Информатика

Грацианова Татьяна Юрьевна
Кандидат физико-математических наук, преподаватель подготовительных курсов факультета Вычислительной математики и кибернетики МГУ им. М.В. Ломоносова

Что может статистика?

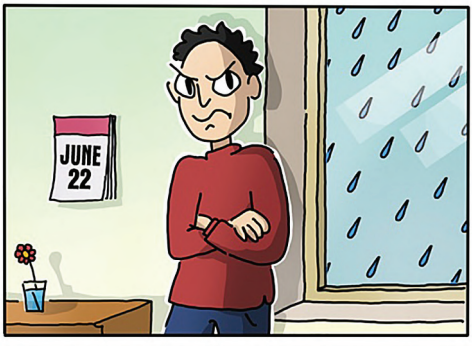
Статистические методы компьютерной лингвистики

Часто мы слышим об очередных достижениях искусственного интеллекта. А как он это делает? За счет чего машина решает сложные задачи часто лучше и быстрее человека? Оказывается, иногда искусственный интеллект для решения задач использует совсем «нечеловеческие» методы. В этой статье рассматривается применение методов математической статистики для решения задач компьютерной лингвистики. Вы найдете несколько программ, с помощью которых сможете провести собственные простые эксперименты. А в следующем номере будет продолжение — с более серьезными программами и гораздо более сложными задачами.

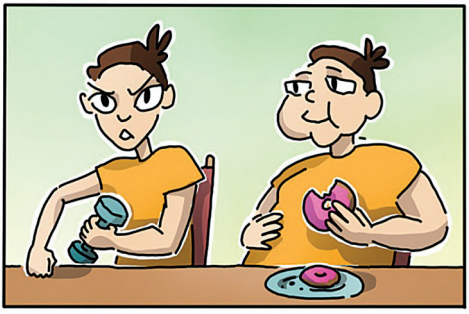
Сначала вспомним, что такое статистика. В википедии можно прочитать, что это наука, занимающаяся сбором, измерением и анализом данных, изучением явлений в числовой форме. Название появилось в 18 веке, а сама наука — много раньше, человечество что-то подсчитывает все время на протяжении своей истории. Сколько у нас в крепости живет народу, сколько поймали оленей — не останется ли население голодным? Сколько выросло зерна, какую часть урожая надо оставить на семена, какую можно продать, а какую съесть?

Правда, определение в википедии какое-то не очень точное: если мы будем просто подсчитывать свой бюджет, смотреть, сколько в карманах денег, хватит ли их на мороженое — это все-таки не статистика, а арифметика. Статистика имеет дело с большими объемами данных, пытается найти некие закономерности — определить, как разные данные связаны друг с другом. При этом часто используется понятие «вероятность» — количественная оценка возможности наступления некоторого события, поэтому статистика тесно связана с наукой под названием «теория вероятностей».

Основы этих наук изучаются в школе, но часто кажутся ученикам ненужными, слишком теоретическими, далекими от реальности, от повседневной жизни. Такое мнение иногда возникает потому, что статистика и теория вероятностей имеют дело с большими объемами данных, именно для них выводятся все закономерности, мы же в реальной жизни оперируем только теми данными, которые видим в настоящий момент. Например, посмотрев, что по статистике в ближайшие выходные будет жарко и вероятность дождя небольшая, человек размечтался, как он будет купаться в речке, и был очень обижен, когда в субботу пошел дождь, и на улицу можно было выйти только в куртке.



На самом деле науки эти в современном мире очень важны, статистические методы исследований, методы теории вероятностей проникли практически во все области жизни, они помогают установить связи между явлениями, распознать причины и следствия многих событий. Например, врачи говорят, что по утрам полезно делать зарядку, а есть много пирожных вредно, объясняют, какие процессы



при этом происходят в организме, исследует их биохимическую сущность. А статистика может провести исследование: взять группу людей, которые делают зарядку, и похожую группу людей, которые вместо этого едят пирожные, посчитать, сколько дней в году в среднем человек из каждой группы болеет. Так, совершенно не зная медицины, можно доказывать (или опровергать) медицинские утверждения.

С помощью статистических исследований можно найти закономерности, и уже потом попытаться объяснить их. Например, исследуя покупки в супермаркетах, выясняем, что они зависят от прогноза погоды: если ожидаются солнечные выходные, то перед ними закупки определенных продуктов больше. Потом уже аналитики подумают и скажут: конечно, люди на выходные отдыхать едут, шашлыки готовить. А пока менеджер, не вдаваясь в причины, делает заказ продуктов в зависимости от прогноза. Таким образом получается, что благодаря анализу числовых данных, статистическим исследованиям можно решать сложные и с первого взгляда совершенно «нематематические», «невычислительные», творческие задачи, задачи искусственного интеллекта.

