**AKADEMIA GÓRNICZO-HUTNICZA**

**IM. STANISŁAWA STASZICA W KRAKOWIE**



**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

Katedra Automatyki

**PRACA MAGISTERSKA**

**Opracowanie narzędzi informatycznych wspomagających analizę statystyczną języka włoskiego**

Gabriela Pastuszka   
Kierunek studiów: Informatyka Stosowana

Opiekun pracy Dr inż. Mirosław Gajer

**Oświadczenie autora**

Oświadczam, świadoma odpowiedzialności karnej za poświadczenie nieprawdy,

że niniejszą pracę dyplomową wykonałam osobiście i samodzielnie,

oraz że nie korzystałam ze źródeł innych niż wymienione w pracy.

Contents

[1. Cel pracy 5](#_Toc393138940)

[2. Przetwarzanie języka naturalnego – wprowadzenie 6](#_Toc393138941)

[2.1. Odkrywanie wiedzy w bazie danych (KDD) 6](#_Toc393138942)

[2.2. Statystyczna analiza języka 8](#_Toc393138943)

[2.2.1. Uczenie maszynowe 8](#_Toc393138944)

[2.2.2. Eksploracja danych 9](#_Toc393138945)

[2.3. Główne zastosowania NLP 10](#_Toc393138946)

[2.4. Problemy NLP 12](#_Toc393138947)

[3. Zagadnienia statystyczne – omówienie 15](#_Toc393138948)

[3.1. Kwantytatywne prawa językowe 15](#_Toc393138949)

[3.2. Oznaczanie części mowy 15](#_Toc393138950)

[3.3. Wyszukiwanie kolokacji 15](#_Toc393138951)

[4. Język włoski – krótka charakterystyka pod kątem analizy statystycznej 16](#_Toc393138952)

[4.1. Wyrazy pochodzenia obcego 16](#_Toc393138953)

[4.2. Oznaczanie części mowy (POS tagging) 17](#_Toc393138954)

[4.2.1. Przyrostki 17](#_Toc393138955)

[4.2.2. Składnia 18](#_Toc393138956)

[5. Implementacja rozwiązania 20](#_Toc393138957)

[5.1. Środowisko programistyczne 20](#_Toc393138958)

[5.2. Interfejs graficzny 20](#_Toc393138959)

[5.3. Struktura aplikacji 21](#_Toc393138960)

[5.4. Rozpoczęcie pracy z programem 22](#_Toc393138961)

[5.4.1. Rozpoznanie kodowania znaków 23](#_Toc393138962)

[5.4.2. Obliczenie średniej długości wyrazów i zdań 23](#_Toc393138963)

[5.4.3. Sporządzenie rozkładu częstotliwości wyrazów i określenie zróżnicowania leksykalnego 23](#_Toc393138964)

[5.4.4. Tokenizacja 24](#_Toc393138965)

[5.5. Analiza zjawisk związanych z częstotliwością występowania wyrazów. 24](#_Toc393138966)

[5.5.1. Ranking częstotliwości wyrazów 24](#_Toc393138967)

[5.5.2. Prawo Zipfa 25](#_Toc393138968)

[5.6. Wyszukiwanie wzorców w tekście 27](#_Toc393138969)

[5.6.1. Wyrazy pochodzenia obcego 27](#_Toc393138970)

[5.6.2. Wyrazy spełniające wyrażenia regularne 28](#_Toc393138971)

[5.7. Oznaczanie części mowy 29](#_Toc393138972)

[5.7.1. Tagger ręczny 29](#_Toc393138973)

[5.7.2. Tagger wykorzystujący wyrażenia regularne 30](#_Toc393138974)

[5.7.3. Tagger składniowy 31](#_Toc393138975)

[5.7.4. Tagger prawdopodobieństwa 31](#_Toc393138976)

[5.7.5. Ocena rezultatów tagowania 32](#_Toc393138977)

[5.7.6. Wzorcowo otagowany korpus 33](#_Toc393138978)

[5.8. Wyszukiwanie związków frazeologicznych 33](#_Toc393138979)

[5.9. Analiza kontekstu słów 33](#_Toc393138980)

[6. Dodatek X 34](#_Toc393138981)

1. Cel pracy

Celem pracy jest opracowanie narzędzi informatycznych wspomagających analizę statystyczną języka włoskiego.

Jest to aplikacja napisana w języku Python, przy użyciu biblioteki graficznej PyQt. Analizy dokonuje się wczytując plik zawierający próbkę tekstu w języku włoskim.

2-3 strony

1. Przetwarzanie języka naturalnego – wprowadzenie

Przetwarzanie języka naturalnego (NLP) to szeroka gałąź nauki, łącząca zagadnienia informatyki, sztucznej inteligencji i lingwistyki, badająca zjawiska zachodzące w językach naturalnych. Jej zastosowanie można sprowadzić do tłumaczeń z jednego języka naturalnego na drugi, do analizy próbek w językach naturalnych i przekształcania ich na język formalny, zrozumiały przez maszyny oraz przedstawiania danych pobranych z bazy danych w języku naturalnym, zrozumiałym przez człowieka. Zwłaszcza analiza i rozumienie języka naturalnego jest tu bardzo złożonym zagadnieniem, wymagającym od komputera szerokiej wiedzy o świecie rzeczywistym, umiejętności wychwytywania sensu słów z kontekstu, interpretacji metafor, ironii i innych środków stylistycznych.

Analiza języka naturalnego dotyczy zarówno języka mówionego jak i pisanego, jednak są to dość odrębne kwestie, zmagające się z innymi problemami. Przekształcenie sygnału analogowego w symbole języka formalnego może być bardzo trudnym procesem. Dużą rolę odgrywa tutaj gwara, intonacja (świadcząca np. o emocjach mówiącego, o charakterze prowadzonej rozmowy), akcent padający na określony wyraz (może zmienić znaczenie całego zdania). W mojej pracy skupiam się na języku pisanym.

* 1. Odkrywanie wiedzy w bazie danych (KDD)

KDD (Knowledge Discovery in Databases) to interaktywny i iteracyjny proces przetwarzania i ekstrakcji wiedzy (wzorców, reguł, informacji) z dużych baz danych. Powstał w latach 90, kiedy w obliczu dramatycznie rosnącej ilości danych cyfrowych (dotyczących wielu dziedzin, zarówno nauki jak i biznesu) dostrzeżono potrzebę opracowania nowych narzędzi i sposobów pozwalających na szybką ekstrakcję wiedzy z baz danych. Jest to proces składający się z kilku etapów, które często wymagają od badacza podejmowania subiektywnych decyzji – nie jest to proces w pełni zautomatyzowany.

Mimo szerokiego zastosowania KDD i idącej za tym rozmaitej specyfiki danych, można wyróżnić następujące podstawowe etapy:

* zrozumienie dziedziny zastosowania i sprecyzowanie celu całego procesu;
* utworzenie zestawu danych źródłowych – wybór odpowiednich próbek;
* wstępne przetworzenie i ujednolicenie danych wejściowych – np. usunięcie szumu, obsłużenie niepełności danych (brakujące pola rekordów);
* redukcja i odpowiednie przedstawienie danych pod kątem określonego celu;
* dobór najlepszej metody ekstrakcji wiedzy z danych do określonego celu, np. streszczenie, klasyfikacja, regresja;
* wybór hipotezy, wybór modelu i parametrów do przedstawienia pod kątem użytkownika końcowego;
* eksploracja wiedzy – poszukiwanie interesujących wzorców w określonych reprezentacjach danych. Poprzez poprawne wykonanie poprzednich kroków badacz może znacząco poprawić efekty poszukiwań;
* interpretacja wykrytych wzorców, ewentualny powrót do kroków poprzednich w celu poprawy wyników;
* wykorzystanie zdobytej wiedzy – sporządzenie raportów, sprawdzenie, czy nie zachodzi konflikt z wcześniej pozyskaną wiedzą.

