



DOI: 10.32517/2221-1993-2022-21-3-67-74

**Н. Н. Самылкина***Московский педагогический государственный университет, г. Москва, Россия***И. А. Калинин***Московский государственный лингвистический университет, г. Москва, Россия*

## РЕШЕНИЕ ЗАДАЧ ОТБОРОЧНОГО И ОСНОВНОГО ЭТАПОВ ПЕРВОЙ ВСЕРОССИЙСКОЙ ОЛИМПИАДЫ ПО ИСКУССТВЕННОМУ ИНТЕЛЛЕКТУ ДЛЯ УЧАЩИХСЯ VIII—XI КЛАССОВ

### Аннотация

В статье рассматриваются подходы к разработке системы олимпиадных заданий, особенности проведения мероприятия, а также решения заданий двух этапов первой всероссийской олимпиады по искусственному интеллекту для VIII—XI классов, проведенной в октябре-ноябре 2021 года Министерством просвещения Российской Федерации в рамках Федерального проекта «Искусственный интеллект» национальной программы «Цифровая экономика Российской Федерации». Тематика заданий олимпиады — обработка текста на естественном языке (Natural Language Processing) и компьютерное зрение (Computer Vision). На всех этапах олимпиады использовался язык программирования Python со специализированными библиотеками для решения задач искусственного интеллекта. Материал с разбором заданий будет полезен при обучении информатике в школе на углубленном уровне, в том числе при подготовке школьников к участию в следующей олимпиаде по искусственному интеллекту в индивидуальном формате.

**Ключевые слова:** всероссийская олимпиада, искусственный интеллект, Python, специализированные библиотеки языка Python, углубленное изучение информатики.

### 1. Актуальность подготовки школьников в области искусственного интеллекта

Осенью 2021 года Министерством просвещения Российской Федерации в рамках Федерального проекта «Искусственный интеллект» национальной программы

«Цифровая экономика Российской Федерации» [5] была проведена первая всероссийская олимпиада по искусственному интеллекту (сайт олимпиады: <https://olimp.edsoo.ru/>). В России традиционно сильная олимпиадная подготовка школьников по многим предметам, в том числе по информатике. Минпросвещения России,

### Контактная информация

**Самылкина Надежда Николаевна**, доктор пед. наук, доцент, профессор кафедры теории и методики обучения математике и информатике, Институт математики и информатики, Московский педагогический государственный университет, г. Москва, Россия; *адрес:* 119991, Россия, г. Москва, ул. Малая Пироговская, д. 1, стр. 1; *e-mail:* nsamylkina@yandex.ru

**Калинин Илья Александрович**, канд. пед. наук, доцент, начальник управления информатизации, доцент кафедры международной информационной безопасности, Институт информационных наук, Московский государственный лингвистический университет, г. Москва, Россия; *адрес:* 119034, Россия, г. Москва, ул. Остоженка, д. 38, стр. 1; *e-mail:* llyakalinin@mail.ru

**N. N. Samylkina**

Moscow Pedagogical State University, Moscow, Russia

**I. A. Kalinin**

Moscow State Linguistic University, Moscow, Russia

### SOLVING THE PROBLEMS OF THE PRELIMINARY AND MAIN STAGES OF THE FIRST ALL-RUSSIAN OLYMPIAD IN ARTIFICIAL INTELLIGENCE FOR STUDENTS OF 8TH—11TH GRADES

#### Abstract

The article discusses approaches to developing a system of Olympiad tasks, features of the event, as well as solving tasks of two stages of the first All-Russian Olympiad in artificial intelligence for 8<sup>th</sup>—11<sup>th</sup> grades, held in October-November 2021 by the Ministry of Education of the Russian Federation as part of the Federal Project "Artificial Intelligence" of the National Program "Digital Economy of the Russian Federation". The topics of the Olympiad tasks were Natural Language Processing and Computer Vision. At all stages of the Olympiad, the Python programming language with specialized libraries for solving artificial intelligence problems was used. The material with the analysis of tasks will be useful when teaching informatics at an advanced level at school, including when preparing schoolchildren to participate in the next Olympiad in artificial intelligence in an individual format.

**Keywords:** All-Russian Olympiad, artificial intelligence, Python, specialized Python libraries, advanced study of informatics.

