Badanie metod ustalania autorstwa tekstu

(Methods of determining the authorship of a text)

Małgorzata Maciejewska

Praca inżynierska

Promotor: dr Paweł Rychlikowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

22 czerwca 2023

Streszczenie

Celem pracy jest zbadanie skuteczności wybranych klasyfikatorów do zadania ustalania autorstwa tekstów na zbiorach danych utworzonych z tekstów literackich wybranych pisarzy w języku polskim (zadanie klasyfikacji wieloklasowej). Przetestowane zostały różne rodzaje klasyfikatorów: od najprostszych, takich jak drzewo decyzyjne, las losowy i algorytm k najbliższych sąsiadów, przez probabilistyczne naiwne klasyfikatory Bayesowskie i SVC, klasyfikatory opierające się o językowe modele n-gramowe, po metody bardziej skomplikowane – sieci neuronowe. Jako narzędzie klasyfikacji sprawdzono także ChatGPT. Zbadano dokładność klasyfikatorów w zależności od sposobu reprezentacji tekstów (na przykład Bag of Words i Word Embedding), podziału zbiorów danych na zbiory uczące i testowe, w tym od wielkości próbki, oraz od rodzaju wstępnego przetworzenia zbioru danych. Najwyższe dokładności uzyskują naiwne klasyfikatory Bayesowskie, SVC i prosta sieć neuronowa, czyli metody oparte o reprezentacje tekstu zliczające tokeny, co jest związane ze stosunkowo niewielkim rozmiarem zbioru danych. Przeważnie dla długich próbek tekstowych uzyskiwane są najwyższe wyniki, a najskuteczniejszy rodzaj wstępnego przetworzenia tekstów zależy od rodzaju klasyfikatora. Kwestią poruszoną w pracy jest też zagadnienie wyjaśnialności działania modeli, w tym zbadanie działania narzędzia LIME.

The aim of the paper is to examine the operation of selected classifiers for the task of determining the authorship of texts on datasets created from Polish writers' literary texts (multiclass classification task). A few types of classifiers were tested: from the simplest ones, such as decision tree, random forest, and k nearest neighbors algorithm, through probabilistic naive Bayes classifiers and SVC, classifiers based on n-gram language models, to more complex methods - neural networks. ChatGPT was also tested as a classification tool. The accuracy of the classifiers was examined as a function of the method of the text representation (for example Bag of Words and Word Embedding), dividing the datasets into training and testing sets, including the size of the sample and the type of preprocessing of the data set. The highest accuracy is obtained by naive Bayes classifiers, SVC and a simple neural network, i.e. methods based on text representations that count tokens, and it is related to the relatively small size of the data set. Typically, long text samples yield the highest scores, and the most efficient type of text preprocessing depends on the type of classifier. An important point in this paper is also the problem of the explainability of models, including the examination of the LIME tool.

Spis treści

Rozdział 1.

Wprowadzenie

Zadanie ustalenia autorstwa tekstu na pozór może nie wydawać się ważnym zadaniem, bo w zdecydowanej większości autorzy podpisują swoje teksty, można jednak wskazać przynajmniej dwa obszary, w których nabiera ono istotności. Po pierwsze, w historii literatury znajdzie się niejeden przykład dzieła literackiego anonimowego lub opublikowanego pod pseudonimem i ustalenie autora teksu niejednokrotnie budzi spory wśród literaturoznawców, będąc dla nich zadaniem o niebagatelnym znaczeniu. W średniowieczu przeważnie nie podpisywano utworów, więc zadaniem ciekawym, lecz trudnym ze względu na niewielką ilość zachowanych tekstów z tego okresu, będzie szukanie zależności między tekstami. Sięgając do współcześniejszej literatury, można przywołać spory o opublikowane anonimowo utwory, takie jak *Pożegnanie* czy *Karylla*, które przypisane zostały Adamowi Mickiewiczowi, i część literaturoznawców autorstwo Mickiewicza potwierdza, część odrzuca, a część uznaje za wątpliwe [26]. Podobne dysputy toczą się m.in. o niektóre utwory Słowackiego i Kasprowicza.

Drugi obszar to pewne zainteresowania lingwistyki kryminalistycznej. Ustalanie autorstwa może być tu przydatne w takich zadaniach jak: identyfikacja autorów anonimów, w tym gróźb i żądań okupu, analiza tekstów pod kątem plagiatu, czy wykrywanie tzw. „trolli" internetowych piszących z różnych kont.

W pracy skupię się na pierwszej spośród omówionych perspektyw. Analiza komputerowa tekstów literackich może być dużą pomocą dla literaturoznawców próbujących ustalić sporne autorstwo. Zadanie ustalenia autora tekstu sprowadzę do zadania klasyfikacji wieloklasowej. Przetestuję różne rodzaje klasyfikatorów: od najprostszych, takich jak drzewo decyzyjne, las losowy i algorytm k najbliższych sąsiadów (rozdział 5.), przez probabilistyczne naiwne klasyfikatory Bayesowskie i SVC (rozdział 6.), klasyfikatory opierające się o językowe modele n-gramowe (rozdział 7.), po metody bardziej skomplikowane – sieci neuronowe (rozdział 8.). W rozdziale 9. przetestuję też, czy zyskujący popularność ChatGPT nadaje się do klasyfikowania tekstów literackich w języku polskim. Klasyfikatory będą uczyły się na zbiorach danych utworzonych z pobranych z biblioteki internetowej Wolne Lektury tekstów literackich w języku polskim pochodzących z XIX w. i pierwszej połowy XX w. Utworzę 3

zbiory danych różniące się objętością tekstów i liczbą klas: zbiór zawierający teksty prozatorskie 3 autorów i zbiory zawierające teksty liryczne 6 oraz 10 poetów. Zbiory te zostaną poddane różnym rodzajom obróbki wstępnej (preprocessing), takim jak anonimizacja lub usunięcie słów o małym znaczeniu – zbadam zależność skuteczności klasyfikatorów od rodzaju preprocessingu. Istotny będzie też sposób podziału zbiorów danych na zbiory uczące i testowe, a także liczba wyrazów w próbkach tekstu.

Warto zaznaczyć, że specyfiką zbiorów danych utworzonych z tekstów nieżyjących pisarzy jest ich ograniczoność. Tak jak na przykład przy zadaniach analizy wydźwięku recenzji filmowych lub określania tematów tekstów prasowych, zbory danych mogą być dowolnie duże, a nowe dane są generowane cały czas, tak w wypadku zbioru utworzonego z tekstów literackich, każda klasa jest ograniczona do tego, co napisał dany autor. Jeśli więc pisarz stworzył niewiele tekstów lub tylko nieliczne zachowały się do naszych czasów, klasa tego autora będzie zawierała mało tekstów, co może wpłynąć na wyniki klasyfikatorów.

Przy zadaniu klasyfikacji coraz częściej mówi się o wyjaśnialności modeli. Model przestaje być „czarną skrzynką", a istotnym pytaniem staje się: na jakiej podstawie model określił, że próbkę x należy zaklasyfikować do klasy A. Dla prostych klasyfikatorów interpretowalność może być uzyskana trywialnie, na przykład, aby dowiedzieć się, jak przebiegał proces klasyfikacji próbki przez drzewo decyzyjne, wystarczy wyświetlić to drzewo. Dla bardziej skomplikowanych klasyfikatorów powstały narzędzia do wyjaśnialności. Jednym z nich jest LIME, którego działanie zdecydowałam się przetestować. Sprawdzę też, jak zachowuje się ChatGPT przy zapytaniach o wyjaśnienie wyniku klasyfikacji.

Zadanie klasyfikacji tekstów jest jednym z zadań NLP, czyli przetwarzania języka naturalnego. Język naturalny to język stosowany do komunikacji przez ludzi w życiu codziennym, w tym także do pisania książek. Celem NLP jest umożliwienie komputerom zrozumienia, interpretacji, przetwarzania i generowania języka naturalnego w sposób zbliżony do ludzkiego, a zadaniami NLP, oprócz wspomnianej klasyfikacji tekstów, są między innymi rozpoznawanie mowy, tłumaczenie maszynowe, analiza wydźwięku (sentiment analysis) i rozpoznawanie nazw własnych.

Podsumowując, celem niniejszej pracy jest zbadanie działania wymienionych wyżej klasyfikatorów na zbiorach danych utworzonych z tekstów literackich w języku polskim w zależności od podziału zbiorów danych na zbiory uczące i testowe, w tym od wielkości próbki, oraz rodzaju wstępnego przetworzenia zbioru danych. Ważną kwestią poruszoną w pracy będzie zagadnienie wyjaśnialności działania modeli.

Kod projektu został napisany w języku Python (wersja 3.8.10), użyte zostały między innymi następujące biblioteki: *spacy*, *scikit-learn*, *keras*, *kenlm*, *numpy* oraz *typing*, natomiast „interfejs" projektu stanowią notatniki Google Colab. Wszystkie testy zostały przeprowadzone na Ubuntu w wersji 20.04.5 LTS (Focal Fossa) z dostępną pamięcią RAM wielkości 12.7GB. Kod projektu został umieszczony w serwisie GitHub w repozytorium `https://github.com/Gosia967/authorship_analysis_project`.

Rozdział 2.

Wstępna obróbka danych

Zbiór danych uczących został utworzony z darmowych tekstów książek pobranych z biblioteki internetowej Wolne Lektury (`wolnelektury.pl`) z wykorzystaniem dostępnego API (`wolnelektury.pl/api`), pozwalającego pobrać między innymi listy wszystkich dostępnych utworów, autorów, epok i rodzajów literackich. Istotnym zadaniem było oddzielenie autorów piszących w języku polskim od autorów niepolskojęzycznych, a tłumaczonych na język polski (niestety API nie zawiera informacji o oryginalnym języku tekstu, więc zadanie to trzeba było wykonać „ręcznie", oddzielając z listy wszystkich autorów twórców piszących po polsku). Mimo że w pracy zajmuję się klasyfikowaniem tylko kilkunastu autorów, to stworzony kod pozwala testować klasyfikatory także na tekstach innych autorów.

2.1. Wstępne przetworzenie tekstów

Surowy tekst pobrany z Wolnych Lektur może zawierać komentarz lub omówienie innego autora niż autor utworu. W celu zminimalizowania ryzyka takiej sytuacji napisałam skrypt, który usunął teksty lektur od zaczynających nową linię słów „Komentarz" lub „Omówienie" do końca pliku.

Żywy język, którym pisane są utwory literackie, zawiera pewną ilość redundantnych informacji z punktu widzenia przetwarzania tekstu metodami komputerowymi, na przykład w języku istnieją słowa o małym znaczeniu (stop words), do których zalicza się między innymi część przyimków (w, po, nad,...). Zdecydowałam się więc na kilka opcji preprocessingu tekstów, opisanych poniżej. Do tych zadań, które wymagały wykorzystania metod NLP, użyłam biblioteki *spacy* [12] z wytrenowanym dla języka polskiego przetwarzaniem potokowym (pipeline) *pl_core_news_lg*, której jedną z funkcjonalności jest dzielenie tekstu na tokeny i określanie dla każdego tokenu między innymi części mowy i formy podstawowej wyrazu. Preprocessing tekstów zawiera:

- usunięcie słów o małym znaczeniu (stop words) – użyłam domyślnej listy *stop_words* z biblioteki *spacy* dla języka polskiego zawierającej 381 wyrazów;
- usunięcie interpunkcji – skorzystałam z funkcji biblioteki *spacy* wyszczególniającej dla tokentów część mowy „PUNCT";
- zamianę na małe litery;
- lematyzację – wykorzystałam określanie formy podstawowej dla tokenu przez bibliotekę *spacy*;
- zostawienie tylko określonych części mowy – biblioteka *spacy* określa części mowy tokenów zgodnie z uniwersalnym taggowaniem (universal POS tagging [19]) – zdecydowałam się na pozostawianie tylko rzeczowników, czasowników i przymiotników;
- anonimizację – nie jest ona powiązana z redundancją, jednak niepożądane byłoby, aby modele rozpoznawały autorów poprzez „zapamiętanie" nazw własnych występujących w utworach. Z tego też powodu podczas tworzenia zbioru danych uczących i testowych, wszystkie teksty domyślnie są anonimizowane. Podczas anonimizowania skorzystałam z funkcji biblioteki *spacy* rozwiązującej zadanie znajdowania nazw własnych (NER – Named Entity Recognition [12]) dla danego tekstu.

Podczas badań przetestowałam wszystkie metody preprocessingu dla każdej z metod klasyfikacji, mając świadomość, że niektóre metody preprocessingu dla niektórych metod mają więcej sensu, a dla innych mniej (na przykład usunięcie wyrazów o małym znaczeniu wydaje się sensownym działaniem w wypadku metod opartych o reprezentacę Bag of Words, takich jak naiwne klasyfikatory Bayesowskie, natomiast może mieć mniej sensu dla modeli językowych). Jednym z celów tej pracy jest zbadanie skuteczności różnych klasyfikatorów dla różnych metod preprocessingu.

2.2. Zbiory danych

Opisany w poprzednim podrozdziale preprocessing jest kosztowny, dlatego też zdecydowałam się na preprocessing przy tworzeniu zbiorów danych, a nie podczas ładowania zbioru danych do modeli. Dla skrótowości opisu zastosowanych zadań preprocessingu wprowadzę następujące oznaczenia: A dla anonimizacji, **S** dla usunięcia słów o małym znaczeniu, **P** dla usunięcia interpunkcji, **M** dla lematyzacji, **O** dla pozostawienia tylko określonych części mowy i **L** dla zamiany na małe litery. Utworzyłam następujące zbiory danych:

- Zbiór składający się z tekstów epickich płodnych autorów pozytywistycznych: H. Sienkiewicza, B. Prusa oraz E. Orzeszkowej. Zbiór został poddany następującym metodom preprocessingu:

– A – teksty poddane wyłącznie anominimizacji zawierają w przybliżeniu tyle wyrazów, ile zawierały teksty wyjściowe. Tabela 2.1 zawiera liczbę wyrazów i znaków dla wszystkich autorów.

Autor	Liczba wyrazów	Liczba znaków
Eliza Orzeszkowa	$1.6 * 10^6$	$10.3 * 10^6$
Henryk Sienkiewicz	$2.1 * 10^6$	$13.3 * 10^6$
Bolesław Prus	$1.0 * 10^6$	$6.6 * 10^6$

Tabela 2.1: Liczba wyrazów i znaków dla poszczególnych autorów zbioru $\mathbf{E3}_A$

– ALS – usunięcie wyrazów mało znaczących powoduje zmniejszenie liczby wyrazów w tekstach o 37% dla Orzeszkowej i Prusa i o 40% dla Sienkiewicza. Warto zaznaczyć, że wyrazy mało znaczące są przeważenie krótkie – objętość znakowa tekstów spadła jedynie o 23% - 25%.

– ALO – teksty zredukowane do czasowników, rzeczowników i przymiotników bez usunięcia interpunkcji zawierają nieco mniej wyrazów niż w wypadku usunięcia tylko wyrazów mało znaczących (redukcja wyrazów o 41% dla Prusa, 42% dla Orzeszkowej i 45% dla Sienkiewicza).

– ALP – usunięcie interpunkcji redukuje liczbę znaków o 3% dla każdego z autorów.

– ALOMP – zbiór przy takim preprocessingu jest najbardziej ograniczony spośród wszystkich prezentowanych. Teksty – zredukowane do czasowników, rzeczowników i przymiotników w formach podstawowych – redukcja wyrazowa sięga 50%, znakowa $34 - 37\%$. Teksty Orzeszkowej zawierają $0.8 * 10^6$ wyrazów ($6.8 * 10^6$ znaków), Sienkiewicza $1.0 * 10^6$ wyrazów ($8.4 * 10^6$ znaków), a Prusa $0.5 * 10^6$ wyrazów ($4.4 * 10^6$ znaków).

Dla skrótowości i jasności w dalszej części pracy, zbiór ten oznaczymy jako **E3** (zbór tekstów epickich 3 autorów), a po uwzględnieniu preprocessingu zbiory będą oznaczane jako: $\mathbf{E3}_A$, $\mathbf{E3}_{ALS}$, $\mathbf{E3}_{ALO}$, $\mathbf{E3}_{ALP}$, i $\mathbf{E3}_{ALSPM}$.

- Zbiór składający się z tekstów lirycznych poetów barokowych: M. Sępa-Szarzyńskiego i E. Drużbackiej, romantycznych: A. Mickiewicza, J. Słowackiego i C. K. Norwida, pozytywistycznych: A. Asnyka i M. Konopnickiej oraz młodopolskich: K. Przerwy-Tetmajera, B. Leśmiana i J. Kasprowicza. Obfitość tekstów każdego autora jest rzędu 10^3 razy mniejsza niż w wypadku pierwszego zbioru. Podobnie jak w wypadku zbioru **E3**, zbiór ten ten został poddany preprocessingowi:

  – A – przy zastosowaniu wyłącznie anonimizacji tworzy się najmniej ograniczony zbiór tekstów. Tabela 2.2 przedstawia liczbę wyrazów i znaków dla poszczególnych autorów.

  – ALS – usunięcie wyrazów o małym znaczeniu zmniejsza liczbę wyrazów o 32% - 37%, a liczbę znaków o 18% - 22% (w zależności od autora).

Autor	Liczba wyrazów	Liczba znaków
Mikołaj Sęp Szarzyński	$6.1 * 10^3$	$41.5 * 10^3$
Elżbieta Drużbacka	$3.3 * 10^3$	$23.2 * 10^3$
Adam Mickiewicz	$26.0 * 10^3$	$174.0 * 10^3$
Juliusz Słowacki	$80.1 * 10^3$	$521.4 * 10^3$
Cyprian Kamil Norwid	$6.2 * 10^3$	$37.9 * 10^3$
Adam Asnyk	$10.3 * 10^3$	$70.9 * 10^3$
Maria Konopnicka	$39.8 * 10^3$	$260.0 * 10^3$
Kazimierz Przerwa-Tetmajer	$24.4 * 10^3$	$158.0 * 10^3$
Bolesław Leśmian	$43.9 * 10^3$	$278.4 * 10^3$
Jan Kasprowicz	$35.3 * 10^3$	$233.6 * 10^3$

Tabela 2.2: Liczba wyrazów i znaków dla poszczególnych autorów zbioru $\mathbf{L10}_A$

- ALO – ograniczenie wyrazów wyłącznie do czasowników, rzeczowników i przymiotników, bez usunięcia interpunkcji, zmniejsza liczbę wyrazów o 36% - 41%, a liczbę znaków o 22% - 30% (w zależności od autora).
- ALP – przy usunięciu interpunkcji można zaobserwować różnicę nie tylko między epiką a liryką – dla większości autorów powoduje to bowiem zmniejszenie liczby znaków o około 1.5% – ale również między autorami – dla poezji Asnyka liczba znaków zmniejsza się tylko o 0.8%, a dla Norwida aż o 4.1%.
- ALOMP – najmniejszy zbiór z największym ograniczeniem nałożonym na teksty, w wyniku którego teksty zawierają wyłącznie czasowniki, rzeczowniki i przymiotniki w formach podstawowych. Ograniczenie wyrazowe wynosi od 40% do 54%. Najmniejszą objętość mają teksty Drużbackiej $1.9 * 10^3$ wyrazów ($17.7 * 10^3$ znaków), największą teksty Słowackiego $45.4 * 10^3$ wyrazów ($394.3 * 10^3$ znaków), a najmniejszą większą od 10^4 wyrazów teksty Tetmajera ($14.0 * 10^3$ wyrazów i $121.2 * 10^3$ znaków).

Ten zbiór pozwoli na testowanie, jak zachowują się modele dla danych o niewielkiej objętości, ale zawierających dużo klas. Warto zauważyć, że zbiór taki jest bliższy rzeczywistemu zastosowaniu programów ustalających autorstwo – można spodziewać się, że odnaleziony anonimowy rękopis najlepiej sprawdzać w modelu zawierającym dużo potencjalnych klas. Dobranie autorów po 2-3 z konkretnych epok (baroku, romantyzmu, pozytywizmu i Młodej Polski) pozwoli również badać podobieństwa między tekstami z poszczególnych epok. Dla tego zbioru wprowadzam oznaczenie **L10** (teksty liryczne 10 autorów), a zbiory z uwzględnieniem preprocessingu będą oznaczane jako: $\mathbf{L10}_A$, $\mathbf{L10}_{ALS}$, $\mathbf{L10}_{ALO}$, $\mathbf{L10}_{ALP}$ i $\mathbf{L10}_{ALSPM}$.

Warto zauważyć, że objętość tekstów różni się znacząco między autorami – teksty Drużbackiej mają nieco ponad 3000 wyrazów, podczas gdy teksty Słowackiego zawierają ponad 80000 wyrazów. Spośród wszystkich autorów, których

teksty występują w zbiorze **L10** można wybrać tylko tych autorów, których objętość wyrazowa tekstów jest większa niż 20000, a po zastosowaniu najbardziej ograniczającego preprocessingu mają więcej niż 10000 wyrazów. Otrzymany w ten sposób zbiór oznaczę jako **L6** (po uwzględnieniu preprocessingu $\mathbf{L6}_A$, $\mathbf{L6}_{ALS}$, $\mathbf{L6}_{ALO}$, $\mathbf{L6}_{ALP}$ i $\mathbf{L6}_{ALSPM}$) i będzie on zawierał teksty liryczne sześciu poetów: Mickiewicza, Słowackiego, Konopnickiej, Tetmajera, Leśmiana i Kasprowicza.

2.3. Podział na zbiory uczące i testowe

Klasyfikatory na wejściu oczekują wektorów cech. Wektorów tych musi być na tyle dużo, aby model był w stanie na ich podstawie podejmować sensowne decyzje klasyfikacyjne, dlatego też teksty utworów zostaną podzielone na fragmenty, na podstawie których wyliczane będą wektory cech. Jednym z paramentów, których wpływ na modele będzie badany w rozdziałach 5., 6., 7., 8. jest więc sposób podziału zbiorów opisanych w podrozdziale 2.2. na próbki teksów oraz wielkość zbiorów uczących i testowych.

Dla każdego autora zbiór tekstów poddany odpowiednim preprocessingom podzieliłam na fragmenty zawierające jednakową liczbę wyrazów[1]. Zdecydowałam się na badanie następujących wielkości próbek tekstów: 15, 30, 50, 100, 300, 1000. Dla nabrania intuicji wspomnę, że za krótkie zdanie uznaje się posiadające do 20 wyrazów, bardzo długie zdania mogą zawierać ponad 100 wyrazów, natomiast przeciętna strona A4 tekstu to około 350 wyrazów.

Zbiory treningowe i testowe zawierają taką samą liczbę fragmentów dla każdego autora. Podczas podziałów starałam się maksymalnie wykorzystać dostępne dane dla autora o najmniejszej objętości wyrazowej dla każdego zbioru. Tabele 2.3, 2.3. i 2.5 przedstawiają liczby wyrazów we fragmentach i wielkości zbiorów treningowych i testowych dla zbiorów **E3**, **L10** i **L6**, rozumiane jako liczba wszystkich fragmentów w zbiorze, czyli aby otrzymać liczbę fragmentów dla pojedynczego autora, trzeba wielkość zbioru podzielić przez liczbę autorów dla tego zbioru (3 dla **E3**, 10 dla **L10** i 6 dla **L6**). Dla skrótowości zapisu w dalszej części pracy wprowadzam oznaczenia: przez $\mathbf{E3}_{train}$, $\mathbf{E3}_{test}$, $\mathbf{L10}_{train}$, $\mathbf{L10}_{test}$, $\mathbf{L6}_{train}$, $\mathbf{L6}_{test}$ rozumiem zbiór treningowy lub testowy z dowolnym preprocessingiem utworzony ze zbiorów **E3**, **L10**, **L6**, a n_{words} to liczba wyrazów we fragmencie.

[1] Dzieliłam teksty poszczególnych utworów według następującego algorytmu: dla utworu o liczbie wyrazów $d = n_{words}n + m$, gdzie n_{words} jest długością fragmentu, n liczbą naturalną taką, żeby zachodziła nierówność $0 \leq m \in \mathbb{N} < n_{words}$, dla $n = 0$ brałam cały utwór, dla $m < 0.1 * n_{words}$ brałam n fragmentów (fragment o długości m odrzucano), a dla $m \geq 0.1 * n_{words}$ brałam $n + 1$ fragmentów (w tym jeden fragment o długości m). Stąd liczba dostępnych fragmentów dla danego autora może być mniejsza niż liczba wyrazów podzielona przez liczbę znaków.

[2] Dla preprocessingu ALOMP teksty Drużbackiej miały zbyt małą objętość, więc wtedy $|\mathbf{L10}_{train}| = 480$.

n_{words}	$\lvert\mathbf{E3}_{train}\rvert$	$\lvert\mathbf{E3}_{test}\rvert$
30	3000	1200
300	3000	1200
1000	900	360

Tabela 2.3: Liczba wyrazów we fragmentach i wielkości zbiorów treningowych i testowych dla zbioru **E3**.

n_{words}	$\lvert\mathbf{L10}_{train}\rvert$	$\lvert\mathbf{L10}_{test}\rvert$
15	1000	200
30	500 [2]	100

Tabela 2.4: Liczba wyrazów we fragmentach i wielkości zbiorów treningowych i testowych dla zbioru **L10**.

n_{words}	$\lvert\mathbf{L6}_{train}\rvert$	$\lvert\mathbf{L6}_{test}\rvert$
50	1200	240
100	600	120
30	1800	360

Tabela 2.5: Liczba wyrazów we fragmentach i wielkości zbiorów treningowych i testowych dla zbioru **L6**.

2.4. Teksty spoza zbiorów E3, L6 i L10

Określenie skuteczności przy klasyfikowaniu zbioru testowego mówi dużo o modelu, jednak rzeczywiste wykorzystanie modelu jako narzędzia klasyfikacji (na przykład użytego w celu komputerowego wspomagania odpowiedzi na konkretne pytanie badawcze z zakresu literaturoznawstwa) będzie odbywać się na danych spoza zbioru treningowego i testowego. Dlatego też zdecydowałam się na wybór kilku tekstów spoza zbiorów **E3**, **L6** i **L10**. Teksty te posłużą też jako przykłady przy omawianiu wyjaśnialności klasyfikatorów. Wybrałam następujące teksty:

- *Do *** (Gdybym się zmienił w wstęgę złocistą...)* [22] – przez część badaczy uznawany za wiersz Mickiewicza, według innych autorstwo jest niepewne, utwór przypisywany jest również m.in. Norwidowi. Tekst zawiera 81 wyrazów. Jest to przykładowy tekst, gdy pytaniem badawczym będzie określanie autora tekstów o niepewnym autorstwie.

- *Juliusz Słowacki: Nieznane strofy „Beniowskiego"* [28] – pastisz utworu Słowackiego p.t. *Beniowski*[3] autorstwa K. Wyki. Zawiera 199 wyrazów. Teksty tego typu będą klasyfikowane, gdy rozważane będzie pytanie badawcze o podobieństwo pastiszu do oryginału i skuteczność naśladowania stylu wybranego

[3]Wolne Lektury klasyfikują *Beniowskiego* do epiki, mimo że jest to poemat dygresyjny, czyli gatunek synkretyczny. Wobec takiej klasyfikacji przez Wolne Lektury, *Beniowski* nie jest uwzględniony w zbiorze **L10**.

autora.

- List H. Sienkiewicza do M. Radziejewskiej z 3 marca 1903 [25]. Zawiera 703 wyrazy. Listy są przykładem tekstów, dla których można rozważać zależności między pisarstwem oficjalnym a prywatnym danego pisarza.

