Содержание

Введение

В современном мире, где объём информации растёт с каждым днём, качество текстовых данных становится критически важным. Ошибки, возникающие при вводе текста, такие как опечатки, могут существенно повлиять на восприятие информации и её дальнейшую обработку. Алгоритмы исправления опечаток играют ключевую роль в повышении точности и эффективности обработки текстов, будь то в системах автоматического перевода, поисковых системах или текстовых редакторах.

В настоящее время известны различные способы решения проблемы опечаток. Приведём лишь некоторые из них.

- Словарный подход: использование предварительно заданного списка слов (словаря) для проверки корректности написания. Если слово не найдено в словаре, оно помечается как ошибочное.

- Коррекция на основе вероятностей: оценка вероятности правильного написания слова на основе его частоты в языке и контекста.

- Нейронные сети: применение рекуррентных нейронных сетей (RNN) или трансформеров для анализа последовательностей текста и предсказания корректных вариантов.

Эти и другие методы уже применяются в существующих программах. Например, текстовые редакторы вроде Microsoft Word и Google Docs имеют встроенные инструменты проверки правописания и автоматически предлагают исправления. Похожими возможностями обладают также и среды разработки вроде Microsoft Visual Studio. Поисковые системы, например, Яндекс или Google используют алгоритмы исправления опечаток для улучшения результатов поиска.

Целью настоящей работы является построение в математической форме алгоритма исправления опечаток, основанного на вычислении численных оценок попарного сходства слов, а также заключение о пригодности такого подхода для решения проблемы опечаток. Достижение этой цели обеспечивается выполнением следующих задач:

1. разработка необходимого математического аппарата;

2. разработка программной реализации на языке программирования Python;

3. проведение вычислительного эксперимента и анализ его результатов.

Список используемых обозначений

$\mathbf{E}$	—	пустой кортеж (пустое слово)
$\lvert w \rvert$	—	длина кортежа (слова) w; таким образом, $\lvert \mathbf{E} \rvert = 0$
$\mathbb{A} := \{a_1, \ldots, a_{N_{\mathbb{A}}}\}$	—	алфавит
$\mathbb{A}^* := \bigcup\limits_{n=0}^{+\infty} \mathbb{A}^n$	—	множество всех слов в алфавите $\mathbb{A}$ (универсальный язык над алфавитом $\mathbb{A}$)
$\mathbb{A}^+ := \bigcup\limits_{n=1}^{+\infty} \mathbb{A}^n$	—	множество всех непустых слов в алфавите $\mathbb{A}$; таким образом, $\mathbb{A}^* = \mathbb{A}^+ \cup \{\mathbf{E}\}$
$\mathbb{V}$	—	словарь корректных слов
$\mathbb{N}_n := \{1, \ldots, n\}$	—	множество натуральных чисел не больше $n \in \mathbb{N}$
2^A	—	множество всех подмножеств множества A (булеан A)
$\operatorname{dom} f$	—	область определения функции f
$\operatorname{ran} f := f(\operatorname{dom} f)$	—	множество значений функции f
$A \wedge B$	—	конъюнкция высказываний A и B
$f \circ g$	—	композиция функций f и g

1. Математическая модель слов

Целью работы является разработка алгоритма исправления опечаток в словах из некоторого языка над алфавитом

$$\mathbb{A} := \{a_1, \ldots, a_{N_{\mathbb{A}}}\},$$

представленных в некотором заданном множестве $\mathbb{V} \subseteq \mathbb{A}^*$, где

$$\mathbb{A}^* := \bigcup_{n=0}^{+\infty} \mathbb{A}^n$$

— универсальный язык. Напомним, что слово в алфавите $\mathbb{A}$ в смысле теории формальных языков — это кортеж

$$w := (w_1, \ldots, w_n), \quad \text{где} \quad w_i \in \mathbb{A}, \quad i \in \{1, \ldots, n\},$$

а длина слова w — это неотрицательное целое число $|w| := n$ при

$$w := (w_1, \ldots, w_n).$$

Множество $\mathbb{V}$ будем называть словарём. Все слова, представленные в словаре, будем называть корректными, а все прочие, т. е. слова из $\mathbb{A}^* \setminus \mathbb{V}$, — искажёнными.

Конкретные буквы будем записывать моноширинным шрифтом, конкретные слова — без использования скобок, запятых и иных дополнительных символов. Например, `hello` — слово из английского языка над алфавитом

$$\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m} \\ \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$$

(заглавными буквами пренебрегаем).

2. Функции искажения

Введём функции, которые будем называть функциями однократного искажения слов:

- функция добавления

$$A_k^a(w) := \begin{cases} (w_1, \ldots, w_{k-1}, a, w_k, w_{k+1}, \ldots, w_n), & k \in \mathbb{N}_{|w|}, \\ (w_1, \ldots, w_n, a), & k = |w| + 1, \end{cases}$$