инициируя новую олимпиаду, ставило целью создание условий для поддержки одаренных школьников, а именно: выявление и развитие творческих способностей, развитие познавательного интереса обучающихся, повышение у них мотивации к углубленному изучению информатики и математики как основе предпрофессиональной подготовки в области информационных технологий.

Тематика искусственного интеллекта как сквозной цифровой технологии поддерживается государственными структурами и активно отражается в научных публикациях и учебной литературе (см., например, [1–4, 6, 7, 9, 10]). Представление этой темы расширилось в федеральных государственных образовательных стандартах общего образования всех уровней. Готовится учебно-методическая литература для обучающихся и педагогов по искусственному интеллекту. Тематика искусственного интеллекта включена в обязательную подготовку будущих учителей информатики на уровне как бакалавриата, так и магистратуры. Несколько лет проводятся вузовские олимпиады по искусственному интеллекту для школьных команд. Появляются доступные для школьников цифровые ресурсы для подготовки к олимпиадам по искусственному интеллекту.

Всероссийская олимпиада по искусственному интеллекту для VIII–XI классов общеобразовательных организаций проводилась в октябре–ноябре 2021 года впервые, и льготы для ее победителей при поступлении в вузы еще не предусматривались.

Разработчики заданий выбрали наиболее актуальные в настоящее время направления искусственного интеллекта — обработка текста на естественном языке (Natural Language Processing) и компьютерное зрение (Computer Vision).

На всех этапах олимпиады использовался язык программирования Python. Это пока единственный язык программирования, имеющий специализированные библиотеки для решения задач искусственного интеллекта.

## 2. Особенности проведения олимпиады. Система заданий

Первая всероссийская олимпиада по искусственному интеллекту для VIII–XI классов **состояла из трех этапов и проводилась в дистанционном режиме.**

Для обеспечения прозрачности контингента участников предполагалась их регистрация через образовательные организации, которые своим приказом направляют своих обучающихся для участия в олимпиаде. При этом допускалась возможность доступа для учащихся к контенту олимпиады на площадках образовательных организаций при консультативной поддержке учителей информатики.

В олимпиаде рассматривались индивидуальные решения заданий, т. е. это были **индивидуальные соревнования.**

Решенные задания участников проходили **полностью автоматизированную проверку**, при этом участник не видел результата выполнения своей программы, а видел только оценку соответствия ее результатов предусмотренным критериям по одному из двух вариантов:

- 1) количество правильно пройденных тестовых наборов данных (входные данные, предусмотренные в условии задания);
- 2) процентное соотношение результатов работы программы реальному набору данных.

Содержательно олимпиадные задания всех этапов были тематически и инструментально взаимосвязаны, что позволяло каждому участнику нарастить свои тематические и программистские навыки.

**На первом, отборочном, этапе** олимпиады предлагалось 10 заданий для проверки навыков программирования на Python. Проверялись навыки:

- ввода данных и вывода результатов при помощи файлов;
- обработки числовой и текстовой информации, в том числе работы с различными кодировками;
- планирования структуры хранения данных для последующей обработки этих данных.

Можно было использовать только стандартные библиотеки языка.

Более половины заданий были ориентированы на использование регулярных выражений при обработке текста — именно этот ключевой инструмент требуется при решении множества задач на обработку текста. Поэтому на первом этапе проверялись навыки использования регулярных выражений пока без использования специализированных библиотек [8].

Задания можно было решать в любом порядке и направлять для проверки в любое время в течение пяти дней.

**На втором, основном, этапе** олимпиады предлагалось пять заданий на использование специализированных библиотек: Numpy, Pandas, Pillow, sklearn, NLTK, Apyory.

Тематика заданий расширилась — к заданиям на обработку текста добавились задания на компьютерное зрение и анализ данных.

Проверялись навыки грамотного использования указанных библиотек, ориентированных на работу с числовыми, текстовыми и графическими данными, с ориентацией на подготовку и анализ наборов данных для последующего решения задач интеллектуальной обработки.

Все алгоритмы задач требовали не только наличия у участников программистских навыков, но и вдумчивого анализа, подбора и настройки методов выделения и анализа данных.

К тому же изменился регламент работы: решения необходимо было направить на проверку в любой из пяти соревновательных дней, но в один непрерывный временной промежуток — четыре часа.

