Uniwersytet im. Adama Mickiewicza w Poznaniu
Wydział Matematyki i Informatyki

Praca magisterska

Probabilistyczne metody korekty pisowni

Probabilistic methods for spell-checking

Tomasz Posiadała

nr albumu: 362669
kierunek: informatyka

Promotor:
prof. UAM dr hab. Krzysztof Jassem

Poznań, 2017

Oświadczenie

Ja, niżej podpisany Tomasz Posiadała student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: Probabilistyczne metody korekty pisowni napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[ ]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....

(czytelny podpis studenta)

Streszczenie

Celem niniejszej pracy magisterskiej jest stworzenie korektora pisowni z wykorzystaniem wybranych metod probabilistycznych, o zbliżonej jakości działania do korektorów wykorzystujących reguły do wykrywania, klasyfikacji oraz poprawy błędów. Program poprawia wprowadzany tekst w języku polskim. Zgodnie z wiedzą autora do tej pory nie stworzono jeszcze narzędzia, mimo dużej konkurencji wśród korektorów pisowni, korzystającego z metod probabilistycznych do sprawdzania pisowni języka polskiego.

W pracy zawarto: opis metod wykrywania błędów w pisowni, sposobu szukania kandydatów do poprawy (algorytm Damerau-Levenshteina) oraz metod probabilistycznych takich jak: metoda N-gramów, „Backoff", algorytm Bayesa. Omówiono ponadto otwarto-źródłowy korektor pisowni *LanguageTool* oraz korpus błędów *PLEWi*. Przedstawiono również zasadę działania autorskiego programu i oceniono wyniki zwracane przez program na podstawie ostatecznej ewaluacji.

Abstract

The purpose of the thesis is to create a spelling corrector, which applies selected probabilistic methods of similar performance quality to correctors, which use rules to detect, classify and correct errors. The program corrects the text written in the Polish language. According to the author's knowledge, despite the high competition among spelling correctors, no such tool using probabilistic methods for spell checking has been created for the Polish language.

The thesis includes: methods for detecting spelling errors, algorithms for searching for correct candidates (Damerau-Levenshtein algorithm) as well probabilistic methods such as: the N-gram method, „Backoff", Bayesian algorithm. Moreover, the thesis includes description of the open-source project *LanguageTool* and the *PLEWi* error corps. It also discusses the basics of the created author's program and evaluates the results returned by the program of the final evaluation.

Spis treści

Wstęp

Zakres pracy

W pracy omawia się wykorzystanie metod probabilistycznych w korekcji błędów języka. Opisany jest cały proces począwszy od detekcji błędu, a skończywszy na procesie poprawiania. W opisywanym eksperymencie wykorzystuje się następujące pojęcia oraz metody: odległość edycyjna Damerau-Levenshteina, metoda N-gramów, algorytm Bayes'a, macierz błędów, model języka, model błędów oraz model zaszumionego kanału.

Cele pracy

Celem pracy jest wykorzystanie metod probabilistycznych do stworzenia korektora pisowni, o zbliżonej jakości działania do korektorów wykorzystujących reguły do wykrywania, klasyfikacji oraz poprawy błędów. Efektem końcowym jest aplikacja, która we wprowadzonym tekście w języku polskim wyszukuje i poprawia błędy nienależące do słownika języka polskiego.

Układ pracy

Wstęp zawiera zakres, cele oraz układ pracy. Rozdział 1. przedstawia podstawy teoretyczne projektu magisterskiego. Rozdział 2. to metody korekty pisowni języka polskiego. Rozdział 3. zawiera dokładny opis eksperymentu oraz stworzonej aplikacji. Rozdział 4. obejmuje ewaluację i wnioski. Zakończenie to podsumowanie pracy i perspektywy dalszych badań z użyciem podobnych metod.

Rozdział 1

Podstawy teoretyczne

Rozdział zawiera wszystkie definicje pojęć potrzebnych do dogłębnego zrozumienia poruszanych problemów oraz opis modeli, które są powszechnie używane w korekcji pisowni. Aby korektor pisowni działał jak najlepiej, potrzebny jest zestaw narzędzi, które umożliwiają wyszukanie błędu, dokładne oszacowanie prawdopodobieństwa błędu oraz wybór najlepszego kandydata do korekty.

1.1 Definicja błędu językowego

Zgodnie z definicją zawartą w Encyklopedii Języka Polskiego PWN błąd językowy to „uchybienie zwyczajowi powszechnej mowy zwanej kulturalną, niemające oparcia i usprawiedliwienia ani w panującym współcześnie ładzie, systemie językowym, ani w jakiejś starszej lub nowszej dążności ewolucyjnej języka" [1]. Język jednak stale ewoluuje, dlatego nie wszystkie nowe wyrażenia są traktowane jako błędy. Jeśli mają one funkcjonalne uzasadnienie, nie mogą być traktowane jako błędy, ale jako innowacja językowa [1].

1.2 Typy błędów językowych

Wszystkie elementy tekstu niespełniające kryteriów poprawności językowej określa się jako błąd językowy. Ze względu na różnorodność odstępstw od norm wszelkie błędy zostały podzielone na dwie główne kategorie - wewnętrznojęzykowe i zewnętrznojęzykowe.

Błędy wewnętrznojęzykowe naruszają ogólne reguły systemowe i zasady rozwoju języka. Poniżej jest przedstawiony ich szczegółowy podział:

1. Systemowe:
   (a) gramatyczne:
      - fleksyjne, polegające m.in. na wyborze niewłaściwej postaci wyrazu, np. *jesteśmy przyjacielami* zamiast *jesteśmy przyjaciółmi*,
      - składniowe, polegające m.in. na wyborze niewłaściwego wzorca wyrazowego, np. *Oprowadzała nas niemiecka przewodniczka po Krakowie* zamiast *Niemiecka przewodniczka oprowadzała nas po Krakowie*,

   (b) leksykalne:
      - słownikowe (wyrazowe), polegające m.in. na posługiwaniu się pleonazmami, np. *cofnąć się do tyłu* zamiast *cofnąć się*,
      - frazeologiczne, polegające m.in. na zmianie formy związków frazeologicznych, np. *ciężki orzech do zgryzienia* zamiast *twardy orzech do zgryzienia*,

   (c) fonetyczne - związane z błędną wymową głosek,
2. Stylistyczne - polegają na niedostosowaniu wyrazów do charakteru wypowiedzi, np. używanie kolokwializmów w wypowiedziach oficjalnych.

Błędy zewnętrznojęzykowe dotyczą błędów zapisu, ale nie naruszają powszechnych zasad gramatycznych języka. Dzielą się one na:

1. ortograficzne, np. *mruwka, codzień, maria* (poprawnie: *mrówka, co dzień, Maria*)
2. interpunkcyjne - polegające na nadużywaniu, braku, albo użyciu błędnego znaku interpunkcyjnego, np. *Nie wiedziałem że masz z tym problem.* (poprawnie: *Nie wiedziałem, że masz z tym problem.*) [2].

1.3 Typy błędów w pisowni

Powyżej zostały przedstawione rodzaje błędów, jakie popełniane są w języku polskim. Jednak dla celów korekcji pisowni wprowadza się inny podział. Wszystko dlatego, że błędy popełniane podczas pisania na klawiaturze to, przede wszystkim, literówki. Ich podział wygląda następująco [3]:

Błędy nie będące słowami języka polskiego - jest to prostszy przypadek do wykrycia. Należą do nich wszystkie wyrazy, których nie ma w słowniku języka polskiego.
Przykład:

żrafa → żyrafa

Błędy będące słowami języka polskiego - wyrazy, które istnieją w języku polskim, ale mimo to są błędnie użyte w zdaniu, przez co jest większa trudność ich wykrycia. Dzielą się na dwa przypadki:

1. Błędy typograficzne (literówki) - powstają po wstawieniu, usunięciu, zamianie lub transpozycji liter tworząc inny, należący do języka polskiego, wyraz.
   Przykład: granat → granit

2. Błędy poznawcze (homofony) - powstają przy błędnym użyciu homofonów (wyrazów brzmiących tak samo, ale różniących się pisownią i znaczeniem) w kontekście.
   Przykład: lud → lód

1.4 Proces wykrywania błędów w pisowni

Głównymi zadaniami korekcji pisowni jest znalezienie błędu i jego poprawienie. Jest jednak różnica w wykrywaniu błędów, które nie należą do słownika języka polskiego oraz tych, które do niego należą.

Pierwszy przypadek jest bardzo prosty - każdy wyraz, który nie należy do słownika jest uważany za błąd. Wczesne badania J. L. Petersona [12] z 1986 roku

sugerowały, że słowniki powinny być stosunkowo małe, ponieważ wielkie słowniki języka zawierały bardzo rzadkie słowa, które przypominają błędy popełniane z innych słów. Peterson pokazał, że nawet 10%-15% takich błędów może być ukrytych za prawdziwymi słowami. Te badania zostały jednak szybko zweryfikowane. Damerau i Mays [13] w 1989 roku pokazali, że w praktyce takich błędów jest o wiele mniej. Wprawdzie niektóre z nich były ukryte za prawdziwymi wyrazami, jednak większy słownik bardziej pomagał niż szkodził. Zatem im większy słownik mamy do dyspozycji, tym łatwiej wykrywać błędy pisowni. W projekcie magisterskim używany jest elektroniczny słownik zawierający formy podstawowe słów z odpowiednimi dla nich formami fleksyjnymi. Opisany on został w sekcji 1.10.

Zupełnie inaczej wykrywa się błędnie zapisane wyrazy, które są słowami języka polskiego. Nie można określić błędów poprzez weryfikację słownikową. Mogą się zdarzyć nieumyślnie poprzez popełnienie literówki, która stworzy inny wyraz należący do języka polskiego lub użycie homofonu. Takich błędów nie da się jednoznacznie wykryć, a trzeba stworzyć zbiór potencjalnych kandydatów dla każdego słowa w zdaniu [4], o czym mowa jest sekcji 2.5. Wszystko dlatego, że każdy wyraz w zdaniu może być błędem.

W stworzonym projekcie magisterskim będą rozpoznawane tylko błędy nienależące do słownika języka polskiego.

1.5 Proces korekcji błędów

Po wykryciu potencjalnych błędów należy przejść do procesu korekcji. Składa się on z dwóch głównych zadań:

1. Generowanie kandydatów - polega na dostarczeniu listy wyrazów, które najprawdopodobniej miały zostać wpisane przez użytkownika, ale w wyniku błędu zostały zmienione. Należy znaleźć najbardziej podobne słowa do błędu. W tym celu stosuje się algorytm wyszukiwania opartego o odległość edycyjną Damerau-Levenshteina. Algorytm ten znajduje wyrazy, do utworzenia których niezbędna jest niewielka liczba korekt edycyjnych dokonanych na wyrazie błędnym.

2. Obliczanie prawdopodobieństwa - do jego wyliczenia potrzebne są dane pochodzące z dwóch źródeł. Jednym z nich jest korpus języka polskiego, za

pomocą którego tworzy się model języka. Dzięki niemu wiadomo, z jakim prawdopodobieństwem słowo, bądź sekwencja słów pojawia się w języku. Drugie ze źródeł to korpus błędów, dzięki któremu będzie można sprawdzić, jak często pewne słowo generuje daną pomyłkę. Z tego korpusu tworzy się tzw. model błędów, który zawiera prawdopodobieństwo popełnienia każdego błędu. Każdy z wymienionych modeli tworzy się normalizując dane. Po przemnożeniu uzyskanych wyników otrzymuje się końcowe prawdopodobieństwo dla kandydata do poprawy. Proces ten powtarzany jest dla każdego z wygenerowanych wcześniej kandydatów i przypisywany jest do niego wyliczony wynik.

Na końcu, gdy prawdopodobieństwo zostało wyliczone dla każdego z kandydatów, poprawiane są błędy. Oto sposoby ich poprawy w programie [21]:

- Autokorekta - program sam zamienia błąd na kandydata o największym prawdopodobieństwie,
- Propozycja korekty - program wyświetla kandydata do korekty z największym prawdopodobieństwem,
- Lista propozycji korekty - program wyświetla listę kandydatów wraz z ich prawdopodobieństwami, z której użytkownik może wybrać odpowiednią podpowiedź.

W projekcie magisterskim program dokonuje korekty automatycznie na podstawie obliczonych prawdopodobieństw każdego z kandydatów, ale daje również możliwość jego zmiany przy użyciu rozwijanej listy pięciu najbardziej prawdopodobnych kandydatów.

1.6 Odległość edycyjna Damerau-Levenshteina

Odległość edycyjna [5] - to minimalna liczba edycji potrzebna do zmiany jednego ciągu znaków w drugi. Została zaproponowana w 1965 roku przez Władimira Levenshteina, a później uogólniona przez Fredericka J. Damerau, który uznał również za działanie proste zamianę miejscami dwóch sąsiednich znaków (transpozycja).

Dozwolone edycje (działania proste):

- wstawienie nowego znaku do napisu
- usunięcie znaku z napisu
- zamiana znaku w napisie na inny znak
- zamiana miejscami sąsiednich dwu znaków

Aby obliczyć odległość Damerau-Levenshteina pomiędzy dwoma ciągami znaków a i b definiuje się funkcję $d_{a,b}(i,j)$, której wartość jest odległością między prefiksem i (początkowym podciągiem) ciągu znaków a, a prefiksem j ciągu b.

