable(digitblock))))

. . . . . . .

*@staticmethod*

def **hamm\_dist**(etalonvect: np.ndarray, vect: np.ndarray):

*"""*

*Вычисление расстояния Хемминга между бинарными векторами с помощью numpy*

**:param** *etalonvect:*

**:param** *vect:*

*"""*

vect\_dif = etalonvect - vect # Разность векторов. Если совпадает координата - то на этой позиции будет 0

nz = np.nonzero(vect\_dif) # Выделить из вектора разности ненулевые значения и вернуть позции на которых они есть

sp = np.shape(nz[0]) # Размерность массива + длинна, в случае вектора кортеж из одного элемента

return sp[0]

. . . . . . .

*@staticmethod*

def **get\_generalized\_hamm\_dist**(symbol: str, quad\_0\_dist, quad\_1\_dist, quad\_2\_dist, quad\_3\_dist):

*"""*

*Общее расстояние Хемминга за символ (по четвертям сумма)*

**:param** *symbol:*

**:param** *quad\_0\_dist:*

**:param** *quad\_1\_dist:*

**:param** *quad\_2\_dist:*

**:param** *quad\_3\_dist:*

*"""*

. . . . . . .

return (koefs[0] \* quad\_0\_dist + koefs[1] \* quad\_1\_dist + koefs[2] \* quad\_2\_dist + koefs[3] \* quad\_3\_dist) / sum(koefs)

class **PixelDigit**(Classificator):

*"""*

*Субкласс Классификатора символов - цифр*

*"""*

def **\_\_init\_\_**(*self*, string\_with\_lb: str):

*"""*

*Инициализация с текстом с обрывами строк в котором записаны пиксели*

**:param** *string\_with\_lb:*

*"""*

*self*.digit = -1

*self*.exact = False

*self*.recog\_hamm\_dist = -1

*self*.simple\_hamm\_dist = -1

*self*.\_pixel\_data = list(map(lambda x: x.strip(), string\_with\_lb.splitlines()))

*self*.percent\_like = 0

*self*.recognite()

def **recognite**(*self*):

*"""*

*Распознать*

*"""*

. . . . . . .

def **\_\_eq\_\_**(*self*, other):

*"""*

*Перегрузка операции равенства*

**:param** *other:*

*"""*

return True if (*self* - other) < *self*.MAX\_HAMM\_DIST else False

def **\_\_sub\_\_**(*self*, other):

*"""*

*Перегрузка операции разности (отличия) - расстояние Хемминга между цифрами*

**:param** *other:*

*"""*

etalonvect = *self*.bin\_vectorize\_block(*self*.ETALON\_PIXEL\_DATA[*self*.digit])

othervect = *self*.bin\_vectorize\_block(*self*.\_pixel\_data)

return *self*.hamm\_dist(etalonvect, othervect)

class **Clock**:

*"""*

*Класс Показаний часов*

*"""*

def **devide\_digits**(*self*, string\_time: str):

. . . . . . .

return time\_digits[0], time\_digits[1], time\_digits[2], time\_digits[3]

def **\_\_init\_\_**(*self*, string\_time: str):

*"""*

*Передаем значение показаний записанное с пробельными разделителями, 4 цифры, по типу:*

*\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\**

*\*....\* \*....\* \*....\* \*....\**

*.....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* + \*....\* \*.....*

*\*....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* + .....\* \*....\**

*\*....\* \*....\* \*....\* \*....\**

*\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\**

**:param** *string\_time:*

*"""*

. . . . . . .

def **\_\_str\_\_**(*self*):

if *self*.first\_digit.digit != -1 and *self*.second\_digit.digit != -1 and *self*.third\_digit.digit != -1 and *self*.four\_digit.digit != -1:

return *f'{self.first\_digit.digit}{self.second\_digit.digit}:{self.third\_digit.digit}{self.four\_digit.digit}'*

return *'[Невозможно определить]'*

time = property(\_\_str\_\_)

if \_\_name\_\_ == *"\_\_main\_\_"*:

print(*'Введите строку для распознавания: '*)

. . . . . . .