Таким образом, **к третьему, заключительному, этапу** участники были подготовлены к необходимости самостоятельной разработки программной модели обработки данных в соответствии с заданием.

Участникам было предложено два задания по темам «Компьютерное зрение» (сегментация изображений) и «Обработка текста на естественных языках» (задача оценки тональности).

Подготовка и обучение каждой из моделей могли занимать по три часа, поэтому участникам было дано достаточное время для необходимой корректировки мо-

делей — 12 часов непрерывной работы в течение любых пяти соревновательных дней.

Перед проведением очередного этапа олимпиады и после завершения соревновательных дней проводились вебинары для участников олимпиады. На этих встречах разработчики заданий рассматривали типовые задания, рациональные приемы решения и отвечали на вопросы участников. Условия всех задач и записи вебинаров можно посмотреть на сайте олимпиады.

В данной статье мы постарались разобрать задания, решения к которым не представлены на сайте олимпиады.

### 3. Разбор и решение задач отборочного и основного этапов олимпиады

Рассмотрим решение двух задач первого, отборочного, этапа (задачи 1 и 2).

#### Задача 1.

Лена и Вася собираются в Красивый Город на восемь дней и имеют на руках некоторую сумму денег (пока они не знают, сколько).

Они хотят посетить самые интересные места. О каждом месте они знают, насколько оно интересно (в баллах) и сколько стоит его посетить (в деньгах), — список есть в файле, имя которого вводится с клавиатуры.

К сожалению, они могут посещать только два места в день, а в списке — до 24 мест.

Разработайте программу, которая получит с клавиатуры максимальную сумму, имя файла-справочника и выведет на экран список мест, которые им стоит посетить, — максимальной интересности (в сумме баллов) и допустимо по деньгам. Места в списке надо перечислить через точку с запятой, в том числе в конце списка. Порядок вывода мест должен совпадать с порядком в исходном списке.

*Тест работоспособности.*

*Ввод:*

```
8000
test1.txt
```

*Вывод:*

```
Александр-Невская Лавра;Большой каскад
в Петергофе;Дворец «Марли»;Дворец
Монплеизир;Домик Петра I;Крейсер «Аврора»;
Летний дворец Петра I;Мариинский дворец;
Памятник Медный всадник;Смольный собор
Санкт-Петербурга; Собор Петра и Павла
в Петергофе;Спас на Крови;Чижик-пыжик;
```

*Содержание файла test1.txt:*

```
Александринский театр;8;1500
Александр-Невская Лавра;7;300
Большой каскад в Петергофе;8;100
Государственный Русский музей;10;2000
Дворец «Марли»;8;1000
Дворец Монплеизир;8;1000
Домик Петра I;8;1000
Зимний дворец;10;2000
```

```
Исаакиевский собор;9;1500
Казанский собор в Петербурге;9;1500
Крейсер «Аврора»;8;1000
Кунсткамера, или Кабинет редкостей;9;3000
Летний дворец Петра I;8;1000
Мариинский дворец;7;1000
Мариинский театр;9;4000
Михайловский дворец;7;1500
Музей Достоевского;8;2000
Памятник Медный всадник;8;100
Петропавловская крепость;9;2000
Смольный собор Санкт-Петербурга;8;200
Собор Петра и Павла в Петергофе;7;300
Спас на Крови;8;500
Чижик-пыжик;8;50
Эрмитаж;10;3500
```

#### *Разбор решения задачи.*

Задача представляет собой вариант классической переборной задачи, которую в олимпиадной литературе называют «задачей о дискретном рюкзаке».

Общий принцип решения задачи несложен: мы перебираем все возможные варианты, откидываем неподходящие варианты, а подходящие сравниваем по заданному критерию.

Самая простая часть задачи, практически не требующая пояснений, — это чтение файла с вариантами:

```
maxPrice = int(input(''))
fileName = input()

places = []
count = 0

with open(fileName, 'r', encoding='utf-8') as f:
    for line in f:
        stringDescribe = line.strip().split(';')
        places.append([stringDescribe[0],
                        int(stringDescribe[1]),
                        int(stringDescribe[2])])
        count = count + 1
```

Мы просто создаем список мест, в котором каждое место описывается списком из трех элементов: названием, количеством баллов, стоимостью.