Rozdział 3.

Metody reprezentacji tekstu

Teksty ze zbioru danych trzeba zareprezentować w taki sposób, aby był on wspierany przez poszczególne algorytmy ML, co rozwiązuje się poprzez wyciągnięcie określonych cech z tekstów (feature extraction). Można rozważać różne kategorie cech, w niniejszej pracy będą to: cechy numeryczne – da się je numerycznie wyliczyć z tekstu (na przykład średnia liczba liter w wyrazie lub stosunek liczby rzeczowników do zaimków); cechy słownikowe – cechą jest obecność w tekście określonego tokenu (wyrazu, fragmentu wyrazu lub grupy wyrazów); oraz cechy odkrywane, które zostają wykryte na przykład przez sieć neutronową.

Proces przekształcania zbioru dokumentów tekstowych w numeryczne wektory cech nazywany jest wektoryzacją. Poniżej przedstawione zostały reprezentacje tekstu wykorzystywane przez modele klasyfikujące.

3.1. Wybór cech tekstu (FOT – feature of text)

Wektoryzacja poprzez wybór określonych numerycznych cech z tekstu jest metodą bardzo intuicyjną – trzeba znaleźć charakterystyczne dla tekstu cechy (na przykład liczba rzeczowników w tekście) i zamienić je na liczby. W naszej realizacji omawianej reprezentacji każda próbka tekstu ze zbioru danych (wydzielona w sposób opisany w podrozdziale 2.3.) jest reprezentowana przez wektor 27 liczb wyliczonych z fragmentu tekstu. Pod uwagę wzięto cechy ilościowe, interpunkcyjne i gramatyczne wymienione w tabeli 3.1. Do określania części mowy pomocne okazało się tagowanie POS udostępnione przez bibliotekę *spacy*, opisane w podrozdziale 2.1.. Ostateczny format wektora cech uzyskałam za pomocą wektoryzatora *DictVectorizer* z pakietu *scikit-learn*.

[1]Liczba słów dla ustalonej z góry liczby wyrazów we fragmencie (podrozdział 2.3.) będzie taka sama dla wszystkich tekstów w zbiorze, więc nie będzie miała znaczenia, jednak jeśli chcielibyśmy badać fragmenty o różnej długości wyrazowej, będzie to sensowna cecha.

Grupa cech	Cecha
ilościowe	liczba znaków
	liczba słów[1]
	liczba zdań
	średnia liczba znaków w słowie
	średnia liczba słów w zdaniu
	średnia liczba znaków w zdaniu
	stosunek liczby różnych słów do liczby wszystkich słów
interpunkcyjne	liczba przecinków
	liczba znaków interpunkcyjnych
	średnia liczba znaków interpunkcyjnych w zdaniu
	średnia liczba przecinków w zdaniu
gramatyczne	liczba przymiotników
	liczba czasowników
	liczba rzeczowników
	liczba spójników
	liczba przysłówków
	liczba zaimków
	liczba nazw własnych
	stosunek liczby przymiotników do liczby rzeczowników
	stosunek liczby nazw własnych do liczby rzeczowników
	stosunek liczby zaimków do liczby rzeczowników
	stosunek liczby rzeczowników do liczby czasowników
	stosunek liczby przysłówków do liczby czasowników
	średnie liczby przymiotników w zdaniu
	średnie liczby rzeczowników w zdaniu
	średnie liczby czasowników w zdaniu
	średnie liczby nazw własnych w zdaniu

Tabela 3.1: 27 cech wybranych z tekstu.

3.2. Reprezentacje zliczające tokeny

Omówione w tym rozdziale reprezentacje będą opierały się o cechy słownikowe, czyli będą wykrywały w tekstach obecność konkretnych tokenów.

3.2.1. Bag of Words (BOW)

Aby otrzymać tekst w reprezentacji Bag of Words, trzeba najpierw przeprowadzić tokenizację surowego tekstu, a następnie zliczyć wystąpienia każdego tokenu w dokumencie. Z pomocą przychodzi wektoryzator *CountVectorizer* z pakietu *scikit-learn*, który przeprowadza tokenizację tekstu, dzieląc tekst na słowa (przy użyciu bia-

łych znaków z ignorowaniem interpunkcji), zlicza tokeny i (ewentualnie) normalizuje macierz. Warto zaznaczyć, że dokumenty mają zwykle małą liczbę słów w stosunku do wszystkich słów użytych w korpusie, więc macierze reprezentujące zbiór tekstów, będą rzadkie. *CountVectorizer* radzi sobie z tym problemem, używając odpowiedniej reprezentacji (sparce reprezentation z pakietu *scipy.sparse*) [21].

3.2.2. Bag of n-grams

N-gramy to sekwencje słów o długości n. Zamiast zliczać wyłącznie pojedyncze słowa, można zliczać również dłuższe szeregi wyrazów. W pracy zdecydowałam się na reprezentację bigramową, która zlicza ciągi wyrazów o długościach 1 i 2, i trigramową zliczającą sekwencje o długościach 1, 2 i 3. Wspomniany w podrozdziale 3.2.1. *CountVectorizer* umożliwia zliczanie n-gamów.

3.2.3. Fragmenty słów (Bag of Word Pieces)

Można postawić pytanie: czy tokenizacja na pojedyncze słowa jest jedyną sensowną opcją tokenizacji, czy może wyrazy dałoby się podzielić na mniejsze części? Intuicja stojąca za tym pytaniem leży w słowotwórstwie, przykładowo słowo *kotek* składa się z dwóch cząstek niosących znaczenie: podstawy słowotwórczej *kot-* oraz formantu *-ek*, który tworzy wyraz o innym znaczeniu niż wyraz podstawowy. W porównaniu do reprezentacji Bag of Words opisanej w podrozdziale 3.2.1. reprezentacja dzieląca słowa na mniejsze fragmenty wymaga innego tokenizatora.

BERT, czyli jeden z najpopularniejszych transformerów[2] do szerokiego zakresu uczenia maszynowego związanego z przetwarzaniem języków, używa tokenizatora *WordPieces* [16]. Powstało kilka polskich wersji BERT-a, jedną z nich jest wersja Polbert [18]. Zawiera ona wytrenowany tokenizator, z którego zdecydowałam się skorzystać. Tabela 3.2 przedstawia podział kilku przykładowych wyrazów przez tokenizator Polbert – zauważmy, że podział tokenizatora nie zawsze pokrywa się z podziałem słowotwórczym.

Podzielone tokenizatorem teksty zostały zamienione na wektory przy pomocy wektoryzatora *CountVectorizer* z pakietu *scikit-learn* i w ten sposób powstała reprezentacja Bag of Word Pieces.

[2]Transformery to typ sieci neuronowych o architekturze *Transformer*. Używane są do zadań związanych z przetwarzaniem tekstu, a ich główną przewagą nad innymi architekturami jest wykorzystanie mechanizmu uwagi, który określa istotność słowa w kontekście. Architektura *Transformer* została opisana w pracy *Attention is all you need* autorstwa A. Vaswaniego i innych [27].

Wyraz	Podział tokenizatora Polbert
kotek	kot | ek
piesek	piesek
wyimaginowany	wy | ima | gin | owany
najfajniejszy	naj | faj | niejszy
programuję	programu | ję
języki	języki

Tabela 3.2: Podział tokenizatora Polbert kilku przykładowych wyrazów.

3.3. Wektory osadzeń słów (Word Embedding)

Word Embedding należy do kategorii reprezentacji tekstu, w których cechy są odkrywane przez sieci neuronowe i polega na reprezentowaniu poszczególnych słów jako gęstych wektorów. Wektory te są trenowane: semantyka słów jest mapowana na geometryczną przestrzeń nazywaną przestrzenią osadzeń (embedding space). O wektorach osadzeń można myśleć jako o „znaczeniach" słów, a transformacje pozwalają przechodzić od jednych słów do innych – dla nabrania intuicji podam przykład: w pewnej przestrzeni osadzeń różnica między wektorem słowa „Warszawa" a wektorem słowa „Polska" będzie taka sama, jak między wektorem słowa „Berlin" i wektorem słowa „Niemcy". Są dwie możliwości uzyskania wektorów osadzeń: trenowanie tych wektorów podczas trenowania modelu sieci neuronowych albo skorzystanie z wytrenowanych wektorów osadzeń. W pierwszym wypadku do tokenizacji i wektoryzacji tekstów użyłam tokenizatora *Tokenizer* z biblioteki *keras*, który pozwala na ograniczenie zbioru słów do n najczęściej występujących wyrazów, następnie zamieniłam teksty na listy całkowitoliczbowych sekwencji za pomocą metody tokenizatora *texts_to_sequences* i przystosowałam je do formy oczekiwanej przez sieć neuronową (dwuwymiarowy tensor zamiast listy list) za pomocą funkcji *pad_sequences*. Do nauczenia przestrzeni osadzeń skorzystałam z warstwy *Embedding* modelu *Sequential* z biblioteki *Keras*, co zostanie dokładniej opisane w podrozdziale 8.1.. W drugim wypadku skorzystałam z polskiej wersji *GloVe* – metody opracowanej przez Stanford NLP Group [13].

Rozdział 4.

Metody oceny poprawności modeli

Ważnym etapem przy tworzeniu klasyfikatorów jest ocena ich poprawności, bo pozwala ona ocenić ich skuteczność i porównywać między sobą. Istnieje wiele różnych metod oceny poprawności klasyfikatorów. W niniejszej pracy zdecydowałam się na dokładność, macierz pomyłek i współczynnik zgodności κ.

Dokładność (accuracy)

Dokładność określa stopień zgodności wyników klasyfikatora z prawdziwymi wartościami. Jest to stosunek poprawnych klasyfikacji do wszystkich klasyfikacji. Dla przewidywanych przez klasyfikator wartości klas dla wszystkich n badanych próbek $c_{pred} = (c_1, c_2, \ldots, c_n)$ i prawdziwych wartości klas $c_{true} = (t_1, t_2, \ldots, t_n)$ dokładność wyraża się wzorem:

$$acc = \frac{1}{n} \sum_{i=1}^{n} I(c_i = t_i) \tag{4.1}$$

gdzie I jest funkcją charakterystyczną (indicator function).

Macierz pomyłek (confussion matrix)

Macierz pomyłek szczegółowiej niż dokładność pokazuje wyniki klasyfikatora. Ma wymiar $n_c \times n_c$, gdzie n_c to liczba klas. Pole (i, j) w macierzy pomyłek to liczba próbek, których prawdziwa klasa to i, a przewidziana przez klasyfikator to j [21]. Poprawna klasyfikacja to przekątna macierzy. Na podstawie macierzy pomyłek można określić dla każdej klasy liczbę wyników prawdziwych (odpowiednikiem jest true positive [TP] dla klasyfikacji binarnej), sumę wyników nadmiarowo przypisanych do klasy (odpowiednikiem w klasyfikacji binarnej byłby false positive [FP]) oraz sumę wyników tej klasy przypisanych do innych klas (odpowiednik dla klasyfikacji binarnej to false negative [FN]).

Współczynnik zgodności κ (Cohen's Kappa)

Miara ta pozwoli łatwiej porównywać wyniki między różnymi zbiorami i klasyfikatorami. Współczynnik κ przyjmuje wartości z przedziału $[-1; 1]$, 0 oznacza klasyfikator losowy, a wartości ujemne informują, że klasyfikator jest gorszy od losowego. Wyniki powyżej 0.8 uważane są za dobre [21]. Wzór na współczynnik κ to:

$$\kappa = \frac{acc - acc_{random}}{1 - acc_{random}} \tag{4.2}$$

Można go też wyliczyć korzystając z wartości z macierzy pomyłek C:

$$\kappa = \frac{nd - \sum_{i=j}^{m} C_{i.} C_{.j}}{n^2 - \sum_{i=j}^{m} C_{i.} C_{.j}} \tag{4.3}$$

gdzie d to liczba poprawnych klasyfikacji, $C_{i.}$ to suma wiersza macierzy, a $C_{.j}$ jest sumą kolumny.

Dokładność klasyfikatora losowego dla poszczególnych zbiorów

Najprostsze klasyfikatory to klasyfikator losowy i klasyfikator wybierający najliczniejszą klasę. W naszym wypadku klasyfikatory te mają taką samą oczekiwaną dokładność, bo zbiory są idealnie zrównoważone (podrozdział 2.3.). Do klasyfikatora losowego będą porównywane omawiane w pracy klasyfikatory, dlatego przedstawię ich dokładność: zbiór **L10** ma 10 autorów, więc oczekiwana dokładność klasyfikatora losowego na tym zbiorze to 10%; zbiór **L6** ma 6 autorów, stąd oczekiwana dokładność klasyfikatora losowego na tym zbiorze to 16.7%; zbiór **E3** zawiera 3 autorów, czyli oczekiwana dokładność klasyfikatora losowego na tym zbiorze to 33.3%.

Rozdział 5.

Klasyfikacja autorstwa oparta o reprezentację FOT

Metody opisane w poniższym rozdziale będą korzystały z reprezentacji wyboru cech tekstu FOT opisanej w rozdziale 3.1.. Wektory samodzielnie wybranych cech podanych w tabeli 3.1 mają małą i stałą długość (27 liczb), więc będą dobrze działały z tymi modelami, które wymagają wektorów niewielkiej długości. Modelami klasyfikującymi będą: drzewo decyzyjne, las losowy i algorytm k najbliższych sąsiadów. Badana będzie dokładność modeli na zbiorach **E3**, **L10** i **L6** w zależności od preprocessigu opisanego w 2.2. oraz sposobu podziału zbioru na próbki tekstów opisanego w podrozdziale 2.3..

5.1. Teoretyczny opis metod

5.1.1. Model drzewa decyzyjnego (Decision Tree Classifier – DTC)

Drzewa decyzyjne (DT) to nieparametryczna metoda uczenia nadzorowanego używana między innymi do zadania klasyfikacji. Celem jest stworzenie modelu, który przewiduje wartość klasy, ucząc się prostych reguł decyzyjnych wywnioskowanych z cech danych. Drzewo buduje gałęzie (mogą być one traktowane jako instrukcje *if-else*), które dzielą zbiór danych na podzbiory na podstawie najistotniejszych cech, a ostateczną klasyfikację reprezentują liście drzewa [9].

Niewątpliwą zaletą tej metody jest jej prostota oraz łatwość interpretacji i wizualizacji, a także logarytmiczny koszt predykcji. Drzewa mają jednak tendencję do overfittingu dla danych o dużej liczbie cech a małej liczbie próbek, który objawia się zbytnim skomplikowaniem drzew, powodującym brak dobrej klasyfikacji danych. Problemem może być też niestabilność – dla niewiele różniących się danych mogą być generowane zupełnie różne drzewa, co łagodzi się, stosując zespoły drzew. Warto też wspomnieć o problemie budowania optymalnego drzewa, który jest NP-zupełny, co

powoduje, że algorytmy uczenia drzewa często są oparte na algorytmach heurystycznych, podejmujących lokalnie optymalne decyzje w każdym węźle, wzmacniając je jeszcze uczeniem wielu drzew w zespole uczącym [21].

Czasowa złożoność skonstruowania zbalansowanego drzewa to w ogólności $O(n_s n_f \log n_s)$, a przeszukiwania $O(\log n_s)$, gdzie n_s jest liczbą próbek, a n_f liczbą cech, jednak takie drzewo nie zawsze będzie zbalansowane. Aby poddrzewa były zawsze zbalansowane, w każdym wierzchołku trzeba wykonać przeszukanie w celu znalezienia cechy, która pozwoli utrzymać najlepszy balans, co powoduje, że koszt uczenia wzrośnie do $O(n_f n_s^2 \log n_s)$[21].

W pracy skorzystałam z modelu *DecisionTreeClasiffier* z pakietu *scikit-learn*. Model ten używa zoptymalizowanej wersji algorytmu CART (Classification and Regression Trees).

5.1.2. Model lasu losowego (Random Forest Classifier – RFC)

Las losowy należy do metod uczenia zespołowego (*ensemble learning*). Model ten buduje kolekcję prostych drzew decyzyjnych, a następnie ją uśrednia. W ten sposób można osiągnąć znaczną poprawę dokładności klasyfikacji w stosunku do pojedynczego drzewa decyzyjnego. Breiman [1] przedstawia następującą definicję:

Definicja 1 *Las losowy to klasyfikator składającym się z kolekcji klasyfikatorów o strukturze drzewiastej* $\{h(\boldsymbol{x}, \Theta_k), k = 1, ...\}$*, gdzie* $\{\Theta_k\}$ *są losowymi wektorami o niezależnych i jednakowych rozkładach (IID), a każde drzewo oddaje jednostkowy głos na najbardziej popularną klasę* $\mathbf{x}$.

Metoda lasu losowego czerpie z baggingu (*bootstrap aggregating*), czyli metody uczenia zespołowego, która do uczenia prostych klasyfikatorów dobiera losową próbę ze zwracaniem w taki sposób, że każdy element ma takie samo prawdopodobieństwo pojawienia się w nowym zbiorze danych, a następnie każdy prosty model uczony jest niezależnie. Bagging pozwala na redukcję wariancji estymowanej funkcji predykcyjnej i działa szczególnie dobrze dla procedur o wysokiej wariancji i niskim obciążeniu (bias), takich jak drzewa. Las losowy poprawia redukcję wariancji osiąganą w wyniku baggingu poprzez zmniejszenie korelacji między drzewami, bez zbytniego zwiększania wariancji. Osiąga się to przez losowy wybór zmiennych podczas tworzenia drzewa [10, s. 587-588].

Złożoność czasowa uczenia lasu losowego to $O(k n_s n_f \log(n_s))$, gdzie k to liczba prostych drzew decyzyjnych, n_s jest liczbą próbek, a n_f liczbą cech.

W pracy skorzystałam z modelu *RandomForestClasiffier* z pakietu *scikit-learn*. Warto podkreślić, że implementacja z tego pakietu różni się od wersji oryginalnej opisanej przez Breimana [1] – zamiast pozwolić każdemu klasyfikatorowi głosować

na pojedynczą klasę, łączy ona klasyfikatory poprzez uśrednienie ich prognoz probabilistycznych [21].

5.1.3. Metoda k najbliższych sąsiadów (KNN)

Metoda k najbliższych sąsiadów nie konstruuje modelu, a jedynie przechowuje instancje danych treningowych i na ich podstawie wyznacza odpowiednią klasę danych testowych. Klasyfikacja danej próbki polega na znalezieniu k najbliższych według określonej metryki sąsiadów i znalezieniu wśród nich najbardziej popularnej klasy.

W pracy skorzystałam z modelu *KNeighborsClassifier* z pakietu *scikit-learn*. W tej implementacji domyślną metryką do obliczania odległości między próbami jest metryka Minkowskiego[1]. Na złożoność obliczeniową tej metody wpływa dobór algorytmu obliczającego odległość między próbkami. Dostępne są 3 algorytmy: *brute force* obliczający odległość między każdą parą punktów o złożoności $O(n_f n_s^2)$, gdzie n_f jest liczbą cech (w naszym wypadku wynosi 27), a n_s liczbą próbek, algorytm *K-D Tree*, który nie oblicza dokładnego dystansu między punktami, korzystając z założenia, że jeśli punkt A jest blisko punktu B, a B daleko od punktu C, to punkty A i C są daleko, dzięki czemu osiąga złożoność $O(n_f n_s \log(n_s))$, oraz algorytm *Ball Tree*, który dzieli dane na hipersfery, co powoduje, że konstrukcja drzewa jest bardziej kosztowna niż dla algorytmu *K-D Tree*, ale jest ono bardziej wydajne dla danych o dużych wymiarach. Ponieważ liczba cech d jest większa od 15, model wybierze algorytm *brute force* (autorzy modelu uzasadniają to tym, że wymiarowość danych powyżej 15 jest zbyt wysoka dla drzew)[21].

5.2. Omówienie uzyskanych wyników

Przeprowadziłam testy modeli KNN, DTC, i RFC, ucząc je na zbiorach **L10**, **L6** i **E3** poddanych kolejno wszystkim preprocessingom opisanym w podrozdziale 2.1.. Tabele 5.1, 5.2. i 5.3 przedstawiają wybrane (najlepsze i najgorsze) wyniki eksperymentów (dokładność oraz współczynnik κ). We wszystkich tabelach kolorem niebieskim wyróżniłam najwyższe, a pomarańczowym najniższe rezultaty uzyskane przez klasyfikatory DTC, RFC i KNN. Ponadto kolorami zielonym i czerwonym zaznaczyłam najlepszy i najgorszy rezultat dla każdego podziału prezentowanego w tabeli. Dla KNN najpierw wyznaczyłam dokładność dla różnych podziałów zbioru dla $k = 10$, a następnie dla najlepszego podziału i preprocessingu, wyznaczyłam

[1]Metryka Minkowskiego jest uogólnieniem odległości Euklidesowej. Odległość między dwoma punktami $X = (x_1, \ldots, x_n) \in \mathbf{R}^n$ i $Y = (y_1, \ldots, y_n) \in \mathbf{R}^n$ jest zdefiniowana jako

$$D(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

.

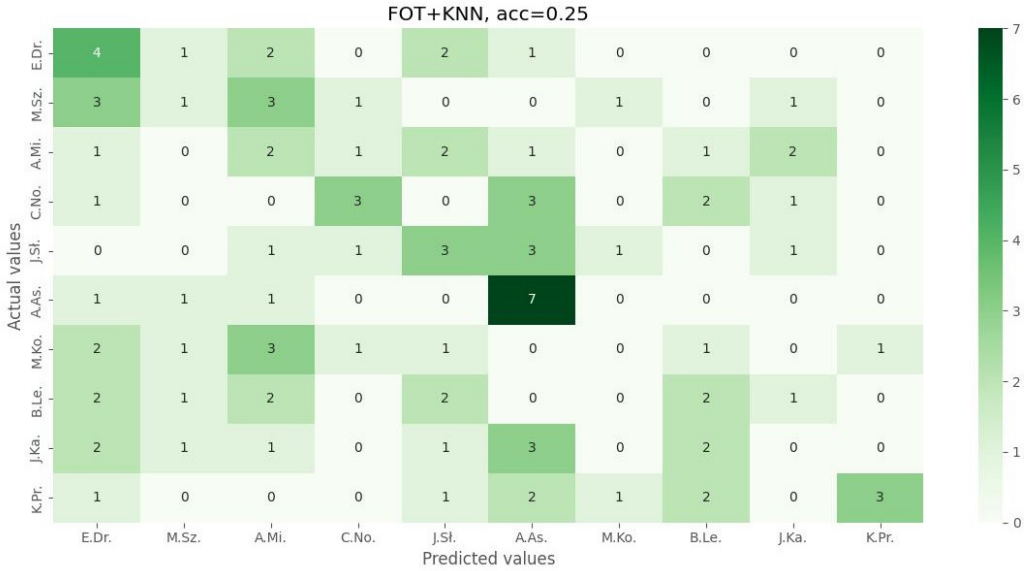
współczynnik k poprzez próbkowanie kilkudziesięciu wybranych wartości (granicą górną jest oczywiście wielkość zbioru testowego).

Zbiór L10

Dla zbioru **L10** wybrane wyniki przeprowadzonych testów przedstawia tabela 5.1. Przypomnijmy, że dla $n_{words} = 15$ zbiór treningowy zawierał 1000 próbek, a testowy 200, a dla $n_{words} = 30$ wielkość zbioru treningowego to 500, a testowego 100. W wypadku tego zbioru żaden z modeli nie jest wyraźnie lepszy od klasyfikatora losowego – wartość współczynnika κ nie przekracza 0.19, a w jednym uczeniu wyszła nawet ujemna. Największą dokładność, wynoszącą 27%, osiągnął RFC, natomiast DTC i KNN osiągają nieco niższą maksymalną dokładność – odpowiednio 22% oraz 25%. Watro zwrócić uwagę na dużą różnicę wyników między podziałem, w którym fragmenty zawierają 30 wyrazów, a takim, w którym fragmenty mają 15 wyrazów – w drugim wypadku maksymalna wartość współczynnika κ jest mniejsza niż 0.1, przy maksymalnej dokładności równej 18.5%. Macierz pomyłek modelu KNN dla zbioru $\mathbf{L10}_{ALO}$ i $n_{words} = 30$, czyli podziału, dla którego model KNN osiąga najwyższe rezultaty, przedstawiona na rysunku 5.1, nie ma wyraźnie zaznaczonej przekątnej i wygląda losowo, a na stosunkowo wysoki wynik wpłynęło rozpoznawanie klasy Asnyka (7 z 10 próbek klasy Asnyka zostało rozpoznanych poprawnie). Warto zaznaczyć, że dla zbioru **L10** wszystkie macierze pomyłek wyglądają podobnie do przedstawionej macierzy – widoczna jest spora losowość wyników.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	30	RFC	**0.27**	**0.189**
ALS	15	RFC	0.185	0.094
ALS	30	DTC	0.08	−0.022
ALO	30	KNN	0.25	0.167
ALP	15	DTC	0.115	0.017
ALP	15	RFC	0.13	0.033
ALP	15	KNN	0.135	0.039
ALP	30	DTC	0.22	0.133

Tabela 5.1: Dokładność modeli opartych o reprezentację FOT dla różnych podziałów zbiorów z grupy zbiorów **L10**. Dla $n_{words} = 15$ zbiór treningowy zawierał 1000 próbek, a testowy 200, a dla $n_{words} = 30$ wielkość zbioru treningowego to 500, a testowego 100. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia tylko wybrane wyniki.



Rysunek 5.1: Macierz pomyłek modelu KNN dla zbioru $\mathbf{L10}_{ALO}$. Wielkość zbioru treningowego to 500, wielkość zbioru testowego to 100, a n_{words} =30. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Wyniki nie są zbyt dobre – macierz wygląda losowo.

Zbiór L6

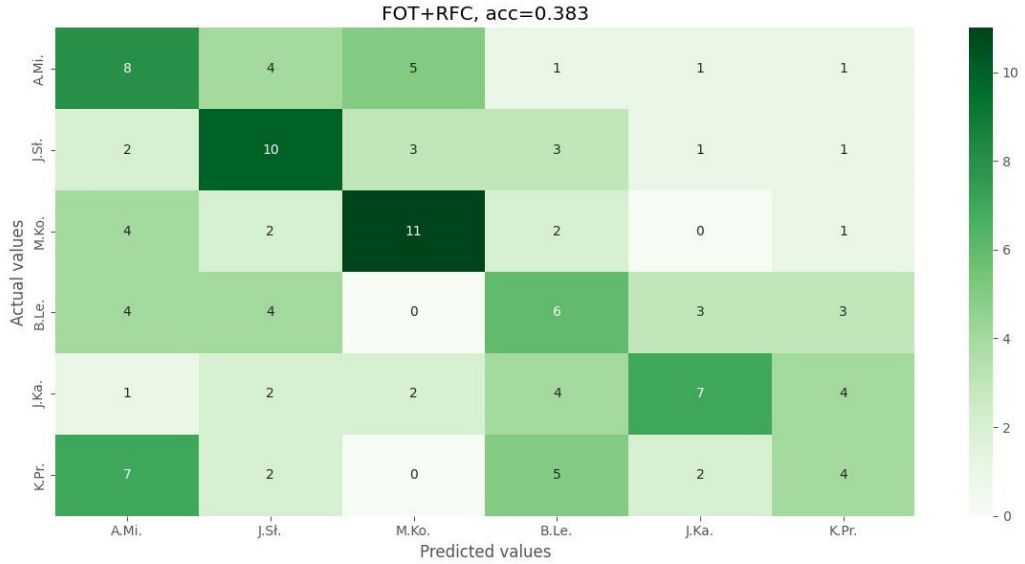
Nieco wyższe wyniki[2], przedstawione w tabeli 5.2., osiągane są dla zbioru **L6**, dla którego maksymalna wartość współczynnika κ wynosi 0.26, a maksymalna dokładność to 38% dla RFC, 32.5% dla DTC i 34% dla KNN. Współczynnik κ dla podziału, w którym fragmenty mają 30 wyrazów, osiąga podobne wartości, jak dla zbioru **L10**. Zauważmy, że usunięcie interpunkcji (preprocessingi ALP i ALOMP) przeważnie negatywnie wpływa na dokładność uzyskiwaną przez model. Rysunek 5.2 przedstawia macierz pomyłek modelu RCF dla $\mathbf{L6}_{ALS}$. Najlepiej rozpoznawana jest klasa Konopnickiej, widać delikatnie zarysowaną przekątną, można też zauważyć ciemniejszy kwadrat 3x3 (Leśmian, Kasprowicz, Przerwa-Tetmajer) wyróżniający poetów młodopolskich oraz dość częste przypisywanie tekstów Przerwy-Tetmajera do klasy Mickiewicza[3]. Lekko zarysowana przekątna charakteryzuje sporo macierzy pomyłek dla zbioru **L6**, czasem można też zaobserwować częstsze mylenie poetów z jednej epoki.