Uproszczony przebieg KDD uwzględniający powroty do poprzednich etapów został przedstawiony na rysunku 1.



Rysunek Przebieg procesu odkrywania wiedzy w bazie danych

W kontekście przetwarzania języka naturalnego, pierwszym etapem jest wybór odpowiednich tekstów (korpusu języka). Surowy tekst korpusu należy ujednolicić, np. poprzez zamianę dużych liter na małe, usunięcie niepożądanych znaków. Następnie, w zależności od charakteru poszukiwań, trzeba przygotować odpowiednią prezentację danych, np. rozkład częstotliwości wyrazów czy zestawienie par wyrazów często występujących razem. W tak przygotowanych danych można wyszukiwać wzorce za pomocą różnych metod, opisanych szerzej w rozdziale 2.2.2

* 1. Statystyczna analiza języka

W celu zautomatyzowania procesów przetwarzania oraz umożliwienia analizy dużych ilości danych, stosuje się metody stochastyczne, statystyczne oraz probabilistyczne. Szczególnie przydatne są one w przypadku długich, wieloznacznych zdań, do których analizy nie wystarczają proste zasady gramatyczne. Statystyczne NLP opiera się na podejściu ilościowym – modelach probabilistycznych, teorii informacji i algebrze liniowej. Główne techniki działania to uczenie maszynowe i eksploracja danych.

* + 1. Uczenie maszynowe

Uczenie maszynowe jest cechą nowoczesnych algorytmów NLP. W przeciwieństwie do algorytmów opartych na z góry narzuconych regułach, polega ono na automatycznym uczeniu się tych reguł przez system poprzez analizę dostarczonego korpusu języka (bazy tekstów w języku naturalnym), opatrzonego prawidłowymi wartościami (np. tagami części mowy, wyjątkami specyficznymi dla danego języka, itd.). Algorytmy oparte na uczeniu maszynowym posiadają wiele zalet w porównaniu do algorytmów opartych na ręcznie wprowadzanych regułach:

* są efektywne, gdyż skupiają się na najczęściej występujących przypadkach, podczas gdy przy ręcznym definiowaniu reguł trudno określić, na co położyć największy nacisk,
* korzystając z metod wnioskowania statystycznego (estymacje, weryfikacje hipotez), są odporne na błędne lub niespotkane wcześniej dane wejściowe,
* dokładność wyników można zwiększyć poprzez dodanie nowych danych wejściowych – większa próba oznacza lepsze działanie algorytmu; ręcznie wprowadzane reguły muszą zaś być coraz bardziej skomplikowane dla polepszenia wyniku.
  + 1. Eksploracja danych

Eksploracja danych (data mining) to etap analizy w odkrywaniu wiedzy z baz danych (KDD). Polega na wykrywaniu wzorców w dużych zbiorach danych z wykorzystaniem różnych metod, głównie wywodzących się z badań nad sztuczną inteligencją, np. . Idea eksploracji danych sprowadza się do wykorzystania szybkości komputera w celu wykrywania prawidłowości ukrytych dla człowieka ze względu na ograniczone możliwości czasowe.

W przypadku NLP, wzorce te mogą dostarczyć nam wiedzy o różnych aspektach językowych danego tekstu. Przykładowo, różnorodność leksykalna czy średnia długość wyrazu może być diametralnie różna dla napisów filmowych i dla tekstu rozprawy naukowej. Duża liczba wystąpień słowa typowego dla danego dialektu, slangu czy żargonu może zdradzić informację o pochodzeniu i środowisku autora tekstu.

Istnieją różne metody przetwarzania stosowane w eksploracji danych:

* wykrywanie anomalii – nietypowe dane, mogą świadczyć o błędzie;
* modelowanie zależności – wykrywanie związków pomiędzy pozornie niezależnymi od siebie zmiennymi. Istnieje tutaj ryzyko nadinterpretacji (korelacja nie zawsze oznacza związek przyczynowo – skutkowy);
* klasteryzacja – odkrywanie grup i struktur danych mających jakieś cechy wspólne;
* klasyfikacja – stosowanie istniejących (wykrytych wcześniej) grup na nowych danych (przypisywanie im klas)
* regresja – próba znalezienia funkcji obrazującej zestaw danych z najmniejszym błędem
* streszczenie – opracowanie zwięzłej formy zestawu danych poprzez wizualizację i raporty.
  1. Główne zastosowania NLP

Przetwarzanie języka naturalnego znajduje bardzo różnorodne zastosowania, od praktycznych (np. tłumaczenie tekstów) do czysto teoretycznych (badanie zjawisk zachodzących w językach naturalnych). A oto niektóre z nich:

* automatyczne streszczanie tekstu – tworzenie krótkiego streszczenia, zawierającego najważniejsze informacje oryginalnego tekstu. Istnieją dwa rodzaje streszczeń: ekstrakt i abstrakt. Ekstrakt składa się ze słów i zdań wybranych z oryginalnego tekstu, mających największe (według oceny systemu) znaczenie. Abstrakt powstaje poprzez utworzenie semantycznej reprezentacji danych i wygenerowanie na jej podstawie tekstu w języku naturalnym. Jest to trudniejsze zadanie, wymagające zarówno trafnej intepretacji danych, jak i spójnej prezentacji w formie tekstu w języku naturalnym. Analogicznie do streszczenia można także dokonywać upraszczania – przedstawiania danego tekstu prostszym językiem, np. za pomocą zdań pojedynczych.
* tłumaczenie tekstu z jednego języka naturalnego na drugi – to trudne zadanie, a jego złożoność jest silnie zależna od stopnia skomplikowania gramatyki poszczególnych języków podlegających tłumaczeniu. Ważną kwestią jest odróżnienie idiomów i innych metaforycznych wyrażeń, których nie należy tłumaczyć dosłownie. Potrzebna jest więc dostarczona uprzednio baza takich zwrotów.
* generowanie języka naturalnego – przekształcanie treści zapisanych w bazach danych do formy czytelnej dla człowieka. Ważnym zadaniem jest dobór odpowiednich słów, poprawność gramatyczna i naturalność.
* rozumienie języka naturalnego – proces odwrotny do generacji języka o znacznie większej złożoności, wymagający trafnej analizy kontekstu, anafor, odniesień do świata zewnętrznego, a także umiejętności rozstrzygania wieloznaczności. Stanowi podstawę wielu wymienionych tutaj zastosowań, a także bardzo atrakcyjny sposób porozumiewania się z komputerem.
* rozróżnianie części mowy – zadanie pośrednie, pomocne m. in. w wykrywaniu konkretnych kolokacji (np. par czasownik + rzeczownik) i w tłumaczeniu. Automatyczne rozstrzygnięcie części mowy danego słowa nie zawsze jest możliwe, często można tego dokonać tylko z pewnym prawdopodobieństwem. Algorytmy tagujące działają iteracyjnie – najpierw oznaczane są wyrazy, których przynależność do danej części mowy jest oczywista. W kolejnych iteracjach łatwiej można określić część mowy wyrazów sąsiadujących.
* podział morfologiczny – wyodrębnienie z tekstu i klasyfikacja morfemów (najmniejszej jednostki gramatycznej). Kluczowy wpływ ma tu złożoność morfologiczna (struktura wyrazów) danego języka. Nie jest to zastosowanie samo w sobie, stanowi jeden z etapów rozumienia języka naturalnego czy dokonywania tłumaczeń.
* wykrywanie kolokacji – odnajdywanie par wyrazów często występujących razem, mogących stanowić utarte związki frazeologiczne, powiedzenia, memy ulegające modzie. Pozwalają one śledzić zmiany zachodzące w języku na przełomie lat.
* badanie specyfiki tekstów różnego pochodzenia – na podstawie zróżnicowania leksykalnego, średniej długości słowa, liczbie pomyłek w tekście i wielu innych czynników można się wiele dowiedzieć na temat danej próbki tekstu – czy jest to np. zapis nieoficjalnej rozmowy telefonicznej, czy też przemówienie polityka. Informacje te są bardzo przydatne w generowaniu tekstu w języku naturalnym, gdy ważne jest nadanie wypowiedzi odpowiedniego stylu.
* dostarczanie informacji o języku – każdy język naturalny posiada swoje specyficzne cechy, uwarunkowane kulturowo i historycznie. Stosując proste metody statystyczne można badać język pod kątem podatności na wpływ wyrazów pochodzenia obcego, neologizmów, tworzenia się dialektów i gwar.
* pomoc w nauce języka – osoba chcąca szybko nauczyć się danego języka może położyć nacisk na wyrazy najczęściej występujące w całym języku lub jego podzbiorze odpowiadającym jakiemuś zagadnieniu, np. nomenklatura prawnicza. Bardzo przydatne jest też wyszukiwanie kontekstów czy związków frazeologiczych w jakich często występuje dane słowo. Istnieją także programy pomagające w nauce wymowy lub pisma, oceniające podobieństwo próbek dźwiękowych i graficznych.
* automatyczne ocenianie wypracowań, esejów – jako pomoc dla nauczyciela, program taki może szybko ocenić różnorodność leksykalną, liczbę błędów, a także wartość merytoryczną wypracowań, np. zgodność z faktami historycznymi.
* rozpoznawanie i definicja nazw własnych – wyszukiwanie w tekście wyrazów będących nazwami własnymi i szerszych informacji o nich (przydatne zwłaszcza w przypadku języków, w których nazw własnych nie pisze się wielką literą).
  1. Problemy NLP