rec\_time = Clock(txt)

print(*f'''\n*

*Введенное значение:*

*{txt}*

*Предполагаемое время:*

*{rec\_time.time}*

*Точное совпадение:*

*{rec\_time.exact}*

*Расстояние Хемминга (общее):*

*{rec\_time.simple\_hamm\_dist}*

*Процент похожести (общий):*

*{rec\_time.percent\_like}*

*Процент похожести (десятки часов):*

*{rec\_time.first\_digit.percent\_like}*

*Процент похожести (часы):*

*{rec\_time.second\_digit.percent\_like}*

*Процент похожести (десятки минут):*

*{rec\_time.third\_digit.percent\_like}*

*Процент похожести (минуты):*

*{rec\_time.four\_digit.percent\_like}*

*'''*)

Вначале опишем абстрактный класс классификатора **Classificator**, который содержит базовые методы, используемые в данной задаче классификации, и некоторые константы. При определении класс подгружает эталонные значения символов из файла etalon.txt, переводит их во внутреннее представление класса для последующих операций. Класс содержит метод **bin\_vectorize\_block**(cls, digitblock: list) – для представления «списка строк» полученных пикселей символа в бинарный вектор (координаты-признаки 0, 1). После векторного представления можно найти расстояние Хемминга, расстояние между эталонным вектором цифры и введенными данными с помощью метода def **hamm\_dist**(cls, etalonvect: np.ndarray, vect: np.ndarray). С помощью пакета математических вычислений numpy это можно сделать, определив разность векторов, затем узнать количество ненулевых значений координат (признаков) этого вектора разности.

Далее определим класс **PixelDigit**(Classificator) – наследуемый методы от абстрактного класса **Classificator**. Этот класс содержит конструктор и дополнительный метод **recognite**(*self*) – для распознания полученных данных в цифру и проставления необходимых величин о расстоянии Хемминга и проценте похожести.

Далее определим результирующий класс показаний часов **Clock**. Этот класс содержит конструктор, позволяющий разбить показания часов, записанные как группу из пикселей 4 цифр с пробельными разделителями и логику распознавания каждой цифры в отдельности (с учетом невозможности, например, часа быть больше 24) а также общую логику распознавания показаний времени. Класс проводит описанную в лабораторной работе логику преобразованиями исходных данных в выходные.

Результаты работы программы представлены на рисунке 3.

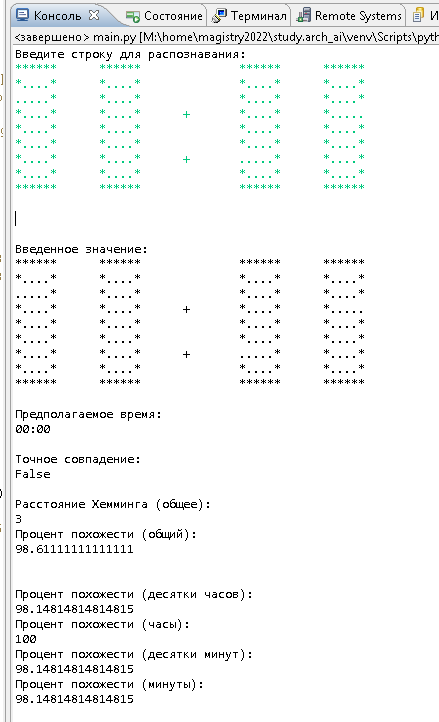


Рисунок 3. Результаты работы программы

# Вывод

В ходе лабораторной работы были изучены общие характеристики задачи распознавания и реализовано распознавание «битых» значений циферблата часов с использованием модели, основанной на вычислении оценок, близости объектов между собой.

# Приложение 1

Полный листинг программы main.py

*'''*

*Created on 15 сент. 2022 г.*

**@author:** *ASZabavin*

*'''*

import sys

import numpy as np

from collections import defaultdict

import itertools

import re

def **notempty**(x): return bool(x and not x.isspace())

def **lchunks**(lst, n):

for i\_ in range(0, len(lst), n):

yield lst[i\_:i\_ + n]

class **Classificator**:

*"""*

*Классификатор блоков пикселей (символов), записанных в текстовых файлах*

*При определении класс подгружает эталоны и определеяте производные значения*

*"""*

# Init Class

def **\_\_new\_\_**(cls, \*args, \*\*kwargs):

if cls is Classificator:

raise TypeError()

return super().\_\_new\_\_(cls)