Funkcja jest definiowana rekurencyjnie, jako [14]:

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{jeśli } \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{jeśli} i,j > 1 \text{ i } a_i = b_{j-1} \text{ i } a_{i-1} = b_j \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{w przeciwnym wypadku.} \end{cases}$$

gdzie $1_{(a_i \neq b_j)}$ to funkcja charakterystyczna zbioru, której wartość wynosi 0 gdy $a_i = b_i$, a w przeciwnym wypadku wynosi 1.

Każde wywołanie rekurencyjne pasuje do jednego z przypadków objętych odległością edycyjną Damerau-Levenshteina:

- $d_{a,b}(i-1,j)+1$, odpowiada usunięciu,
- $d_{a,b}(i,j-1)+1$, odpowiada wstawieniu,
- $d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}$, odpowiada zamianie,
- $d_{a,b}(i-2,j-2)+1$, odpowiada transpozycji.

Odległość Damerau-Levenshteina pomiędzy a i b jest wartością funkcji dla pełnych ciągów: $d_{a,b}(|a|,|b|)$, gdzie $i = |a|$ oznacza długość łańcucha a, a $j = |b|$ jest długością łańcucha b.

Badanie odległości edycyjnej między wyrazami ukazuje poniższy przykład:

marka → czakra

W tym przykładzie odległość edycyjna jest równa 3, ponieważ do przeprowadzenia pierwszego wyrazu na drugi potrzebne są 3 edycje:

- dodanie litery "c" na początku wyrazu → cmarka
- zamiana litery "m" na "z" → czarka
- zamiana kolejnością sąsiednich liter "rk" na "kr" → czakra

Zaprezentowany przykład błędu z tak dużą odległością edycyjną jest jednak prawie niemożliwy do zaobserwowania. W 1964 roku F.J.Damerau [14] opublikował artykuł, a w nim statystykę, która pokazała, że 80% błędnie napisanych słów znajduje się w odległości edycyjnej równej 1, natomiast prawie wszystkie w odległości edycyjnej nie większej niż 2. Używając innego korpusu, J. L. Peterson [12] w 1986 roku stwierdził, że odsetek ten przy pojedynczych błędach jest jeszcze większy i zawiera się w przedziale 93%-95%.
Na podstawie powyższych statystyk została podjęta decyzja, iż w projekcie będą wyszukiwani kandydaci w odległości Damerau-Levenshteina równej 1.

1.7 Model języka

Model języka jest rozkładem prawdopodobieństwa na sekwencji słów w danym korpusie. Aby go obliczyć, trzeba zdefiniować słownik V, jako zbiór wszystkich słów w języku. Zdaniem języka jest sekwencja słów $x_1, x_2, ...x_n$, gdzie każdy wyraz x (poza ostatnim x_n, który jest symbolem końca zdania) należy do słownika V. Model języka [19] składa się ze skończonego zbioru V i funkcji $p(x_1, x_2, ...x_n)$ takiej, że:

1. Dla każdego zdania $\langle x_1, x_2, \ldots x_n \rangle \in V^\dagger$, $p(x_1, x_2, \ldots x_n) \geq 0$,
2. Dodatkowo,
$$\sum_{\langle x_1 \ldots x_n \rangle \in V^\dagger} p(x_1, x_2 \ldots x_n) = 1$$

gdzie $V^\dagger$ jest zbiorem wszystkich zdań na słowniku V: zbiór ten jest nieskończony, ponieważ zdania mogą być dowolnej długości.

Za pomocą modelu języka można oszacować prawdopodobieństwo wystąpienia wyrazu lub pewnej sekwencji wyrazów w korpusie. Wiadomo, że napotkanie następującej sekwencji słów napisanych w języku polskim ma wysokie prawdopodobieństwo [6]:

czy chciałbyś ze mną pójść...

jednak prawdopodobieństwo napotkania tego samego zbioru słów w zupełnie innej kolejności jest bardzo małe:

pójść mną ze chciałbyś czy

Model języka jest przydatny w procesie korekcji tekstu zarówno w przypadku błędów nienależących do słownika, jak również tych, które do niego należą. Do obliczania rozkładu prawdopodobieństwa wystąpienia pewnej sekwencji wyrazów $w_1 w_2 \dots w_n$ w korpusie służą następujące metody:

1. Metoda N-gramów [7] - metoda ta pozwala na przewidywanie kolejnego wyrazu na podstawie wystąpień sekwencji $N-1$ dotychczasowych. Opiera się na prawdopodobieństwie warunkowym. Oblicza się je za pomocą wzoru:

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i | w_{i-N+1} \dots w_{i-1})$$

   gdzie N zmienia się w zależności od tego, jakiego N-gramu używamy.

   N-gram trenuje się poprzez zliczanie i normalizację. Dla modeli probabilistycznych normalizacja oznacza dzielenie przez pewną liczbę całkowitą tak, żeby otrzymane w wyniku działania prawdopodobieństwo zawierało się w przedziale od 0 do 1. Oznacza to, iż każde prawdopodobieństwo warunkowe oblicza się za pomocą wzoru [8]:

$$P(w_i | w_{i-N+1} \dots w_{i-1}) = \frac{count(w_{i-N+1}, \dots, w_{i-1}, w_i)}{count(w_{i-N+1}, \dots, w_{i-1})}$$

Do obliczenia prawdopodobieństwa całej sekwencji używając bigramu powyższy wzór wygląda następująco:

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i|w_{i-1})$$

Poszczególne składowe oblicza się ze wzoru:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Oznacza to, że oblicza się ilość wystąpień sekwencji w_{i-1}, w_i w korpusie i dzieli się przez ilość wystąpień słowa w_{i-1}.

Model N-gramów nazywany jest również modelem Markowa: model bigramowy jest nazywany modelem Markowa pierwszego rzędu, a model trigramowy modelem drugiego rzędu itd. Najczęściej używa się modeli: unigram, bigram, bądź trigram.

2. Backoff [9] - opiera się na algorytmie N-gramów. Metoda pozwala radzić sobie z problemem wartości zerowych wyliczonych tym algorytmem. Może się bowiem zdarzyć sytuacja, że pewna sekwencja wyrazów w ogóle nie występuje w korpusie języka, z którego się korzysta, więc jej prawdopodobieństwo wynosi zero.
   Jeśli korpus języka nie zawiera pewnego trigramu $w_{n-2}w_{n-1}w_n$ i nie można obliczyć $P(w_n|w_{n-1}w_{n-2})$, to szacuje się jego prawdopodobieństwo używając bigramu $P(w_n|w_{n-1})$. Podobnie, gdy nie mamy wystąpień sekwencji wyrazów do obliczenia $P(w_n|w_{n-1})$, korzystamy z metody unigram $P(w_n)$, która pozwala nam obliczyć prawdopodobieństwo wystąpienia jednego wyrazu w korpusie. Można stwierdzić, że jest to udoskonalenie metody N-gramów. Działanie algorytmu określa się wzorem:

$$\hat{P} = (w_i|w_{i-2}w_{i-1}) = \begin{cases} P(w_i|w_{i-2}w_{i-1}), & \text{gdy } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha P(w_i|w_{i-1}), & \text{gdy } C(w_{i-2}w_{i-1}w_i) = 0 \\ & \text{oraz } C(w_{i-1}w_i) > 0 \\ \alpha_2 P(w_i), & \text{w innym przypadku} \end{cases}$$

W eksperymencie autorskim zastosowano algorytm stosujący unigramy, którego wzór jest przedstawiony poniżej:

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Prawdopodobieństwo wystąpienia danej sekwencji wyrazów w korpusie jest iloczynem prawdopodobieństw wystąpienia każdego wyrazu w. Warto zwrócić uwagę, że tylko w metodzie unigramów nie używamy wzorów na prawdopodobieństwo warunkowe, ponieważ zakłada się, że słowa nie są zależne od siebie.

1.8 Model błędów

Obliczanie prawdopodobieństwa popełnienia błędu podczas pisania pewnego słowa jest wymagającym problemem. Uzyskanie dokładnego wyniku jest praktycznie niemożliwe, ponieważ zależy od tego, kto pisze tekst, jak dobrze zna klawiaturę, której używa, czy któraś z rąk piszącego jest zmęczona itd. Można je natomiast precyzyjnie oszacować, ponieważ najważniejsze czynniki przewidujące wstawianie, usuwanie, zamianę, transpozycję są prostymi czynnikami lokalnymi (nie zależą od piszącego i innych czynników zewnętrznych), takimi jak podobieństwo prawidłowej litery, sposób, w jaki litera została nieprawidłowo napisana i otaczający kontekst. Przykładowo litery m i n są często błędnie zamieniane wzajemnie: częściowo przez podobieństwo (mają podobną wymowę i znajdują się obok siebie na klawiaturze) oraz częściowo przez podobny kontekst (mogą występować w podobnych kontekstach, ponieważ ich wymowa jest podobna) [10].

Kernighan i in. [15] w 1990 roku zaproponowali prostą metodę do oszacowania prawdopodobieństw tych błędów. Zignorowali oni większość czynników wpływających na prawdopodobieństwo błędu i oszacowali np. $P(grubt|gruby)$ można oszacować, licząc, ile razy litera t została zamieniona na y w dużym korpusie błędów (jest to prawdopodobna pomyłka, ponieważ na klawiaturze znajdują się obok siebie). Liczby te reprezentują macierze błędów dla każdej możliwej edycji:

- del[x,y]: liczba(xy wpisanych jako x),
- ins[x,y]: liczba(x wpisanych jako xy),
- sub[x,y]: liczba(x wpisanych jako y),
- trans[x,y]: liczba(xy wpisanych jako yx).

(należy zwrócić uwagę, że usunięcie i wstawienie znaku jest uwarunkowane poprzednią literą, choć można je również uwarunkować następną literą).

Używając tych macierzy można obliczyć $P(x|w)$ za pomocą wzoru [11]:

$$P(x|w) = \begin{cases} \frac{\mathbf{del}[w_{i-1},w_i]}{\mathbf{count}[w_{i-1}w_i]} & \text{, dla usunięcia} \\ \frac{\mathbf{ins}[w_{i-1},x_i]}{\mathbf{count}[w_{i-1}]} & \text{, dla wstawienia} \\ \frac{\mathbf{sub}[x_i,w_i]}{\mathbf{count}[w_i]} & \text{, dla zamiany} \\ \frac{\mathbf{trans}[w_i,w_{i+1}]}{\mathbf{count}[w_iw_{i+1}]} & \text{, dla transpozycji} \end{cases}$$

gdzie w_i jest i-tą literą wyrazu w.

1.9 Model zaszumionego kanału

Do obliczania prawdopodobieństwa poprawnego wyrazu służy model zaszumionego kanału. Metoda polega na tym, że staramy się wybrać najlepszego kandydata z zestawu potencjalnych słów, które w wyniku błędu mogło zostać zamienione. Zbiór ten uzyskujemy przez wyszukanie słów w pewnej odległości edycyjnej. Można sobie wyobrazić, że poprawne słowo przechodzi przez pewien kanał (np. przez telefon) i zostaje zniekształcone przez szum. W tym przypadku kanałem zniekształcającym słowo będzie klawiatura, przez którą dany wyraz został źle napisany. Celem jest, dla wyrazu wyjściowego, zbudowanie listy kandydatów w odległości edycyjnej równej 1, następnie za pomocą modelu zaszumionego kanału stworzonego metodami probabilistycznymi, sprawdzenie, który z nich jest najbardziej prawdopodobny, przy założeniu wpisania tego błędnego wyrazu. Model zaszumionego kanału jest obrazowany przez wzór:

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}}\, P(w|x)$$

gdzie:

- x jest obserwacją błędnego wyrazu,
- w jest kandydatem należącym do słownika.

Załóżmy, że dany jest niepoprawny łańcuch znaków x i słownik V zawierający poprawne słowa. Szukamy takiego w, które należy do słownika V i najprawdopodobniej było słowem, które zostało zmienione w wyniku błędu. Chcemy znaleźć taki wyraz w, dla którego prawdopodobieństwo $P(w|x)$ jest największe. Na mocy twierdzenia Bayesa i po odrzuceniu mianownika otrzymujemy $P(x|w)P(w)$. Mianownik możemy odrzucić, ponieważ $P(x)$ jest tutaj stałe dla każdego przypadku w. Działania te są przedstawione poniżej:

$$\begin{aligned}\hat{w} &= \underset{w \in V}{\operatorname{argmax}} P(w|x) \\ &= \underset{w \in V}{\operatorname{argmax}} \frac{P(x|w)P(w)}{P(x)} \\ &= \underset{w \in V}{\operatorname{argmax}} P(x|w)P(w)\end{aligned}$$

gdzie:

- $P(x|w)$ - prawdopodobieństwo popełnienia błędu x podczas pisania słowa w uzyskane za pomocą modelu błędów,
- $P(w)$ - prawdopodobieństwo wystąpienia poprawnego wyrazu w w korpusie języka uzyskane za pomocą modelu języka.

1.10 Elektroniczny słownik języka polskiego

„Podstawowym zadaniem słownika języka polskiego jest dostarczenie wiedzy o znaczeniu ujętych w nim słów poprzez opisowe wyjaśnienie. W zależności od objętości słownik zawiera elementy fleksji, ortografii, etymologii oraz przykłady użycia. Słowa (hasła) w słowniku są uszeregowane alfabetycznie, zgodnie z przyjętym porządkiem liter”[1].