Przetwarzanie języka naturalnego, a zwłaszcza jego rozumienie natrafia na wiele problemów i barier, dlatego zagadnienie to jest często nazywane problemem AI-complete (problemem sztucznej inteligencji). Aby z powodzeniem rozwikłać liczne wieloznaczności języka naturalnego, potrzebna jest bowiem szeroka wiedza o świecie zewnętrznym, a także rozumienie aluzji i skojarzeń – sposób myślenia właściwy dla ludzkiego mózgu.

Główne problemy NLP można przedstawić następująco:

* wieloznaczność składniowa zdania – zwana też amfibologią - zwłaszcza długie, wielokrotnie złożone zdania dopuszczają mnogość interpretacji. W niektórych przypadkach poprawna interpretacja wymaga elementarnej wiedzy o świecie i nie stanowi problemu dla człowieka. Np. w przypadku zdań *Daliśmy małpom banany, bo były głodne* i *Daliśmy małpom banany, bo były przejrzałe* czasownik *były* odnosi się za każdym razem do innego rzeczownika. Poprawne ich zintepretowanie wymaga wiedzy na temat małp i bananów.
* wieloznaczność zakresu – rodzaj błędu logicznego, wynikający z nieprawidłowego uzycia kwantyfikatorów lub podobnych określeń, takich jak każdy, wiele, ktoś, nikt. Np. angielskie zdanie *Many people visit Europe every month* może oznaczać zarówno fakt, iż istnieje grupa wielu ludzi zwiedzających Europę co miesiąc, oraz że Europa jest odwiedzana co miesiąc przez wielu ludzi (niekoniecznie tych samych).
* rozstrzyganie odniesień – w języku angielskim problematyczne jest jednoznaczne określenie, jakiego słowa tyczy się dany przymiotnik, np. wyrażenie *pretty little girls’ school* można rozumieć aż na 4 sposoby (przymiotniki *pretty* i *little* mogą dotyczyć zarówno słowa *girls*, jak i *school*).
* błędy językowe, logiczne, stylistyczne - w języku polskim wieloznaczności rodzi np. rzeczownik w funkcji przydawki lub rzeczownik mogący spełniać zarówno funkcję podmiotu jak i dopełnienia, np. *Samochód wyprzedził motocykl*. Kolejny rodzaj wieloznaczności powodują homonimy, np. Problemy tego typu są cechą języków naturalnych i nie tylko komputer miałby trudności z ich interpretacją.
* homonimy – wiele wyrazów ma więcej niż jedno znaczenie, więc tylko kontekst, w jakim dane słowo występuje, może pomóc w określeniu jego znaczenia. Filozof języka, Ludwig Wittgenstein, zajmował się m. in. problemem znaczenia słowa i ujął to w ten sposób: „*Znaczeniem słowa jest sposób użycia go w języku.”*
* niemożność określenia części mowy – problem ten dotyczy głównie języków analitycznych, posiadających znikomą morfologię fleksyjną, która w innych językach pomaga odróżnić od siebie części mowy. Przykładem jest angielskie zdanie *Time flies like an arrow.* Oprócz oczywistego porównania upływu czasu do lotu strzały można wyróżnić co najmniej 6 innych znaczeń, w zależności od zaklasyfikowania poszczególnych wyrazów do odpowiednich części mowy. Również w językach syntaktycznych, np. w języku polskim, nie zawsze da się jednoznacznie określić część mowy – przykładem może być słowo *śledź*, które można interpretować jako rzeczownik lub czasownik w trybie rozkazującym (może to prowadzić do zabawnych konsekwencji – pozycję w menu pewnej restauracji brzmiącą po polsku ‘śledź w oleju’ przetłumaczono na angielski jako ‘follow in oil’).
* interpretacja idiomów – ponieważ nie można ich traktować dosłownie, wymagają one wcześniej zdefiniowanej bazy znaczeń. W przypadku dokonywania tłumaczenia, można także spróbować znaleźć analogiczny idiom w języku docelowym.
* błędne dane wejściowe – literówki, brak znaków interpunkcyjnych mogą niekiedy całkowicie uniemożliwiać ich poprawne zrozumienie przez komputer (a także przez człowieka). Drobne błędy pisowni mogą być łatwo wykryte i naprawione automatycznie, o ile nie zachodzi wieloznaczność.
* wpływ akcentu na znaczenie zdania – w interpretacji języka mówionego dużą rolę odgrywa akcent, który wskazując na wybrane słowo zmienia zupełnie znaczenie całego zdania.
* wieloznaczność mowy, homofonów – kolejny rodzaj wieloznaczności, dotyczący tylko języka mówionego, np. zlepek słów *mamamanastroje* można rozbić na co najmniej 8 różnych zdań, każde o innym znaczeniu.

Podsumowując, dobry system NLP mający znaleźć praktyczne zastosowanie, musi być dobry w podejmowaniu decyzji ujednoznaczniających znaczenie wyrazu oraz część mowy, do której należy, strukturę składniową zdania i zakres semantyczny. Powinien być także odporny na błędne dane wejściowe (w miarę możliwości poprawiać błędy pisowni). Dodatkowo, jeśli system ma rozumieć mowę, musi radzić sobie także z jej niuansami – wieloznacznością homonimów i zmianą znaczenia zdania w zależności od wyrazu, na który położony jest akcent.