[2]Wyższe wyniki dla zbioru L6 nie są oczywiście zaskoczeniem, bo przy mniejszej liczbie klas można spodziewać się większych liczb.

[3]Mylenie klas przez RCF może wskazywać na podobieństwo między twórczością Mickiewicza i Tetmajera, a problem, czy takie podobieństwo w rzeczywistości istnieje, mógłby stać się przedmiotem dyskusji literaturoznawców.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	50	RFC	0.371	0.245
A	30	RFC	0.314	0.177
ALS	100	DTC	0.325	0.19
ALS	100	RFC	**0.383**	**0.26**
ALP	50	DTC	0.196	0.035
ALP	100	DTC	0.183	0.02
ALP	100	KNN	0.342	0.21
ALOMP	30	DTC	0.181	0.017
ALOMP	30	RFC	0.217	0.06
ALOMP	30	KNN	0.211	0.053

Tabela 5.2: Dokładność modeli opartych o reprezentację FOT dla różnych podziałów zbiorów z grupy zbiorów **L6**. Dla $n_{words} = 30$ zbiór treningowy zawierał 1800 próbek, a testowy 360, dla $n_{words} = 50$ wielkość zbioru treningowego to 1200, a testowego 240, a dla $n_{words} = 100$ odpowiednio 600 i 120. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.



Rysunek 5.2: Macierz pomyłek modelu RFC dla zbioru $\mathbf{L6}_{ALS}$. Wielkość zbioru treningowego to 600, wielkość zbioru testowego to 120, a n_{words} =100. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Macierz jest wyraźnie mniej losowa niż dla zbioru **L10**.

Zbiór E3

Uczenie modeli dla zbioru **E3** daje wyraźnie wyższe wyniki, niż dla dwóch poprzednich zbiorów, szczególnie gdy $n_{words} = 1000$. Wyższe wyniki są spodziewane, bo E3 ma najmniej klas, ponadto zbiór ten zawiera najwięcej danych dla każdej

klasy. Maksymalna wartość κ przekracza 0.6 dla DTC, 0.7 dla KNN i 0.8 dla RCF, a dokładność osiąga nawet 74% dla DTC, 82% dla KNN i 88% dla RCF. Niezależnie od podziału najgorsze wyniki osiągane są dla preprocessingu ALP. Dla podziału, w którym $n_{words} = 30$, współczynnik κ przyjmuje wartości podobne jak dla zbiorów **L10** i **L6**, chociaż wartość maksymalna jest nieco wyższa (0.245), a jednokrotnie jest on mniejszy od 0. W macierzy pomyłek modelu RFC i $n_{words} = 1000$ przedstawionej na rysunku 5.3, widać wyraźną przekątną.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	1000	DTC	0.744	0.617
A	1000	RFC	**0.883**	**0.825**
A	1000	KNN	0.822	0.733
ALS	300	RFC	0.7	0.55
ALO	30	RFC	0.497	0.245
ALP	30	DTC	0.324	−0.014
ALP	30	RFC	0.379	0.069
ALP	30	KNN	0.387	0.08
ALP	300	DTC	0.475	0.213
ALP	1000	DTC	0.644	0.467

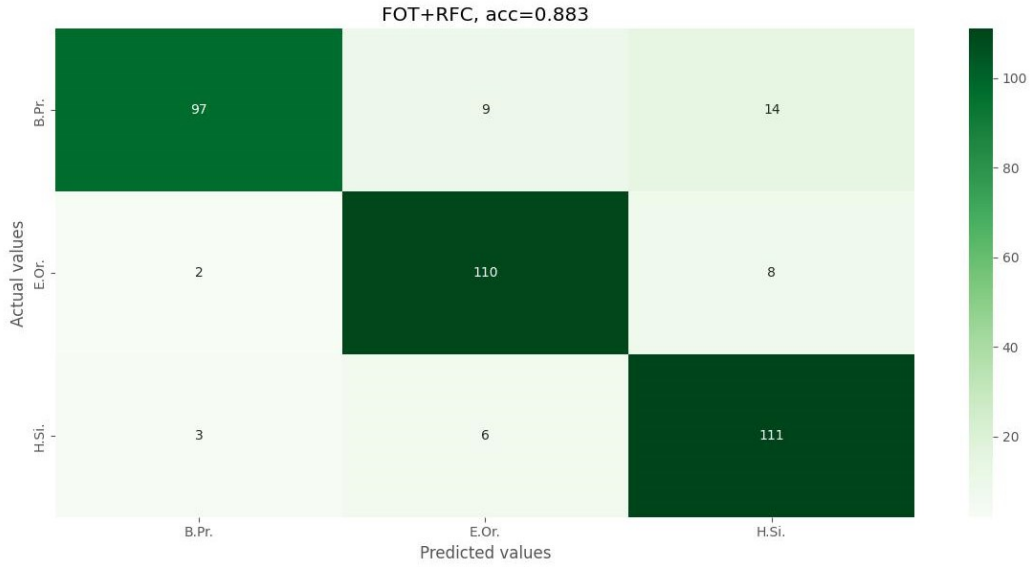
Tabela 5.3: Dokładność modeli opartych o reprezentację FOT dla różnych podziałów zbiorów z grupy zbiorów **E3**. Dla $n_{words} = 30$ i $n_{words} = 300$ zbiory treningowe zawierały 3000 próbek, a testowe 1200, a dla $n_{words} = 1000$ wielkość zbioru treningowego to 900, a testowego 360. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.

Optymalne k dla KNN

Osobno przeprowadziłam eksperyment w celu wyznaczenia optymalnego k dla modelu KNN. Dla $\mathbf{L10}_{ALO}$ i $n_{words} = 30$, $\mathbf{L6}_{ALP}$ i $n_{words} = 100$ oraz $\mathbf{E3}_{A}$ i $n_{words} = 1000$, czyli zbiorów i podziałów, dla których KNN uzyskiwał najwyższą dokładność, modele były uczone dla $k \in \{1, ..., 20\} \cup \{30, 40, 50\}$. Wyniki dla wybranych k przedstawia tabela 5.2.. Dla zbioru $\mathbf{L10}_{ALO}$ największą dokładność wynoszącą 30% otrzymałam dla $k = 13$, dla $\mathbf{L6}_{ALP}$ maksymalną dokładność równą 37% uzyskałam dla $k = 15$, a w wypadku zbioru $\mathbf{E3}_{A}$ dla $k = 8$ uzyskałam dokładność 84%.

Wnioski

Dla wszystkich trzech zbiorów wyraźnie widać, że wpływ preprocessingu jest dużo mniejszy niż wpływ rozmiaru fragmentu – dla ustalonego zbioru i określonego n_{words} zastosowanie różnych preprocessingów daje dość podobne wyniki (niewielka



Rysunek 5.3: Macierz pomyłek modelu RFC dla zbioru $\mathbf{E3}_A$. Wielkość zbioru treningowego to 900, wielkość zbioru testowego to 360, a $n_{words} = 1000$. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Widać wyraźną przekątną.

k	Dokładność dla $\mathbf{L10}_{ALO}$ i $n_{words} = 30$	Dokładność dla $\mathbf{L6}_{ALP}$ i $n_{words} = 100$	Dokładność dla $\mathbf{E3}_A$ i $n_{words} = 1000$
1	0.2	0.23	0.74
5	0.25	0.3	0.81
8	0.25	0.36	**0.84**
11	0.25	0.33	0.83
13	0.3	0.30	0.82
15	0.24	0.37	0.82
18	0.19	0.33	0.8
30	0.18	0.34	0.8
50	0.21	0.32	0.78

Tabela 5.4: Dokładność modelu KNN dla różnych k.

amplituda dokładności), natomiast dla danego zbioru i preprocessingu wyniki dla różnych n_{words} wyraźnie się różnią (duża amplituda dokładności).

Dla wszystkich zbiorów i modeli można zauważyć, że dla podziałów o mniejszej liczbie próbek treningowych i testowych, ale większej liczbie wyrazów we fragmencie, dokładność jest dużo wyższa. Powodem tak dużej rozbieżności dokładności jest duża różnica w stosunku długości wektora cech (27 cech) do długości fragmentu tekstu (15, 30, 50, 100, 300, 1000), na podstawie którego został stworzony ten wektor, między podziałami o małej i dużej liczbie wyrazów we fragmencie. Zauważmy, że

dla najmniejszego $n_{words} = 15$ z fragmentu tworzy się wektor prawie 2 razy dłuższy niż liczba wyrazów – zaproponowana metoda reprezentacji FOT wybierająca 27 cech jest przeznaczona do dłuższych fragmentów tekstów, co potwierdzają wyniki.

Część cech reprezentacji FOT opiera się na interpunkcji, więc wyniki klasyfikowania zbiorów z preprocessingiem (ALP, ALOMP) usuwającym interpunkcję są zwykle nieco gorsze, preprocessing ALOMP, zostawiając wyłącznie czasowniki, rzeczowniki i przymiotniki w podstawowych formach, dodatkowo wpływa na cechę „stosunek liczby różnych słów do liczby wszystkich słów" i wyzerowuje kilka cech gramatycznych, takich jak „liczba spojników" lub „liczba zaimków". Skutecznym doborem preprocessingu jest często tylko anonimizacja, co jest *de facto* brakiem preprocessingu (przypomnę, że wszystkie teksty są anonimizowane, bo nie chcemy, żeby modele uczyły się poprzez „zapamiętanie" nazw własnych). Zgodnie z zapowiedzią z opisu teoretycznego z podrozdziału 5.1.2. RF ma wyższą dokładność niż DC.

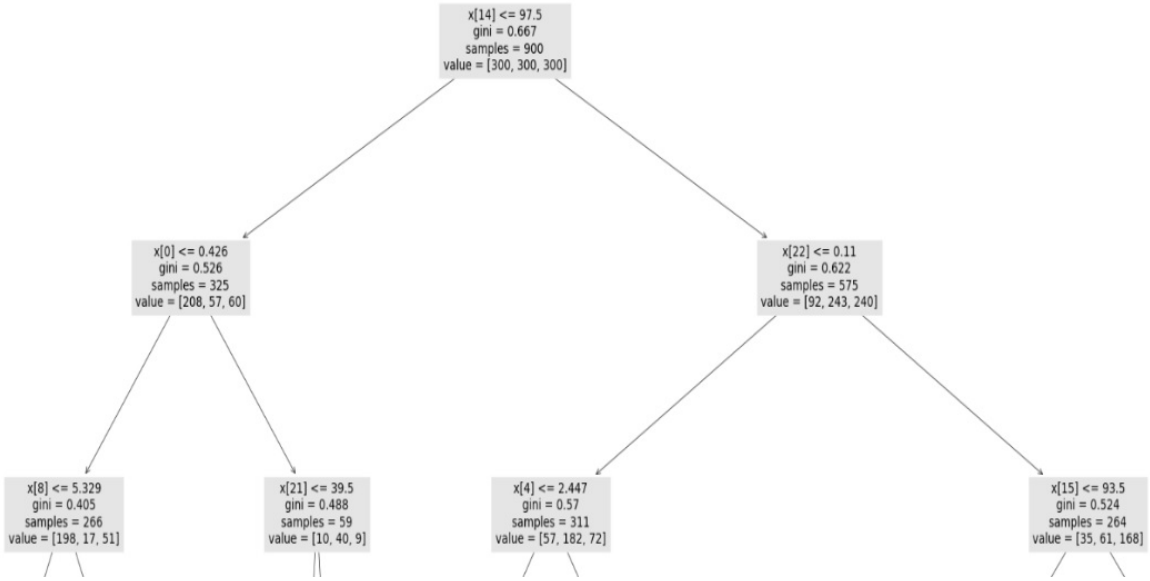
5.3. Wyjaśnialność modeli

Jak wspomniałam w podrozdziale 5.1.1., zaletą drzew decyzyjnych jest ich interpretowalność. Po pierwsze, drzewo takie można łatwo zwizualizować (biblioteka *scikit-learn* posiada funkcję wizualizującą drzewo, fragment takiego drzewa pokazany został na rysunku 5.4). W każdym wierzchołku zbiór danych jest dzielony na dwa podzbiory za pomocą pewnej reguły dzielącej (*splitting rule*) biorącej pod uwagę konkretną cechę. To, która cecha zastanie wybrana, określa wartość GINI, czyli funkcji niejednorodności (impurity function) – wybrana zostaje taka cecha i reguła, dla której wartość GINI jest najmniejsza. Sposobu wyznaczania wartości GINI nie będę tu opisywać, został on jednakże wyjaśniony w dokumentacji biblioteki *scikit-learn* [21].

Po drugie, da się określić istotność cech (feature importance). Wyznaczanie istotności cech opiera się na wzorze zaproponowanym przez Breimana [10, s. 368], którego nie będę tu dokładnie przytaczać[4], ale wyjaśnię ideę obliczania istotności cech. Dla każdego wierzchołka drzewa jest obliczana wartość GINI i waga będąca prawdopodobieństwem wpadnięcia obserwacji do danego wierzchołka. Następnie obliczana jest istotność wierzchołka w jako różnica iloczynu wagi i GINI wierzchołka w i sumy iloczynów wag i GINI prawego w_r i lewego w_l dziecka wierzchołka w. Istotność cechy c to suma istotności wierzchołków, w których reguła dzieląca bierze pod uwagę cechę c.

Na wykresach 5.5, 5.6 i 5.7 pokazano istotności cech wyliczonych dla przykładowych podziałów zbiorów $\mathbf{L10}_{ALO}$, $\mathbf{L6}_{ALS}$ i $\mathbf{E3}_{A}$. Zauważmy, że dla zbioru $\mathbf{E3}_{A}$ liczba przecinków jest cechą o wyraźnie większej istotności niż reszta cech, istotne są też stosunki liczby zaimków do rzeczowników i liczby przymiotników do rzeczowników.

[4]Należy zaznaczyć podstawową różnicę między wzorami – wzór Breimana jako metodę wyznaczania niejednorodności bierze błąd średniokwadratowy, natomiast dla prezentowanego tu problemu klasyfikacji jest to GINI.



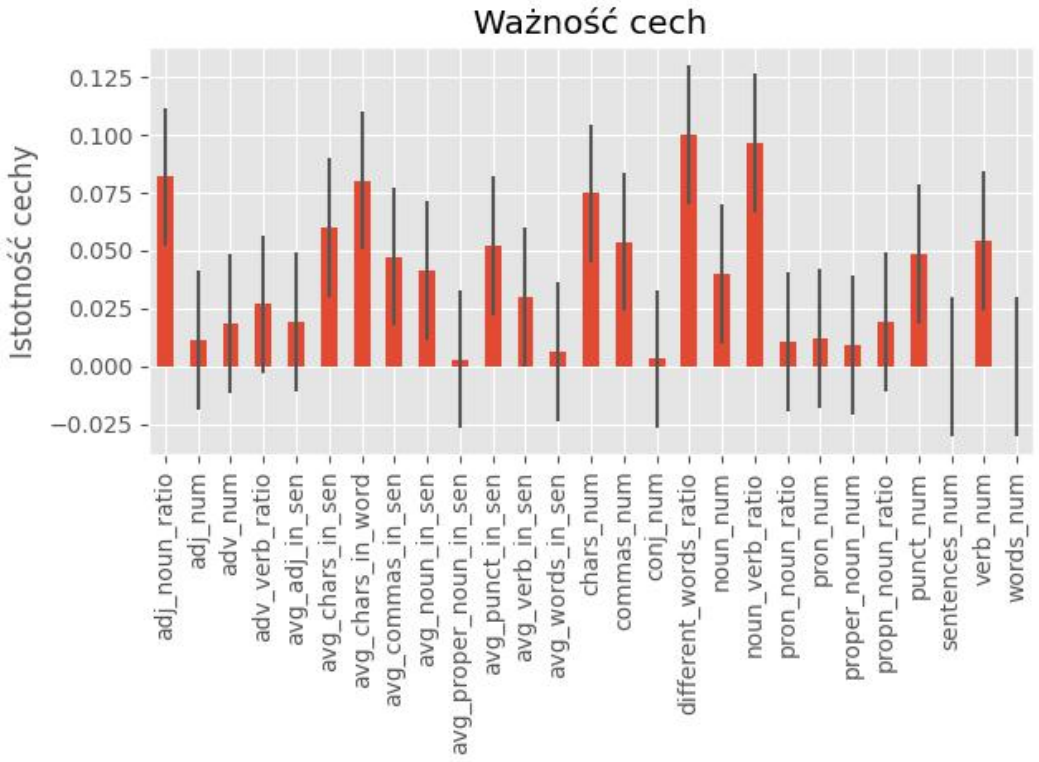
Rysunek 5.4: Pierwsze 3 poziomy drzewa dla $\mathbf{E3}_A$ i $n_{words} = 1000$. x[14] to liczba przecinków, x[0] to stosunek liczby przymiotników do rzeczowników, a x[22] to stosunek liczby nazw własnych do rzeczowników.

Oznacza to, że drzewo decyzyjne „zauważa" wyraźną różnicę między stosowaniem interpunkcji przez Orzeszkową, Sienkiewicza i Prusa. Stosunek liczby zaimków do rzeczowników wskazuje na prawdopodobne różnice w stosowaniu zaimków i synonimów w zdaniach rematycznych[5], a stosunek liczby przymiotników do rzeczowników może ujawniać różny stopień opisowości tekstów pozytywistycznych autorów (w uproszczeniu – im większe nagromadzenie przymiotników, tym tekst jest bardziej opisowy).

Dla zbiorów $\mathbf{L10}_{ALO}$ i $\mathbf{L6}_{ALS}$ ważność cech jest bardziej zrównoważona. Najistotniejszymi cechami dla zbioru $\mathbf{L10}_{ALO}$ okazują się stosunki: liczby różnych słów do liczby wszystkich słów, liczby rzeczowników do czasowników i przymiotników do rzeczowników, a dla zbioru $\mathbf{L6}_{ALS}$ średnia liczba znaków w zdaniu, stosunek przysłówków do czasowników i średnia liczba znaków w słowie. Na każdym z trzech wykresów można zauważyć 1-2 cechy nieznaczące (są to liczba zdań lub liczba słów – ze względu na to, że liczba słów we fragmencie jest stała, można się spodziewać, że powinna być to cecha nieznacząca).

Istotność cech wyznacza się również dla lasu losowego. Wykresów tu nie przedstawiam, bo dla tych samych podziałów zbiorów $\mathbf{L10}_{ALO}$, $\mathbf{L6}_{ALS}$ i $\mathbf{E3}_A$ wykresy dla DTC i RFC są bardzo podobne – te cechy, które dla określonego zbioru i podziału miały dużą istotność dla drzewa, mają także dużą istotność dla lasu, a te o małej

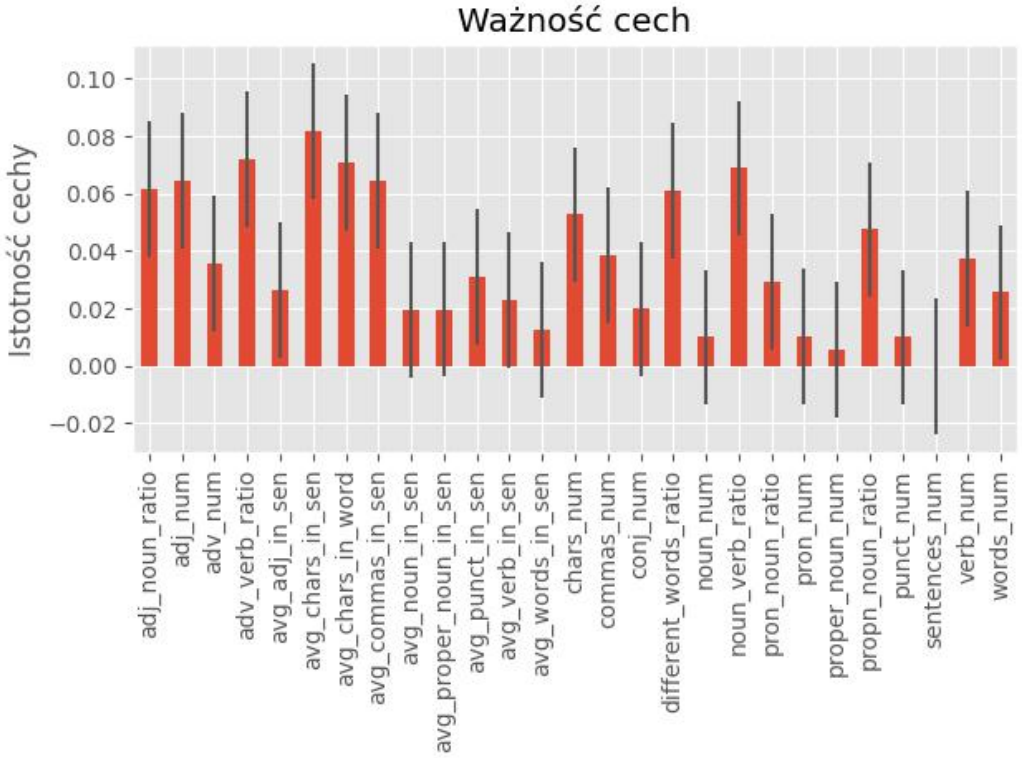
[5]Dla lepszego zrozumienia podam przykład opozycji wypowiedzi różniących się zastosowaniem zaimka i synonimu w drugim zdaniu (rematycznym): "Chłopcy szli przez las. Młodzieńcy nie spodziewali się, że zaraz zacznie padać."oraz "Chłopcy szli przez las. Oni nie spodziewali się, że zaraz zacznie padać".



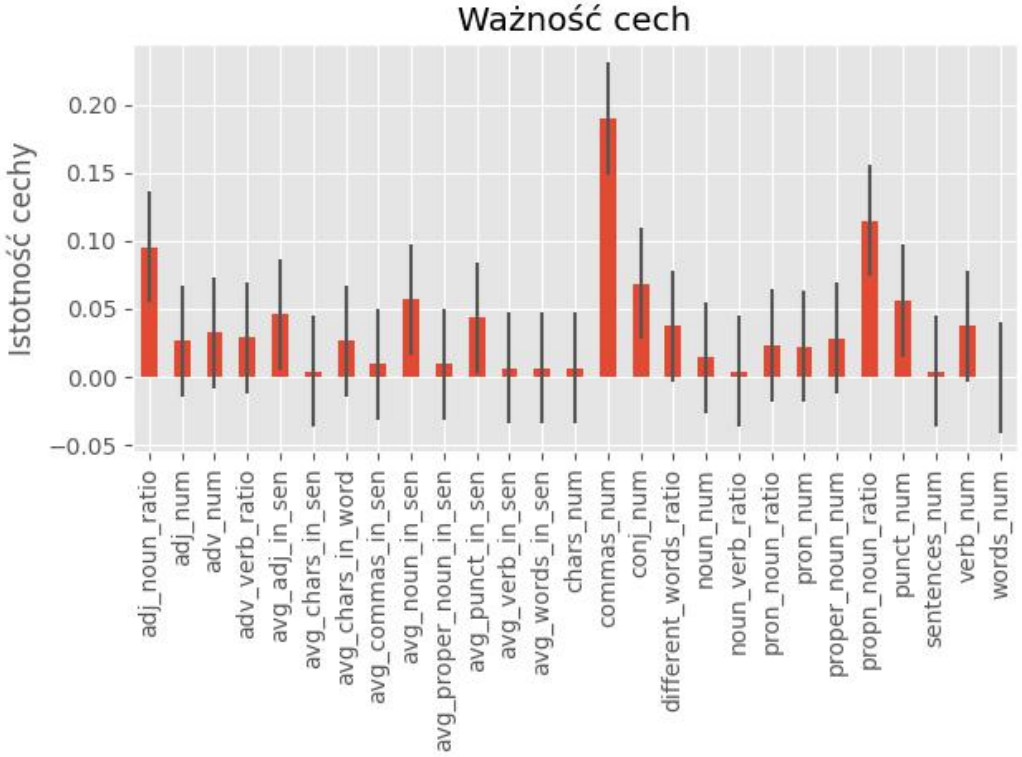
Rysunek 5.5: Istotność cech dla DTC dla $\mathbf{L10}_{ALO}$ i $n_{words} = 30$.

istotności dla modelu drzewa mają także małą istotność w modelu lasu losowego.

Zauważmy, że uzyskane podczas badania istotności wyniki, szczególnie dla zbioru **E3**, zgadzają się z wcześniejszą obserwacją – liczba przecinków jest istotną cechą, więc usunięcie interpunkcji powoduje gorsze wyniki klasyfikatora. Badanie istotności cech pozwala nie tylko zrozumieć zachowanie modelu, ale także mogłoby pozwolić lepiej dopasować na tej podstawie cechy i rozmiar wektora FOT poprzez usunięcie najmniej istotnych cech.



Rysunek 5.6: Istotność cech dla DTC dla $\mathbf{L10}_{ALS}$ i $n_{words} = 100$.



Rysunek 5.7: Istotność cech dla DTC dla $\mathbf{E3}_A$ i $n_{words} = 1000$.

Rozdział 6.

Klasyfikacja modelami korzystającymi z reprezentacji zliczających tokeny

Metody opisane w poniższym rozdziale będą korzystały z reprezentacji Bag of Words, Bag of 2-grams i Bag of 3-grams opisanych w rozdziale 3.2.. Wektory tych reprezentacji są o wiele dłuższe niż w wypadku FOT i nie mają stałej, ustalonej z góry długości. Modelami klasyfikującymi będą: Gaussowski klasyfikator Bayesa, wielomianowy klasyfikator Bayesa oraz SVM. Tak jak w poprzednim rozdziale, badana będzie dokładność modeli na zbiorach **E3**, **L10** i **L6** w zależności od preprocessigu opisanego w 2.2. oraz sposobu podziału zbioru na próbki tekstów opisanego w 2.3..