  где $k \in \mathbb{N}_{|w|+1}$, $a \in \mathbb{A}$, причём

$$\operatorname{dom} A_k^a = \bigcup_{n=k-1}^{+\infty} \mathbb{A}^n, \quad \operatorname{ran} A_k^a = \bigcup_{n=k}^{+\infty} \mathbb{A}^n, \quad |A_k^a(w)| = |w| + 1,$$

$$\forall\, a \in \mathbb{A} \;\; A_1^a(\mathbf{E}) = a;$$

- функция удаления

$$D_k(w) := (w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_n),$$

  где $k \in \mathbb{N}_{|w|}$, причём

$$\operatorname{dom} D_k = \bigcup_{n=k}^{+\infty} \mathbb{A}^n, \quad \operatorname{ran} D_k = \bigcup_{n=k-1}^{+\infty} \mathbb{A}^n, \quad |D_k(w)| = |w| - 1,$$

$$\forall\, a \in \mathbb{A} \;\; D_1(a) = \mathbf{E};$$

- функция замены

$$R_k^a(w) := (w_1, \ldots, w_{k-1}, a, w_{k+1}, \ldots, w_n),$$

  где $k \in \mathbb{N}_{|w|}$, $a \in \mathbb{A}$, причём

$$\operatorname{dom} R_k^a = \operatorname{ran} R_k^a = \bigcup_{n=k}^{+\infty} \mathbb{A}^n, \quad |R_k^a(w)| = |w|,$$

$$\forall\, a, b \in \mathbb{A} \;\; R_1^a(b) = a;$$

- функция перестановки

$$S_{ij}(u) := (v_1, \ldots, v_n),$$

  где $i, j \in \mathbb{N}_{|w|}$, $i \neq j$,

$$v_k := \begin{cases} u_j, & k = i, \\ u_i, & k = j, \\ u_k, & \text{иначе}, \end{cases}$$

  причём $\quad \operatorname{dom} S_{ij} = \operatorname{ran} S_{ij} = \displaystyle\bigcup_{n=\max\{i,j\}}^{+\infty} \mathbb{A}^n, \quad |S_{ij}(w)| = |w|,$

$$\forall\, a, b \in \mathbb{A} \;\; S_{12}(a, b) = S_{21}(a, b) = (b, a),$$

$$\forall\, i, j \in \mathbb{N},\, i \neq j \;\; \forall\, w \in \operatorname{dom} S_{ij} \;\; S_{ij}(w) = S_{ji}(w).$$

Введём функции, отображающие слова на множества искажённых слов, соответствующие функциям однократного искажения слов:

$$\begin{aligned}
&A(w) := \left\{A_k^a(w)\colon k \in \mathbb{N}_{|w|+1} \ \wedge \ a \in \mathbb{A}\right\}, && \operatorname{dom} A = \mathbb{A}^*, && \operatorname{ran} A \subset 2^{\mathbb{A}^+},\\
&D(w) := \left\{D_k(w)\colon k \in \mathbb{N}_{|w|}\right\}, && \operatorname{dom} D = \mathbb{A}^+, && \operatorname{ran} D \subset 2^{\mathbb{A}^*},\\
&R(w) := \left\{R_k^a(w)\colon k \in \mathbb{N}_{|w|} \ \wedge \ a \in \mathbb{A}\right\}, && \operatorname{dom} R = \mathbb{A}^+, && \operatorname{ran} R \subset 2^{\mathbb{A}^+},\\
&S(w) := \left\{S_{ij}(w)\colon i,\, j \in \mathbb{N}_{|w|} \ \wedge \ i \neq j\right\}, && \operatorname{dom} S = \mathbb{A}^+ \setminus \mathbb{A}, && \operatorname{ran} S \subset 2^{\mathbb{A}^+ \setminus \mathbb{A}}.
\end{aligned}$$

Введём также функцию, объединяющую все вышеперечисленные множества:

$$E(w) := A(w) \cup D(w) \cup R(w) \cup S(w),$$

где $w \in \mathbb{A}^+ \setminus \mathbb{A}$. Слова из $E(w)$ будем называть однократными искажениями слова w.

Наконец, введём случайные функции, которые будем называть функциями случайного однократного искажения слов. Для этого введём дискретное равномерное распределение $\mathcal{U}(S)$, где $S := \{s_1, \ldots, s_n\}$ — конечное множество, являющееся множеством значений случайной величины, и определим его следующим образом:

$\xi \sim \mathcal{U}(S)$	s_1	s_2	$\cdots$	s_n
$\mathbb{P}$	$1/n$	$1/n$	$\cdots$	$1/n$

Функции случайного однократного искажения слов введём как случайные величины, распределённые по законам, зависящим от слов:

$$\begin{aligned}
&\widehat{A}(w) \sim \mathcal{U}(A(w)) && \text{— функция случайного добавления,}\\
&\widehat{D}(w) \sim \mathcal{U}(D(w)) && \text{— функция случайного удаления,}\\
&\widehat{R}(w) \sim \mathcal{U}(R(w)) && \text{— функция случайной замены,}\\
&\widehat{S}(w) \sim \mathcal{U}(S(w)) && \text{— функция случайной перестановки.}
\end{aligned}$$

Представления искажений слов в виде случайных функций имеют преимущество: из таких функций можно свободно строить композиции, не требующие установки большинства ограничений, связанных с длинами исходных и результирующих слов, которые необходимо устанавливать при построении композиций функций A_k^a, D_k, R_k^a, S_{ij}. Например, для того, чтобы выражение

$$A_i^a\left(R_j^b(D_k(w))\right), \quad \text{где} \quad w \in \mathbb{A}^*, \quad a,\, b \in \mathbb{A}, \quad i,\, j,\, k \in \mathbb{N},$$

имело смысл, необходимо установить следующие дополнительные ограничения на i, j, k и w: $|w| \geq 2$ (т. е. $w \in \mathbb{A}^+ \setminus \mathbb{A}$), $k \leq |w|$, $j \leq |D_k(w)| = |w| - 1$, $i \leq \left|R_j^b(D_k(w))\right| + 1 = |D_k(w)| + 1 = |w|$. С другой стороны, для выражения $\widehat{A}(\widehat{R}(\widehat{D}(w)))$ достаточно лишь $|w| \geq 2$ (т. е. $w \in \mathbb{A}^+ \setminus \mathbb{A}$).