Для организации перебора мы воспользуемся простым фактом: каждое место мы либо посещаем, либо нет. То есть факт посещения, очевидно, кодируется одним битом. Тогда мы можем просто договориться, что возьмем целое число, в котором каждый бит будет обозначать одно место — по номеру позиции.

Тогда полный перебор будет просто процедурой перебора всех чисел, которые уместятся в заданное количество бит (начиная с 1 — все-таки в одно место мы точно заедем). В нашем условии — максимум 24 места, т. е. 24 бита.

Остается подготовить функцию оценки каждого варианта.

В функции нам прежде всего надо будет оценить, не слишком ли много мест в предложенном варианте.

Для ускорения этого процесса подготовим небольшой массив, в котором каждому варианту значения 1 байта соответствует количество заполненных в нем битов:

```
import array

BitCount8 = array.array('I')

for i in range (256):
    BitCount8.extend( [(1 if i & 1 > 0 else 0)+
                      (1 if i & 2 > 0 else 0)+
                      (1 if i & 4 > 0 else 0)+
                      (1 if i & 8 > 0 else 0)+
                      (1 if i & 16 > 0 else 0)+
                      (1 if i & 32 > 0 else 0)+
                      (1 if i & 64 > 0 else 0)+
                      (1 if i & 128 > 0 else 0)])
```

Теперь, собственно, сама функция:

```
def calculateVariant ( varNum ) :
    global BitCount8 # Подготовленный заранее
                        массив для подсчета
    global places     # Подготовленный заранее
                        список мест
    global maxPrice

    current = 1
    price = 0
    ball = 0
    i=0
    placeCount = BitCount8[ varNum & 0xFF ]+ \
                BitCount8[ (varNum >> 8)& 0xFF ]+ \
                BitCount8[ (varNum >> 16)& 0xFF ]
    if placeCount > 16: # Не более 2 мест в день,
                        в течение 8 дней
        return (0,0)
    while current <= varNum and price <= maxPrice:
        if varNum & current != 0:
            price += places[i][2]
            ball += places[i][1]
        i += 1
        current = current << 1
    if price > maxPrice:
        return (0,0)
    return (price, ball)
```

Функция получит номер (фактически — описание) варианта посещения разных мест и вернет кортеж из двух чисел — стоимость варианта и «интересность» в баллах.

В этой функции мы начинаем с подсчета бит, равных 1, т. е. с подсчета мест, которые в этом варианте надо посетить. Сделаем это с помощью битовых операций, выбрав байты и сложив количество битов в них (берем количество из массива).

Если их больше 16 — возвращаем кортеж (0,0), т. е. вариант не рассматриваем и время на подсчеты не тратим.

Теперь в цикле перебираем все биты, начиная с первого (current = 1), пока не дойдем до конца, т. е. пока число с установленным «текущим» битом не станет больше номера варианта или пока не будет превышен максимум стоимости.

Если логическая операция «И» с этим битом нашего варианта дает не нулевое значение — место надо учесть, т. е. добавить его баллы и его стоимость к накопленной оценке.

В основном коде программы перебор довольно прост:

```
maxThings = 1 << count # Максимальный вариант
                        2 в степени количество

curVar = 1
maxBalls = 0
bestVariant = 0

while curVar < maxThings:
    curPrice, curBalls =
        calculateVariant ( curVar )
    if curPrice <= maxPrice and
        curBalls > maxBalls:
        bestVariant = curVar
        maxBalls = curBalls
    curVar += 1
```

В итоге мы выбираем вариант с максимальным «уровнем интереса», заранее оценив как «совсем не интересные» варианты, не соответствующие условию: слишком долгие или слишком дорогие.

Остается вывести найденный вариант, в соответствии с условием задачи:

```
pos = 0
while ( 1 << pos) <= bestVariant:
    if bestVariant & (1 << pos) > 0:
        print( places[pos][0], end=';')
    pos += 1
```

## Задача 2.

В издательство прислали два произведения разных авторов в жанре фантастики с множеством уникальных слов. Требуется выявить совпадения по уникальным словам. Программисту издательства поставили эту задачу в следующей формулировке:

«Даны имена двух больших (до 10 Мбайт) текстовых файлов в кодировке UTF-8 с текстом на русском языке.

Написать программу, которая введет с клавиатуры имена файлов и выдаст на экран списки 50 уникальных слов первого файла (по убыванию частоты) и 50 уникальных слов второго файла (по убыванию частоты), используя не более чем 5 Мбайт памяти.

