

# Проект опечаточника для школьных работ (с выставлением оценки)

Серова Александра,  
Группа 194,  
03.06.20

# Что сделано?

Программа, которая:

- Принимает на вход текстовый файл с опечатками
- Для каждого слова в файле находит множество возможных исправлений с расстоянием Левенштейна 0 (нет ошибки), 1 (нужно заменить/выкинуть/добавить 1 букву/сделать 1 перестановку букв), 2
- На основании частотности каждого из возможных вариантов исправления предлагает наиболее вероятную замену для слова (при этом слова с расстоянием 0 имеют преимущество перед словами с расстоянием 1 от исходного и т.д.)

# Что сделано?

- Все предложенные программой замены (кроме слов с расстоянием в 0) записываются в отдельный файл, в него же записывается оценка:

0-39% от исходного текста верно – «2»

40-59% - «3»

60-79% - «4»

80-100% - «5»

Ссылка на репозиторий:

<https://github.com/SerovaAlexandra/Project-spellchecker>

Часть материала для программы взята из

<https://norvig.com/spell-correct.html>

# Расстояние Левенштейна

- Расстояние Левенштейна =1, если, чтобы получить правильное слово, к слову с опечаткой нужно применить одну из операций:
  - Удаление (лишней буквы) *котеноЕк* > *котенок*
  - Замена (1 буквы на другую) *котенИк* > *котенок*
  - Вставка (недостающей буквы) *ктенок* > *котенок*
  - Транспозиция (поменять 2 буквы местами) *кТОенок* > *котенок*

# Необходимые данные

- Текстовый файл, содержащий список слов с их частотностями (в формате «слово(пробел)число вхождений\n»)
- Текстовый файл с опечатками, который будет проверяться

# Как работает?

- Функция **words()** работает с текстом из файла, полученного на входе: с помощью регулярного выражения находит все слова в нем, в которых больше 1 буквы (однобуквенные служебные части речи не учитываются), независимо от регистра

```
1 import re
2
3 def words(text):...
4     ...return re.findall('[a-я\-]{2,}', text.lower())
5
```

# Как работает?

- Функция **words\_counter()** работает со списком частотностей: преобразует его в словарь с парами <слово><его частотность>, переменная RUSWORDS как раз представляет собой этот словарь
- Переменная **russian\_letters** – кириллический алфавит+дефис (на основе этого будет высчитываться расстояние Левенштейна в функции **edits1()**)

```
5
6 def words_counter(filename):
7     with open(filename) as f:
8         text=f.read()
9         my_dict={}
10        text=text.splitlines()
11        for line in text:
12            talken, number=line.split(' ')
13            number=int(number)
14            my_dict[talken]=number
15        return my_dict
16 RUSWORDS=words_counter('a.txt')
17 russian_letters='абвгдеёжзийклмнопрстуфхцчщъыьэя-'
```

# Как работает?

- **prob()** рассчитывает вероятность того, что нам нужно именно это слово, используется в функции **correct()** для расчета наиболее возможного варианта исправления для слова с опечаткой

```
18
19 def prob(word, N=sum(RUSWORDS.values())):
20     if word in RUSWORDS:
21         return RUSWORDS[word] / N
22
23
24
```



# Как работает?

- **edits1()** возвращает множество слов с расстоянием Левенштейна 1 для слова с опечаткой: все слова, которые могут получиться в результате удаления, или транспозиции, или замены буквы, или вставки буквы
- **edits2()** так же считает расстояние 2 (проделывает ту же операцию с результатами **edits1()**, что сама **edits1()** делает с исходным словом)

```
23
24
25 def edits1(word):
26     b=len(word)
27     deletes=[word[0:i]+word[i+1:] for i in range(b)] #операция удаления (в слове с опечаткой -- 1 лишняя буква)
28     transposes=[word[0:i]+word[i+1]+word[i]+word[i+2:] for i in range(b-1)] #транспозиция (две буквы нужно поменять местами)
29     replaces=[word[0:i]+c+word[i+1:] for i in range(b) for c in russian_letters] #замена неправильной буквы
30     inserts=[word[0:i]+c+word[i:] for i in range(b+1) for c in russian_letters] #вставка недостающей буквы
31     return set(deletes+transposes+replaces+inserts)
32
33
34 def edits2(word):
35     return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in RUSWORDS)
36
```