Композиции $F_1 \circ F_2 \circ \ldots \circ F_n$, где $F_1, \ldots, F_n \in \{\widehat{A}, \widehat{D}, \widehat{R}, \widehat{S}\}$, будем называть функциями случайного n-кратного искажения слов.

3. Функции сходства

Будем называть функцией сходства числовую функцию Φ от пары элментов некоторого множества A произвольной природы, для которой выполняются следующие условия:

1. значения функции Φ всегда лежат на отрезке от 0 до 1, т. е.
$$\forall\, x,\, y \in A \;\; \Phi\,(x,\, y) \in [0,\, 1];$$
2. если аргументы функции Φ совпадают, то она принимает значение 1, т. е.
$$\forall\, x \in A \;\; \Phi\,(x,\, x) = 1.$$

Будем использовать функции сходства как способ оценить, насколько некоторые два объекта некоторой природы «похожи» друг на друга. Будем считать, что чем больше значение $\Phi\,(x,\, y)$ функции сходства Φ, тем более «похожи» друг на друга элементы x и y. Таким образом, наибольшему возможному значению функции сходства отвечает пара одинаковых объектов, что соответствует пунктам 1 и 2 определения функции сходства.

Построим функции сходства для слов из $\mathbb{A}^*$, при использовании которых считаются «похожими» пары слов вида w и w^*, где $w \in \mathbb{V}$ и $w^* \in E\,(w)$ — однократное искажение слова w. Для этого построим преобразования слов во множества, а также функцию сходства для результирующих множеств.

Для преобразования слов во множества может использоваться любой из следующих операторов:

$$\mathcal{L}w := \big\{ w_i \colon i \in \mathbb{N}_{|w|} \big\},$$
$$\mathcal{O}w := \big\{ (w_i,\, w_j) \colon i,\, j \in \mathbb{N}_{|w|} \;\wedge\; i < j \big\}.$$

Выражение $\mathcal{L}w$ — это множество всех букв в слове w, например,

$$\mathcal{L}\,(\texttt{hello}) = \{\texttt{h},\, \texttt{e},\, \texttt{l},\, \texttt{o}\},$$

а выражение $\mathcal{O}w$ — это множество всех таких упорядоченных пар букв $(a,\, b)$, что буква a расположена левее буквы b в слове w, например,

$$\mathcal{O}\,(\texttt{hello}) = \{\texttt{he},\, \texttt{hl},\, \texttt{ho},\, \texttt{el},\, \texttt{eo},\, \texttt{ll},\, \texttt{lo}\}.$$

Функция сходства таких конечных множеств, что хотя бы одно из них не пусто, может быть задана следующим образом:

$$\Psi\,(X,\, Y) := \frac{|X \cap Y|}{|X \cup Y|},$$

где $X = \{x_1,\, \ldots,\, x_{N_X}\}$, $Y = \{y_1,\, \ldots,\, y_{N_Y}\}$. Действительно,

$$0 \le |X \cap Y| \le |X \cup Y| \quad \Rightarrow \quad \Psi\,(X,\, Y) \in [0,\, 1],$$

а также

$$|X \cap X| = |X \cup X| = |X| \quad \Rightarrow \quad \Psi\,(X,\,X) = 1.$$

Определим расширение функции Ψ на случай, когда оба его множества-аргумента пусты:

$$\widetilde{\Psi}\,(X,\,Y) := \begin{cases} 1, & X = Y = \varnothing, \\ \Psi\,(X,\,Y), & \text{иначе.} \end{cases}$$

Согласно этому определению, $\widetilde{\Psi}\,(\varnothing,\,\varnothing) = 1$, значит, функция $\widetilde{\Psi}$ также является функцией сходства.

Таким образом, используя операторы $\mathcal{L}$ и $\mathcal{O}$, а также функцию $\widetilde{\Psi}$, мы уже можем задать по крайней мере две функции сходства слов:

$$\Psi_{\mathcal{L}}(u,\,v) := \widetilde{\Psi}\,(\mathcal{L}u,\,\mathcal{L}v), \quad \Psi_{\mathcal{O}}(u,\,v) := \widetilde{\Psi}\,(\mathcal{O}u,\,\mathcal{O}v),$$

где $u,\,v \in \mathbb{A}^*$.

Определим функцию

$$\Psi_M(u,\,v) := \frac{\Psi_{\mathcal{L}}(u,\,v) + \Psi_{\mathcal{O}}(u,\,v)}{2},$$

где $u,\,v \in \mathbb{A}^*$. Докажем, что Ψ_M — функция сходства. Во-первых,

$$\forall\, u,\,v \in \mathbb{A}^* \ \begin{cases} \Psi_{\mathcal{L}}(u,\,v) \in [0,\,1] \\ \Psi_{\mathcal{O}}(u,\,v) \in [0,\,1] \end{cases} \Rightarrow$$
$$\Rightarrow \quad \forall\, u,\,v \in \mathbb{A}^* \ \tfrac{1}{2}\Psi_{\mathcal{L}}(u,\,v) + \tfrac{1}{2}\Psi_{\mathcal{O}}(u,\,v) = \Psi_M(u,\,v) \in [0,\,1],$$

т. е. функция Ψ_M выполняет пункт 1 определения функции сходства. Во-вторых,

$$\forall\, w \in \mathbb{A}^* \ \begin{cases} \Psi_{\mathcal{L}}(w,\,w) = 1 \\ \Psi_{\mathcal{O}}(w,\,w) = 1 \end{cases} \Rightarrow$$
$$\Rightarrow \quad \forall\, w \in \mathbb{A}^* \ \tfrac{1}{2}\Psi_{\mathcal{L}}(w,\,w) + \tfrac{1}{2}\Psi_{\mathcal{O}}(w,\,w) = \Psi_M(w,\,w) = 1,$$

т. е. функция Ψ_M выполняет также и пункт 2 определения функции сходства. Итого, функция Ψ_M является функцией сходства слов.