Слова вывести через один пробел».

*Тест работоспособности.*

*Ввод:*

```
Test0-1.txt
Test0-2.txt
```

*Пример вывода:*

было эти записки у руки условиях частично почерка  
эксперт его хранились дело всякий таких папке  
машинописный написанный от документ текст которых  
разными многие в-листы кое-где оба двух запись  
уверяет он левши то есть ли были переданы лицом  
пожелавшим чтобы предавались гласности покуда  
будет жив передавший указанное лицо изложило  
мотивы пожелания  
записок автора имя такого имени всех трех  
названия наибольшую сложность представляло  
приведение форме последовательного изложения

разработать применить приемов алгоритмизации приведения сведения приведенные документах настолько недостаточны или два почувствовал себя обязанным сделать вставки дабы пояснить канву хотя меру понимания приношу покорнейшие извинения читателям пронизательным теперь об названиях текстах упоминается разу почти

### Разбор решения задачи.

Задача несложная, но требующая некоторых навыков организации анализа текстовых файлов.

Начнем с ввода исходных данных и подготовки к работе:

```
import re

firstWord = {}
secondWord = {}
firstFileName = input()
secondFileName = input()
```

Мы будем использовать для деления текста на слова механизм регулярных выражений, а сами слова хранить в двух словарях — для ускорения проверки.

Сразу введем имена файлов, чтобы не столкнуться с потерей второго имени из-за времени обработки первого файла.

Теперь создадим словарь слов первого файла:

```
with open(firstFileName, 'r', encoding='utf-8')
    as firstFile:
    for line in firstFile:
        line = line.strip().lower()
        for fword in re.findall("[A-яa-я]+")
            ([A-яa-я]+)?", line):
            word = ''.join(fword) # Объединяем
                                группы
            if not (word in firstWord):
                firstWord[word] = 1
            else:
                firstWord[word] =
firstWord[word] + 1
```

С точки зрения формальной обработки файла мы считаем «словом» просто последовательность символов между разделителями. Обратите внимание: в русском языке есть двойные слова, которые записываются через дефис (напомним, что дефис — это разделение именно частей слова). К сожалению, в текстовых файлах, особенно старых, тире и дефис записываются одним и тем же символом, так что будем считать, что если есть знак «-» без пробелов вокруг, то это дефис.

Наше регулярное выражение отбирает именно русские слова (т. е. слова из букв русского алфавита) и предполагает, что в слове может быть несколько групп — как минимум одна и еще несколько через тире. Поэтому мы находим вхождения этого шаблона — регулярного выражения, а потом объединяем все группы в одно слово.

Дальше мы просто проверяем: если слово в словаре есть, мы увеличиваем счетчик его вхождений, если нет — добавляем в словарь.

С анализом второго файла все немного сложнее. Встретив слово во втором файле, мы должны проверить, встречается ли оно в первом. Если встречается — оно не

уникально для обоих файлов. Но просто удалить слово из первого словаря и не добавлять во второй недостаточно, потому что оно может появиться и второй раз, и третий, и т. д. Поэтому мы заводим еще один словарь — словарь удаленных слов, куда и будем помещать удаляемые слова из первого словаря, и при анализе проверим не один, а два словаря.

```
with open(secondFileName, 'r', encoding='utf-8')
    as secondFile:
    for line in secondFile:
        line = line.strip().lower()
        for fword in re.findall("[A-яa-я]+")
            ([A-яa-я]+)?", line):
            word = ''.join(fword)
            if word in firstWord: # Было ли
                                слово в первом файле?
                deletedWord.append(word) # Если
                                да - оно не уникально
                del firstWord[word] # ... и из
                                словаря его убрать
            else:
                if not word in deletedWord:
                    if word in secondWord:
                        secondWord[word] =
                        secondWord[word] + 1
                    else:
                        secondWord[word] = 1
```

Словари сформированы, остается их отсортировать и вывести в соответствии с условием:

```
sortedFirst = [k for k, v in
                sorted(firstWord.items(), \
                        key=lambda item: item[1], reverse=True)]
print(' '.join(sortedFirst[0:50]))
sortedSecond = [k for k, v in
                sorted(secondWord.items(), \
                        key=lambda item: item[1], reverse=True)]
print(' '.join(sortedSecond[0:50]))
```

Приводить в статье содержимое проверочных файлов было бы затруднительно, работу программ читатель вполне может проверить на своих примерах.