W dzisiejszych czasach, mając na uwadze wygodę użytkowania, dużą dostępność komputerów z dostępem do Internetu, coraz bardziej powszechne stają się

[1] https://pl.wikipedia.org/wiki/Słownik_języka_polskiego, dostęp 07.06.2017r.

słowniki języka polskiego w wersji elektronicznej. W projekcie jest wykorzystany elektroniczny słownik języka polskiego pobrany ze strony *sjp.pl* [28], ponieważ najbardziej odpowiadał on wymogom projektu. Zawiera on formy podstawowe słów wraz z odpowiednimi dla nich formami fleksyjnymi. W sumie jest to ponad 4 miliony wyrazów.

1.11 Korpus języka polskiego

Korpus języka jest zbiorem tekstów pisanych lub mówionych w postaci elektronicznej. Używany jest w badaniach lingwistycznych. Pokazuje, jak język jest stosowany w rzeczywistości[2]. Analizując korpus możemy zobaczyć w jakich kontekstach są używane dane słowa, dowiedzieć się o pojawieniu nowych, ale również określić jakie są trendy w pisowni. Korzystanie z korpusu pozwala na dokładniejsze zbadanie występowania słowa w różnych kontekstach.

W projekcie zastosowany jest polski korpus Wikipedii stworzony przez Instytut Podstaw Informatyki PAN. Zawiera on ok. 900 tysięcy artykułów, w których zawarte jest w sumie 169 milionów wyrazów[3].

Używając korpusu języka oraz odpowiednich algorytmów można utworzyć model języka, który z kolei pozwala oszacować prawdopodobieństwo wystąpienia wyrazu, albo całej sekwencji wyrazów w języku.

1.12 Korpus błędów

Korpus błędów, w przeciwieństwie do korpusu języka, to zbiór tekstów, które zawierają potencjalne pomyłki w danym języku[4]. Do każdego błędu podany jest poprawny wyraz lub sekwencja wyrazów. Są dwa główne sposoby na pozyskanie takiego korpusu:

- Ręczne anotowanie - zbieranie błędów języka przez lingwistów lub entuzjastów,
- Analiza historii edycji - błędy pozyskiwane są na podstawie badania historii poprawek w dokumencie.

[2]https://en.oxforddictionaries.com/explore/what-is-a-corpus, dostęp 07.06.2017r.
[3]http://clip.ipipan.waw.pl/PolishWikipediaCorpus, dostęp 07.06. 2017r.
[4]http://morfologik.blogspot.com/2006/05/korpus-bdw-jzykowych.html, dostęp 08.06.2017r.

W projekcie użyty jest korpus błędów językowych pozyskany na podstawie historii edycji polskiej Wikipedii (opisany w sekcji 2.2), ponieważ jest bardzo dobrym odzwierciedleniem przytrafiających się użytkownikom pomyłek.

Za pomocą korpusu błędów tworzy się model błędów, który pozwala obliczyć prawdopodobieństwo wygenerowania pomyłki przy pisaniu konkretnej sekwencji.

Rozdział 2

Metody korekty pisowni języka polskiego

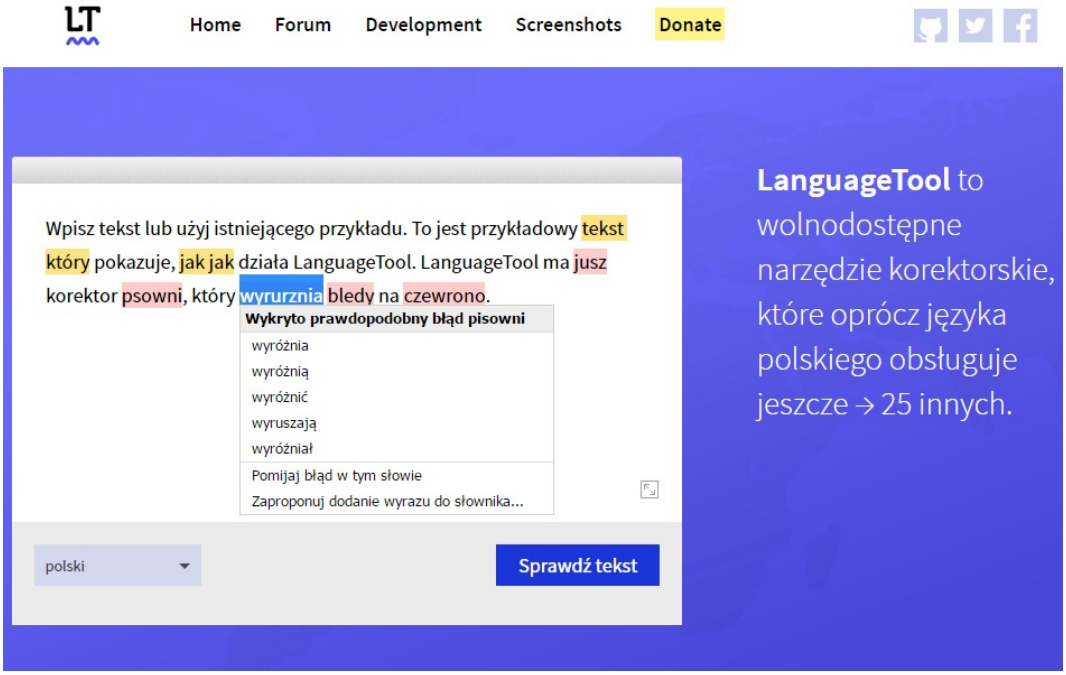
W tym rozdziale są przedstawione obecnie najbardziej zaawansowany projekt otwarto-źródłowy stworzony do korekcji pisowni *LanguageTool* oraz korpus błędów *PLEWi*. Program *LanguageTool*, w przeciwieństwie do projektu magisterskiego, używa reguł do wykrywania błędów, gdzie każda reguła przedstawia potencjalny błąd oraz jego poprawę. Za pomocą określonych wcześniej reguł program wnioskuje, jakich kandydatów do korekty zasugerować użytkownikowi. Korpus *PLEWi* jest źródłem naturalnie występujących błędów i poprawek w języku polskim.

2.1 LanguageTool

LanguageTool to darmowe, wolnodostępne narzędzie do sprawdzania poprawności pisowni, wykorzystujące reguły do wykrywania i poprawiania błędów. Obecnie obsługuje 26 języków, w tym język polski. Praca nad nim została zapoczątkowana przez niemieckiego studenta Daniela Nabera jako jego praca dyplomowa. Później do projektu dołączył Marcin Miłkowski - obecnie profesor nadzwyczajny w Zakładzie Logiki i Kognitywistyki w Instytucie Filozofii i Socjologii PAN. Za pracę nad rozwojem LanguageTool naukowcy zdobyli nagrodę „Gold Prize in Sun Community Innovation Program[1]".

[1] http://marcinmilkowski.pl/en/about-me1, dostęp 22.05.2017r.

Rysunek 2.1: Strona główna LanguageTool

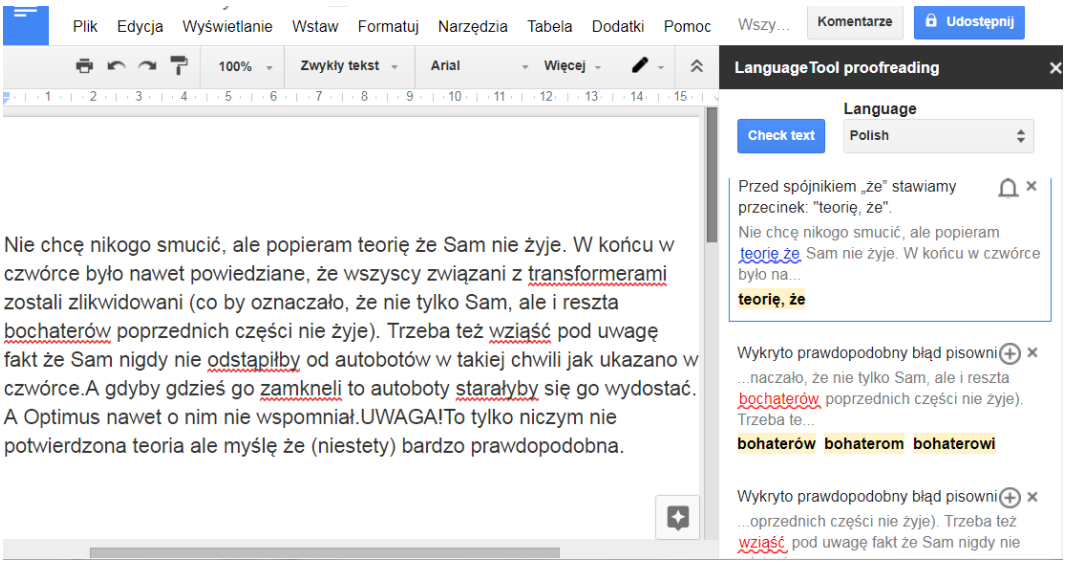


Źródło: https://languagetool.org/pl/, dostęp 25.05.2017r.

Rysunek 2.1 przedstawia stronię główną projektu, na której znajduje się sam korektor w postaci aplikacji internetowej. Jest on jednak głównie używany jako rozszerzenie (wtyczka) do innych programów m.in. *LibreOffice*, *OpenOffice*, *Firefox*, *Chrome*, *Google Docs*, *Thunderbird*, *Vim*, czy *Emacs*[2]. Może być również zintegrowany ze stronami internetowymi. Rysunek 2.2 przedstawia dodatek *LanguageTool* w dokumencie *Google*:

[2]https://languagetool.org/pl/, dostęp 05.06.2017r.

Rysunek 2.2: LanguageTool jako dodatek do dokumentów Google



Źródło: opracowanie własne.

LanguageTool rozróżnia cztery typy błędów: ortograficzne, gramatyczne, stylistyczne, typograficzne. Rozpoznaje on zarówno błędy należące, jak i nienależące do słownika języka polskiego[3]. Aby błędy zostały rozpoznane, wprowadzony tekst jest dzielony na zdania, po czym następuje tokenizacja każdego zdania. W kolejnym kroku, w wyniku lematyzacji, każdemu tokenowi przypisywana jest forma podstawowa, a następnie znacznik części mowy (Part-of-Speech tag) [16]. Tekst po analizie obrazuje Rysunek 2.3:

[3]Ibidem.

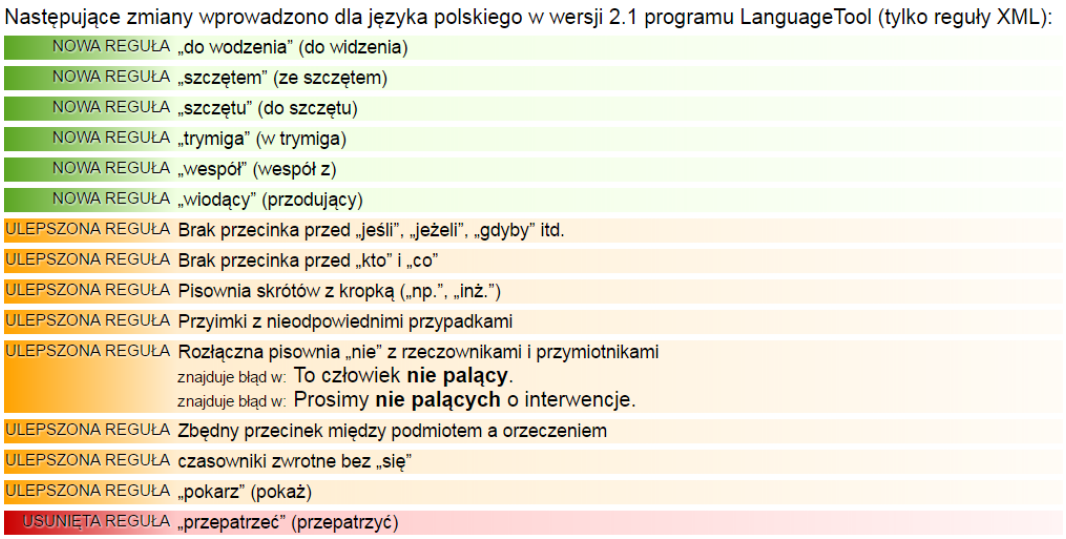
Rysunek 2.3: Analiza tekstu dla zdania w LanguageTool

Token	Lemma	Part-of-speech	Chunk
	-	SENT_START	
Nie	nie	qub	
chcę	chcieć	verb:fin:sg:pri:imperf:nonrefl	
nikogo	nikt	subst:sg:gen:m1	
smucic	-	-	
,	,	interp:comma	
ale	ale	conj comp:comma	
popieram	popierać	verb:fin:sg:pri:imperf:refl.nonrefl	
teorię	teoria	subst:sg:acc:f	
że	że	comp qub comp:comma	
Sam	Sam sam	subst:sg:nom:m1 adj:sg:acc:m3:pos adj:sg:nom.voc:m1.m2.m3:pos adv subst:sg:acc:m3 subst:sg:nom:m3	
nie	nie	qub	
żyje	żyć	verb:fin:sg:ter:imperf:nonrefl	
.	-	SENT_END PARA_END	

Źródło: https://community.languagetool.org/analysis/analyzeText, dostęp 07.06.2017r.

Po zakończeniu tych operacji przechodzi się do procesu wykrywania potencjalnych błędów. Jest on oparty o ustalone reguły. Każda z nich zawiera wzorzec opisujący potencjalny błąd, który można znaleźć w tekście, np. może to być sekwencja „**lód** przemówił", która zawiera błąd, gdyż powinno być prawdopodobnie „lud przemówił". Reguły są ustalane na podstawie popularnych błędów w danym języku. LanguageTool sprawdza wprowadzony tekst szukając dla niego odpowiedniego wzorca. Jeśli tekst pasujący do wzorca zostaje odnaleziony, to sygnalizowany jest jako błąd poprzez podświetlenie wyrazu lub sekwencji wyrazów.