4. Связь функций сходства с функциями искажения

Рассмотрим связи функций сходства $\Psi_{\mathcal{L}}$ и $\Psi_{\mathcal{O}}$ с функциями однократного искажения A_k^a, D_k, R_k^a, S_{ij}. При этом будем рассматривать только случаи, когда в словах нет повторяющихся букв.

Пусть $w := (w_1, \ldots, w_n)$, причём $w_1, \ldots, w_n$, $a \in \mathbb{A}$ — неповторяющиеся буквы. Тогда для любого оператора $\mathcal{T} \in \{\mathcal{L}, \mathcal{O}\}$ справедливо

$$\begin{aligned} &\mathcal{T} A_k^a(w) \cap \mathcal{T} w = \mathcal{T} w, && \mathcal{T} A_k^a(w) \cup \mathcal{T} w = \mathcal{T} A_k^a(w), \\ &\mathcal{T} D_k(w) \cap \mathcal{T} w = \mathcal{T} D_k(w), && \mathcal{T} D_k(w) \cup \mathcal{T} w = \mathcal{T} w, \\ &\mathcal{T} R_k^a(w) \cap \mathcal{T} w = \mathcal{T} D_k(w), && \mathcal{T} R_k^a(w) \cup \mathcal{T} w = \mathcal{T} A_k^a(w). \end{aligned} \tag{1}$$

Теперь рассмотрим отдельно оператор $\mathcal{L}$ и функцию $\Psi_{\mathcal{L}}$. Справедливы равенства

$$\mathcal{L} S_{ij}(w) = \mathcal{L} w, \quad |\mathcal{L} w| = n, \quad |\mathcal{L} A_k^a(w)| = n + 1, \quad |\mathcal{L} D_k(w)| = n - 1. \tag{2}$$

Из равенств (1) и (2) получаем

$$\begin{aligned} &\Psi_{\mathcal{L}}(A_k^a(w), w) = \frac{n}{n+1}, \\ &\Psi_{\mathcal{L}}(D_k(w), w) = \frac{n-1}{n}, \\ &\Psi_{\mathcal{L}}(R_k^a(w), w) = \frac{n-1}{n+1}, \\ &\Psi_{\mathcal{L}}(S_{ij}(w), w) = 1. \end{aligned} \tag{3}$$

Перейдём к рассмотрению оператора $\mathcal{O}$ и функции $\Psi_{\mathcal{O}}$. Найдём значения $|\mathcal{O} w|$, $|\mathcal{O} A_k^a(w)|$, $|\mathcal{O} D_k(w)|$:

$$\begin{aligned} |\mathcal{O} w| = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} \quad \Rightarrow \\ \Rightarrow \quad |\mathcal{O} A_k^a(w)| = \frac{n(n+1)}{2}, \quad |\mathcal{O} D_k(w)| = \frac{(n-1)(n-2)}{2}. \end{aligned} \tag{4}$$

Из равенств (1) и (4) получаем

$$\begin{aligned} &\Psi_{\mathcal{O}}(A_k^a(w), w) = \begin{cases} 1, & n = 0, \\ \dfrac{n-1}{n+1}, & \text{иначе}, \end{cases} \\ &\Psi_{\mathcal{O}}(D_k(w), w) = \begin{cases} 1, & n = 1, \\ \dfrac{n-2}{n}, & \text{иначе}, \end{cases} \\ &\Psi_{\mathcal{O}}(R_k^a(w), w) = \begin{cases} 1, & n = 1, \\ \dfrac{(n-1)(n-2)}{n(n+1)}, & \text{иначе}. \end{cases} \end{aligned} \tag{5}$$

Отдельно найдём значение $\Psi_{\mathcal{O}}(S_{ij}(w),\, w)$. Пусть для определённости $i < j$. Во множестве $\mathcal{O}w$ буквы w_i и w_j содержатся в парах

$$\begin{gathered}(w_i,\, w_j),\\ (w_i,\, w_{i+1}),\, (w_i,\, w_{i+2}),\, \ldots,\, (w_i,\, w_{j-1}),\\ (w_{i+1},\, w_j),\, (w_{i+2},\, w_j),\, \ldots,\, (w_{j-1},\, w_j),\end{gathered}$$