Считаем буквы

Давайте посмотрим, какие задачи можем решить с помощью статистики мы, используя имеющиеся школьные знания. Поскольку нам сложно и даже невозможно производить какие-то большие вычислительные исследования в области экономики, биологии и во многих других областях, займемся лингвистикой. Русский язык (и еще какой-нибудь) в школе проходим, тексты всегда под рукой (ведь для статистики надо много материала!).

Сначала научимся считать буквы.

Задача. *На входе текст (кодировка известна). Подсчитать, сколько раз в него входит каждый символ.*

Это общая формулировка задачи. Давайте для начала решим ее в частном случае: подсчитаем количество вхождений для каждой из маленьких латинских букв. Алгоритм от этого не изменится, но это поможет нам для начала сосредоточиться именно на подсчете, не задумываясь о кодировке, да и результат будет удобнее просматривать. Пусть текст находится в файле text.txt. На Паскале можно написать следующую программу: напишем функцию, которая умеет посчитать, сколько раз заданная буква входит в заданный файл, а потом применим эту функцию нужное количество раз, задавая подряд все буквы по алфавиту

```
Type FChar = File of char;  {тип данных для файла с текстом}
Function KVO(var F:Fchar; x:char):Integer;
{Функция подсчитывает количество вхождений буквы x в файл F}
Var K:Integer; {здесь будем вести подсчет}
    a:char; {здесь будем хранить исследуемую букву}
Begin
  K:=0; {вначале обнулим счетчик}
  Reset(F); {откроем файл для чтения}
  While not EOF(F) do {пока файл не закончился}
  Begin Read(F, a); {прочитаем из него очередной символ}
        If a=x Then K:=K+1 {если это заданная буква, увеличим
счетчик}
  End;
  KVO:=K {Найденное значение и есть ответ}
End;
{Основная программа}
Var F : FChar; {файл с текстом}
    c : char; {счетчик цикла, здесь удобнее сделать его не цело-
численный, а «буквенный»}
```

```
Begin
  Assign(F, 'text.txt'); {ассоциируем файл с файловой переменной}
  For c:='a' to 'z' do {в цикле для всех нужных нам букв}
     Writeln(c, ' ', KVO(F, c)) {в каждой строчке печатаем очередную букву и подсчитанное количество ее вхождений в текст}
End.
```

В результате будет напечатано 26 строк с буквами и числами. Здесь мы пользуемся свойством кодировки латинских букв — они расположены по алфавиту. Для русских букв это верно не для всех кодировок. Если захочется подсчитать не только буквы, а все символы, надо иметь в виду, что некоторые символы (с небольшими кодами) являются «непечатными», не отображаются на экране (это, например, символ перевода строки, символ, соответствующий клавише Ecs). В такой программе есть смысл для переменной цикла использовать не символы, а их коды.

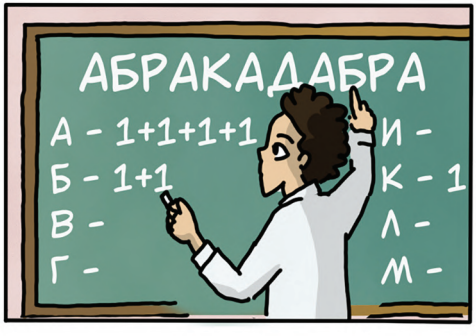
Программа получилась компактная, задачу решает, но всем ли она хороша? Сразу можно назвать целых 2 недостатка:

1. Программа просматривает файл много раз — столько, сколько у нас исследуемых символов. А ведь мы пишем программу для статистических исследований, мы начали с того, что для них данных должно быть много, то есть файл должен быть большой. Рациональнее, быстрее выполнить все подсчеты за один просмотр файла.

2. Программа печатает результаты, мы можем на них посмотреть и анализировать только «в уме». Для того, чтобы делать с результатами что-то еще, надо их сохранить, например, в массиве.