Далее рассмотрим решение двух задач второго этапа (задачи 3 и 4).

### Задача 3.

Имеется база отзывов на фильмы в виде общего каталога с текстовыми файлами, разделенными внутри на три каталога — pos, neg и neu. В каждом из этих каталогов несколько (> 10) текстовых файлов на русском языке в кодировке UTF-8. Выведите через один пробел список из 50 самых информативных слов, т. е. тех, которые лучше всего позволят различать, к какой категории относятся отзывы. Информативность слов оценивают с помощью критерия взаимной информации ([https://ru.abcdef.wiki/wiki/Mutual\\_information#Conditional\\_mutual\\_information](https://ru.abcdef.wiki/wiki/Mutual_information#Conditional_mutual_information)).

*Тест работоспособности.*

*Ввод:*

dataset

*Вывод:*

так это как части же теперь плохо они но бы он эпизод меликян исключительно общество никакого того между очень или веры мессия чернобыль аварии из серии его показать во меня вот ощущение где начинается достойно сделал лжи смогли место от чтобы вы для до интервью ради фильм есть может которая

*Разбор решения задачи.*

Несмотря на пугающий вид формул, приведенных по указанной в условии задачи ссылке, общая идея «взаимной информации» довольно проста: если у нас есть два набора данных со сходными наборами признаков, то мы можем определить, какие из этих признаков вносят больший вклад в отделение одного набора от другого.

В случае наборов текстов такими признаками являются слова. Очевидно, что, если слово встречается в одном наборе текстов 50 раз, а в другом — 51, оно малоинформативно, а вот если в одном наборе оно встречается 11 раз, а во втором — 205, очевидно, что слово гораздо более информативно.

Выглядит нужная формула довольно громоздко, но реализовывать ее вычисление самостоятельно и не требовалось — вычисление необходимой характеристики есть в составе библиотеки ScikitLearn.

Основная цель задачи — позволить участникам (тем, кто не поленится поискать необходимую информацию и решить «скучную» задачу) подготовиться к решению одной из задач третьего тура. В частности, изучив набор данных.

Начинаем, как всегда, с подключения библиотек:

```
import re
import os
from sklearn.feature_selection
    import mutual_info_classif
from sklearn.feature_extraction.text
    import CountVectorizer
```

Определим регулярное выражение, выделяющее слова:

```
wordSplit = re.compile(r"[А-Яа-я]+")
```

Оно проще, чем то, что мы использовали ранее, но для выделения наиболее информативных слов вполне подойдет (можно взять более сложное, но на результат это не повлияет — двойные слова встречаются сравнительно редко).

Каждый файл нам надо будет представить в виде единой текстовой строки слов. Файлов много, так что напишем для этого несложную функцию:

```
def preprocess_file( filename, enc):
    words = ''

    for string in open(filename, 'r', encoding=enc):
        for word in re.
            findall(wordSplit, string):
                words = words + ' '+word
    return words.strip()
```

Как видно, функция получает имя файла, открывает этот файл, выделяет из него слова и собирает их в длин-

ную строку. Строка будет для нас частью нашего набора данных в последующей обработке.

```
basePath = input()
paths = ['pos', 'neg', 'neu']
corpus = []
target = []

for path in paths:
    for file in os.listdir('./'+basePath+'/'+
        path):
        corpus.append(
preprocess_file('./'+basePath+'/'+path+'/'+
            file, "utf-8")
        )
        target.append(path)
```

В основной программе мы анализируем наш набор данных, т. е. готовим корпус текстов (тех самых строк) и отдельно список классов, к которым каждый текст относится. Поскольку файлов много и нам дают только название корневого каталога, мы организуем получение имен файлов из каждого подкаталога — по имени класса. Обрабатываем файл и добавляем его в список corpus, а его класс — в список target.