которых всего $D_{ij} := 2d_{ij} + 1$, где $d_{ij} := |i - j| - 1$ — число букв между буквами w_i и w_j в слове w, т. е. $D_{ij} := 2\,|i - j| - 1$. Во множестве $\mathcal{O}S_{ij}(w)$ перечисленные пары отсутствуют и вместо них в таком же количестве присутствуют аналогичные пары, в которых буквы w_i и w_j заменяют друг друга на своих старых местах. Таким образом,

$$\left.\begin{aligned}|\mathcal{O}S_{ij}(w) \cap \mathcal{O}w| = |\mathcal{O}w| - D_{ij}\\ |\mathcal{O}S_{ij}(w) \cup \mathcal{O}w| = |\mathcal{O}w| + D_{ij}\end{aligned}\right\} \quad \Rightarrow$$

$$\Rightarrow \quad \Psi_{\mathcal{O}}(S_{ij}(w),\, w) = \frac{|\mathcal{O}w| - D_{ij}}{|\mathcal{O}w| + D_{ij}} = \frac{n\,(n-1) - 4\,|i-j| + 2}{n\,(n-1) + 4\,|i-j| - 2}. \tag{6}$$

Если буквы w_i и w_j являются соселними в слове w, то $|i - j| = 1$ и

$$\Psi_{\mathcal{O}}(S_{ij}(w),\, w) = \frac{n\,(n-1) - 2}{n\,(n-1) + 2}.$$

Если буквы w_i и w_j максимально удалены друг от друга в слове w, то

$$|i - j| = n - 1 \quad \text{и} \quad \Psi_{\mathcal{O}}(S_{ij}(w),\, w) = \frac{(n-4)\,(n-1) + 2}{(n+4)\,(n-1) - 2}.$$

Итого, установлены связи функций сходства $\Psi_{\mathcal{L}}$ и $\Psi_{\mathcal{O}}$ с функциями однократного искажения A_k^a, D_k, R_k^a, S_{ij}, имеющие вид (3), (5), (6). Отметим также, что при $n \to +\infty$ значения выражений (3), (5), (6) стремятся к единице, из чего следует, что при использовании функций сходства $\Psi_{\mathcal{L}}$ и $\Psi_{\mathcal{O}}$ слова вида $w \in \mathbb{V}$ и $w^* = E\,(w)$ действительно считаются похожими.

5. Алгоритм исправления опечаток

Запишем алгоритм в виде функции

$$W_\Phi^{\mathbb{V}}(w^*) := \operatorname*{arg\,max}_{w \in \mathbb{V}} \Phi(w^*,\, w), \tag{7}$$

где $w^* \in \mathbb{A}^*$ — слово, содержащее опечатку (или несколько опечаток), Φ — функция сходства слов. Идея очень проста и заключается в следующем: функция $W_\Phi^{\mathbb{V}}$ при применении к слову w^* находит в словаре $\mathbb{V}$ слово, которое оценивается при помощи функции сходства Φ как наиболее «похожее» на слово w^* среди всех слов словаря, и возвращает это слово как свой результат. Полученное слово считается ответом алгоритма на вопрос, какому корректному слову соответствует искажённое слово w^*.

6. Вычислительный эксперимент

В ходе выполнения работы для алгоритма (7) с возможностью использования функций $\Psi_{\mathcal{L}}$, $\Psi_{\mathcal{O}}$ и Ψ_M в качестве функций сходства была написана программная реализация на языке программирования Python с использованием библиотеки `numpy`. Проведён вычислительный эксперимент, по результатам которого получены данные о точности алгоритма со словарём $\mathbb{V}$, содержащим 1000 разных слов русского языка над алфавитом

$$\{\texttt{а, б, в, г, д, е, ё, ж, з, и, й}$$
$$\texttt{к, л, м, н, о, п, р, с, т, у, ф}$$
$$\texttt{х, ц, ч, ш, щ, ъ, ы, ь, э, ю, я}\},$$

для разных видов случайных искажений и разных функций сходства.

Словарь $\mathbb{V}$ формируется как случайная выборка без повторений объёмом в 1000 слов, каждое из которых состоит из как минимум трёх букв, из 10 тыс. наиболее часто встречающихся слов русского языка. Каждая из функций $W^{\mathbb{V}}_{\Psi_{\mathcal{L}}}$, $W^{\mathbb{V}}_{\Psi_{\mathcal{O}}}$, $W^{\mathbb{V}}_{\Psi_M}$ проходит серию тестов, которую можно разделить на 3 набора: с однократными, двукратными и трёхкратными искажениями слов. Массивы входных данных генерируются путём применения случайных функций $\widehat{A}$, $\widehat{D}$, $\widehat{R}$, $\widehat{S}$ и их композиций к элементам словаря $\mathbb{V}$. Таким образом, набор с однократными искажениями слов состоит из 4 тестов, с двукратными — из $4\times 4 = 16$ тестов, с трёхкратными — из $4\times 4\times 4 = 64$ тестов. В результате каждого теста вычисляется относительная ошибка

$$\delta = \frac{N - T}{N},$$

где $N = 1000$ — число всех испытаний в тесте, $T \in \{0, 1, \ldots, N\}$ — число испытаний в тесте, в которых алгоритм восстановил слово правильно.

Результаты всех тестов для всех функций сходства приведены в процентах ниже в таблице для однократных искажений и на тепловых картах для дву- и трёхкратных искажений. Словами add, del, repl, swap обозначены применения для генерирования массивов входных данных функций $\widehat{A}$, $\widehat{D}$, $\widehat{R}$, $\widehat{S}$ соответственно.