Напишем другую программу, которая будет лишена этих недостатков. Опишем массив для хранения результатов, назовем его Dop (дополнительный). В нем должно быть столько компонент, сколько символов мы хотим исследовать. Для латинских букв это 26. В Паскале индексами массивов могут быть не только числа, но и символы. В нашем случае удобно нумеровать компоненты массива буквами. В компоненте под номером 'a' будет храниться количество букв 'a', в следующей компоненте под номером 'b' будет храниться количество букв 'b' — и так далее.

Сначала массив надо обнулить, а потом в цикле (пока не закончится файл) считывать из файла очередной символ и обрабатывать. В чем будет заключаться обработка? Если символ является латинской буквой, то надо на единичку увеличить соответствующую компоненту массива Dop. Как это сделать, как вычислить нужную компоненту? Оказывается, при нашей нумерации элементов массива ничего вычислять не надо. Если встретилась буква 'a', то надо



увеличить компоненту с индексом 'a', если буква 'b' — работаем с компонентой с индексом 'b' и так далее. Таким образом, если букву из файла мы считали в переменную a (не путаем с буквой 'a'), то увеличивать надо компоненту с индексом [a].

Обратим внимание на необходимость проверки, является ли исследуемый символ маленькой латинской буквой. Если этого не делать, то мы заставим программу работать с компонентой массива с несуществующим индексом. Если Паскаль настроен на обнаружение такой ошибки, программа просто не будет работать; если же не настроен, ошибка может вызвать различные последствия (иногда совсем незаметные, а иногда весьма серьезные), так как несанкционированно изменится содержимое какой-то ячейки памяти.

```
Const N=26;{количество исследуемых символов,размерность массива}
Var   Dop   : Array['a'..'z'] of Integer; {Массив для хранения
результатов}
      F : File of char;
      a  : Char;
Begin
  Assign(F, 'text.txt');
  Reset(F); {открываем файл для чтения}
  For a:='a' to 'z' do  {обнуляем все компоненты массива результатов}
       Dop[a]:=0;
  While not EOF(F) do {Обрабатываем символы пока они не закончатся}
  Begin Read(F, a);{В переменную a считываем очередной символ}
        if (a>='a') and (a<='z')
        {символ является маленькой латинской буквой?}
                              Then Dop[a]:=Dop[a]+1;
        {увеличиваем нужную компоненту на 1}
  End;
  For a:='a' to 'z' do   {Печать результатов}
    Writeln(a, ' ', Dop[a])
End.
```

Результат работы этой программы выглядит точно так же, как и результат предыдущей, только работает она существенно быстрее (правда, при современных скоростях заметить это можно только при обработке очень большого файла) и сохраняет результаты — их можно обрабатывать дальше.

Напишем теперь то же самое на Питоне. Алгоритм будет отличаться только тем, что при обработке файла удобно использовать двойной цикл: файл в питоне состоит из строк, так что надо считывать строку, а потом ее обрабатывать. Программа получается чуть короче за счет того, что в Питоне не нужны описания, и присваивание начальных значений элементам массива очень компактное. Немного сложнее работа с индексами массива, так как нумерация не буквами, как в Паскале, а числами от 0 до N-1

```
N=26 # для английских букв - 26
dop=[0]*N # здесь будет подсчет, в начале обнулили
f1=open('text.txt','r', encoding="UTF-8") # открыли файл для чтения
for line in f1:  # считываем из файла строку
    for a in line: # обрабатываем символ в строке
        if ord('a')<=ord(a)<=ord('z'):  # если это маленькая латинская буква
            dop[ord(a)-ord('a')]+=1  # увеличиваем соответствующую компоненту массива
for i in range(N): # печатаем N компонент массива
    print(chr(ord('a')+i), '  ',dop[i])
f1.close()
```

Упорядочиваем результаты

Как же мы можем распорядиться полученной информацией? Давайте более детально рассмотрим, что мы сделали. На входе был набор символов, на выходе — массив чисел. Причем размер этого массива не зависит от размера начального набора символов, зависит только от количества символов, которые участвуют в наших подсчетах. Таким образом мы, во-первых, вытащили из текста интересующие нас символы (в данном случае маленькие латинские буквы), а во-вторых, сжали первоначальный (возможно, огромный) текст в массив из N элементов. Вместо большого текста небольшой массив — это значительная экономия памяти, и понятно, что при этом некоторые свойства текста мы потеряли. Скажем, восстановить обратно текст по массиву не получится (даже если в нем были только латинские буквы). Мы потеряли порядок, в котором эти буквы стояли, а вот все данные о количестве букв не только сохранили, но еще и оформили их удобным образом. Поэтому теперь можем отвечать на любые вопросы по тексту, касающиеся количества букв в нем.