1. Zagadnienia statystyczne – omówienie
   1. Kwantytatywne prawa językowe
   2. Oznaczanie części mowy

Istnieje wiele sposobów określania części mowy danego słowa w tekście.

* 1. Wyszukiwanie kolokacji

1. Język włoski – krótka charakterystyka pod kątem analizy statystycznej

Język włoski ma wiele interesujących cech, które ułatwiają jego analizę statystyczną. Dokładniejszy opis języka znajduje się w ..................... – w tym rozdziale powtórzę tylko najważniejsze informacje.

* 1. Wyrazy pochodzenia obcego

Ponieważ każdy wyraz w języku włoskim (poza kilkoma wyjątkami) kończy się na samogłoskę, łatwo znaleźć w tekście wyrazy obcego pochodzenia. Co więcej, w alfabecie języka włoskiego nie występują litery *k, j ,w, x, y,* toteż wyrazy zawierające choć jedną z nich również można zakwalifikować jako zapożyczone z innych języków. Wyjątkiem są tu słowa, które na stałe zagościły w słowniku języka włoskiego, np. *extracommunitari, xenofobia.*

Innym, o wiele skuteczniejszym sposobem wykrywania wyrazów obcych jest sprawdzanie słów pod kątem występowania w nich bigramów nietypowych dla języka włoskiego. Metoda ta działa dla każdego języka – w istocie jest to popularny sposób automatycznego rozpoznawania języków. Posiada jednak dużą wadę – przy alfabecie zawierającym 21 liter podstawowych i 10 liter ze znakami diakrytycznymi (5 liter obcych dla języka włoskiego także może być tu brane pod uwagę, jeśli chcemy wykryć wspomniane wcześniej wyrazy z obcą pisownią) istnieje aż 961 bigramów, z czego większość stanowią te wymagające sprawdzenia, nietypowe dla języka włoskiego. Metoda ta jest zatem bardzo złożona obliczeniowo. W mojej pracy poprzestanę więc na wspomnianych wcześniej metodach wykorzystujących specyficzne cechy języka włoskiego.

* 1. Oznaczanie części mowy (POS tagging)

Język włoski charakteryzuje się dużą regularnością, bardzo pomocną w określaniu części mowy. Tę właściwość można wykorzystać programując taggery części mowy.

* + 1. Przyrostki

Istnieje wiele przyrostków wyrazów typowych tylko dla czasowników, rzeczowników i przymiotników (trzeba jednak wziąć pod uwagę nieliczne wyjątki). Za pomocą taggera wykorzystującego wyrażenia regularne można zatem określić pewien odsetek słów. A oto najczęściej występujące wzorce:

* Przysłówki:
  + –mente;
* Rzeczowniki:
  + -ina, -ino (z wyjątkiem przymiotników divino, vicino) – końcówka zdrabniająca wyraz
  + -etto – (z wyjątkiem detto, letto – czasowniki w formie biernej) – jw.
  + -one, -ona (z wyjątkiem przymiotnika *buono*) – końcówka oznaczająca zgrubienie wyrazu
  + -ore z wyjątkiem peggiore, maggiore;
  + -ezza – rzeczowniki utworzone z czasowników (np. *bellezza - piękno)*
  + –tà – rzeczowniki rodzaju żeńskiego nie zmieniające formy w liczbie mnogiej
  + -aggio
  + -ismo
* Czasowniki:
  + -are, -ere, -ire – w formie bezokolicznika;
  + -armi, -arti, -arla, -arle, -arli, -arci, arvi, argli, arsi – bezokolicznik zakończony zaimkiem (forma skrócona). Należy tu wykluczyć czasownik *parlare*, posiadający w odmianie formy *parla*, *parli.*
  + -amo – czasownik w drugiej osobie liczby mnogiej;
  + -rà – końcówka czasownika w trzeciej osobie liczby pojedynczej w czasie Futuro Semplice
  + -ando, -endo - imiesłów
  + -andola, -andole, -andone, -andosi – imiesłów zlożony (zakończony zaimkiem)
* Przymiotniki:
  + -issimo, -issima, -issimi, -issime –stopień najwyższy bezwzględny;
  + -ile;
  + -oso, -osa.
    1. Składnia

Informacji o przynależności wyrazu do części mowy mogą dostarczyć części mowy w jego sąsiedztwie (jeśli zostały już określone).

* Zaimki dzierżawcze zwykle poprzedzają rzeczownik, którego dotyczą (w niektórych przypadkach kolejność jest odwrotna, wtedy po zaimku następuje jakaś określona wcześniej część mowy, np. spójnik);
* Zaimki dopełnienia bliższego w przeważającej liczbie przypadków poprzedzają swój czasownik zwrotny (w zdaniach z czasownikami modalnymi mogą dołączyć do bezokolicznika – sytuacja omówiona w 3.2.1 lub wystąpić bezpośrednio przed czasownikiem modalnym).
* Przymiotniki w stopniu najwyższym są tworzone przez strukturę il/la più [przymiotnik], np. *Marco è il ragazzo più alto.* Zatem, gdy tagger napotka parę *il/la più* , lub *il/la* [rzeczownik] *più* , kolejnym wyrazem musi być przymiotnik.
* Rodzajniki i przyimki, które uległy zabiegowi elizji w większości przypadków stoją przed rzeczownikiem (zaczynającym się samogłoską).
* ..........

Istnieją też utrudnienia w oznaczaniu części mowy:

* wyrazy, które mogą być zarówno rzeczownikiem jak i przymiotnikiem – np. w języku polskim wyrazy *chory, zakochany.* W języku włoskim może to być zwrot *il mio [ragazzo],* w którym zaimek dzierżawczy w domyśle pełni funkcję rzeczownika *chłopak.*
* przyimki złożone, pełniące jednocześnie funkcję rodzajnika – della, nello;
* zaimki nieokreślone liczby mnogiej (*dei, degli, delle*) – brzmią tak samo jak wspomniane w poprzednim podpunkcie przymki złożone;
* czasowniki złożone, z doklejonym na końcu zaimkiem osobowym;
* homonimy – wyrazy o tej samej pisowni, lecz innym znaczeniu i części mowy, np. *letto* może być rzeczownikiem, czasownikiem (imiesłów czasu przeszłego od czasownika *leggere)* orazprzymiotnikiem (pochodzącym od tego samego czasownika).
* słówko *ci* ma wiele zastosowań – jest to zaimek dopełnienia bliższego i dalszego dla pierwszej osoby liczby mnogiej, a także partykuła zastępująca dłuższe wyrażenia, m. in.. określenia czasu i miejsca, dopełnienia wprowadzanie przez przyimki *a, da, con, su*. Ponadto wchodzi w skład wielu czasowników złożonych, np. *esserci, volerci, avercela* (w odmianie tych czasowników składnik *ci* jest odrębnym słowem).