Rysunek 2.4: Zmiany w regułach LanguageTool



Źródło: https://languagetool.org/changes/V_2_0_to_languagetool-2.1/changes_pl.html, dostęp 25.05.2017r.

Na Rysunku 2.4 przedstawione są zmiany w zbiorze reguł, jakie zostały dokonane w nowszej wersji programu, w celu jego usprawnienia. Oprócz usuwania oraz dodawania nowych, reguły można usprawniać tak, aby błędy były znajdowane w różnych kontekstach. Każdy użytkownik może pomóc w doskonaleniu LanguageTool tworząc swoje reguły na stronie projektu. Można to zrobić używając generatora, który tworzy je w formacie XML, a następnie wysłać do deweloperów. Przesyłane reguły są sprawdzane i jeśli okażą się przydatne, zostaną dodane do kolejnej wersji programu.

2.2 PLEWiC

PLEWiC (Polish Language Errors from Wikipedia Corpus) jest korpusem błędów języka polskiego, który powstał dzięki wykorzystaniu metody automatycznej ekstrakcji błędów językowych z historii edycji tekstu do artykułów polskiej wersji Wikipedii. Metoda ta została opisana w pracy Romana Grundkiewicza [17].

Problem został poruszony z powodu braku dużego, cyfrowego korpusu błędów języka polskiego, który by odzwierciedlał faktyczny stan popełnianych błędów przy pisaniu tekstu. Stworzenie takiego korpusu manualnie wiąże się z dużym kosztem i czasem. Można zredukować te dwa czynniki generując go automatycznie.

Do tego musi być jednak dostęp do cyfrowych historii edycji danych. Nabycie takich dokumentów stanowi jednak niemałe wyzwanie, ponieważ historia ich edycji zwykle nie jest przechowywana. Wyjątek stanowią jednak strony z rodziny *Wiki*, pliki w systemach kontroli wersji, czy *GoogleDocs* [17]. Jako źródło błędów została wybrana *Wikipedia*, głownie ze względu na jej rozmiar, dostępność i szerokie spektrum użytkowników.

2.2.1 Ekstrakcja błędów językowych

Za pomocą skryptu ściągnięto z Wikipedii każde dwie sąsiednie wersje artykułu. Edytowane fragmenty zostały wyekstrahowane z tekstu przy użyciu algorytmu LCS (Longest Common Subsequence) oraz oczyszczone ze znaczników Wikipedii. Następnie, na wszystkich fragmentach, zostało użyte narzędzie *PSI-Toolkit toolbox* [18] w celu segmentacji, tokenizacji i lematyzacji. Wszystkie edycje, które ograniczały się tylko do dodania lub usunięcia artykułu, zostały pominięte. Spośród uzyskanych w ten sposób zdań wybrano te, które spełniały następujące warunki [17]:

- zawierały między 4 a 60 tokenów
- różnica w długości była mniejsza niż 4 tokeny
- stosunek wyrazów do tokenów był większy niż 0.75
- liczba znaków niebędąca literami była mniejsza niż $\frac{1}{4}$ wszystkich znaków

Sentencje spełniające te warunki zostały ponownie poddane działaniu algorytmu LCS, tym razem na tokenach zamiast na liniach. W rezultacie zostały otrzymane przykłady edycji dla każdej pary zdań.

2.2.2 Rozpoznawanie błędów języka

W wyniku działań opisanych w sekcji 2.2.1, dla każdej pary zdań, otrzymano sekwencję edycji $((u_0, v_0), (u_1, v_1), ...)$, gdzie każda z nich (u, v) jest parą zawierającą wyrazy przed i po poprawie. Obrazuje to poniższy przykład[4]:

[4] http://www.staff.amu.edu.pl/~romang/slides/romang-plewic-pl.pdf, s. 8, dostęp 14.06.2017r.

$$u = \textit{Biologia to nauka o } \textbf{\textit{życie}} \textit{ i organizmach żywych.}$$
$$v = \textit{Biologia to nauka o } \textbf{\textit{życiu}} \textit{ i organizmach żywych.}$$
$$(u_i, v_i) = \textit{(życie, życiu)}$$

Zdania zawierające więcej niż cztery edycje zostały odrzucone, ponieważ zwykle wiążą się one ze zmianą lub rozszerzeniem zdania, a nie naprawą pojedynczych błędów.

W ten sposób znalezione i oznaczone błędy zostały zaklasyfikowane do określonej kategorii błędów:

- **Błędy proste** - to nadużywanie znaków interpunkcyjnych, zmiana wielkości liter oraz błędy związane z pisownią rozłączną i nierozłączną, gdy wyraz u oraz v różni jedynie znak spacji lub łącznik.
- **Błędy pisowni** - do ich wykrycia potrzebny jest słownik. Jeśli wyraz nie należy do słownika, jest uważany za literówkę. Rozróżniane są tutaj dwa typy: błędy z pominięciem znaków diakrytycznych oraz inne błędy ortograficzne.
- **Błędy gramatyczne** - wyrazy u oraz v należą do słownika. Dopiero użycie lematyzatora umożliwia ich analizę oraz klasyfikację jako specyficzne błędy gramatyczne: fleksyjne, semantyczne i składniowe. Te pierwsze występują, gdy słowa u i v mają te same lematy (formy podstawowe wyrazu), z kolei edycje składające się z wyrazów o różnych lematach będą prawdopodobnie korektą błędów składniowych (jeśli u i v należą do różnych klas gramatycznych) lub semantycznych (w innych przypadkach).
- **Błędy stylistyczne** - wyrazy w edycji należą do słownika. Jeśli są to edycje na skrótach i akronimach, przechwytywane są wraz z dodatkowymi informacjami dostarczonymi przez lematyzator. Inne korekcje błędów tego rodzaju są rozpoznawane przez tezaurus. Użycie go przed detekcją błędów gramatycznych zapobiegło klasyfikowaniu błędów stylistycznych z niewielką odległością edycyjną do złej kategorii [17].

2.2.3 Warunki akceptacji

Ponadto w każdym zdaniu dozwolona jest maksymalnie jedna nierozpoznana edycja, z wyjątkiem sytuacji, gdy pozostałe edycje zostały rozpoznane jako wszelkiego rodzaju błędy gramatyczne. Decyzja została oparta na obserwacji, że niektóre edycje (u_i, v_i) mogą być spowodowane wcześniejszymi edycjami (u_j, v_j) w

jednym zdaniu, głownie w przypadku zmian fleksyjnych. Na końcu odrzucone zostały zdania, w których jedyną zmianą było usunięcie kropki na końcu, dodanie dwukropka na końcu lub zmiana wielkości pierwszej litery [17].

2.2.4 Trudności

Główną trudnością, z którą można się spotkać, są zmiany dokonane przez wandali, ponieważ edycja zawartości artykułów w Wikipedii jest swobodnie dostępna. Problem ten został rozwiązany na trzech poziomach:

- uwzględnianie komentarzy przy korektach,
- pominięcie edycji zawierających wulgaryzmy i internetowych akronimów,
- usunięcie symetrycznych edycji, gdzie $(u_i, v_i) = (v_i, u_i)$ [17].

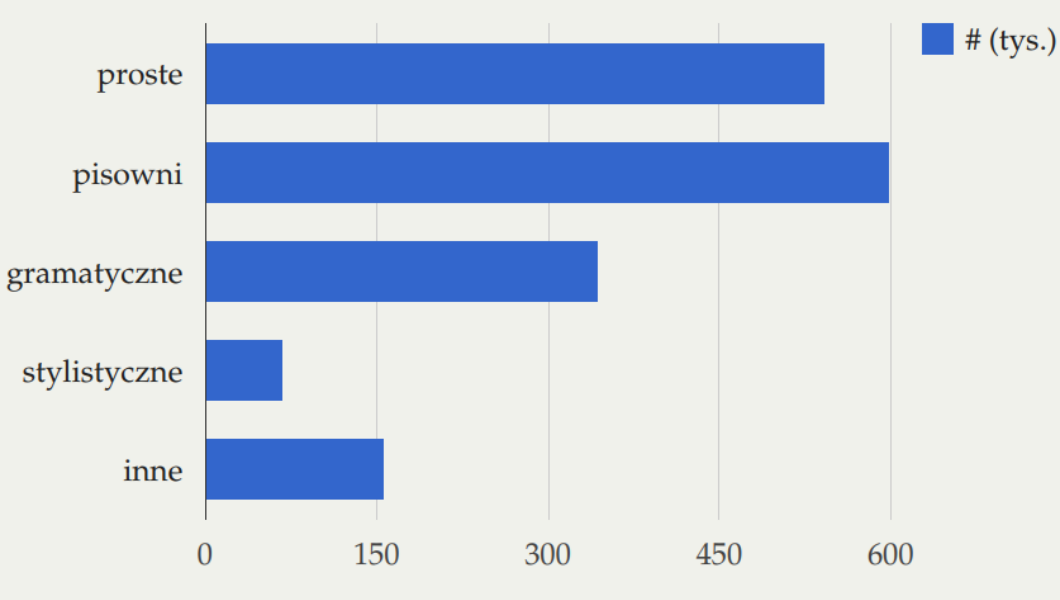
2.2.5 Korpus błędów

Przy użyciu powyższych metod i klasyfikacji do historii poprawek w polskiej Wikipedii został utworzony korpus PLEWi zawierający naturalnie występujące błędy w języku polskim. Cały zrzut z Wikipedii w postaci plików XML to ok. 330 GB danych, 1,747,083 stron z ok. 910,000 artykułów.

Liczba wyciągniętych fragmentów tekstu z co najmniej jednym potencjalnym błędem językowym wyniosła 1,532,275, w tym 1,303,806 (85.1%) poprawnie ułożonych zdań. Tylko 23% edycji pochodziło od niezarejestrowanych użytkowników, co znacznie zmniejsza ryzyko wandalizmu.

Całkowita liczba zebranych edycji wyniosła 1,713,835, wliczając w to 157,043 (10.9%) edycji oznaczonych jako nierozpoznane (słowa nieistniejące w słowniku, będące w odległości edycyjnej mniejszej niż 4) [17]. Biorąc pod uwagę kategorie błędów, rozkład przedstawiony jest na Rysunku 2.5:

Rysunek 2.5: Rozkład z podziałem na kategorie.



Źródło: http://www.staff.amu.edu.pl/~romang/slides/romang-plewic-pl.pdf, dostęp 15.06.2017r.

Natomiast szczegółowy rozkład wszystkich typów błędów przedstawiono na Tablicy 2.1:

Tablica 2.1: Częstotliwość występowania błędów w korpusie PLEWi.

Kategoria	**Typ błędu**	**Liczba (tys.)**
proste	interpunkcyjne	308,8
	wielkość liter	220,5
	pisownia łączna-rozłączna	13,7
pisowni	diakrytyczne (kontekst)	241,8 (39,5)
	słownikowe	356,8
gramatyczne	fleksyjne (czas/liczba)	164,7 (43,3)
	semantyczne (aspekt/st.)	64,6 (13,8)
	składniowe	19,4
	zaimki, przyimki itp.	94,6
stylistyczne	synonimy	29,6
	skróty (rok/wiek)	38,8 (21,4)
inne	nierozpoznane	157,0

Źródło: http://www.staff.amu.edu.pl/~romang/slides/romang-plewic-pl.pdf, dostęp 15.06.2017r.

2.2.6 Ewaluacja

Ocena skuteczności metod ekstrakcji błędów językowych i samego korpusu PLEWi została dokonana manualnie. Z każdej kategorii wybrano losowo 200 fragmentów. Z kolei dla każdej pierwszej edycji w każdym wybranym fragmencie sprawdzono, czy poprawnie został nadany znacznik błędu. W ten sposób, po podzieleniu przez siebie wyników została sprawdzona precyzja wskazań edycji. Wyniki przedstawia Tablica 2.2:

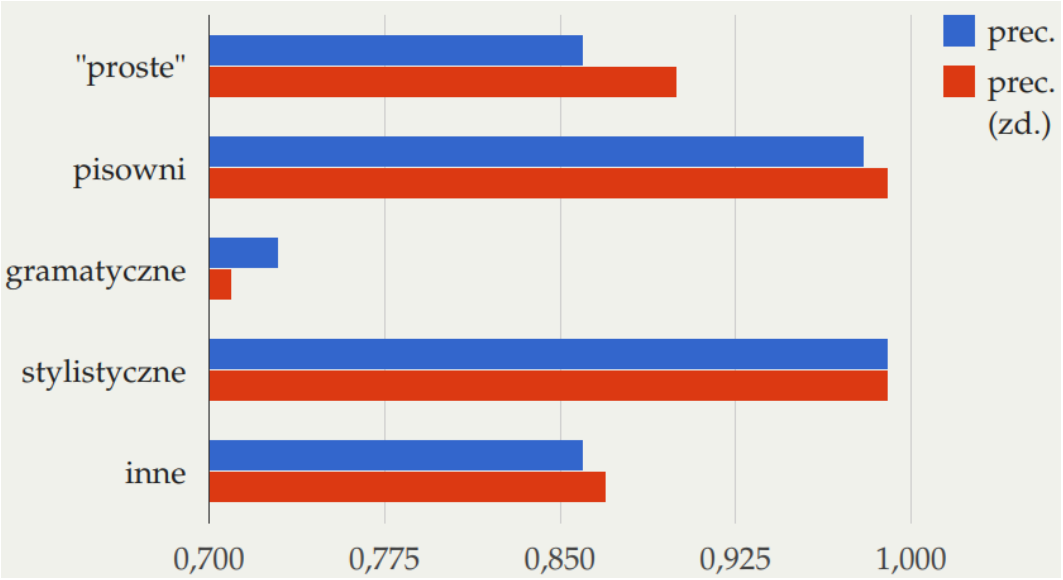
Tablica 2.2: Precyzja wskazań na próbie 200 losowych fragmentów.