Сформулируем некоторые из таких вопросов

1. Сколько всего разных букв в тексте?
2. Какая буква самая частая, какая самая редкая?
3. Каких букв в тексте нет?
4. Какова частота появления каждой буквы?

Прежде чем думать, как ответить на эти вопросы, проясним термин «частота». Пишется через «а», потому что от слова «часто», а не «чисто». Чтобы определить частоту буквы, надо посчитать, сколько раз буква встречается в тексте, и разделить это число на общее количество букв в тексте. Это понятие дает нам возможность сравнивать разные тексты. Понятно, что в маленьком тексте значения чисел, показывающие, сколько раз буква входит в текст, будут меньше, чем в большом. А вот частота от длины текста уже не зависит.

Почти на все вопросы ответит программа, которая напечатает буквы в порядке убывания частоты встречаемости. Для этого нам надо отсортировать полученный массив

(в котором мы храним количество букв) и полученные числа разделить на общее количество букв в тексте. Однако здесь нас подстерегает небольшая неприятность: в массиве хранятся числа, отсортировав его, мы получим числа, описывающие, сколько раз встречаются в тексте буквы. Но информация о том, какие это буквы, потеряется, ведь она у нас «хранится» в индексах массива, а мы их при сортировке поменяем. Придется создать какую-нибудь двумерную структуру, в которой будем хранить пары (буква — количество) и в зависимости от числа, обозначающего количество, сортировать будем целиком пары.

На Паскале в качестве такой структуры можно использовать запись (тип Record) с двумя полями: буква и количество. Массив таких записей нам придется создать в программе: в поле «буква» записать последовательно все латинские буквы (у нас это поле называется l от слова letter), а в поле «количество» (назовем его d) — подсчитанные значения, которые хранятся в массиве DOP.

Получившиеся записи в зависимости от значения поля d отсортируем по убыванию. Будем использовать метод пузырька. Получившийся массив записей распечатаем, не забыв вместо количества вхождений буквы печатать частоту (то есть делить на количество букв).

Представим получившуюся программу на Паскале (массив Dop теперь распечатывать не будем):

```
Const N=26;{количество исследуемых символов,размерность массива}
Var   Dop   : Array['a'..'z'] of Integer;
      F : File of char;
      a : Char;
      Kvo, k, i, j: Integer;
      Buk : Array[1..N] of Record l:char;
                                  d:integer
                           End;

Begin
  Assign(F, 'text.txt');
  Reset(F);
  For a:='a' to 'z' do
       Dop[a]:=0;
  While not EOF(F) do
  Begin Read(F, a);
        if (a>='a') and (a<='z')
                           Then Dop[a]:=Dop[a]+1;
  End;
    {Количество маленьких латинских букв в тексте}
  Kvo:=0;
 For a:='a' to 'z' do
     Kvo:=Kvo+Dop[a];
 Writeln ('Всего в тексте маленьких латинских букв', Kvo);
 a:='a';
```

```
 For k:=1 to N do {сформируем массив записей}
 Begin   Buk[k].l:= a; {в это поле ставим буквы}
         Buk[k].d:=Dop[a]; {в это поле количества,подсчитаные в
массиве dop)}
         a:=succ(a) {переходим к следующей по алфавиту букве}
 End;
 For i:=1 to N-1 do  {сортировка методом пузырька по убыванию}
   For j:=1 to N-i do
     if Buk[j+1].d>Buk[j].d Then {Если порядок неверный}
                        SWAP (Buk[j],Buk[j+1]);  {меняем местами}
 For i:=1 to N do  {распечатываем результат}
   Writeln(Buk[i].l, '  ', Buk[i].d/Kvo:8:4)
End.
```

Заметим, что, если изначально поставить перед собой такую задачу, можно сразу подсчет вести в поле записи, не использовать дополнительный массив. Именно так сделаем на Питоне. Для хранения результатов подсчетов будем использовать словарь, в котором ключом будет буква, а содержимым — количество букв. Если мы хотим хранить в словаре только те буквы, которые есть в тексте, можно формировать словарь в процессе обработки файла (если буква уже есть, добавлять единичку к количеству вхождений, если нет — добавлять запись в словарь). Мы же, чтобы программа была аналогична паскалевской, будем хранить в словаре все 26 букв (если буква не встречается, значение будет 0), поэтому сначала сформируем словарь, запишем в него все ключи и нули.