6.1. Teoretyczny opis metod

6.1.1. Naiwne klasyfikatory Bayesowskie (Naive Bayes – NB)

Naive Bayes to probabilistyczny nieskomplikowany klasyfikator często używany do zadania klasyfikacji. Dla klasyfikowanego fragmentu (dokumentu) d klasyfikator opiera się na pomyśle, że przewidywana klasa c_e to:

$$c_e = \underset{c \in C}{\operatorname{argmax}} P(c|d) \tag{6.1}$$

Bez straty ogólności można założyć, że dokument d jest zbiorem cech $f_1, f_2, \ldots, f_n$. Przypomnę, że w tym rozdziale korzystamy z reprezentacji zliczających tokeny, więc są to cechy liczbowe uzyskane w wyniku zliczania wystąpień słów. Po uwzględnieniu *naiwnego założenia Bayesowskiego* (naive Bayes assumption) o niezależności prawdopodobieństw $P(f_i|c)$ i odpowiednich przekształceniach opisanych przez Jurafsky'ego

i Martina [14], wzór na klasę zwracaną przez klasyfikator Bayesowski to:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i=1}^{n} P(f_i|c) \tag{6.2}$$

Istnieją różne naiwne klasyfikatory Bayesowskie różniące się założeniami dotyczącymi rozkładu $P(f_i|c)$. Skupimy się na dwóch z nich:

- Gaussowski naiwny klasyfikator Bayesa (Gaussian Naive Bayes – GNB)
  Dla tego klasyfikatora zakłada się, że prawdopodobieństwo ma rozkład Gaussa:

  $$P(f_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(f_i-\mu_c)^2}{2\sigma_c^2}} \tag{6.3}$$

  Parametry σ_c i μ_c są szacowane za pomocą MLE (Maximum Likelihood Estimation) [21].

- Wielomianowy naiwny klasyfikator Bayesa (Multinomial Naive Bayes – MNB)
  Klasyfikator ten zakłada wielomianowy rozkład prawdopodobieństw i szczególnie często jest używany do zadania klasyfikacji tekstów reprezentowanych przy pomocy wektorów zliczeń wystąpień słów (reprezentacja Bag of Words). Prawdopodobieństwo $P(f_i|c)$ jest estymowane jako:

  $$P(f_i|c) = \frac{N_{ci} + \alpha}{N_c + \alpha m} \tag{6.4}$$

  gdzie m jest liczbą cech (wielkością słownika w wypadku reprezentacji Bag of Words), $N_{ci} = \sum_{f \in T} f_i$ to liczba wystąpień cechy f_i we wszystkich próbkach klasy c w zbiorze treningowym T, $N_c = \sum_{i=1}^{m} N_{ci}$ to liczba wystąpień wszystkich cech w próbkach klasy c, a smoothing $\alpha \geq 0$ uwzględnia cechy, których nie ma w zbiorze treningowym i zapobiega zerowemu prawdopodobieństwu [21].

Skorzystałam z modeli *GaussianNB* oraz *MultinomialNB* z pakietu *scikit-learn*.

6.1.2. SVC

SVC (Support Vector Classifier) to klasyfikator uczenia nadzorowanego SVM (Support Vector Machine). Problemem SVM jest podział przestrzeni wektorów hiperpłaszczyznami, które maksymalizują dystans między danymi z różnych klas [9]. Biorąc pod uwagę dyskusję i wyniki przedstawione przez Colasa i Brazdila [6], testy ograniczyłam do liniowego SVC. Skorzystałam z modelu *LinearSVC* z pakietu *scikit-learn*, który jest szybszą implementacją SVC z liniową funkcją jądra (funkcją decydującą o podziale przestrzeni hiperpłaszczyznami). Wzory matematyczne definiujące SVC można znaleźć w rozdziale *Support Vector Machines* dokumentacji biblioteki *scikit-learn* [21].

Należy zaznaczyć, że w wersji podstawowej SVC wspiera wyłącznie binarną klasyfikację. Implementacja w *scikit-learn* dostosowana jest jednak również do problemów wieloklasowych. *LinearSVC* implementuje strategię „one-vs-the-rest"[1], w której konstruowane jest tyle modeli, ile klas, i każdy model tworzy hiperpłaszczyznę w najlepszy możliwy sposób oddzielającą konkretną klasę od reszty.

6.2. Omówienie uzyskanych wyników

Tak jak w poprzednim rozdziale, przeprowadziłam testy modeli SVC, GNB i MNB, ucząc je na zbiorach **L10**, **L6** i **E3** poddanych preprocessingom opisanym w podrozdziale 2.1.. Tabele 6.1, 6.2 i 6.3 przedstawiają wybrane wyniki (dokładność oraz współczynnik κ) modeli dla zbiorów **L10**, **L6** i **E3** w reprezentacji Bag of Words. Wykonałam też testy dla reprezentacji Bag of 2-grams oraz Bag of 3-grams, jednak zrezygnowałam z prezentowania ich w tabelach, bo wyniki były bardzo zbliżone do wyników reprezentacji BOW . We wszystkich tabelach kolorem niebieskim wyróżnione zostały najwyższe rezultaty uzyskane przez klasyfikatory SVC, GNB i MNB, a pomarańczowym najniższe. Ponadto kolorami zielonym i czerwonym zaznaczone zostały najlepszy i najgorszy rezultat dla każdego podziału prezentowanego w tabeli.

Zbiór L10

Dla zbioru **L10** wybrane wyniki przedstawia tabela 6.1. Wszystkie klasyfikatory spisują się wyraźnie lepiej od klasyfikatora losowego – wartość współczynnika κ znajduje się między 0.19 a 0.57. Największą dokładność, wynoszącą 61%, osiągnął SVC, niewiele niższą MNB – 60%, a GNB – 54%. Różnica między podziałem, w którym fragmenty zawierają 30 wyrazów, a takim, w którym fragmenty mają tylko 15 wyrazów, jest zauważalna, ale dużo mniejsza niż zaobserwowana w rozdziale 5. – maksymalna dokładność dla podziału z $n_{words} = 15$ wynosi 41.5%, a minimalne dokładności różnią się o $1.5 p.p.$ Najlepsze rezultaty osiągane są dla preprocessingów ALO i ALOMP, a najgorsze dla ALP. Macierze pomyłek modeli MNB i SVC dla zbioru $\mathbf{L10}_{ALO}$ przedstawione na rysunku 6.1 dla obu modeli mają wyraźnie zaznaczoną przekątną, dla SVC można zauważyć, że klasyfikator chętnie wybiera klasę Norwida (jest to spowodowane najprawdopodobniej binarnym podziałem przestrzeni przez SVC opisanym w podrozdziale 6.1.2. i przestaje być to widoczne w macierzach pomyłek dla najlepszych wyników osiąganych przez model SVC).

[1] *LinearSVC* implementuje też strategię „cramer-singer", jednak, jak zaznacza dokumentacja *scikit-learn*, jest ona rzadko używana, bo daje podobne rezultaty, a czas obliczeń jest wyraźnie dłuższy.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	15	SVC	0.3	0.222
ALS	15	MNB	0.415	0.35
ALO	30	GNB	0.54	0.489
ALO	30	MNB	0.6	0.556
ALP	15	GNB	0.275	0.194
ALP	15	MNB	0.31	0.233
ALP	30	GNB	0.29	0.211
ALOMP	30	SVC	**0.61**	**0.567**

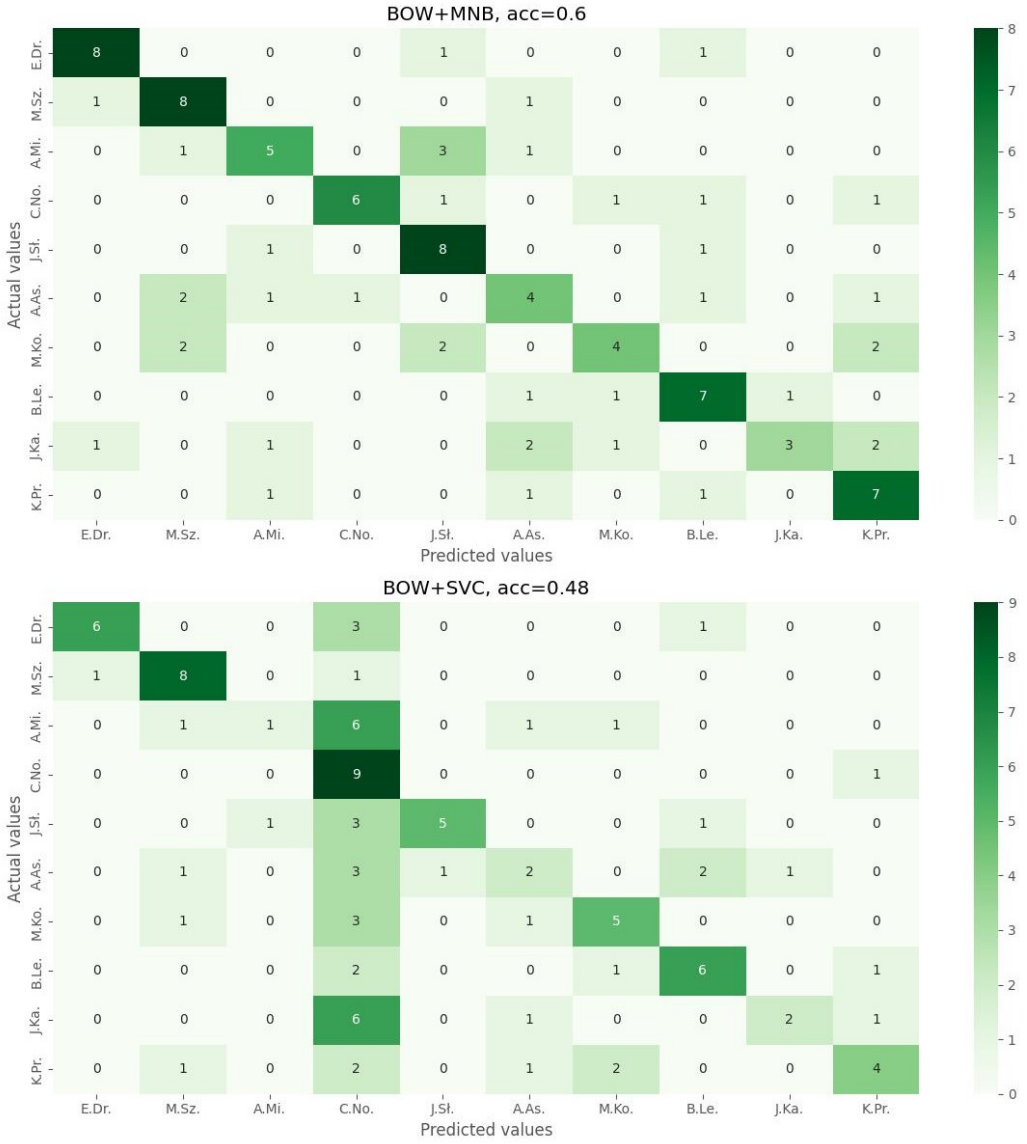
Tabela 6.1: Dokładność modeli opartych o reprezentację BOW dla różnych podziałów zbiorów z grupy zbiorów **L10**. Dla $n_{words} = 15$ zbiór treningowy zawierał 1000 próbek, a testowy 200, a dla $n_{words} = 30$ wielkość zbioru treningowego to 500, a testowego 100. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.

Zbiór L6

Dla zbioru **L6** (tabela 6.2) wartości współczynnika κ mieszczą się w przedziale $[0.44; 0.82]$, a maksymalna dokładność wynosi 85% dla MNB, 77% dla GNB i 80% dla SVC. Najlepsze rezultaty osiągane są dla preprocessingu ALS i ALOMP, a najgorsze, gdy zbiór został poddany wyłącznie anonimizacji. Rysunek 6.2 przedstawia macierz pomyłek modelu MNB dla $\mathbf{L6}_{ALS}$ – klasa Leśmiana rozpoznawana jest bezbłędnie i wiele pól macierzy poza przekątną wypełnionych jest zerami.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	50	GNB	0.629	0.555
A	100	SVC	0.625	0.55
A	30	GNB	0.533	0.44
A	30	MNB	0.611	0.533
ALS	100	GNB	0.767	0.72
ALS	100	MNB	**0.85**	**0.82**
ALO	100	SVC	0.8	0.76
ALP	30	SVC	0.597	0.517
ALOMP	50	MNB	0.838	0.805
ALOMP	100	MNB	**0.85**	**0.82**
ALOMP	30	MNB	0.736	0.683

Tabela 6.2: Dokładność modeli opartych o reprezentację BOW dla różnych podziałów zbiorów z grupy zbiorów **L6**. Dla $n_{words} = 30$ zbiór treningowy zawierał 1800 próbek, a testowy 360, dla $n_{words} = 50$ wielkość zbioru treningowego to 1200, a testowego 240, a dla $n_{words} = 100$ odpowiednio 600 i 120. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.

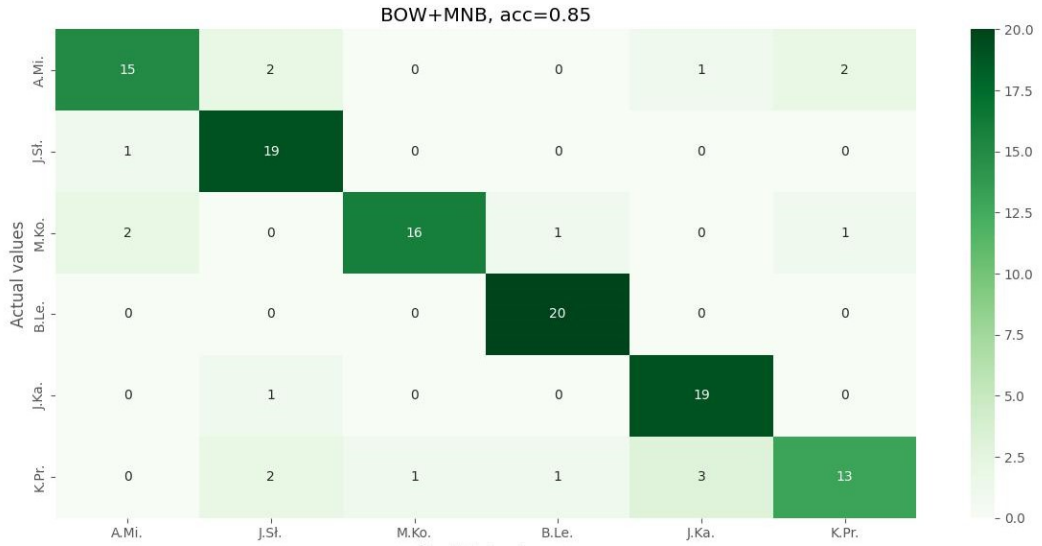


Rysunek 6.1: Macierze pomyłek modeli MNB i SVC dla zbioru $\mathbf{L10}_{ALO}$. Wielkość zbioru treningowego to 500, wielkość zbioru testowego to 100, a n_{words} =30. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Model SVC faworyzuje klasę Norwida.

Zbiór E3

Dla zbioru **E3** (tabela 6.3) poddanego preprocessingom ALS i ALO wszystkie trzy omawiane modele osiągają dobre wyniki[2], zarówno gdy $n_{words} = 1000$, jak i dla $n_{words} = 300$. Najwyższa dokładność wynosi 99.7% i jest osiągana przez modele MNB i SVC, jeśli jednak zamiast BOW skorzystamy z BO2 lub BO3, dla preprocessingu A i $n_{words} = 1000$ dokładność modelu MNB wynosi 100%. Najgorsze wyniki osiągane są dla preprocessingu ALP. Rysunek 6.3 przedstawia macierz pomyłek modelu SVC

[2]Przypomnijmy, że za dobry wynik uznaje się $\kappa \geq 0.8$ (rozdział 4.).

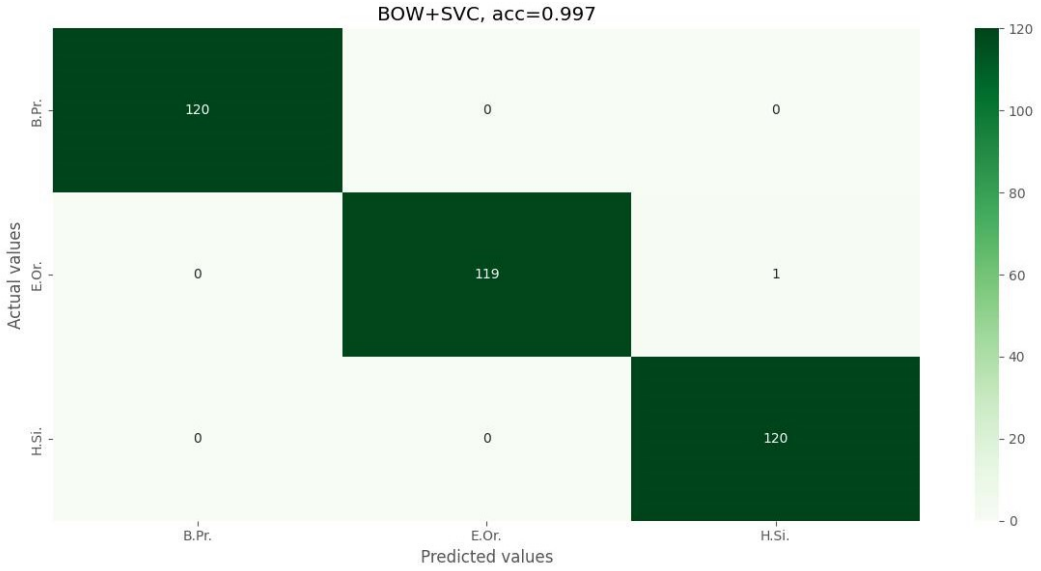


Rysunek 6.2: Macierz pomyłek modelu MNB dla zbioru $\mathbf{L10}_{ALS}$. Wielkość zbioru treningowego to 600, wielkość zbioru testowego to 120, a n_{words} =100. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Wiele pól poza przekątną ma wartość 0.

dla $\mathbf{E3}_A$ i $n_{words} = 1000$ – model pomylił się tylko raz.

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	300	GNB	0.844	0.766
A	1000	MNB	**0.997**	**0.996**
A	1000	SVC	**0.997**	**0.996**
ALS	300	GNB	0.965	0.948
ALS	300	SVC	0.989	0.984
ALS	1000	SVC	**0.997**	**0.996**
ALP	30	GNB	0.528	0.291
ALP	30	MNB	0.647	0.47
ALP	30	SVC	0.616	0.424
ALP	1000	GNB	0.781	0.671
ALP	1000	SVC	**0.997**	**0.996**
ALOMP	30	MNB	0.79	0.685

Tabela 6.3: Dokładność modeli opartych o reprezentację BOW dla różnych podziałów zbiorów z grupy zbiorów **E3**. Dla $n_{words} = 30$ i $n_{words} = 300$ zbiory treningowe zawierały 3000 próbek, a testowe 1200, a dla $n_{words} = 1000$ wielkość zbioru treningowego to 900, a testowego 360. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.



Rysunek 6.3: Macierz pomyłek modelu SVC dla zbioru $\mathbf{L10}_A$. Wielkość zbioru treningowego to 900, wielkość zbioru testowego to 360, a n_{words} =1000. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów.

Wnioski

Metody GNB, MNB i SVC oparte na reprezentacjach BOW, BO2 i BO3 lubią wyjmowanie z tekstów najistotniejszych informacji – pokazują to zauważalnie wyższe skuteczności przy preprocessingach ALS, ALO i ALOMP.

Różnice między różnymi podziałami są zauważalne, ale przeważanie nie są tak duże, jak dla wyników opisanych w rozdziale 5.. W pierwszym momencie może wydać się to zaskakujące, bo w rozdziale 5. reprezentacja tekstu FOT miała stałą długość, niezależnie od fragmentu, a reprezentacja BOW zależy od długości tekstu, więc i wynik mógłby być wyraźnie zależny, jednak spowodowane jest to wspomnianym w podrozdziale 5.2. dużą różnicą w stosunku długości wektora cech (27 cech) do długości fragmentu tekstu (15, 30, 50, 100, 300, 1000 wyrazów). Reprezentacje zliczające tokeny są mniej czułe na zmianę długości fragmentu tekstu i dają przyzwoite wyniki nawet dla niewielkich n_{words}.

6.3. Wyjaśnialność modeli

6.3.1. LIME

LIME (Local Interpretable Model-agnostic Explanations) to biblioteka, której celem jest wyjaśnianie w sposób czytelny dla człowieka, co robią klasyfikatory ML. Projekt został oparty na pracy teoretycznej opisanej w artykule M. Ribeira, S. Singh'a i C. Guestrina *„Why Should I Trust You?": Explaining the Predictions of Any*

Classifier [24]. Główną funkcjonalnością LIME'a jest określenie pewności klasyfikacji i cech, na podstawie których model przypisał konkretną próbkę do konkretniej klasy. LIME działa między innymi dla danych tekstowych, dla których interpretacja polega na znalezieniu obecności słów najbardziej wpływających na wynik klasyfikacji.

Autorzy LIME'a definiują dobry „wyjaśniacz modeli" jako: interpretowalny (łatwy do zrozumienia przez człowieka), lokalnie wierny (musi odtwarzać zachowanie modelu w pobliżu klasyfikowanej próbki), działający na dowolnym modelu i wyjaśniający globalną perspektywę (wybór reprezentatywnych dla modelu wyjaśnień). Problemem jest kompromis między interpretowalnością a wiernością, który rozwiązuje się przez minimalizowanie miary niewierności modelu przy jednoczesnym utrzymaniu wystarczająco niskiej miary złożoności model explainera[3], aby był on interpretowalny przez człowieka [24].

LIME wyjaśnia działanie klasyfikatora na przykładzie konkretnego tekstu. Najistotniejszymi argumentami funkcji wyjaśniającej LIME'a są więc klasyfikator (a dokładniej lista prawdopodobieństw *predict proba* wyestymowanych przez klasyfikator) i próbka tekstu. LIME może pomóc w wyborze najbardziej godnego zaufania klasyfikatora spośród kilku (np. uczonych na różnych danych) [24].

6.3.2. Interpretowalność modeli na przykładach

Wyjaśnialności poddane zostały klasyfikatory[4] osiągające największa dokładność na zbiorach **L10**, **L6** i **E3** klasyfikujące teksty opisane w podrozdziale 2.4.. Tekst niepewnego autorstwa przypisywany Mickiewiczowi i pastisz Słowackiego były klasyfikowane przez modele uczone na zbiorach **L10** i **L6**, natomiast list Sienkiewicza przez model uczony na zbiorze **E3**. Teksty poddane zostały takiemu samemu preprocessingowi, jakiemu poddany był zbiór, na którym uczył się klasyfikator.

Tabela 6.4 przedstawia wyniki klasyfikacji przykładowych tekstów. 4 z 6 klasyfikatorów uznaje, że pastisz Wyki jest tekstem Słowackiego – wydaje się to dobrym wynikiem. Niepewny tekst przypisywany Mickiewiczowi jest przypisywany przez różne klasyfikatory do 5 różnych twórców (raz pojawia się Mickiewicz), prawdopodobieństwo tylko w jednym wypadku przekracza 0.5 (poza GNB, dla którego prawdopodobieństwo różnych klas to 0 lub 1) – na podstawie różnorodności wyników można uznać, że jest to tekst o niepewnym autorstwie. List Sienkiewicza klasyfikatory Bayesowskie przypisują Sienkiewiczowi, natomiast SVC – Prusowi.

Rysunki 6.4, 6.5 i 6.6 przedstawiają wyniki LIME dla 3 wybranych klasyfikatorów i tekstów. Wyjście LIME'a składa się z 3 części:

[3]Złożoność jest definiowana jako opozycja do interpretowalności. Na przykład dla drzew miarą złożoności jest ich głębokość, a dla modeli liniowych miarą złożoności jest liczba niezerowych wag.

[4]Z powodu niedopasowania klasyfikatora *LinearSVC* z pakietu *scikit-learn* od wymagań LIME, *LinearSVC* zastąpiłam klasyfikatorem *SVC* z liniową funkcją jądra z tego samego pakietu, który jest wolniejszy i, jak pokazały testy, osiąga nieco mniejszą dokładność.

Zbiór	n_{words}	Model	Dokładność	Prawdziwa klasa	Przewidywana klasa	P-stwo
$\mathbf{L10}_{ALO}$	30	GNB	0.54	Słowacki	Słowacki	1.0
$\mathbf{L10}_{ALO}$	30	MNB	0.6	Słowacki	Słowacki	0.68
$\mathbf{L10}_{ALOMP}$	30	SVC	0.52	Słowacki	Słowacki	0.44
$\mathbf{L6}_{ALS}$	100	GNB	0.767	Słowacki	Konopnicka	1.0
$\mathbf{L6}_{ALS}$	100	MNB	0.85	Słowacki	Słowacki	0.91
$\mathbf{L6}_{ALO}$	100	SVC	0.717	Słowacki	Tetmajer	0.6
$\mathbf{L10}_{ALO}$	30	GNB	0.54	—	Konopnicka	1.0
$\mathbf{L10}_{ALO}$	30	MNB	0.6	—	Leśmian	0.34
$\mathbf{L10}_{ALOMP}$	30	SVC	0.52	—	Sęp Szarzyński	0.23
$\mathbf{L6}_{ALS}$	100	GNB	0.767	—	Mickiewicz	1.0
$\mathbf{L6}_{ALS}$	100	MNB	0.85	—	Tetmajer	0.52
$\mathbf{L6}_{ALO}$	100	SVC	0.717	—	Tetmajer	0.47
$\mathbf{E3}_{ALS}$	300	GNB	0.965	Sienkiewicz	Sienkiewicz	1.0
$\mathbf{E3}_{A}$	1000	MNB	0.997	Sienkiewicz	Sienkiewicz	1.0
$\mathbf{E3}_{A}$	1000	SVC	0.997	Sienkiewicz	Prus	0.55

Tabela 6.4: Klasyfikacja tekstów spoza zbiorów **E3** i **L10**

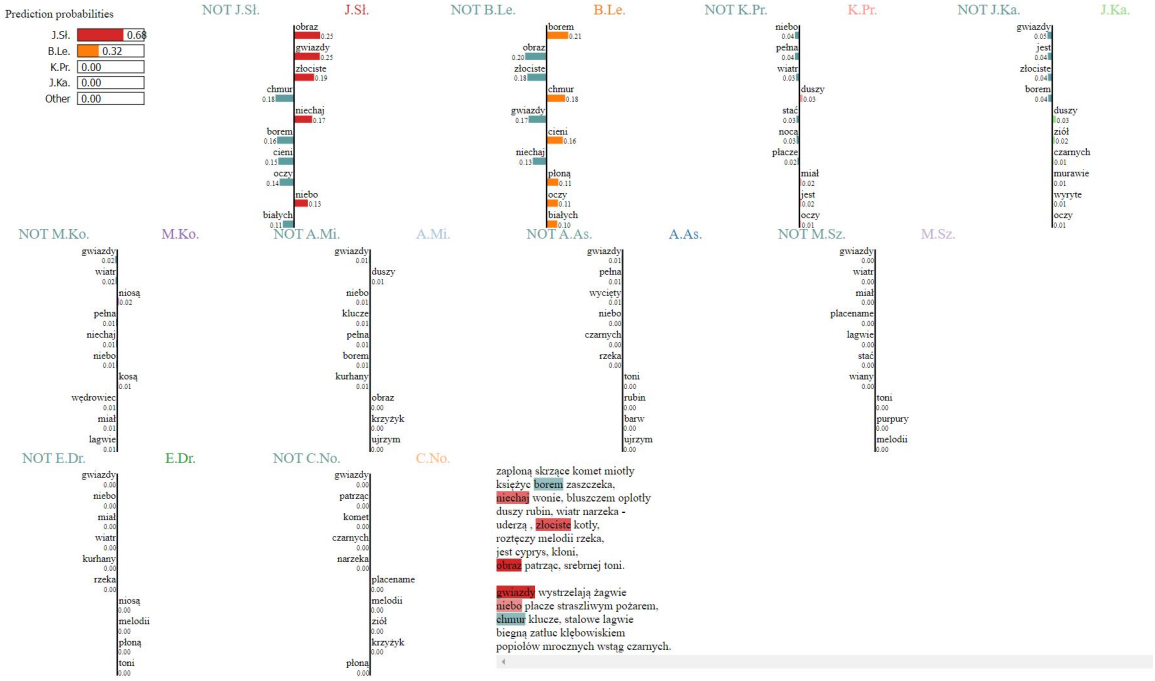
- prawdopodobieństw poszczególnych klas – graficzna forma listy *predict proba* klasyfikatora;
- 10 najistotniejszych cech dla każdej z klas potwierdzających (po prawej) lub odrzucających (po lewej) klasę; liczby przy każdej cesze oznaczają relatywną ważność cech wyliczoną przez LIME'a;
- tekstu poddawanego klasyfikacji z zaznaczonymi najistotniejszymi wyrazami wpływającymi na wynik – na grafikach tylko dla niepewnego tekstu Mickiewicza widać pełną treść utworu, dla pozostałych tylko fragment.