ETALON\_PIXEL\_DATA = defaultdict(list)

MAX\_HAMM\_DIST = 0

MATRIX\_MAX\_X = 0

MATRIX\_MAX\_Y = 0

def **\_\_pixels\_load**(cls, filepath, moduletdict):

with open(filepath, *'r'*) as fl:

counter = 0

for line in fl.readlines():

if counter > 10:

fl.close()

return

if notempty(line):

line = line.strip()

moduletdict[counter].append(line)

else:

counter += 1

\_\_pixels\_load(None, *'etalon.txt'*, ETALON\_PIXEL\_DATA)

if len(ETALON\_PIXEL\_DATA):

MATRIX\_MAX\_X = len(ETALON\_PIXEL\_DATA[0][0])

MATRIX\_MAX\_Y = len(ETALON\_PIXEL\_DATA[0])

\_\_rl2 = MATRIX\_MAX\_Y // 2

MAX\_HAMM\_DIST = \_\_rl2 \* (MATRIX\_MAX\_X // 2) # Считаем что макс растояние Хемминга для одной цифры должно быть не более четвверти пикселей

pixels\_load = classmethod(\_\_pixels\_load)

# Other methods

*@staticmethod*

def **quartering**(digitsimb: list):

*"""*

*Поделитть пиксели на четверти*

**:param** *digitsimb: список вида:*

*[*

*'\*\*\*\*\*\*\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*....\*',*

*'\*\*\*\*\*\*\*',*

*]*

*"""*

x\_len = 0

y\_len = len(digitsimb)

if y\_len and notempty(digitsimb[0]):

x\_len = len(digitsimb[0])

x\_count = x\_len // 2

y\_count = y\_len // 2

devided\_x = []

for i in range(y\_len):

cnks = list(lchunks(digitsimb[i], x\_count))

cnks = [cnks[0], *''*.join(cnks[1:])]

devided\_x.append(cnks)

devided\_y = list(lchunks(devided\_x, y\_count))

devided\_y = [devided\_y[0], list(itertools.chain.from\_iterable(devided\_y[1:]))]

quadrants = []

for i, j in itertools.product(range(2), range(2)):

quadrants.append([x[j] for x in devided\_y[i]])

return quadrants

*@staticmethod*

def **bin\_vectorize\_block**(digitblock: list):

*"""*

*С помощью numpy предстваить список "пикселей" как бинарный вектор*

**:param** *digitblock: список строк пикселей типаЖ*

*[*

*'\*\*\*\*',*

*'\*..',*

*'\*..',*

*]*

*"""*

return np.array(list(map(lambda x: 1 if x == *'\*'* else 0, itertools.chain.from\_iterable(digitblock))))

*@staticmethod*

def **vect\_dist**(vect1: np.ndarray, vect2: np.ndarray):

*"""*

*Математическое растояние между векторами, (не пригодилось в задаче)*

**:param** *vect1:*

**:param** *vect2:*

*"""*

return np.linalg.norm(vect1 - vect2)

*@staticmethod*

def **hamm\_dist**(etalonvect: np.ndarray, vect: np.ndarray):

*"""*

*Вычисление расстояния хемминга между бинарными векторми с помощью numpy*

**:param** *etalonvect:*

**:param** *vect:*

*"""*

vect\_dif = etalonvect - vect # Разность векторов. Если совпадает координата - то на этой позиции будет 0

nz = np.nonzero(vect\_dif) # Выделить из вектора разности ненулевые значения и вернуть позции на которых они есть

sp = np.shape(nz[0]) # Размерность массива + длинна, в случае вектора кортеж из одного элемента

return sp[0]

*@staticmethod*

def **get\_quadrant\_koef\_for\_symbol**(symbol: str, quadrant: int):

*"""*

*Коэффициент (вес) четверти цифры, символа, (не пригодилось в задаче)*

**:param** *symbol:*

**:param** *quadrant:*

*"""*

return 1

*@staticmethod*

def **get\_generalized\_hamm\_dist**(symbol: str, quad\_0\_dist, quad\_1\_dist, quad\_2\_dist, quad\_3\_dist):