Для сортировки воспользуемся встроенным методом sorted. В Паскале тоже есть библиотеки, в которых есть всякие нужные функции, в том числе и сортировка, но там мы писали сортировку сами, это не сложно. В Питоне же в данном случае это сложнее, так как словарь отсортировать нельзя (это неупорядоченная структура по определению), его надо сначала превратить в какую-то другую структуру, которую можно упорядочить. С помощью метода items превратим его в список кортежей и этот список отсортируем. При сортировке кортежа надо указать, по какому полю производится сортировку. У нас это второе поле, ставим единичку (так как нумерация с нуля). Получаем новую отсортированную структуру — список кортежей (в программе она называется sorted_d), которую и распечатываем.

Сразу скажем, что программа на Питоне будет работать медленнее (в том числе из-за свойств Питона) и требует больше памяти.

Обратите внимание, и в Паскале, и на Питоне для печати вещественных чисел мы используем форматный вывод (чтобы выводились не все цифры после точки, а только 5).

Отладим наши программы на небольшом файле, чтобы было легко подсчитать результат вручную и проверить. Файл должен быть в той же папке, что и программа, или в дочерней (если система не настроена изначально как-то иначе). Естественно, на одинаковых файлах результат должен быть одинаковый.

Если все получилось, надо немножко подправить нашу программу. Мы в ней считаем только маленькие латинские буквы, но в реальном тексте большие буквы обычно тоже встречаются (например, в начале предложений) и должны участвовать в наших подсчетах. Для этого нужно «поймать» большую букву(аналогично тому, как мы исследуем, является ли символ маленькой латинской буквой) и превратить ее в маленькую. Для этого можно воспользоваться как стандартной функцией, так и тем фактом, что в любой кодировке расстояние между большой и соответствующей маленькой буквой одинаковое и равно ord('A') — ord('a').

Если хочется написать аналогичную программу для работы с русскими буквами, надо заменить значение N и латинские буквы на русские ('а' И ''z' на 'а' и 'я'). Программа будет правильно работать, если в используемой кодировке русские буквы стоят подряд.

Проводим статистические исследования

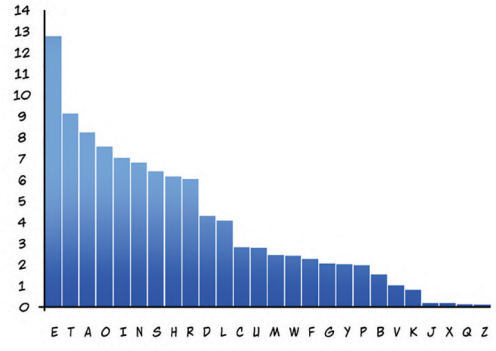
Все исследования надо проводить с большим количеством данных. Именно поэтому мы для ввода используем файл, а не строку. В файле должен быть текст, в котором несколько тысяч символов.

Обработайте с помощью своей программы несколько файлов. Посмотрите на результаты.

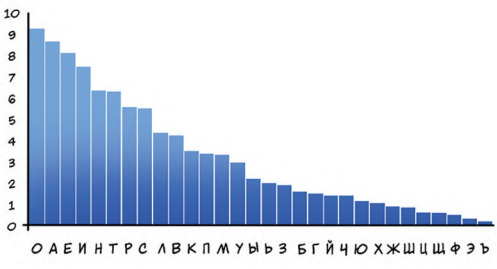
Что же мы можем исследовать? Во-первых, сразу видно, что буквы как в английском, так и в русском языке в текстах используются в разной частотой. Причем различия очень существенные — между самой частой и самой редкой буквой больше, чем в 100 раз! Сверьте полученные Вами значения для самых частых букв со значениями, полученными учеными (таблицу можно посмотреть, например, в википедии, статья «Английский алфавит», раздел «Частота букв» и статья «Частотность», раздел «Частотность букв русского языка»). В разных статьях могут быть немного разные значения для чисел — частот букв (это зависит от того, на каких материалах производился подсчет), но порядок букв будет одинаковый.

Наглядно частоты букв в текстах можно представить следующим образом (по материалам из открытых источников) — отображается частота встречаемости букв в процентах:

— для английского языка



— для русского языка



Как видим, в английском языке самая частая буква — e, а самая редкая — z. В русском самая частая — о, самая редкая – твердый знак. Существуют мнемонические «словосочетания» для запоминания десятка самых частых букв. Для русского языка это придуманное слово СЕНОВАЛИТР, а для английского TETRISHONDA. В этих «словах» буквы расположены не в порядке возрастания или убывания их встречаемости, просто взяты самые частые 10 букв и из них составлено нечто удобно запоминаемое. Результаты работы Вашей программы должны быть похожими для этих 10 букв. Для редких букв результаты могут сильно отличаться (при этом буквы остаются редкими, просто могут быть в другом порядке, с другой частотой).