$\Psi_{\mathcal{L}}$				$\Psi_{\mathcal{O}}$				Ψ_M			
add	del	repl	swap	add	del	repl	swap	add	del	repl	swap
3.1%	6.1%	16.1%	0.9%	0.2%	1.3%	2.7%	4.1%	0.2%	1.7%	5.1%	0.7%

На рис. 1–3 приведены результаты тестов с двукратными искажениями для функций сходства $\Psi_{\mathcal{L}}$, $\Psi_{\mathcal{O}}$ и Ψ_M. Индекс строки представляет вторую в порядке применения функцию искажения, применённую при генерировании массива входных данных, индекс столбца — первую. Например, в позиции (del, repl)

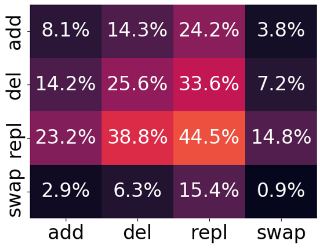

Рис. 1: $\Psi_{\mathcal{L}}$

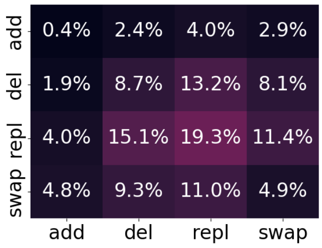

Рис. 2: $\Psi_{\mathcal{O}}$

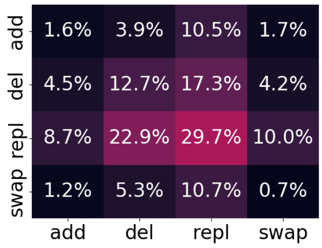

Рис. 3: Ψ_M

записана относительная ошибка, полученная при применении алгоритма к массиву входных данных $\left[\widehat{D}\left(\widehat{R}\left(w\right)\right)\right]_{w\in\mathbb{V}}$.

На рис. 4–15 приведены результаты тестов с трёхкратными искажениями для функций сходства $\Psi_{\mathcal{L}}$, $\Psi_{\mathcal{O}}$ и Ψ_M. Индексы строк и столбцов имеют тот же смысл, что и на рис. 1–3. Слова add, del, repl, swap в подписях рисунков обозначают третью в порядке применения функцию искажения, применённую при генерировании массива входных данных. Например, в позиции (del, repl) матрицы, подписанной словом add, записана относительная ошибка, полученная при применении алгоритма к массиву входных данных $\left[\widehat{A}\left(\widehat{D}\left(\widehat{R}\left(w\right)\right)\right)\right]_{w\in\mathbb{V}}$.

По результатам вычислительного эксперимента можно сделать следующие выводы:

- алгоритм с высокой точностью (а при использовании функции сходства $\Psi_{\mathcal{O}}$ или Ψ_M — с крайне высокой точностью) восстанавливает слова, содержащие одну опечатку;

- алгоритм справляется с заменами значительно хуже, чем с другими искажениями; также заметные трудности у алгоритма вызывают удаления;

- алгоритм, использующий функцию сходства $\Psi_{\mathcal{L}}$, лучше справляется с перестановками, а алгоритм, использующий функцию сходства $\Psi_{\mathcal{O}}$ — с другими искажениями и композициями;

- алгоритм, использующий функцию сходства Ψ_M, очень хорошо справляется с перестановками, а с иными искажениями — хуже алгоритма с $\Psi_{\mathcal{O}}$ и лучше алгоритма с $\Psi_{\mathcal{L}}$.