# Как работает?

- **correct()** выбирает наиболее возможный вариант исправления из словаря RUSWORDS: слово с наибольшей частотой встречаемости и наименьшим расстоянием Левенштейна относительно слова с опечаткой

```
36
37
38 def known(words):
39     return set(w for w in words if w in RUSWORDS)
40
41 def candidates(my_word):
42     return (known([my_word]) or known(edits1(my_word)) or edits2(my_word) or [my_word])
43
44 def correct(word):
45     return max(candidates(word), key=prob)
46
```

# Как работает?

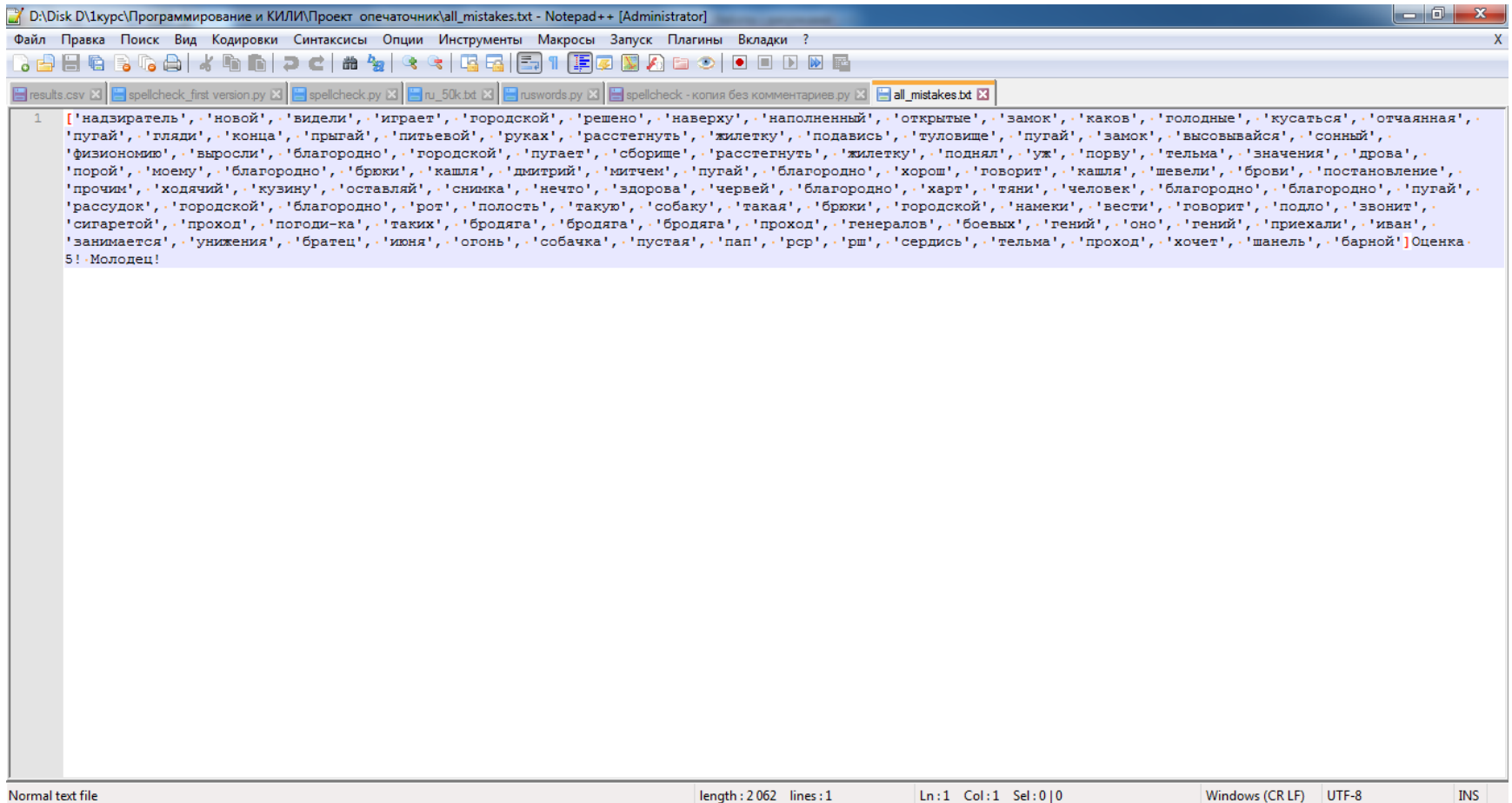
- Запуск – через функцию **main()**, которая работает непосредственно с файлом с опечатками

```

46 .....
47 def main(filename):
48     with open(filename) as f:
49         my_text=f.read()
50         list_of_all_words=words(my_text) #нахождение в тексте всех слов (функция words())
51         corrected_words=[]
52         for elem in list_of_all_words:
53             corrected_word=correct(elem) #нахождение наиболее подходящего исправления для каждого слова
54             if corrected_word!=elem: #"отсеиваем" все слова с расстоянием Левенштейна 0 (они не будут считаться в "ошибках" в переменной 'mistakes_numb')
55                 corrected_words.append(corrected_word)
56             mistakes_numb=len(corrected_words) #подсчет количества слов с ошибками
57             perc=mistakes_numb/len(list_of_all_words)*100 #какой процент занимают слова с опечатками от всего текста?
58         with open('all_mistakes.txt','w',encoding='utf-8') as myf: #в новый файл записываем все исправленные слова, оценку на основе данных переменной 'perc'
59             myf.write(str(corrected_words))
60             if perc<=20:
61                 myf.write('Оценка 5! Молодец!')
62             elif perc>20 and perc<=40:
63                 myf.write('Оценка 4')
64             elif perc>40 and perc<=60:
65                 myf.write('Оценка 3')
66             else:
67                 myf.write('Оценка 2! Надо переписать!')
68