Zauważmy, że 10 cech (10 wyrazów) to 23% tekstu przypisywanego Mickiewiczowi (po preprocessingu ALO zostały 43 wyrazy), 8% pastiszu autorstwa Wyki (po preprocessingu ALO zostało 121 wyrazów) i 3% listu Sienkiewicza (po preprocessingu ALS zostało 376 wyrazów). Jednocześnie w graficznej formie oferowanej przez LIME'a trudno byłoby przedstawić dużo więcej cech dla każdej klasy – grafika taka byłaby duża i nieczytelna.

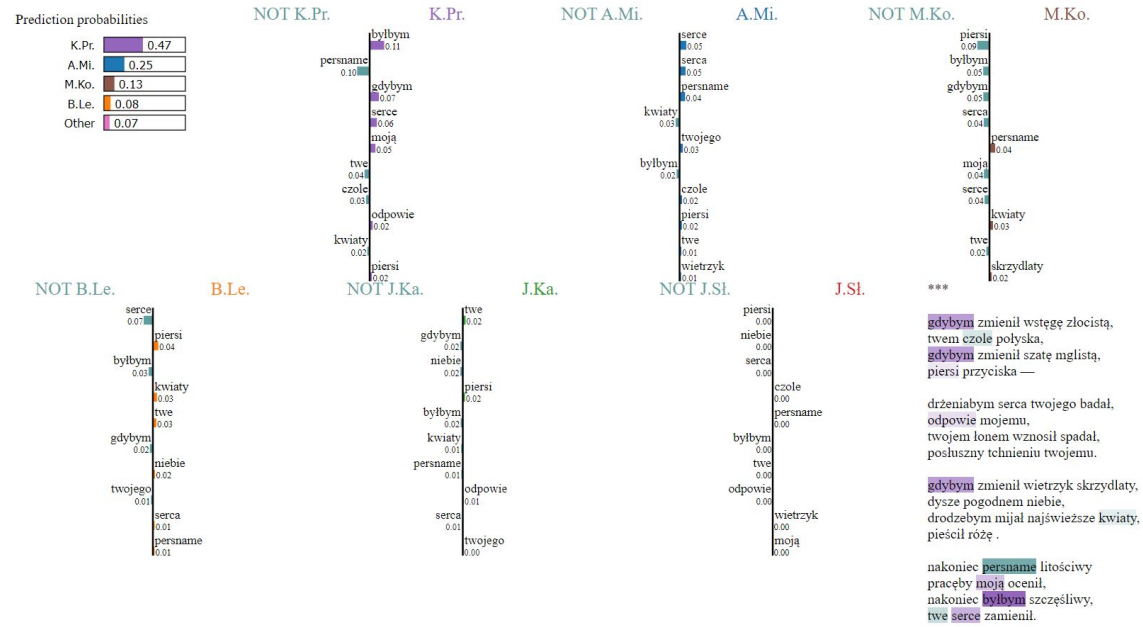
Dla klasyfikacji pastiszu tekstu Słowackiego przez klasyfikator MNB najistotniejszymi słowami wskazującymi na autorstwo Słowackiego są: *obraz*, *gwiazdy*, *złociste*, *niechaj* i *niebo*, a wskazującymi na autorstwo nie Słowackiego a Leśmiana, jednak o sumarycznej niższej ważności, są: *chmur*, *borem*, *cieni*, *oczy* i *białych*. Pozostałe cechy i klasy mają nieistotnie małą ważność. Wybrane cechy można uznać za sensowne – są to w większości rzeczowniki niosące znaczenie i w pewien sposób streszczające tekst z ludzkiego punktu widzenia.

Model SVC klasyfikuje niepewny tekst przypisywany Mickiewiczowi jako autorstwo Tetmajera na podstawie słów: *byłbym*, *gdybym*, *serce*, *moją*, *odpowie*, *piersi*, a wyrazy wskazujące na autorstwo inne inż Tetmajera to: anonimizowana nazwa osobowa, *twe*, *czole* i *kwiaty*. Cechy te niewiele mówią o klasyfikowanym tekście – są wśród nich 3 czasowniki, w tym dwa w trybie przypuszczającym są często używane, 4 rzeczowniki, ale o niewielkiej względnej istotności, anonimizowana nazwa osobowa i 2 zaimki dzierżawcze.

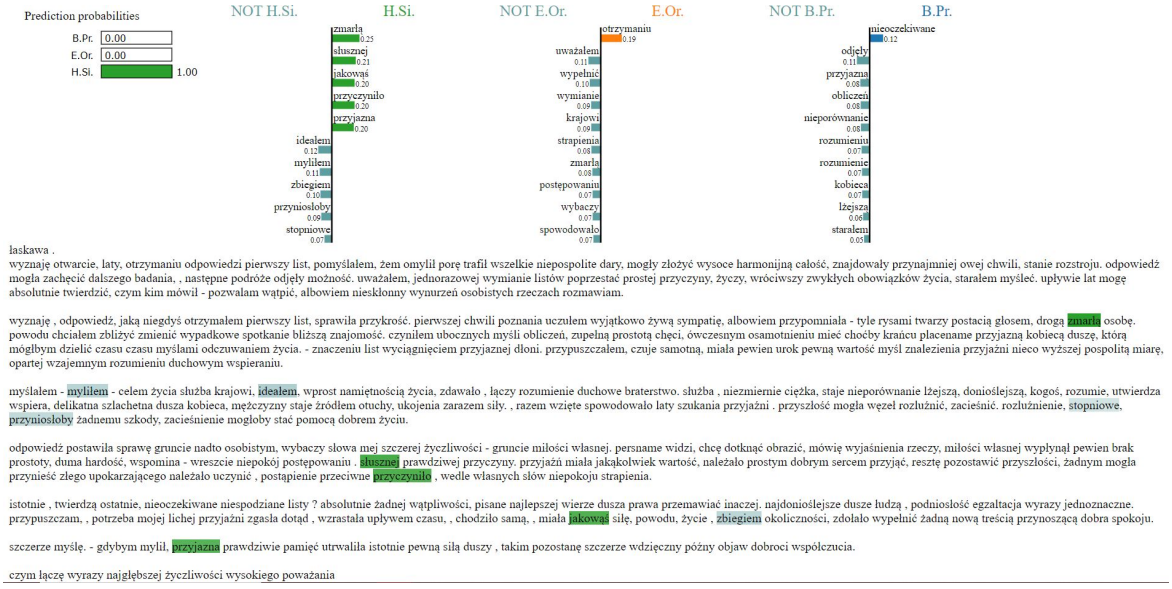
Model GNB potwierdza autorstwo Sienkiewicza w wypadku listu na podstawie cech: *zmarłą*, *słusznej*, *jakowąś*, *przyczyniło*, *przyjazna*, natomiast wyrazy przeczące autorstwu Sienkiewicza (o mniejszej względnej istotności) to: *ideałem*, *myliłem*, *zbiegiem*, *przyniosłoby*, *stopniowe*. Cechy te z ludzkiego punktu widzenia wydają się sensowne, jednak jest ich mało w stosunku do objętości wyrazowej tekstu.



Rysunek 6.4: Wyjście LIME'a wyjaśniające działanie MNB na tekście pastiszu Słowackiego.



Rysunek 6.5: Wyjście LIME'a wyjaśniające działanie SVC na tekście przypisywanym Mickiewiczowi.



Rysunek 6.6: Wyjście LIME'a wyjaśniające działanie GNB na tekście listu Sienkiewicza.

Rozdział 7.

Modele n-gramowe

W tym rozdziale prezentowane jest inne podejście do problemu klasyfikacji niż w poprzednich rozdziałach – dla każdego autora zostanie zbudowany model języka składający się z n-gramów. Nie będzie więc modeli klasyfikujących uczących się na zbiorze treningowym, a metoda nie będzie korzystała z żadnej reprezentacji spośród opisanych w rozdziale 3.2.. Jednak tak jak w poprzednich rozdziałach badana będzie dokładność na zbiorach **E3**, **L10** i **L6** w zależności od preprocessigu opisanego w podrozdziale 2.2., na podstawie których został zbudowany model językowy dla każdego autora. Modele językowe są probabilistycznymi modelami przewidującymi prawdopodobieństwo słowa lub sekwencji w oparciu o poprzednie słowa. Klasyfikator w tym rozdziale będzie modelem deskryptywnym – klasy będą opisywane przez prawdopodobieństwa wystąpień sekwencji i spośród wszystkich klas wybierana będzie najbardziej prawdopodobna.

Warto też od razu podkreślić różnicę między modelami n-gramowymi a reprezentacją Bag of n-grams – w wypadku reprezentacji Bag of n-grams zliczane są wystąpienia poszczególnych n-gramów, a model n-gramowy uwzględnia historię (kontekst) słowa. Przyjrzyjmy się bigramowi „Ala ma" w sekwencji „Ala ma kota. Ala jest dziewczynką.". Reprezentacja Bag of bigrams po prostu zliczy jedno wystąpienie tego bigramu, natomiast model językowy obliczy prawdopodobieństwo wystąpienia słowa „ma" po słowie „Ala" i uwzględni w tym też wystąpienie słowa „Ala" w drugim zdaniu.

7.1. Teoretyczny opis metod

Ideą n-gramowych modeli językowych jest przewidywanie ciągów wyrazów w określonym języku poprzez przypisywanie prawdopodobieństwa każdemu możliwemu $m+1$-szemu wyrazowi dla danego m elementowego ciągu wyrazów. Na przykład w języku polskim dla ciągu wyrazów *Ubrudziłeś się, więc przebierz* najbardziej prawdopodobnym kolejnym wyrazem będzie wyraz *się*, a dla ciągu *Mamo, proszę kup*

mi kolejny wyraz *lalkę* jest bardziej sensowny niż wyraz *biegać*. Formalnie zadanie modeli językowych sprowadza się do obliczania $P(w|h)$, czyli prawdopodobieństwa słowa w dla danej historii h. Historię h aproksymuje się przez kilka ostatnich słów i w ten sposób powstają należące do klasy modeli Markowa[1] n-gramowe modele językowe, które patrzą na $n-1$ słów z przeszłości. Dla N będącego rozmiarem n-gramu, aproksymację prawdopodobieństwa wyraża się wzorem:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}) \tag{7.1}$$

gdzie $w_{k:l}$ jest skrótowym zapisem ciągu słów $w_k, w_{k+1}, \ldots, w_{l-1}, w_l$, a $P(w_{k:l})$ to skrótowy zapis prawdopodobieństwa tego, że kolejne zmienne losowe X_i przyjmą wartość w_i, czyli $P(X_k = w_k, X_{k+1} = w_{k+1}, \ldots, X_{l-1} = w_{l-1}, X_l = w_l)$. Prawdopodobieństwo $P(w_n|w_{n-N+1:n-1})$ oblicza się, korzystając oszacowania maksymalnego prawdopodobieństwa MLE:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})} \tag{7.2}$$

gdzie $C(w_{k:l})$ oznacza zliczenie wszystkich wystąpień ciągów wyrazów $w_{k:l}$ w korpusie.

Istotnym zadaniem modeli językowych jest obliczanie prawdopodobieństwa całego ciągu wyrazów $P(w_{1:n})$. Korzystając z oznaczeń wprowadzonych powyżej, prawdopodobieństwo to wyliczymy z reguły łańcuchowej:

$$P(w_{1:n}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \tag{7.3}$$

Po uwzględnieniu aproksymacji n-gramowej i szacowania MLE otrzymujemy:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1:k-1}) = \prod_{k=1}^{n} \frac{C(w_{k-N+1:k-1}w_k)}{C(w_{k-N+1:k-1})} \tag{7.4}$$

Jeśli w ciągu słów $w_{1:n}$ pojawi się słowo lub n-gram, którego nie ma w zbiorze treningowym, wyliczone z powyższego wzoru $P(w_{1:n})$ będzie równe 0. Z tego też powodu wprowadza się wygładzanie (*smoothing*), który pozwala radzić sobie z nieznanymi wyrazami (OOV – *out of vocabluary*) i n-gramami. Poza bardzo prostymi metodami smoothingu takimi jak add-one (Laplace) lub add-k, polegającymi na małej modyfikacji wzoru 7.2 przez dodanie 1 lub k do mianownika, a do licznika liczby wyrazów w słowniku przemnożonej przez 1 lub k, tak aby prawdopodobieństwo nigdy nie było równe 0, istnieją też bardziej zaawansowane metody, które $P(w_n|w_{n-N+1:n-1})$ wyliczają jako interpolację prawdopodobieństw n-gramów od 1 do N (czyli nieistniejące n-gramy będą zastępowane n-gramami niższych rzędów). Jednym z takich interpolacyjnych rodzajów wygładzania jest Kneser-Ney smoothing. Metoda ta nie

[1] Bez zagłębiania się w szczegóły: modele Markowa to klasa modeli probabilistycznych, które zakładają, że możemy przewidzieć prawdopodobieństwo wystąpienia jakiejś przyszłej jednostki bez patrzenia daleko w przeszłość [15].

tylko radzi sobie z nieistniejącymi n-gramami, ale również, biorąc pod uwagę kontekst słowa, koryguje wypaczone przez specyficzne dane wyniki, na przykład, jeśli w korpusie szkoleniowym wiele razy pojawi się bigram *San Francisco*, to Kneser-Ney smoothing uwzględni, że słowo *Francisco* występuje głównie po słowie *San* i spowoduje, że unigramowi *Francisco* nie zostanie przypisane wysokie prawdopodobieństwo [3]. Wzór na wygładzenie Knesera-Neya wygląda następująco:

$$\begin{aligned} &P_{KN}(w_n|w_{n-N+1:n-1}) = \\ &\quad = \frac{\max(c_{KN}(w_{n-N+1:n}) - d, 0)}{\sum_v c_{KN}(w_{n-N+1:n-1}v)} + \lambda(w_{n-N+1:n-1})P_{KN}(w_n|w_{n-N+2:n-1}) \end{aligned} \tag{7.5}$$

gdzie $0 \leq d \leq 1$ to wyznaczony współczynnik wygładzania, w literaturze angielskiej określana jako discount (są różne metody wyznaczania wartości d, w najprostszym wypadku może być to stała), λ to waga interpolacji, czyli stała normalizująca, która odpowiada za rozkład prawdopodobieństwa, a sposób zliczania c_{KN} zależy od tego, czy interpolowany n-gram jest najwyższego rzędu, czy jest to jeden z n-gramów niższego rzędu. Wyprowadzenie wzoru i dokładniejszą dyskusję na jego temat można znaleźć w książce Jurafsky'ego i Martina [15] lub w artykule Chena i Goodmana [3].

Modele językowe często wykorzystywane są do zadań rozpoznawania mowy, poprawiania błędów pisowni lub gramatycznych i tłumaczenia maszynowego [15], jednak na podstawie tej metody można też zbudować klasyfikatory.

Jedną z implementacji modeli n-gramowych jest narzędzie *KenLM* [11]. Implementację tę wyróżnia szybkość działania i małe zużycie pamięci w porównaniu do innych implementacji modeli językowych, na przykład *KenLM* jest 2.4 razy szybsze niż często używana implementacja SRILM przy jednoczesnym zużyciu 57% pamięci, którą używa SRILM. Aby uzyskać taką efektywność, twórcy biblioteki wprowadzili dwie struktury danych: nastawionej na szybkość struktury PROBING, używającej tablic haszujących z próbkowaniem liniowym (*linear probing hash tables*) oraz struktury TRIE, czyli drzewa trie przeznaczonego do przechowywania ciągów znaków z pakowaniem bitowym, wyszukiwaniem interpolacyjnym i posortowanymi rekordami, które jednocześnie optymalizują czas obliczeń i pamięć [11]. *KenLM* używa zmodyfikowanego Kneser-Ney smoothingu. Sposobu modyfikacji nie będę tu opisywać, ale można go znaleźć w artykule Chena i Goodmana [3].

Przy tworzeniu modeli językowych *KenLM* korzysta z plików w formacie ARPA, które tworzy się przy pomocy narzędzia *lmplz*. Narzędzie *lmplz* wymaga podania liczby N będącej rozmiarem największego n-gramu; w pliku ARPA znajdują się wszystkie n-gramy o wielkościach mniejszych lub równych N. Maksymalne N dopuszczalne przez *KenLM* to 6. Przykładowy fragment takiego pliku został pokazany na rysunku 7.1. Każdy n-gram poprzedzony jest logarytmem prawdopodobieństwa: dla unigramów jest to logarytm prawdopodobieństwa, że słowo w pojawi się w języku, a dla $N > 2$ logarytm prawdopodobieństwa warunkowego opisanego wzorami przedstawionymi powyżej. Występują znaki specjalne: *<s>* oznacza początek zdania, *</s>* koniec zdania, a *<unk>* nieznany wyraz (OOV – *out of vocabluary*). Dokumen-

tacja narzędzie *lmplz* określa, że prawdopodobieństwo nieznanego wyrazu traktowanego jako unigram pochodzi z interpolacji i jest wyliczane jako $P(< unk >) = \frac{s_i}{n_v}$, gdzie s_i jest stałą interpolacji, a n_v to liczba wszystkich wyrazów występujących w słowniku modelu.

```
\data\
ngram 1=1034
ngram 2=1615
ngram 3=1451
ngram 4=1171
ngram 5=903

\1-grams:
-3.2413802  <unk>   0
0   <s> -0.9351075
-0.75253505 </s>    0
-2.806667   persName    -0.30103
-3.1954882  głębiami    -0.30103
 ...

\2-grams:
-0.15384252 <s> </s>    0
-0.85887504 persName </s>   0
-0.23032993 głębiami </s>   0
 ...

-1.653291   <s> I siecią    -0.30103
-0.12484646 I siecią złudzeń    -0.30103
-0.12484646 siecią złudzeń usidli   -0.30103
-0.12484646 złudzeń usidli pajęczą. -0.30103
 ...
-0.02801028 Był jak ta harfa eolska,
-0.05283277 ta harfa eolska, Co drży
-0.05283277 eolska, Co drży za każdym
-0.02801028 Co drży za każdym powiewem,
-0.02801028 drży za każdym powiewem, Miotany
```

Rysunek 7.1: Przykładowe fragmenty pliku ARPA.

7.2. Przygotowanie klasyfikatorów n-gramowych

Przygotowanie modeli językowych

Aby zbudować model językowy, zacząć należy od przygotowania danych. Dla każdego ze zbiorów spośród **E3**, **L10** i **L6** zostały wybrane liczby wyrazów n_{model}, z których zostaną utworzone modele językowe dla każdego autora w zbiorze. Dla zbioru **E3** $n_{model} = 300000$, dla **L10** $n_{model} = 10000$, a dla **L6** $n_{model} = 1500$[2]. Z tekstów utworów, poddanych preprocessingom opisanym w podrozdziale 2.1., dla każdego autora zostały utworzone pliki tekstowe zawierające n_{model} wyrazów (do pliku tekstowego były dopisywane kolejne utwory, dopóki liczba wyrazów nie osiągnęła n_{model}

[2]Liczby n_{model} zostały dobrane tak, aby maksymalnie wykorzystywać dostępne dane utworów, tak jak przy tworzeniu zbiorów uczących w rozdziale 2.3..

dla każdego zbioru[3]). Następnie z każdego pliku autora za pomocą narzędzia *lmplz* z pakietu *KenLM* został utworzony plik ARPA. Z plików ARPA biblioteka *KenLM* tworzy modele językowe.

Przygotowanie klasyfikatorów

Dla danej próbki tekstu klasyfikator n-gramowy (NG) jako wynikową przewidywaną klasę oddaje klasę tego autora, którego model językowy uzyskuje najwyższy wynik. Wynik ten jest wyliczany przez bibliotekę *KenLM* jako logarytm z prawdopodobieństwa tego, że dana próbka tekstu jest w języku modelu.

Zbiory testowe

Zbiory testowe o określonej liczbie próbek zostały utworzone z utworów (niewchodzących w skład modeli językowych dla prozy, a mogących wchodzić w skład modeli dla poezji), podzielonych na fragmenty o odpowiedniej liczbie wyrazów dla każdego zbioru. Wielkości zbiorów testowych i liczby wyrazów dla każdego ze zbiorów spośród **E3**, **L10** i **L6** są takie, jak podane w tabelach 2.3, 2.3. i 2.5 w rozdziale 2.3..

7.3. Omówienie uzyskanych wyników

Wykonałam testy klasyfikatora na zbiorach **L10**, **L6** i **E3** dla N będącego maksymalną wielkością n-gramów równego 4, 5 i 6. Wyniki między różnymi N różniły się nieznacznie (zwykle dokładność różniła się o kilka punktów procentowych, przy czym dla zbiorów **L6** i **E3** przeważnie najwyższa dokładność była osiągana przy $N = 6$, a dla **L10** przy $N = 4$)[4]. Tabele 7.1, 7.2 i 7.3 przedstawiają wyniki klasyfikacji dla $N = 6$. We wszystkich tabelach kolorem niebieskim wyróżniony został najwyższy rezultat uzyskany przez klasyfikator NG, kolorem pomarańczowym najgorszy, a kolorami zielonym i czerwonym zaznaczone zostały najlepszy i najgorszy rezultat dla każdego podziału prezentowanego w tabeli.

Zbiór L10

Dla zbioru **L10** wyniki przedstawia tabela 7.1. Dla tego zbioru wyniki są przeważnie nieco lepsze od klasyfikatora losowego – wartość współczynnika κ znajduje

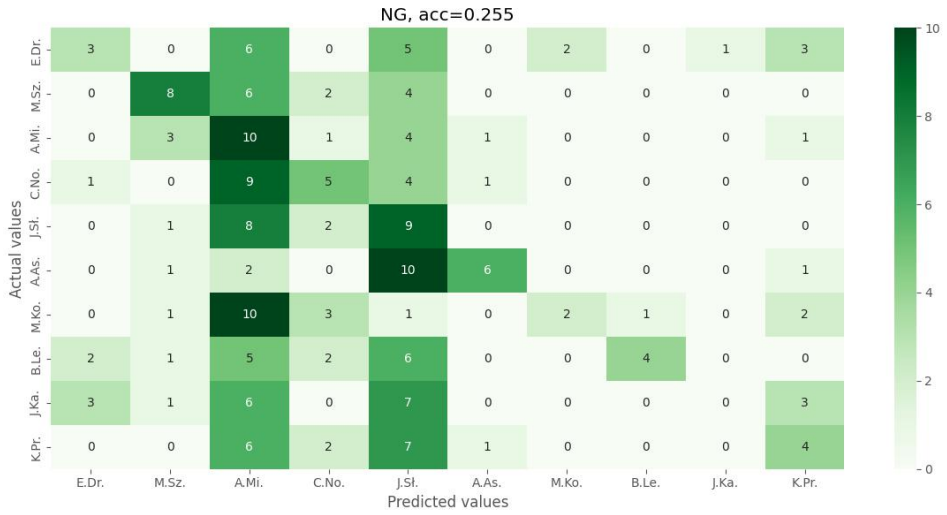
[3]Jeśli jakiś utwór nie mieścił się w pliku, tj. liczba wyrazów po dodaniu utworu przekroczyłaby n_{model}, był brany fragment utworu o odpowiedniej liczbie wyrazów i dla zbiorów o mniejszej liczbie danych, czyli poezji, pozostały fragment był zwracany, aby mógł być wykorzystany w zbiorze testowym, a dla prozy nie był zwracany.

[4]Pominęłam testowanie dla $N < 4$, bo przy wstępnych testach okazało się, że przy $N = 3$ dla zbioru o dużej ilości danych (**E3**) wyniki są nieco gorsze niż przy $N = 4$, a dla zbioru o małej ilości danych (**L10**) wyniki dla $N = 3$ i $N = 4$ prawie się nie różniły.

się między 0.04 a 0.23. Wyróżnia się preprocessing ALOMP – to w tym wypadku osiągana jest największa dokładność wynosząca 31%. Różnica między podziałem, w którym fragmenty zbioru testowego zawierają 30 wyrazów, a takim, w którym fragmenty mają 15 wyrazów, jest niewielka – dla ustalonego preprocessingu dokładności różnią się o między $0.7p.p.$ do $6.2p.p.$. Macierz pomyłek modelu dla zbioru $\mathbf{L10}_{ALOMP}$ i podziału z $n_{words} = 15$ pokazana na rysunku 7.2 wskazuje, że klasyfikator najchętniej wybiera dwie klasy: Mickiewicza i Słowackiego (w macierzach pomyłek modelu NG często można było zaobserwować faworyzowanie 1 lub 2 klas).

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	15	NG	0.202	0.116
A	30	NG	0.14	0.044
ALS	15	NG	0.133	0.05
ALS	30	NG	0.14	0.044
$ALOMP$	15	NG	0.255	0.172
$ALOMP$	30	NG	**0.31**	**0.233**

Tabela 7.1: Dokładność klasyfikatora n-gramowego dla $N = 6$ dla różnych podziałów zbiorów z grupy zbiorów **L10**. Model językowy dla każdego autora uczony był na próbce 1500 wyrazów. Dla $n_{words} = 15$ zbiór testowy zawierał 200 próbek[6], a dla $n_{words} = 30$ – 100 próbek. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.



Rysunek 7.2: Macierz pomyłek klasyfikatora NG dla zbioru $\mathbf{L10}_{ALOMP}$. Liczba wyrazów, z których został utworzony model językowy dla każdego autora, to 1500, wielkość zbioru testowego to 200, a n_{words} =15. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Widać wyraźną faworyzację klas Słowackiego i Mickiewicza.

Zbiór L6

Dla zbioru **L6** (tabela 7.2) wartości współczynnika κ mieszczą się w przedziale $[0.27; 0.64]$, a maksymalna dokładność wynosi 70%. Tak samo jak w wypadku zbioru **L10** zauważalnie najwyższe rezultaty dla różnych podziałów zbioru testowego osiągane są dla preprocessingu ALOMP, a najgorsze, gdy zbiór został poddany wyłącznie anonimizacji. Rysunek 7.3 przedstawia macierz pomyłek modelu dla $\mathbf{L6}_{ALS}$ z podziałem $n_{words} = 100$. Widoczna jest wyraźne przekątna i nieznaczna faworyzacja klasy Słowackiego oraz dobre rozpoznawanie klasy Leśmiana (wszystkie 60 próbek Leśmiana zostało poprawnie zaklasyfikowane i dodatkowo została mu przypisana nadmiarowa próbka).