Чтобы преобразовать корпус к нужному нам виду, т. е. к матрице, где каждый документ — это отдельная строка, состоящая из количества вхождений каждого слова (те слова, которых нет в конкретном документе, имеют 0 вхождений), мы воспользуемся векторизатором (средство превращения документа в вектор, т. е. в набор чисел), который и обучим на нашем корпусе:

```
cv = CountVectorizer(max_df=0.95, min_df=2)
corpus_vec = cv.fit_transform(corpus)
```

Векторизатору мы указываем, что не будем обращать внимания на слова, которые встречаются в 95 % документов корпуса или встречаются всего один раз, — информативность у таких слов гарантированно низкая.

После этого мы можем обучить собственно классификатор взаимной информации:

```
res = dict(zip(cv.get_feature_names(),
mutual_info_classif(corpus_vec, target,
discrete_features=True)
))
```

Результат мы собираем в виде словаря, в котором каждому слову будет соответствовать информативность. Сами слова мы получим уже как признаки из подготовленного нами векторизатора: cv.get\_feature\_names() и соберем в единый словарь с показателями их информативности (функция zip создаст список парных кортежей, который мы и преобразуем в словарь).

Остается отсортировать словарь и выдать результат:

```
wordsGain = [k for k, v in sorted(res.items(),
    key=lambda item: item[1], reverse=True)]

print(' '.join(wordsGain[0:50]))
```

**Задача 4.**

Есть фото стенки до начала покраски и во время покраски, в формате JPEG.

Написать программу, которая получит с клавиатуры название файла с участком непокрашенной стены, название файла с покрашенной стеной и определит, сколько процентов стены уже покрашено. Учтите, что ровным и точным цвет не бывает.

Ответ вывести в целых процентах, возможно отклонение до 2 % включительно. Знак «%» выводить не надо.

*Тест работоспособности.*

*Ввод:*

wall1-1.jpg

wall1.jpg

*Функция оценки ответа:*

```
lambda answer: 1 if abs(answer-74)<2 else 0
```

**Разбор решения задачи.**

Очевидная (нехарактерная для обычного школьного курса информатики) особенность этой задачи — работа с графической информацией, причем не со сгенерированным простым примером, а с участком реальной фотографии. При решении данной задачи нам придется учесть, что в реальности при покраске стены стена не становится ровной, а значит, и о ровном точном цвете говорить нельзя.

Тем не менее изображение, как мы знаем, представляется в виде матрицы пикселей. Если сформулировать решение задачи, то можно сказать, что наша задача сводится к подсчету «покрашенных» и «непокрашенных» точек.

Если мы посмотрим на примеры такой частично покрашенной стены (достаточно воспользоваться поиском в сети), то можно отметить очевидный факт — недокрашенная стена делится на две части, которые заметно отличаются друг от друга. Таким образом, нам нужно найти способ разделить пиксели на два класса и выяснить, какой из них больше похож на ранее данный пример.

Способов это сделать достаточно много, но быстрее всего будет исходить из того, что возможности «учить» такой классификатор у нас не будет (фото очень разнообразны), так что выберем простой метод обучения без учителя.

Самый известный метод, который позволяет быстро разделить множество объектов на классы, — метод К-средних. Суть его в том, что мы делим имеющиеся у нас объекты на заданное количество групп и каждую описываем наиболее характерной «средней» точкой.

Для этого прежде всего подключим нужные библиотеки:

```
from PIL import Image
import numpy as np
import math
from sklearn.cluster import KMeans
```

Используем для работы с изображениями библиотеку Pillow (она предусматривалась правилами), а для разбиения на кластеры — класс KMeans из библиотеки ScikitLearn. Сначала загрузим изображения:

```
imgBaseFile = Image.open(input()) # Исходная
                                стенка
img = Image.open(input()) # Покрашенная стенка
```

Теперь найдем, исходя из условия, «средний» пиксель для неокрашенной стены:

```
colorPixelBase = [0,0,0,0]
for i in range(imgBaseFile.size[0]):
    for j in range(imgBaseFile.size[1]):
        pixel = imgBaseFile.getpixel((i, j))
        colorPixelBase[0] += pixel[0]
        colorPixelBase[1] += pixel[1]
        colorPixelBase[2] += pixel[2]
        colorPixelBase[3] += 1
```

Для этого создадим список из четырех элементов — суммы трех базовых цветов и общего количества пикселей. Используя Pillow, получим каждый пиксель и прибавим его к накопленным значениям. После чего останется усреднить компоненты:

```
colorPixelVector =
    ( colorPixelBase[0]/colorPixelBase[3],
      colorPixelBase[1]/colorPixelBase[3],
      colorPixelBase[2]/colorPixelBase[3]
    )
```