1. Implementacja rozwiązania
   1. Środowisko programistyczne

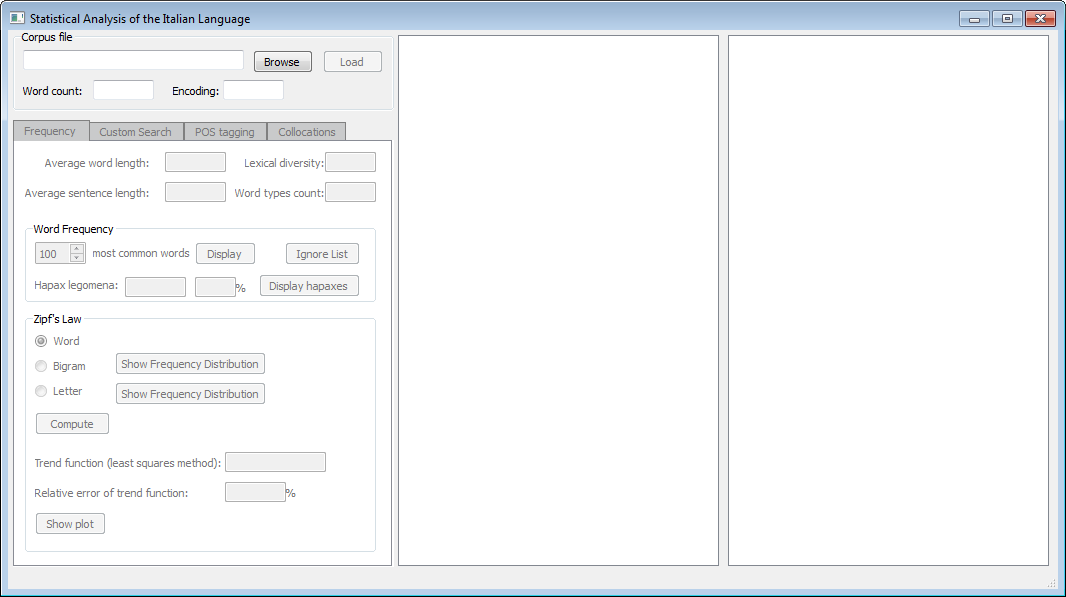
Aplikacja została napisana w języku Python (wersja 2.7). Interfejs graficzny powstał przy użyciu PyQt (nakładka na popularną bibliotekę Qt dla języka Python). Cały projekt powstał w środowisku Eclipse z nakładką PyDev, umożliwiającą integrację Pythona i QtDesignera (edytor GUI). W celu kompilacji projektu w systemie operacyjnym Windows potrzebny był także zestaw narzędzi deweloperskich MinGW.

Oprócz tego, pomocne były moduły dla Pythona: wspomagający obliczenia NumPy, wyświetlający wykresy PyQtGraph oraz biblioteka NLTK dostarczająca wiele narzędzi wspomagających obliczenia statystyczne.

* 1. Interfejs graficzny

Interfejs został zaprojektowany przy użyciu dostarczonego wraz z PyQt QtDesignera. Główne okno aplikacji składa się z trzech części – interfejsu właściwego oraz dwóch okien podglądu (do wyświetlania tekstu korpusu i bieżących wyników jego przetwarzania). Interfejs podzielony jest na cztery zakładki odpowiadające różnym zagadnieniom z zakresu analizy statystycznej języka:

* Frequency – są tu informacje związane z częstotliwością występowania i prawem Zipfa dla wyrazów, bigramów i liter oraz wartości średnich długości słowa i zdania dla korpusu;
* Custom Search – umożliwia wyszukiwanie kontekstu o dowolnej długości dla podanego słowa oraz słów spełniających dane wyrażenie regularne;
* POS tagging – pozwala na taggowanie wyrazów korpusu częściami mowy przy użyciu 4 rodzajów taggerów. Możliwe jest załadowanie korpusu już otagowanego, w celu sprawdzenia efektywności tagowania.
* Collocations – umożliwia wyszukiwanie związków frazeologicznych za pomocą 6 różnych testów statystycznych, z możliwym uwzględnieniem określonych w poprzedniej zakładce części mowy.



Rysunek Główne okno aplikacji

* 1. Struktura aplikacji

Aplikacja została zaprojektowana zgodnie ze wzorcem Model-View-Presenter, będącym pochodną popularnego wzorca Model-View-Controller.



Rysunek Diagram ukazujący zależności pomiędzy elementami wzorców MVC i MVP

Model to klasa zawierająca całą logikę aplikacji. Przechowuje wczytany korpus, wyodrębnione z niego tokeny, rozmaite dane potrzebne do przetwarzania tekstu (np. reguły wyrażeń regularnych) oraz wyniki bieżących wyszukiwań. Udostępnia szereg metod przeprowadzających obliczenia i pozwalających na pobranie ich wyników. Jest to model pasywny – zmiany zachodzą jedynie na żądanie Prezentera. Istnieje niezależnie od Prezentera i Widoku („nie wie” o ich istnieniu).

Widok to zbiór widżetów biblioteki PyQt umożliwiający prezentację danych. Jest wygenerowany przez QtDesignera na podstawie projektu interfejsu opisanego w poprzednim podrozdziale. Podobnie jak Model, Widok jest niezależny od pozostałych komponentów.

Prezenter posiada informacje o Widoku i Modelu oraz zarządza przepływem danych. W konstruktorze klasy Prezentera następuje wczytanie widoku z pliku wygenerowanego przez QtDesignera oraz połączenie sygnałów tego widoku ze slotami – metodami Prezentera. W metodach tych Prezenter sprawdza, jeśli jest to konieczne, zaznaczone opcje w interfejsie i wywołuje odpowiednie metody Modelu przeprowadzające pożądane obliczenia i dostarczające ich wyniki. Pobrane wyniki są wyświetlane w widżetach Widoku. Prezenter jest zatem ściśle zależny od interfejsu Modelu i Widoku.

* 1. Rozpoczęcie pracy z programem

Pracę z aplikacją rozpoczyna się poprzez wczytanie pliku z korpusem języka za pomocą przycisków Browse i Load (slotem dla sygnału – wciśnięcia przycisku Browse jest standardowa funkcja biblioteki PyQt otwierająca okienko dialogowe umożliwiające wskazanie pliku). Przyciski i ścieżka do pliku z korpusem są widoczne z każdego miejsca programu i w dowolnej chwili można zmienić plik korpusu, z którym się pracuje. Podczas każdego wywołania metody ładowania pliku są przeprowadzane procedury opisane w podpunktach 5.3.1 – 5.3.4, które mają na celu przygotowanie wczytywanego tekstu z korpusem i obsłużenie ewentualnych wyjątków, dokonanie prostych obliczeń dostarczających podstawowe informacje statystyczne o tekście oraz podział na tokeny umożliwiający dalsze analizy pod kątem części mowy i wyszukiwania związków frazeologicznych.

* + 1. Rozpoznanie kodowania znaków

W trakcie wczytywania tekstu konieczne jest określenie jego kodowania – znaki specjalne języka włoskiego są najczęściej kodowane za pomocą standardu Unicode lub ISO. Domyślnie plik jest odczytywany za pomocą popularniejszego kodowania UTF-8, a w razie wystąpienia wyjątku stosowany jest ISO-8859-1 (zwany też Latin-1). Gdy i tym razem dojdzie do wyjątku, plik nie zostaje wczytany, a w polu zatytułowanym ‘Encoding’ pojawia się komunikat o błędzie. Natomiast po udanym załadowaniu pliku surowy tekst jest zapisywany w pamięci programu.

* + 1. Obliczenie średniej długości wyrazów i zdań

W celu znalezienia średniej długości zdania należy wyodrębnić je z tekstu, zliczyć i wyrazić ich długość w liczbie składających się na nie słów. W tym celu użyłam klasy PunktSentenceTokenizer bliblioteki NLTK. Domyślnie tokenizer ten traktuje kropki, wykrzykniki i pytajniki jako koniec zdania. Można go skonfigurować, dostarczając listę skrótów (ciągów znaków, po których kropka nie oznacza końca zdania). Lista użytych skrótów znajduje się w dodatku A. Wynikiem działania tokenizera jest lista stringów wraz ze wszystkimi znakami specjalnymi (białe znaki i inne znaki specjalne).