Kategoria	**Liczba fragmentów**	**Precyzja**	**Liczba zdań**	**Precyzja**
proste	200	0.86	146	0.90
pisowni	200	0.98	173	0.99
gramatyczne	200	0.73	170	0.71
stylistyczne	200	0.99	169	0.99
inne	200	0.86	164	0.87

Źródło: Grundkiewicz R., 2013, Automatic Extraction of Polish Language Errors from Text Edition History, s. 7.

Dokładność metod została zmierzona dla wszystkich fragmentów oraz, osobno, dla tych, które są poprawnymi zdaniami. Powyższe wyniki ewaluacji przedstawiono również na Rysunku 2.6:

Rysunek 2.6: Precyzja kategoryzacji.



Źródło: http://www.staff.amu.edu.pl/~romang/slides/romang-plewic-pl.pdf, dostęp 15.06.2017r.

Procentowy skład edycji, które nie zostały bezpośrednio rozpoznane wygląda następująco [17]:

- 56% - nazwy własne,
- 18% - błędy fleksyjne,
- 16% - błędy pisowni,
- 6% - stylistyczne,
- 5% - błędne przypadki, gdy błędny wyraz został zamieniony na inny błędny wyraz,
- 4% - wyrazy obce.

Suma procent wszystkich błędów wynosi 105. Jest to najprawdopodobniej błąd spowodowany zaokrąglaniem poszczególnych wyników. Średnia precyzja kategoryzacji błędów wyniosła 88%.

Rozdział 3

Opis eksperymentu

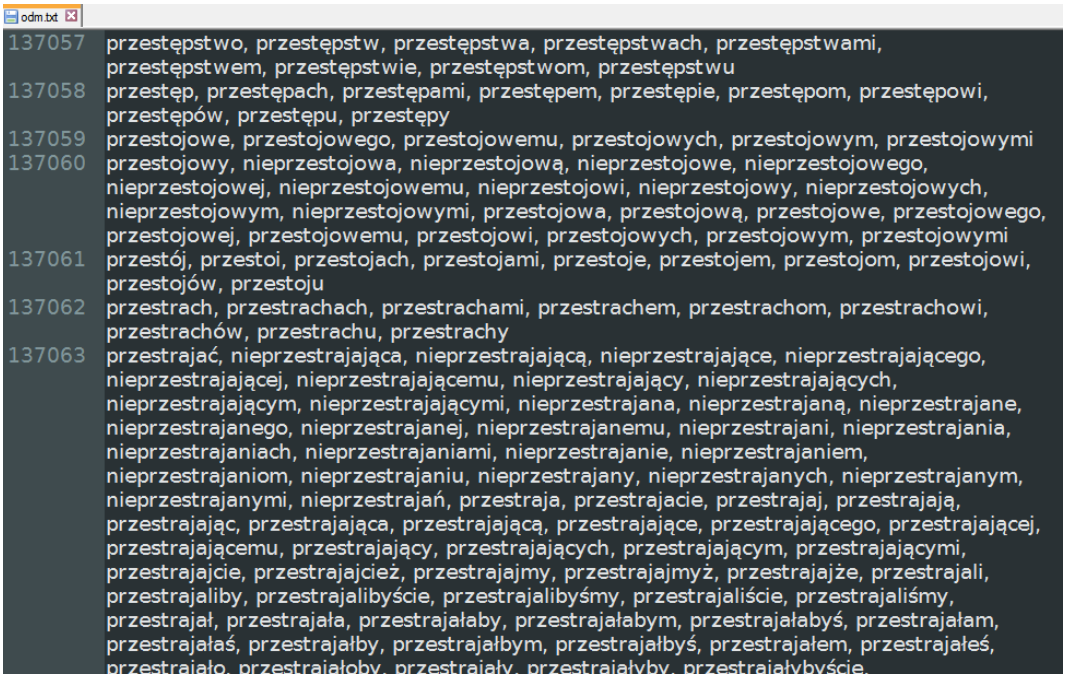
Rozdział przedstawia opis projektu magisterskiego oraz wykorzystanych w nim technik. Zadaniem jest stworzenie korektora pisowni języka polskiego wykorzystującego słownik języka polskiego, korpus języka polskiego oraz korpus błędów języka polskiego. Ten pierwszy służy do sprawdzania, czy wyraz w zdaniu należy do słownika oraz wyszukiwania potencjalnych kandydatów z użyciem algorytmu Damerau-Levenshteina. Dane zawarte w korpusie języka zawierają artykuły z Wikipedii. W oparciu o te dane zbudowany został probabilistyczny (unigramowy) model języka, obrazujący, jak często dane wyrazy są używane w języku polskim. Korpus błędów PLEWi stworzony i opisany przez Romana Grundkiewicza [17] dostarcza historię edycji artykułów z Wikipedii. Zawarte w nim pary (błąd, poprawka) zawierają informację o najczęściej popełnianych błędach na klawiaturze. Na podstawie korpusu PLEWi stworzono model błędów. Mając opracowane oba modele probabilistyczne w postaci plików JSON, za pomocą modelu zaszumionego kanału, obliczane jest prawdopodobieństwo dla każdego wygenerowanego kandydata i wybierany jest ten, dla którego jest ono największe.

3.1 Opracowanie elektronicznego słownika języka polskiego

W projekcie wykorzystany został elektroniczny słownik języka polskiego pobrany ze strony *sjp.pl* [28] w postaci pliku *txt*. Zawiera on formy podstawowe słów oraz ich formy fleksyjne. Wszystkich form razem jest ponad 4 miliony. Każdy wiersz w pliku jest przeznaczony dla jednego leksemu: formy podstawowej i wszystkich form fleksyjnych, które są przedzielone przecinkami. Strukturę

słownika obrazuje Rysunek 3.1:

Rysunek 3.1: Struktura użytego w projekcie słownika języka polskiego.



Źródło: opracowanie własne.

Słownik jest wczytywany od razu po włączeniu programu. W celach optymalizacyjnych przechowywany jest jako tablica list $Words[]$, gdzie każdy element $Words[i]$ reprezentuje listę przechowującą wyrazy i-tej długości posortowane alfabetycznie. W ten sposób czas sprawdzania, czy dane słowo należy do słownika, zmniejszył się diametralnie (w zależności od ilości wyrazów tej samej długości). Rozwiązanie to pozwoliło również zmniejszyć zakres poszukiwań potencjalnych kandydatów słów do długości z przedziału $\langle i-1, i+1\rangle$, ponieważ kandydaci w odległości edycyjnej równej 1 nie mogą się różnić większą liczbą znaków. Oznacza to, że algorytm Damerau-Levenshteina również jest wykonywany tylko dla wyrazów z tego przedziału, dzięki czemu wszystkie metody korzystające ze słownika wykonują się nawet kilkukrotnie szybciej. Rozwiązanie tego problemu miało krytyczne znaczenie dla jakości działania programu, ponieważ obliczenia te są prowadzone w czasie rzeczywistym.

3.2 Opracowanie modelu języka

Do opracowania modelu języka posłużył polski korpus Wikipedii, który został zebrany przez Instytut Podstaw Informatyki PAN [29]. Zawarta jest w nim treść 900 tysięcy artykułów, w których łącznie jest 169 milionów wyrazów.

Model języka został utworzony metodą unigramów za pomocą skryptu napisanego w języku PHP. Skrypt ten liczył wystąpienia każdego wyrazu i dzielił je przez liczbę wszystkich wyrazów w korpusie. Działanie to przedstawia wzór:

$$\underset{w \in V}{P(w)} = \frac{count(w)}{N}$$

gdzie:

- w jest słowem należącym do słownika V,
- $count(w)$ jest liczbą wystąpień w w korpusie,
- N jest liczbą wszystkich tokenów w korpusie (moga się powtarzać),

W ten sposób zostały wygenerowane pliki w formacie JSON, zawierające rozkład prawdopodobieństwa metodą unigramów. Struktura przykładowego pliku zaprezentowana jest na Rysunku 3.2:

Rysunek 3.2: Struktura korpusu językowego w jednym z plików JSON.

Źródło: opracowanie własne.

Główne obiekty to wyekstrahowane z korpusu wyrazy. Każdy z nich zawiera dwie właściwości: liczbę wystąpień w korpusie oraz wartość probabilistyczną $P(w)$.

Skrypt tworzący model języka został stworzony tak, że można łatwo dodać jeszcze inne korpusy, co z kolei daje możliwość stworzenia w przyszłości bardziej uniwersalnego modelu języka, korzystającego z różnorodnych tekstów, nie tylko o charakterze encyklopedycznym.

3.3 Opracowanie modelu błędów

Model błędów został opracowany przy użyciu korpusu błędów PLEWi [30]. Został on stworzony na podstawie historii edycji z polskiej Wikipedii i zawiera 1,713,835 przykładów błędów.

Model błędów został wygenerowany za pomocą skryptu napisanego w języku PHP. Sprawdza on każdą parę (błąd, poprawka) z korpusu PLEWi, która różni się jedną edycją Damerau-Levenshteina (sekcja 3.4) i kategoryzuje błąd następująco:

1) **Usunięcie**, gdy długość błędu jest mniejsza niż długość poprawki. Dwa wyrazy porównuje się ze sobą sprawdzając, który znak został wstawiony w

poprawce. Bierze się również znak poprzedni, tak więc mając parę:

błąd: *histori*

poprawka: *historii*

wartość **del**$[i, i]$ (ii wpisano jako i) jest zwiększana o jeden.

2) **Wstawienie**, gdy długość błędu jest większa niż długość poprawki. Wyrazy porównuje się sprawdzając, który znak został usunięty przy poprawianiu. Tutaj również bierze się poprzedni znak. Dla pary:

   błąd: *mamną*

   poprawka: *mamą*

   wartość **ins**$[m, n]$ (m wpisano jako mn) zwiększana jest o jeden.

3) **Zamiana/Transpozycja**, gdy obie długości są takie same. Nie można jednak od razu stwierdzić, który przypadek jest badany, bez dokładnego sprawdzenia pary wyrazów. W każdej z nich porównywane są więc poszczególne litery błędnego wyrazu oraz jego poprawki w poszukiwaniu różnic. Są tutaj dwie możliwości:

   - **Zamiana**, gdy różnią się jednym znakiem. Sprawdzana jest litera, która została zamieniona przy korekcji. Dla pary:

     błąd: *odmuenny*

     poprawka: *odmienny*

     wartość **sub**$[i, u]$ (i wpisano jako u) zwiększana jest o jeden.

   - **Transpozycja**, gdy któreś z dwóch kolejnych znaków występują w odwróconej kolejności w poprawce. Dla pary:

     błąd: *jezscze*

     poprawka: *jeszcze*

     wartość **trans**$[s, z]$ (sz wpisano jako zs) ziększana jest o jeden.

Na koniec szacuje się prawdopodobieństwo wystąpienia każdej literówki. Liczone jest wystąpienie konkretnego błędu, które dzieli się przez wystąpienia sekwencji liter wg wzoru:

- $\frac{\mathbf{del}[w_{i-1},w_i]}{\mathbf{count}[w_{i-1}w_i]}$, w przypadku usunięcia
- $\frac{\mathbf{ins}[w_{i-1},x_i]}{\mathbf{count}[w_{i-1}]}$, dla wstawienia
- $\frac{\mathbf{sub}[x_i,w_i]}{\mathbf{count}[w_i]}$, dla zamiany
- $\frac{\mathbf{trans}[w_i,w_{i+1}]}{\mathbf{count}[w_iw_{i+1}]}$, dla transpozycji

W ten sposób zostały wygenerowane 4 pliki w formacie JSON. Każdy z nich zawiera jednego rodzaju błędy oraz ich konkretne poprawki wraz z wartościami probabilistycznymi *P(x/w)*.

Rysunek 3.3: Struktura przykładowego pliku JSON (tutaj z zamianą znaków).

Źródło: opracowanie własne.

Rysunek 3.3 obrazuje strukturę przykładowego pliku z pomyłkami. Głównymi obiektami są w nim błędy. W każdym z nich znajdują się poprawki, a po dwukropku oszacowane prawdopodobieństwo wpisania złej litery pod warunkiem poprawki. Jeśli w programie wystąpi błąd, którego nie ma w pliku, jego prawdopodobieństwu przypisuje się wartość zerową.

Wszystkie cztery pliki tworzące model błędów naturalnie występujących w języku polskim są wczytywane przy starcie programu.

3.4 Opracowanie algorytmu Damerau-Levenshteina oraz kategoryzacja błędów

Kwestia optymalizacji algorytmu Damerau-Levenshteina ma ogromne znaczenie dla jakości działania programu, ponieważ wyszukiwanie kandydatów odbywa się w czasie działania aplikacji, po wpisaniu błędnego słowa przez użytkownika. Implementacje dostępne w Internecie nie zawierają jednego rodzaju edycji - transpozycji znaków. Koniecznym więc jest stworzenie jego własnej implementacji.