Далее сформируем массив пикселей уже второй картинки. Его не получится просто усреднить — его нужно будет делить на кластеры. Поэтому мы сформируем список пикселей (мы выделяем кластеры, исходя только из цвета, поэтому положение не важно) и преобразуем его в массив библиотеки Numpy (именно с такими массивами работает KMeans):

```
colorPixel = []
for i in range(img.size[0]):
    for j in range(img.size[1]):
        pixel = img.getpixel((i, j))
        colorPixel.append(pixel)
colors = np.array( colorPixel )
```

Теперь можем выделить нужное количество кластеров (выучить модель):

```
kmeans = KMeans(n_clusters = 2)
kmeans.fit(colors)
```

Получив два кластера, мы можем посчитать, сколько пикселей в каждом из них. Для этого создадим словарь и посчитаем количество упоминаний каждого полученного кластера:

```
clusterSize = {}
for label in kmeans.labels_:
    if label in clusterSize:
        clusterSize[label] += 1
    else:
        clusterSize[label] = 1
```

Но поскольку нам надо сравнивать с образцом, то необходимо выяснить, кто из них кто. Самый очевидный способ это сделать — посчитать евклидово расстояние: кто ближе в пространстве цветов к нашему образцу, тот и представляет неокрашенную стенку.

```
minDistanse = 255 * 3**0.5
label = -1
order = 0
```

```

for l in kmeans.cluster_centers_:
    distance = ((l[0]-colorPixelVector[0])**2+
                (l[1]-colorPixelVector[1])**2+
                (l[2]-colorPixelVector[2])**2 )**0.5
    if distance < minDistance:
        minDistance = distance
        label = order
    order += 1

```

После этого остается только вычислить и вывести соотношение:

```

print( math.floor((1 - clusterSize[label] /
                  (clusterSize[0]+clusterSize[1]))*100 ) )

```

Конечно, предложенный метод для «промышленного» решения малопригоден — он медленный, не может учесть наличие посторонних объектов и т. д. Но для быстрого решения поставленной узкой задачи вполне подходил.

Как вы видели в формуле оценки ответа, для оценки результат сравнивался с ручным подсчетом, выполненным с помощью программы обработки графической информации.

## 4. Заключение

Проведенная первая всероссийская олимпиада по искусственному интеллекту для VIII—XI классов показала повышенный интерес учащихся и педагогов к вопросам искусственного интеллекта, а также высокий

потенциал языка программирования Python для использования в качестве основного языка для олимпиады по искусственному интеллекту.

## Список источников

1. Богданова А. Н. Элективный курс «Основы искусственного интеллекта» для учащихся старших классов // Информатика в школе. 2021. № 7. С. 27–33.
2. Брусард М. Искусственный интеллект: пределы возможного / пер. с англ. М.: Альпина нон-фикшн, 2020. 362 с.
3. Калинин И. А., Самылкина Н. Н. Информатика. Углубленный уровень. 11 класс. М.: БИНОМ. Лаборатория знаний, 2013. 216 с.
4. Пратик Дж. Искусственный интеллект с примерами на Python / пер. с англ. СПб.: Диалектика, 2019. 448 с.
5. Программа «Цифровая экономика Российской Федерации». Утв. распоряжением Правительства Российской Федерации от 28.07.2017 г. № 1632-р. <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf>
6. Самылкина Н. Н., Салахова А. А. Обучение основам искусственного интеллекта и анализа данных в курсе информатики на уровне среднего общего образования: монография. М.: МПГУ, 2022. 228 с.
7. Стоянов С., Глушкова Т., Тодоров Й. Искусствен интеллект. Решаване на проблеми посредством търсене. София: Изкуства, 2019. 120 с.
8. Фридл Дж. Регулярные выражения. 3-е изд. / пер. с англ. СПб.: Символ-Плюс, 2008. 608 с.
9. Ясницкий Л. Н. Введение в искусственный интеллект: учебное пособие для вузов. 2-е изд., испр. М.: Академия, 2008. 176 с.
10. Ясницкий Л. Н. Искусственный интеллект. Элективный курс: учебное пособие. М.: БИНОМ. Лаборатория знаний, 2012. 197 с.