*"""*

*Общее расстояние Хемминга за символ (по четвертям сумма)*

**:param** *symbol:*

**:param** *quad\_0\_dist:*

**:param** *quad\_1\_dist:*

**:param** *quad\_2\_dist:*

**:param** *quad\_3\_dist:*

*"""*

qk = Classificator.get\_quadrant\_koef\_for\_symbol

koefs = [

qk(symbol, 0),

qk(symbol, 1),

qk(symbol, 2),

qk(symbol, 3),

]

return (koefs[0] \* quad\_0\_dist + koefs[1] \* quad\_1\_dist + koefs[2] \* quad\_2\_dist + koefs[3] \* quad\_3\_dist) / sum(koefs)

class **PixelDigit**(Classificator):

*"""*

*Субкласс Класификатора символов - цифер*

*"""*

def **\_\_init\_\_**(*self*, string\_with\_lb: str):

*"""*

*Инициализация с текстом с обрывами строк в котором записаны пиксели*

**:param** *string\_with\_lb:*

*"""*

*self*.digit = -1

*self*.exact = False

*self*.recog\_hamm\_dist = -1

*self*.simple\_hamm\_dist = -1

*self*.\_pixel\_data = list(map(lambda x: x.strip(), string\_with\_lb.splitlines()))

*self*.percent\_like = 0

*self*.recognite()

def **recognite**(*self*):

*"""*

*Распознать*

*"""*

digit\_hamings = []

digit\_simple\_hammings = []

for i in range(10):

etalon\_quads = *self*.quartering(*self*.ETALON\_PIXEL\_DATA[i])

reco\_quads = *self*.quartering(*self*.\_pixel\_data)

vb = *self*.bin\_vectorize\_block

digit\_simple\_hammings.append(*self*.hamm\_dist(vb(*self*.ETALON\_PIXEL\_DATA[i]), vb(*self*.\_pixel\_data)))

hammings = [

*self*.hamm\_dist(vb(etalon\_quads[0]), vb(reco\_quads[0])),

*self*.hamm\_dist(vb(etalon\_quads[1]), vb(reco\_quads[1])),

*self*.hamm\_dist(vb(etalon\_quads[2]), vb(reco\_quads[2])),

*self*.hamm\_dist(vb(etalon\_quads[3]), vb(reco\_quads[3])),

]

genham = *self*.get\_generalized\_hamm\_dist(i, \*hammings)

if genham == 0 == hammings[0] == hammings[1] == hammings[2] == hammings[3]:

*self*.recog\_hamm\_dist = 0

*self*.exact = True

*self*.digit = i

*self*.simple\_hamm\_dist = digit\_simple\_hammings[i]

*self*.percent\_like = 100

return

digit\_hamings.append(genham)

*self*.exact = False

*self*.recog\_hamm\_dist = min(digit\_hamings)

*self*.digit = digit\_hamings.index(*self*.recog\_hamm\_dist)

*self*.simple\_hamm\_dist = digit\_simple\_hammings[*self*.digit]

if *self*.simple\_hamm\_dist > *self*.MAX\_HAMM\_DIST:

*self*.digit = -1

*self*.percent\_like = (

((*self*.MATRIX\_MAX\_X \* *self*.MATRIX\_MAX\_Y) -

*self*.simple\_hamm\_dist) \* 100 /

(*self*.MATRIX\_MAX\_X \* *self*.MATRIX\_MAX\_Y)

) if *self*.simple\_hamm\_dist > 0 else 100

def **\_\_eq\_\_**(*self*, other):

*"""*

*Перегрузка операции равенства*

**:param** *other:*

*"""*

return True if (*self* - other) < *self*.MAX\_HAMM\_DIST else False

def **\_\_sub\_\_**(*self*, other):

*"""*

*Перегрузка операции разности (отличия) - расстояние Хемминга между цифрами*

**:param** *other:*

*"""*

etalonvect = *self*.bin\_vectorize\_block(*self*.ETALON\_PIXEL\_DATA[*self*.digit])

othervect = *self*.bin\_vectorize\_block(*self*.\_pixel\_data)

return *self*.hamm\_dist(etalonvect, othervect)

class **Clock**:

*"""*

*Класс Показаний часов*

*"""*

def **devide\_digits**(*self*, string\_time: str):

lipixels = string\_time.splitlines(False)

maxx = Classificator.MATRIX\_MAX\_X

maxy = Classificator.MATRIX\_MAX\_Y

time\_digits = defaultdict(str)

for i in range(len(lipixels)):

lipixels[i] = re.sub(*'[^.\*]+'*, *''*, lipixels[i])

cnks = list(lchunks(lipixels[i], maxx))

time\_digits[0] = *'\n'*.join([cnks[0], time\_digits[0]])

time\_digits[1] = *'\n'*.join([cnks[1], time\_digits[1]])

time\_digits[2] = *'\n'*.join([cnks[2], time\_digits[2]])

time\_digits[3] = *'\n'*.join([cnks[3], time\_digits[3]])

return time\_digits[0], time\_digits[1], time\_digits[2], time\_digits[3]

def **\_\_init\_\_**(*self*, string\_time: str):

*"""*

*Передаем значение показаний записанное с пробельными разделитлями, 4 цифры, по типу:*

*\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\**

*\*....\* \*....\* \*....\* \*....\**

*.....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* + \*....\* \*.....*

*\*....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* \*....\* \*....\**

*\*....\* \*....\* + .....\* \*....\**

*\*....\* \*....\* \*....\* \*....\**

*\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\**

**:param** *string\_time:*

*"""*

*self*.\_d1, *self*.\_d2, *self*.\_d3, *self*.\_d4 = *self*.devide\_digits(string\_time)

*self*.first\_digit = PixelDigit(*self*.\_d1)

*self*.second\_digit = PixelDigit(*self*.\_d2)

*self*.third\_digit = PixelDigit(*self*.\_d3)

*self*.four\_digit = PixelDigit(*self*.\_d4)

# Поправки для контекста "Часы и минуты"

if *self*.first\_digit.digit > 2:

*self*.first\_digit.digit = -1

else:

if *self*.first\_digit.digit == 2 and *self*.second\_digit.digit > 4:

*self*.second\_digit.digit = -1

if *self*.third\_digit.digit > 5:

*self*.third\_digit.digit = -1

if *self*.first\_digit.exact == *self*.second\_digit.exact == *self*.third\_digit.exact == *self*.four\_digit.exact:

*self*.exact = *self*.first\_digit.exact

else:

*self*.exact = False

# Значения хемминга для показаний времени

*self*.simple\_hamm\_dist = *self*.first\_digit.simple\_hamm\_dist + *self*.second\_digit.simple\_hamm\_dist + *self*.third\_digit.simple\_hamm\_dist + *self*.four\_digit.simple\_hamm\_dist

*self*.percent\_like = (

(((Classificator.MATRIX\_MAX\_X \* Classificator.MATRIX\_MAX\_Y) \* 4) -

*self*.simple\_hamm\_dist) \* 100 /

((Classificator.MATRIX\_MAX\_X \* Classificator.MATRIX\_MAX\_Y) \* 4)

) if *self*.simple\_hamm\_dist > 0 else 100

def **\_\_str\_\_**(*self*):

if *self*.first\_digit.digit != -1 and *self*.second\_digit.digit != -1 and *self*.third\_digit.digit != -1 and *self*.four\_digit.digit != -1:

return *f'{self.first\_digit.digit}{self.second\_digit.digit}:{self.third\_digit.digit}{self.four\_digit.digit}'*

return *'[Невозможно определить]'*

time = property(\_\_str\_\_)

if \_\_name\_\_ == *"\_\_main\_\_"*:

print(*'Введите строку для распознавания: '*)

contents = []

while True:

try:

line = input()

except EOFError:

break

if not notempty(line):

break

contents.append(line)

txt = *'\n'*.join(contents)

rec\_time = Clock(txt)

print(*f'''\n*

*Введенное значение:*

*{txt}*

*Предполагаемое время:*

*{rec\_time.time}*

*Точное совпадение:*

*{rec\_time.exact}*

*Расстояние Хемминга (общее):*

*{rec\_time.simple\_hamm\_dist}*

*Процент похожести (общий):*

*{rec\_time.percent\_like}*

*Процент похожести (десятки часов):*

*{rec\_time.first\_digit.percent\_like}*

*Процент похожести (часы):*

*{rec\_time.second\_digit.percent\_like}*

*Процент похожести (десятки минут):*

*{rec\_time.third\_digit.percent\_like}*

*Процент похожести (минуты):*

*{rec\_time.four\_digit.percent\_like}*

*'''*)