Stworzony algorytm Damerau-Levenshteina ma zwracać tylko tych kandydatów ze słownika, którzy są w odległości edycyjnej równej 1 od błędnie wpisanego słowa. Porównuje on zawsze parę *(błąd, kandydat)*. Polega to na sprawdzaniu każdego znaku z wyrazu w poszukiwaniu różnic. Specjalny licznik przechowuje liczbę różnic między tymi wyrazami. Gdy algorytm natrafi na usunięcie, wstawienie, zamianę lub transpozycję znaków (opisane w sekcji 3.3), licznik zwiększany jest o 1. Jeśli natomiast wartość licznika będzie większa niż 1, wykonywanie algorytmu dla tego wyrazu zostaje przerwane i przechodzi on do kolejnego kandydata. Nadana jest mu wartość ujemna, przez co nie jest on brany pod uwagę przy tworzeniu listy kandydatów do poprawy.

Algorytm Damerau-Levenshteina wykonuje się na wyrazach długości z przedziału $\langle i-1, i+1\rangle$, gdzie i jest ich długością. Przeszukiwane są zatem tylko te listy słownika $Words[]$, które zawierają słowa długości z tego przedziału, co zwiększa jego efektywność.

3.5 Opracowanie modelu zaszumionego kanału

Mając dane z modelu błędów oraz modelu języka przechodzi się do tworzenia modelu zaszumionego kanału. Dla każdego kandydata do poprawy, wygenerowanego przy użyciu algorytmu Damerau-Levenshteina, obliczane jest prawdopodobieństwo końcowe. Idealną strukturą do tego jest słownik, który zawiera pary *<klucz, wartość>*. Kluczem jest kandydat, a wartością prawdopodobieństwo końcowe obliczane przez przemnożenie odpowiednich liczb dla kandydata z modelu języka i modelu błędów. Działania te przedstawia wzór (opisany w sekcji 1.9):

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}}\, P(x|w)P(w)$$

Na końcu wybierany jest kandydat zawierający największe prawdopodobieństwo końcowe. Model zaszumionego kanału jest tworzony w czasie działania programu.

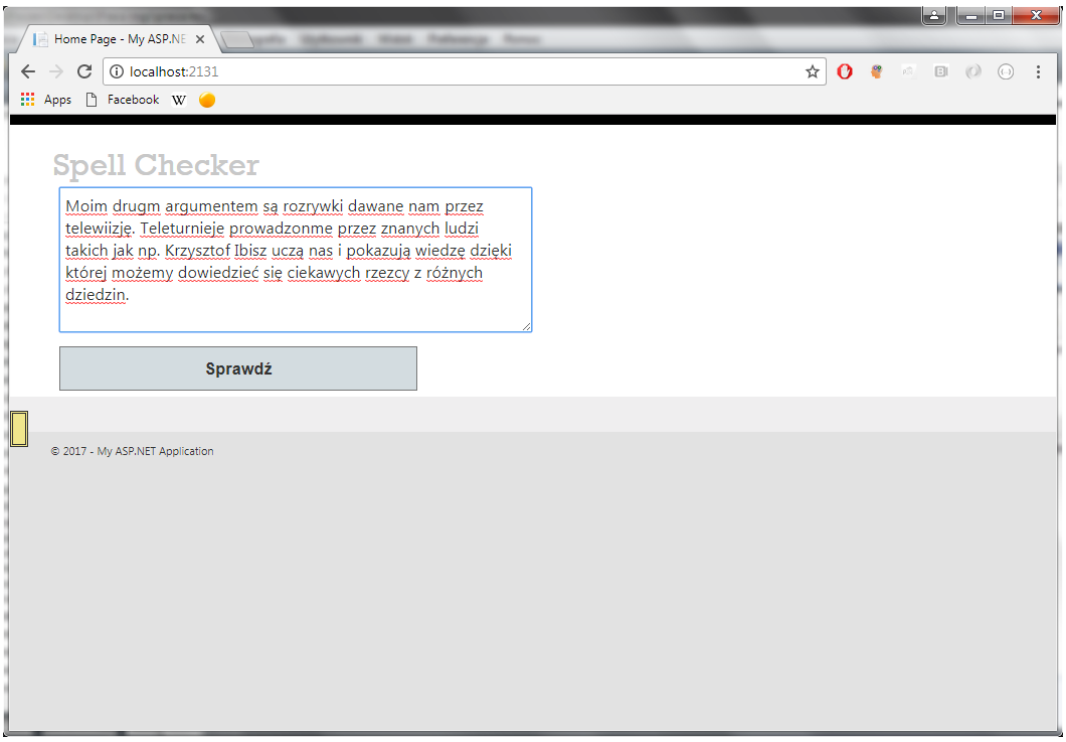
3.6 Opis działania korektora

Działanie korektora jest bardzo proste. Główny interfejs oraz obsługa narzędzia wzorowane są na *TumaczuGoogle* - darmowym serwisie internetowym firmy *Google* umożliwiającym tłumaczenie tekstów w wielu językach. Poprawiony tekst wyświetla się obok tekstu wprowadzonego. Projekt jest głównie wykonany w języku *C#*, przy użyciu technologii *ASP.NET Web Forms Application*, w środowisku *Visual Studio 2012* firmy *Microsoft*. Do dynamicznego wyświetlania oraz zamiany kandydatów do poprawy użyty jest jednak język *JavaScript* wraz z biblioteką *jQuery*, która znacznie ułatwia tego typu operacje. Gotowy projekt został upubliczniony w serwisie *GitHub* [31].

3.6.1 Wprowadzanie tekstu

Użytkownik wprowadza tekst przedzielony kropkami w przeznaczone do tego pole tekstowe. Po naniesieniu tekstu, który chce się poprawić, należy nacisnąć przycisk *Sprawdź*. Po tej akcji rozpoczyna się cały proces korekcji. Interfejs aplikacji przedstawia Rysunek 3.4:

Rysunek 3.4: Interfejs użytkownika z wprowadzonym tekstem.



Źródło: opracowanie własne.

3.6.2 Segmentacja i tokenizacja tekstu

Segmentacja i tokenizacja to pierwszy etap przetwarzania tekstu. Po naciśnięciu przez użytkownika przycisku *Sprawdź*, cały ciąg znaków znajdujący się w polu tekstowym jest wczytany przez program. Ze względu na wyrazy rozpoczynające zdanie, które są pisane z wielkiej litery (często nie występują one w takiej formie w słowniku), podany tekst jest sprowadzany do małych liter. Całość segmentowana jest na zdania po każdej napotkanej kropce, znaku zapytania lub wykrzykniku. Wspomniane znaki są uważane za znaki końca zdania. Natomiast po procesie segmentacji następuje podział tekstu na części elementarne, czyli na ciąg pojedynczych tokenów. Jako separator przyjęty został znak spacji. Opisane procesy obrazuje przykład:

Mam na imię Andrzej. Lubię sport, podróże i dobre piwo.

Wpisując takie zdanie tekst zostanie podzielony następująco:

[mam] [na] [imię] [andrzej.] [lubię] [sport,] [podróże] [i] [dobre] [piwo.]

Jak widać na przykładzie, nazwy własne, imiona czy nazwiska, są również sprowadzane do małych liter. Później jednak, podczas sprawdzania tokenów, wielka litera zostanie im przywrócona. W kolejnym kroku znaki interpunkcyjne występujące w tokenach *[andrzej.]*, *[sport,]* oraz *[piwo.]* zostaną usunięte. Po sprawdzeniu każdego z nich w poszukiwaniu ewentualnych błędów, znaki interpunkcyjne ponownie zostają umieszczone przy wyrazach, które je zawierały na wejściu.

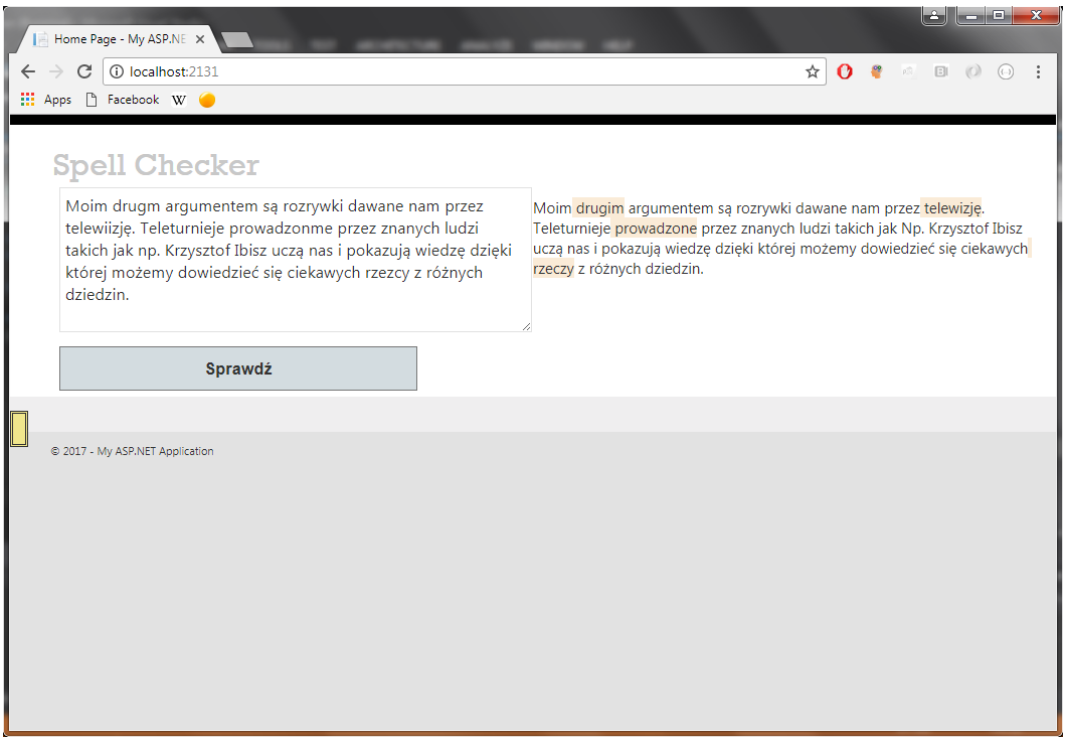
3.6.3 Rozpoznawanie błędów

Mając już podzielony tekst na tokeny, program przechodzi do korekcji pisowni. Do rozpoznawania błędów został wykorzystany elektroniczny słownik języka polskiego. Plik *txt* z wyrazami oraz ich odmianami został wczytany przy starcie programu do tablicy list *Words[i]*, gdzie w każdej takiej liście przechowywane są tylko wyrazy i-tej długości (opisane w sekcji 3.1).

Dla każdego tokenu wykonywana jest metoda *Contains()*. Zwraca ona wartość *true*, gdy długość podanego tokenu wynosi 0, albo token należy do *Words[i]* (i to długość sprawdzanego wyrazu).

Należy pamiętać, że niektóre wyrazy występują w słowniku tylko z wielką pierwszą literą (np. nazwy własne czy imiona), a wszystkie tokeny zostały sprowadzone do małych liter. Przykładowo, mając token *[andrzej]*, najpierw sprawdza się, czy występuje on w słowniku (dokładniej w *Words[7]*, ponieważ długość wyrazu wynosi 7 znaków). W takiej postaci nie występuje, więc sprawdza się czy słownik zawiera ten wyraz z wielką literą na początku *[Andrzej]*. Taki wyraz już w słowniku występuje, zatem przypisuje się go do tekstu wyjściowego. Program sprawdza również, jeśli wyraz nie spełnia poprzednich warunków, czy słownik zawiera ten wyraz złożony z samych wielkich liter (przydatne szczególnie przy sprawdzaniu skrótów). Sprawdzając jednak token *[czewrony]* nie zostaną spełnione trzy poprzednie warunki i w tym momencie jest on rozpoznany jako błąd. Wtedy program przechodzi do kategoryzacji błędu. Tekst wyjściowy ma podświetlone na czerwono miejsca, w których znajdowały się wyrazy zakwalifikowane jako błędy. Przedstawia to Rysunek 3.5:

Rysunek 3.5: Tekst wyjściowy (z prawej) z podświetlonymi błędami.



Źródło: opracowanie własne.

3.6.4 Kategoryzacja błędu i wyszukiwanie kandydatów do poprawy

Procesy kategoryzacji błędu oraz wyszukiwania kandydatów odbywają się w tym samym czasie, za pomocą jednej metody. Korzysta ona z algorytmu Damerau-Levenshteina (sekcja 3.4) oraz algorytmu kategoryzacji błędu (sekcja 3.3). Metoda zwraca wartość probabilistyczną dla kandydatów znajdujących się w odległości edycyjnej równej 1, a dla innych zwraca wartość ujemną. W pierwszym przypadku zwracana jest liczba $P(x|w)$ z jednego z czterech plików JSON, w zależności od nadanej kategorii błędu.

Kategoria nadawana jest w ten sam sposób, który użyto przy opracowaniu modelu błędów (sekcja 3.3). Następnie, podczas wyszukiwania listy kandydatów, dodawani są tylko ci, którym wcześniejsza metoda zwróciła wartość większą bądź równą 0 (w ten sposób wykluczeni są wszyscy kandydaci z większą odległością edycyjną). Do tej listy dodawany jest kandydat oraz jego prawdopodobieństwo wystąpienia $P(x|w)$. Służy do tego słownik *<klucz, wartość>*, gdzie kluczem jest wyraz, a wartością oszacowanie wystąpienia konkretnego błędu pod warunkiem

jego poprawki w odpowiednim pliku JSON.