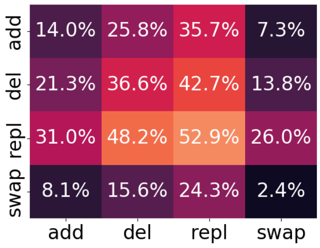

Рис. 4: $\Psi_{\mathcal{L}}$, add

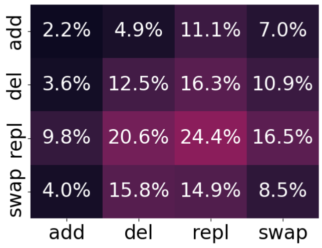

Рис. 5: $\Psi_{\mathcal{O}}$, add

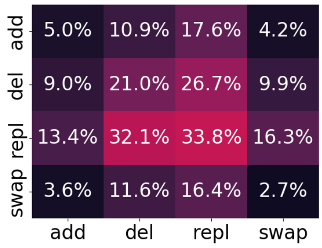

Рис. 6: Ψ_M, add

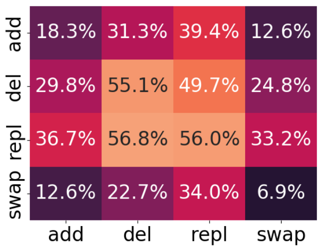

Рис. 7: $\Psi_{\mathcal{L}}$, del

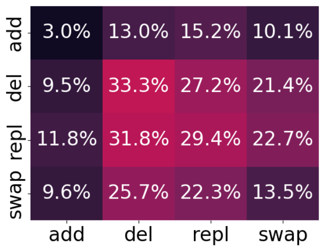

Рис. 8: $\Psi_{\mathcal{O}}$, del

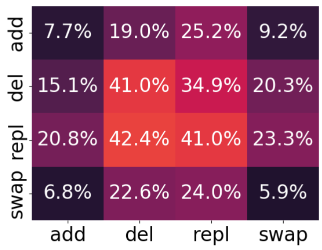

Рис. 9: Ψ_M, del

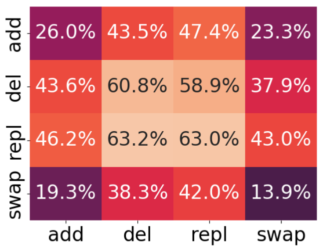

Рис. 10: $\Psi_{\mathcal{L}}$, repl

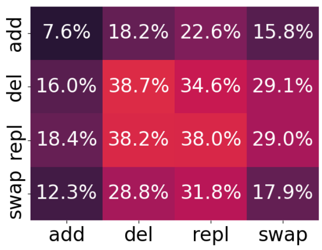

Рис. 11: $\Psi_{\mathcal{O}}$, repl

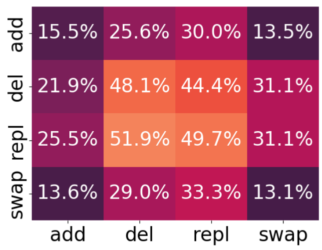

Рис. 12: Ψ_M, repl

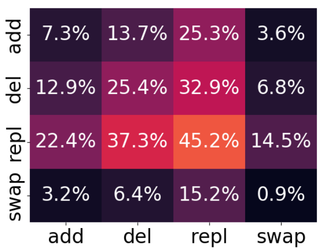

Рис. 13: $\Psi_{\mathcal{L}}$, swap

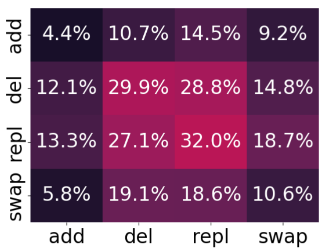

Рис. 14: $\Psi_{\mathcal{O}}$, swap

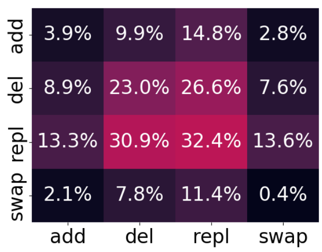

Рис. 15: Ψ_M, swap

Заключение

В результате выполнения работы заявленная цель была достигнута. Все поставленные задачи также были выполнены. Построены математические модели слов и различных опечаток в них, функции сходства, алгоритм исправления опечаток. Высокая точность результатов работы построенного алгоритма, полученная во многих из проведённых тестов, позволяет сделать вывод о пригодности его использования для исправления опечаток и перспективности исследований, разработок и развития подобных алгоритмов. Полученные сведения о преимуществах и недостатках различных функций сходства при их использовании для исправления опечаток могут быть полезны в будущих исследованиях.

Список литературы

[1] Белоусов А. И., Ткачёв С. Б. Дискретная математика: Учеб. для вузов / Под ред. В. С. Зарубина, А. П. Крищенко. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2001.

[2] Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академий Наук СССР, 1965.

[3] Гасфилд. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология // Невский Диалект, БВХ-Петербург, 2003.

[4] Church K. W., Gale W. A. Probability scoring for spelling correction // Stat Comput 1, 93–103 (1991). https://doi.org/10.1007/BF01889984.

[5] Stankevičius L., Lukoševičius M., Kapočiūtė-Dzikienė J. Correcting diacritics and typos with a ByT5 transformer model.

Приложение

Файл `operations.py` исходного кода программной реализации:

```
import numpy as np

# (x_1, ..., x_n) -> { (x_i, x_j) : 1 <= i < j <= n }
def set_of_orders(word):
    n = len(word)
    return {
        (word[i], word[j])
            for i in range(n - 1) for j in range(i + 1, n)
    }

# (x_1, ..., x_n) -> { x_i : 1 <= i <= n }
class SetOfLetters:

    def __init__(self):
        self.cache = dict()

    def __call__(self, word):
        if word not in self.cache:
            self.cache[word] = set(word)
        return self.cache[word]

# (x_1, ..., x_n) -> { (x_i, x_j) : 1 <= i < j <= n }
class SetOfOrders:

    def __init__(self):
        self.cache = dict()

    def __call__(self, word):
        if word not in self.cache:
            n = len(word)
            self.cache[word] = {
                (word[i], word[j])
                    for i in range(n - 1) for j in range(i + 1, n)
            }
        return self.cache[word]

# (A, B) -> measure(A & B) / measure(A | B)
```

```
def andor_similarity(A, B):

    emptyA = (len(A) == 0)
    emptyB = (len(B) == 0)

    if emptyA or emptyB:
        return float(emptyA and emptyB)

    return len(A & B) / len(A | B)

def mean(*x):
    return np.array(x).mean()

# (f, (g_1, ..., g_n)) -> (
#     (x_1, ..., x_n) -> f(g_1(x_1), ..., g_n(x_n))
# )
class Compound:

    def __init__(self, outer, inners):
        self.outer = outer
        self.inners = inners

    def __call__(self, *x):
        return self.outer(
            *[f_i(x_i) for f_i, x_i in zip(self.inners, x)]
        )

# (f, (g_1, ..., g_n)) -> (x -> f(g_1(x), ..., g_n(x)))
class Ensemble:

    def __init__(self, parents, crossover):
        self.parents = parents
        self.crossover = crossover

    def __call__(self, *x):
        return self.crossover(
            *[parent(*x) for parent in self.parents]
        )
```

Файл `distortions.py` папки `word_recovery` исходного кода программной реализации:

```
import random

alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

def rand_add(word):
    li = list(word)
    i = random.randint(0, len(word))
    c = random.choice(alphabet)
    if i == len(word):
        li.append(c)
    else:
        li.insert(i, c)
    return ''.join(li)

def rand_delete(word):
    li = list(word)
    li.pop(random.randint(0, len(word) - 1))
    return ''.join(li)

def rand_replace(word):
    li = list(word)
    i = random.randint(0, len(word) - 1)
    li[i] = random.choice(alphabet)
    return ''.join(li)

def rand_swap(word):
    li = list(word)
    i, j = random.sample(range(len(word)), 2)
    li[i], li[j] = li[j], li[i]
    return ''.join(li)
```