Preprocessing	n_{words}	Model	Dokładność	Kappa
A	50	NG	0.408	0.29
A	100	NG	0.425	0.31
A	30	NG	0.389	0.267
$ALOMP$	50	NG	**0.7**	**0.64**
$ALOMP$	100	NG	0.658	0.59
$ALOMP$	30	NG	0.614	0.537

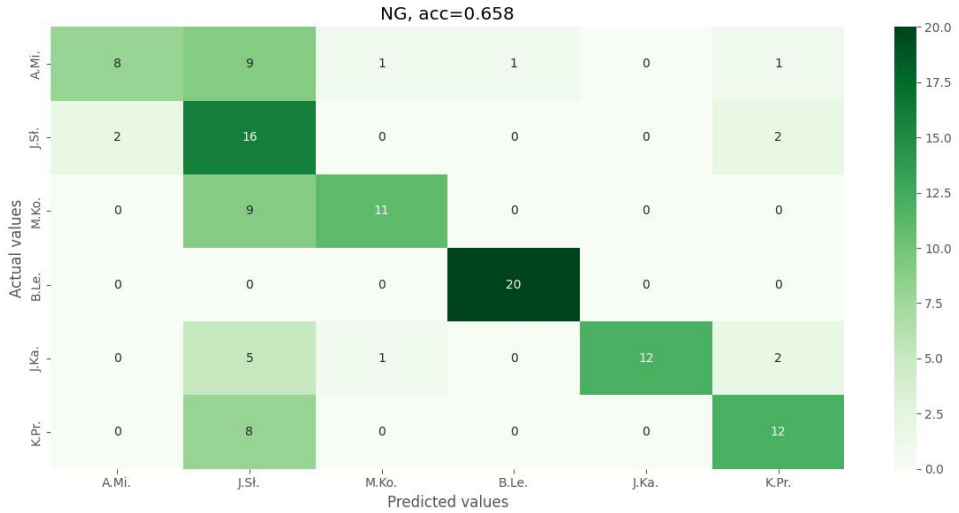
Tabela 7.2: Dokładność klasyfikatora n-gramowego dla $N = 6$ dla różnych podziałów zbiorów z grupy zbiorów **L6**. Model językowy dla każdego autora uczony był na próbce 10000 wyrazów. Dla $n_{words} = 30$ zbiór testowy zawierał 360 próbek, dla $n_{words} = 50$ – 240 próbek, a dla $n_{words} = 100$ – 120 fragmentów. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.

Zbiór E3

Dla zbioru **E3** (tabela 7.3) model NG osiąga dobre wyniki[7], gdy próbki w zbiorze testowym mają dużo wyrazów ($n_{words} = 1000$ i $n_{words} = 300$). Wówczas wartości współczynnika κ mieszczą się w przedziale $[0.89; 0.99]$, a najwyższa dokładność wynosi 99.4%. Dla małej liczby wyrazów w próbce ($n_{words} = 30$) współczynnik κ tylko dla preprocessingu ALOMP nieznacznie przekracza 0.7. Najsłabsze wyniki osiągane są dla preprocessingu ALP. Na rysunku 7.4 przedstawiającym macierz pomyłek dla $\mathbf{E3}_A$ i $n_{words} = 1000$ widać, że klasa Prusa została 2 razy pomylona z klasą Sienkiewicza.

[6] Dla kilku zbiorów z różnymi preprocessingami zabrakło kilku tekstów do 200 (mały zbiór tekstów Drużbackiej), zbiór zawiera jednak co najmniej 195 tekstów.

[7] Za dobry wynik uznaje się $\kappa \geq 0.8$ (rozdział 4.).



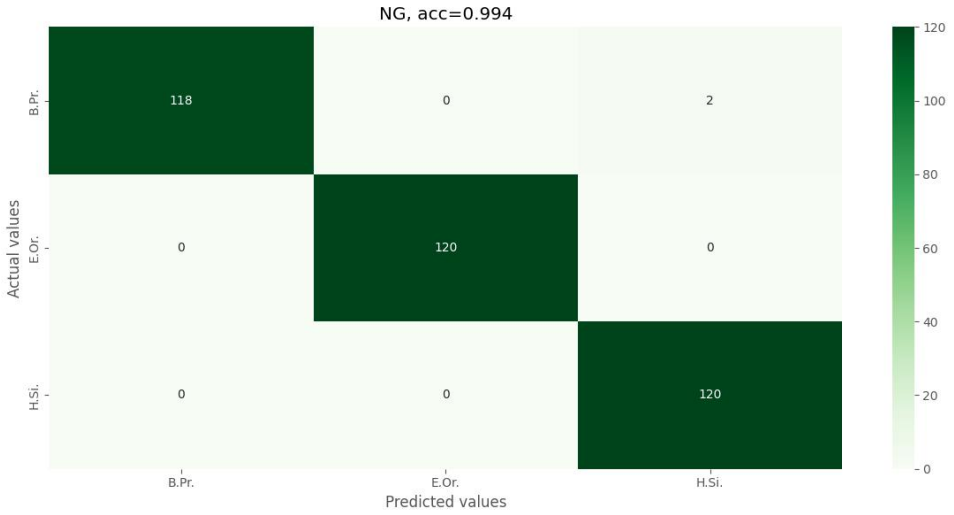
Rysunek 7.3: Macierz pomyłek klasyfikatora NG dla zbioru $\mathbf{L6}_{ALOMP}$. Liczba wyrazów, z których został utworzony model językowy dla każdego autora, to 10000, wielkość zbioru testowego to 120, a n_{words} =100. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów. Widać lekką faworyzację klasy Słowackiego.

Preprocessing	n_{words}	Model	Dokładność	Kappa
ALO	300	NG	0.964	0.946
ALO	1000	NG	**0.994**	**0.992**
ALP	30	NG	0.746	0.619
ALP	300	NG	0.911	0.866
ALP	1000	NG	0.919	0.879
ALOMP	30	NG	0.809	0.714
ALOMP	300	NG	0.957	0.935
ALOMP	1000	NG	0.969	0.954

Tabela 7.3: Dokładność klasyfikatora n-gramowego dla $N = 6$ dla różnych podziałów zbiorów z grupy zbiorów **E3**. Modele językowe dla każdego autora uczone były na próbce 300000 wyrazów. Dla $n_{words} = 30$ i $n_{words} = 300$ zbiór testowy zawierał 1200 próbek, a dla $n_{words} = 1000$ – 360 próbek. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki.

Wnioski

Intuicyjnie mogłoby się wydawać, że do wyznaczenia indywidualnego stylu autora przez model językowy teksty powinny być jak najbliżej oryginału, jednak okazało się, że dla wszystkich zbiorów największą dokładność uzyskuje się z preprocessingiem zostawiającym tylko rzeczowniki, czasowniki i przymiotniki (**O**), w szczególności z preprocessingiem ALOMP. Stąd można wysnuć wniosek, że z punktu widzenia kla-



Rysunek 7.4: Macierz pomyłek klasyfikatora NG dla zbioru $\mathbf{E3}_{ALO}$. Liczba wyrazów, z których został utworzony model językowy dla każdego autora, to 300000, wielkość zbioru testowego to 360, a n_{words} =1000. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów.

syfikatora n-gramowego istotniejsza jest nie kolejność wszystkich słów w tekście, a kolejność słów niosących znaczenie. Wydaje się, że klasyfikator „zauważa", że część słów, szczególnie słów o małym znaczeniu, jest wymagana przez język polski i nie będzie ona wyznaczała indywidualnego stylu autora – chodzi tu przede wszystkim o popularne połączenia wyrazowe, w których jeden wyraz determinuje istnienie innego o małym znaczeniu, na przykład *z pewnością*, *w ogóle*, *mimo że* lub zachodzi korelacja między występowaniem wyrazów, np. *przede wszystkim*, *przede mną*, *nade mną*, *dzień dobry*.

Rozdział 8.

Klasyfikacja tekstów przy pomocy sieci neuronowych

W tym rozdziale pochylimy się nad popularnym w ostatnich latach działem uczenia maszynowego, jakim jest uczenie głębokie. Modelami uczenia głębokiego są sieci neuronowe tworzące wielowarstwowe reprezentacje danych. Sieci neuronowe mogą być między innymi używane do zadania klasyfikacji. Przetestuję zarówno sieci proste, składające się z minimalnej liczby warstw, jak i bardziej skomplikowane sieci rekurencyjne. Tak jak w poprzednich rozdziałach, badana będzie dokładność klasyfikacji na zbiorach **E3**, **L10** i **L6** w zależności od preprocessigu opisanego w 2.2. oraz sposobu podziału zbioru na próbki tekstów opisanego w 2.3..

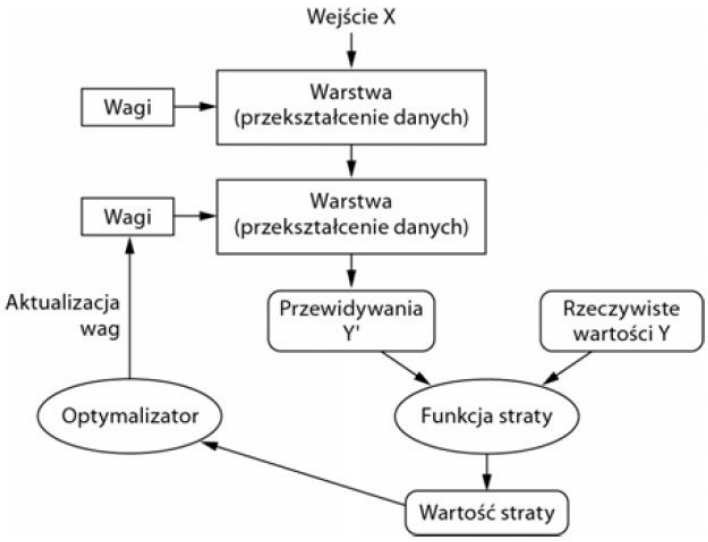
8.1. Teoretyczny opis metod

8.1.1. Działanie sieci neuronowych

Sieci neuronowe składają się z warstw (głębokość sieci to liczba warstw), a każda warstwa określana jest przez wagi, przy pomocy których przekształca dane. Celem jest znalezienie takich wag, aby sieć osiągała jak najlepsze rezultaty. Proces uczenia, pokazany schematycznie na rysunku 8.1, przebiega następująco:

1. Warstwa dostaje odpowiednio przygotowane dane.
2. Warstwa transformuje dane przy użyciu wag, generując przewidywane wartości.
3. Funkcja straty oblicza odległość przewidywanych wartości od wartości prawdziwych.
4. Optymalizator implementujący algorytm propagacji wstecznej dostraja wagi na podstawie obliczonej straty.

Działania te wykonywane są w pętli trenowania (zwykle powtarza się ją kilkadziesiąt razy) i z każdą iteracją wartość straty maleje. Gdy dane wyjściowe są maksymalnie zbliżone do wartości prawdziwych, sieć uznaje się za wytrenowaną [4].



Rysunek 8.1: Schemat uczenia sieci neuronowej. Źródło rysunku: F. Chollet, *Deep Learning* [4].

Podstawową strukturą danych we współczesnych systemach uczenia głębokiego są tensory, czyli uogólnienie macierzy (2-wymiarowych) na struktury o większej liczbie wymiarów, a przekształcenia dokonywane przez sieci to ciąg operacji tensorowych[1] [4]. Różne warstwy wykonują różne przekształcenia, ponadto warstwy mogą mieć różne funkcje aktywacji, które decydują, w jaki dokładnie sposób dane zostaną przekształcone.

Warto przyjrzeć się dokładniej 4. krokowi algorytmu uczenia, czyli kwestii aktualizacji wag. Aby wyniki sieci były jak najlepsze, funkcja straty f_{loss} powinna oddawać jak najmniejsze wartości, trzeba zatem znaleźć jej globalne minimum. Funkcja straty jest różniczkowalna, więc sprowadza się to obliczenia gradientu ∇f_{loss}, wskazującego kierunek największego wzrostu funkcji straty, i rozwiązania równania $\nabla f_{loss}(W) = 0$, gdzie W jest tensorem wag.[2]. Sieci mają jednak zwykle tysiące lub nawet miliony parametrów, więc rozwiązanie tego równania jest bardzo trudne, dlatego też do rozwiązania używa się różnych algorytmów. Jednym z prostszych jest

[1] Operacje tensorowe mogą być interpretowane geometrycznie. F. Chollet podaje trafną metaforę: wyobraźmy sobie dwie kartki położone jedna na drugiej, a następnie zgniecione w kulę – to są dane wejściowe. Sieć neuronowa opracowuje ciąg przekształceń geometrycznych, które pozwolą na rozwinięcie kuli i z powrotem uzyskanie dwóch oddzielnych kartek [4].

[2] Dla uproszczenia podaję jeden tensor W, jednak w praktyce mamy do czynienia z łańcuchem operacji tensorowych, czyli trzeba obliczyć gradient iloczynu funkcji, korzystając z reguły łańcuchowej. Zastosowanie tej reguły prowadzi do algorytmu propagacji wstecznej (określanego też mianem odwrotnego różniczkowania). Współczesne implementacje, takie jak *TensorFlow*, często wykorzystują to tego różniczkowanie symboliczne [4].

algorytm stochastycznego spadku wzdłuż gradientu mini-batch (mini-batch SGD) [4], który polega na tym, że w każdej iteracji na losowo wybranym zbiorze próbek treningowych uruchamia się sieć i uzyskuje przewidywane wartości, oblicza wartość funkcji straty i gradientu w odniesieniu do aktualnych parametrów $W_{current}$ sieci, a na koniec przesuwa się wartości parametrów: $W_{next} = W_{current} - d_{step} * \nabla f_{loss}(W_{current})$, gdzie d_{step} jest odpowiednio dobranym krokiem. W praktyce jednak korzysta się z bardziej skomplikowanych algorytmów nazywanych optymalizatorami uwzględniających wcześniejsze aktualizacje. Popularnym optymalizatorem jest optymalizator *Adam*, który jest wydajny, ma niewielkie wymagania pamięciowe i szczególnie dobrze działa dla problemów z dużą liczbą parametrów. Dokładne działanie optymalizatora opisane jest w artykule D. Kingmy i J. Ba *Adam: a method for stochastic optimization* [17].

8.1.2. Warstwy w pakiecie *Keras*

Keras to pakiet implementujący pythonowy interfejs do uczenia głębokiego. Oparty jest na bibliotece *TensorFlow*[3]. Jego głównym założeniem i zaletą jest prostota i łatwość użycia: pakiet reklamuje się hasłem „Deep learning for humans"[5], a twórca pakietu F. Chollet porównuje łączenie warstw sieci do budowy z klocków LEGO [4]. Budowa sieci przy pomocy pakietu *Keras* przebiega następująco: najpierw wybiera się model sieci, następnie dodaje się odpowiednie warstwy, definiując tym samym architekturę sieci, na koniec model trzeba skompilować, wybierając przy tym odpowiedni optymalizator, funkcję straty i metrykę ocenianą przez model podczas uczenia i testowania, i tak zbudowana sieć jest gotowa do trenowania.

Dla liniowego stosu warstw, gdy każda warstwa ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy (nasze zadanie klasyfikacji) odpowiedni jest model sekwencyjny *Sequential*. Pakiet *Keras* oferuje wiele rodzajów warstw, omówię 4 z nich, które zdecydowałam się użyć w projekcie:

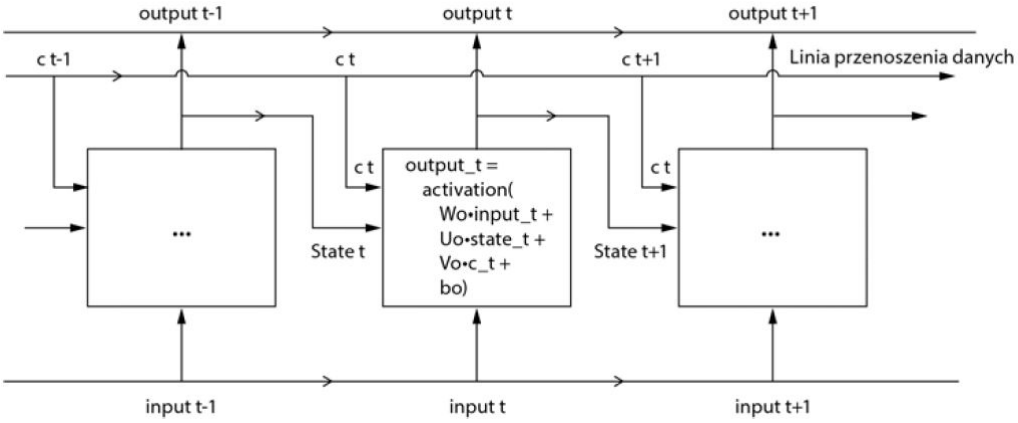
- Dense – warstwa gęsto połączona, używana do przetwarzania prostych dwuwymiarowych tensorów o kształcie (liczba próbek, liczba cech). Na wejściu oczekuje argumentu *liczba jednostek wyjściowych*, który jest jednym z wymiarów wektora wyjściowego. Warstwa ta jest implementacją operacji: $v_o = f_a(W * v_i + b)$, gdzie v_i i v_o to wektor wejściowy i wyjściowy, W i b, czyli tensor wag i bias są atrybutami warstwy, a f_a to funkcja aktywacji. Najbardziej podstawową funkcją aktywacji jest *relu*, będąca odpowiednikiem funkcji $\max(0, x)$. Odpowiednią funkcją aktywacji ostatniej warstwy dla zadania wieloklasowej klasyfikacji jednoetykietowej jest *softmax*, która na wejściu dostaje liczbę klas n_c i zwraca tablicę wartości prawdopodobieństwa każdej klasy (wektor długości n_c liczb z przedziału $[0; 1]$ sumujących się do 1).

[3] *TensorFlow* jest domyślnym i bardziej popularnym backendem pakietu *Kreas*, ale dostępny jest też backend *Theano*.

- Embedding – warstwę tę, jak zaznacza F. Chollet, „najlepiej jest rozumieć jako słownik mapujący całkowitoliczbowe indeksy oznaczające określone słowa na gęste wektory"[4, s. 196] (por. przykład z państwami i stolicami z podrozdziału 3.3.). Na wejściu przyjmuje tensor wielkości (liczba próbek, długość sekwencji) – sposób zamiany tekstów na tensor został opisany w podrozdziale 3.3. – i zwraca tensor o kształcie (liczba próbek, długość sekwencji, liczba wymiarów osadzenia). Liczba wymiarów osadzenia słowa to długość wektora, za pomocą którego jest reprezentowane każde słowo. Podaje się ją jako argument warstwy; drugim wymaganym argumentem jest liczba tokenów występujących w reprezentacji Word Embedding. Wektory osadzeń inicjalizowane są albo losowo, albo wagami wziętymi z wytrenowanego modelu (np. *GloVe*). Na wyjściu z tej warstwy każda próbka tekstu jest reprezentowana jako dwuwymiarowy tensor składający się z wektorów osadzeń reprezentujących kolejne słowa sekwencji.

- Flatten – warstwa spłaszczająca trójwymiarowy tensor osadzeń uzyskany jako wyjście warstwy *Embedding* do dwuwymiarowego tensora oczekiwanego przez warstwę *Dense*. Spłaszczenie dokonuje się poprzez konkatenację wektorów osadzeń, co oznacza, że każda próbka tekstu jest reprezentowana przez wektor o długości iloczynu liczby wymiarów osadzenia i długości sekwencji. Wato zaznaczyć, że dzięki konkatenacji nie jest tracona informacja o kolejności słów w próbkach, jednak wektor taki jest duży – jeśli jako metodę spłaszczania wybrać na przykład sumę wektorów osadzeń zamiast konkatenacji wektor reprezentujący każdą próbkę będzie dużo mniejszy.

- LSTM – warstwa rekurencyjna. Sieci rekurencyjne przetwarzają sekwencję, iterując się po kolejnych elementach i utrzymując stan. Charakteryzuje je funkcja kroku i są parametryzowane nie tylko przez tensor wag W przetwarzający wektor wejściowy, ale również przez tensor U przetwarzający stan. W wersji podstawowej (np. warstwa *SimpleRNN* z pakietu *Keras*) występuje problem zanikającego gradientu, który powoduje, że sieć nie jest w stanie nauczyć się większych zależności i osiąga często słabe wyniki. Architektura LSTM (Long Short-Term Memeory) rozwiązuje problem zanikającego gradientu poprzez dodanie do podstawowej wersji sieci rekurencyjnej sposobu przekazywania informacji między krokami czasu. F. Chollet opisuje ten sposób, jako „pas transmisyjny biegnący równolegle do przetwarzanej sekwencji. Informacje z sekwencji mogą [. . .] zostać przeniesione na ten pas i trafić do następnego kroku czasu bez [. . .] modyfikacji" [4, s. 212]. Rysunek 8.2 pokazuje schemat warstwy LSTM, w tym pseudokod obliczania wektora wyjściowego. Tak jak warstwa *Dense*, warstwa *LSTM* na wejściu oczekuje argumentu *liczba jednostek wyjściowych*. Jest odpowiednia do przetwarzania tensorów zwróconych przez warstwę *Embedding*, bo oczekuje na wejściu trzywymiarowych tensorów (liczba próbek, liczba kroków czasu, liczba cech wejściowych), przy czym liczba kroków czasu będzie długością sekwencji, a liczba cech wyjściowych to liczba wymiarów osadzenia warstwy *Embedding*.



Rysunek 8.2: Schemat warstwy LSTM. Źródło rysunku: F. Chollet, *Deep Learning*[4].

Przy kompilowaniu sieci zdecydowałam się na wspomniany wyżej optymalizator *Adam*, funkcję straty *categorical_ crossentropy* i metrykę *accuracy*, opisaną w rozdziale 4.. Funkcja straty *categorical_ crossentropy*, czyli entropia krzyżowa, jest odpowiednia dla problemu jednoetykietowej klasyfikacji wieloklasowej i informuje o poziomie niepewności między szacowanym rozkładem prawdopodobieństwa p, a rozkładem prawdziwym t. W naszym wypadku dla n_c klas, prawdziwego rozkładu prawdopodobieństwa klas $t = (t_1, \ldots, t_{n_c})$, $t_i = \frac{1}{n_c}$ (bo zbiory są idealnie zrównoważone) i rozkładu prawdopodobieństwa $p = (p_1, \ldots, p_{n_c})$ uzyskanego przez warstwę *Dense* z funkcją aktywacji *softmax*, entropię krzyżową definiuje się wzorem:

$$L(t,p) = -\sum_{i=1}^{n_c} t_i \log(p_i) = -\frac{1}{n_c}\sum_{i=1}^{n_c} \log(p_i) \tag{8.1}$$

8.1.3. Dobór architektury warstw i parametrów sieci

Zdecydowałam się na przetestowanie kilku architektur sieci, różniących się warstwami i oczekiwaną reprezentacją danych wejściowych. Warstwy z pakietu *Keras* mają wiele parametrów i przeważnie zostawiłam te parametry z wartościami domyślnymi, jednak każda sieć ma kilka parametrów, których wartość ustaliłam eksperymentalnie, ucząc sieć kilkukrotnie i wybierając takie, dla których sieć osiąga najwyższą dokładność. Najważniejsze spośród parametrów omówię przy opisywaniu poszczególnych architektur sieci:

- Sieć podstawowa – przyjmuje dane w reprezentacji zliczających tokeny: Bag of Words oraz Bag of Word Pieces. Składa się warstwy *Dense* z funkcją aktywacji *relu* i liczbą jednostek wyjściowych równą 64 i warstwy wyjściowej *Dense* z funkcją aktywacji *softmax*. Dane w reprezentacjach Bag of Words oraz Bag of Word Pieces[4] zostały ograniczone do k najistotniejszych cech przy pomocy

[4]Warto podkreślić, że w tym wypadku przy zastosowaniu reprezentacji Bag of Words i Bag of Word Pieces nie mamy informacji o kolejności słów w tekście.

funkcji *SelectKBest* z biblioteki *scikit-learn* z testem statystycznym χ^2. Dla zbioru **E3** wartość k ustaliłam na 8000, dla **L6** $k = 5000$, a dla **L10** $k = 1000$, czyli sieć na wejściu dostaje zbiór wektorów zliczeń słów o długościach odpowiednio 8000, 5000 i 1000.

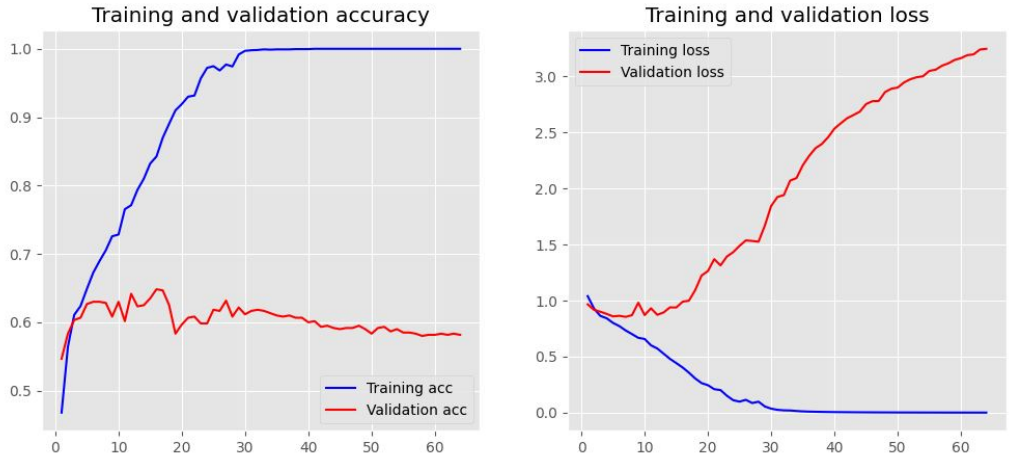
- Sieć z embeddingiem – przyjmuje na wejściu dwuwymiarowy tensor reprezentujący próbki tekstów jako całkowitoliczbowe sekwencje (por. podrozdział 3.3.) i przetwarza je warstwą *Embedding*. Liczba wymiarów osadzenia, czyli długość wektora, za pomocą którego jest reprezentowane słowo, ma wartość 32 – na wyjściu z tej warstwy każda próbka tekstu jest reprezentowana jako dwuwymiarowy tensor przedstawiający kolejne słowa próbki jako wektory długości 32. Kolejną warstwą jest *Flatten*, która dla wszystkich próbek konkatenuje wektory osadzeń, a ostatnią warstwa *Dense* z funkcją aktywacji *softmax*. Liczba słów występujących w sekwencjach została ograniczona przez *Tokenizer* z pakietu *Keras* do 1000 dla zbioru **E3** i 800 dla zbiorów **L6** i **L10**.

- Sieć LSTM z embeddingiem – przyjmuje na wejściu dwuwymiarowy tensor reprezentujący próbki tekstów jako całkowitoliczbowe sekwencje i składa się z warstw: *Embedding* (trenowalnej), *LSTM* z liczbą jednostek wyjściowych ustawioną na 128 oraz *Dense* z funkcją aktywacji *softmax*. Liczba słów została ograniczona przez *Tokenizer* z pakietu *Keras* do 1000 najczęściej występujących dla zbiorów **E3** i **L10** i 800 najczęściej występujących dla zbioru **L6** (wartości te wyznaczyłam eksperymentalnie w kilku próbach uczenia sieci).