3.6.5 Obliczanie prawdopodobieństwa

Mając listę kandydatów do poprawy oraz oszacowane dla nich prawdopodobieństwa:

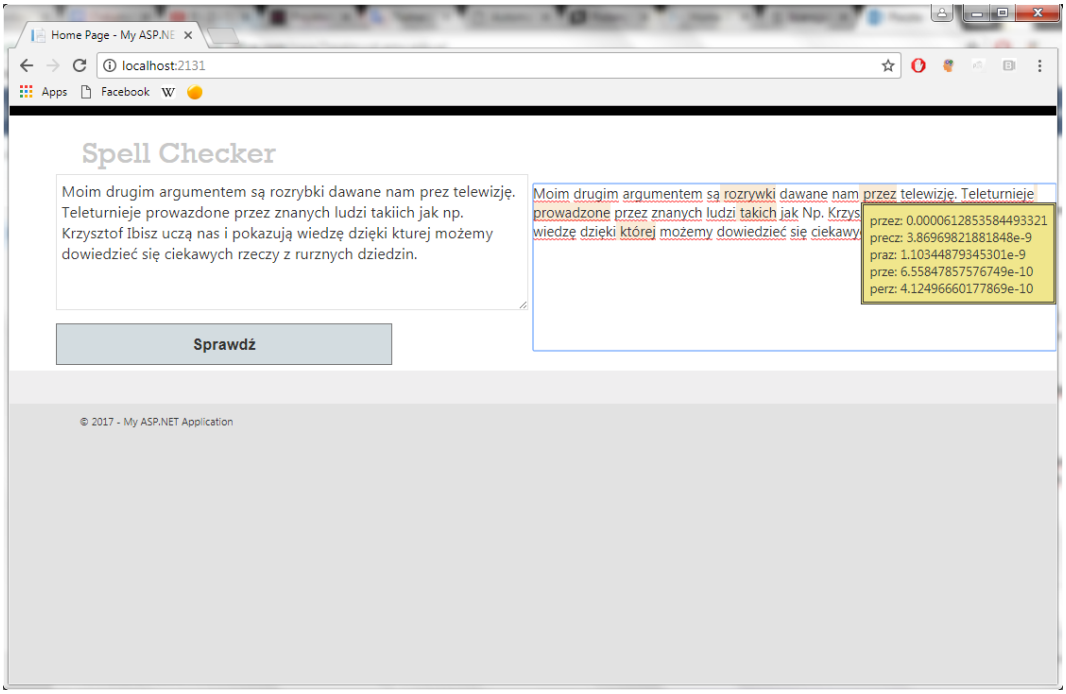
- $P(x|w)$ z modelu błędów
- $P(w)$ z modelu języka

szacowane jest prawdopodobieństwo końcowe dla każdego z nich mnożąc wyżej wymienione wartości. Jeśli w programie wystąpi kandydat, którego nie ma w korpusie języka, z modelu językowego zwracana jest wartość $P(w) = 0$. Jeśli natomiast w pliku nie ma danej poprawki (nie wystąpiła w korpusie błędów), zwracana jest dla niej wartość zerowa.

3.6.6 Wybór kandydata

Do każdego kandydata z listy przypisana jest końcowa wartość probabilistyczna. Lista ta jest zmniejszana do maksymalnie 5 kandydatów z największym prawdopodobieństwem, posortowanych malejąco względem tej wartości. Kandydat z pierwszego miejsca jest wstawiany automatycznie do tekstu wyjściowego, zgodnie z modelem zaszumionego kanału. Użytkownik ma jednak możliwość jego zmiany. Stan ten ukazuje Rysunek 3.6:

Rysunek 3.6: Tekst wyjściowy (z prawej) z podświetlonymi błędami oraz rozwiniętą listą kandydatów do poprawy.



Źródło: opracowanie własne.

Po kliknięciu myszką na błąd (podświetlony na czerwono) wyświetla się lista dostępnych kandydatów. Kliknięcie myszką na dowolnego z nich skutkuje jego zamianą w tekście wyjściowym.

Jeżeli jednak żaden kandydat do poprawy nie zostanie znaleziony (słownik z kandydatami jest pusty), program zostawia wyraz taki, jaki był na wejściu.

Rozdział 4

Ewaluacja i wnioski

Rozdział przedstawia cały proces ewaluacji projektu magisterskiego począwszy od sposobu ewaluacji, poprzez uzyskane wyniki, a skończywszy na wnioskach. W sposobie ewaluacji ujęte są kryteria ewaluacyjne oraz wybrane miary jakości użyte do oceny efektywności metod zaimplementowanych w projekcie. Wyniki przedstawiają aktualną precyzję oraz pokrycie wskazań i popraw błędów w programie. Na końcu przedstawione są wnioski wyciągnięte na podstawie uzyskanych wcześniej wyników.

4.1 Sposób ewaluacji

Aby ocenić efektywność przedstawionych metod użytych do wykrywania oraz poprawy błędów, jak i samego projektu magisterskiego, zostało manualnie sprawdzonych 100 zdań zawierających w sumie 160 przykładów błędów (19 usunięć, 17 wstawień, 113 zamian, 11 transpozycji) w odległości edycyjnej równej 1. Znajdowało się w nich również 39 błędów (24% wszystkich błędów) w większej odległości edycyjnej, które nie były brane pod uwagę ze względu na założenia projektowe. Przyjęte zostały dwa sposoby ewaluacji na podstawie dwóch macierzy błędów, przedstawiających się następująco:

I. Tablica, w której brany pod uwagę jest tylko pierwszy element z listy kandydatów (automatyczna poprawa wyrazu przez program):

   - **True Positive** (TP) - błędny wyraz z tekstu, dla którego program automatycznie wybrał dobrego kandydata,

- **False Positive** (FP) - poprawny wyraz z tekstu, który został automatycznie poprawiony na błędny,
- **True Negative** (TN) - poprawny wyraz z tekstu, dla którego program (słusznie) nie zareagował,
- **False Negative** (FN) - błędny wyraz z tekstu, dla którego program (niesłusznie) nie zareagował lub automatycznie wybrał złego kandydata.

II. Tablica, w której branych pod uwagę jest pięciu pierwszych kandydatów (oczywiście w tym wypadku otrzymane wyniki będą lepsze):

- **True Positive** (TP) - błędny wyraz z tekstu, dla którego znaleziony został poprawny wyraz w pierwszych pięciu podpowiedziach,
- **False Positive** (FP) - poprawny wyraz z tekstu, który został automatycznie poprawiony na błędny i wyświetlone zostały dla niego podpowiedzi,
- **True Negative** (TN) - poprawny wyraz z tekstu, dla którego program (słusznie) nie zareagował,
- **False Negative** (FN) - błędny wyraz z tekstu, dla którego program (niesłusznie) nie zareagował lub w pierwszych pięciu podpowiedziach nie znalazł się poprawny kandydat.

Dla każdego sposobu celem jest zaprezentowanie wybranych miar jakości pokazujących wartość projektu. Oto wybrane z nich [20]:

- $precyzja = \frac{TP}{TP+FP}$,
  określa, jaka część wyników wskazanych przez klasyfikator jako dodatnie jest faktycznie dodatnia.
- $pokrycie = \frac{TP}{TP+FN}$,
  określa, jaką część dodatnich wyników wykrył klasyfikator.
- $F - measure = \frac{2 * precyzja * pokrycie}{(precyzja + pokrycie)}$,
  jest średnią harmoniczną z precyzji i pokrycia.

Obydwie ewaluacje zostaną przeprowadzone dla projektu magisterskiego *SpellChecker* oraz dla narzędzia korektorskiego *LanguageTool* w celu porównania ich wyników.

4.2 Wyniki

Wyniki klasyfikacji błędów są zaprezentowane z podziałem na programy - osobno dla projektu magisterskiego *SpellChecker* oraz dla *LanguageTool*. Przedstawione są rezultaty każdego z programów przy dwóch rodzajach ewaluacji. Na końcu ukazane jest porównanie jakości programów dla każdego sposobu ewaluacji przy użyciu przyjętych miar (sekcja 4.1). Wszystkie rezultaty zaprezentowane są w formie tabel, a ponadto porównanie jakości programów jest również przedstawione na wykresach.

4.2.1 SpellChecker

Wyniki uzyskane przez SpellChecker przedstawiają się następująco:

Sposób I

Tablica 4.1: Klasyfikacja wyrazów dla programu SpellChecker (Sposób I).

Kategoria	Typ błędu	Liczba błędów	Wszystkie
TP	usunięcie	15	115
	wstawienie	14	
	zamiana	76	
	transpozycja	10	
FP	usunięcie	4	12
	wstawienie	1	
	zamiana	7	
	transpozycja	0	
TN	brak	689	689
FN	usunięcie	4	45
	wstawienie	3	
	zamiana	37	
	transpozycja	1	

Źródło: opracowanie własne.

Sposób II

Tablica 4.2: Klasyfikacja wyrazów dla programu SpellChecker (Sposób II).

Kategoria	**Typ błędu**	**Liczba błędów**	**Wszystkie**
TP	usunięcie	17	135
	wstawienie	17	
	zamiana	90	
	transpozycja	11	
FP	usunięcie	4	12
	wstawienie	1	
	zamiana	7	
	transpozycja	0	
TN	brak	689	689
FN	usunięcie	2	25
	wstawienie	0	
	zamiana	23	
	transpozycja	0	

Źródło: opracowanie własne.

4.2.2 LanguageTool

Wyniki uzyskane przez LanguageTool przedstawiają się następująco:

Sposób I

Tablica 4.3: Klasyfikacja wyrazów dla programu LanguageTool (Sposób I).

Kategoria	**Typ błędu**	**Liczba błędów**	**Wszystkie**
TP	usunięcie	12	134
	wstawienie	15	
	zamiana	98	
	transpozycja	9	
FP	usunięcie	3	9
	wstawienie	0	
	zamiana	5	
	transpozycja	1	
TN	brak	692	692
FN	usunięcie	5	26
	wstawienie	2	
	zamiana	17	
	transpozycja	2	

Źródło: opracowanie własne.

Sposób II

Tablica 4.4: Klasyfikacja wyrazów dla programu LanguageTool (Sposób II).

Kategoria	**Typ błędu**	**Liczba błędów**	**Wszystkie**
TP	usunięcie	14	145
	wstawienie	17	
	zamiana	103	
	transpozycja	11	
FP	usunięcie	3	9
	wstawienie	0	
	zamiana	5	
	transpozycja	1	
TN	brak	692	692
FN	usunięcie	4	15
	wstawienie	0	
	zamiana	11	
	transpozycja	0	

Źródło: opracowanie własne.

4.2.3 Porównanie

Porównanie jakości programów jest podzielone ze względu na sposoby ewaluacji. Wyniki każdego z nich przedstawia tabela oraz wykres:

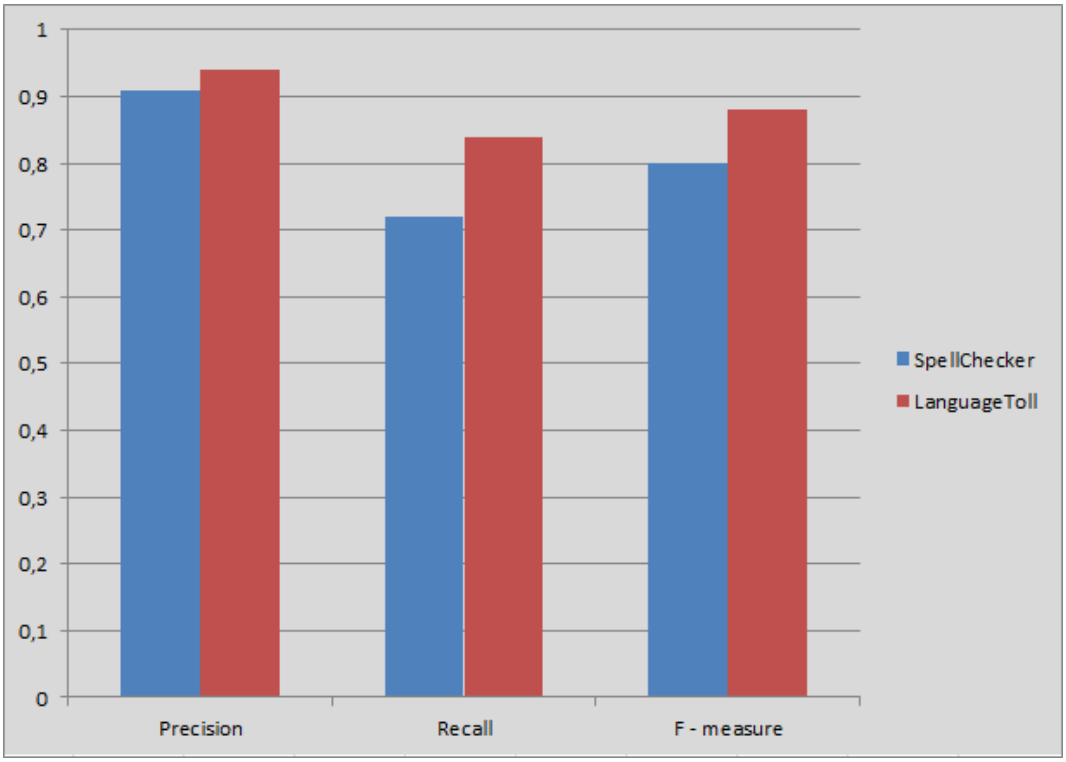
Sposób I

Tablica 4.5: Wyniki obu programów (Sposób I).