Do wydzielenia słów w zdaniach użyłam kolejnego tokenizera, bazującego na wyrażeniach regularnych. Jako słowa traktuje on ciągi liter alfabetu włoskiego (ignoruje wszelkie inne znaki). Na podstawie wyznaczonych list zdań i wyrazów obliczone zostają średnia długość zdania i wyrazu w korpusie.

* + 1. Sporządzenie rozkładu częstotliwości wyrazów i określenie zróżnicowania leksykalnego

Wykonany wcześniej podział tekstu na słowa można wykorzystać do utworzenia rozkładu częstotliwości występowania wyrazów, uprzednio zamieniając wszystkie litery na małe, by uniknąć duplikatów. Dobrym narzędziem jest klasa FreqDist z biblioteki NLTK. Ma strukturę opartą na bazie słownika i szereguje wyrazy od najczęściej występującego. Wyznaczony rozkład zostaje zapisany w pamięci programu.

Dzięki rozkładowi częstotliwości można łatwo obliczyć różnorodność leksykalną. Jest to iloraz liczby różnych słów (typów słów) oraz liczby wszystkich słów.

* + 1. Tokenizacja

Korzystając z listy wyodrębnionych zdań (zawierających także m. in. znaki przestankowe) można utworzyć listę tokenów. Tokenem może być wyraz (ciąg liter), ciąg cyfr, a także dowolny inny znak mogący pojawić się w tekście. Ponieważ znaki przestankowe (kropki, przecinki, pytajniki, wykrzykniki, nawiasy, itd.) mają znaczenie dla interpretacji składni zdania, należy wydzielić je jako osobne tokeny. W tym celu użyłam kolejnego taggera opartego na wyrażeniach regularnych. Rezultatem działania taggera jest lista tokenów, zapisana do pamięci aplikacji.

Po wykonaniu tych procedur wszystkie zakładki programu stają się aktywne, a obliczone charakterystyki tekstu: całkowita liczba wyrazów, kodowanie, średnia długość wyrazu, zdania, współczynnik różnorodności leksykalnej zostają wyświetlone w odpowiednich polach. W lewym oknie podglądu pojawia się nieprzetworzony tekst wczytanego korpusu, a w prawym – lista tokenów całego tekstu. Program jest gotowy do pracy.

* 1. Analiza zjawisk związanych z częstotliwością występowania wyrazów.

Pierwsza zakładka, zatytułowana ‘Frequency’ (częstotliwość), zawiera narzędzia pozwalające na badanie częstotliwości wyrazów i innych powiązanych zjawisk. Jej zawartość została podzielona na grupy omówione w kolejnych podrozdziałach.

* + 1. Ranking częstotliwości wyrazów

Grupa przycisków zatytułowana ‘Word Frequency’, przedstawiona na rysunku nr x umożliwia poznanie wybranej liczby najczęściej występujących wyrazów.



Rysunek lol

Przycisk „Ignore List” wyświetla okno dialogowe z edytowalną listą wyrazów ignorowanych, które nie będą pojawiać się w zestawieniu, można ją również dostarczyć poprzez plik o nazwie xxx znajdujący się w katalogu programu (domyślnie lista ta jest pusta). Po kliknięciu przycisku ‘Display’ pożądana lista wyrazów wraz z liczbą ich wystąpień zostaje pobrana ze sporządzonego wcześniej rozkładu częstotliwości i wyświetlona w osobnym okienku dialogowym.

Innym zjawiskiem związanym z częstotliwością występowania wyrazów są tzw. hapax legomena – wyrazy pojawiające się w tekście tylko raz. Można poznać ilość takich wyrazów, ich procentowy udział w tekście, a także wyświetlić je wszystkie w osobnym oknie dialogowym za pomocą przycisku ‘Display Hapaxes’ (obiekt klasy FreqDist z biblioteki NLTK posiada metodę hapaxes()).

* + 1. Prawo Zipfa

W kolejnej grupie zatytułowanej ‘Zipf’s Law’ (Prawo Zipfa) znajdują się narzędzia do analizy prawa Zipfa dla wyrazów, bigramów literowych i pojedynczych liter.



Rysunek lol

Po wybraniu interesującej nas jednostki tekstu i kliknięciu przycisku ‘Compute’, następuje przygotowanie danych do sporządzenia wykresu ilustrującego prawo Zipfa – wykresu zależności liczby wystąpień danej jednostki tekstu od jej pozycji w rankingu częstotliwości występowania. Dane te - współrzędne punktów na wykresie - są uzyskiwane w różny sposób, w zależności od wybranej jednostki tekstu:

* w przypadku wyrazu, punktów na wykresie musi być tyle, ile w tekście jest typów wyrazów. Liczby wystąpień są pobrane z gotowego rozkładu częstotliwości wyrazów.
* bigramy literowe – liczba wszystkich możliwych bigramów jest niezależna od tekstu i wynosi 1296 - jest to moc zbioru znaków alfabetu włoskiego (36) podniesiona do potęgi drugiej (zostały uwzględnione także znaki specjalne). Ilość wystąpień każdego bigramu w tekście jest obliczana poprzez iterację po zbiorze wyodrębnionych wcześniej tokenów i zliczenie bigramów występujących w każdym z nich.
* litery – analogicznie do bigramów, liczba liter jest naturalnie ograniczona przez moc zbioru znaków alfabetu włoskiego – 36, uwzględniając znaki specjalne; podobnie jest też określana liczba ich wystąpień w tekście (poprzez iterację po zbiorze tokenów i zliczenie liter).

W przypadku bigramów i liter znaki nienależące do z góry zdefiniowanego zbioru znaków alfabetu włoskiego są ignorowane, w przeciwieństwie do wyrazów.

Dla dobrego zilustrowania prawa Zipfa, przy pomocy metody najmniejszych kwadratów, zostaje wyliczona linia trendu. W tym celu użyłam funkcji polyfit biblioteki Numpy, która zwraca wektor współczynników wielomianu aproksymującego o zadanym stopniu – w tym przypadku równym 1. Współczynniki zostają zapisane w pamięci programu w celu dalszego użycia. Znalazłszy współczynniki wielomianu drugiego stopnia (linii trendu) można również obliczyć średni względny błąd przybliżenia, wg wzoru:

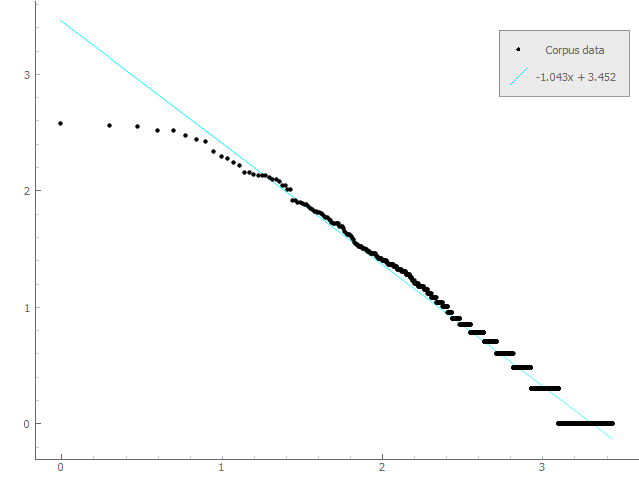
gdzie:

n – ilość punktów pomiarowych,

x­­­­­­i – i-ta wartość mierzona,

xia - i-ta wartość aproksymowana.