Получилось похоже? Это ожидаемый результат. Если получилось не так (совсем другие значения, самыми частыми буквами являются другие) причин может быть три:

• ошибка в программе. Возвращаемся к отладке, проверим работу программу на маленьком файле, для которого можно все посчитать вручную

• маленький файл. Статистические закономерности верны для больших данных. Например, если текст в файле будет состоять из одного предложения «Подъезд к объекту», мы получим, что одним из самых частых символов является твердый знак, а для больших текстов это самый редкий символ. В этом случае добавьте в файл еще какой-то текст.

• специфический текст. Статистика на то и статистика, что ее закономерности верны для «среднестатистических», «обычных» текстов.

Вот о последней причине несоответствия показателей давайте поговорим отдельно. Дело в том, что статистические исследования на частоту букв как раз и можно использовать для проверки текста на «обычность». Когда это может понадобиться? Например, в результате ошибок кодировки, передачи текст сильно изменился — с помощью описанных статистических методов такие изменения можно быстро заметить. Иногда вместо файла с реальным текстом предлагается файл с набором букв (так могут делать нехорошие люди для нехороших целей). Специальные программы отлавливают такие файлы. По несоответствию «средней» статистике можно также обнаружить направленность текста. Например, в научных текстах буква «ф» встречается чаще, чем обычно (за счет слов «коэффициент», «функция», «диффузия»), а в поэтических, лирических текстах может часто встречаться буква «й» за счет большего употребления прилагательных.

Попробуйте подать на вход программе, вычисляющей статистику для английских букв, файл, содержащий не текст на английском языке, а программный код. Несмотря на то, что программный код содержит английские слова, результаты будут далеки от среднестатистических. Таким образом, мы умеем автоматически отличить текст на английском языке от текста на языке программирования.

Статистический анализ букв используется при дешифровке текстов. Если все буквы в тексте заменить на какие-то значки (или числа), причем разным буквам в соответствие поставить разные значки, а одинаковым — одинаковые, то начать расшифровку такого текста можно с поиска самых часто встречающихся значков. А да-

лее, подставив на их место соответствующие буквы, уже использовать правила языка.

И наоборот, когда создавались шифры, использовалась статистика. Вспомним неравномерное кодирование — когда буквы кодируются несколькими значками, причем для разных букв этих значков может быть разное количество. Чтобы код получился покороче, надо частые буквы кодировать меньшим количеством значков, а редкие — большим. Именно так создана азбука Морзе. В ней буквы E и T кодируются всего одним значком (точка и тире), буква A — двумя значками (точка тире), а вот буква Z — четырьмя.

Статистические данные об употреблении букв используются при создании раскладки клавиатуры. Раскладка, которой мы пользуемся, появилась давно, еще до эпохи компьютеров. Клавиатуру делали для механических пишущих машинок для печати десятью пальцами. Частые буквы на ней располагаются посередине (под сильные указательные пальцы), редкие буквы — с краев под мизинцы.

Помогает статистика и при распознавании языка. Например, у многих европейских языков похожий набор букв, но частота встречаемости разная, так что для текста, написанного латинскими буквами, можно делать предположения о языке, на котором он написан.

Задание. *Мы с Вами определили, что можем автоматически отличить текст на английском языке от текста программы. Проверьте, можно ли с помощью описанных здесь статистических методов определить, на каком языке программирования написана программа. Для этого придется подсчитывать не только латинские буквы, но и всякие значки, используемые в программах. А потом надо взять много программ на разных языках и посмотреть, различаются ли для них частоты употреблений букв и символов.*

Конечно, с помощью описанной методики можно получить не очень много данных об исследуемом тексте. Но мы ведь и подсчетов не так много сделали. В следующем номере — что еще можно посчитать и что на основе этого определить.

Юмор Юмор Юмор Юмор

Я говорю: «Я программист».

Другие слышат: «Я могу решить любую вашу техническую проблему с любой железякой, которая существует во Вселенной».

На самом деле я имею в виду: «Иногда я говорю компьютеру, что ему нужно делать, и расстраиваюсь, когда это не срабатывает».