Miara	SpellChecker	LanguageTool
Precision	0,91	0,94
Recall	0,72	0,84
F-measure	0,80	0,88

Źródło: opracowanie własne.

Rysunek 4.1: Wykres przedstawiający wyniki (Sposób I).



Źródło: opracowanie własne.

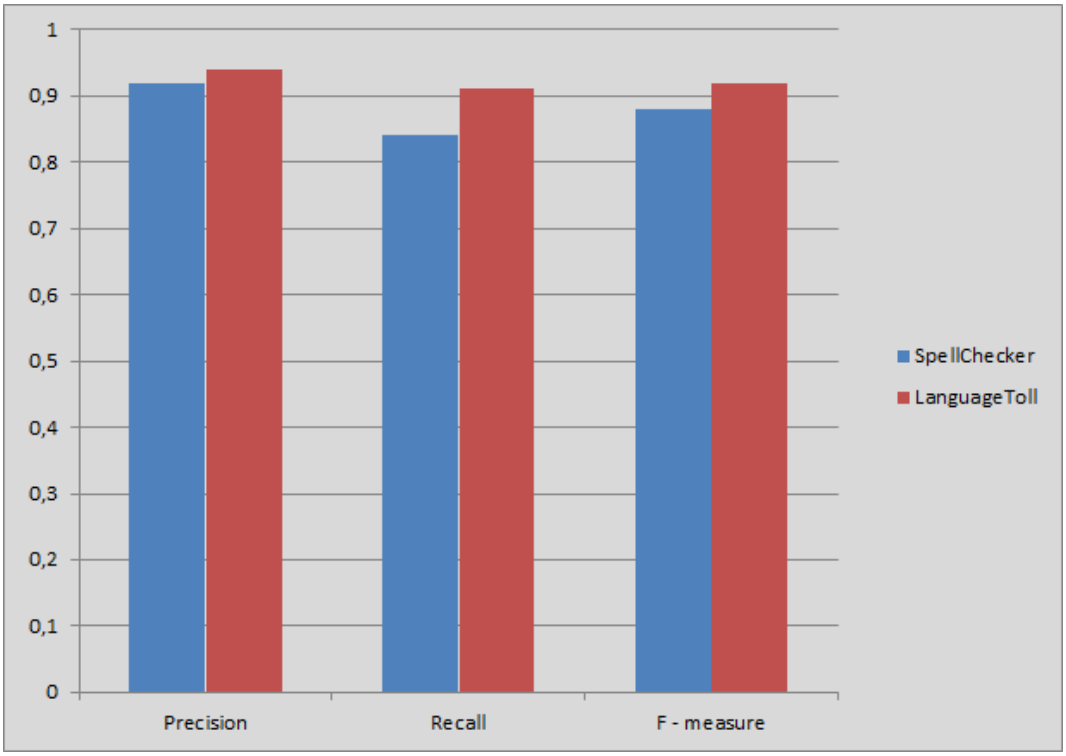
Sposób II

Tablica 4.6: Wyniki obu programów (Sposób II).

Miara	SpellChecker	LanguageTool
Precision	0,92	0,94
Recall	0,84	0,91
F-measure	0,88	0,92

Źródło: opracowanie własne.

Rysunek 4.2: Wykres przedstawiający wyniki (Sposób II).



Źródło: opracowanie własne.

4.3 Wnioski

Ze wszystkich 160 błędów występujących w tekstach, statystycznie najwięcej popełnianych jest zamian znaków (70%). Wiąże się to najczęściej z tym, że piszący często nie używają znaków diakrytycznych tworząc tekst na klawiaturze. Następnie plasują się usunięcia (12%), wstawienia (11%), a na końcu transpozycje (7%).

Przy błędnie zakwalifikowanych wyrazach niewielkiej długości, winę za złą ich korektę ponosi korpus błędów. Zazwyczaj można jednak wybrać poprawny wyraz z rozwijanej listy kandydatów. Natomiast prawdopodobieństwa niektórych długich wyrazów są zerowane przez to, że nie ma ich w korpusie języka np. brak w korpusie zdań pisanych w pierwszej osobie powoduje, że zerują się prawdopodobieństwa niektórych wyrazów ($P(w) = 0$), zwłaszcza w tekstach będących komentarzami z różnych forów internetowych. Korpus ma charakter encyklopedyczny, więc zawiera specyficzne dla siebie formy wyrazów. W przyszłości trzeba będzie stworzyć bardziej uniwersalny korpus, zawierający teksty o różnym charakterze. To działanie pozwoli rozwiązać większość problemów związanych ze złym wyborem kandydata.

Program nie radzi sobie też często z nazwami własnymi (np. zagranicznymi bez zamienników w języku polskim) oraz niektórymi skrótami, ale również, gdy błąd stworzy inny wyraz, należący do słownika. Ten ostatni problem jest bardzo trudny do rozwiązania dla każdego korektora pisowni. Aby sobie z tym poradzić, korektor musiałby brać pod uwagę kontekst wyrazowy, a kandydatów trzeba by generować również dla wyrazów należących do słownika. Problem nazw własnych lub skrótów w dużej mierze można rozwiązać dodając je do słownika.

Ponadto warto wspomnieć o tym, że przyjęcie drugiego sposobu ewaluacji pozwoliło zmniejszyć liczbę źle zakwalifikowanych błędów o 45%, co zwiększyło jakość samego programu, zatem pomysł z pięcioelementową listą okazał się trafny. Różnica ta jest najbardziej widoczna na mierze *pokrycia*, która bierze pod uwagę dobrze oraz źle zakwalifikowane błędy.

Słusznym okazało się przyjęcie założenia projektowego o ograniczeniu odległości edycyjnej do 1, ponieważ tylko 24% wszystkich błędów nie było sprawdzanych z powodu większej odległości edycyjnej, niż założona. Zdecydowana większość tych błędów to wyrazy z dużą ilością znaków diakrytycznych.

SpellChecker, mimo że nieznacznie przegrywa rywalizację w poprawianiu błędów, wypada bardzo dobrze na tle *LanguageTool'a*. Rozpoznawalność błędów jest na podobnym poziomie, a różnice w jakości są niewielkie. Mogą się one jeszcze zmniejszyć po wprowadzeniu ulepszeń (np. generowanie kandydatów w odległości edycyjnej równej 2) oraz wyeliminowaniu wcześniej wspomnianych problemów.

Zakończenie

Za pomocą metod automatycznych w krótkim czasie można stworzyć korektor o podobnej jakości działania, co korektory z ręcznie dodawanymi przez lata regułami do wykrywania, klasyfikacji i poprawy błędów. Można więc przyjąć, że eksperyment się powiódł. Jest jednak sporo możliwości poprawy wyników.

Można stworzyć bardziej uniwersalny model języka, składający się nie tylko z tekstów pisanych w jednym, specyficznym stylu, jak miało to miejsce w projekcie (korpus stworzony z artykułów Wikipedii). Nie ma bowiem żadnych przeszkód, żeby zbudować go z różnorodnych korpusów np. stworzonych z napisów do filmów, tekstów staropolskich, mowy współczesnej. W rezultacie powinno to dać jeszcze lepsze efekty, co przełoży się na jakość działania programu. Stworzenie bigramowego bądź trigramowego modelu języka na tak wielkiej ilości różnorodnych danych może dać satysfakcjonujące wyniki, nawet dla błędów należących do języka polskiego, a to już jest poważne ulepszenie dla programu.

Model unigramowy okazał się w pełni wystarczający dla korektora pisowni słów nienależących do słownika. Jeśli jednak program miałby poprawiać błędy należące do słownika, trzeba by stworzyć korpus języka metodą bigramów, a nawet trigramów, aby badany był kontekst wyrazowy. Wtedy tworzy się kandydatów do poprawy dla każdego wpisanego wyrazu. Aby uniknąć też zerującego prawdopodobieństwa można zastosować jedną z technik wygładzania, np. algorytm *Backoff*, który dla każdego przypadku zerowego prawdopodobieństwa w wyższym modelu N-gramów, stosuje model niższego rzędu w poszukiwaniu wystąpienia krótszej sekwencji wyrazów.

Model błędów stworzony na podstawie korpusu *PLEWi* wydaje się być wystarczający. Trudno będzie naprawić problem z błędnie poprawianymi krótkimi wyrazami, spowodowany przez model błędów. Zazwyczaj jednak dobry kandydat znajduje się w liście pięciu najlepszych, więc lista ta w dużej części już ten problem rozwiązuje.

Słownik języka polskiego również spełnia oczekiwania. W celu jego ulepszenia wystarczy dodać do niego wyrazy, których jeszcze nie zawiera (głównie nazwy własne i skróty). Warto rozważyć implementację w programie opcji dodawania wyrazu do słownika, którego jeszcze tam nie ma, a okaże się poprawny.

Mimo, że *SpellChecker* już osiąga satysfakcjonujące wyniki, można teoretycznie dokonać popraw w wielu obszarach jego działania. Sam program można jeszcze usprawnić dzięki wspomnianym ulepszeniom, więc perspektywy dalszych badań w tej dziedzinie są obiecujące.

Niewątpliwie największą zaletą użytych metod jest to, że program je używający można stworzyć w krótkim czasie metodami automatycznymi, bez długoletniego, ręcznego aktualizowania bazy reguł dla danego języka, jak to ma miejsce w przypadku *LanguageTool*. Korzysta się z gotowych zbiorów danych tworząc odpowiednie modele metodami statystycznymi. Oszczędza to wiele czasu, a uzyskany efekt powinien być co najmniej zbliżony do programów wykonanych metodą reguł. Na podstawie osiągniętych przez *SpellChecker* wyników można przyjąć, że eksperyment magisterski zakończył się sukcesem.

Bibliografia

Książki i artykuły:

[1] Red. Urbańczyk S.: Encyklopedia języka polskiego, Wyd. Zakład Narodowy im Ossolińskich, Wrocław, s. 33, 1991

[2] Markowski A.: Kultura języka polskiego, Wydawnictwo naukowe PWN, Warszawa, s. 55-60, 2005

[3] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 141-142, 2000

[4] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 219, 2000

[5] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 151-152, 2000

[6] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 191, 2000

[7] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 195-196, 2000

[8] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 198, 2000

[9] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 214-215, 2000

[10] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 149, 2000

[11] Jurafsky D., Martin J. H.: Speech and Language Processing, Wyd. Alan Apt, New Jersey, s. 150, 2000

[12] Peterson J. L.: A note on undetected typing errors. *Communications of the Association for Computing Machinery*, 29(7), s. 633–637, (1986)

[13] Damerau F. J., Mays, E.: An examination of undetected typing errors. *Information Processing and Management*, 25(6), s. 659–664, (1989)

[14] Damerau, F. J.: A technique for computer detection and correction of spelling errors. *Communications of the Association for Computing Machinery*, 7(3), s. 171–176, (1964)

[15] Kernighan, M. D., Church, K. W., and Gale, W. A.: A spelling correction program base on a noisy channel model. *In COLING-90*, Helsinki, Vol. II, s. 205–211, (1990)

[16] Naber D.: A Rule-Based Style and Grammar Checker, s. 5-7, (2003).

[17] Grundkiewicz R.: Automatic Extraction of Polish Language Errors from Text Edition History, *Proceedings of the 16th International Conference on Text, Speech and Dialogue TSD 2013*, LNCS, s. 1-8, (2013)

[18] Gralinski, F., Jassem, K., Junczys-Dowmunt, M.: PSI-Toolkit: Natural language processing pipeline. *Computational Linguistics - Applications*, s. 27–39, (2012)

Materiały internetowe:

[19] Collins M.: Course notes for NLP by Michael Collins (Language Modeling), Online: http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf, s. 1-2, (dostęp 12.07.2017r.)

[20] Jaworski W.: Miary jakości, Online https://www.mimuw.edu.pl/~wjaworski/SU/SU04_miary_jakosci.pdf (dostęp 12.07.2017r.)

[21] Jurafsky D., Manning C.: Spelling Correction and the Noisy Channel, Online: https://web.stanford.edu/class/cs124/lec/spelling.pdf, s. 44, (dostęp 15.06.2017r.)

[22] Roman G.: Pozyskiwanie przykładów błędów językowych błędów językowych z historii edycji tekstu Online: http://www.staff.amu.edu.pl/~romang/slides/romang-plewic-pl.pdf, s. 3, (dostęp 08.06.2017r.)

Strony internetowe:

[23] https://pl.wikipedia.org/wiki/Słownik_języka_polskiego

[24] https://en.oxforddictionaries.com/explore/what-is-a-corpus

[25] http://morfologik.blogspot.com/2006/05/korpus-bdw-jzykowych.html

[26] http://marcinmilkowski.pl/en/about-me1

[27] https://languagetool.org/pl/

Pliki:

[28] https://sjp.pl/slownik/po.phtml [Słownik języka polskiego] (dostęp 12.07.2017r.)

[29] http://clip.ipipan.waw.pl/PolishWikipediaCorpus [Korpus języka] (dostęp 12.07.2017r.)

[30] http://www.staff.amu.edu.pl/~romang/wiki_errors_pl.php [Korpus błędów] (dostęp 12.07.2017r.)

Projekt magisterski:

[31] https://github.com/362669/SpellChecker [Projekt magisterski] (dostęp 14.07.2017r.)

Spis rysunków

Spis tablic