Obliczone wartości (wzór funkcji – linii trendu i średni błąd względny) pojawiają się w odpowiednio opisanych polach. Przycisk ‘Show Plot’ staje się aktywny. Po jego naciśnięciu pojawia się okienko dialogowe z wykresem punktowym oraz wykresem funkcji aproksymującej (w skali logarytmicznej).



Rysunek Wykres punktowy rozkładu częstotliwości oraz wykres linii trendu

W przypadku bigramów i liter, można poznać ich rozkład częstotliwości za pomocą przycisku ‘Show Frequency Distribution’ (aktywnego po dokonaniu obliczeń dla prawa Zipfa). Podobnie jak w przypadku wyrazów, zostaje wyświetlone okienko dialogowe z listą bigramów lub liter, posortowanych wg liczby wystąpień. Możliwe jest jednoczesne wyświetlenie okna z wykresem Zipfa oraz rozkładu częstotliwości, w celu przenalizowania liczby wystąpień poszczególnych jednostek.

* 1. Wyszukiwanie wzorców w tekście

Kolejnym zagadnieniem dostarczającym wielu informacji o tekście jest wyszukiwanie wzorców. W kolejnych podrozdziałach omówię zawartość zakładki ‘Patterns’ (Wzorce), zawierającej narzędzia do wyszukiwania wyrazów spełniających określone wzorce.

* + 1. Wyrazy pochodzenia obcego

Pierwsza grupa widżetów jest zatytułowana ‘Foreign Words’ (Wyrazy pochodzenia obcego) i zawiera narzędzia do wyszukiwania takich wyrazów na podstawie dwóch reguł omówionych w podrozdziale 4.2. Reguły te można stosować oddzielnie – do ich wybierania służą przyciski typu CheckBox. Oprócz tego, istnieje możliwość zdefiniowania listy wyjątków – wyrazów, które nie zostaną uznane za pochodzenia obcego mimo spełnienia reguły. W tym celu należy kliknąć przycisk ‘Exceptions’ (Wyjątki). Pojawi się wtedy okienko dialogowe, w którym można wpisywać rzeczone wyjątki.



Rysunek lol

Znalezienie wyrazów następuje po wciśnięciu przycisku ‘Find’. Z listy tokenów zostaną wybrane te spełniające wybrane reguły i nienależące do zbioru wyjątków. W pustych polach pojawi się całkowita liczba znalezionych wyrazów oraz ich procentowy udział w tekście. Przycisk ‘Preview’ wyświetla okienko dialogowe z listą - rozkładem częstotliwości znalezionych słów.

Dodatkową funkcjonalnością jest możliwość dodawania wyjątków bezpośrednio w okienku wyników wyszukiwania wyrazów – możemy je usuwać za pomocą klawisza Delete. Zostaną one dodane do listy wyjątków, a wyniki wyszukiwania zostaną automatycznie zaktualizowane.

* + 1. Wyrazy spełniające wyrażenia regularne

Kolejną funkcjonalnością jest wyszukiwarka wyrazów spełniających dowolne wyrażenie regularne.

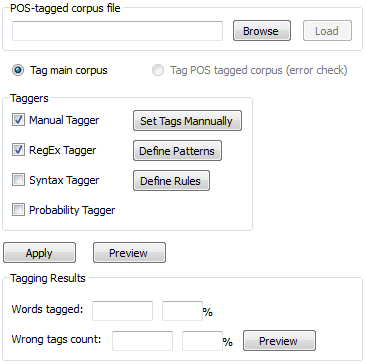


W polu ‘Pattern’ wpisujemy interesujące nas wyrażenie regularne. Po wciśnięciu przycisku ‘Find’, wszystkie tokeny korpusu zostają sprawdzone pod kątem spełniania wyrażenia. Liczba wszystkich wystąpień i odsetek procentowy wyrazów są wyświetlane odpowiednich polach. Po kliknięciu przycisku ‘Preview’, w osobnym okienku zostaje wyświetlona lista znalezionych wyrazów.

* 1. Oznaczanie części mowy

Automatyczne określanie części mowy można zaimplementować na wiele sposobów, wykorzystując różne zjawiska językowe i statystyczne. W mojej aplikacji zaimplementowałam 4 rodzaje taggerów. W celu sprawdzenia ich skuteczności i poprawności istnieje możliwość wczytania wzorcowo otagowanego korpusu i porównania trafności przypisanych tagów.

Na potrzeby przypisywania tagów wyrazom użyłam kolejnej struktury danych przechowującej tokeny. Ponieważ jeden z taggerów wykorzystuje informacje o składni zdania (położenie przecinków i kropek), znaki przestankowe muszą być traktowane jako tokeny. Początkowo wszystkie tokeny mają przypisany tag domyślny – pusty ciąg znaków. Zestaw używanych przeze mnie tagów wraz z krótkim opisem został zamieszczony w dodatku X.



Rysunek lol

W kolejnych podrozdziałach opisane zostały zasady działania poszczególnych taggerów, analiza wyników, a także sposób, w jaki jest oceniana skuteczność taggerów.

* + 1. Tagger ręczny

Najprostszy tagger, polegający na ręcznym określeniu przynależności do części mowy danego wyrazu. Biorąc pod uwagę zasadę Pareto (będącą uogólnieniem prawa Zipfa), ręczne określenie części mowy dla stosunkowo niewielkiej ilości najczęściej występujących wyrazów może przynieść zaskakująco duże efekty. Nie zawsze jednak można jednoznacznie określić część mowy.

Ręczne określenie tagów odbywa się za pomocą przycisku ‘Set Tags Manually’. W okienku dialogowym pojawia się edytowalna lista słów, czytana z pliku o nazwie manualTaggingRules.txt. Po edycji listy w okienku następuje zapis do pliku o tej samej nazwie. W pliku tym każda linia zaczyna się tagiem, po którym następuje lista odpowiadających mu wyrazów, np:

ART l il i un una

ART jest symbolem części mowy – rodzajnika, a kolejne wyrazy to tokeny, które zostaną oznaczone jako rodzajniki.

Działanie taggera sprowadza się do przeparsowania wspomnianego pliku i stworzenia struktury – słownika, w której kluczem jest tag (część mowy), a wartością – zbiór tokenów. Następnie tagger iteruje po wszystkich tokenach i sprawdza ich przynależność do zbioru tokenów każdego taga. Jeśli token należy do zbioru, zostanie mu przypisany odpowiedni tag.

Zawartość pliku nie jest walidowana pod kątem opisania danego tokena różnymi tagami. Słownik tokenów jest tworzony w kolejności występowania w pliku, zatem token zostanie oznaczony takim tagiem, jaki pierwszy został przeczytany.

* + 1. Tagger wykorzystujący wyrażenia regularne

Podobnie jak w taggerze manualnym, reguły dla tagowania można edytować poprzez okno dialogowe, dostępne pod przyciskiem ‘Define Patterns’. Reguły są przechowywane w pliku o nazwie ‘RegexpTaggingRules.txt. Składnia pliku również jest podobna – każda linia zaczyna się symbolem taga, po czym następuje wyrażenie regularne, którego spełnienie kwalifikuje token do otrzymania danego taga. Po wyrażeniu opcjonalnie mogą wystąpić wyjątki od reguły, np:

*NOUN .+on[ei]$ buone buoni*

NOUN jest symbolem rzeczownika. Wyrażenie regularne oznacza, że każdy token kończący się na ciąg *–one* lub *–oni* zostanie uznany za rzeczownik, z wyjątkiem wyrazów *buone*, *buoni* (są to przymiotniki).