Файл `word_recovery.py` папки `word_recovery` исходного кода программной реализации:

```
import numpy as np

class WordRecovery:

    def __init__(self, similarity, vocabulary):
        self.similarity = similarity
        self.vocabulary = vocabulary

    def __call__(self, word):
        mapsims = map(
            lambda w: self.similarity(word, w),
            self.vocabulary
        )
        npsims = np.array(list(mapsims))
        return self.vocabulary[npsims.argmax()]
```

Файл test_recovery.py (точка входа) исходного кода программной реализации:

```
import time
import numpy as np

import operations as op

import word_recovery.distortions as dt
import word_recovery.word_recovery as wr


class ListMapper:

    def __init__(self, fun):
        self.fun = fun

    def __call__(self, args):
        return list(map(self.fun, args))


def relative_error(original, recovered):
    total_count = len(original)
    if total_count != len(recovered):
        print('relative_error: different lengths')
        exit()
    true_count = sum(
        map(lambda x: x[0] == x[1], zip(original, recovered))
    )
    return (total_count - true_count) / total_count


print('Words reading...')

words = []
with open(
    'input/10000-russian-words-cyrillic-only.txt',
    'r', encoding = 'utf-8'
) as fin:
    words = fin.read().splitlines()
```

```
print('Done.\n')


word_count = 1000

print(f'Sampling {word_count} words...')

words = np.random.choice(
    list(filter(lambda w: len(w) > 3, words)),
    size = word_count, replace = False
)

print('Done.\n')


print('Preparing distortion functions and model...')

distortfun_LM_list = [
    ListMapper(dt.rand_add),
    ListMapper(dt.rand_delete),
    ListMapper(dt.rand_replace),
    ListMapper(dt.rand_swap),
]

model = wr.WordRecovery(
    #similarity = op.Compound(
    #    outer = op.andor_similarity,
    #    inners = [set, op.SetOfLetters()]
    #),
    #similarity = op.Compound(
    #    outer = op.andor_similarity,
    #    inners = [op.set_of_orders, op.SetOfOrders()]
    #),
    similarity = op.Ensemble(
        parents = [
            op.Compound(
                outer = op.andor_similarity,
                inners = [set, op.SetOfLetters()]
            ),
            op.Compound(
```

```
                outer = op.andor_similarity,
                inners = [op.set_of_orders, op.SetOfOrders()]
            ),
        ],
        crossover = op.mean
    ),
    vocabulary = words
)

model_LM = ListMapper(model)

print('Done.\n')


print('Words distortion...')

D_1D = [
    F_i(words)
        for F_i in distortfun_LM_list
]

D_2D = [
    [
        F_i(F_j(words))
            for F_j in distortfun_LM_list
    ]
        for F_i in distortfun_LM_list
]

D_3D = [
    [
        [
            F_i(F_j(F_k(words)))
                for F_k in distortfun_LM_list
        ]
            for F_j in distortfun_LM_list
    ]
        for F_i in distortfun_LM_list
]
```

```
print('Done.\n')


print('Words 1D recovery...')

time_start = time.time()

R_1D = [model_LM(D_i) for D_i in D_1D]

time_finish = time.time()

total_s = time_finish - time_start
average_ms = total_s / (len(distortfun_LM_list) * word_count) * 1000

print('Done.')
print(f'Average word recovery time: {average_ms:.3f} ms.
Total time: {total_s:.3f} s.\n')


print('Words 2D recovery...')

time_start = time.time()

R_2D = [[model_LM(D_ij) for D_ij in D_i] for D_i in D_2D]

time_finish = time.time()

total_s = time_finish - time_start
average_ms = total_s / (
    (len(distortfun_LM_list) ** 2) * word_count
) * 1000

print('Done.')
print(f'Average word recovery time: {average_ms:.3f} ms.
Total time: {total_s:.3f} s.\n')


print('Words 3D recovery...')

time_start = time.time()
```

```
R_3D = [
    [
        [
            model_LM(D_ijk)
                for D_ijk in D_ij
        ]
            for D_ij in D_i
    ]
        for D_i in D_3D
]

time_finish = time.time()

total_s = time_finish - time_start
average_ms = total_s / (
    (len(distortfun_LM_list) ** 3) * word_count
) * 1000

print('Done.')
print(f'Average word recovery time: {average_ms:.3f} ms.
Total time: {total_s:.3f} s.\n')


relerrors_1D = [relative_error(words, R_i) for R_i in R_1D]

relerrors_2D = [
    [
        relative_error(words, R_ij)
            for R_ij in R_i
    ]
        for R_i in R_2D
]

relerrors_3D = [
    [
        [
            relative_error(words, R_ijk)
                for R_ijk in R_ij
        ]
```

```
            for R_ij in R_i
    ]
        for R_i in R_3D
]

with open(
    'output/test_recovery/relerrors_M.txt',
    'w', encoding = 'utf-8'
) as fout:
    with np.printoptions(
        formatter = {'all': lambda x: f'{x * 100:>4.1f} % '}
    ):
        print(
            f'Relative errors 1D:\n{np.array(relerrors_1D)}\n',
            file = fout
        )
        print(
            f'Relative errors 2D:\n{np.array(relerrors_2D)}\n',
            file = fout
        )
        print(
            f'Relative errors 3D:\n{np.array(relerrors_3D)}\n',
            file = fout
        )
```