- Sieć z embeddingiem GloVe – sieć przyjmuje na wejściu dwuwymiarowy tensor reprezentujący próbki tekstów jako całkowitoliczbowe sekwencje i składa się z warstwy *Embedding*, jednak nie jest ona trenowalna, a wagi wzięte są z wyuczonego dla języka polskiego modelu *GloVe* (por. podrozdział 3.3.) o liczbie wymiarów osadzenia wynoszącej 100, warstwy LSTM z liczbą jednostek wyjściowych ustawioną na 128, warstwy *Dense* z liczbą jednostek wyjściowych ustawioną na 64 i wyjściowej warstwy *Dense* z funkcją aktywacji *softmax*. Maksymalna długość sekwencji została ograniczona do 100, a liczba słów została ograniczona przy tokenizacji za pomocą *Tokenizera* z pakietu *Keras* do 800 najczęściej występujących dla wszystkich zbiorów (wartości te wyznaczyłam eksperymentalnie w kilku próbach uczenia sieci).

Gdy architektura sieci jest zdefiniowana, sieć trzeba wytrenować metodą *fit*. Przy wywoływaniu tej metody określa się kilka parametrów. Liczba epok, czyli powtórzeń procesu trenowania, została ustawiona na 32 dla sieci podstawowej i na 64 w wypadku sieci zawierających embedding. Liczba powtórzeń może być jednak mniejsza, jeśli dokładność przez 5 epok się będzie pogarszać (określa to *callback EarlyStopping* z pakietu *Keras*) – chcemy zapobiec zjawisku nadmiernego dopasowania (overfitting). Sieci zwykle nie trenuje się na całym zbiorze treningowym naraz, ale w seriach (*batch*) o określonym rozmiarze; rozmiar jest stały dla wszystkich zbiorów i wynosi 64. W metodzie *fit* określa się też sposób podziału zbioru treningowego na zbiór walidacyjny

i zbiór treningowy właściwy – został on podzielony w stosunku 1 : 4. Proces uczenia można kontrolować na wykresach zależności numeru epoki od dokładności i numeru epoki od straty na zbiorze treningowym i walidacyjnym. Przykładowe wykresy zostały przedstawione na rysunku 8.3 – w tym wypadku zatrzymanie przy rosnącej wartości straty (*EarlyStopping*) było wyłączone, więc doszło do nadmiernego dopasowania.



Rysunek 8.3: Przykładowe wykresy zależności numeru epoki od dokładności i numeru epoki od wartości straty na zbiorze treningowym i walidacyjnym przy uczeniu sieci z embeddingiem *GloVe*. Widoczne jest nadmierne dopasowanie.

8.2. Omówienie uzyskanych wyników

Uruchomiłam wszystkie opisane powyżej architektury sieci na zbiorach **E3**, **L6** i **L10**. Okazało się, że dla sieci podstawowej wyniki są wyraźnie lepsze niż dla którejkolwiek sieci z embeddingiem. Dlatego też dla każdego zbioru przedstawiam osobną tabelę z wybranymi wynikami sieci podstawowej i zbiorczą zawierającą najwyższe wyniki sieci z embeddingiem dla różnych podziałów. Dla sieci podstawowej to, czy dane były w reprezentacji Bag of Words, czy Bag of Word Pieces, nie wpływało znacząco na wyniki (zwykle różnica między reprezentacjami wynosiła około $1 p.p.$), więc w tabelach prezentuję tylko wyniki dla reprezentacji Bag of Word Pieces. Standardowo, we wszystkich tabelach kolorem niebieskim wyróżniony został najwyższy rezultat uzyskany przez sieci, kolorem pomarańczowym najgorszy, a kolorami zielonym i czerwonym zaznaczone zostały najlepszy i najgorszy rezultat dla każdego podziału prezentowanego w tabeli.

Zbiór L10

Dla zbioru **L10** wyniki przedstawiają tabele 8.1 i 8.2. Dokładność dla sieci przy $n_{words} = 30$ i preprocessingu ALOMP wynisi 57%, co jest dobrym wynikiem, jak

na ten zbiór – podobne dokładności osiągały klasyfikatory Bayesowskie i SVC. Podział zbioru zauważalnie wpływa na wyniki w wypadku sieci podstawowej – różnica dokładności między $n_{words} = 15$ a $n_{words} = 30$ to 22.5$p.p.$ – a w wypadku sieci z embeddingiem znacząca różnica (11.5$p.p.$) występuje tylko przy embeddingu *GloVe*, a samodzielne trenowanie warstwy *Embedding* powoduje, że sposób podziału jest nieznaczący. Spośród sieci z embeddingiem najwyższą dokładność ma sieć, która korzysta z wytrenowanej przestrzeni osadzeń *GloVe*. Zauważmy, że najdokładniejsze klasyfikacje dla wszystkich sieci osiągane są przy preprocessingach ALS, ALO i ALOMP. W macierzy pomyłek przedstawionej na rysunku 8.4 dla $n_{words} = 30$ i ALOMP można spostrzec, że sieć szczególnie dobrze nauczyła się rozpoznawania klasy Sępa-Szarzyńskiego.

Preprocessing	n_{words}	Dokładność	Kappa
A	15	0.235	0.15
ALS	15	0.345	0.272
ALP	30	0.29	0.211
ALOMP	30	**0.57**	**0.522**

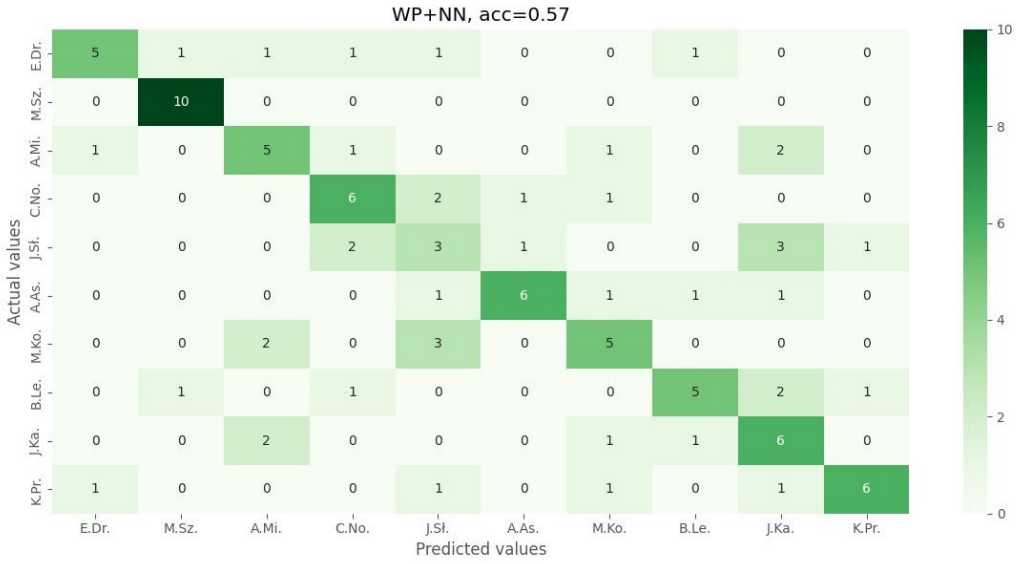
Tabela 8.1: Dokładność sieci podstawowej dla różnych podziałów zbiorów z grupy zbiorów **L10**. Dla $n_{words} = 15$ zbiór treningowy zawierał 1000 próbek, a testowy 200, a dla $n_{words} = 30$ wielkość zbioru treningowego to 500, a testowego 100. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki dla reprezentacji Bag of Word Pieces.

Preprocessing	n_{words}	Rodzaj sieci	Dokładność	Kappa
ALO	15	EMBEDDING	0.265	0.183
ALS	30	EMBEDDING	0.25	0.167
ALS	15	LSTM	0.295	0.217
ALO	30	LSTM	0.33	0.256
ALOMP	15	GLOVE	0.275	0.194
ALOMP	30	GLOVE	**0.39**	**0.322**

Tabela 8.2: Dokładność sieci z embeddingiem (podstawowej, LSTM i z embeddingiem *GloVe*) dla różnych podziałów zbiorów z grupy zbiorów **L10**. Dla $n_{words} = 15$ zbiór treningowy zawierał 1000 próbek, a testowy 200, a dla $n_{words} = 30$ wielkość zbioru treningowego to 500, a testowego 100. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia najwyższe wyniki dla poszczególnych podziałów.

Zbiór L6

Tabele 8.3 i 8.4 przedstawiają wyniki uczenia sieci na zbiorze **L6**. W jednym uczeniu sieci podstawowej (ALS i $n_{words} = 30$) osiągane wyniki są dobre – współ-



Rysunek 8.4: Macierz pomyłek sieci podstawowej dla zbioru $\mathbf{L10}_{ALOMP}$ z reprezentacją Bag of Word Pieces. Wielkość zbioru treningowego to 500, wielkość zbioru testowego to 100, a n_{words} =30. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów.

czynnik κ wynosi 0.8. Różnica między siecią podstawową a sieciami z embeddingem jest znacząca – najgorsza dokładność osiągnięta przez sieć podstawową jest niewiele gorsza od najwyższych dokładności sieci z embeddingiem (a lepsza niż najlepszy wynik sieci LSTM). Tak jak dla zbioru **L10** przy samodzielnym trenowaniu warstwy *Embedding* długość próbki jest nieznacząca. Zwróćmy uwagę na dość małą rozbieżność najlepszych wyników rożnych sieci z embeddingiem (dla ustalonego n_{words} maksymalnie $6.7p.p.$). Rysunek 8.5 przedstawia macierz pomyłek dla podstawowej sieci z embeddingiem i $n_{words} = 100$ – sieć najlepiej rozpoznaje Leśmiana, najsłabiej Mickiewicza, a klasy Przerwy-Tetmajera i Kasprowicza są najczęściej mylone, o czym świadczy ciemniejszy prostokąt w prawym dolnym rogu.

Zbiór E3

Sieć podstawowa z dowolnym podziałem zbioru i rodzajem preprocessingu daje bardzo dobre rezultaty dla $n_{words} = 300$ i $n_{words} = 1000$ – wartość współczynnika κ mieści się odpowiednio w przedziałach $[0.896; 0.982]$ i $[0.967; 1.0]$. Odnotujmy, że są to najwyższe dokładności spośród wszystkich omawianych klasyfikatorów. Przy $n_{words} = 30$ ta sieć uzyskuje porównywalnie dobre wyniki przy uczeniu na zbiorze **L6** i **E3** – najwyższa wartość współczynnika κ w obu wypadkach wynosi około 0.67. Wśród sieci z embeddingiem warto zwrócić uwagę na dwie sprawy. Po pierwsze, na niższą dokładność sieci korzystającej z gotowego embeddingu *GloVe* w porównaniu z sieciami z samodzielnie uczoną warstwą *Embedding*. Po drugie na fakt, że dla wszystkich sieci najwyższe rezultaty osiągane są, gdy $n_{words} = 300$, co jest rezulta-

Preprocessing	n_{words}	Dokładność	Kappa
A	100	0.675	0.61
ALS	100	**0.833**	**0.8**
ALP	50	0.675	0.61
ALP	30	0.567	0.48
ALOMP	50	0.804	0.765
ALOMP	30	0.728	0.673

Tabela 8.3: Dokładność sieci podstawowej dla różnych podziałów zbiorów z grupy zbiorów **L6**. Dla $n_{words} = 30$ zbiór treningowy zawierał 1800 próbek, a testowy 360, dla $n_{words} = 50$ wielkość zbioru treningowego to 1200, a testowego 240, a dla $n_{words} = 100$ odpowiednio 600 i 120. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki dla reprezentacji Bag of Word Pieces.

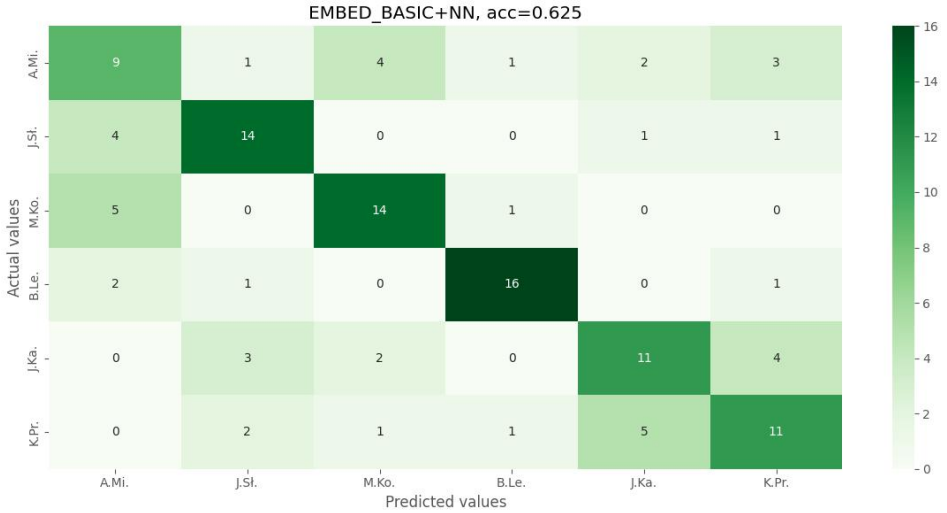
Preprocessing	n_{words}	Rodzaj sieci	Dokładność	Kappa
ALOMP	50	EMBEDDING	0.588	0.505
ALOMP	100	EMBEDDING	**0.625**	**0.55**
ALOMP	30	EMBEDDING	0.569	0.483
ALOMP	50	LSTM	0.554	0.465
ALS	100	LSTM	0.558	0.47
ALOMP	30	LSTM	0.542	0.45
ALOMP	50	GLOVE	0.55	0.46
ALOMP	100	GLOVE	**0.625**	**0.55**
ALOMP	30	GLOVE	0.506	0.407

Tabela 8.4: Dokładność sieci z embeddingiem (podstawowej, LSTM i z embeddingiem *GloVe*) dla różnych podziałów zbiorów z grupy zbiorów **L6**. Dla $n_{words} = 30$ zbiór treningowy zawierał 1800 próbek, a testowy 360, dla $n_{words} = 50$ wielkość zbioru treningowego to 1200, a testowego 240, a dla $n_{words} = 100$ odpowiednio 600 i 120. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia najwyższe wyniki dla poszczególnych podziałów.

tem różniącym się od wyników wszystkich testowanych dotąd klasyfikatorów (zawsze najlepsze wyniki były uzyskiwane, gdy $n_{words} = 1000$) i może być to spowodowane doborem parametrów sieci.

Wnioski

Warto zaznaczyć, że sieci zwykle trenowane są na kilkudziesięciu lub nawet kilkuset tysiącach przykładów. W naszym wypadku mamy stosunkowo mały zbiór danych, wielkości zbiorów treningowych mają od 500 do 3000 próbek, co znacząco wpływa na wyniki.



Rysunek 8.5: Macierz pomyłek podstawowej sieci z embeddingiem dla zbioru **L6**$_{ALOMP}$. Wielkość zbioru treningowego to 600, wielkość zbioru testowego to 120, a n_{words} =100. Klasy podpisane są pierwszą literą imienia i dwiema pierwszymi nazwiska autorów.

Preprocessing	n_{words}	Dokładność	Kappa
A	300	0.931	0.896
ALS	300	0.988	0.982
ALO	300	0.988	0.981
ALO	1000	0.978	0.967
ALP	30	0.622	0.434
$ALOMP$	30	0.783	0.675
$ALOMP$	1000	**1.0**	**1.0**

Tabela 8.5: Dokładność sieci podstawowej dla różnych podziałów zbiorów z grupy zbiorów **E3**. Dla $n_{words} = 30$ i $n_{words} = 300$ zbiory treningowe zawierały 3000 próbek, a testowe 1200, a dla $n_{words} = 1000$ wielkość zbioru treningowego to 900, a testowego 360. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia wybrane wyniki dla reprezentacji Bag of Word Pieces.

Wszystkie rodzaje sieci najdokładniej klasyfikowały teksty poddane preprocessingom ALS, ALO i ALOMP. Oznacza to, że sieci lubią wybór najważniejszych informacji z tekstu. Nie powinno to dziwić, biorąc pod uwagę, że liczba wag sieci jest proporcjonalna do liczby wyrazów, więc wyrazy powinny nieść jak najwięcej informacji.

Sieć podstawowa jest skuteczniejsza niż jakakolwiek sieć z embeddingiem. Wpływ na takie wyniki mogą mieć co najmniej 2 czynniki. Po pierwsze, jak zaznaczyłam wyżej, mamy stosunkowo mały zbiór danych, wielkości zbiorów treningowych mają od 500 do 3000 próbek. Warstwa embeddingu jest czuła na liczbę próbek w zbiorze

Preprocessing	n_{words}	Rodzaj sieci	Dokładność	Kappa
ALO	30	EMBEDDING	0.69	0.535
ALS	300	EMBEDDING	**0.98**	**0.97**
ALS	1000	EMBEDDING	0.975	0.962
ALOMP	30	LSTM	0.662	0.494
ALS	300	LSTM	0.958	0.936
ALOMP	1000	LSTM	0.869	0.804
ALOMP	30	GLOVE	0.616	0.424
ALS	300	GLOVE	0.85	0.775
ALS	1000	GLOVE	0.767	0.65

Tabela 8.6: Dokładność sieci z embeddingiem (podstawowej, LSTM i z embeddingiem *GloVe*) dla różnych podziałów zbiorów z grupy zbiorów **E3**. Dla $n_{words} = 30$ i $n_{words} = 300$ zbiory treningowe zawierały 3000 próbek, a testowe 1200, a dla $n_{words} = 1000$ wielkość zbioru treningowego to 900, a testowego 360. Testy zostały przeprowadzone dla wszystkich rodzajów preprocessingu prezentowanych w podrozdziale 2.1., a tabela przedstawia najwyższe wyniki dla poszczególnych podziałów.

uczącym, bo na podstawie tych próbek tworzy semantyczną sieć powiązań między słowami. Po drugie, parametry sieci z embeddingiem, których jest więcej niż sieci podstawowej, mogły nie zostać optymalnie dobrane, bo sprawdzałam wybrane wartości parametrów na kilku lub kilkunastu powtórzeniach uruchomienia sieci i wybraną wartość stosowałam dla wszystkich zbiorów i preprocessingów, a część parametrów zostawiłam z wartością domyślną. Prawdopodobnie każdy zbiór z preprocessingiem mógłby mieć swój własny, dobrany zestaw parametrów.

Sieci z wektorami osadzeń uczonymi na zbiorze są skuteczniejsze niż sieć z gotowym embeddingiem z *GloVe* dla zbioru **E3**, osiągają podobną skuteczność dla zbioru **L6** i są mniej skuteczne dla zbioru **L10**. Skorzystanie z gotowych wektorów osadzeń jest sugerowane w przypadku małej objętości zbioru uczącego, natomiast dla większej ilości danych warto trenować przestrzeń dla problemu, bo w różnych problemach ważne są różne zależności semantyczne (pisze o tym między innymi F. Chollet w książce *Deep learning* [4]). Tę teorię potwierdzają wspomniane różnice skuteczności: zbiór **L10** ma małą objętość, więc przestrzeń osadzeń nie jest właściwie wytrenowana, a objętość zbioru **E3** jest wystarczająca, żeby odpowiednio wytrenować przestrzeń osadzeń.

Rozdział 9.

Klasyfikacja tekstów przy pomocy narzędzia ChatGPT

W niniejszym rozdziale przetestuję popularne ostatnio narzędzie – ChatGPT – do zadania ustalania autorstwa. Do testów wykorzystam teksty spoza zbiorów **E3**, **L6** i **L10** opisane w podrozdziale 2.4.. Istotne będzie nie tylko to, jakiego pisarza ChatGPT uznaje za autora tekstu, ale również wyjaśnienie chatbota, dlaczego wybrał konkretną klasę.

9.1. Krótki opis ChatGPT i sposobu wykorzystania go do zadania klasyfikacji

ChatGPT [20] to chatbot stworzony przez laboratorium badawcze *OpenAI* i uruchomiony w listopadzie 2022. Jego celem jest prowadzenie dialogu na dowolny temat z użytkownikami w języku naturalnym w sposób jak najbardziej zbliżony do człowieka. Wykorzystuje model GPT[1] (Generative Pre-trained Transformer), czyli rodzaj modelu językowego LLM (Large Language Model) opartego na architekturze sieci neuronowej Transformer. Model ten został wytrenowany na bardzo dużym zbiorze tekstów z różnych źródeł – zbiór treningowy modelu GPT-3 zawierał około 500 miliardów tokenów i większość tego zbioru pochodziła z Common Crawl, czyli publicznie dostępnego zbioru danych pochodzących z przeszukiwania internetu, ale wykorzystane zostały też korpusy książek oraz Wikipedia. Warto zaznaczyć, że 93% danych uczących było w języku angielskim, a pozostałe 7% stanowiły teksty w innych językach, w tym 0.156% zbioru reprezentowało język polski (przekłada się to na $303.8 * 10^6$ słów[2]) [2].

[1] Obecnie wykorzystywana wersja to GPT-3.5, ale w edycji płatnej dostępna jest już wersja GPT-4.

[2] Dla porównania cała pula danych w Narodowym Korpusie Języka Polskiego to 1500 milionów słów, a wersja zrównoważona zawiera 250 milionów słów [23].

Biorąc pod uwagę, że ChatGPT jest w stanie generować odpowiedzi na dowolny temat, zdecydowałam się na próbę przetestowania ChatGPT jako narzędzia do określania autorstwa. W ogólności cały eksperyment sprowadzał się do podania chatbotowi testowanego tekstu i zadania pytania: „kto jest autorem tego tekstu?". Teksty testowe zostały opisane w podrozdziale 2.4. i są to: pastisz Słowackiego autorstwa Wyki, wiersz o niepewnym autorstwie przypisywany Mickiewiczowi i list Sienkiewicza. Jako że ChatGPT ma ograniczoną długość sekwencji wejściowej wprowadzanej przez użytkownika (4096 tokenów), skorzystałam z dwóch podejść uczenia maszynowego:

- Zero-Shot learning (ZSL) – w tym podejściu nie podawałam żadnych próbek tekstu, na których ChatGPT mógłby się uczyć. Testowałam to podejście w dwóch wariantach: w pierwszym nie podałam listy potencjalnych autorów, a w drugim podałam – dla pastiszu Słowackiego i niepewnego tekstu przypisywanego Mickiewiczowi byli to wszyscy autorzy, których teksty występują w zbiorze **L10**, a dla listu Sienkiewicza – Orzeszkowa, Sienkiewicz i Prus.
- Few-Shot learning (FSL) – w tym podejściu model dostaje kilka próbek treningowych i na ich podstawie klasyfikuje próbki tekstowe. W wypadku pastiszu Słowackiego i niepewnego tekstu przypisywanego Mickiewiczowi ChatGPT na wejściu dostawał po 4 próbki tekstu po 100 tokenów na każdego autora ze zbioru **L6** (Mickiewicz, Słowacki, Konopnicka, Kasprowicz, Przerwa-Tetmajer i Leśmian), a w wypadku listu Sienkiewicza chatbot dostawał po 3 fragmenty po 300 tokenów dla każdego autora ze zbioru **E3**. Tekst nie był anonimizowny. Przy podejściu FSL mamy 2 opcje: można zaznaczyć, że ChatGPT powinien klasyfikować tylko na podstawie wiedzy uzyskanej z fragmentów, albo pozwolić chatbotowi na korzystanie także z wiedzy spoza fragmentów.

Utwory testowe nie były zanonimizowane, ale nie zawierały tytułów. Każdą odpowiedź ChatGPT regenerowałam 10 razy, aby zobaczyć, w jakim stopniu kolejne odpowiedzi różnią się od siebie.

9.2. Wyniki uzyskane przy pomocy ChatGPT

Pastisz Słowackiego autorstwa Wyki

W podejściu ZSL bez podania listy poetów ChatGPT w każdym spośród 10 powtórzeń pytania wymienił innego autora, nie rozpoznając nawet poprawnie epoki, byli to bowiem mniej lub bardziej znani poeci tworzący od romantyzmu po współczesność. Co więcej, prawie w każdej odpowiedzi chatbot dodatkowo podawał dzieło lub tom, z którego pochodzi wiersz, co brzmiało prawdopodobnie, jednak było zupełnie bezsensowne, na przykład według chatbota tekst raz pochodził z wiersza Norwida *Fortepian Szopena*, innym razem był fragmentem wiersza *Cyprysy* Wyspiańskiego

(wiersz taki nie istnieje) lub pochodził ze zbioru poezji Tetmajera *Mózg Zbigniewa Herberta* z 1901 roku (taki zbiór nie istnieje, a Herbert urodził się w 1924 roku).

W wypadku zawężenia potencjalnych autorów wiersza do listy poetów ze zbioru **L10** ChatGPT w 7 powtórzeniach stwierdził autorstwo Norwida, 2 razy Przerwy-Tetmajera i raz Sępa-Szarzyńskiego. Tym razem przy pytaniu „Kto spośród wyżej wymienionych poetów jest autorem powyższego wiersza?" odpowiedzi były krótsze i bardziej konkretne, przeważnie były modyfikacją zdania: „Autor powyższego wiersza to Cyprian Kamil Norwid.", a nonsensowna odpowiedź zawierająca tytuł nieistniejącego wiersza pojawiła się tylko raz. Zadałam również pytanie, jakie cechy wiersza wskazują na autorstwo Norwida. Odpowiedź, pokazana na rysunku 9.1 wydaje się sensowna, odwołuje się zarówno do wiersza, jak i do stylu charakterystycznego dla Norwida i jest bardzo zbliżona do odpowiedzi człowieka na to pytanie.

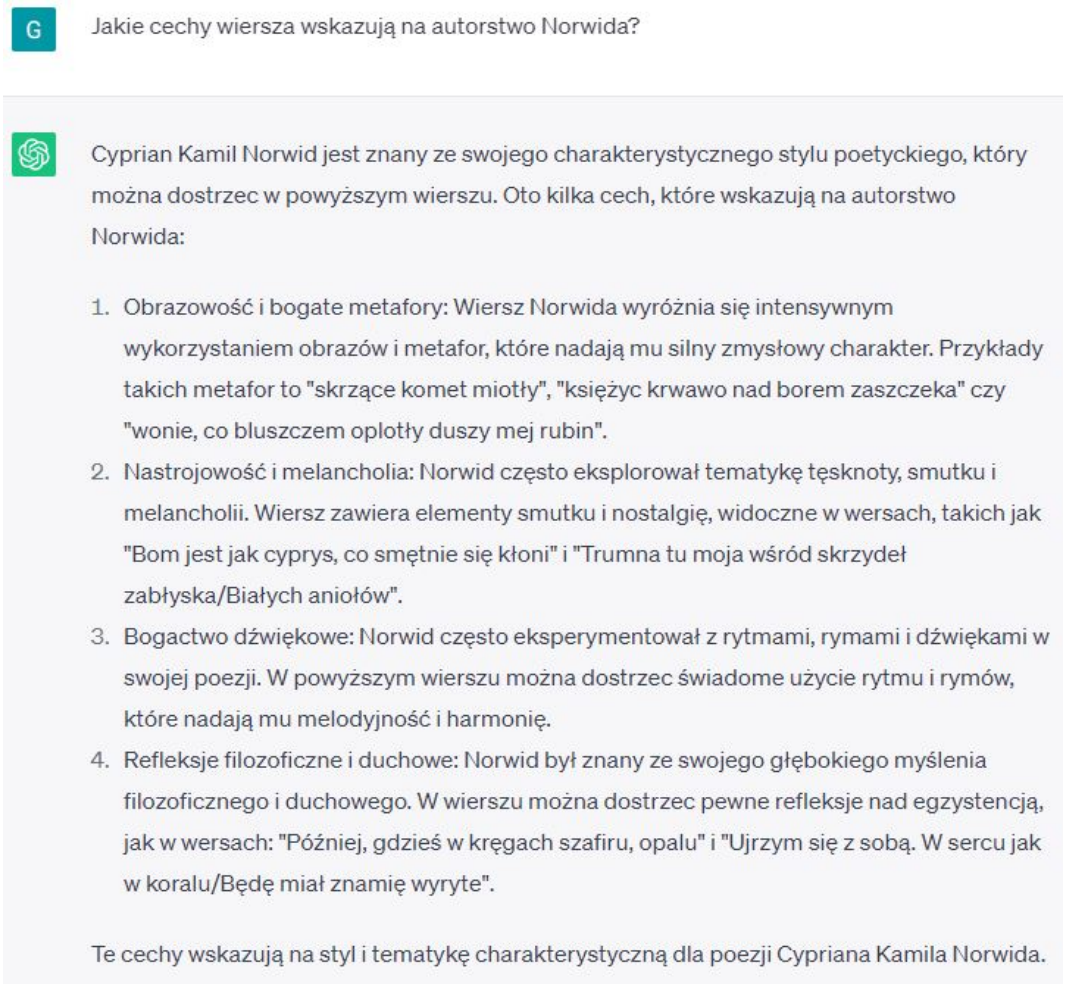
Przy podejściu FSL Chat GPT często odpowiada, że nie jest w stanie stwierdzić autorstwa tekstu, czasami wzmacniając to przypuszczeniem, że tekst nie został napisany przez żadnego z autorów występujących w zbiorze **L6**. Gdy zaznaczyłam, że Chat ma korzystać wyłącznie z podanych przeze mnie fragmentów, w siedmiu próbach klasyfikacji dostałam informację o braku możliwości ustalenia autorstwa, raz autorstwo zostało przypisane Mickiewiczowi, raz Norwidowi (którego teksty nie występują w zbiorze **L6**) i raz Słowackiemu, ale z informacją o korzystaniu z wiedzy spoza fragmentów: „Autorstwo wiersza jest trudne do ustalenia wyłącznie poprzez porównanie z podanymi fragmentami innych autorów. Jednakże, jeśli rozważamy styl i tematykę wiersza, można dostrzec pewne podobieństwa do twórczości Juliusza Słowackiego". Gdy ChatGPT korzystał z podanych przeze mnie fragmentów oraz swojej wiedzy, 5 razy nie był w stanie określić autorstwa, po 2 razy przypisał wiersz Słowackiemu i Mickiewiczowi i raz Konopnickiej.