* + 1. Tagger składniowy

Do określenia taga danego wyrazu tagger składniowy wykorzystuje znane już tagi sąsiednich wyrazów lub wyrazy same w sobie. Reguły dla tego taggera znajdują się w pliku *syntaxTaggingRules.txt.* Edycja reguł jest taka sama jak w przypadku taggerów opisanych w 5.7.1 i 5.7.2. Pojedyncza reguła zawiera ciąg tagów lub wyrazów, z czego dokładnie jeden tag jest poprzedzony znakiem dolara – jest to szukany tag. Przykładowo, reguła:

ADJ $NOUN PUNCT

oznacza, iż wyraz następujący po przymiotniku i poprzedzający znak interpunkcyjny powinien otrzymać tag rzeczownika. Analogicznie można również określić wyrazy, mające występować w sąsiedztwie, np. reguła:

si $VERB

oznacza, że każdy (jeszcze nieoznaczony tagiem) wyraz następujący po słowe *si* jest czasownikiem.

Każda reguła może zawierać ciąg tagów lub wyrazów o dowolnej długości. Podczas parsowania linii odpowiadającej danej regule, w przypadku nienapotkania na tag rozpoczynający się od znaku dolara, reguła ta jest pomijana.

Ponieważ działanie tego taggera opiera się między innymi na już przypisanych tagach, zastosowanie taggera z regułami wykorzystującymi tylko i wyłącznie sąsiednie tagi (w przeciwieństwie do wyrazów) nie przyniesie żadnych rezultatów. Dlatego też, jeśli do tagowania korpusu nie zostaną wybrane inne taggery (manualny lub oparty na wyrażeniach regularnych), reguły taggera składniowego muszą być sprawdzone pod kątem zawierania wyrazów bądź tagów. Jeśli nie zostanie dostarczona taka reguła, która opiera się tylko i wyłącznie na sąsiadujących wyrazach, tagowanie korpusu nie zostanie wykonane (pojawi się stosowny komunikat). Jeżeli taka reguła istnieje, musi mieć największy priorytet (być pierwszą sprawdzaną regułą) podczas działania taggera.

* + 1. Tagger prawdopodobieństwa

Ostatnim zaimplementowanym przeze mnie taggerem jest tagger wykorzystujący inny, już otagowany korpus. Warunkiem zastosowania tego taggera jest więc uprzednie załadowanie wzorcowo otagowanego korpusu – służą do tego specjalne przyciski, opisane w podrozdziale 5.7.6

Na jego podstawie zostaje sporządzona mapa kojarząca każdy występujący w nim wyraz z listą tagów (i liczbą wystąpień), którymi został opisany. Następnie dla każdego wyrazu zostaje znaleziony tag, którym wyraz ten był tagowany najczęściej. Tak znaleziony zbiór par wyraz – tag zostaje użyty do otagowania właściwego korpusu. Oczywiście, nie wszystkie występujące w nim wyrazy muszą znajdować się w zbiorze, jednak zgodnie z wnioskami płynącymi z prawa Zipfa, znaczna większość wyrazów zostanie w ten sposób otagowana.

* + 1. Ocena rezultatów tagowania

Jakość rezultatów tagowania korpusu częściami mowy przez taggery – ich skuteczność oraz poprawność można poznać po każdym zastosowaniu dowolnego taggera.

Skuteczność rozumiana jako ogólny odsetek otagowanych słów jest prosta do określenia – wystarczy liczbę przypisanych tagów podzielić przez liczbę wszystkich tokenów i pomnożyć razy 100%. Wartość ta jest obliczana i wyświetlana po każdym zastosowaniu taggera.

Ocena poprawności, czyli informacja o ilości źle otagowanych wyrazów, wymaga naturalnie wiedzy o poprawnych tagach wyrazów w celu porównania ich z tymi przydzielonymi przez tagger. Dlatego, by poznać trafność taggerów, trzeba dokonywać tagowania na wzorcowo otagowanym korpusie, który można osobno załadować w aplikacji – proces ten jest opisany w podrozdziale 5.7.6. Po załadowaniu takiego korpusu należy wybrać opcję *‘Tag POS tagged corpus (error check)’* (w przeciwieństwie do domyślnej opcji ‘*Tag main corpus*’). W ten sposób, oprócz informacji o skuteczności taggera, pojawiają się także statystyki dotyczące jego poprawności – liczba i odsetek źle przydzielonych tagów, oraz uaktywnia się przycisk ‘*Wrong Tags’* umożliwiający wyświetlenie okienka z listą wyrazów źle otagowanych (wraz z tagiem przydzielonym i wzorcowym).

Tak przygotowane narzędzia pozwalają na samodzielne opracowywanie i testowanie zasad tagowania – znajdowanie wyrażeń regularnych typowych dla części mowy, czy też zależności składniowych pomiędzy konkretnymi częściami mowy.

* + 1. Korpus wzorcowo otagowany częściami mowy

Dla celów opisanych w podrozdziałach 5.7.4. i 5.7.5 – użycie taggera prawdopodobieństwa i sprawdzanie poprawności działania taggerów - w programie można załadować korpus wzorcowo otagowany częściami mowy. Plik z takim korpusem musi posiadać określoną strukturę: w każdej linii znajduje się osobny wyraz, po którym następuje jeden ze ściśle określonej listy tagów (lista ta jest dostępna w dodatku X). Linia nie spełniająca tego wzorca zostanie pominięta.

Wczytany korpus jest przechowywany jako lista par token – tag. Przed określaniem poprawności taggerów lista tokenów jest kopiowana, a tagi ustawiane na domyślne (tagiem „domyślnym” jest pusty string – brak taga).

* 1. Wyszukiwanie związków frazeologicznych
  2. Kontekst słów

Ostatnią zaimplementowaną przeze mnie funkcjonalnością jest parametryzowana wyszukiwarka kontekstu słowa w korpusie. Znajduje się ona w ostatniej zakładce zatytułowanej *‘Context’*.



Oprócz poszukiwanego słowa można także wybrać liczbę wyników oraz długość kontekstu (w słowach, w obie strony – szukany wyraz jest w środku kontekstu, zatem całkowita długość kontekstu będzie wynosić 2\*n + 1, gdzie n – podana długość kontekstu).

Lista wyników jest budowana w oparciu o klasę ConcordanceIndex biblioteki NLTK. Podczas pierwszego użycia wyszukiwarki kontekstu dla aktywnego korpusu sporządzany jest indeks wszystkich wyrazów (wywoływany jest konstruktor wspomnianej klasy, biorący jako parametr listę wyodrębnionych tokenów korpusu), wspomnieć o lambdzie zamieniającej na małe litery

a utworzony obiekt jest zapisywany dla dalszych wyszukiwań, jako pole składowe Modelu). Na istniejącym indeksie wywoływana jest metoda *offsets,* przyjmująca jako parametr poszukiwany token i zwracająca listę pozycji, na których znajduje się token. Bazując na liście wyników, do poszukiwanego wyrazu doklejana jest odpowiednia ilość wyrazów sąsiednich. W ten sposób powstaje lista kontekstów, która zostaje wyświetlona w osobnym oknie. Dla czytelności, szukany wyraz jest pisany dużymi literami.

przykład

1. Dodatek X

Tagset:

NOUN - rzeczownik

CONJ - spójnik

ADJ - przymiotnik

VERB - czasownik

ADV - przysłówek

PREP - przyimek

, ART - rodzajnik

, DPREP – zaimek z rodzajnikiem określonym

, NUM - liczebnik

, PRON - zaimek

, PRONVERB – czasownik z zaimkiem osobowym