```

# Результат



```
1 ['надзиратель', 'новой', 'видели', 'играет', 'городской', 'решено', 'наверху', 'наполненный', 'открытые', 'замок', 'каков', 'голодные', 'кусаться', 'отчаянная', 'пугай', 'гляди', 'конца', 'прыгай', 'питьевой', 'руках', 'расстегнуть', 'жилетку', 'подавись', 'туловище', 'пугай', 'замок', 'высовывайся', 'сонный', 'физиономию', 'выросли', 'благородно', 'городской', 'пугает', 'сборище', 'расстегнуть', 'жилетку', 'поднял', 'уж', 'порву', 'тельма', 'значения', 'дрова', 'порой', 'моему', 'благородно', 'брюки', 'кашля', 'дмитрий', 'митчем', 'пугай', 'благородно', 'хорош', 'говорит', 'кашля', 'шевели', 'брови', 'постановление', 'прочим', 'ходячий', 'кузину', 'оставляй', 'снимка', 'нечто', 'здоровая', 'червей', 'благородно', 'харт', 'тяни', 'человек', 'благородно', 'благородно', 'пугай', 'рассудок', 'городской', 'благородно', 'рот', 'полость', 'такую', 'собаку', 'такая', 'брюки', 'городской', 'намeki', 'вести', 'говорит', 'подло', 'звонит', 'сигаретой', 'проход', 'погоди-ка', 'таких', 'бродяга', 'бродяга', 'бродяга', 'проход', 'генералов', 'боевых', 'гений', 'оно', 'гений', 'приехали', 'иван', 'занимается', 'унижения', 'братец', 'июня', 'огонь', 'собачка', 'пустая', 'пап', 'рер', 'рш', 'сердись', 'тельма', 'проход', 'хочет', 'шанель', 'барной'] Оценка: 5! Молодец!
```

# Как можно доработать?

- Подключить контекстный анализ для повышения точности исправлений
- Объединять слова по типам операций, которые нужно совершить, чтобы их исправить, и писать в файле с оценкой, например «в этих словах (список слов) ты пропустил одну букву/добавил лишнюю и т.д.»
- Анализ частотных ошибок на основе нескольких текстов