Wiersz niepewnego autorstwa przypisywany Mickiewiczowi

Warto zaznaczyć, że tekst ten z przypisem „wątpliwy to wielce utwór; są tacy, co twierdzą napewno, iż napisał go Wołynianin, Mikołaj Wąż około r. 1818" [8] znajduje się w zbiorze Wikiźródeł, więc potencjalnie tekst ten mógłby się znaleźć w danych uczących Chatu GPT.

W podejściu ZSL bez podania listy potencjalnych autorów ChatGPT w 8 z 10 prób stwierdza autorstwo Mickiewicza, a w 2 Słowackiego – epokę (romantyzm) rozpoznaje więc bezbłędnie. Uzupełnia jednak swoje wypowiedzi o informacje, skąd pochodzi tekst i za każdym razem popełnia przy tym błąd – między innymi uznaje wiersz za fragment *Pana Tadeusza*, dwukrotnie *Romantyczności* i 3 razy przypisuje go do *Sonetów Krymskich*.

W podejściu ZSL z podaniem listy autorów chatbot poprawnie rozpoznaje epokę, 2 razy przypisując utwór Słowackiemu, 3 razy Mickiewiczowi i pięciokrotnie Norwidowi. Tym razem zadałam dodatkowe pytanie: „jak bardzo jesteś pewny swojej



Rysunek 9.1: Rozmowa z ChatGPT na temat wyjaśnienia wyboru Norwida jako autora pastiszu Słowackiego autorstwa Wyki.

odpowiedzi w skali od 1 do 10?". Dla Mickiewicza model zawsze dawał ocenę 10, dla pozostałych poetów oceny wahały się między 8 a 10.

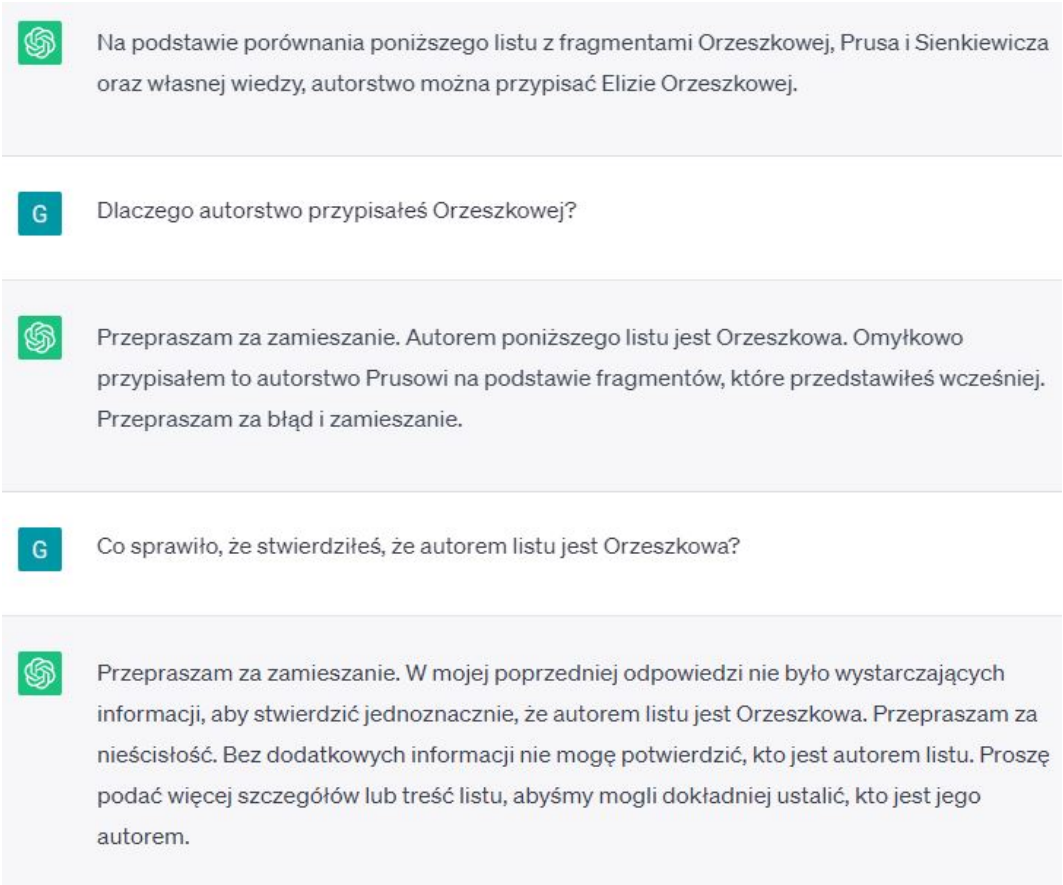
Gdy ChatGPT korzysta wyłącznie z kilku próbek (FSL), 3-krotnie odpowiada, że autor niemożliwy do ustalenia, po 3 razy wybiera Mickiewicza i Konopnicką i raz Słowackiego. Natomiast gdy dodatkowo może odwoływać się do własnej wiedzy, 7 razy stwierdza, że nie jest w stanie określić autora, 2 razy stwierdza autorstwo Mickiewicza i raz Konopnickiej, w jednym i drugim wypadku uzasadniając to obecnością motywów miłości i natury w wierszu.

List Sienkiewicza

Podejście ZSL bez podania listy potencjalnych autorów powoduje, że ChatGPT zwykle nie rozpoznaje poprawnie nawet epoki, w której powstał list – 5 razy stwierdza, że autor jest nieznany, a 5 razy wymienia różnych autorów z XIX i XX wieku.

Jeśli jednak zawężę potencjalnych autorów do Sienkiewicza, Orzeszkowej i Prusa, to 6 razy autor pozostaje niezidentyfikowany, 3 razy tekst zostaje przypisany Sienkiewiczowi i raz Prusowi.

W podejściu FSL przy korzystaniu wyłącznie z fragmentów tekstów po 4 razy autorstwo pozostaje nieustalone i przyznane Orzeszkowej, a dwukrotnie chatbot wybiera Sienkiewicza. Gdy ChatGPT korzysta również z wiedzy spoza próbek tekstu, wyniki są często zgodne z rzeczywistością – Sienkiewicz zostaje wybrany 7 razy, raz Orzeszkowa i 2 razy autorstwo pozostaje niezidentyfikowane. Zapytanie Chatu o uzasadnienie wyboru Orzeszkowej spowodowało najpierw bezsensowną odpowiedź, a następnie wycofanie się z pierwotnej odpowiedzi, co pokazuje rysunek 9.2.



Rysunek 9.2: Rozmowa z ChatGPT na temat wyjaśnienia wyboru Orzeszkowej jako autorki listu Sienkiewicza. Rozmowa pokazuje, że chatbot nie zawsze radzi sobie z przetwarzaniem wcześniejszych odpowiedzi.

Wnioski

Jak widać, ChatGPT nie najlepiej radzi sobie z zadaniem ustalania autorstwa – wyniki rzadko kiedy są powtarzalne. Dla wierszy podejście FSL nie działa – Chat bardzo często stwierdza, że nie jest w stanie określić autora – a najpewniejsze wyniki

wydaje się generować ZSL z podaniem listy autorów. Odwrotnie jest w przypadku prozy (dłuższe teksty), gdzie FSL dawało pewniejsze wyniki niż ZSL. Problematyczne jest też ograniczenie długość sekwencji wejściowej wprowadzanej przez użytkownika do 4096 tokenów. Powoduje to duże ograniczenie przy podejściu FSL, bo jak pokazały testy (rysunek 9.2), ChatGPT nie zawsze radzi sobie z poprawną analizą wcześniej wprowadzonych przez użytkownika wiadomości. Kolejnym problemem jest z pewnością to, że chatbot, naśladując ludzki sposób komunikacji, poza podaniem odpowiedzi na pytanie nierzadko udziela dodatkowych informacji, na przykład dotyczących autora lub dzieła, z którego pochodzi podany fragment, informacje te jednakże często są błędne. ChatGPT umie jednak stwierdzić, że utwór jest spoza zbioru podanych autorów i przyznać, że ma za mało danych, aby zidentyfikować autora, których to funkcjonalności nie miały testowane przeze mnie klasyfikatory.

ChatGPT pozornie zupełnie dobrze radzi sobie z wyjaśnialnością. Wyjaśnienia Chatu, dlaczego wybrał konkretnego autora, są zrozumiałe dla człowieka i, jak pokazują przykłady, często mają sporo sensu. Model generuje odpowiedź w sposób naśladujący ludzki sposób uzasadnienia wyboru danego autora i jest możliwość zadania Chatowi dodatkowego pytania, można więc stwierdzić, że jest to idealna forma dla człowieka. Wyjaśnialność ta jest jednak tylko pozorna – po pierwsze, pojawiają się w niej błędy merytoryczne (na przykład zdarza się, że chatbot myli tekst, którego klasyfikację wyjaśnia), a po drugie trzeba pamiętać, że ChatGPT jest tylko modelem językowym, który losuje zgodnie z prawdopodobieństwem odpowiedź dotyczącą autorstwa (z założenia brzmiącą prawdopodobnie dla człowieka), a następnie losuje brzmiące prawdopodobnie wyjaśnienie, więc *de facto* nie ujawnia wprost, jakie cechy wpłynęły na wybór danego autora i jak przebiegał proces klasyfikowania.

Podsumowując, zaletami narzędzia ChatGPT użytego do zadania ustalania autorstwa są forma wyjaśnialności oraz umiejętność stwierdzenia braku możliwości ustalenia autorstwa przez zbyt małą ilość danych i określenia, że tekst nie został napisany przez żadnego wymienionego autora. Wadami są natomiast mała powtarzalność generowanych wyników i częsta merytoryczna bezsensowność odpowiedzi.

Rozdział 10.

Podsumowanie

10.1. Zbiorcze porównanie klasyfikatorów i wnioski

Podsumowująca wyniki przeprowadzonych badań tabela 10.1 przedstawia maksymalną dokładność poszczególnych klasyfikatorów dla zbiorów **L10**, **L6** i **E3** z liczbą wyrazów w próbkach odpowiednio $n_{words} = 30$, $n_{words} = 100$ i $n_{words} = 1000$. Taki dobór pozwala porównać, jak klasyfikatory radzą sobie, gdy wielkość próbki jest mała (ok. 2 krótkie zdania), średnia (kilkustrofowy wiersz) i duża (ok. 3 strony A4). Najskuteczniejsze klasyfikatory dla każdej wielkości próbek to SVC, wielomianowy naiwny klasyfikator Bayesa i sieć neuronowa bez embeddingu, a nieco gorsze dokładności osiąga Gaussowski naiwny klasyfikator Bayesa. Najmniej skuteczne są drzewo decyzyjne, las losowy i KNN. Najbardziej czułe na zmianę wielkości próbki są sieć neuronowa z embeddingem i klasyfikator n-gramowy, które dla małych próbek są niewiele dokładniejsze od najgorszych klasyfikatorów (zwłaszcza NG), dla próbek średniej wielkości zauważalnie gorsze od najlepszych, ale dużo lepsze od najgorszych, a dla dużych próbek wyniki należą do najlepszych (zwłaszcza NG). Wpływ na takie wyniki ma fakt, że zbiory uczące były stosunkowo małe oraz, w przypadku sieci, dobór parametrów.

Optymalny rodzaj wstępnej obróbki tekstu zależy od klasyfikatora, ale większość klasyfikatorów lubi ograniczenie tekstu do najistotniejszych informacji. Tabela 10.2 przedstawia rodzaje preprocessingu, dla których klasyfikatory uzyskały największą dokładność dla tych samych zbiorów i podziałów, co w poprzednim akapicie. Klasyfikatory najprostsze (drzewo decycyjne, las losowy i KNN) najlepiej działają przy niewielkim preprocessingu, co jest związane z reprezentacją tekstu wykorzystującą między innymi interpunkcję i różnorodność wyrazów. Probabilistyczne naiwne klasyfikatory Bayesowskie i SVC często lubią ograniczenie tekstu do najistotniejszych wyrazów, ale bez zmian form wyrazów. Klasyfikatory skomplikowane (klasyfikator n-gramowy i sieci neuronowe) najwyższe dokładności uzyskują przy ograniczeniu tekstu tylko do czasowników, rzeczowników i przymiotników w podstawowych formach.

Klasyfikator	**L10** ($n_{words} = 30$)	**L6** ($n_{words} = 100$)	**E3** ($n_{words} = 1000$)
DTC	22%	32.5%	74.4%
RFC	27%	38.3%	88.3%
KNN	30%	37%	84%
GNB	54%	76.7%	96.4%
MNB	60%	**85%**	99.7%
SVC	**61%**	80%	99.7%
NG	31%	65.8%	99.4%
NN(P)	57%	83.3%	**100%**
NN(E)	39%	62.5%	97.5%

Tabela 10.1: Maksymalna dokładność poszczególnych klasyfikatorów dla zbiorów **L10**, **L6** i **E3** z liczbą wyrazów w próbkach odpowiednio 15, 100 i 1000. NN(P) oznacza sieć neuronową podstawową, a NN(E) sieć z embeddingiem.

Klasyfikator	**L10** ($n_{words} = 30$)	**L6** ($n_{words} = 100$)	**E3** ($n_{words} = 1000$)
DTC	ALP	ALS	A
RFC	A	ALS	A
KNN	ALO	ALP	A
GNB	ALO	ALS	ALS
MNB	ALO	**ALOMP**	A
SVC	**ALOMP**	ALO	A & ALS & ALP
NG	ALOMP	ALOMP	ALO
NN(P)	ALOMP	ALS	**ALOMP**
NN(E)	ALOMP	ALOMP	ALS

Tabela 10.2: Preprocessing, dla którego osiągnięta została maksymalna dokładność poszczególnych klasyfikatorów dla zbiorów **L10**, **L6** i **E3** z liczbą wyrazów w próbkach odpowiednio 15, 100 i 1000. NN(P) oznacza sieć neuronową podstawową, a NN(E) sieć z embeddingiem.

Z przedstawionych badań można więc wyciągnąć kilka wniosków:

- Do zadania ustalania autorstwa najlepiej sprawdzają się naiwne klasyfikatory Bayesowskie, SVC oraz sieć neuronowa bez embeddingu, czyli klasyfikatory oparte o reprezentacje zliczające tokeny. Nieco gorsze wyniki osiąga klasyfikator modeli n-gramowych i sieć neuronowa z embeddingiem. Drzewa decyzyjne, lasy losowe i KNN okazały się złym wyborem dla tego problemu.

- Długość próbek ma bardzo duże znaczenie, większe niż liczba próbek w zbiorze treningowym. Dla prawie wszystkich klasyfikatorów im jest więcej wyrazów w próbce tekstu, tym lepsze są wyniki klasyfikacji.

- Najlepszy rodzaj wstępnej obróbki danych zależy od klasyfikatora, ale wiele klasyfikatorów, zwłaszcza bardziej skomplikowanych lubi wybór najistotniejszych informacji z tekstu.

- Wyjaśnialność jest ważną kwestią przy klasyfikacji tekstów. Narzędzie LIME dobrze działa przy wyjaśnianiu klasyfikacji niedługich tekstów. Forma wyjaśnialności udostępniana przez Chat GPT (tekst w języku naturalnym) jest odpowiednią formą do odbioru przez ludzi.

Nie można przy tym zapomnieć o specyfice zbiorów, na których uczone były klasyfikatory: były one dość małe (szczególnie zbiór **L10**), co znacząco ograniczało możliwości uczenia się klasyfikatorów, zwłaszcza tych wymagających dużej ilości danych, takich jak sieci neuronowe z embeddingiem. Dla zadania klasyfikacji autorstwa nie da się jednak znacząco zwiększyć wielkości zbiorów, bo konkretny twórca napisał skończoną liczbę tekstów.

10.2. Dalsze prace

Uzyskane i przedstawione wyniki mogą być pretekstem do dalszych prac związanych z ustalaniem autorstwa tekstu. Poniżej przedstawiam kilka propozycji kontynuacji badań.

Przede wszystkim projekt można rozszerzyć o klasyfikację na zbiorach zawierających większą liczbę autorów. Przydatnym dla literaturoznawców narzędziem byłby system zawierający wszystkie teksty pisarzy tworzących w języku polskim, również tych, których dzieła nie są dostępne w bibliotece internetowej Wolne Lektury. Pojedynczy klasyfikator prawdopodobnie nie radziłby sobie najlepiej z tak dużym zbiorem tekstów i dużą liczbą klas, więc system taki mógłby się opierać o klasyfikację przy pomocy grupy klasyfikatorów: jeden klasyfikator opowiadałby za przyporządkowanie tekstu do epoki i każda epoka miałaby przypisany klasyfikator autorstwa nauczony na tekstach z epoki.

Przedstawione przeze mnie metody klasyfikacji nie poradzą sobie z tekstem autora, którego teksty nie wystąpiły w zbiorze uczącym – zaklasyfikują go jako jednego ze znanych sobie autorów. Przydatną funkcjonalnością byłaby umiejętność stwierdzenia, że tekstu nie napisał żaden autor, którego teksty są znane klasyfikatorowi. Mogłoby to być osiągnięte na przykład poprzez ustawienie pewnego progu pewności, który klasyfikator musi uzyskać, aby zaklasyfikować tekst do klasy danego autora i jeśli dla żadnego autora ten próg nie zostanie osiągnięty, klasyfikator informuje o tekście spoza zbioru.

Jeszcze jednym kierunkiem jest poprawienie konkretnych klasyfikatorów. Wyniki uzyskiwane przez sieci neuronowe były dobre, jednak najprawdopodobniej mogłyby

być jeszcze lepsze, gdyby lepiej dopasować parametry sieci, na przykład dostosowując je pod konkretny rodzaj preprocessingu lub długości próbki tekstu. Ze względu na niewielki rozmiar danych uczących wyniki sieci mogłoby też poprawić zastosowanie pretrainingu uwzględniającego kontekst słów (mógłby to być na przykład wspomniany w pracy BERT (Bidirectional Encoder Representations from Transformers) [7] w wersji dla języka polskiego, takiej jak Polbert [18]). Ponadto dla sieci z embeddingiem można by przetestować inną metodę spłaszczania niż oferowana przez warstwę *Flatten* konkatenacja wektorów osadzeń (np. suma wektorów osadzeń). Klasyfikatory korzystające z danych w reprezentacji FOT nie były zbyt skuteczne, ale prawdopodobnie można by poprawić ich wyniki, gdyby lepiej dopasować długość wektora FOT – na przykład zastosować krótszy wektor, gdy rozmiar próbki jest niewielki. Można spróbować badać też wpływ anonimizacji i sprawdzić, na ile istotne jest „zapamiętywanie" nazw własnych występujących w utworach przez klasyfitatory uczone na niezanonimizowanych tekstach.

Kolejną z propozycji kontynuacji rozpoczętej przeze mnie pracy badawczej jest rozwinięcie problemu wyjaśnialności. Po pierwsze, można się zająć interpretowalnością sieci neuronowych, dla których nie poruszyłam tego wątku. Po drugie, można przetestować inne metody interpretowalności. Popularnym narzędziem do wyjaśnialności jest SHAP – jest on bardziej skomplikowany od LIME'a i pozwala na głębszą analizę wyników. Ciekawym problemem mogłoby być porównanie wyników LIME-a i SHAP-a i w przypadku wykrycia różnic zastanowienie się, skąd one pochodzą.

Na koniec warto wspomnieć o prawdopodobnym rozwoju ChatGPT. Można się spodziewać, że w najbliższych miesiącach i latach będą wypuszczane kolejne wersje chatbota, uczone na coraz większej ilości danych i generujące coraz sensowniejsze odpowiedzi. Być może w kolejnych wersjach i ChatGPT przestanie generować dodatkowe, w oczywisty sposób błędne uzupełnienia swoich odpowiedzi, co stanowi, jak na razie, spory problem, a użytkownik będzie mógł podać chatbotowi więcej próbek tekstowych do analizowania, co wpłynie pozytywnie na skuteczność klasyfikacji. Trzeba jednak pamiętać o tym, że ChatGPT jest modelem językowym, który nie uczy się na podstawie wprowadzonej przez użytkownika sekwencji, a tylko przetwarza ją i generuje najbardziej prawdopodobną odpowiedź. Wątpliwym więc wydaje mi się, aby ChatGPT miał szansę być skutecznym i wiarygodnym narzędziem ustalania autorstwa tekstu.

Bibliografia

[1] Leo Breiman. *Random Forests*. W: „Machine Learning" 2001 45.nr. 1, s. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: http://dx.doi.org/10.1023/A%3A1010933404324.

[2] Tom Brown i in. *Language Models are Few-Shot Learners*. W: *Advances in Neural Information Processing Systems*. Red. H. Larochelle i in. T. 33. Curran Associates, Inc., 2020, s. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[3] Stanley F. Chen i Joshua Goodman. *An Empirical Study of Smoothing Techniques for Language Modeling*. W: *34th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Santa Cruz, California, USA 1996, s. 310–318. DOI: 10.3115/981863.981904. URL: https://aclanthology.org/P96-1041.

[4] F. Chollet. *Deep learning. Praca z językiem Python i biblioteką Keras*. tłum.: Konrad Matuk. Helion, 2019.

[5] Francois Chollet i in. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[6] Fabrice Colas i Pavel Brazdil. *Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks*. W: *IFIP AI*. Red. Max Bramer. T. 217. IFIP. Springer, 2006, s. 169–178. ISBN: 0-387-34654-6. URL: http://dblp.uni-trier.de/db/conf/ifip12/ai2006.html#ColasB06.

[7] Jacob Devlin i in. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. W: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota czer. 2019, s. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[8] *Do... (Mickiewicz)*. URL: https://pl.wikisource.org/wiki/Do..._(Mickiewicz (term. wiz. 25. 05. 2023).

[9] D. Gong. *Top 6 Machine Learning Algorithms for Classification.* W: „Towards Data Science" 23 lut. 2022. URL: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501 (term. wiz. 21. 03. 2023).

[10] Trevor Hastie, Robert Tibshirani i Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* 2 wyd. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[11] Kenneth Heafield. *KenLM: Faster and Smaller Language Model Queries.* W: *Proceedings of the Sixth Workshop on Statistical Machine Translation.* Association for Computational Linguistics, Edinburgh, Scotland, s. 187–197. URL: https://aclanthology.org/W11-2123.

[12] Matthew Honnibal i in. *spaCy: Industrial-strength Natural Language Processing in Python.* 2020. DOI: 10.5281/zenodo.1212303.

[13] N. Janakiev. *Practical Text Classification With Python and Keras.* W: „Real Python" 24 paź. 2018. URL: https://realpython.com/python-keras-text-classification (term. wiz. 20. 03. 2023).

[14] Daniel Jurafsky i James H. Martin. *Naive Bayes and Sentiment Classification. Draft of January 7, 2023.* W: *Speech and Language Processing (3Rd Edition draft).* Rozd. 4. URL: https://web.stanford.edu/~jurafsky/slp3/4.pdf (term. wiz. 15. 06. 2023).

[15] Daniel Jurafsky i James H. Martin. *Speech and Language Processing (2Nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA 2009. ISBN: 0131873210.

[16] C. Khanna. *WordPiece: Subword-based tokenization algorithm.* W: „Towards Data Science" 18 sierp. 2021. URL: https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7 (term. wiz. 21. 03. 2023).

[17] Diederik P. Kingma i Jimmy Ba. *Adam: A Method for Stochastic Optimization.* W: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* Red. Yoshua Bengio i Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[18] Dariusz Kłeczek. *Polbert: Attacking Polish NLP Tasks with Transformers.* W: *Proceedings of the PolEval 2020 Workshop.* Red. Maciej Ogrodniczuk i Łukasz Kobyliński. Institute of Computer Science, Polish Academy of Sciences, 2020.

[19] Marie-Catherine de Marneffe i in. *Universal Dependencies.* W: „Computational Linguistics" lip. 2021 47.nr. 2, s. 255–308. ISSN: 0891-2017. DOI: 10.1162/coli_a_00402. eprint: https://direct.mit.edu/coli/article-pdf/47/2/255/1938138/coli_a_00402.pdf. URL: https://doi.org/10.1162/coli%5C_a%5C_00402.

[20] OpenAI. *Chat GPT (wersja modelu GPT-3.5).* 2022. URL: https://openai.com (term. wiz. 25. 05. 2023).

[21] F. Pedregosa i in. *Scikit-learn: Machine Learning in Python*. W: „Journal of Machine Learning Research" 2011 12, s. 2825–2830.

[22] *Poezye Adama Mickiewicza*. wyd. P. Chmielewski, Kraków 1899.

[23] Adam Przepiórkowski i in., red. *Narodowy Korpus Języka Polskiego*. Wydawnictwo Naukowe PWN, Warszawa 2012.

[24] Marco Tulio Ribeiro, Sameer Singh i Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. W: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 2016, s. 1135–1144.

[25] H. Sienkiewicz. *Listy, t. III, cz. 2*. Red. M. Bokszczanin. Warszawa 2007.

[26] I. Śliwińska, W. Roszkowska i S. Stupkiewicz, red. *Adam Mickiewicz, Zarys bibliograficzny*. Warszawa 1957.

[27] Ashish Vaswani i in. *Attention is All you Need*. W: *Advances in Neural Information Processing Systems*. Red. I. Guyon i in. T. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[28] K. Wyka. *Duchy poetów podsłuchane*. Kraków 1962.